Pacific Northwest
NATIONAL LABORATORY

PNNL-32687

Smart Contract Architectures and Templates for Blockchain-based Energy Markets (V1.0)

March 2022

D. Jonathan Sebastian-Cardenas
Sri Nikhil Gupta Gourisetti
Peng Wang
Jesse Smith
Mark Borkum
Monish Mukherjee

U.S. DEPARTMENT OF ENERGY

Prepared for the U.S. Department of Energy
under Contract DE-AC05-76RL01830

PACIFIC NORTHWEST NATIONAL LABORATORY
operated by
BATTELLE
for the
UNITED STATES DEPARTMENT OF ENERGY
under Contract DE-AC05-76RL01830

Printed in the United States of America

Available to DOE and DOE contractors from the
Office of Scientific and Technical Information,
P.O. Box 62, Oak Ridge, TN 37831-0062;
ph: (865) 576-8401
fax: (865) 576-5728
email: reports@adonis.osti.gov

Available to the public from the National Technical Information Service
5301 Shawnee Rd., Alexandria, VA 22312
ph: (800) 553-NTIS (6847)
email: orders@ntis.gov <https://www.ntis.gov/about>
Online ordering: http://www.ntis.gov

PNNL-32687

Smart Contract Architectures and Templates for Blockchain-based Energy Markets (V1.0)

March 2022

D. Jonathan Sebastian-Cardenas
Sri Nikhil Gupta Gourisetti
Peng Wang
Jesse Smith
Mark Borkum
Monish Mukherjee

Prepared for
the U.S. Department of Energy
under Contract DE-AC05-76RL01830

Pacific Northwest National Laboratory
Richland, Washington 99354

Abstract

Within the field of Transactive Energy Systems (TES), there is an active need for tools that can support and accelerate the development of these new grid solutions. Among the many tools available, blockchain stands out as a viable instrument that can help researchers develop decentralized, autonomous, and tamper-resistant grid applications. In this work, we explore the use of smart contracts (SCs), a subset of blockchain technology, and analyze their applicability to facilitating the implementation of TES solutions. In particular, we focus on presenting areas of opportunity and potential drawbacks, along with use cases that can benefit from this technology building upon previous research developed by Pacific Northwest National Laboratory and other research organizations.

This work builds upon the fundamentals of TES and smart contract technology to develop a series of software templates that can be used by industry to build TES-oriented grid solutions. These templates are intended to be platform agnostic and take into consideration the unique properties of SCs and distributed ledger storage mechanisms to ensure actual code implementations remain aware of the limitations of the technology. The proposed templates have the potential to enable software architects to mix and match components to satisfy their application requirements, thereby reducing the number of resources required to implement blockchain-based solutions.

These templates are divided into two main components—data and behavioral models. The data models are intended to help software engineers represent the underlying grid objects along with their properties in a ledger-based storage system. The behavioral models are used to describe the processes and actions that actors within a system must perform to achieve a given outcome such as registering an asset, placing a bid, and performing bid clearances. These two components are documented in a Unified Modeling Language (UML) format and are intended for use in SC-based implementations, with special behavioral considerations to account for the asynchronous properties of the underlying ledger and the typical execution model of smart contracts.

Finally, future research ideas and potential extensions to this work are discussed. In particular, known limitations and potential improvements of the developed product are identified and expected to be addressed in future revisions of the template model.

Acronyms and Abbreviations

ABAC	Attribute-Based Access Control
ACL	Access Control List
B-A TES	Blockchain Architecture for Transactive Energy Systems
CA	California
DER	Distributed Energy Resource
DSO	Distribution System Operator
DLT	Distributed Ledger Technology
DOE	U.S. Department of Energy
ESI	Energy Service Interface
ISP	Identity Service Provider
LMP	Local Marginal Pricing
LPC	Locational Pricing Calculator
MSP	Membership Service Provider
NIST	National Institute of Standards and Technology
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PIP	Policy Information Point
PKI	Public Key Infrastructure
PNNL	Pacific Northwest National Laboratory
PRP	Policy Retrieval Point
RBAC	Role Based Access Control
RT	Real Time
SC	Smart Contract
TE	Transactive Energy
TEAC	Transactive Energy Abstract Component
TES	Transactive Energy System
TESC	Transactive Energy Systems Conference
TNT	Transactive Network Template (V 2.0 or below)
TENT	Transactive Energy Network template (New versions of TNT)
TPS	Transaction Per Second
UML	Unified Modeling Language

Contents

Figures

Tables

1.0 Introduction

Blockchain technology has continued to receive attention over the last decade due to its unique ability to store data in an immutable datastore (known as a ledger), while simultaneously enabling a diverse set of agents to reach a consensus about its contents. Although the term blockchain is often used in the literature, blockchain represents only a subset of the possible implementations within the field of Distributed Ledger Technologies (DLTs). DLTs represent a much more general term that encompasses several storage and consensus mechanisms used to communicate, compute, agree on the outcome, and eventually store data. Nevertheless, in this report both terms are used interchangeably because the presented work is implementation and feature agnostic.

Although blockchain use is often related to cryptocurrencies, its underlying features can be leveraged to address a wide variety of use-case applications that may benefit from the following features:

1. A distributed, open, and verifiable transaction platform that relies on distributed-system architecture to agree on a common state, with a broad emphasis on sustaining ad hoc applications. Although the agreement is often bound to the data layer, agreement can also be ensured at the logical level.
2. Strong immutability properties that can enable participants to efficiently detect data modification attempts. This immutability can be used to support strong data-provenance guarantees for end-user applications.
3. A multi-entity, multi-factor, identity management service, which can be used to provide strong non-repudiation properties to the system, thereby fostering a distributed, yet highly trusted computing platform.

Based on the aforementioned features, an individual’s application needs maybe partially satisfied by blockchain technology. Viable candidate applications are usually related to those that require or have a need for (1) decentralized operations or where participants present ad hoc behaviors; (2) require strong immutability/non-repudiation properties that trump other performance demands (e.g., speed, throughput); (3) cannot be efficiently handled by existent or traditional computing solutions. Within the electrical industry, and based on ongoing/previously reported research efforts, use cases can be broadly grouped into the following categories:

- **Consumer-facing applications:** Applications that can increase end-user trust by increasing transparency along with other governance attributes. Examples include determining the economic value of energy credits, keeping track of carbon-free credits and clean energy tokens, and documenting decision-making processes, actions, or plans, examples include (Patel, et al. 2020).
- **Market places and trading:** Applications that require multiple parties to participate in open trading operations in a scalable manner. Well-designed systems can ensure equitable and inclusive agent participation regardless of their size or competitive advantages, thereby enabling participants to accurately determine the true value of energy, examples include (Eisele, Barreto, et al., Blockchains for Transactive Energy Systems: Opportunities, Challenges, and Approaches 2020), (Hahn, et al. 2017). Specifically, blockchain technology has the potential to eliminate intermediaries while protecting data-in-transit and data-at-rest.
- **Supply chain:** Applications that need to track an asset’s lifecycle with high degree of certainty. Although usually tied to physical assets, the concept can be expanded to provide

data provenance services, identity management (e.g., managing prosumer credentials), and support process lifecycles (e.g., track customer interconnection requests), among many other applications, examples include (Mylrea and Gourisetti, Blockchain for Supply Chain Cybersecurity, Optimization and Compliance 2018), (Liang, et al. 2018).

- **Enabling multi-organizational integrations:** Applications that require multi-organizational vertical or horizontal participation that may benefit from neutral platforms that enable organizations with competing interests to reach consensus (Tonghe, et al. 2021). Typical use cases may include integrating organizations with operational or ownership boundaries, such as utilities, regulatory entities, and regional/system operators within the same platform.
- **Digital enforcement of contractual obligations:** Applications that require agents to follow procedural processes that can be tracked and enforced by digital means (Hahn, et al. 2017). Examples include but are not limited to the tracking of asset exchanges (if digital representations can be achieved), neutral enforcement of legal contracts among parties with competing interests, and the automation of processes that benefit from a distributed, decentralized architectures.

As stated in the preceding category descriptions, blockchain can assist a variety of grid applications, with some authors such as (Andoni 2019) providing extensive reviews on potential grid applications. However, an application's reliance on blockchain must be dictated by actual needs rather than want-to-use obligations. To this end, this report focuses on the use of blockchain technology as an enabling technology to support the requirements of a Transactive Energy System (TES). According to (GridWise Architecture Council 2015), a TES is *a system of economic and control mechanisms that allows the dynamic balance of supply and demand across the entire electricity infrastructure using value as a key operatio*nal parameter. In a more general sense, a TES is a mixture of components that work together to bring the below outlined benefits, these benefits have been derived from (Gourisetti, et al. 2021), (Gourisetti, et al. 2019), and (GridWise Architecture Council 2015).

- **Optimization-oriented capabilities:** Transactive systems can be configured to achieve a common, predefined goal, which can serve to bring benefits to individual groups or to an entire system depending on an organizational policy.
- **Improved reliability and maintainability:** A well-designed transactive system can increase a system's overall reliability by supporting automated recovery solutions. The solutions could leverage individual agents, system-level automations, and communication links to achieve their desired functionality. Furthermore, due to its multi-domain capabilities, a TES may enable implementation of solutions that rely on vertical or horizontal integrations to achieve its end goal.
- **Allows fair and equitable operations:** By providing standardized interfaces that follow procedural behaviors, agents are ensured fair participation. Moreover, these procedures can be tailored to enable equitable participation if desired.
- **Increased observability:** Because a TES is expected to follow strict procedural behaviors, a record of the decision-making process should be available for posterior analysis. This transparency promise can further encourage participants' engagement regardless of their size, limited capabilities, and/or prior experience (as opposed to more established, dedicated service providers).
- **Scalability, extensibility, and adaptability:** A TES enables a wide number of agents dispersed across the entire grid system to participate toward fulfillment of a common goal.

Furthermore, a well-designed TES can accommodate future expansions and adapt to changing conditions, thereby ensuring a future-proof system.

- **Participant's accountability:** Due to its traceability properties, a TES holds all participants accountable for their actions, which may include keeping track of an individual's participation history, compliance behavior, and predictability. This accountability property helps ensure a fair system that can be corrected if issues arise.

As can be observed, a TES can be applicable to a wide variety of scenarios. However, in this work, the focus is on those services that can satisfy common industry needs, such as those described by (Cazalet, et al. 2016). Specifically, uses cases that demonstrate or enable the interoperability of systems that are currently isolated or have limited operational connectivity may be particularly valuable (e.g., enabling behind-the-meter DERs to support DSO's needs). In addition, use cases that can enable decarbonization and integration of renewable energy into more traditional processes can be a welcome addition. Based on these ideas, generic market interfaces that can enable the exchange of services, goods, and non-tangible assets across a wide variety of systems by relying on the TES model and blockchain technology are explored in this work.

In particular, we aim to leverage the automation features of blockchain—features referred to as smart contracts (SCs) within the context of this report. SCs are a collection of tools and data mechanisms that enable participating peers to agree on a logical state using a complex state machine that runs on top of blockchain peers. These pieces of logic can be used to assemble complex algorithms that run on a distributed platform (the degree of Turing-completeness depends on the blockchain implementation). These SCs enable end-users to develop solutions that inherit many of the traits of blockchain technology without having to worry about the complexities of developing a distributed system from scratch, thereby reducing potential costs and implementation risks.

However, SCs still require application developers to be aware of the limitations and unique processing requirements of SC technology—a task that may prove daunting and limit an interested party's ability of to experiment with the technology. To ease with this task, this report presents a series of pre-vetted data and behavioral models that can speed blockchain development. These models are intended to serve as a reference guide for software architects, developers, and any interested party that seeks to build blockchain-based solutions.

1.1 Goals

Based on the previously identified gaps, and perceived industry needs, this report focuses on satisfying the following goals.

1. Design a set of SC templates that can be used to deploy TES-based applications irrespective of the underlying blockchain solution. The design operates under the assumption that Turing-complete logic algorithms can be executed by the underlying blockchain.
2. The proposed designs should aim to simplify the process of end-to-end connectivity across a variety of systems, enabling both vertical and horizontal integration of service providers irrespective of their scale or aggregation capabilities.
3. The developed templates must enable application designers to mix and match components as they see fit. The design should enable wide compatibility with TES-oriented applications, and the templates should rely on the Energy Systems Interface *as the common point of*

coupling between the service provider and the grid (Widergren, Interoperability Strategic Vision 2018).

4. The implemented model should remain language neutral and avoid middleware or protocol-specific dependencies that limit its eventual implementation.
5. The TES templates are intended to facilitate end-user adoption by facilitating the initial configuration and providing software architects with a generic platform that can be further refined to suit specific needs. However, providing pre-built libraries or reference code implementations is outside the scope of this work
6. The TES templates should include access control mechanisms that prevent unauthorized access at the SC level. Furthermore, the feasibility of ledger-based access control mechanisms should be explored.
7. A set of sample applications that demonstrate the applicability of the models should be explored, and the applications should demonstrate the ability of the system to mix and match components. These applications will remain at the UML-level

1.2 Report Overview

This report is divided into five main sections. In Section 2.0, we start by presenting some of the needs of TES applications along with prior research, while section 3.0 focuses on blockchain technology from a low-level perspective, particularly in the field of SCs. In Section 4.0, we present the proposed template architectural models, including a discussion of cybersecurity considerations along with an attribute-based access control mechanism. In Section 5.0, we present the conclusion of this work.

2.0 Exploration of Existent TE Models, and the Need for Blockchain-aware Models

Over the last two decades the electrical grid has undergone a rapid transformation, primarily fueled by an increase in sustainability and resiliency goals. This transformation has been supported by a multitude of technologies that enable a wide array of physical devices, digital systems, and human operators to efficiently communicate over an extensive set of network systems and architectures. Nevertheless, this process remains largely centralized, with device-level participation mostly restricted to distribution and transmission operators / or large, previously qualified entities that must comply with strict operational policies. However, at the same time, there has been an explosion of customer-located resources that remain subject to more traditional operational models that are reminiscent of an age when customers only played a passive energy consumption role.

Across the years, these distributed energy resources have been part of experimental research studies of varying levels of maturity that have demonstrated a wide array of potential benefits, leading to standards and rulings, like the Institute of Electrical and Electronics Engineers' (IEEE's) Standard 2030.5, *Standard for Smart Energy Profile Application Protocol*, and California Electric Rule 21 (CA rule 21) that currently being used to provide active grid support services for Distributed Energy Resources (DERs). However, these real-world deployments only represent a subset of the potential applications of TESs. In this context, it is reasonable to expect, that with the correct tools, industry engagement, and correct policy drivers these experimental results can be advanced and deployed to assist with grid operations, thereby enabling greater integration of renewable resources, while enhancing grid reliability and resiliency in a manner that benefits all participants.

However, these novel developments are sometimes hindered by regulatory limitations, or a lack of technology solutions that can support real-world use cases. In some cases, certain technologies can seem promising, but their true value cannot be correctly assessed until more development tools and wide-range testing is performed. In particular, blockchain has raised interest in the TES field because of its decentralized architecture that promises to empower agents to participate in complex grid operations. This idea has led to the exploration of a wide variety of grid-related use cases that use blockchain as an enabling technology. Examples include, enabling peer-to-peer (p2p) energy exchanges (Troncia, et al. 2019), enabling collaborative microgrid environments, tracking grid assets across their lifecycle, among many others.

Although proving that a technology can work for a very specific case has value, these sorts of experiments can be hard to translate/adapt to other problems, even within the same field, potentially requiring extensive retooling and code refactoring before this can be achieved. In addition, it can be difficult for researchers to perform comparisons due to tool dependencies (e.g., licenses) and data unknowns. Organizations such as the U.S. Department of Energy (DOE) and National Institute of Standards and Technology (NIST) have recognized this issue and have encouraged the development of interoperable tools that can allow users to create solutions in a platform-agnostic manner. Current and past projects include those that use middleware to enable the integration of multiple technologies (e.g., HELICS, VOLTTRON BLOSEM), and those that seek to create interoperable data models that enable users to exchange information across multiple domains (e.g., The North American Energy Resilience Model [NAERM], the Transactive Energy [TE] challenge, and the Transactive Energy Network Template [TENT]).

In this section, we review some of the previous research that has proposed the use of platform-agnostic tools to support TES development, with a focus on generic data models rather than the tools needed to simulate/implement them. It is important to note that this report only focuses on the data models that are relevant to a decision-making agent and does not consider the data needed to model the physics of the grid, it is assumed that this can be either simulated by dedicated tools or extracted from a real-world, operational grid.

2.1 The 2016 NIST Transactive Energy Challenge

The TE challenge was a NIST-sponsored competition that sought to create a repository of co-simulation tools, models, and documentation that would enable stakeholders to quickly understand, test, and apply TE solutions to address their grid challenges (Holmberg, et al. 2019). The challenge originated from a desire to evaluate the integration of renewable resources into TESs using simulation tools rather than demonstration projects; therefore, a significant part of the challenge included the development and identification of co-simulation tools that would accelerate the development, testing, and evaluation of potential TE solutions.

Multiple teams participated in the challenge, and certain teams were dedicated to addressing specific interoperability issues related to software or data models. In particular, the “Tiger” team, developed the *Transactive Energy Abstract Component (TEAC) model,* which proposes the use of an abstract model that can be used to explore the benefits and impacts of transactive solutions in the day-to-day operation of energy systems (Burns, Song and Holmberg 2018). The TEAC model is intended to be compatible with any TES and provides an abstract representation of typical TES participants (e.g., loads, generators, controllers, markets).

In their work, the authors developed a framework that includes data models, as well as interface descriptors that enable stakeholders to express and develop their own solutions based on a common set of objects (see Figure 1). These objects are built around the concept of five core components (i.e., parent objects) from which other resources can be derived using an object-oriented approach. A summary of these components is presented in Based on the components described in **Error! Not a valid bookmark self-reference.**, developers can create specializations that are able to capture low-level details. The TEAC contains several examples that illustrate how such specializations can be performed; in particular, the authors present a beta use case that maps the objects found in typical grid simulation models such as GridLAB-D to the TEAC framework. A subset of these examples has been reproduced in Table 2.

Table 1; these components are mostly defined at the data layer and must be complemented by an end-user–provided interface that enables information exchange. It is important to note that the TEAC model defines object properties that have been removed from this summary.

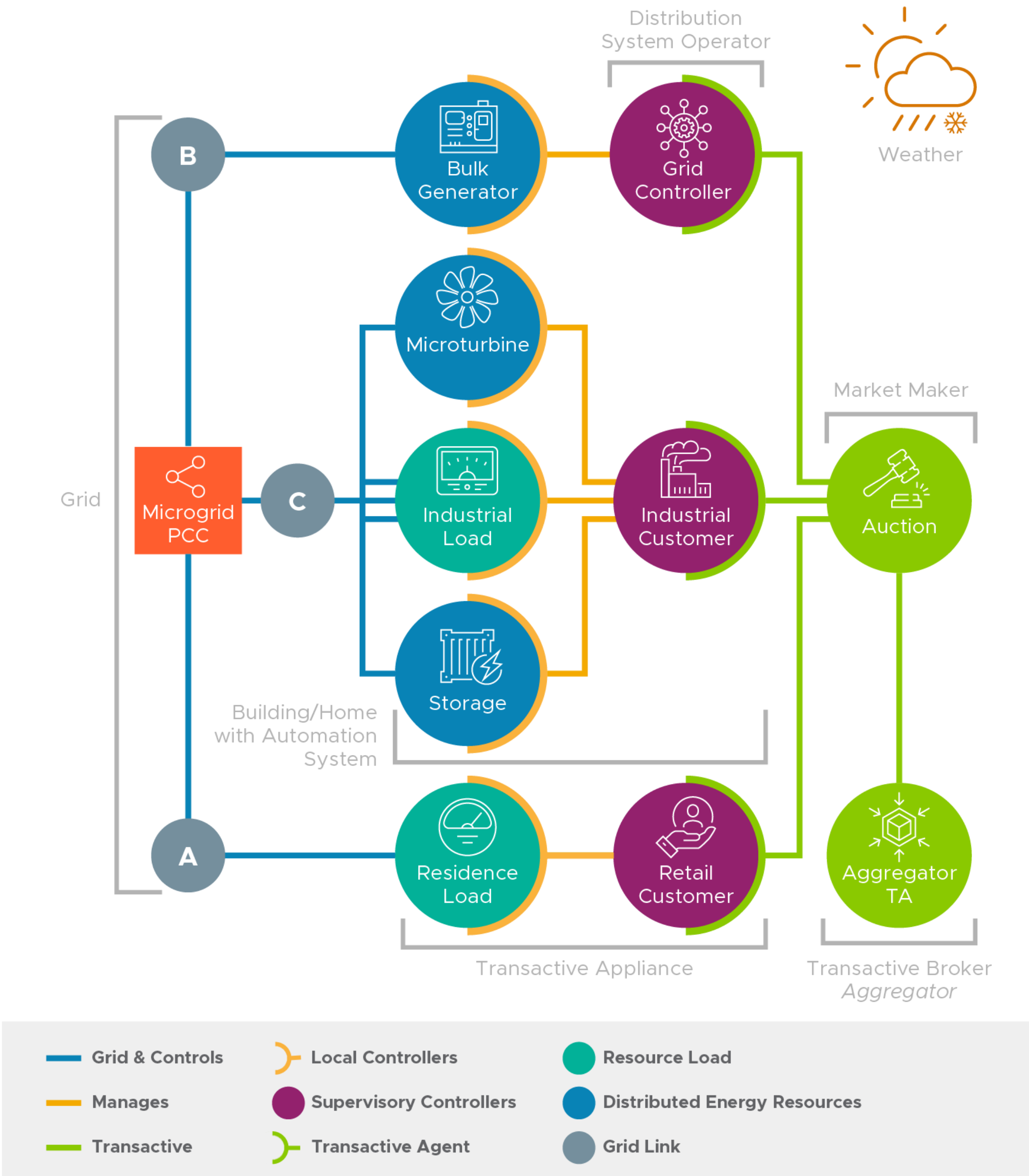


Figure 1. An overview of a generic TE model topology, using the TEAC model building blocks, adapted from (Burns, Song and Holmberg 2018).

Based on the components described in **Error! Not a valid bookmark self-reference.**, developers can create specializations that are able to capture low-level details. The TEAC contains several examples that illustrate how such specializations can be performed; in particular, the authors present a beta use case that maps the objects found in typical grid simulation models such as GridLAB-D to the TEAC framework. A subset of these examples has been reproduced in Table 2.

Table 1. Core components of the TEAC model.

Core Components	Description
Local controllers	These represent non-decision-making controllers that follow rules received from a higher hierarchy system but remain aware of the physics of the underlying system. Typical examples include thermostats (which regulate temperature based on user preference and available power), voltage regulators and in general, any energy demand/injection controller (i.e., to follow a generator's capability curve).
Supervisory controllers	These represent controllers that aggregate local controllers. These devices remain unaware of the underlaying system constraints and express required changes in units of power over a unit of time ($\Delta S/\Delta t$).
Resources	These represent any traditional grid resource that affects demand. These include loads, generators, or energy storage systems.
Weather	This component is used to represent the weather characteristics, serves as a data oracle.
Grid	The actual grid model: for the purposes of the TEAC, the model is assumed to be virtual.
Transactive agents	These represents the core functionality of a TES; they are the agents responsible for offering, bidding, negotiating, and participating in any energy exchange. They rely on all of the above objects and user-defined interfaces to achieve their functionality.

Table 2. Sample list of specialized components in the TEAC model.

Core Components	Special-ization	Description	Relevant Parameters/Attributes (not exhaustive)	
			Inherited	Specialized
Supervisory controller	Grid controller	Typically used to model the local grid operator that manages a grid net flows according to physical limitations	*Resources (list of)* *WeatherInfo (function)* *Tender (function)* *Quote (function)* *Transaction (function)*	*checkLineLimits()*
Resource	ZIP load	A load that can be controlled (i.e., available for demand response)	*Current & Voltage* *NodeID (location)* *Power* *Status*	LoadModel: *ImpedanceFraction* *CurrentFraction* *PowerFraction*
Resource	Generator	A power injection source that can be controlled (i.e., can provide grid support functions)	*Current & Voltage* *NodeID (location)* *Power* *Status*	*InternalImpedance* *IsSolar?* *hasInverter?*
Transactive Agent	Auction	This can act as a system-level market broker in centralized environments or be a service that allows decentralized energy exchanges to occur.	*WeatherInfo (function)* *Tender (function)* *Quote (function)* *Transaction (function)*	*Auction(function)*

As mentioned earlier, the TEAC model is composed of data models and interface descriptors. These interfaces are logical constructs that enforce the methods/functions that must be supported by the objects that choose to expose these interfaces. Under this paradigm, an air-conditioner and a water heater must support the same function calls, thereby enabling external systems to communicate with them regardless of an individual's principle of operation. It is important to note though that the TEAC reference model remains at a very high level and it only provides basic interfaces, such getting a device on/off status and invoking functions that represent the quote, tender, and transaction processes. It is up to the developer to define how these interfaces are actually implemented (not only from a communication and logical perspective, but also relative to the algorithms that are used to perform the process).

The TEAC model also supports the use of composite classes, that is, it enables end-users to logically join functionalities of different systems into a single device model. This for example could enable grid operators to represent photovoltaics (PV)-based smart inverters acting as a *resource, local controller, and supervisory controller* at the same time as a single, integrated object, thereby reducing the number of objects that must be maintained and communicated. Although the TEAC offers many object-oriented relational capabilities, potential users should refrain from regrouping or modifying the core models to ensure that systems developed by different entities remain comparable across the board. This means that end-users should rely on composition and inheritance to build new specializations rather than adding new core components.

In summary, the TEAC model provides an excellent set of reference models that enable potential users to leverage a well-defined skeleton that can be used to represent and compare different TES implementations. This, for example, could enable competitors to share architectural diagrams without risking the loss of proprietary information, while also ensuring that everyone understands the underlying communication/data dependencies among resources. However, the TEAC remains at a very high level, and still requires the end-user to develop and refine the implementation details.

2.2 The Transactive Energy Network Template

The transactive network template (TNT) was developed by Pacific Northwest National Laboratory (PNNL) in 2019 to advance the implementation and use of TESs (Hammerstrom 2019), and further developed into its current form, the Transactive Energy Network Template (TENT). The TENT represents a metamodel architecture, or a system of models that work together to achieve a common goal (a TES). In this context, the metamodel serves as a highly abstracted template that seeks to replace single-use, custom-engineered TES solutions with a more standardized architecture that enables future integration of these isolated solutions under a single umbrella. This forward-thinking logic should prevent the eventual isolation of existing and ongoing TES projects by providing common objects that enable inter-system interoperability, regardless of the individual problem being solved.

Since its inception, the TENT has allowed decentralized and distributed scheduling, control, and coordination of electric power systems. Transactive agents within a TENT are considered truly independent, with no de-facto centralized authority (participants may join a central-like authority if they want to). Transactive agents negotiate prices and energy-related assets by exchanging transactive signals in a fully distributed manner. A transactive agent is aware of its own assets, flexibilities, and capabilities, but is (and should remain) incapable of determining any other agent's resources and capabilities unless the other party wishes to disclose that information.

To achieve the aforementioned goals, there must be a communication system that enables all agents to gain equal access to the system and equal opportunity to offer and receive grid resources/flexibilities to and from the system. However, the TENT appropriately notes that reference implementations should remain communication-system-agnostic, i.e., a reference implementation should not specify or rely on the features provided by a single communication solution/platform to provide their services. To accomplish this, the TENT relies on high-level UML-based representations of the different participating entities to assemble its metamodel. These representations have been designed to fulfill the following features:

- **Multi-domain, multi-capability representation:** The TENT enables the representation of diverse agents. All agents are capable of providing supply or demand services as they see fit without a predetermined flow direction.
- **Ability to announce capabilities and needs**: Agents are capable of expressing their future plans, resources, and needs. However, they should not expect the same level of reciprocity from all other agents.
- **Enable negotiation:** Agents can send transactive signals to their neighbors and are able to coordinate using communication channels and other grid-specific attributes such as prices.
- **Enable the representation of multi-domain architectures:** The TENT has been designed to enable the representation of multiple TES use cases, ranging from asset exchanges between peers (p2p) to more traditional, centralized market operations.
- **Enable the collaboration of multiple actors:** The TENT maintains an open representation of an agent. An agent can be a wholesale market actor, or a generation, transmission, distribution, commercial, or residential participant. All assets across all subcircuits can be engaged in the TENT.

The TENT is composed of 11 base objects, which can be extended/customized to satisfy the stakeholders needs. A brief description of the most important components is provided below:

- **Transactive agent object:** The transactive agent represents a business entity that is in charge of a specific circuit region, circuit element, or a specific generating or consuming device. It keeps track of its local assets, its local market, and its neighboring transactive agents with which it communicates, negotiates, and exchanges signals. There may be multiple transactive agents in a TNT, and each agent can manage multiple devices/services within the same electrical node.
- **Market object:** This object balances the power supply/demand for a transactive agent (the market is its own agent). A successful balancing means that the sum of generated, imported, consumed, and exported electricity power or electricity energy must be zero in every market period. The balancing is achieved with a price-discovery mechanism. There may be multiple market instances within a TES. For example, instances of a day-ahead market and a real-time hourly market can co-exist at the same time. In addition, different types of markets (i.e., energy and ancillary services) can co-exist under the same system. Market time intervals must be communicated to agents and must be aligned with local scheduling and coordination processes, it is assumed that markets are operated using forward time intervals (e.g., the price is known before consumption).
- **Local asset model object:** A local asset model contains all elements that are known and managed by the transactive agent. Local assets are fully transparent to their transactive agents, which means that transactive agents know everything about the local assets (e.g., demand, or storage needs). The model is responsible for forecasting and scheduling all of

the local assets' power generation or consumption demands using forward time intervals. Once dispatches/prices are known, the local asset model must ensure that the physical assets follow the pre-scheduled actions. The local asset model object is capable of integrating low-level controls and the necessary communication protocols.

- **Transactive neighborhood model object:** This model represents the neighbor agents from the perspective of a single agent (e.g., an agent's neighbor). The model should be instantiated for every neighbor agent. Two agents are neighbors if they make transactions with each other. A transactive neighbor model manages information sent to and received from the neighboring transactive agent. The information may include price, schedule, flexibility, and other related information. This information should flow using flexible, but standardized, interfaces to ensure interoperability across all participants; flexible and event-triggered communications are preferred. The transactive agent is responsible for scheduling the power that imports from or exports to the neighboring location. A neighborhood model object is effectively a view containing a subset of the information contained within each of the local asset models (see Figure 2).

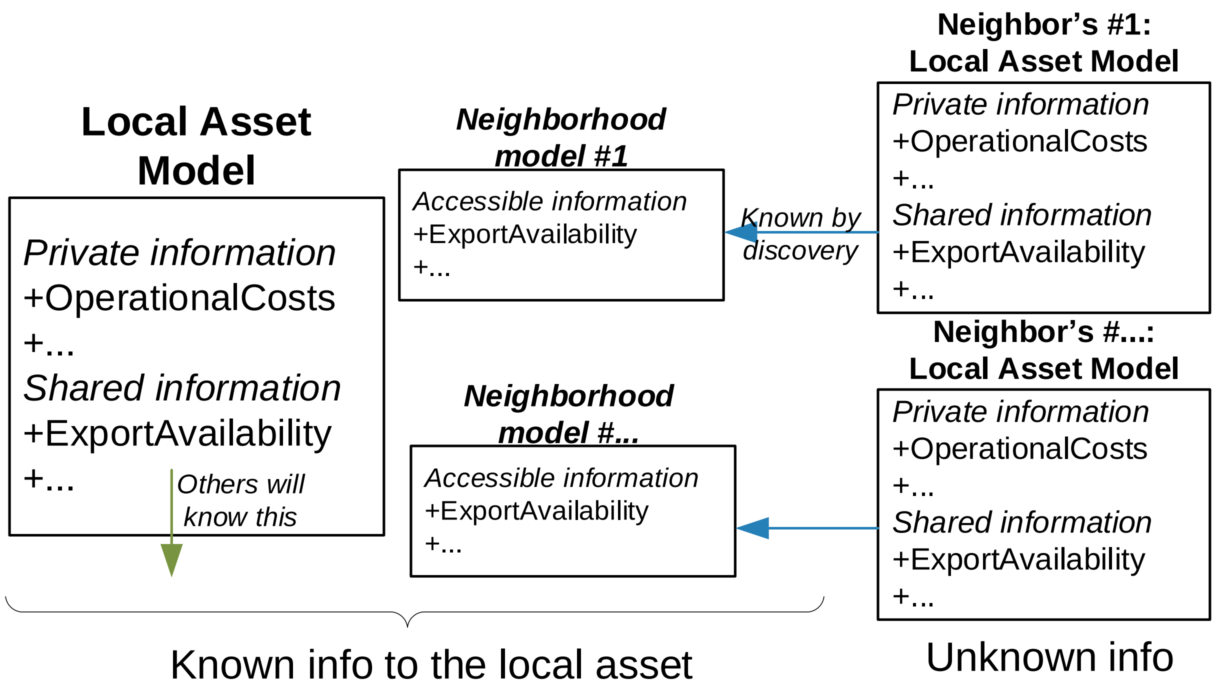


Figure 2. Comparing the local asset model and the neighborhood model.

In addition to the four key TENT components described above, other relevant objects within the TENT are presented in Table 3,. Their primary purpose is to assist with coordination and to provide agents with system-wide information that enables discovery to occur.

Table 3. Other relevant components within the TENT model.

Object	Intended Use	Object	Intended Use
TimeInterval	Used to define an event interval	*MarketState*	Used to keep track of the market state (e.g., its lifecycle)
IntervalValue	Measured quantity over specified interval	*MeterPoint*	An agent that is limited to reporting measurements
TransactiveRecord	Basic price vs demand curve for a given *TimeInterval*	*Vertex*	Point within a curve to define an operational point, e.g., the cost vs demand (in a tuple)
InformationServiceModel	A generic information provider within a TES. (e.g., climate, system net load, market agents and identities)		

A key contribution of the TENT is its market-handling characteristics. Within the TENT, a market is executed by an agent, and this market can receive a multitude of signals from one, a set, or all agents within the region, depending on the market structure that is being deployed. However, all markets rely on an the same eight-stage lifecycle, thereby creating a model that is agnostic to the underlying market theory of operation (e.g., it does not care about the price-discovery or clearance mechanism). This prototypical lifecycle is reproduced verbatim in Table 4, and it offers multiple benefits, as follows:

- Ability to develop event-driven TES implementations, this enables automation at the agent side, while enabling other agents to subscribe or monitor as needed, thereby helping to decouple individual actors' communication dependencies.
- Ability to perform market pipelining. This means that multiple markets can *trail* each other; for example, while one market is in *delivery*, the upcoming market is in *delivery lead*, disseminating market results so that agents can schedule and prepare accordingly.
- Ability to run parallel markets. The TENT does not limit the number of markets that can run at the same time within a single TES. This not only enables the execution of multiple commodity markets but enables the deployment of correction markets. This could enable real-time markets to correct for scheduling and forecasting deficiencies. An example diagram of this process is presented in Figure 3.

Table 4. Methods used by the market's state machine in the TENT v2, taken from (Hammerstrom 2019).

State	Methods Automatically Invoked	Triggers or Actions
Inactive		Initial state upon instantiation. Market is added to agent's list of active markets
	transition_to_active	Active period starts.
Active	while_in_active	
	transition_from_active_to_negotiation	Negotiation period starts.
Negotiation	while_in_negotiation	Negotiate.
	transition_from_negotiation_to_market_lead	Negotiation period ends.
Market Lead	while_in_market_lead	Collect market bids. Market calculations.
	transition_from_market_lead_to_delivery_lead	The market clears.
Delivery Lead	while_in_delivery_lead	Disseminate final market results. Prepare for asset controls.
	transition_from_delivery_lead_to_delivery	Delivery of market periods begins.
Delivery	while_in_delivery	Meter delivered electricity. Control scheduled electric power.
	transition_from_delivery_to_reconcile	Last delivery period ends.
Reconcile	while_in_reconcile	Reconcile transactions.
	transition_from_reconcile_to_expire	Market is reconciled. Market is removed from agent's list of active markets.
Expire	on_expire	A historical record may be kept

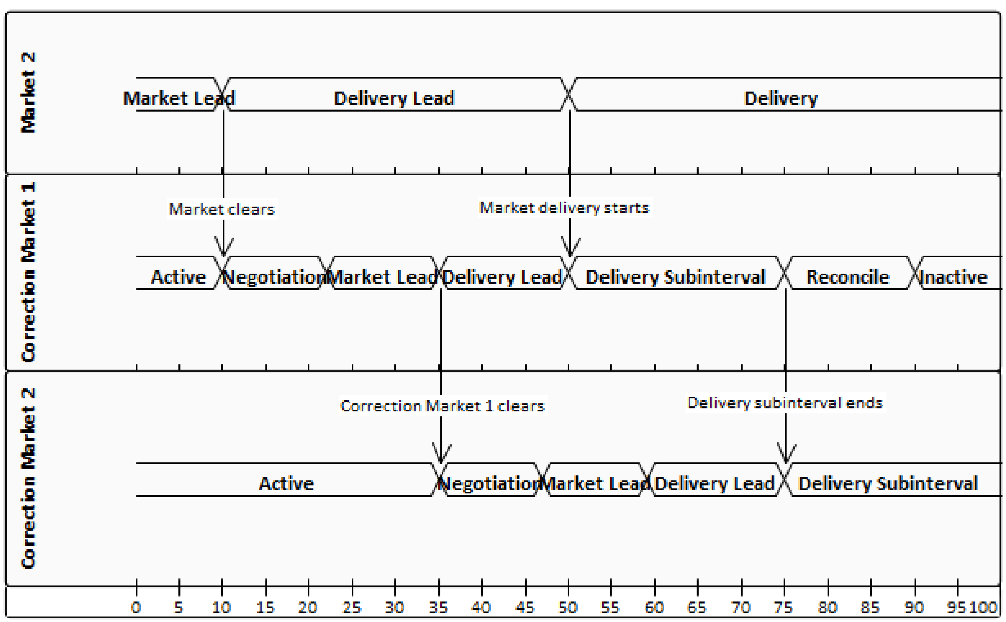


Figure 3. Hierarchical market architecture, demonstrating the use of correction markets, taken from (Hammerstrom 2019).

In summary, the TENT provides an interesting perspective toward achieving a template-based representation of TESs. It offers a clear decoupling of the physical grid devices and its transactive energy counterparts, which may prove useful when decoupling TES solutions from the simulation or physical grid components. An agent within the TENT is fully responsible for managing its own assets, while at the same time being required to expose a standardized interface, thereby greatly simplifying the interoperability of diverse systems. A known limitation of the current TENT model is its lack of data models to enable non-power demand-related asset exchanges; for example, no support for ancillary services or for derived markets exists. However, most of these limitations, if not all of them could be addressed by performing specializations over the base classes.

2.3 Supporting TES Services Using Blockchain, Explorations

As outlined by Section 1.0, TES solutions based on DLTs (e.g., blockchain) have shown great potential. However, most of the literature has focused on solving specific problems that may prove difficult to generalize to the entire TES domain. Furthermore, some fundamental questions regarding the applicability of blockchain technology remain unanswered. Figure 4 illustrates some of the pending questions, which represent some of the key unknowns identified during the development of this report. It is expected that these types of questions could be better understood and addressed if more experimentation/research is carried out.

As can be observed in Figure 4, these questions cover a multitude of domains, with a wide range of topics that are relevant to an ample spectrum of potential users—ranging from prosumers, stakeholders to software developers who must implement the solutions. For end-users, industry stakeholders, and asset owners, the most relevant topics revolve around trust, privacy, and fairness. Whereas for researchers and developers, the areas of interest include scalability, enforcement of contracts, and those related to identifying the best practices (cybersecurity, software development cycles, maintenance).

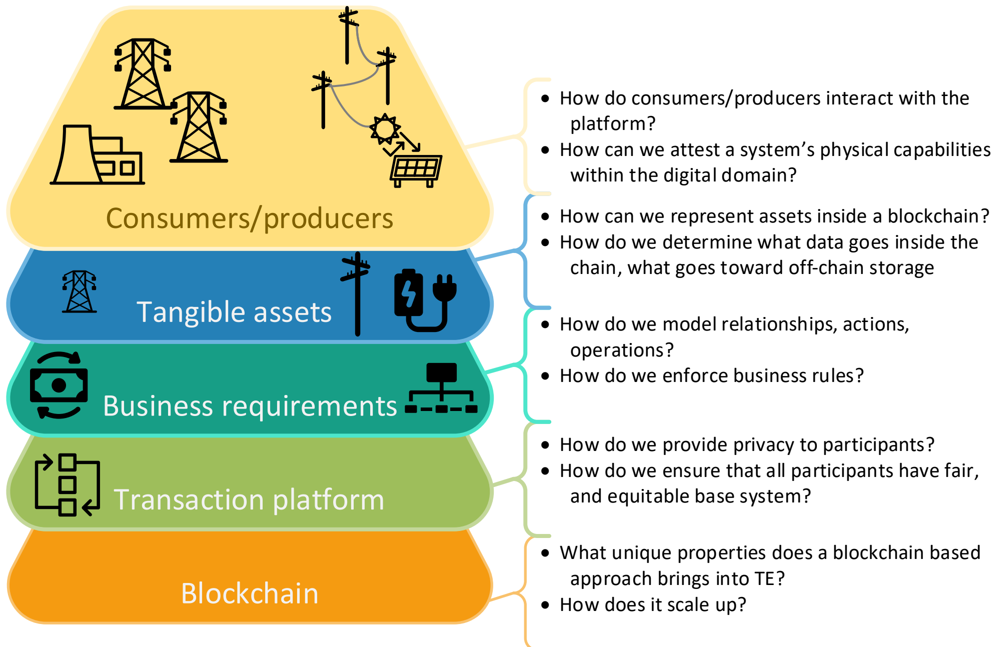


Figure 4. Open research questions related to blockchain applicability in the field of transactive energy.

To address these questions multiple researchers and working groups have started to develop models that abstract the functions provided by blockchain. For example, in (Lima 2018), the author presents a breakdown of blockchain technology using a hierarchical structure that can be used to map common application needs to blockchain services. Whereas in (Cali, et al. 2019), introduced an initial approach to mapping TES-based applications to a blockchain platform; the main takeaway is that blockchain can be used as the backbone to support the future grid needs (in the TES space), while also recognizing the need for standardization and the development of use cases.

Another exploratory work (Gourisetti, et al. 2021), presents a TES-oriented solution that uses blockchain technology to implement a series of processes and objects that enable agents to participate in a double-auction market. Although the demonstration section still falls under the specialization problem described above, its design was based on analyzing the engineering requirements of a TES and then mapping those requirements to the features provided by a blockchain-based environment, placing special emphasis on SCs, which are logical pieces of code that can execute on top of a blockchain. This procedure mimics some of the template work presented in previous sections, where emphasis is put on accurately representing the TES architecture rather than building a system that is only relevant to the particular problem.

Another area of interest within the Transactive Energy Systems field revolves around exploiting the enhanced trust capabilities that blockchain provides. Devices that choose to participate in such a system have the potential to increase the quality and trustworthiness of the operations in which they participate, thereby offering a competitive advantage to cases where participants are owned or operated by different organizations and no traditional trust relationship can be established. Such features have been explored in works such as (Eisele, Barreto, et al., Blockchains for Transactive Energy Systems: Opportunities, Challenges, and Approaches 2020), where a transactive energy network based on smart contracts has been developed, and (Tucker and Johnson 2021) where a mechanism for enabling local grid operators to preemptively evaluate and assess market transactions before they are submitted to the wholesale market.

Another approach to integrate a TES network with existing grid infrastructure is described in (Mokhtari and Rahimi 2021), the approach enables participants to perform peer-to-peer transactions with a distribution system operator having oversight capabilities. The system is based on a token-based system where energy producers are assigned tokens once the amount and source of energy have been validated, participants rely on these tokens to perform transactions. Under the author's premise, tokens can be pegged to an actual currency to enable operations with other external participants.

Similarly, in (Li, et al. 2019) the authors have proposed the use of blockchain for enabling transactive services in microgrid environments. The approach follows a multi-stage operational cycle that enables 1) DSO-level (distribution system operator) level oversight; 2) Participant registration; 3) Distributed state estimation and 4) Final settlement among participants. The goal of the paper is to create automated and traceable mechanisms that can increase efficiency reliability, and resiliency. Although the paper has an interesting approach, it fails to provide a reference implementation for other researchers to explore and build upon it.

As it can be observed from the previous paragraphs, there is a wide array of works that have relied on using blockchain for satisfying TESs requirements. Nevertheless, this list is not comprehensive and thus it is only used as an example of the work being performed across the research space. Although, performing an in-depth review of the existing literature is outside the scope of this report, additional reviews regarding the subject of TESs and blockchain technology can be found within (Gourisetti, et al. 2021).

2.4 Identified Gaps

Based on this short review, it becomes clear that proposing high-level solutions such as those reported by the TEAC and the TENT models can be beneficial to the majority of stakeholders. Key benefits of using a template-based architecture for TESs applications include:

- Supporting and encouraging participation through the use of highly standardized, neutral trading platforms.
- Enhancing failure resistance (due to centralized or single points of failure), by enabling agents to operate in a decentralized fashion.
- Enabling participants to freely negotiate based on their own preferences, thereby fostering competition for better services.

However, in order to achieve these benefits, a significant amount of work needs to be performed by organizations that seek to adopt this model, a task that can be compounded when novel technologies, such as blockchain disrupt traditionally accepted operational paradigms. Therefore, there is a need for creating tools that can facilitate this adoption. Specifically, we propose to develop a framework that extends the TEAC model to operate under a blockchain environment, offering organizations the ability to experiment with the technology without first having to allocate resources towards analyzing its properties and designing mechanisms to address its drawbacks. It is expected that this template system can be beneficial to organizations, software architects and potential developers by allowing them to accelerate deployment of TES-based applications based on pre-engineered templates.

This, however, should be done with care; in particular, the concepts of universality, interoperability, and extensibility should remain a priority over solving predetermined problems. Potential frameworks should not rely on the specific technological features of a product/offering to accomplish tasks. The developed templates should remain at a high level but must still provide insightful information to potential adopters to avoid confusion. Following this logic, in this work we propose a set of blockchain-agnostic reference templates that can be adapted to a wide variety of TES applications. The key contributions of this work are:

- providing stakeholders with a set of blockchain-agnostic templates that can accelerate the development of TES-based applications.
- being built around the concepts of universality, interoperability and extensibility; remain language and implementation agnostic by relying on UML diagrams.
- following an object-oriented approach, enabling them to be extended and composited to suit the end-application needs. We refer to this capacity as mix-and-match.
- being designed to leverage blockchain features, while at the same time avoiding the use of problematic features that can create performance issues (or are incompatible with the technology).
- presenting an attribute-based access control mechanism that has been specifically designed for blockchain environments.
- presenting a high-level, configuration example intended to represent a real-time market operating over a TES.

3.0 Blockchain Use in the Energy Domain

As mentioned in Section 1.0 of this document, blockchain is a relatively new technology that has received wide industry attention due to its potential disruptive solutions. Blockchain promises to facilitate the execution of contractual obligations between parties without the need for a dedicated backend system or a third party that enforces rules. The specific mechanisms that allow such a system to exist are beyond the scope of this work but can be found in the literature (Nakamoto 2008) (Buterin 2013) (Yaga, et al. 2018) . Nevertheless, to frame the characteristics of blockchain and create compatible templates, it is important to present a high-level summary of blockchain core concepts and components, which are as follows:

- **Blockchain categorization:** Blockchain systems can be largely grouped into permissioned or permissionless systems. Permissioned systems require participants to establish and maintain identities before access to a network is granted, ideally creating a trust relationship that can simplify the consensus mechanisms. Permissionless systems represent some of the most common deployments; they enable a truly decentralized operation where decisions follow the behavior dictated by the majority of agents, under the assumption that most agents are ad hoc, independent, and cooperate toward achieving a common goal.
- **Peers:** Within blockchain terminology, a peer is a computational node that is capable of engaging in blockchain activities (subject to authorization). Common activities include being able to get a copy of the data, participating in the consensus decision-making process and submitting transactions. Actual authorizations vary depending on the blockchain implementation and pre-configured access permissions.
- **Data blocks:** Within blockchain, a data block represents the most basic unit of data storage. Individual blocks contain copies of the intended data, along with a digital fingerprint that can be used to verify its authenticity.
- **Block chaining/aggregation:** Blockchain derives its name from the digital-chaining mechanism used to tie blocks in a sequential manner by relying on cryptographic functions to link them together. In a more general sense, DLTs expand this concept to refer to the mechanisms that are used to aggregate data blocks into a data structure that prevents data modification, thereby providing immutability.
- **Ledger:** Once data blocks are aggregated into a data structure, they become part of the ledger. The ledger is intended to be distributed across participating peers (for replication purposes), and provides the open, verifiable platform that makes blockchain attractive for applications that seek transparency.
- **Transactions:** Requests to input and retrieve data from the ledger are submitted to a blockchain system via a transaction request. For most blockchain implementations, transactions that led to writes into the ledger require consensus to be achieved before changes are committed; read operations do not need this consensus to occur and can be obtained from cached versions of a ledger. The number of transactions a blockchain implementation is able to commit per unit of time is an important metric that has a direct impact on application scalability.
- **Consensus models:** Consensus models dictate the mechanisms that agents use to agree on the *global system state*, such as determining the data blocks and the order in which they are committed to the ledger. These mechanisms can be selected based on the nature of the blockchain (permissioned vs permissionless) and the desired speed or security traits that are needed to satisfy the application requirements. Examples of common consensus

algorithms include Proof-of-Work (for permissionless systems), Proof-of-Stake, and Proof-of-Time, among many others (for descriptions, see (Yaga, et al. 2018) and (Cali, et al. 2019)).

- **Smart contracts:** Because consensus models enable blockchain technology to agree on a *common state*, the underlying mechanisms can be extended to support agreement at the logical level without needing to write the state into the ledger. This is achieved by essentially deploying a state machine that transitions to a new state only if a consensus is reached. The complexity of the code that can be deployed into a blockchain (if at all supported) remains implementation-specific.
- **A public ledger:** Perhaps one of the most publicized features of blockchain is its ability to store data in an immutable data source, known as the ledger. This ledger usually remains public and accessible to all participants (thereby enabling them to validate its integrity). It is important for application developers to choose what information needs to be stored in the ledger and what can be sent to an off-the-chain storage mechanism, because large or rapidly expanding ledgers can be problematic to maintain.

As can be deduced from the previous list of core concepts and components, blockchain is a complex ensemble of systems that requires careful selection of components to ensure that the target application needs are satisfied. Specifically, for grid-related applications, authors seem to agree that a permission-based blockchain is the preferred option, because it enables system operators to still maintain control over which agents can participate within the network. Furthermore, the consensus mechanisms can be more computationally efficient if a trust anchor can be leveraged (i.e., by creating an identity based on a physical interconnection point).

The second characteristic that may be relevant to a TES is a platform's SC capabilities. Although SCs are not mandatory (because external systems can query, process results, and eventually post results back to the blockchain network), they may prove useful in simplifying the decision-making process, while providing strong guarantees of the validity of the decisions. These guarantees are achieved by enabling participating peers to explore the code and execute their own copy of the contract within their own system (however, their execution results must agree with other systems to achieve consensus).

Common blockchain implementations include Ethereum and Hyperledger fabric. Ethereum is often cited as a public, permissionless network, although it can also be deployed on a private network. Ethereum is primarily based on a Proof-of-Work system (currently transitioning to Proof-of-Stake), and it requires agents to expend *gas* to propose a new block. *Gas* in this context refers to a transaction fee paid by users to execute a transaction, a fee that is based on its computational complexity. However, private networks can set their own *gas* requirements, effectively resulting in a no-cost network. Ethereum can run SCs using a custom language and a dedicated virtual machine that runs on each of the participating peers. This is a simple, but powerful interpreter-based solution that is a quasi-Turing-complete machine (limited by the number of instructions that a network allows) (Antonopoulos and Wood 2018).

Hyperledger fabric is a project backed by the Linux foundation that intends to provide a modular architecture that can be configured and adapted to suit an application’s needs. It offers a permissioned network, where participants need to register and obtain a digital identity from a Membership Service Provider (MSP) before they are allowed to access the ledger or invoke/process transactions. Multiple MSPs are supported, under the assumption that peers may belong to different organizations, and each organization must be capable of vetting their own members. Hyperledger fabric operates under a channel-based architecture, where multiple, parallel networks can exist. Individual peers can be part of one or more channels, and peers can execute SCs using one of the currently supported languages (GoLang, Java, and JavaScript). Communication among peers is restricted to members of the same channel and can occur via traditional IP/Hostname addresses or a dedicated gateway service that can interconnect peers across multiple networks or organizations. A detailed overview of the Hyperledger Fabric architecture is presented in Figure 5.

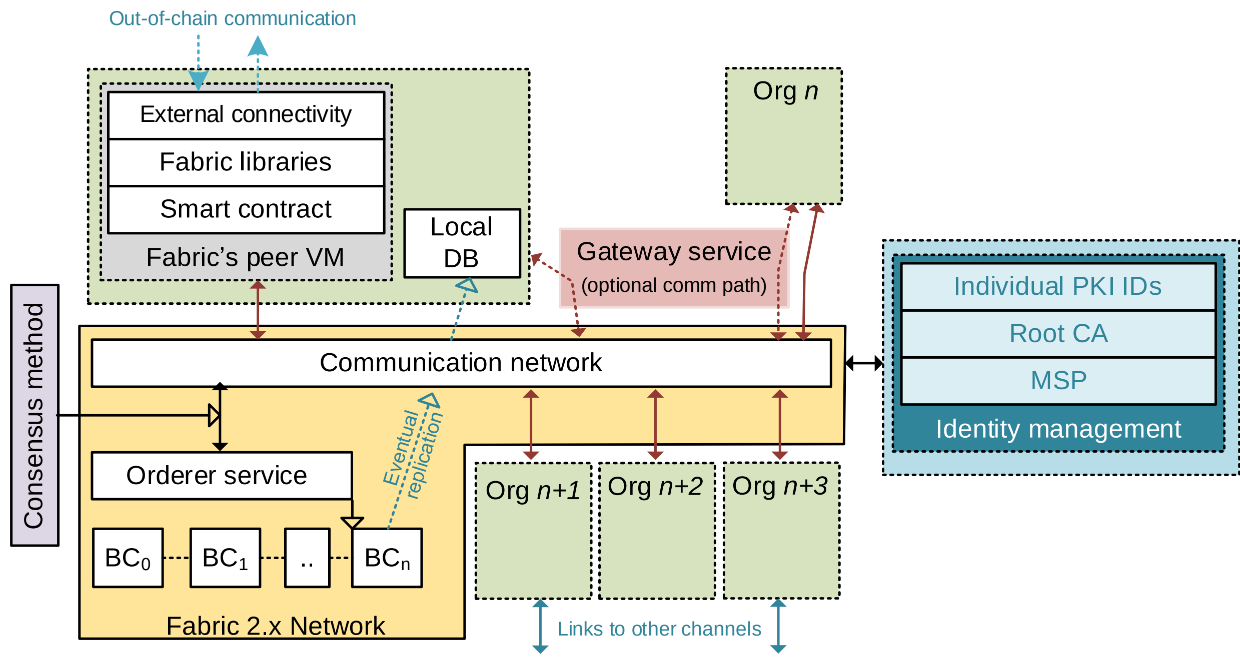


Figure 5. Detailed, service-level components present in a typical Hyperledger fabric deployment.

In the next sections, an in-depth, blockchain-agnostic review of features (and limitations) introduced by blockchain technology that are applicable to TES applications is developed. These features revolve around three main areas of interest: (a) cybersecurity; (b) performance characteristics, and (c) potential pitfalls.

3.1 Cybersecurity Characteristics of Blockchain-based Environments

One of the competing reasons to adopt a blockchain-based platform for TES is to increase the cybersecurity properties of the target application. Commonly cited features include its immutability and its decentralized, consensus-based decision-making abilities. However, these features may not provide a complete picture; for example, immutability does not mean permanent data protection (because deleting the ledger will cause data destruction), but instead refers to the ability to detect data modifications. Similarly, consensus, does not imply correct consensus; in this case a majority-based decision made based on an incorrect data set or by a faulty algorithm will lead to an incorrect ledger state from the perspective of the application. Nevertheless, Table 5 presents some of the properties provided by blockchain and the expected benefits (at the TES level).

Table 5. Cybersecurity characteristics of blockchain environments and their applicability to TESs.

Blockchain Characteristic	BC Domain Descriptions	Potential TES Benefits	Potential TES Drawbacks
Authentication	Writing rights can be enforced by an Identity Service Provider (ISP); also known as a Member Service Provider (MSP).	Access to blockchain applications/operations is permissioned.	Requires blockchain network operators to whitelist participants, creating the possibility of discrimination.
Data privacy	Blockchain does not natively support data privacy; applications can selectively protect data.	Transparency	Most operations can be observed once a copy of the ledger is obtained, and can lead to the loss of competitive advantages,
Data integrity	Blockchain data blocks are digitally signed, enabling peers to verify the integrity of the data,	As long as the TE has access to the live, committed version of the chain, integrity can be assumed,	Mechanisms to ensure connectivity must be provided, otherwise there is a risk of operating with inconsistent data,
Data confidentiality	Blockchain can store data that have been previously protected.	Ability to store business sensitivity data	TE applications must selectively protect data based on their risk profile, limits transparency.
Nonrepudiation	Because writing can only occur after an ISP does Public Key Infrastructure (PKI) validation, an agent cannot deny a signed operation.	Participants have reasonable guarantees to trust ledger-stored, third-party data.	Mechanisms to secure private keys must be considered (credential management).
Data provenance	The current block state depends on the previous blocks.	An operation can be recursively traced back to its previous state, up to a genesis block.	Off-the-chain events cannot be fully secured, only validated against the stored fingerprint.
Distributed consensus	The current database state reflects the agreement of participating peers, based on a predefined set of rules.	The state has been agreed upon by all endorsing peers, increasing the trust level.	Consensus does not imply correct consensus. Mechanisms to correct incorrect ledger states must be developed.
Noncentralized database	Multiple, replicated databases are stored across all participating peers.	Risk of data loss is minimized; snapshot capabilities are already built into the database.	There is a risk of rollback if the application happens to be in an invalid fork.
Smart contract	A peer's behavior can be defined by a smart contract. A "logic" set of steps executes at the peers, and the results are deterministic and should yield the same result across all peers.	Business definitions can be specified in a programmatic manner. Any Turing-complete algorithm can be deployed.	True randomness cannot exist. All algorithms must be deterministic (steps, inputs, and outputs),

3.2 Performance Characteristics of Blockchain

A key interest related to blockchain is determining its ability to scale-up—particularly when it is subject to large amounts of data or transactions, as typically experienced in a real-world TES deployment. To date, there is no clear source of information or procedure that can be used to accurately predict the performance of a TES application just by specifying a blockchain platform. However, certain properties are empirically known and are widely accepted; for example, it is generally acknowledged that permissionless (PL) networks can take a long time to commit to the ledger, and also require a significant amount of computational power to reach consensus. In contrast, permissioned (P) networks are designed to be time and resource efficient at the cost of requiring participants to register before participation is granted.

Based on these limitations, Table 6 was developed to serve as a generic benchmark assessment tool to aid developers in comparing different solutions. In this case, the first two columns are used to describe key performance metrics, while the third column describes a metric's potential implications within the TES domain; Finally the fourth column is used to provide sample metrics to the reader. These metrics are categorized according to the underlying blockchain architecture (*P* is used to denote permissioned blockchains and *PL* is used for permissionless blockchains), the reported metrics have been adcquired from multiple sources, including: (Zheng, Yongxin and Xueming 2019), (Kuzlu, et al. 2019) and (Mylrea, Gourisetti and Culley, Keyless Infrastructure Security: Technology Landscape Analysis Report 2018).

As mentioned earlier, the only reliable mechanism that can be used to gauge a given blockchain performance is to perform experimentation. To aid in this aspect, some tools such as *Blockbench* and Hyperledger *Caliper* can be configured to capture low-level performance metrics from a wide array of blockchain implementations (Wang, Ye and Xu 2019). However, metrics can only be captured once a proof-of-concept or beta version is deployed, a task that is hard to achieve on its own. However, certain approximations can be made if the use case characteristics are known. For example, if the target goal is to implement a real-time market mechanism, then the chosen blockchain system should be capable of:

Table 6. Typical performance metrics of blockchain environments and their impacts on TESs.

Metric	BC Domain Descriptions	Potential TES Impacts	Reported metrics
Transaction throughput	This is the maximum number of transactions that can be processed by the system per unit of time,e.g., Transactions per Second (TPS).	This number may limit the number of TE transactions that can be run per day. The number of TE transactions a system requires is dictated by the number of agents and the periodicity of the transactions.	Hyperledger (*P*): Thousands of TPS Ethereum(*PL*): Hundreds of TPS IOTA(*PL*):~100 TPS
Transaction enqueuing time	This represents the time to submit a transaction, until all agents are ready to start execution.	This acts as a delay in getting a TES transaction processed.	
Consensus time	This is the time needed to reach consensus among peers and is usually lower	This acts as a delay in getting a TES transaction processed. Note that in certain blockchain	Hyperledger(*P*): <10ms

	for permissioned networks.	environments the consensus time is not time-bound.	
Block commit time	Once a consensus has been reached there could be a delay in putting the data into the ledger due to ordering and data replication delays.	This acts as a delay in getting a TES transaction processed.	Hyperledger(*P*)- 1s Ethereum(*P*)-12s Bitcoin (*PL*)-10 minutes
Smart contract execution time	This metric will be tied to the complexity of the SC, the platform overheads, and the language characteristics.	This acts as a delay in getting a TES transaction processed but is a factor that can be managed by optimizing the logic/code.	Hyperledger(*P*)-Dependent on program logic & used language. Ethereum(*P*)-Dependent on program logic. Bitcoin (*PL*)-Not supported
Smart contract memory footprint	This metric will be tied to the complexity of the SC.	High memory requirements may act as barrier for smaller, less capable participants.	Hyperledger(*P*)-Dependent on program logic & used language. Ethereum(*P*)-Stack is limited Bitcoin (*PL*)-Not supported
Average Read speed	This metric represents the average time needed for retrieving and validating data from the ledger. This number can be quite small.	Although directly performing ledger reads may be fast, requiring additional processing via smart contracts may add significant overhead costs.	Dependent on smart contract logic, connectivity, but usually order of magnitudes faster than write operations (no consensus required)
Average Write speed	This time is the accumulation of multiple metrics such as consensus time, block commit time, and SC execution time.	The write speed, or the time to achieve ledger commitment, may limit the number of operations if subsequent processes require such information to be present in the ledger to continue.	Hyperledger(*P*): Tens of seconds, worsens as TPS increase Ethereum(*PL*): ~ 100 seconds IOTA(*PL*):~ sub second

3.3 Potential Pitfalls of Blockchain-based TES Solutions

As described earlier, blockchain offers multiple benefits to applications that wish to leverage its features. However, it also introduces certain limitations that may affect the ability of a TES to fulfill its duties. Based on previous experiences, a non-exhaustive list of potential issues is shown in Table 7. Within this context, a topic that has received community interest is the implications and engineering decisions that must be considered before deciding on using on-chain or off-chain data storage mechanisms. Broadly speaking, existing grid applications produce vast amounts of data (e.g., data from Advanced Metering Infrastructure), making some of them impractical to be stored in-chain. Nevertheless, enough metadata must be stored on the chain to ensure that either 1) data integrity or 2) data recovery will be possible in the event of a system compromise or malfunction. An example, of how such engineering decisions could be applied to determine the data (or metadata) that must be stored is exemplified in (Sebastian, et al. 2021).

Table 7. Potential impacts/drawbacks of using a blockchain environments in TES applications.

Metric	BC Domain Descriptions	Potential TES Impacts
No consensus guarantees	Blockchain systems operate over distributed networks, and as such cannot guarantee that consensus will be reached.	Fail-safe defaults must be considered in case a lack of consensus state is reached. These may include setting default prices or following previous dispatch orders.
Ledger forking risk	Agents operating in a fragmented network may reach an inconsistent state that does not match the global state.	To detect these conditions, mechanisms for comparing the number of agents that are currently participating in the consensus against the expected number of agents.
Delayed commitment time	There could be a significant delay in getting a value committed to the chain.	A TES operational timeframe should be greater than the maximum commitment time to ensure that the blockchain can keep up with the TES.
Smart contracts are restricted to deterministic algorithms	Smart contracts must follow deterministic procedures to ensure eventual consensus. This also includes providing the same data inputs to all agents.	Certain functions, such as true random number generators are not possible. Data inputs should remain the same for all peers. e.g., `getWeatherAt(time=11h)` is preferred over `getWeatherAt(time=Now())` because the `Now()` can be resolved differently by peers.
Risks of relying on off-chain systems	Blockchain can only guarantee the integrity of on-chain systems. An external system can become unavailable or repudiate information previously provided.	Special considerations to securely link external systems to the blockchain must be developed. The overall security of a blockchain application is defined by the weakest link/component in the solution.
Risks of off-chain storage	Data stored off-chain can only be validated for integrity purposes.	In case the off-chain storage gets compromised, there is no built-in mechanism to recover it. The only mechanism provided by blockchain is to verify the integrity of the data.

4.0 The Blockchain-aware TES Template Model

As mentioned in Section 2.4 there is a need for developing specialized blockchain-aware templates that enable stakeholders to implement TES-based solutions using blockchain. These templates should aim to ease overall development, aiding stakeholders in reducing the time required to transition from an idea to implementation, while also providing mechanisms that allow different entities to compare their implementations using uniform models, in addition to enabling interoperability that reduces the number of problem-specific solutions. This section walks the reader through the proposed templates by first discussing the overall approach and then exploring the template constructs in detail.

4.1 High-level Overview

The objective of this research is to develop artifacts that can help researchers follow a methodical and engineering-based approach when using blockchain to implement relevant TES applications. This research is performed in a DLT-agnostic fashion. Therefore, the research artifacts will be applicable for any TES application that intends to use blockchain or DLT. The proposed templates are grouped into five categories that follow the model described by (Widergren, Transactive Energy for Distributed Resource Integration 2016), and a high-level overview of this process is presented in Figure 6. The next section presents the details of this five-stage model.

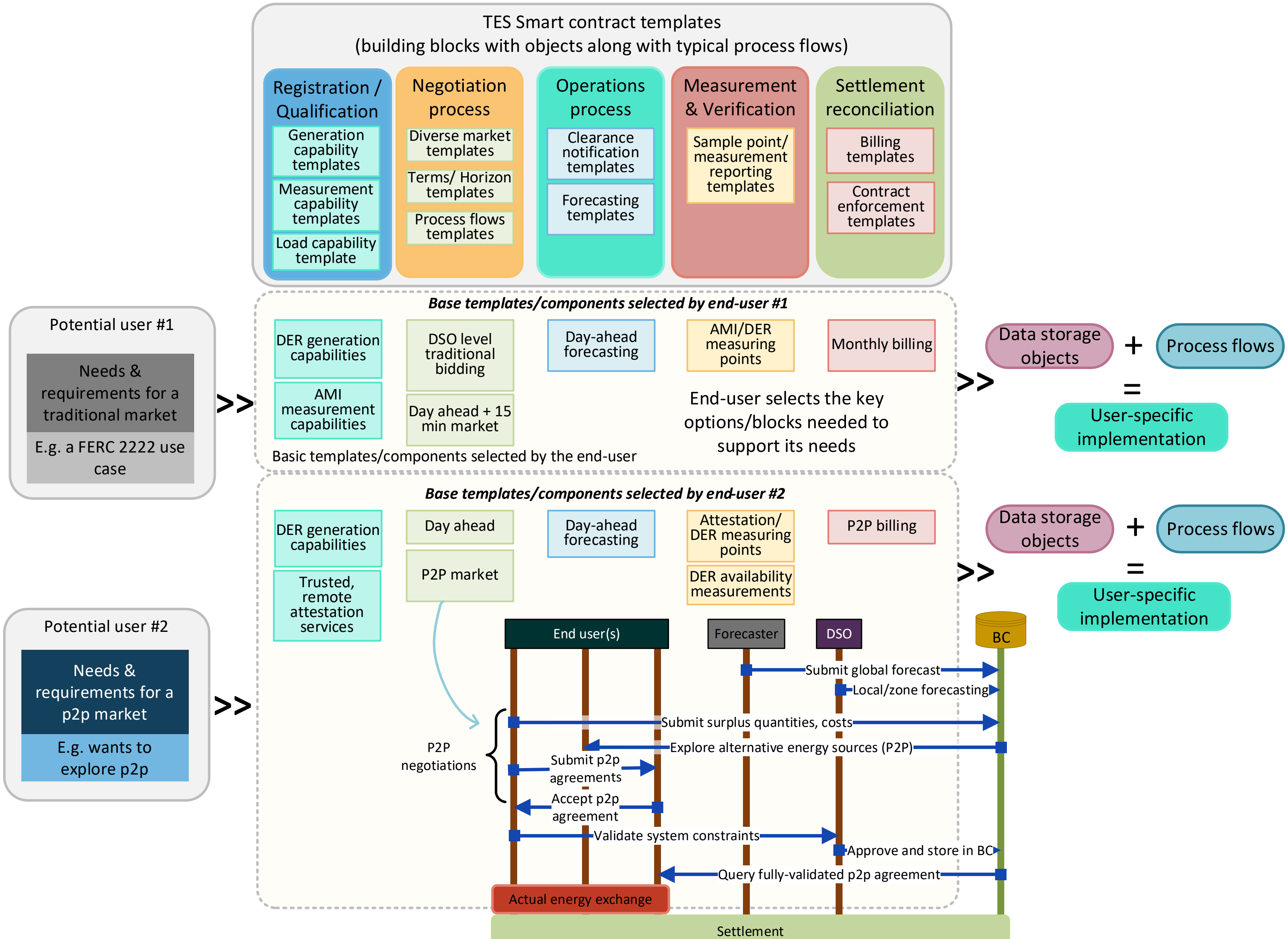


Figure 6. The blockchain-aware TES template model.

4.2 The Five-stage TE Model

For the purposes of this report, the team decided to leverage the transactive node model, which introduces the concept of the Energy Service Interface (ESI) (Widergren, Interoperability Strategic Vision 2018) to serve as the interface between a local node and other neighboring transactive nodes (see Figure 7). To achieve its operational lifecycle the ESI relies on a five-stage process that is intended to be operated over a continuous loop (except for the first stage), and the stages are not necessarily serialized (e.g., an agent may skip certain steps if needed). A brief summary of each stage is summarized below.

1. **Registration/Qualification:** In this stage the transactive node gets registered into the system and its capabilities are recorded, vetted, and approved following the procedure defined by the system operator
2. **Negotiation process:** At this stage the node discovers other neighboring transactive nodes and starts a negotiation process. This can involve tasks such as placing and waiting for bid clearance, but it can also include the discovery of new services.
3. **Operations process:** This stage represents the actual asset exchange process; it is expected that most of the action happens at the physical level (or within the simulation domain if this is the case).
4. **Measurement and Verification:** During this stage measurements are taken to ensure that the negotiated terms are effectively being followed by the participants. Although this process is logically represented after the operations process, in practice, this process should occur in parallel to it, to ensure that measurements capture the actual asset exchange progress.
5. **Settlement and Reconciliation**: Using the information established during the negotiation stage and the measurements collected, the settlement operation occurs by taking into account the deviations between the agreed-upon quantities and the measured/verified quantities. The value assigned to this deviation is determined using the agreed-upon contract negotiated during the second stage.

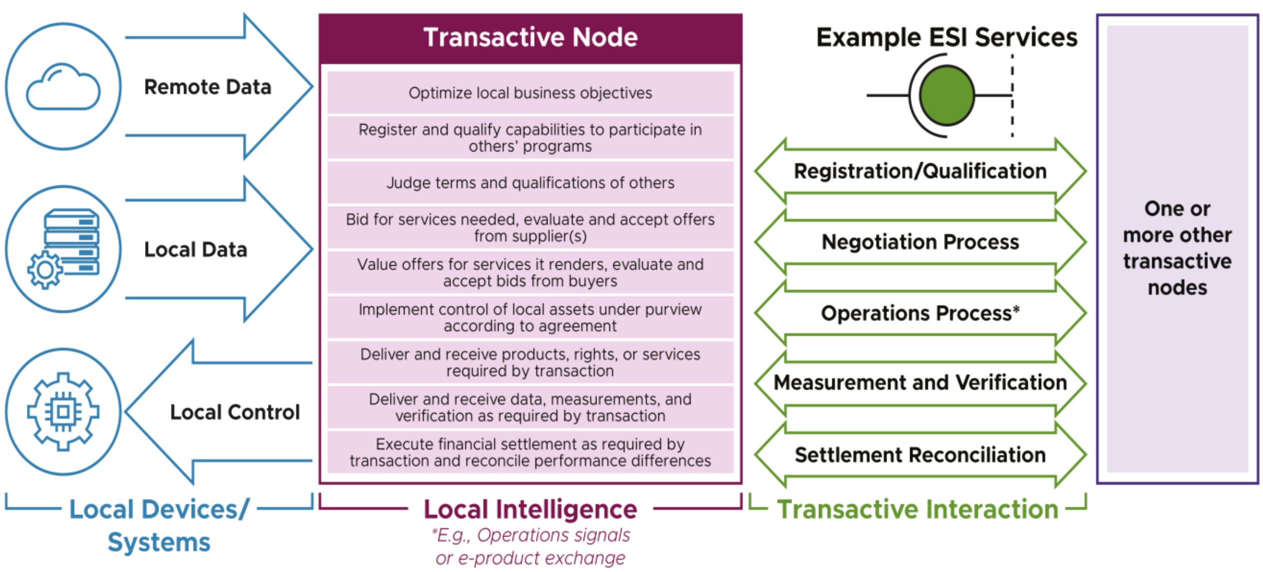


Figure 7. The transactive node model as represented by (Widergren, Transactive Energy for Distributed Resource Integration 2016).

4.3 Basic Data Types

In this section a collection of basic data types that are used within the template are introduced. The details of these objects are summarized during the main body of this document, only presenting an overview of their responsibilities and capabilities along with high-level diagrams. Low-level details of these objects can be found in the appendix section of this document.

4.3.1 Basic object models

The diagram/package shown in Figure 8 provides an overview of foundational classes that are used as the base constructs for the Blockchain Architecture Transactive Energy Systems (B-A TES) template framework. These elements work together to 1) uniquely identify objects; and 2) support an event-based architecture.

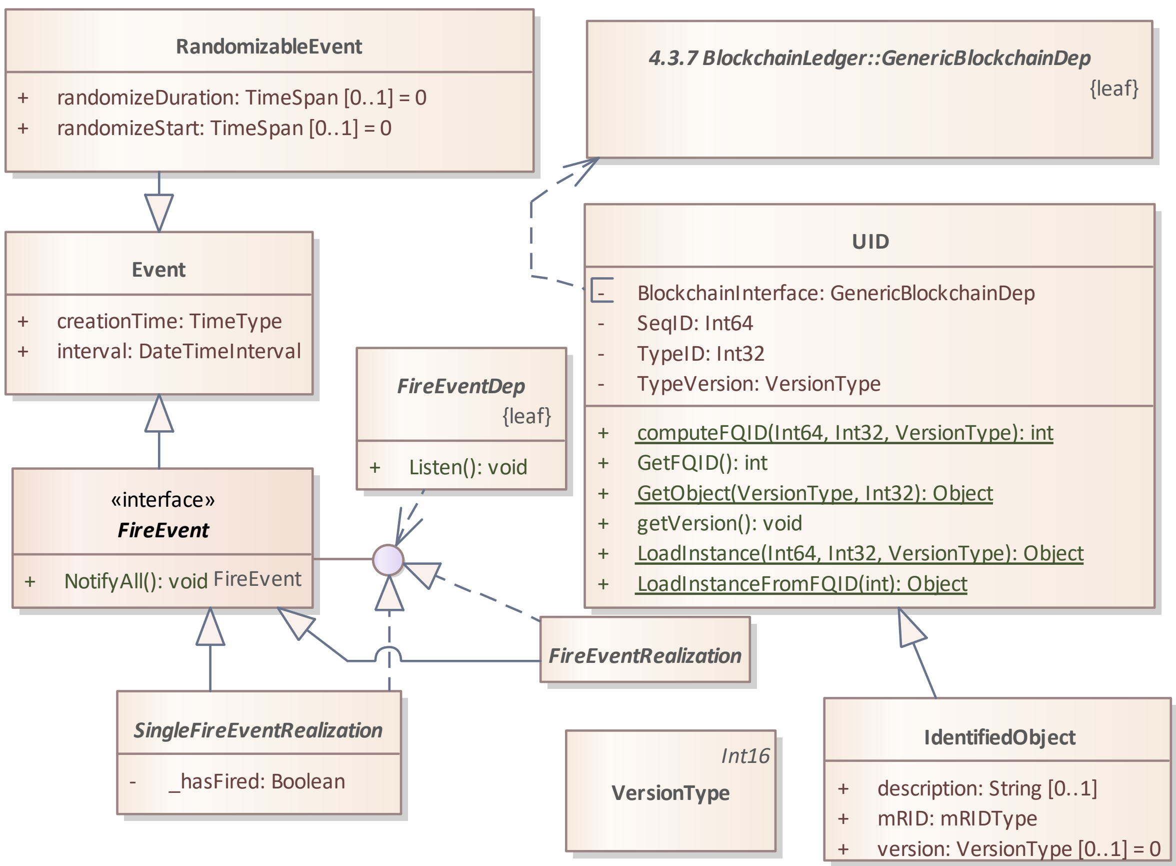


Figure 8. Overview of the *BasicObject's* package components.

4.3.2 Primitives

The diagram/package shown in Figure 9 presents an overview of data primitives. These data type models are intended to support a wide variety of data needs, provide generic structures and in general support more complex template models. In addition, they provide clearance on the size, supported operations, and intended usage.

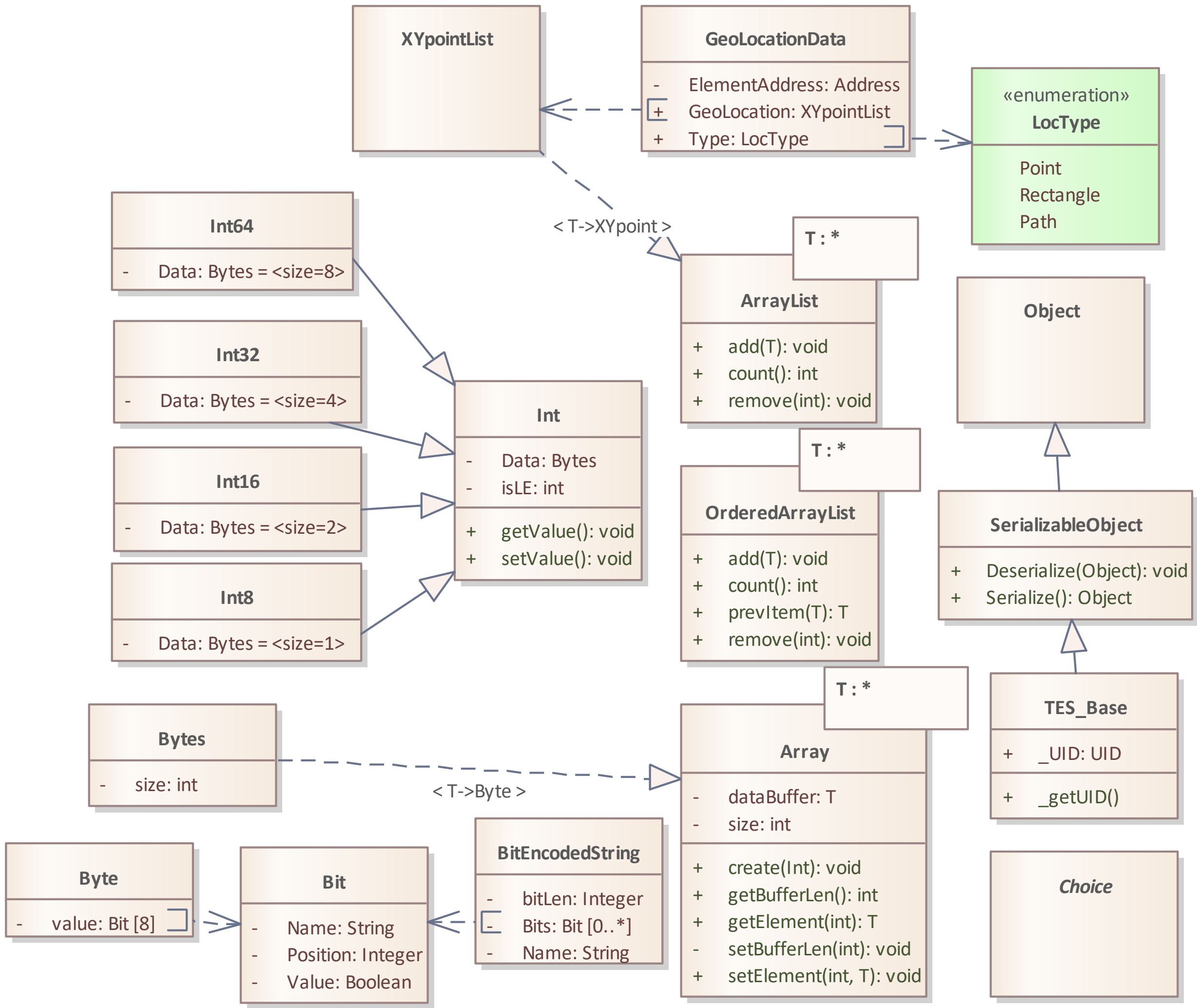


Figure 9. Overview of the *Primitive*'s package components.

4.3.3 Time Objects

The diagram/package shown in Figure 10 presents an overview of time-related constructs that will be used in this work. The data models are intended to support a common ground, in which all applicants are subject to the same time references and requirements.

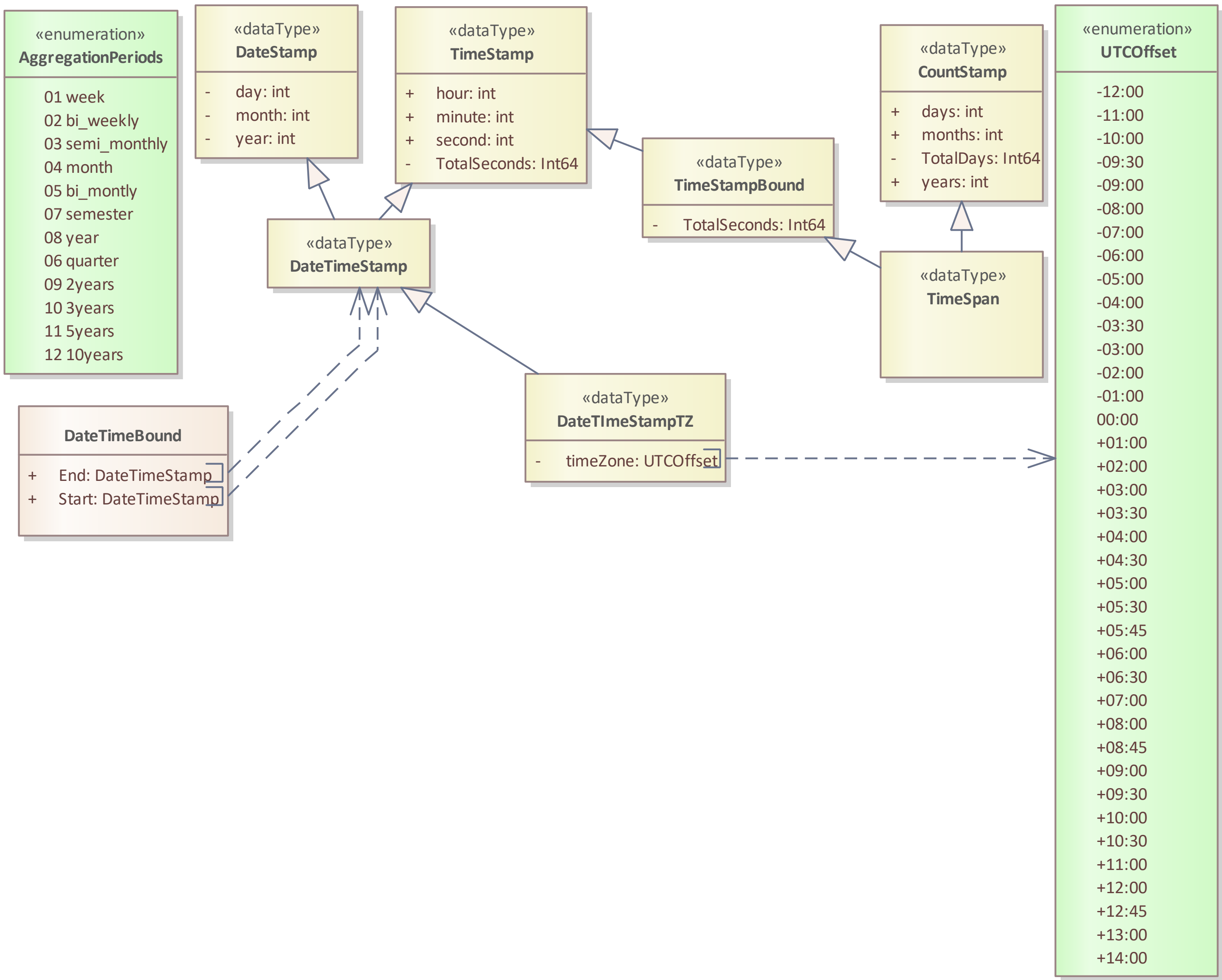


Figure 10. Overview of the *time* objects components.

4.3.4 Math

The diagram/package shown in Figure 11 groups objects that have mathematical or engineering applicability but are usually not natively supported by programming languages. This diagram may be expanded in the future to introduce more definitions.

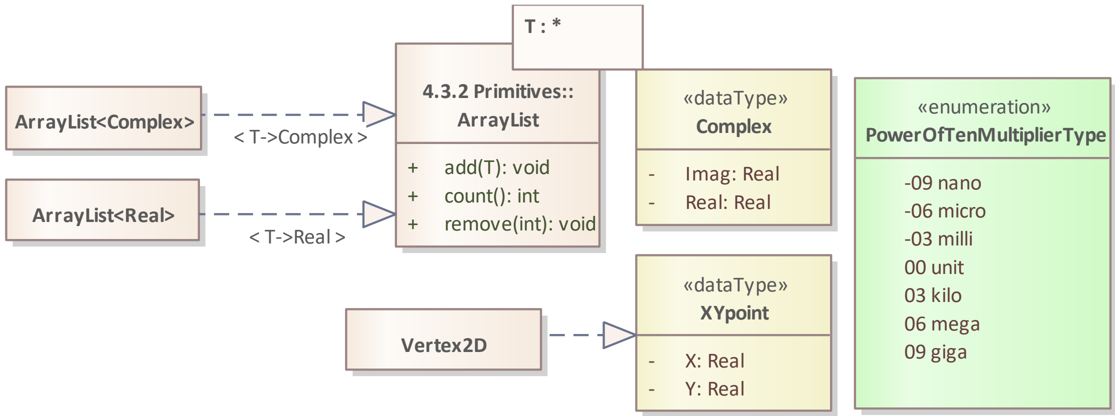


Figure 11. Overview of the *Math* components.

4.3.5 Trackable Objects

In this case, the diagram/package (see Figure 12) contains classes that can be used to track an object's changes through the trackable interface. Objects that inherit these interfaces can be used to track an instance's particular history across its lifecycle. The interface can be used to store a full copy of the previous state (e.g., when a ledger is present), or two store a hash only (for off-chain applications).

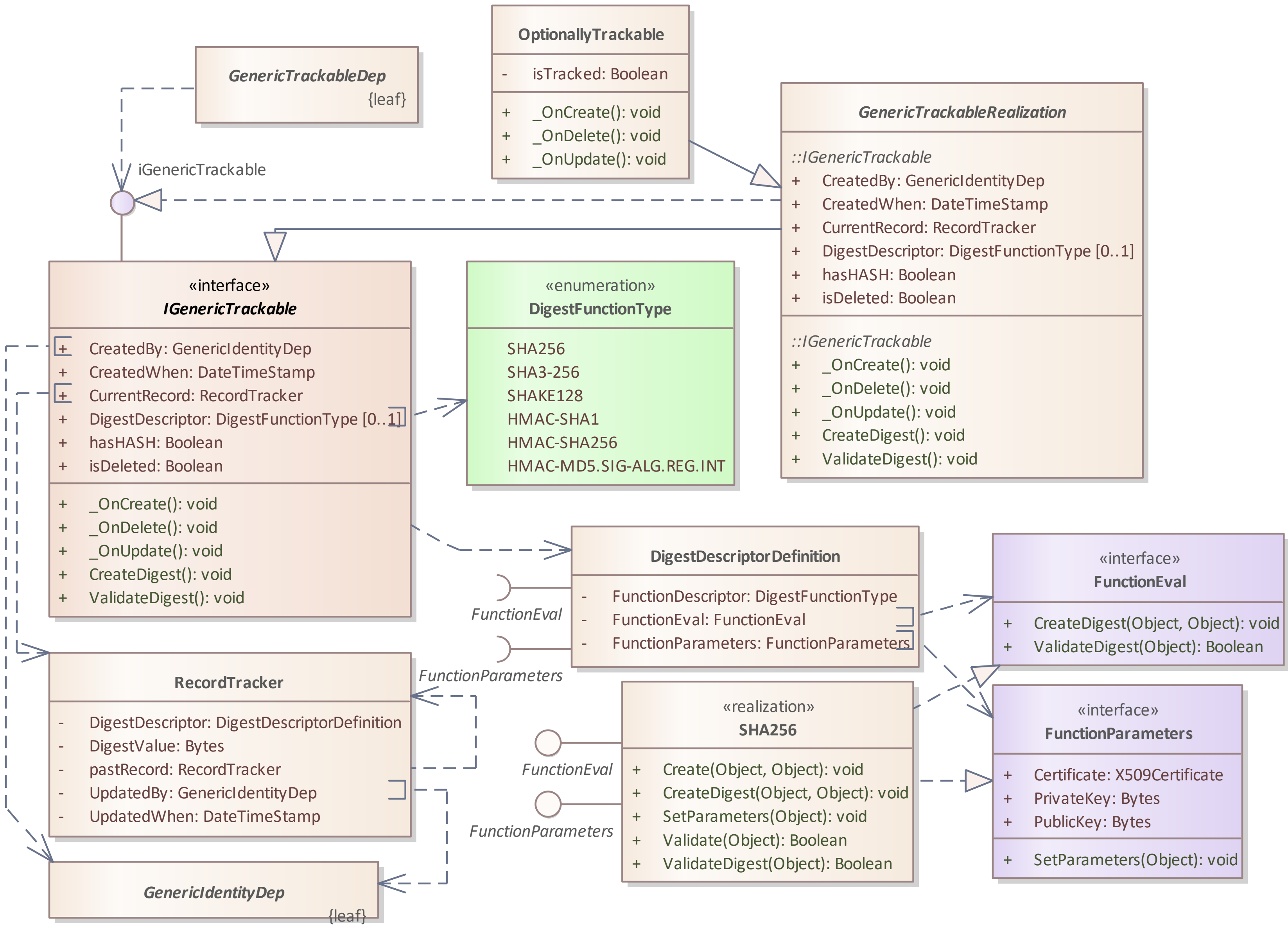


Figure 12. Overview of the *TrackableClass’* package components.

4.3.6 Digital Certificates

This diagram/package diagram shown in Figure 13 and Figure 14 contains assets that can be used to create a secure, digital representation of a subject's identity. This digital identity relies on the X.509 certificate model described by rfc5280 (Boeyen, et al. 2008). The presented interface allows static validation (by walking the certificate tree), and an online verification mechanism that checks for revoked certificates using *Certificate Revocation Lists.*

4.3.7 Blockchain Ledger

This package/diagram shown in Figure 15 provides the basic representation of a blockchain environment. It is intended to serve as a reference for software engineers implementing this technology. Many of the blocks presented in this diagram must be overridden, extended or otherwise adjusted to correctly represent a blockchain environment. Nevertheless, the basic read/write functionalities, and an immutable ledger must continue to hold.

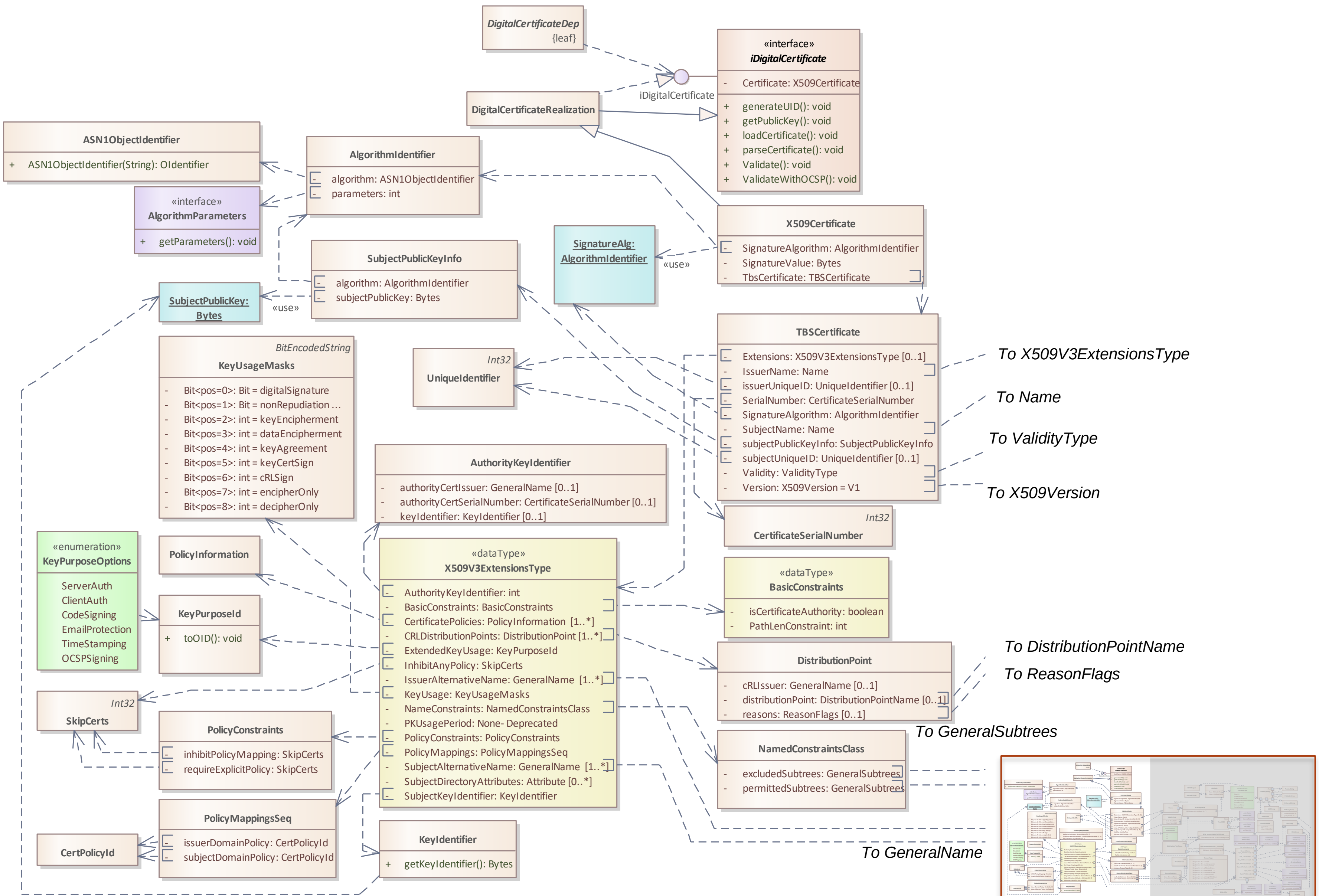


Figure 13. Overview of the *DigitalCertificates'* package components (Left side).

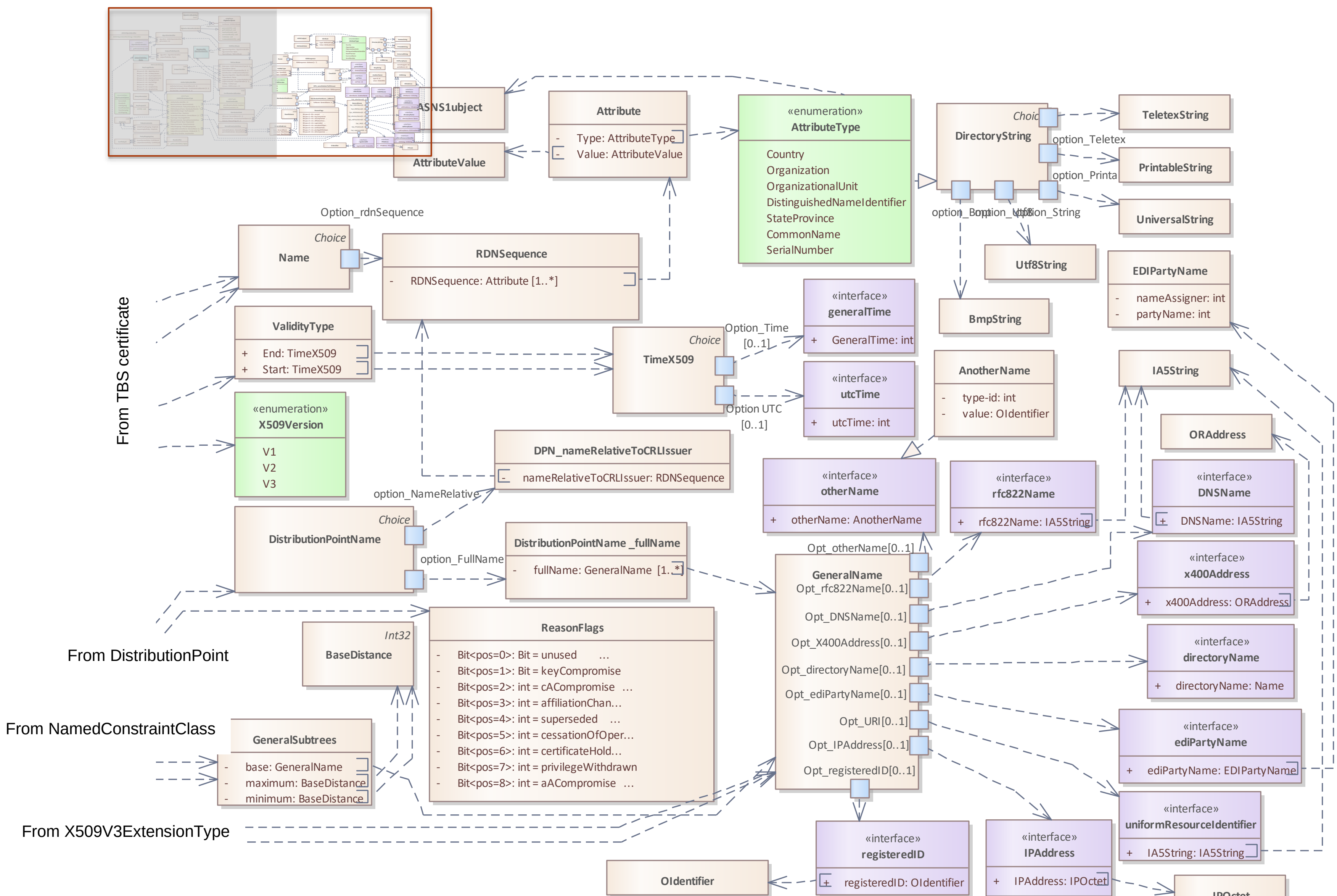


Figure 14. Overview of the *DigitalCertificates'* package components (Right side).

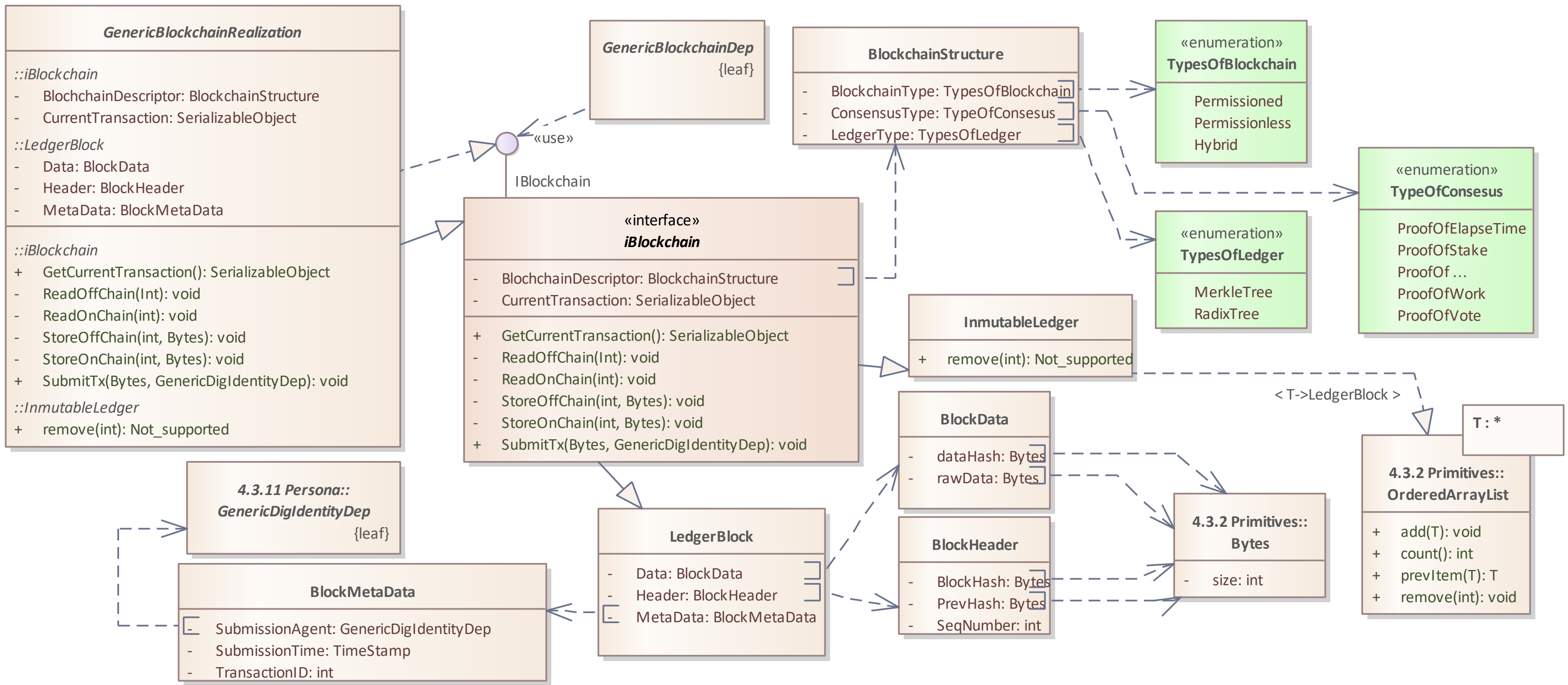


Figure 15. Overview of the *BlockchainLedger*'s package components.

4.3.8 Lifecycle Management

This diagram/package shown in Figure 16 contains the necessary object templates and interfaces required to track an asset's state across its operational lifecycle. The provided interface is designed to function in an ad-hoc behavior (e.g., on demand). This contrasts with the mechanisms provided by the *trackableInterface* which are designed to be inheritable (and thus always tracking the changes in state).

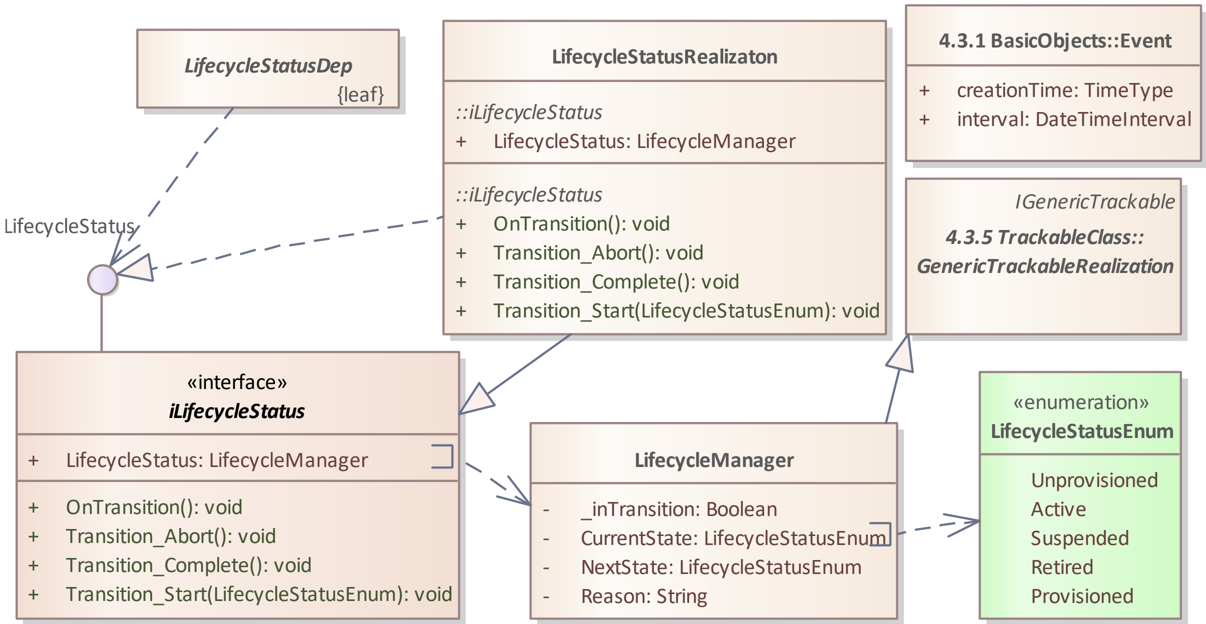


Figure 16. Overview of the *LifecycleManagement*'s package components.

4.3.9 Permissions and Qualifications

The diagram/package shown in Figure 17 contains the basic interfaces for defining access control permissions to resources, as well as mechanisms for assigning qualifications/attributes to entities. These interfaces must be configured to suit the application/use case needs. For example, different types of qualifications may exist within a single TES implementation, each of them applicable to different set of agents, participants or external service providers.

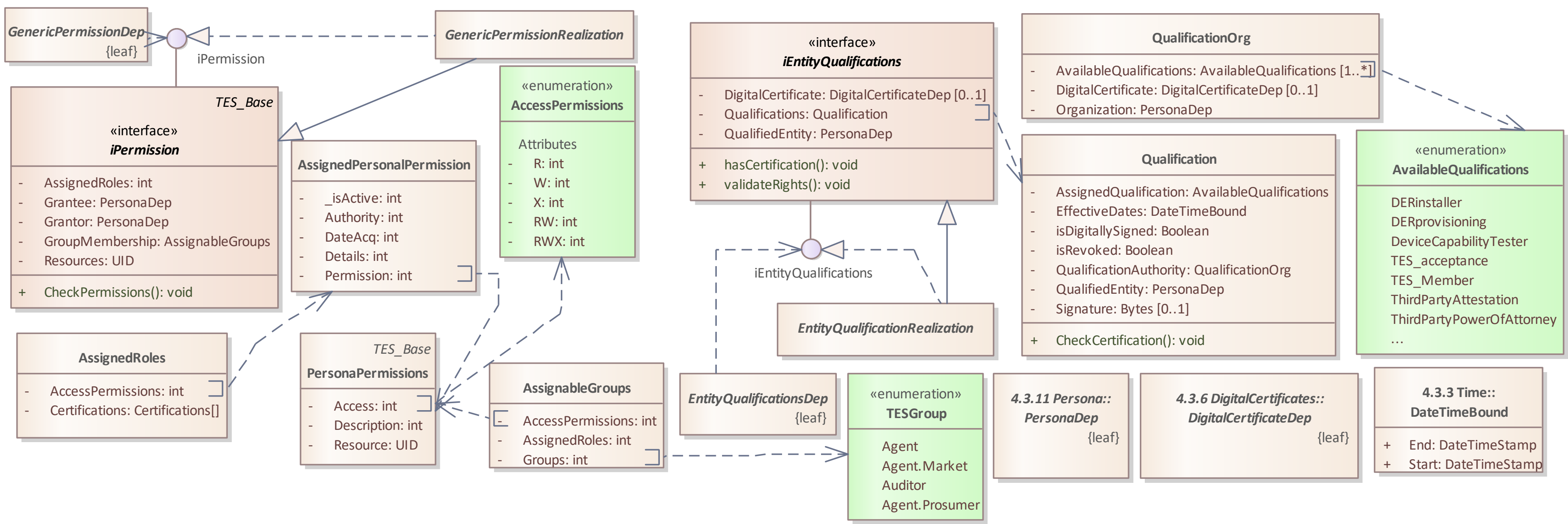


Figure 17. Overview of the *Permissions*' package components.

4.3.10 Grid object models

The diagram/package shown in Figure 18 provides the basic constructs used to model electrical systems. This includes the ability to store matrix data (for impedance), represent complex power and its direction. The contained classes represent only a subset of electrical-related objects and must be updated depending on the TES' application requirements. These base objects leverage the definitions found on IEC 61968-9 and IEEE 2030.5, potentially enabling interoperability with existent technological deployments based on these standards.

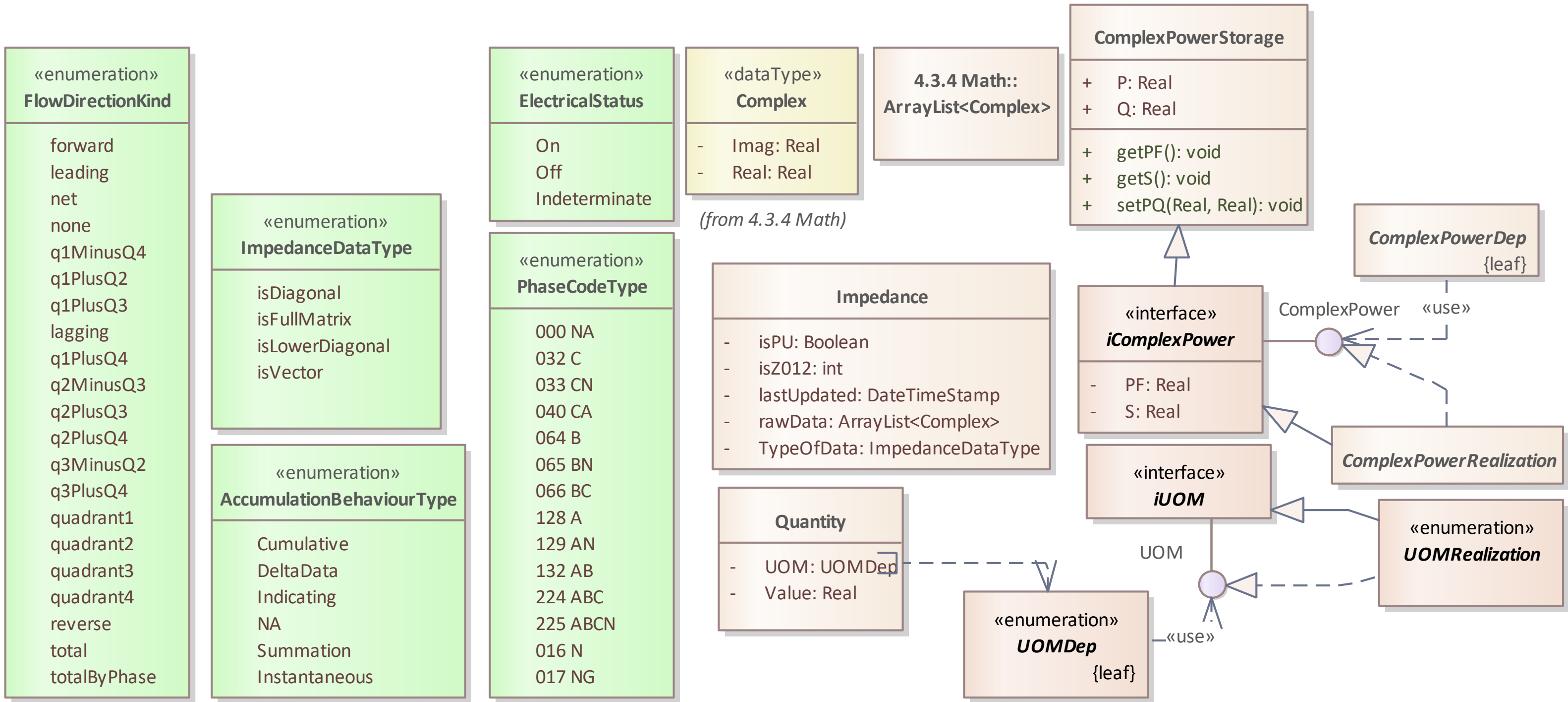


Figure 18. Overview of the *GridObjects* s package components.

4.3.11 Persona object models

The diagram/package shown in Figure 19 groups an assortment of classes that can be used to capture an entity's personal information. In addition, digital certificates and other locational information can be used to support advanced identity services. These identity interfaces can be referenced by other higher-level models to specify participants and provided a trusted-operational platform.

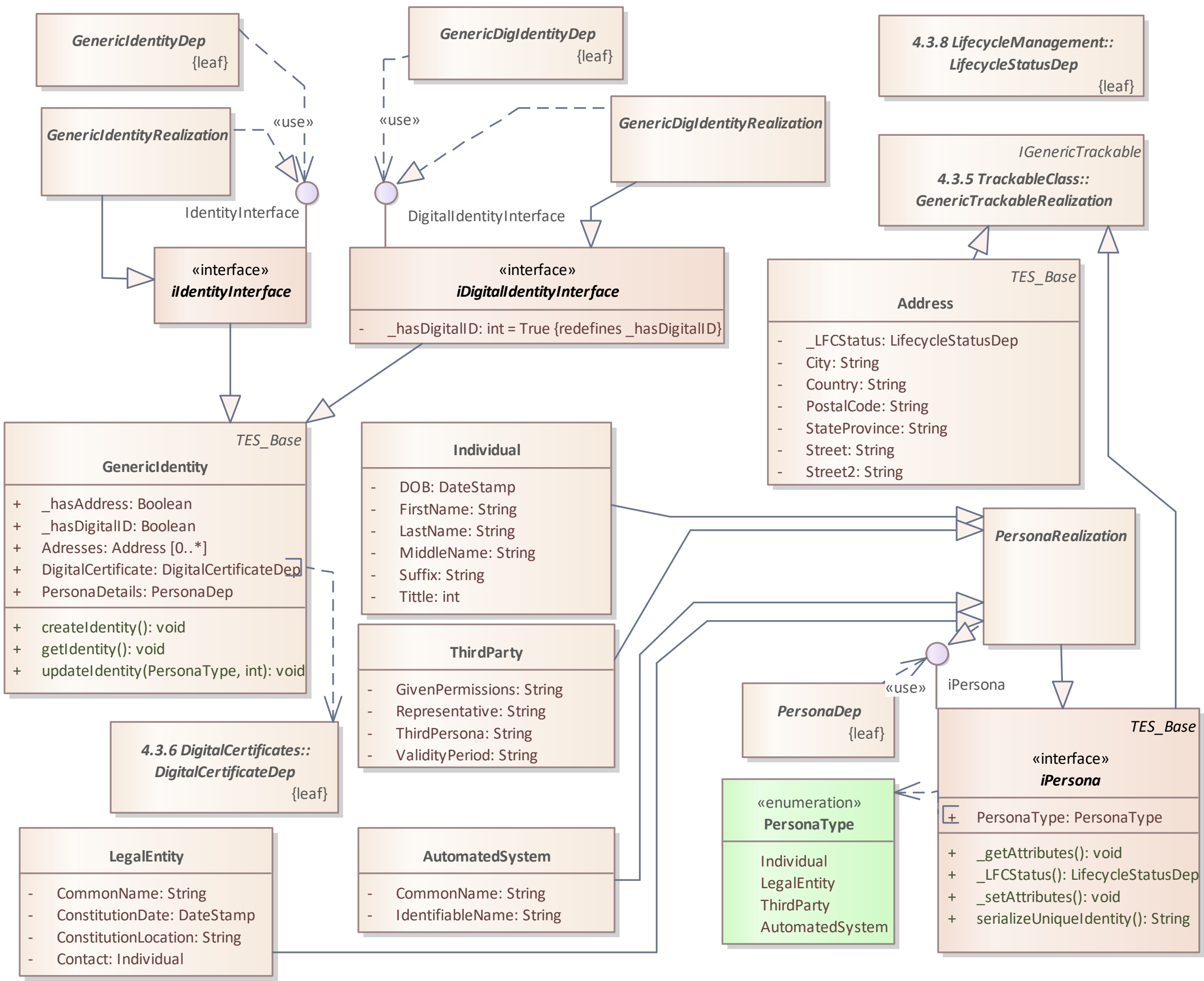


Figure 19. Overview of the *Persona*'s package components.

4.3.12 Memberships

The classes presented in Figure 20 can be used to establish memberships among two different systems. It is assumed that memberships requests are negotiated internally in between parties. The process assumes that a request-approval process occurs in between a solicitor and a target system, the target agent is responsible for evaluating the impacts/consequences of the relationship.

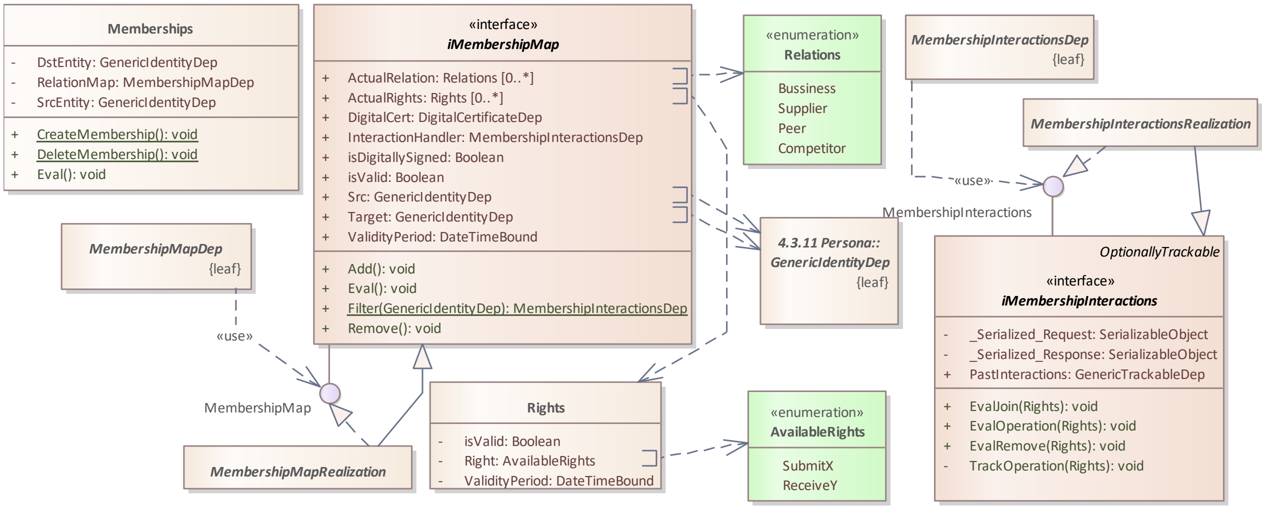


Figure 20. Overview of the *Membership*'s package components.

4.3.13 Summary of basic interfaces

The diagram/package shown in Figure 21-Figure 24 summarizes the basic interfaces provided by this report, the interfaces are generic and can be used to support most of the data and communication needs of grid applications (and specifically TES systems). The provided interfaces remain at the high-level, but still capture most of the common requirements and functionalities that will require different modules to communicate, creating a highly-interoperable network.

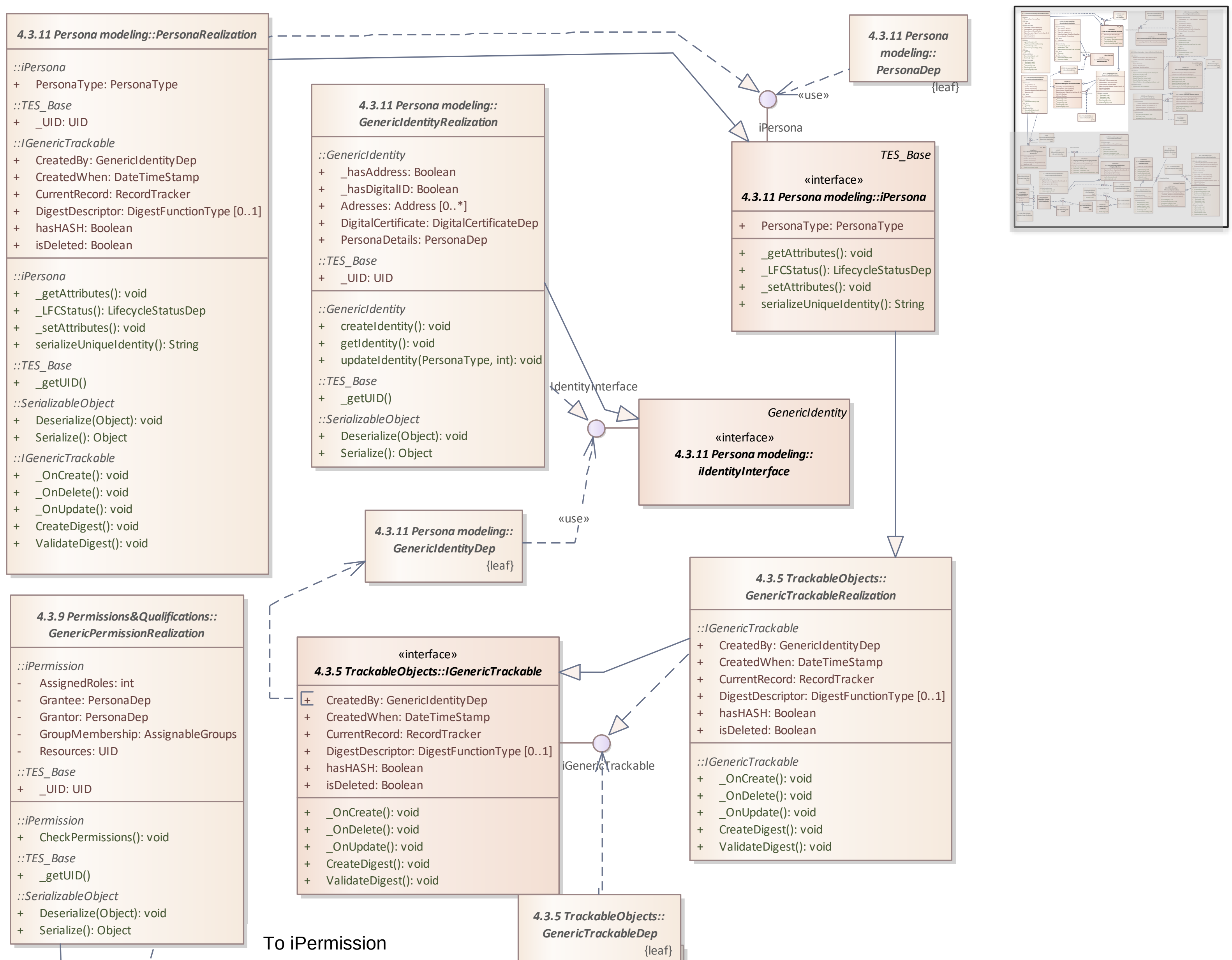


Figure 21. Overview of the *BasicInterface*'s package components (Top-left view).

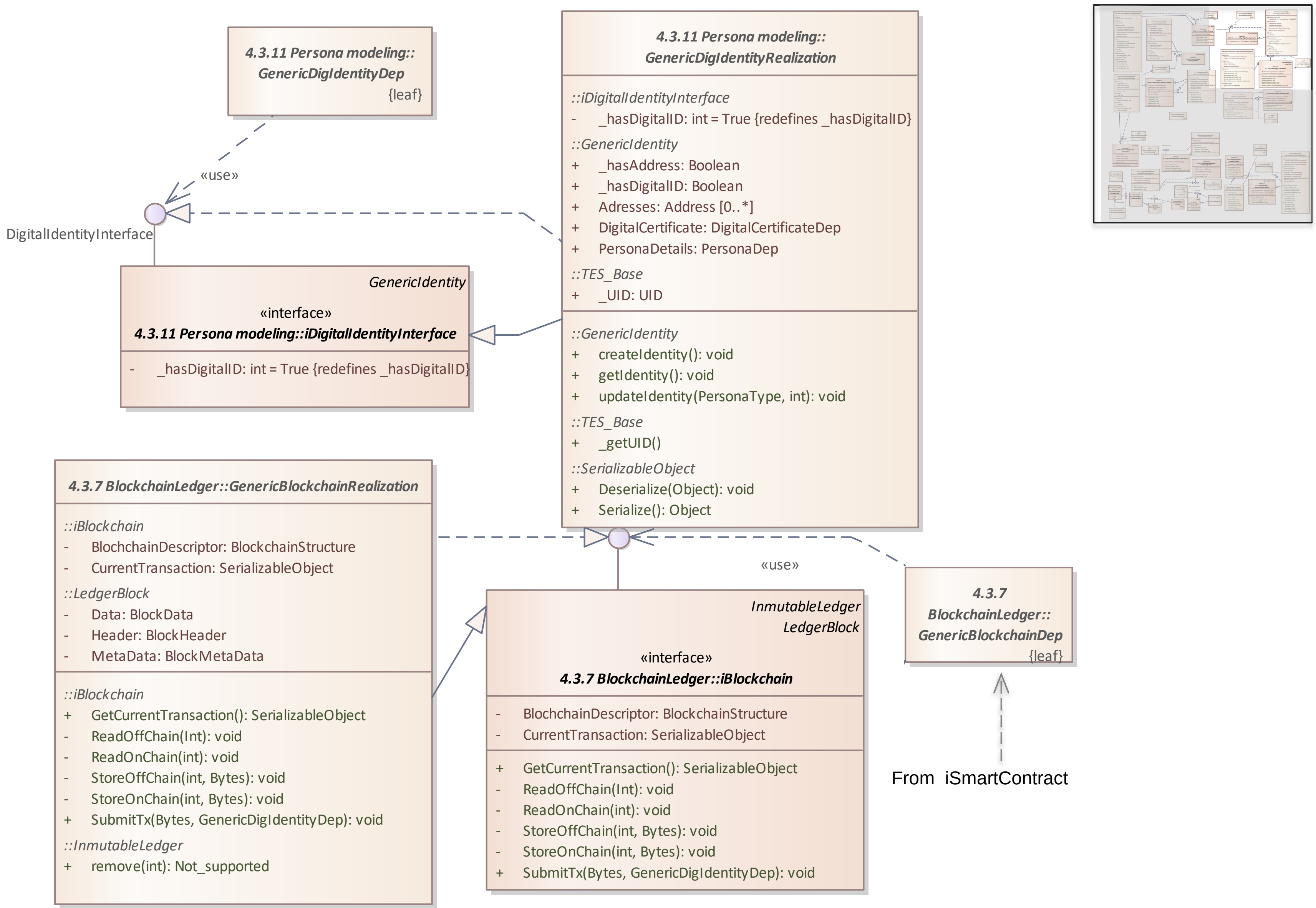


Figure 22. Overview of the *BasicInterface*'s package components (Top-right view).

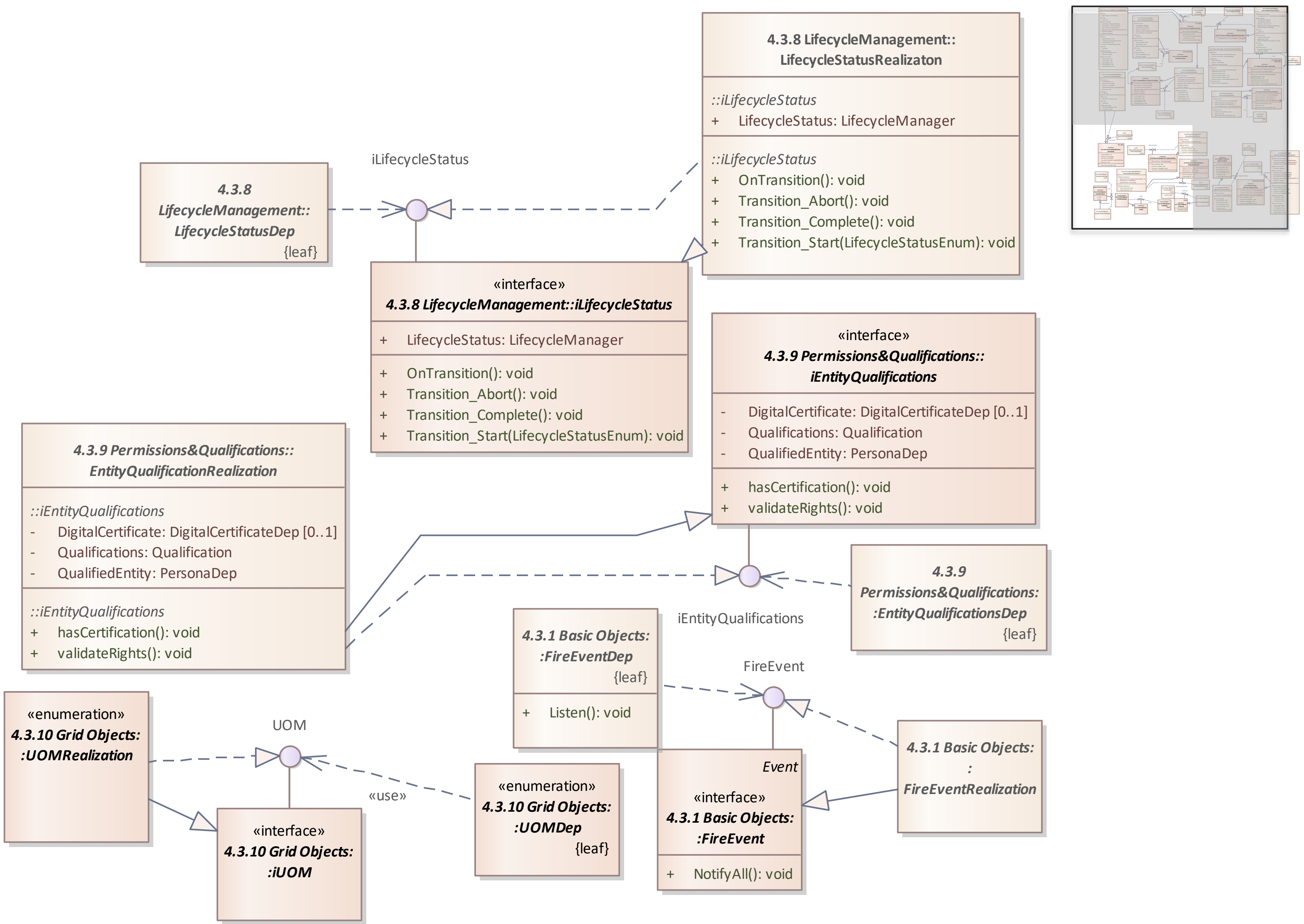


Figure 23. Overview of the *BasicInterface*'s package components (Bottom-left view).

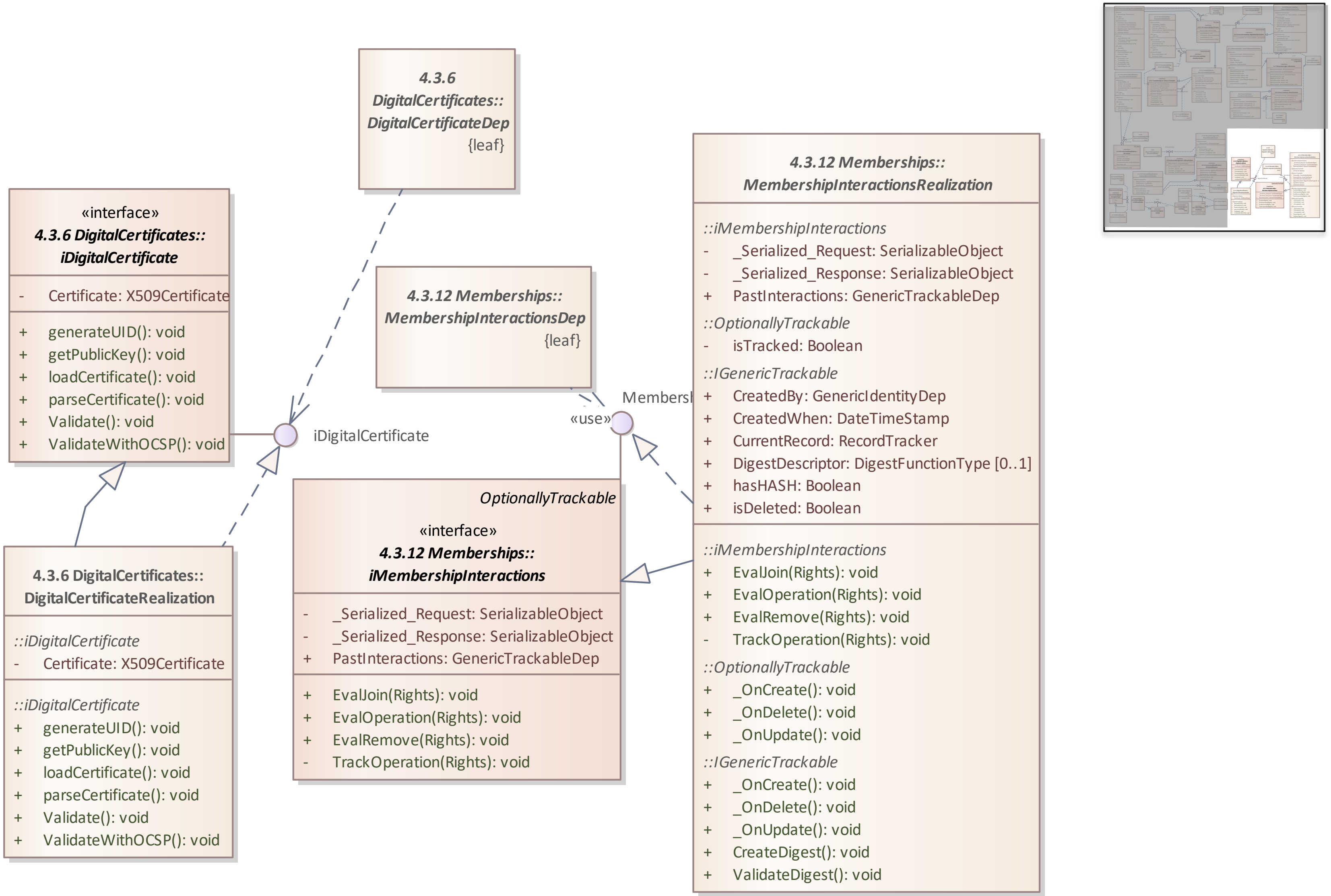


Figure 24. Overview of the *BasicInterface*'s package components (Bottom-right view).

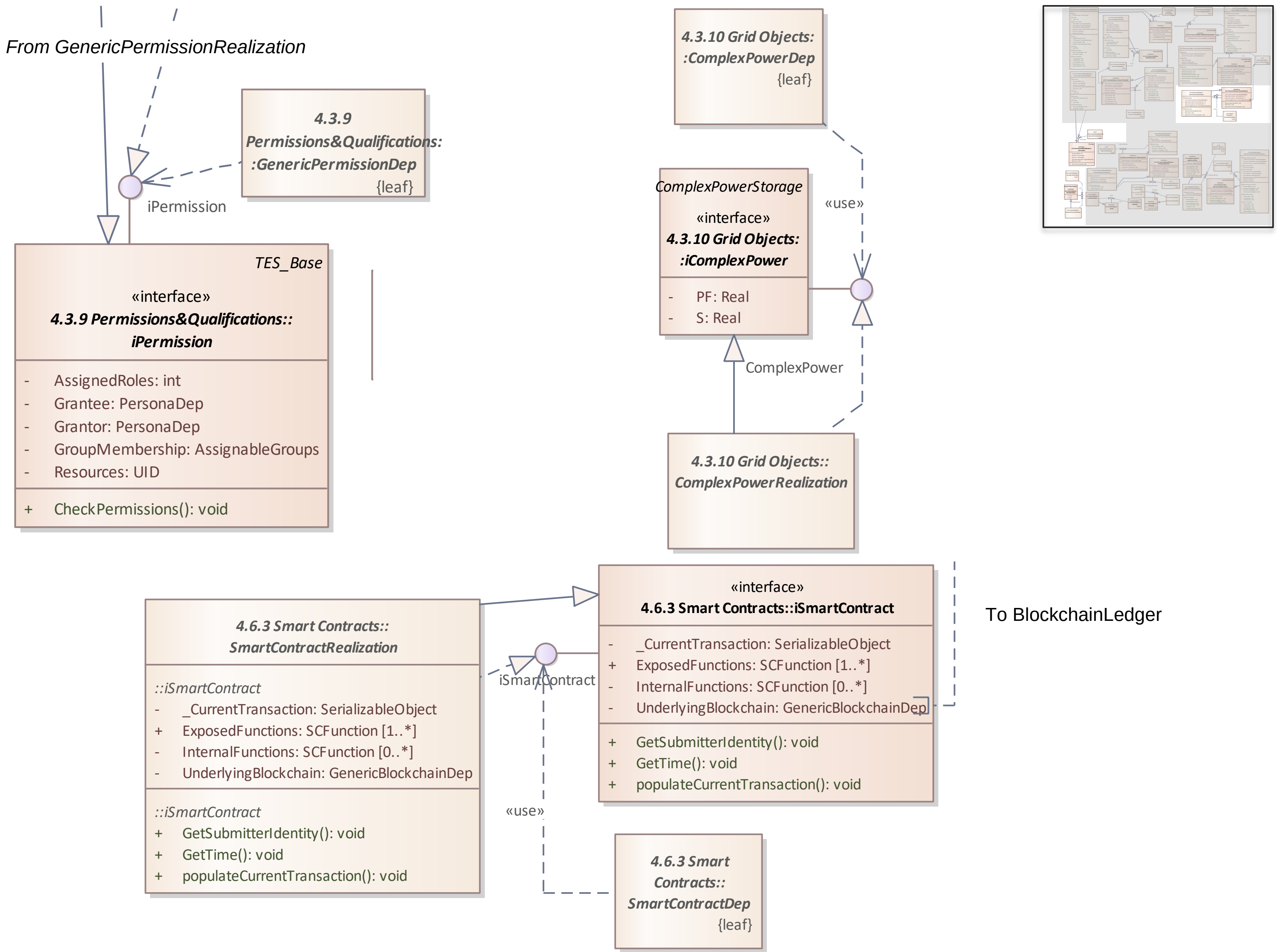


Figure 25. Overview of the *BasicInterface*'s package components (Auxiliary view).

4.4 Resources and Participants modeling

In this section an overview of templates that can be used to model a large variety of grid devices and participants is introduced. These resources are the main building block of any TES-based application, and represent the main contribution of the presented work, similar to the previous section, these resources are explored in-depth within the annex.

4.4.1 Resources

The diagram/package shown in Figure 26 documents a variety of classes that work together to represent grid equipment and expose it to a transactive system. The provided interfaces enable to abstract the different levels of interactions that are expected to occur within a TES. The proposed models have the ability to account for active as well as "dumb" devices, exposing only capable equipment as a grid resource. From this point forward, grid resources can be controlled and managed by dedicated transactive agents.

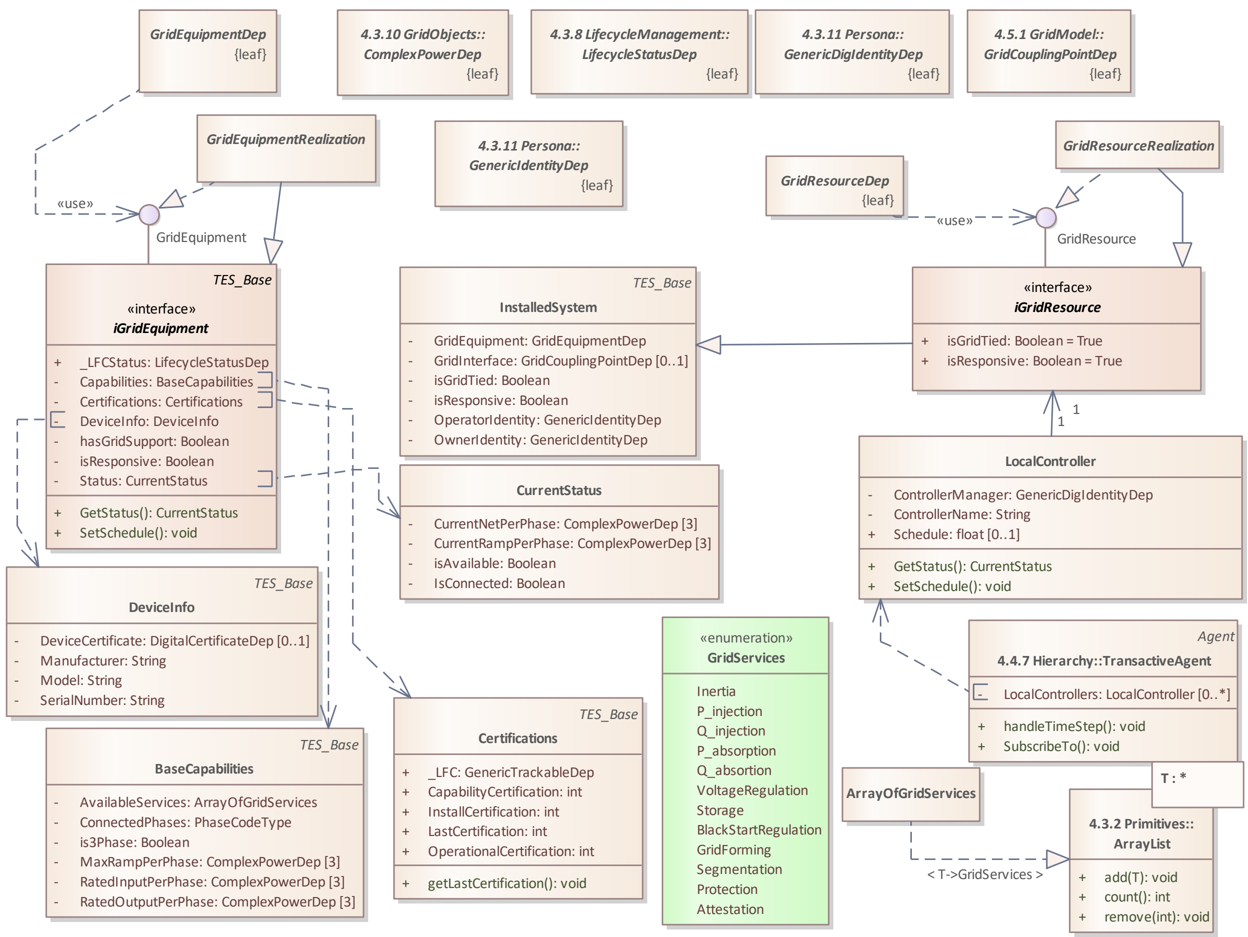


Figure 26. Overview of the *Resources*' package components.

4.4.2 Load resources

The diagram/package shown in Figure 27 is a specialization of a grid resource, it illustrates the two main types of loads that are present on the grid. Both load models present a conformant interface to the *GridEquipment* requirements.

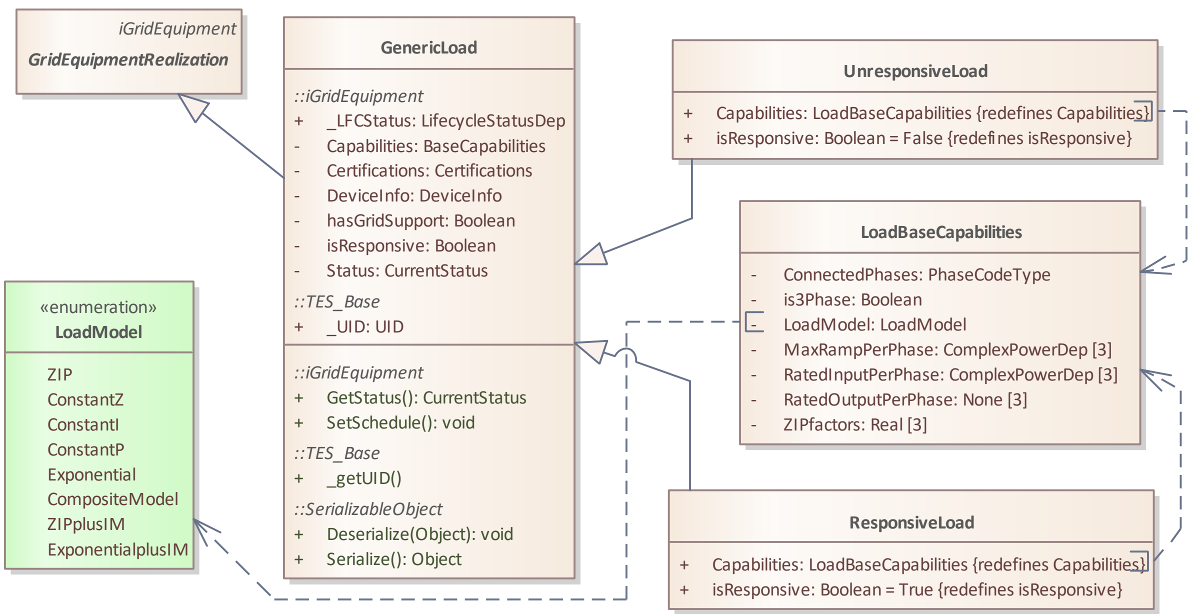


Figure 27. Overview of the *LoadResources*' package components.

4.4.3 IBR-Based Generation Resources

The diagram/package shown in Figure 28 is a specialization of a grid resource, it enables end users to model the features of an Inverter-Based generator. It documents specific examples to model PV-based and wind-based resources which can further refined to satisfy the data capturing needs of the end use.

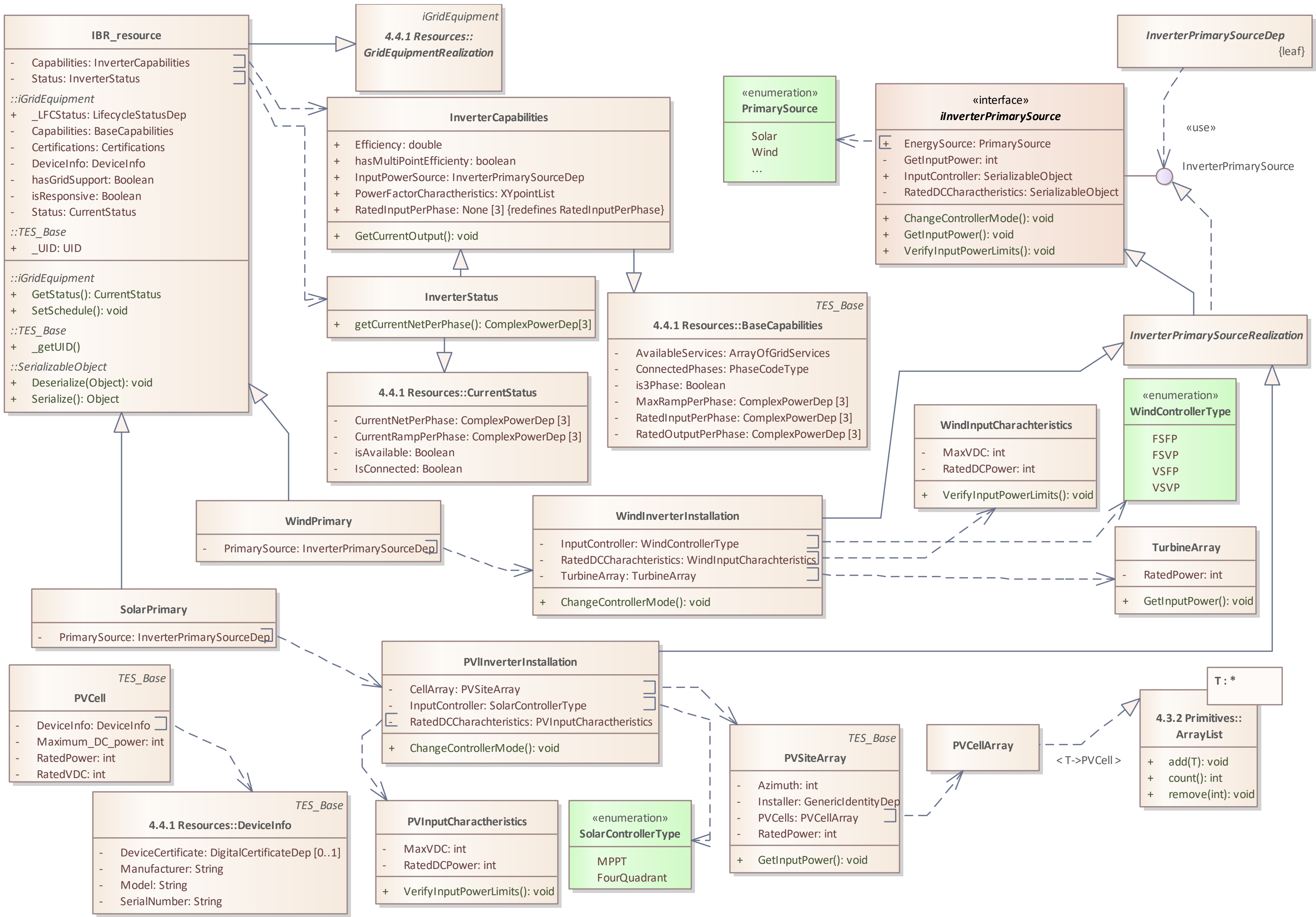


Figure 28. Overview of the *IBR-BasedGeneratorResources*' package components.

4.4.4 Rotational Generation Resources

The diagram/package shown in Figure 29 is a specialization of a grid resource, it enables end users to capture the components of a traditional power plant. The interface can be used to provide (and extract) data from both the electromechanical energy conversion process, as well as the mechanism used to capture the mechanical energy. By including the most common energy generation processes (DER, Bulk, and storage) the provided templates can be applicable a wide variety of application scenarios. Potentially enable the participation of a wide variety of systems.

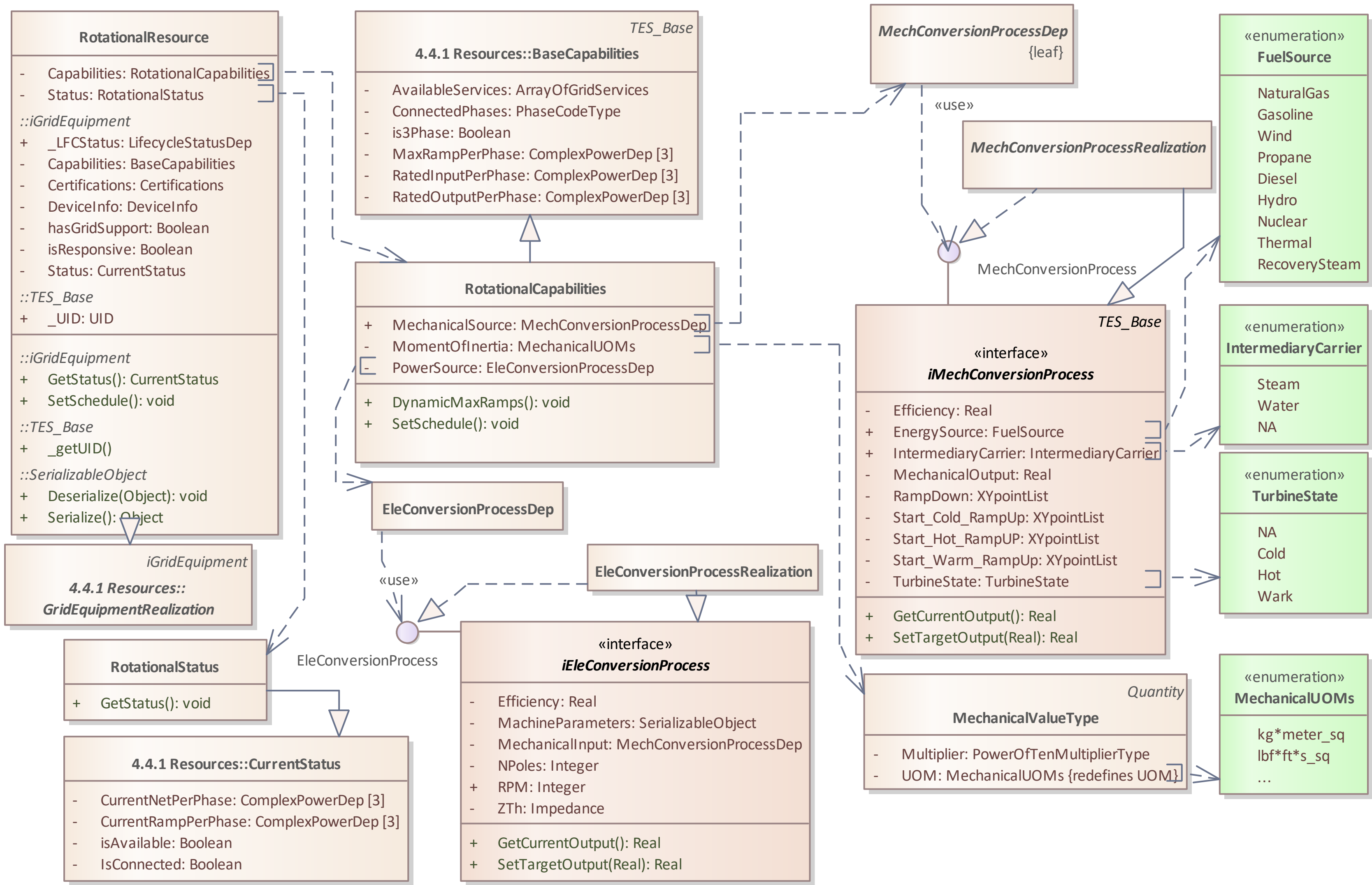


Figure 29. Overview of the *RotationalGenerationResources*' package components.

4.4.5 Storage Resources

The diagram/package shown in Figure 30 presents an overview of a storage-based resource. It expands on the interfaces presented earlier and divides the charging/delivery process into two dedicated systems, which can be specified independently.

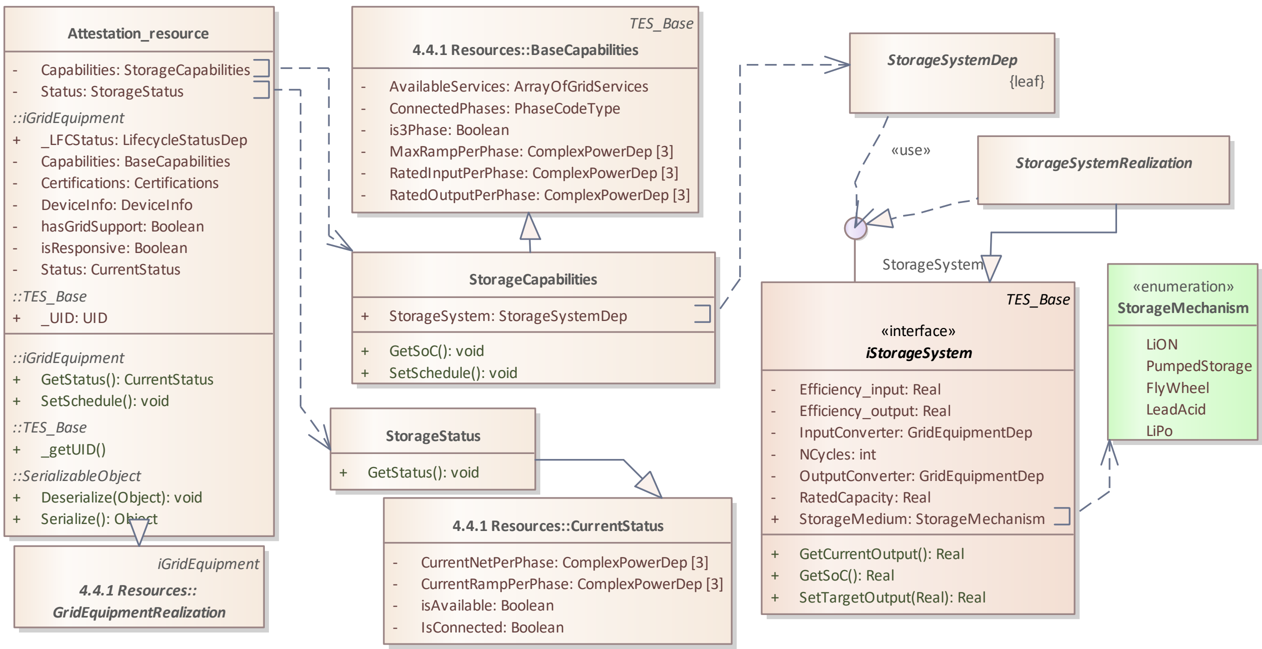


Figure 30. Overview of the *StorageResource*'s package components.

4.4.6 Attestation Resources

The diagram/package shown in Figure 31 presents an overview of an attestation-capable resource. In this case, it is assumed that an attestation device only exists on the digital domain (although signal sampling can occur on the physical side). The proposed model ties the attestation device to another's device sampling/measurement interface and can choose to digitally sign/protect data if desired.

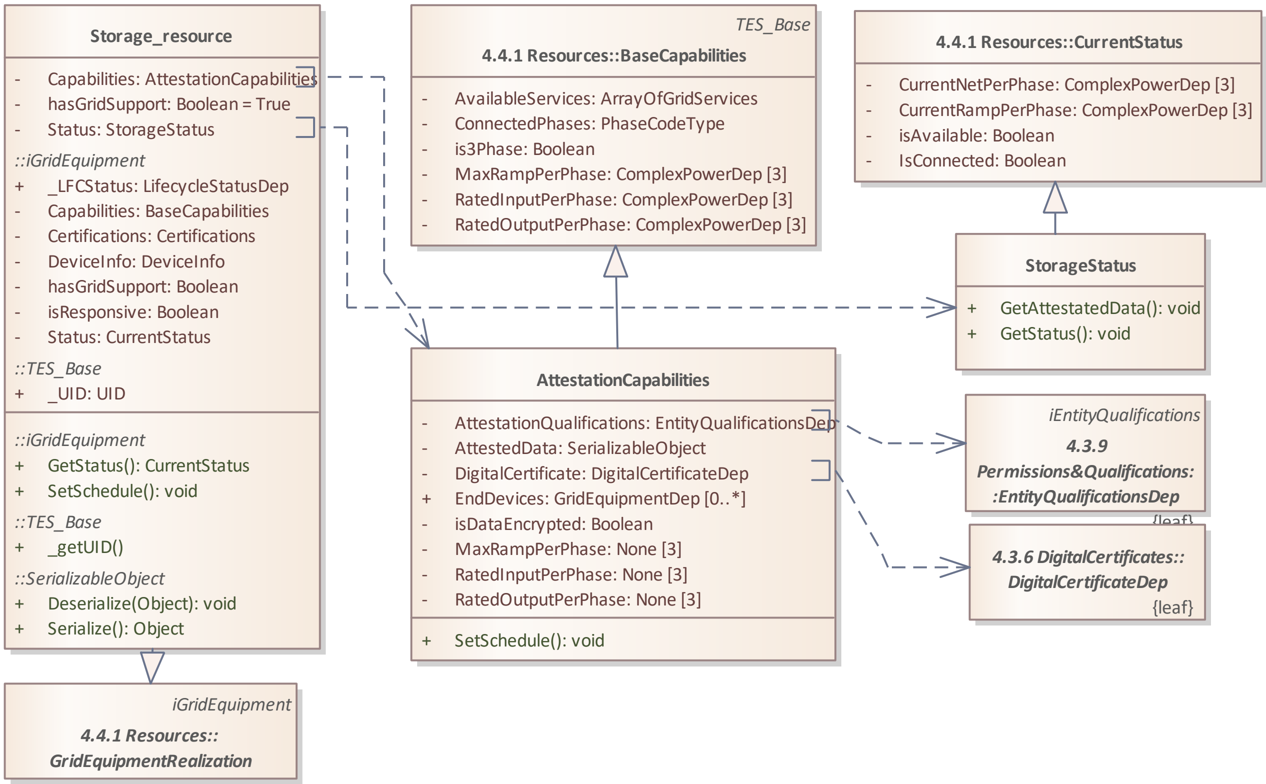


Figure 31. Overview of the *AttestationResources*' package components.

4.4.7 Organizational Hierarchy

The diagram/package shown in Figure 32represents a reference hierarchy that can be used to map the different types of actors/systems that may be present on a typical TES system where a wide variety of participants may interact. This diagram is only intended to be illustrative and can be adjusted to suit the application needs.

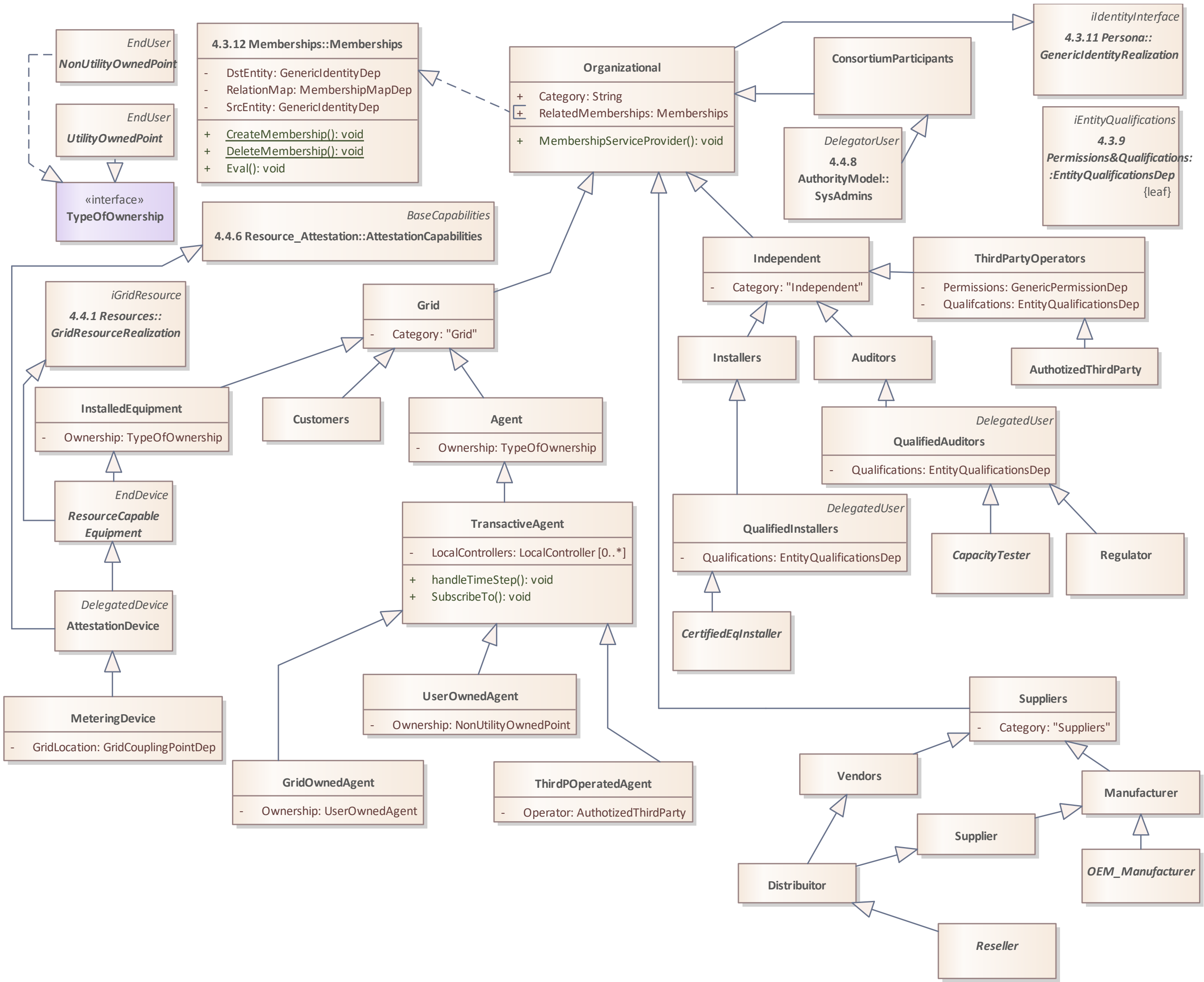


Figure 32. Overview of the *OrganizationalHierarchy*'s package components.

4.4.8 Authority Model

The diagram/package shown in Figure 33 introduces a group of classes that represent the subset of participants that have administrator-like rights over other participants. These participants can manage other participants (such as dictating a role or permissions) as well as defining processes and setting rules.

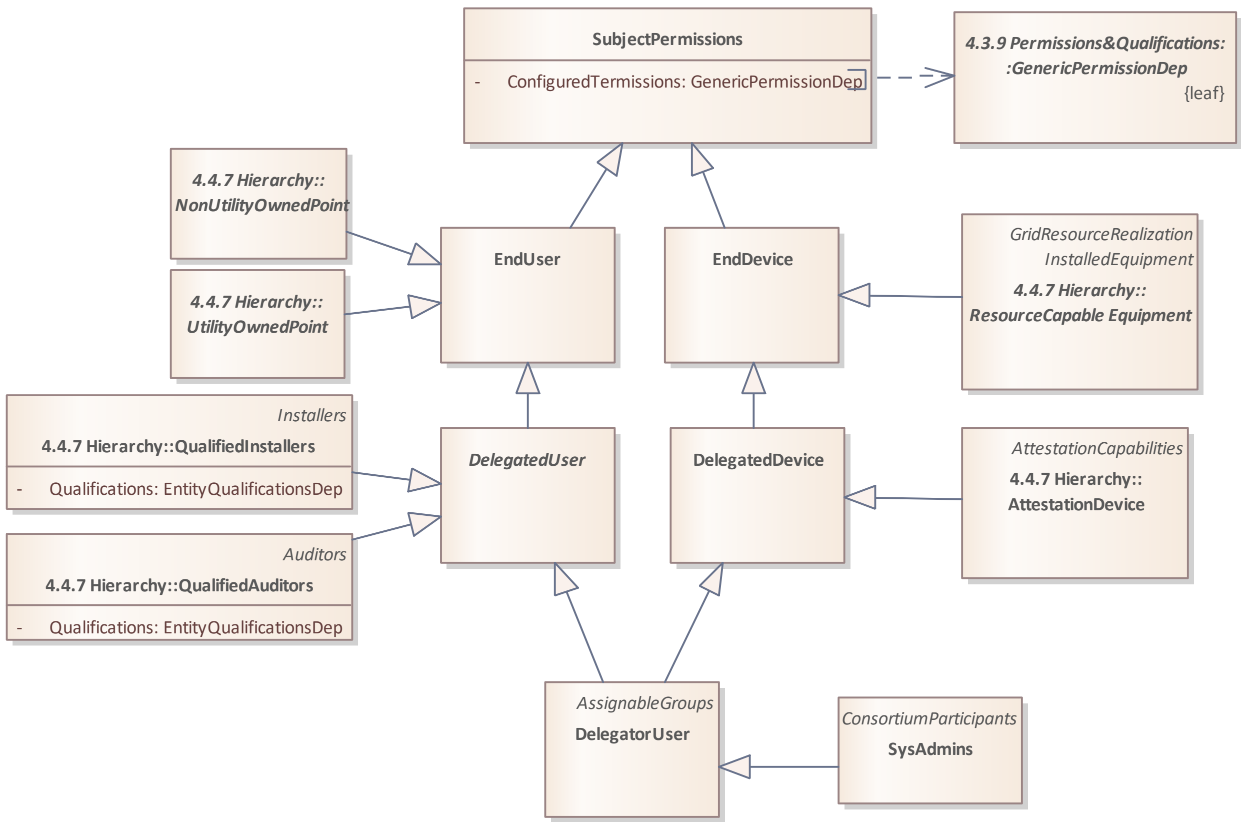


Figure 33. Overview of the *AuthorityModel*'s package components.

4.4.9 Sample Hierarchy with associated actors

The diagram/package shown in Figure 34 represents a reference hierarchy that can be used to map the different types of actors/systems that may be present on a typical TES system where a wide variety of participants may interact. This diagram is only intended to be illustrative and can be adjusted to suit the application needs. The diagram has been populated with actors that will be used to demonstrate potential use cases.

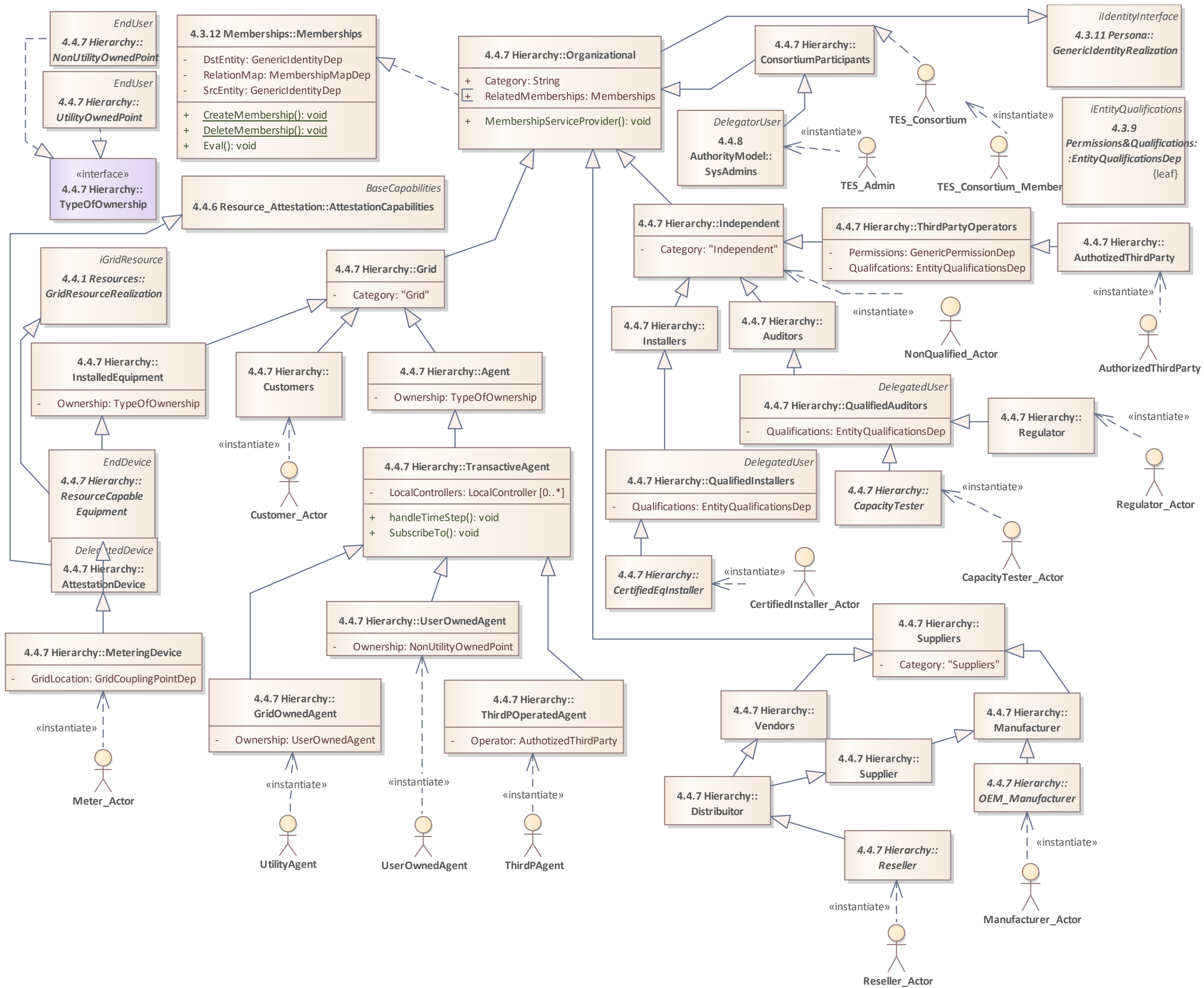


Figure 34. Overview of the *SampleHierarchyWithActors*'s package components.

4.5 Grid components

This section presents a grid modeling proposal that aims to retain the electrical topological hierarchy of power systems while at the same time enabling grid support services to attach to virtual grid points. This grid-resource modeling is expected to enable an efficient mapping in between a traditional grid operation and a TES-enabled one.

4.5.1 Grid Model

The diagram/package shown in Figure 35 provides a reference architecture for modeling grid connectivity on a relational database format. The proposed design exposes a grid coupling interface that serves as a bridge to other TES components. The classes used to represent this grid model were adapted from IEEE 2030.5 and the Common Smart Inverter Profile V2.0 (IEEE Standard for Smart Energy Profile Application Protocol 2018). By leveraging these standards, it is expected that grid participants can seamlessly integrate the proposed template architecture with existing applications, such as Advanced Distribution Management Systems (ADMS) or via DER aggregation services. It is expected, that as the template continues to mature, new (and existent) objects will seek to become more standardized.

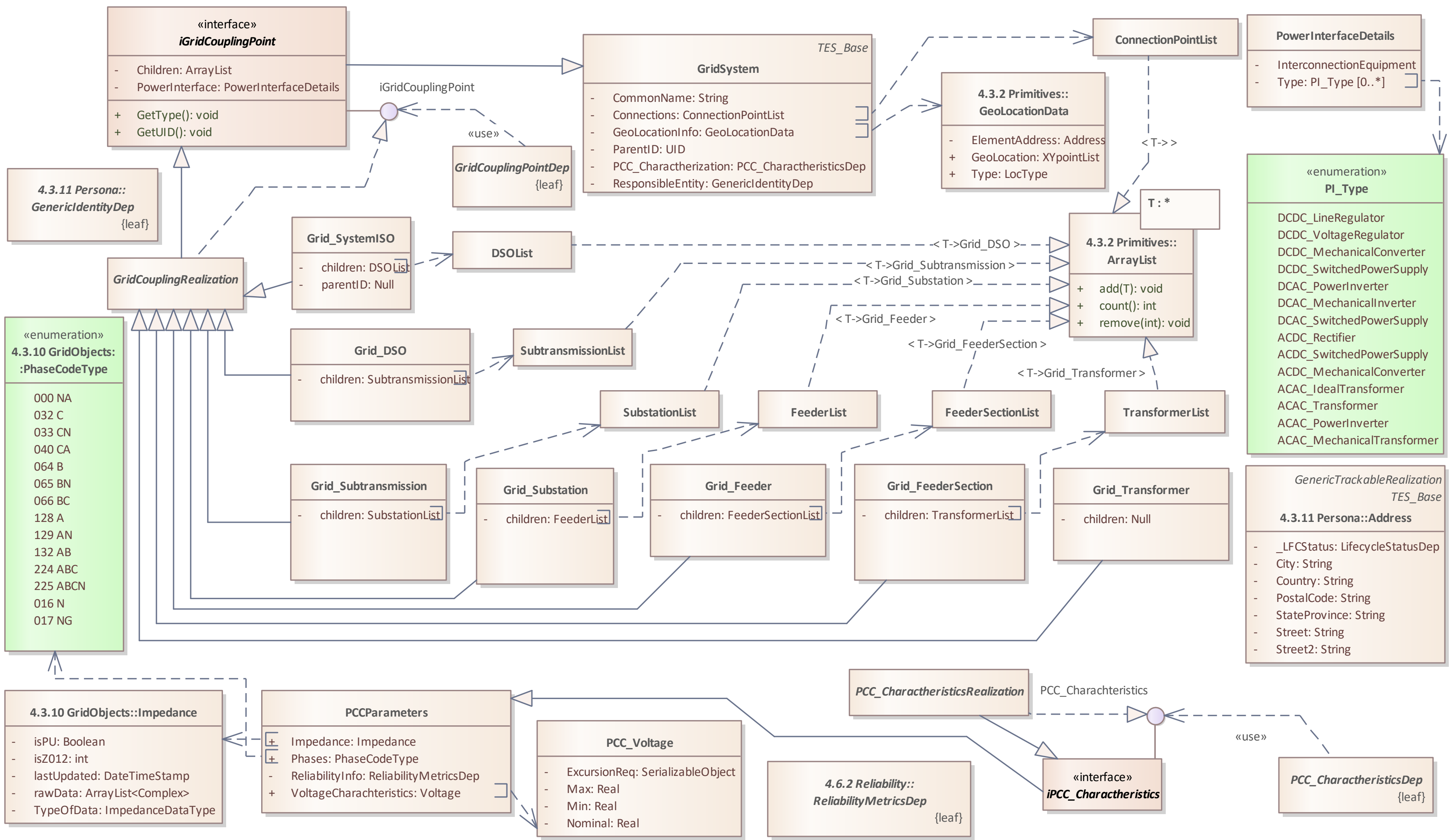


Figure 35. Overview of the *GridModel*'s package components.

4.6 Smart Contract Modeling and Support Services

In this section a series of auxiliary grid monitoring services will be introduced (from a modeling perspective). These services are expected to assist with the measurement, verification, and eventual settlement of TES-based transactions. Following the approach given by the previous sections, the presented diagrams remain at the high-level, the reader is encouraged to consult more details within the annex section of this document.

4.6.1 Measurement and Verification

The diagram/package shown in Figure 36 contains a variety of data models that can be used to record a variety of commodities, quantities, using a wide variety of data aggregation methods. Most of the data models introduced by this section are based on the models contained in IEEE 2030.5

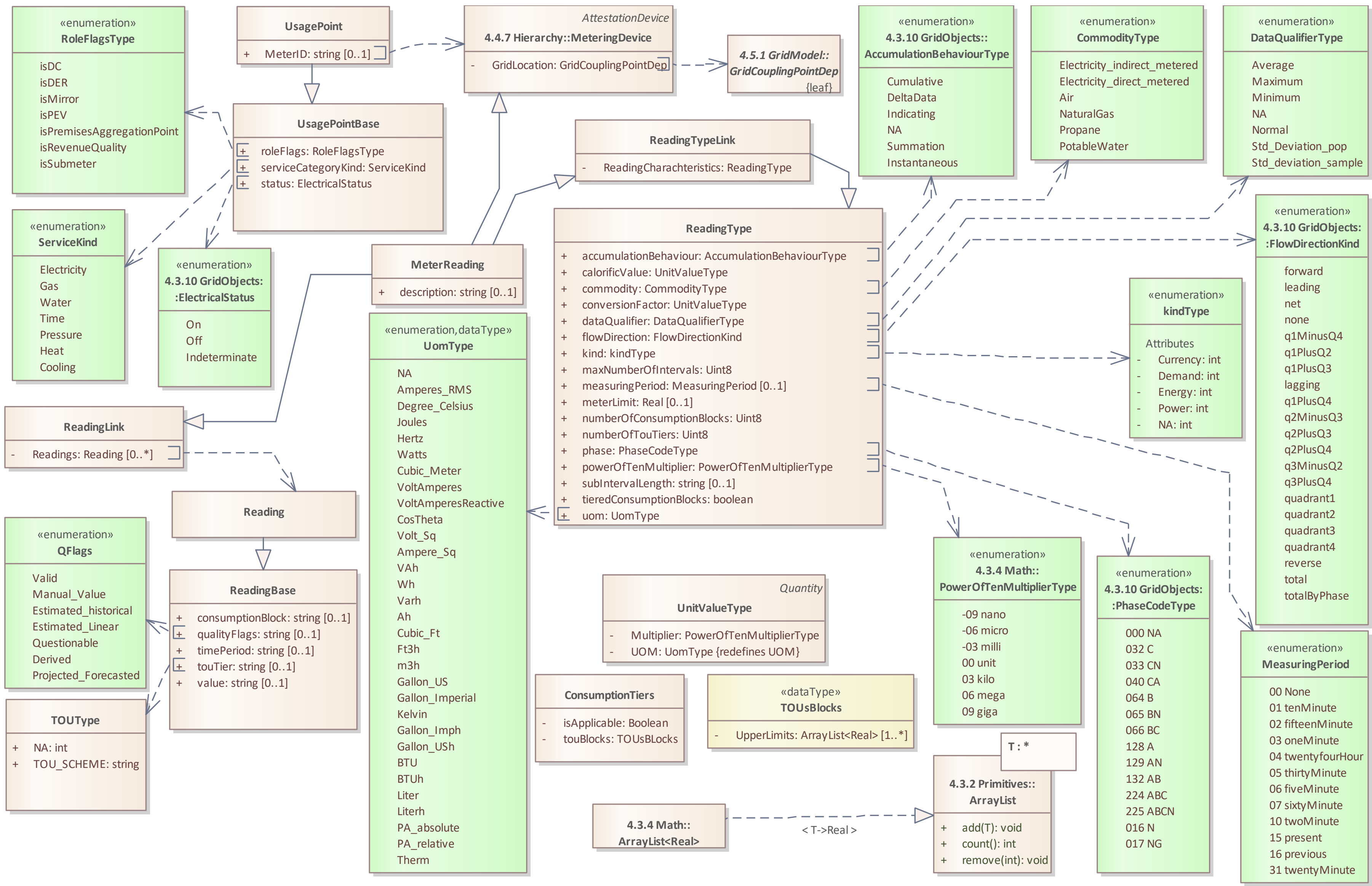


Figure 36. Overview of the *MV*'s package components.

4.6.2 Reliability

The diagram/package shown in Figure 37 provides a reference implementation of a data interface that can be used to capture grid reliability data. This interface can be leveraged to provide additional details to market and monitoring applications that run on top of the TES stack.

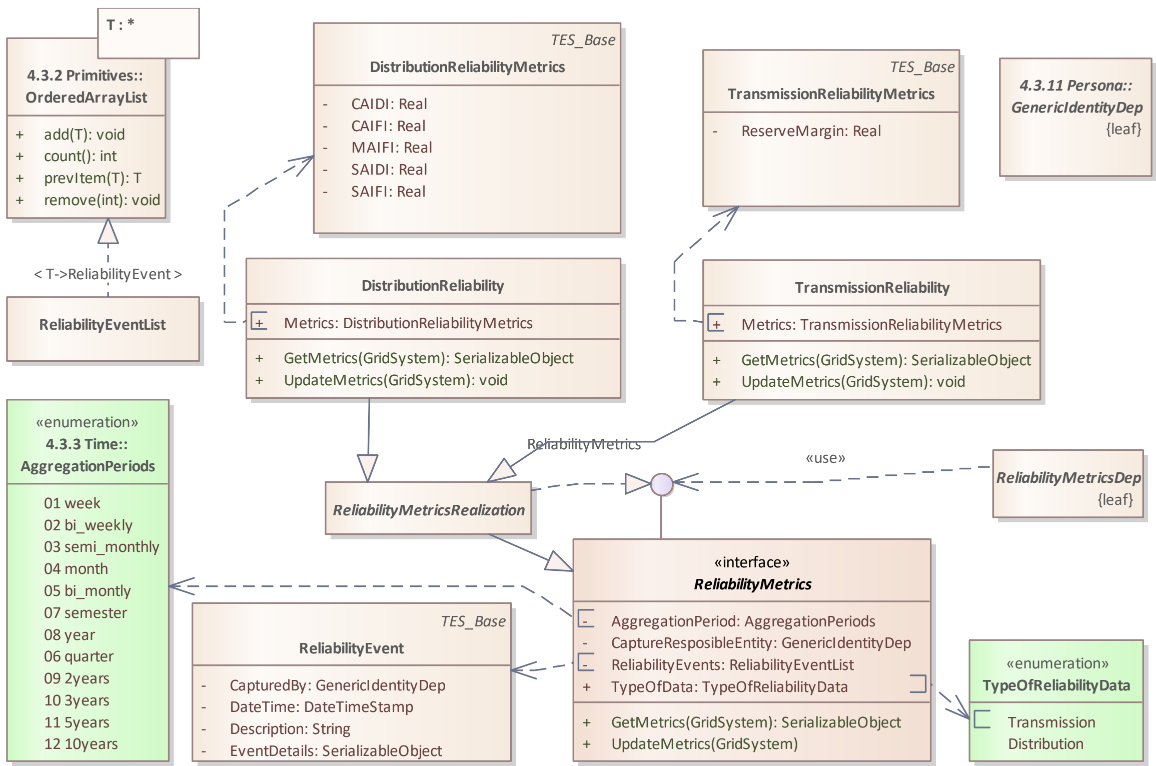


Figure 37. Overview of the *Reliability*'s package components.

4.6.3 Smart Contracts

The diagram/package shown in Figure 38 presents an overview of the components found within a smart contract. Most of the information of this model is abstract, and its functionality must be defined by the underlying blockchain and unique application requirements.

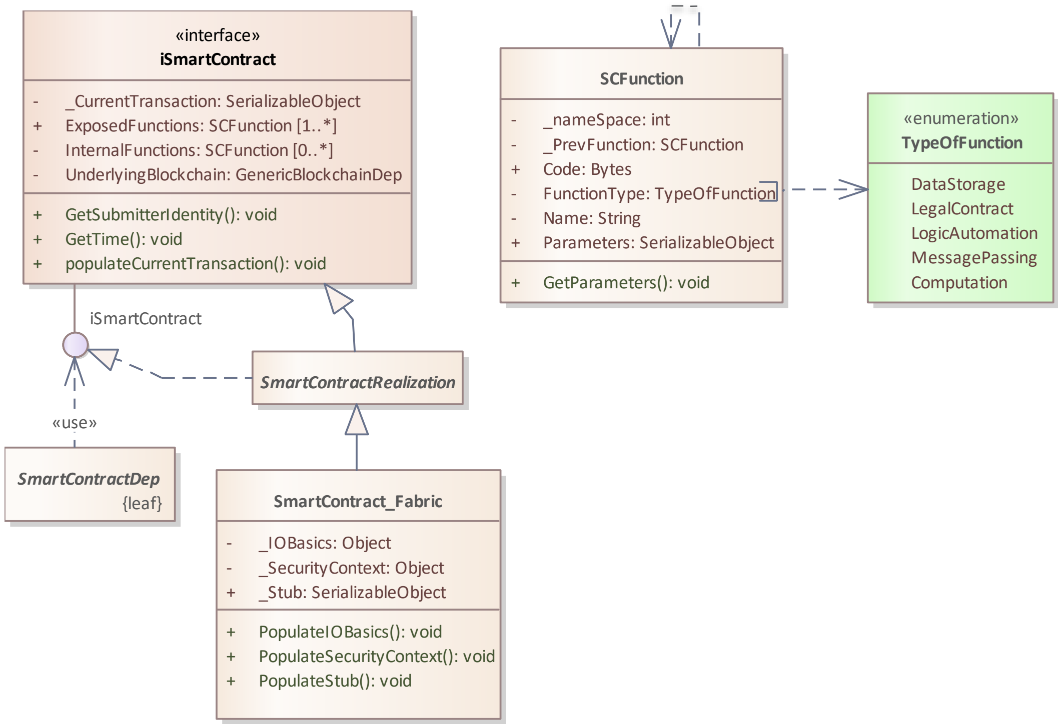


Figure 38. Overview of the *SmartContracts*' package components.

4.7 Operations-Structural components

In this section, a sample set of structural components that may be relevant to a TES five-stage operational model are presented. These structural components are intended to serve as a reference and application developers will need to build their processes based on their needs and templates introduced in the previous sections.

4.7.1 Qualification & Registration

Qualification diagram

The diagram/package shown in Figure 39 demonstrates the ability of the proposed template to enable device-level qualifications assignments. This is done by creating a mapping in between: 1) A qualification instance that holds the qualification attributes, 2) A data model that references an installed system object, and 3) An entity that can certify the physical capabilities of such system.

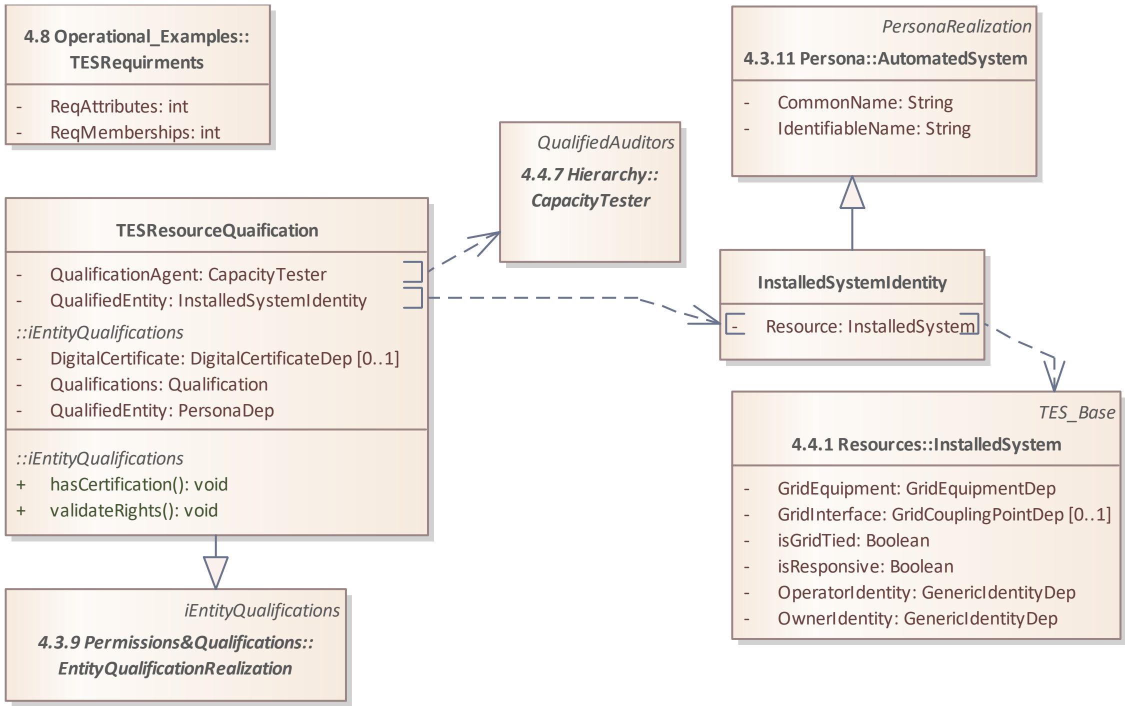


Figure 39. Overview of the *Qualification*'s package components.

4.8 Operations-Examples

This section contains examples that may serve as a reference for building more complex systems. This examples only list the main steps and will need to be adapted to suit an application's needs.

4.8.1 Agent qualification

In this demo we assume that a non-qualified actor is interested in becoming a qualified DER installer. To achieve this state the actor must first get a copy of the terms and conditions (requirements), followed by getting all the documentation ready. Finally, the agent submits this documentation (e.g., proof of courses taken) and its case gets evaluated in a transparent, equitable manner by the blockchain-based solution.

4.8.1.1 Agent Qualification Demo-Structural side

This diagram (Figure 40) presents the data dependencies needed to transition a non-qualifier actor into a qualified actor. For example, a company may want to gain qualifications as a DER-system capacity tester.

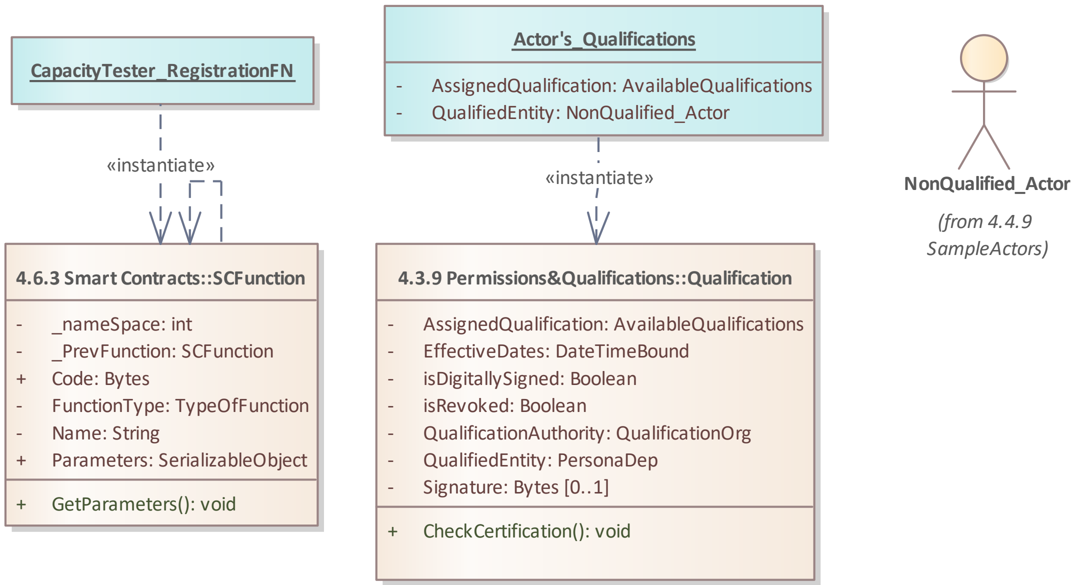


Figure 40. Overview of the *QualificationUseCase*'s package components.

4.8.1.2 Agent Qualification Demo-Behavioral side

This diagram (Figure 41) presents a sequence diagram for transitioning a non-qualifier actor into a qualified actor. It is assumed that a consortium has already decided on the terms and conditions and the registration process for becoming a capacity tester has been outlined.

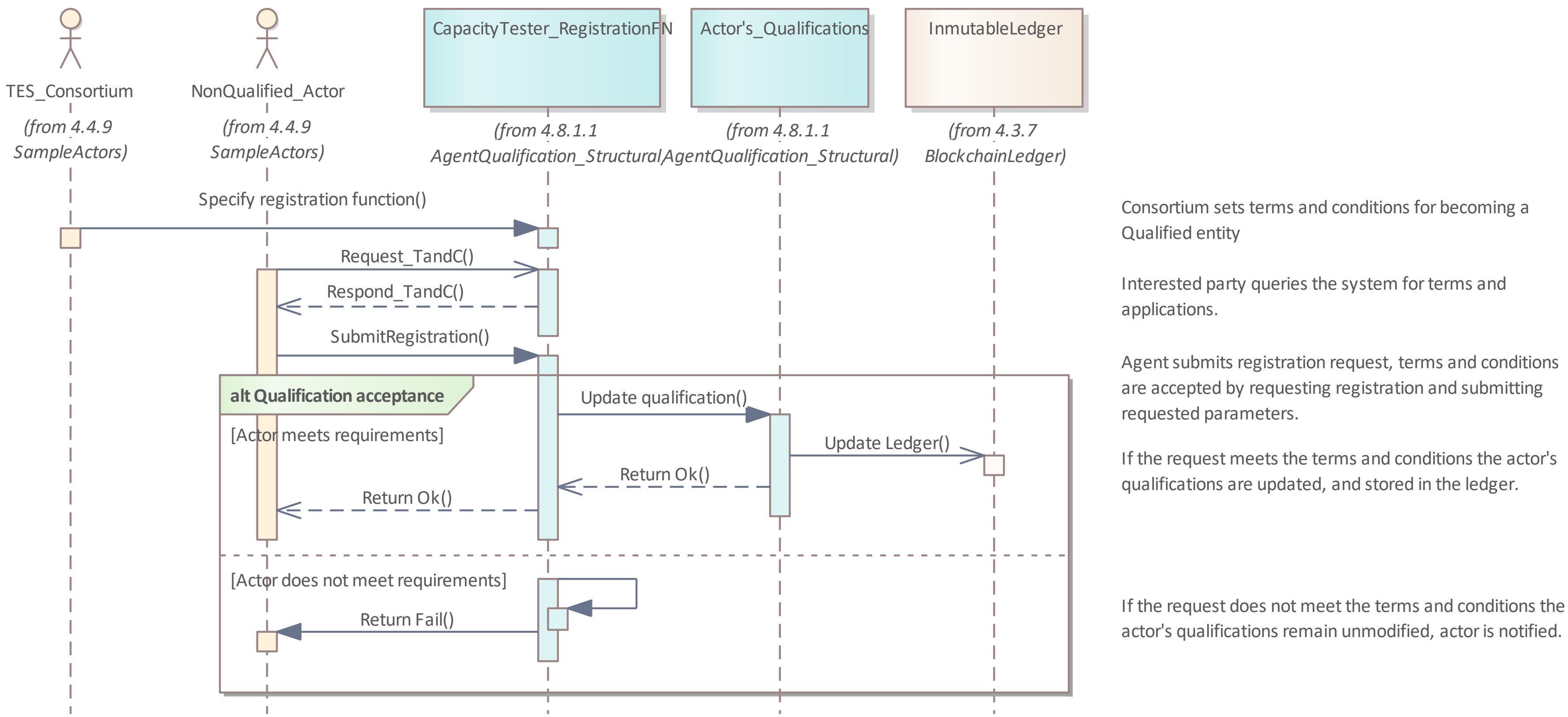


Figure 41. Overview of the 4.8.1 *Registration&Qualification's* package components.

4.9 Sample: Developing a Smart Contract-Based Permission Solution

TES-based solutions need to ensure that an agent's private and competitive information remains hidden from other competing participants. Ideally, this information should only be shared with entities that have a valid need-to-know business requirement, which may include a system operator or a third-party agent who can certify an agent's capabilities. Other types of information may have a more temporal need for privacy; for example, bids must remain sealed at least until the market agent clears them (although in practice it may be beneficial to still restrict access to competitors even after the bids get cleared to prevent behavioral analysis).

These privacy needs are often addressed by using access control mechanisms, which can limit access to information based on a variety of conditions. One of the most commonly used mechanisms is Role Based Access Control (RBAC). This method limits access to information based on a user's particular role in an organization. These roles and rules are typically assigned by an administrator and the rules are usually defined using Access Control Lists (ACLs). Roles define the access level that a group of users has for a particular resource. However, RBAC starts displaying problems when a user has multiple assigned roles within a system and fails to determine the relevant role that must be evaluated relative to the ACL.

A second approach is to use Attribute-Based Access Control (ABAC), a method for controlling access by relying on pre-configured policies to determine a resource's access permissions based on certain attributes. These attributes can represent a wide variety of user, resource, or system-level properties. Determining access with these extended attributes provides a much more flexible, dynamic, and well-defined access control mechanism that would otherwise be impossible with RBAC. However, ABAC has multiple components that must work together to ensure that access is adequately managed. A detailed overview of the ABAC components is given in Figure 42.

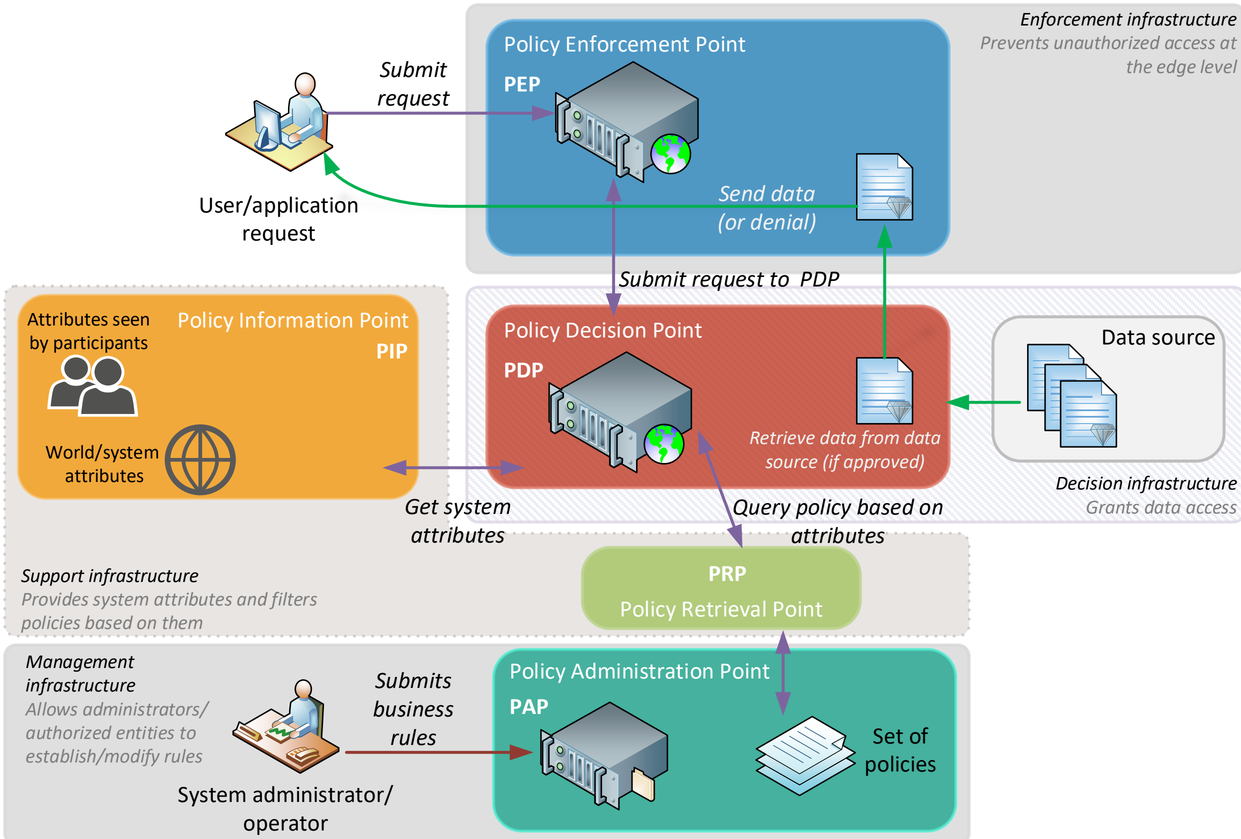


Figure 42. The components of an ABAC system.

4.9.1 TES execution model in Hyperledger fabric

This package/diagram represents an abstract representation of the base-class used to interface any object-oriented class with a Blockchain-based ledger, it contains all the bootstrap functions to streamline the creation, loading, updating of any object.

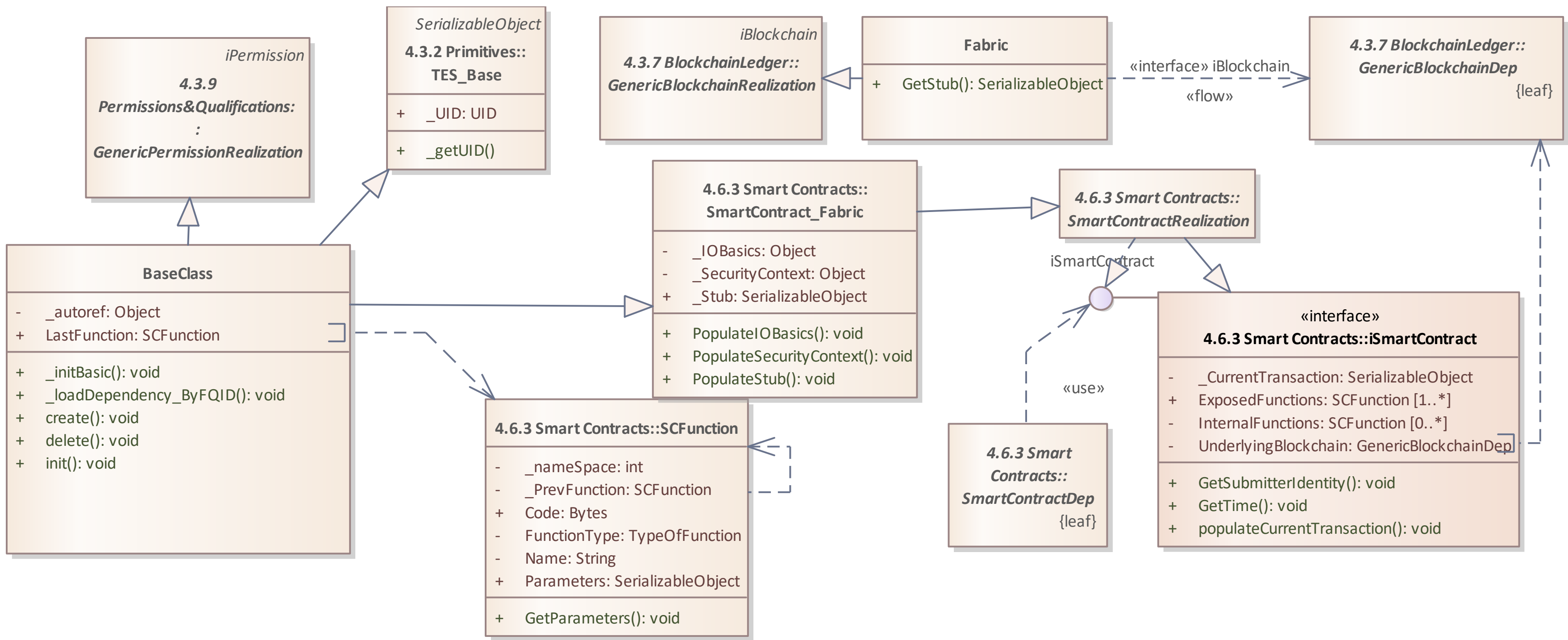


Figure 43. Overview of the *BaseTES*' package components.

4.9.2 ABAC Implementation on Blockchain

Following the architectural overview presented in section 4.9, the structural elements required to implement the Policy Decision Point (PDP), along with the Policy Retrieval Point (PRP) and the Policy Information Point (PIP) were modeled in UML. These components mostly rely on the ledger to serve as the PRP and the SC to serve as the PDP and PIP. It is assumed that a Policy Administration Point (PAP) can be implemented by the developer and presented to the system administrator in a user-friendly manner. It is important to note that the Policy Enforcement Point (PEP) is directly implemented by the PDP.

4.9.2.1 The Policy Resource Class

Based on the ABAC characteristics, it was determined that there is a many-to-many relationship between policies and resources (e.g., a resource can have multiple policies, and a policy can apply to multiple resources). To represent this relationship, an intermediary object—the *PolicyResource* class—was created. This class maintains references to a policy, a resource's name, and a subject's role. This class acts as an associative entity that resolves the many-to-many relationships between policies and resources. When a policy evaluation is requested, a logical function can search the ledger by either looking at *ResourceName* or the subject's role to identify relevant policies. A UML class diagram of the *PolicyResource* class is presented in Figure 44.

4.9.2.2 The Policy Class

In this case a policy represents the basic component of an access control mechanism, and the policy defines the *operations* that must be evaluated as being true to grant a *permission*. In addition, the policy contains a field for storing a comment, plus a reference to the function that is requesting access.

4.9.2.3 The Policy Operation Class

This container is used to store the attribute-based rules that are evaluates as being either true or false. A rule contains three components: a *PolicyOperator* and two *operands*. The first *operand* can be a dynamic attribute or a fixed value that will be compared by the PIP. The second operand is a run-time attribute, which value is determined on demand (again by the PIP). The *PolicyOperator* represents the operation that takes place between the two operands.

4.9.2.4 The Logical Evaluation of ABAC

The proposed ABAC implementation relies on the SC logic to enforce the ABAC logical requirements. The ABAC mechanism has been inherited into all classes by bootstrapping a *CheckPermissions* function in the *BaseClass* object. This function is called during an object-initiation phase to ensure that access is checked before the SC (and therefore the agent) has access to an object, and the access control function is divided into three main phases. The first phase is used to quickly find all policies that are relevant to the resource that is being loaded, followed by a second phase in which an identity-based filter enables the SC to find policies that intersect both the resource and the user role (see Figure 45).

Once the subset of applicable policies has been identified, a third algorithm iterates through all policies to determine a user's effective access level. Within this algorithm, attribute-based rules are evaluated by the PIP using recursion and dynamic comparisons between objects. As a security measure, a string-based dictionary of allowable objects and properties is used to avoid the risks of fully dynamic evaluations.

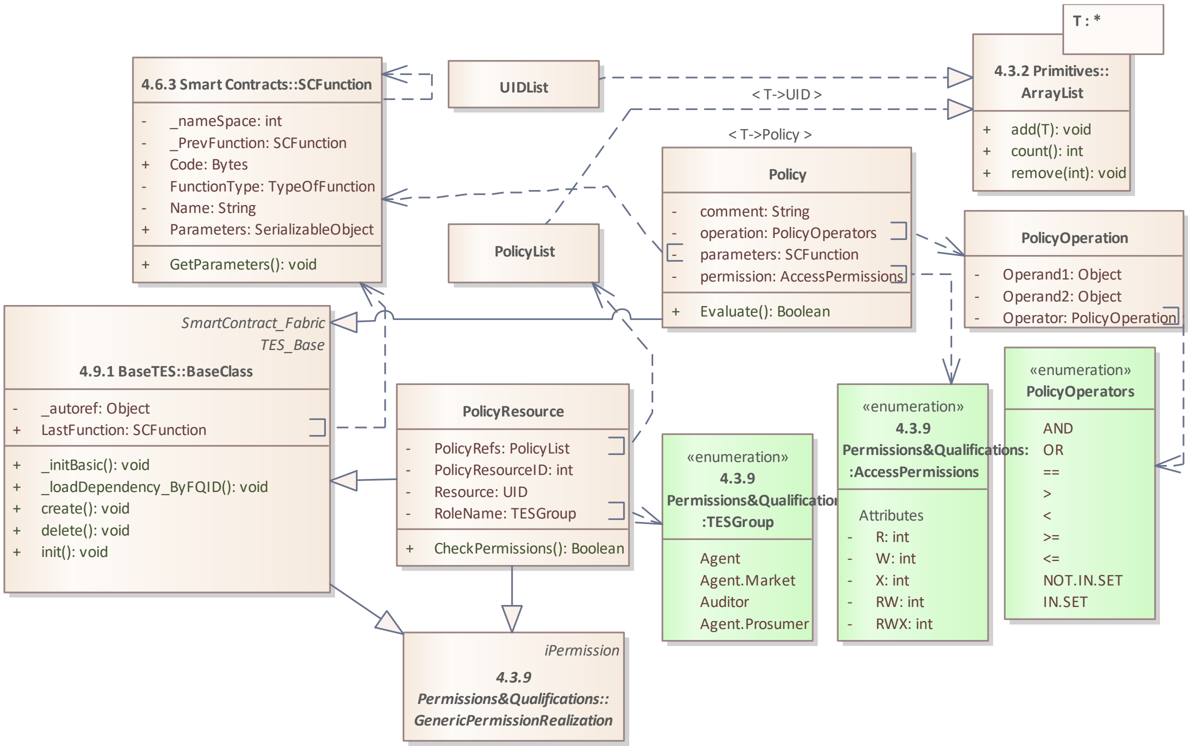


Figure 44. Objects used to represent the ABAC architecture.

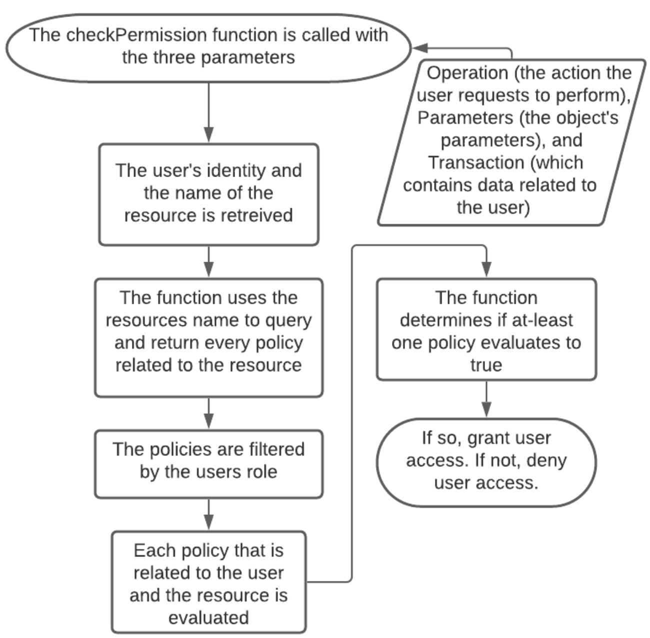


Figure 45. The policy-filtering algorithm finds policies based on the target resource and a user's role (covering the first and second algorithm).

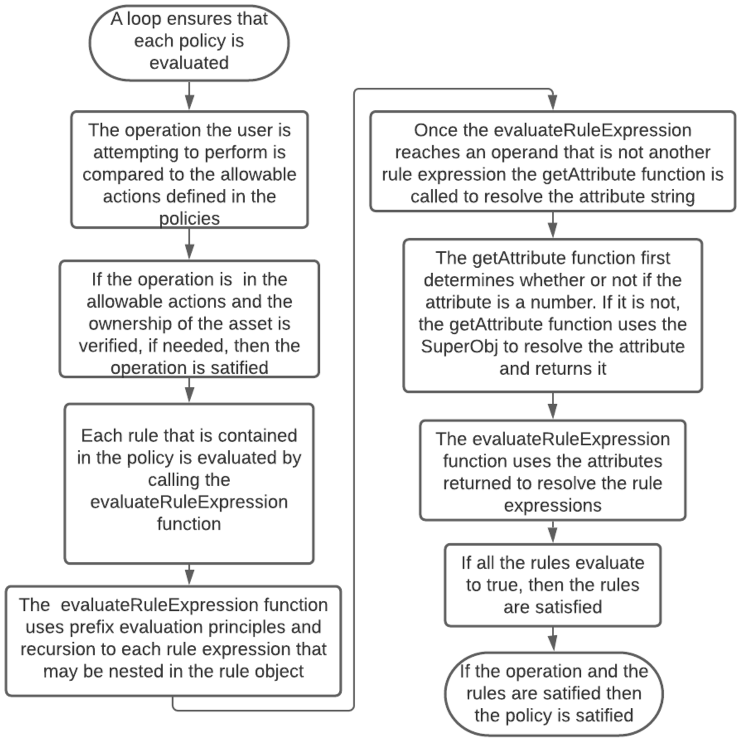


Figure 46. The internal algorithm used to evaluate attribute-based rules (the third algorithm).

In addition to the aforementioned mechanism, an experimental, off-the-chain mechanism is undergoing testing for use in further restricting an agent's access to the ledger. This mechanism works by encrypting data and distributing the keys used to encrypt the data across multiple systems (referred as to the key keepers). When a valid request is received (evaluated by a distributed policy engine), the original key keepers release their partial keys to the system that is requesting access. At this point the requester assembles the partial keys and is able to recover the original information stored in the ledger. An overview of the experimental approach is presented in Figure 47. In this case, the off-chain key provider is implemented in Python and relies on the use of Identity-Based Encryption to encrypt the data blocks. More research is needed to create a seamless solution.

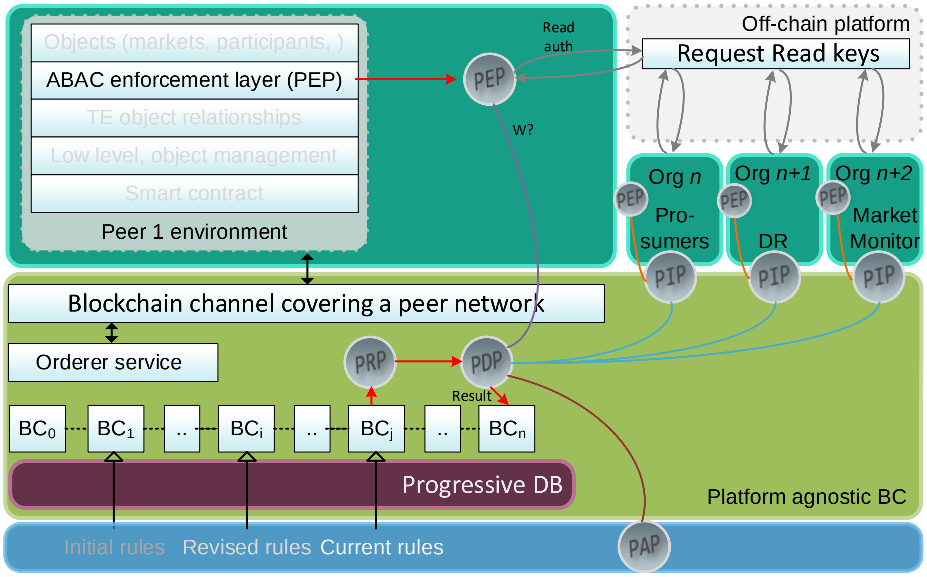


Figure 47. Implementing an ABAC system within a permissioned blockchain environment.

5.0 Conclusion

In this report, a series of templates designed for blockchain-based TES environments were presented. The proposed templates have been developed in a blockchain-agnostic manner, with flexibility and interoperability in mind. These templates are built based on the traits and design goals identified by prior researchers, with key elements been carried having been forward from their proposed models into the now reported templates. In addition, these templates have been based on existing standards, such as IEEE 2030.5 to promote interoperability with existent solutions.

To achieve this, the research team identified the key building blocks of a TES from an engineering perspective and captured them using behavioral and data models with blockchain as a TES-enabling technology. The resulting diagrams are expected to be easily implemented in SC solutions. Due to the template’s inheritance and composite capabilities, researchers will be able to easily customize the template to suit their own TES application needs, irrespective of the underlying DLT, based on an underlying assumption that their choice of blockchain offers SC-like capabilities as an inherent feature.

The current template version contains various objects that are intended to be representative of grid devices. In the current state, the template contains mechanisms for storing and accessing a participant’s identity and permissions while also enabling abstract grid resource modeling. The current components address the registration/qualification, negotiation, and operation processes of the ESI model.

6.0 References

Andoni, Merlinda, Valentin Robu, David Flynn, Simone Abram, Dale Geach, David Jenkins, Peter McCallum, and Andrew Peacock. 2019. "Blockchain technology in the energy sector: A systematic review of challenges and opportunities." *Renewable and Sustainable Energy Reviews*, 143-174.

Antonopoulos, Andreas M., and Gavin Wood. 2018. *Mastering Ethereum.* O'Reilly Media.

Boeyen, Sharon, Stefan Santesson, Tim Polk, Russ Housley, Stephen Farrell, and David Cooper. 2008. "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile." Request for Comments, May.

Burns, Martin, Eugine Song, and David Holmberg. 2018. *The Transactive Energy Abstract Component Model.* NIST.

Buterin, Vitalik. 2013. "Ethereum white paper."

Cali, Umit, Claudio Lima, Xuefei Li, and Yasuhiko Ogushi. 2019. "DLT / Blockchain in Transactive Energy Use Cases Segmentation and Standardization Framework." *IEEE PES Transactive Energy Systems Conference (TESC).* Minneapolis, Mn: IEEE.

Cazalet, Edward, William Cox, Alexander Krstulovic, William Miller, and Wilco Wijbrandi. 2016. "Transactive Energy Challenge: Common Transactive Services."

Eisele, Scott, Carlos Barreto, Abhishek Dubey, Xenofon Koutsoukos, Taha Eghtesad, Aron Laszka, and Anastasia Mavridou. 2020. "Blockchains for Transactive Energy Systems: Opportunities, Challenges, and Approaches." *Computer* 66-76.

Eisele, Scott, Carlos Barreto, Abhishek Dubey, Xenofon Koutsoukos, Taha Eghtesad, Aron Laszka, and Anastasia Mavridou. 2020. "Blockchains for Transactive Energy Systems: Opportunities, Challenges, and Approaches." *Computer* 66-76.

Gourisetti, S., S. Widergren, M. Mylrea, P. Wang, M. Borkum, A. Randall, and B. Bhattarai. 2019. *Blockchain Smart Contracts for Transactive Energy Systems.* PNNL Tech. Report 29017, Pacific Northwest National Laboratory.

Gourisetti, SNG, DJ Sebastian-Cardenas, Bishnu Bhattarai, Peng Wang, Steve Widergren, Mark Borkum, and Alysha. Randall. 2021. "Blockchain smart contract reference framework and program logic architecture for transactive energy systems." *Applied enegy.*

GridWise Architecture Council. 2015. *GridWise Transactive Energy Framework Version 1.0.* PNNL Tech. Report 22946, Richland: Pacific Northwest National Lab.

Hahn, Adam, Rajveer Singh, Chen-Ching Liu, and Sijie Chen. 2017. "Smart contract-based campus demonstration of decentralized transactive energy auctions." *IEEE Power&Energy Society Innovative Smart Grid Technologies Conference .* Arlington, Va.

Hammerstrom, Donald. 2019. *The Tranasctive Network Template Metamodel.* Pacific Northwest National Laboratory (PNNL Report 28420).

Holmberg, David, Martin Burns, Steven Bushby, Avi Gopstein, Tom McDermott, Yingying Tang, Qiuhua Huang, et al. 2019. *NIST Transactive EnergyModeling and Simulation Challenge Phase II Final Report.* NIST.

IEEE Std 2030.5-2018 (Revision of IEEE Std 2030.5-2013). 2018. "IEEE Standard for Smart Energy Profile Application Protocol."

Kuzlu, Murat, Manisa Pipattanasomporn, Levent Gurses, and Saifur Rahman. 2019. "Performance Analysis of a Hyperledger Fabric Blockchain Framework: Throughput, Latency and Scalability." *IEEE International Conference on Blockchain.* 536-540,.

Li, Zhiyi, Shay Bahramirad, Aleksi Paaso, Mingyu Yan, and Mohammad Shahidehpour. 2019. "Blockchain for decentralized transactive energy management system in networked microgrids." *The Electricity Journal.*

Liang, Xueping, Sachin Shetty, Deepak Tosh, Yafei Ji, and Danyi Li. 2018. "Towards a Reliable and Accountable Cyber Supply Chain in Energy Delivery System Using Blockchain." *International Conference on Security and Privacy in Communication Systems.* 122-138.

Lima, Claudio. 2018. "Developing Open and Interoperable DLT\/Blockchain Standards." *Computer.*

Mokhtari, Sasan, and Farrokh Rahimi. 2021. "Grid-Edge Blockchain-Based Transactive Energy Platform: Design and implementation." *IEEE Electrification Magazine.*

Mylrea, Michael, and Sri Nikhil Gupta Gourisetti. 2018. "Blockchain for Supply Chain Cybersecurity, Optimization and Compliance." *Resilience Week.*

Mylrea, Michael, Sri Nikhil Gupta Gourisetti, and H Culley. 2018. *Keyless Infrastructure Security: Technology Landscape Analysis Report.* Tech. Rep. 27453, Richland: Pacific Northwest National Laboratory.

Nakamoto, Satoshi. 2008. "Bitcoin: A Peer-to-Peer Electronic Cash System."

Patel, Dhiren, Benita Britto, Sanidhya Sharma, Kaustubh Gaikwad, Yash Dusing, and Mrinal Gupta. 2020. "Carbon Credits on Blockchain." *International Conference on Innovative Trends in Information Technology (ICITIIT).* 1-5.

Sebastian, David J., Sri Nikhil Gupta Gourisetti, Michael Mylrea, Anthony Morlaez, Garrett Day, Vinod Tatireddy, Craig H. Allwardt, et al. 2021. "Digital data provenance for the power grid based on a Keyless Infrastructure Security Solution." *IEEE Resilience Week.* Salt Lake City, UT.

Tonghe, Wanga, Guobc Jian, Aib Songpu, and Caob Junwei. 2021. "RBT: A distributed reputation system for blockchain-based peer-to-peer energy trading with fairness consideration." *Applied Energy.*

Troncia, M., M. Galici, M. Mureddu, E. Ghiani, and F. Pilo. 2019. "Distributed ledger technologies for peer-to-peer local markets in distribution networks." *Energies* 3249.

Tucker, David, and Grant , Johnson. 2021. "Blockchain for Optimized Security and Energy (BLOSEM)." May 2019. https://netl.doe.gov/sites/default/files/netl-file/21SC_Tucker_B.pdf.

Wang, Rui, Kejiang Ye, and authorCheng-Zhong Xu. 2019. "Performance Benchmarking and Optimization for Blockchain Systems: A Survey." *International Conference on Blockchain.* 171-185.

Widergren, Steve. 2018. *Interoperability Strategic Vision.* Pacific Northwest National Lab, PNNL Report - 27320.

—. 2016. "Transactive Energy for Distributed Resource Integration." Edited by AIT Austrian Institute of Technology.

Yaga, Dylan, Peter Mell, Nik Roby, and Karen Scarfone. 2018. "Blockchain Technology Overview."

Zheng, Xiaoying, Zhu Yongxin, and Si Xueming. 2019. "A Survey on Challenges and Progresses in Blockchain Technologies: A Performance and Security Perspective." *Applied Sciences* 4731.

Appendix A – Blockchain-Architecture for Transactive Energy Systems in-depth review.

A.1 Basic Data Types

In this section a collection of basic data types that are used within the template are introduced. The details of these objects will be explored in these upcoming sections.

A.1.1 Basic Object Models

This package provides an overview of foundational classes that are used as the base constructs for the Blockchain Architecture Transactive Energy Systems (B-A TES) template framework. These elements work together to 1) Uniquely identify objects; and 2) support an event-based architecture.

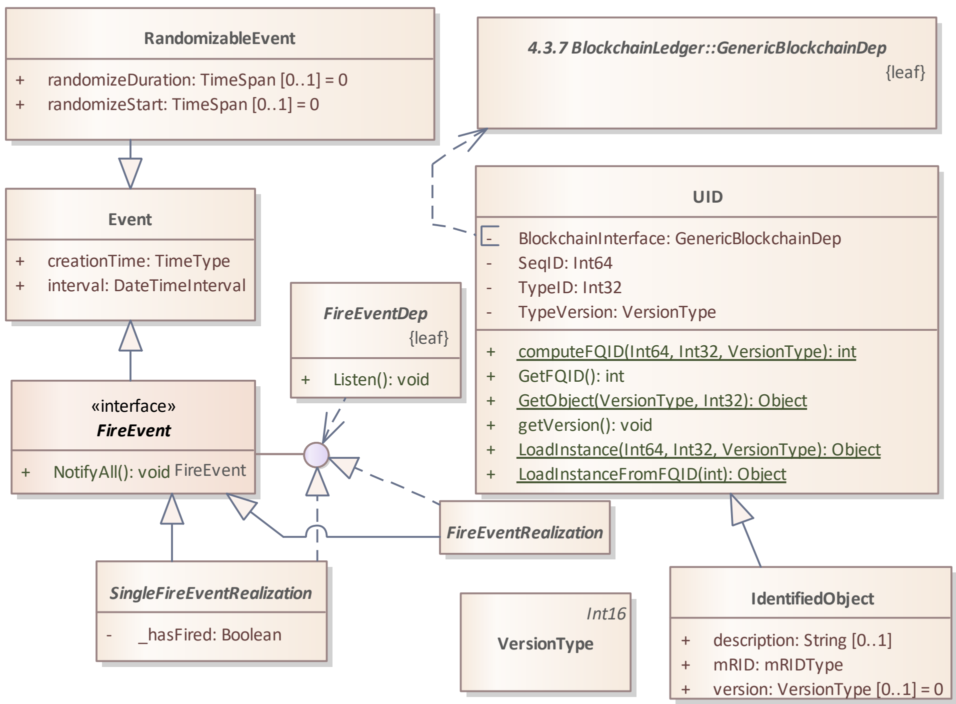


Figure 48. Overview of the BasicObjects' package components

A.1.1.1 FireEvent

Class «interface» in package '4.3.1 Basic Objects'

STRUCTURAL PART OF FireEvent
FireEvent: ProvidedInterface

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from «interface» FireEvent to Event

OPERATIONS
NotifyAll (): **void** Public *Details:*

A.1.1.2 FireEventDep

Class in package '4.3.1 Basic Objects'

Details: This is a listener.

CONNECTORS
Dependency Source -> Destination From: : FireEventDep : Class, Public To: FireEvent : ProvidedInterface, Public

OPERATIONS
Listen (): **void** Public *Details:*

A.1.1.3 FireEventRealization

Class in package '4.3.1 Basic Objects'

OUTGOING STRUCTURAL RELATIONSHIPS
Realization from FireEventRealization to FireEvent
Generalization from FireEventRealization to «interface» FireEvent

A.1.1.4 SingleFireEventRealization

Class in package '4.3.1 Basic Objects'

Details: This type of event can only occur once. The object fires as soon as the start of the interval

OUTGOING STRUCTURAL RELATIONSHIPS
Realization from SingleFireEventRealization to FireEvent
Generalization from SingleFireEventRealization to «interface» FireEvent

ATTRIBUTES
_hasFired: **Boolean** Private *Details:*

A.1.1.5 UID

Class in package '4.3.1 Basic Objects'

Details: This class enables to uniquely identify any instance of an object using two indexes. It also provides static methods from loading data from the blockchain.

CONNECTORS

Dependency Source -> Destination
From: : UID : Class , Public
To: GenericBlockchainDep : Class , Public

ATTRIBUTES

BlockchainInterface : **GenericBlockchainDep** Private
Details:

SeqID : **Int64** Private
Details: This represents an instance number within the object denoted by TypeID
Alias: mRID_High
Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

TypeID : **Int32** Private
Details: This represents a unique number that can be used to globally identify an object type within the template. Similar to a GUID but can be shorten to satisfy an application needs
Alias: mRID_Low
Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

TypeVersion : **VersionType** Private
Details: This is a reference to the object's version.
In general a system should be designed to be backwards compatible, or at least backward-aware. Having this property ensures that new SC versions do not corrupt the world state.
Alias: version
Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

OPERATIONS

computeFQID (SeqID : **Int64** , TypeID : **Int32** , TypeVersion : **VersionType**) : **int** Public
Details: This function enables an SC to compute the Fully Qualified Identity of any given instance if the object ID, instance ID and version information is provided.

GetFQID () : **int** Public
Details: Computes an object's FQID from an object that is already loaded into memory

GetObject (Version : **VersionType** , TypeID : **Int32**) : **Object** Public
Details: Enables to retrieve an object prototype given a unique object ID, and its version. Static method

getVersion () : **void** Public
Details:

LoadInstance (SeqID : **Int64** , TypeID : **Int32** , TypeVersion : **VersionType**) : **Object** Public
Details: Enables to load a specific instance using the object ID, its sequence and version number. This function should rely on the *BlockchainInterface* to load the requested object.

LoadInstanceFromFQID (FQID : **int**) : **Object** Public
Details: Enables to load a specific instance using a FQID

A.1.1.6 IdentifiedObject

Class in package '4.3.1 Basic Objects'

Details: This object was adapted/taken from IEEE 2030.5. This is a root class to provide common identity scheme for all objects needing to be uniquely identifiable.

OUTGOING STRUCTURAL RELATIONSHIPS

Generalization from IdentifiedObject to UID

ATTRIBUTES
description : **String** Public *Details:* This property was adapted/taken from IEEE2030.5. The description is a text describing the object function, it can also capture notes Multiplicity: (0..1, Allow duplicates: 0, Is ordered: False)
mRID : **mRIDType** Public *Details:* This property was adapted/taken from IEEE2030.5. Used to represent the global identifier of the object, is the concatenation of TypeID||SeqID Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
version : **VersionType** Public = 0 *Details:* Contains the version number of the object. Useful in handling multiple versions within the ledger Multiplicity: (0..1, Allow duplicates: 0, Is ordered: False)

A.1.1.7 Event

Class in package '4.3.1 Basic Objects'

Details: This object was adapted/taken from IEEE 2030.5. An Event indicates information that applies to a particular period of time. Events follow the BC-agreed time reference.

ATTRIBUTES
creationTime : **TimeType** Public *Details:* This property was adapted/taken from IEEE2030.5. Indicates the time at which the Event was created. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
interval : **DateTimeInterval** Public *Details:* This property was adapted/taken from IEEE2030.5 The period during which the Event applies. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.1.1.8 RandomizableEvent

Class in package '4.3.1 Basic Objects'

Details: This object was adapted/taken from IEEE 2030.5. This is an Event that can indicate time ranges over which the start time and duration can be randomized over a period of time.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from RandomizableEvent to Event

ATTRIBUTES
randomizeDuration : **TimeSpan** Public = 0 *Details:* Number of seconds boundary inside which a random value must be selected to be applied to the associated interval duration, to avoid sudden synchronized demand changes. If related to price level changes, sign may be ignored. Valid range is -3600 to 3600. If not specified, 0 is the default. Multiplicity: (0..1, Allow duplicates: 0, Is ordered: False)
randomizeStart : **TimeSpan** Public = 0 *Details:* Number of seconds boundary inside which a random value must be selected to be applied to the associated interval start time, to avoid sudden synchronized demand changes. If related to price level changes, sign may be ignored. Valid range is -3600 to 3600. If not specified, 0 is the default. Multiplicity: (0..1, Allow duplicates: 0, Is ordered: False)

A.1.1.9 VersionType

Class in package '4.3.1 Basic Objects'

Details: This object was adapted/taken from IEEE 2030.5. This field indicates an object's version. An object should maintain the same TypeID across its life-cycle, while advancing the version when properties are added or changed

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from VersionType to Int16

A.1.2 Primitives

In this package an overview of data primitives is introduced. These data type models are intended to support a wide variety of data needs, provide generic structures and in general support more complex template models. In addition, they provide clearance on the size, supported operations and intended usage.

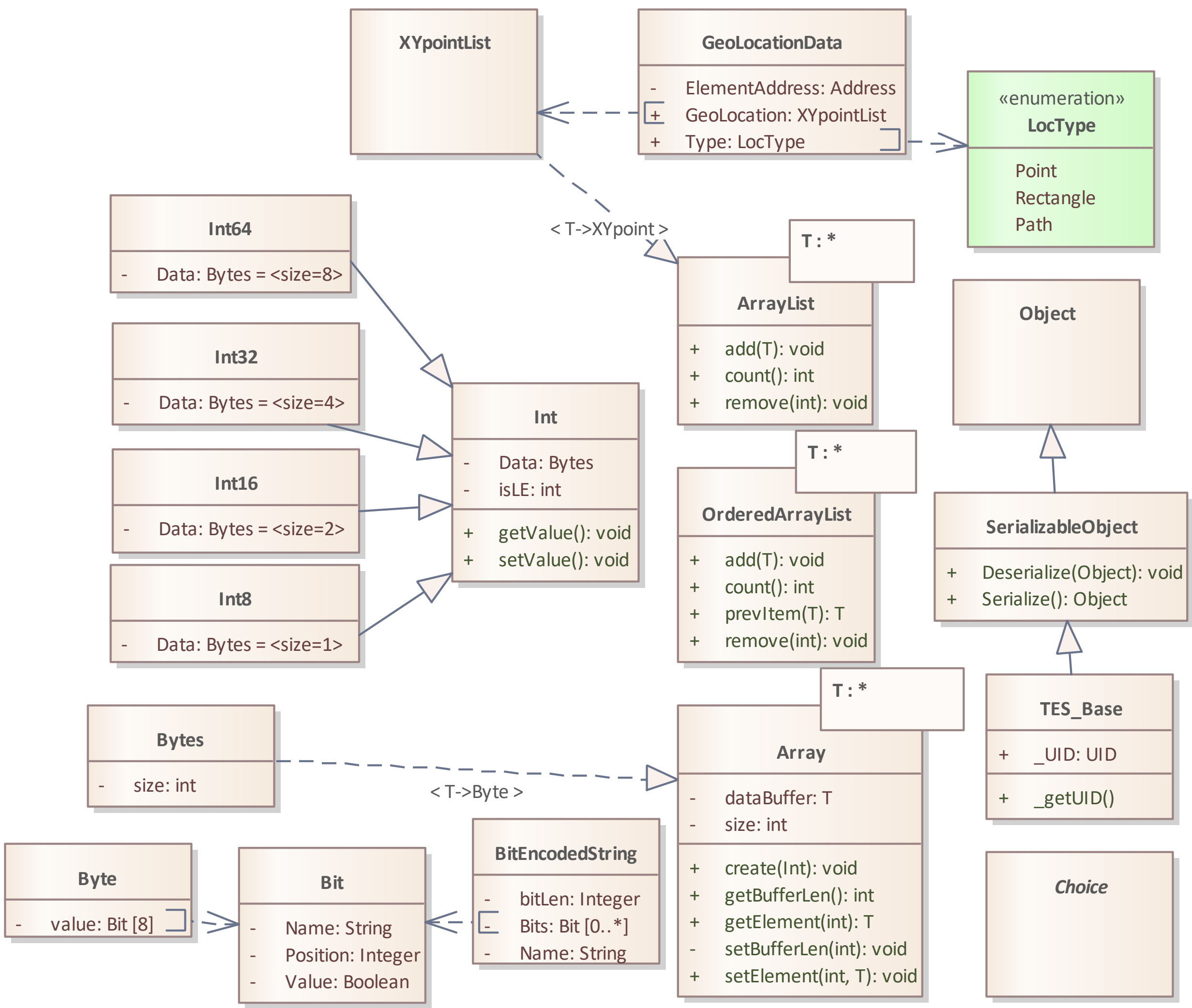


Figure 49. Overview of the Primitives' package components

A.1.2.1 Array

Class in package '4.3.2 Primitives'

Details: This is a generic structure used to represent any array of T elements. An array has a fixed length given by size, this value is not modifiable after the initial setup.

ATTRIBUTES
dataBuffer : **T** Private *Details:* This is the underlying structure that contains array data, its is assumed that Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

ATTRIBUTES
size : **int** Private *Details:*

OPERATIONS
create (Size : **Int**) : **void** Public *Details:* This function initializes the array size.
getBufferLen () : **int** Public *Details:* This gets the total amount of elements that can fit in this array.
getElement (I : **int**) : **T** Public *Details:* This function retrieves an element at position *I*
setBufferLen (len : **int**) : **void** Private *Details:* This is an internal function that gets called during initialization.
setElement (index : **int** , value : **T**) : **void** Public *Details:* This function sets a value at position *I* from the array.

A.1.2.2 ArrayList

Class in package '4.3.2 Primitives'

Details: This is a generic structure used to represent any *arrayList* of T elements. Elements within an *arrayList* can be added/removed at any time.

OPERATIONS
add (T : **T**) : **void** Public *Details:* This function is used to add an element of type T into the array list.
count () : **int** Public *Details:* This function retrieves the number of items contained in the array list.
remove (Index : **int**) : **void** Public *Details:* This function removes an object from the list.

A.1.2.3 Bit

Class in package '4.3.2 Primitives'

Details: This represents a single Bit (0,1). It can have a name and a position, with the LSb referenced as 0.

CONNECTORS
Dependency Source -> Destination From: : BitEncodedString : Class , Public To: Bit : Class , Public
Dependency Source -> Destination From: : Byte : Class , Public To: Bit : Class , Public

ATTRIBUTES
Name : **String** Private

ATTRIBUTES
Details: This can be used to define a bit name, useful for bit masking Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Position : **Integer** Private *Details:* This denotes the bit position, the LSb is considered to be at position 0 Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Value : **Boolean** Private *Details:* This can be a 0 or 1, potentially representing a True/False value Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.1.2.4 BitEncodedString

Class in package '4.3.2 Primitives'

Details: This represents a string of bits with predetermined length, with each bit position representing a T/F flag.

CONNECTORS
Dependency Source -> Destination From: : BitEncodedString : Class , Public To: Bit : Class , Public

ATTRIBUTES
bitLen : **Integer** Private *Details:* This field denotes the size of the bit string (size given in bits) Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Bits : **Bit** Private *Details:* An array of bits, the length of the array matches bitLen Multiplicity: (0..*, Allow duplicates: 0, Is ordered: False)
Name : **String** Private *Details:* This is the name of the string of zeros and ones Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.1.2.5 Byte

Class in package '4.3.2 Primitives'

Details: This represents a collection of 8 bits, order goes from MSb to LSb.

CONNECTORS
Dependency Source -> Destination From: : Byte : Class , Public To: Bit : Class , Public

ATTRIBUTES
value : **Bit** Private *Details:* Constraints: <256 : >=0 :

A.1.2.6 Bytes

Class in package '4.3.2 Primitives'

Details: This represents an array of bytes.

OUTGOING STRUCTURAL RELATIONSHIPS
Realization from Bytes to Array

CONNECTORS
Dependency Source -> Destination From: : BlockData : Class , Public To: Bytes : Class , Public
Dependency Source -> Destination From: : BlockHeader : Class , Public To: Bytes : Class , Public
Dependency Source -> Destination From: : BlockData : Class , Public To: Bytes : Class , Public
Dependency Source -> Destination From: : BlockHeader : Class , Public To: Bytes : Class , Public

ATTRIBUTES
size : **int** Private *Details:* Used to denote the size of the byte array (in bytes) Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.1.2.7 Choice

Class in package '4.3.2 Primitives'

Details: This class symbolizes an object from only one option should be selected. Objects that inherit this class should rely on ports to present options. E.g., select exactly one option from the list of available ports.

A.1.2.8 GeoLocationData

Class in package '4.3.2 Primitives'

STRUCTURAL PART OF GeoLocationData
Property : Property

CONNECTORS
Dependency Source -> Destination From: : GeoLocationData : Class , Public To: LocType : Enumeration , Public
Dependency Source -> Destination From: : GeoLocationData : Class , Public To: XYpointList : Class , Public
Dependency Source -> Destination From: : GridSystem : Class , Public To: GeoLocationData : Class , Public

ATTRIBUTES
ElementAddress : **Address** Private *Details:*
GeoLocation : **XYpointList** Public *Details:*
Type : **LocType** Public *Details:*

A.1.2.9 Int

Class in package '4.3.2 Primitives'

Details: Is a generic Int of size(Data). Default is Big Endian, can be encoded in Little Endian.

ATTRIBUTES
Data : **Bytes** Private *Details:*
isLE : **int** Private *Details:*

OPERATIONS
getValue () : **void** Public *Details:* Properties: native = true
setValue () : **void** Public *Details:*

A.1.2.10 Int16

Class in package '4.3.2 Primitives'

Details: This is a 2 byte integer, unsigned unless otherwise noted. Usually with the range 0-65,535.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from Int16 to Int

ATTRIBUTES
Data : **Bytes** Private = <size=2> *Details:* Data should be transparently handled as an integer of N bytes. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.1.2.11 Int32

Class in package '4.3.2 Primitives'

Details: This is a 4 byte integer, unsigned unless otherwise noted. Usually with the range 0-4,294,967,295.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from Int32 to Int

ATTRIBUTES
Data : **Bytes** Private = <size=4> *Details:* Data should be transparently handled as an integer of N bytes. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.1.2.12 Int64

Class in package '4.3.2 Primitives'

Details: This is a 8 byte integer, unsigned unless otherwise noted. Usually with the range 0-18,446,744,073,709,551,615.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from Int64 to Int

ATTRIBUTES
Data : **Bytes** Private = <size=8> *Details:* Data should be transparently handled as an integer of N bytes. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.1.2.13 Int8

Class in package '4.3.2 Primitives'

Details: This is a 1 byte integer, unsigned unless otherwise noted. Usually with the range 0-255.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from Int8 to Int

ATTRIBUTES
Data : **Bytes** Private = <size=1> *Details:* Data should be transparently handled as an integer of N bytes. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.1.2.14 Object

Class in package '4.3.2 Primitives'

Details: This entity represents a collection of properties, attributes and methods. This is an abstract representation, all objects in this template are assumed to inherit this object.

A.1.2.15 OrderedArrayList

Class in package '4.3.2 Primitives'

Details: This is a generic structure used to represent any ordered arrayList of T elements. This type of array is guaranteed to maintain order, items can be removed from anywhere, but new elements are always added to the end.

OPERATIONS
add (T : **T**) : **void** Public *Details:* This function is used to add an element of type T into the ordered array list.

OPERATIONS
count () : **int** Public *Details:* This function retrieves the total number of items in the list.
prevItem (T : **T**) : **T** Public *Details:* This function return the item located before item T.
remove (Index : **int**) : **void** Public *Details:* This function removes the item given by index from the list.

A.1.2.16 SerializableObject

Class in package '4.3.2 Primitives'

Details: This class represents the ability of an object to be serialized into another object (usually a string). This enables interoperability across systems to occur.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from SerializableObject to Object

OPERATIONS
Deserialize (SerilizedObject : **Object**) : **void** Public *Details:* Takes the underlying object and serializes into another object (usually as string). The serialization results should remain compatible with other systems.
Serialize () : **Object** Public *Details:* This function loads an object based on a previously serialized result. The source object can be any object or format, common examples include: *JSON* *Datapack* *Protocol buffers.*

A.1.2.17 TES_Base

Class in package '4.3.2 Primitives'

Details: This represents the parent object that most objects within the TES template should reference. Its main properties are being able to have a unique ID and being serializable/deserializable.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from TES_Base to SerializableObject

ATTRIBUTES
_UID : **UID** Public *Details:*

OPERATIONS
_getUID () : Public *Details:*

A.1.2.18 XYpointList

Class in package '4.3.2 Primitives'

OUTGOING STRUCTURAL RELATIONSHIPS
Realization from XYpointList to ArrayList

CONNECTORS
Dependency Source -> Destination From: : GeoLocationData : Class , Public To: XYpointList : Class , Public

A.1.2.19 LocType

Enumeration in package '4.3.2 Primitives'

CONNECTORS
Dependency Source -> Destination From: : GeoLocationData : Class , Public To: LocType : Enumeration , Public

ENUMERATION:
Point
Rectangle
Path

A.1.3 Time

This package presents an overview of time-related constructs that will be used in this work. The data models are intended to support a common ground, in which all applicants are subject to the same time references and requirements.

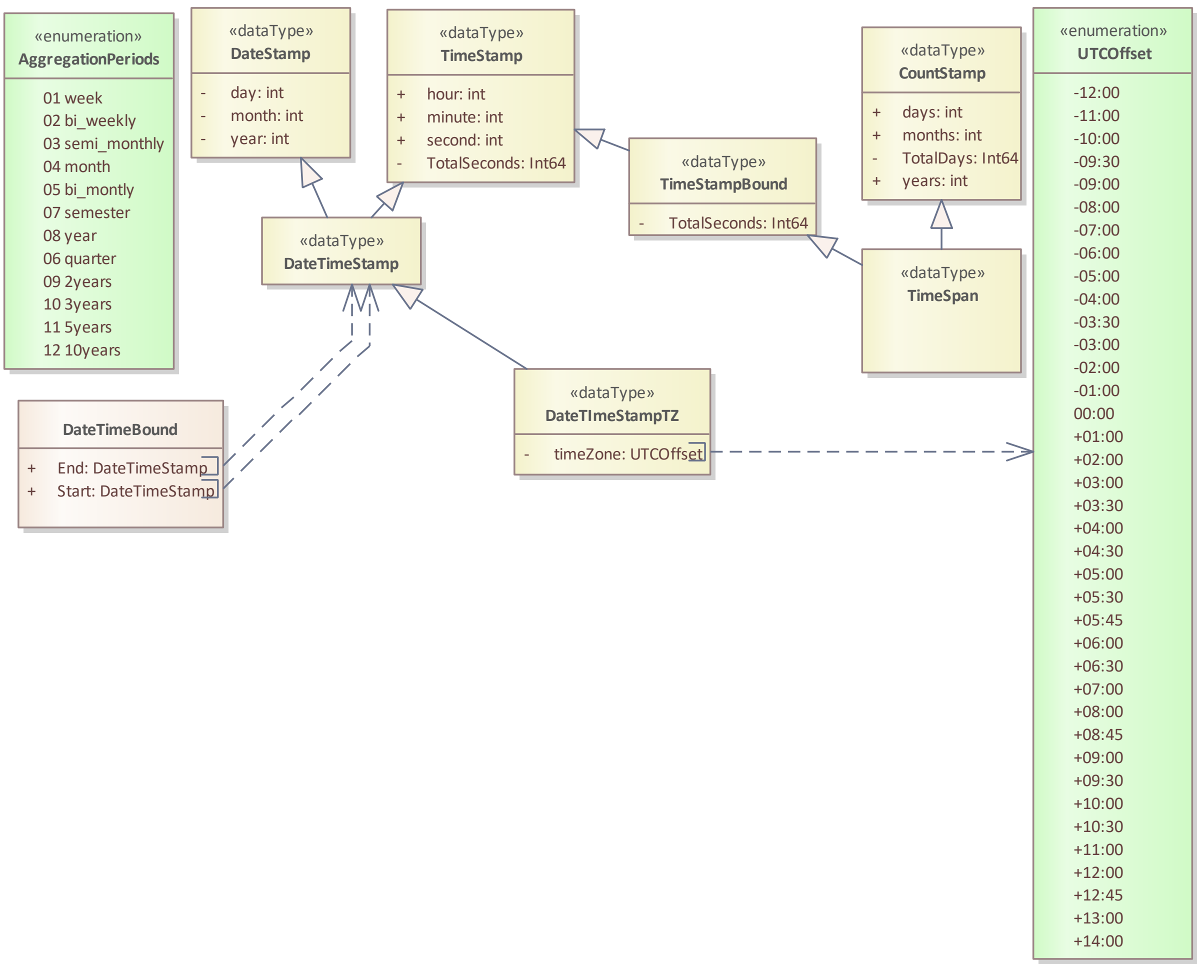


Figure 50. Overview of the TimeObjects' package components

A.1.3.1 DateTimeBound

Class in package '4.3.3 Time'

CONNECTORS
Dependency Source -> Destination From: : DateTimeBound : Class , Public To: TimeX509 : Class , Public
Dependency Source -> Destination From: : DateTimeBound : Class , Public To: DateTimeStamp : DataType , Public

CONNECTORS
Dependency Source -> Destination From: : DateTimeBound : Class , Public To: TimeX509 : Class , Public
Dependency Source -> Destination From: : DateTimeBound : Class , Public To: DateTimeStamp : DataType , Public

ATTRIBUTES
End : **DateTimeStamp** Public *Details:*
Start : **DateTimeStamp** Public *Details:*

A.1.3.2 AggregationPeriods

Enumeration in package '4.3.3 Time'

Details: This enumeration represent aggregation periods, or periods of time in which measurements, records, and other related activity are aggregated.

CONNECTORS
Dependency Source -> Destination From: : ReliabilityMetrics : Class , Public To: AggregationPeriods : Enumeration , Public

ENUMERATION:
01 week
02 bi_weekly
03 semi_monthly
04 month
05 bi_montly
07 semester
08 year
06 quarter
09 2years
10 3years
11 5years
12 10years

A.1.3.3 CountStamp

DataType in package '4.3.3 Time'

ATTRIBUTES
days : **int** Public *Details:*
months : **int** Public *Details:*
TotalDays : **Int64** Private *Details:*
years : **int** Public *Details:*

A.1.3.4 DateStamp

DataType in package '4.3.3 Time'

ATTRIBUTES
day : **int** Private *Details:*
month : **int** Private *Details:*
year : **int** Private *Details:*

A.1.3.5 DateTimeStamp

DataType in package '4.3.3 Time'

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from DateTimeStamp to TimeStamp
Generalization from DateTimeStamp to DateStamp

CONNECTORS
Dependency Source -> Destination From: : DateTimeBound : Class , Public To: DateTimeStamp : DataType , Public
Dependency Source -> Destination From: : DateTimeBound : Class , Public To: DateTimeStamp : DataType , Public

A.1.3.6 DateTImeStampTZ

DataType in package '4.3.3 Time'

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from DateTImeStampTZ to DateTimeStamp

CONNECTORS
Dependency Source -> Destination From: : DateTImeStampTZ : DataType , Public To: UTCOffset : Enumeration , Public

ATTRIBUTES
timeZone : **UTCOffset** Private *Details:*

A.1.3.7 TimeSpan

DataType in package '4.3.3 Time'

Details: This structure can be used to represent long-term time spans that can be on the order of years.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from TimeSpan to CountStamp
Generalization from TimeSpan to TimeStampBound

A.1.3.8 TimeStamp

DataType in package '4.3.3 Time'

ATTRIBUTES
hour : **int** Public *Details:*
minute : **int** Public *Details:*
second : **int** Public *Details:*
TotalSeconds : **Int64** Private *Details:*

A.1.3.9 TimeStampBound

DataType in package '4.3.3 Time'

Details: This object is designed to automatically roll over

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from TimeStampBound to TimeStamp

ATTRIBUTES
TotalSeconds : **Int64** Private *Details:* Constraints: value<86400 : Invariant

A.1.3.10 UTCOffset

Enumeration in package '4.3.3 Time'

Details: This enumeration lists all valid UTC offsets used to express time zones.

CONNECTORS
Dependency Source -> Destination From: : DateTImeStampTZ : DataType , Public To: UTCOffset : Enumeration , Public

ENUMERATION:
-12:00
-11:00
-10:00
-09:30
-09:00
-08:00
-07:00

ENUMERATION:
-06:00
-05:00
-04:00
-03:30
-03:00
-02:00
-01:00
00:00
+01:00
+02:00
+03:00
+03:30
+04:00
+04:30
+05:00
+05:30
+05:45
+06:00
+06:30
+07:00
+08:00
+08:45
+09:00
+09:30
+10:00
+10:30
+11:00
+12:00
+12:45
+13:00
+14:00

A.1.4 Math

This package groups objects that have mathematical or engineering applicability but are usually not natively supported by programming languages. This diagram may be expanded in the future to introduce more definitions.

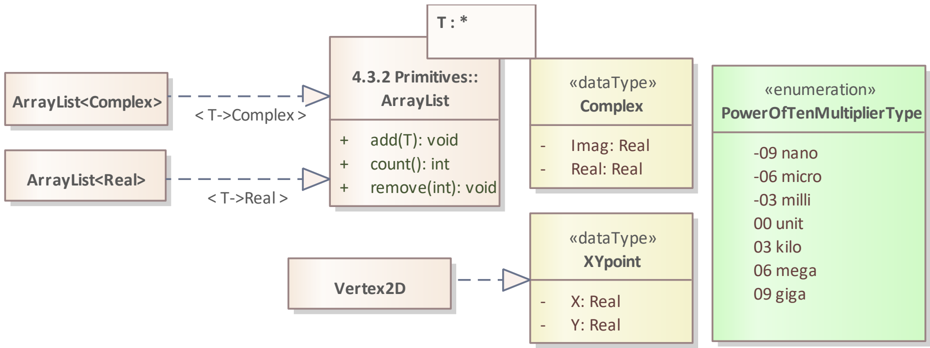


Figure 51. Overview of the Math's package components

A.1.4.1 ArrayList<Complex>

Class in package '4.3.4 Math'

Details: This class represents an *ArrayList* of complex numbers.

OUTGOING STRUCTURAL RELATIONSHIPS
Realization from ArrayList<Complex> to ArrayList

A.1.4.2 ArrayList<Real>

Class in package '4.3.4 Math'

Details: This class represents an array of real numbers.

OUTGOING STRUCTURAL RELATIONSHIPS
Realization from ArrayList<Real> to ArrayList

A.1.4.3 Vertex2D

Class in package '4.3.4 Math'

Details: This object represents a vertex, which is an specialization of an XYpoint

OUTGOING STRUCTURAL RELATIONSHIPS
Realization from Vertex2D to XYpoint

A.1.4.4 Complex

DataType in package '4.3.4 Math'

Details: This data type is used to represent a complex number.

ATTRIBUTES
Imag : **Real** Private *Details:* This represents the imaginary part of a complex number. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Real : **Real** Private *Details:* This represents the real part of a complex number, the underlying data type is numeric/float/double/real. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.1.4.5 PowerOfTenMultiplierType

Enumeration in package '4.3.4 Math'

Details: This enumeration is used to represent a power of ten multiplier, indexes can be negative or positive.

-9 = nano=x10^-9
-6 = micro=x10^-6
-3 = milli=x10^-3
0 = none=x1 (default, if not specified)
1 = deca=x10
2 = hecto=x100
3 = kilo=x1000
6 = Mega=x10^6
9 = Giga=x10^9
This object was taken from IEEE 2030.5

CONNECTORS
Dependency Source -> Destination From: : ReadingType : Class , Public To: PowerOfTenMultiplierType : Enumeration , Public

ENUMERATION:
-09 nano
-06 micro
-03 milli
00 unit
03 kilo
06 mega
09 giga

A.1.4.6 XYpoint

DataType in package '4.3.4 Math'

Details: This data type represents an XY coordinate point.

ATTRIBUTES
X : **Real** Private *Details:* This represents the X coordinate on an XY system. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Y : **Real** Private *Details:* This represents the Y coordinate on an XY system. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.1.5 TrackableObjects

In this case, the diagram contains presents classes that can be used to track an object's changes through the *trackable* interface. Objects that inherit these interfaces can be used to track an instance's particular history across its lifecycle. The interface can be used to store a full copy of the previous state (e.g., when a ledger is present),or two store a hash only (for off-chain applications).

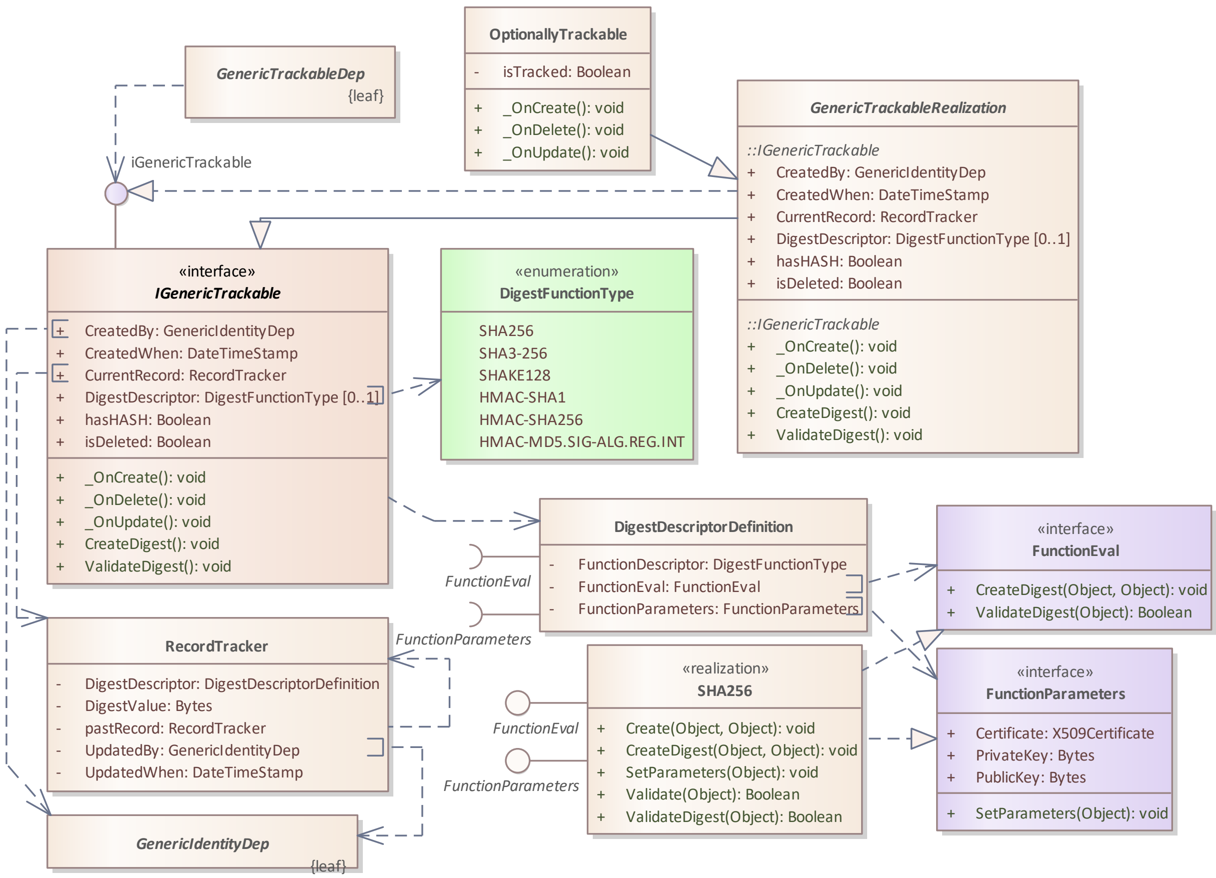


Figure 52. Overview of the TrackableClass' package components

A.1.5.1 DigestDescriptorDefinition

Class in package '4.3.5 TrackableObjects'

CONNECTORS
Dependency Source -> Destination From: : DigestDescriptorDefinition : Class , Public To: FunctionEval : Interface , Public
Dependency Source -> Destination From: : DigestDescriptorDefinition : Class , Public To: FunctionParameters : Interface , Public
Dependency Source -> Destination From: : IGenericTrackable : Class , Public To: DigestDescriptorDefinition : Class , Public

ATTRIBUTES
FunctionDescriptor : **DigestFunctionType** Private *Details:*
FunctionEval : **FunctionEval** Private *Details:*
FunctionParameters : **FunctionParameters** Private *Details:*

A.1.5.2 GenericTrackableDep

Class in package '4.3.5 TrackableObjects'

Details: Objects that inherit this class expose an interface requirement to access an objects past history

CONNECTORS
Dependency Source -> Destination From: : GenericTrackableDep : Class , Public To: iGenericTrackable : ProvidedInterface , Public

A.1.5.3 GenericTrackableRealization

Class in package '4.3.5 TrackableObjects'

Details: This inheritable class provides the basic mechanisms to track an object's existence though a system. Specific callbacks can be attached to the *OnCreate(), onUpdate(), onDelete()* functions. For blockchain-based implementations *onDelete()* can be used to mark a record as active.

STRUCTURAL PART OF GenericTrackableRealization
ProvidedInterface2 : ProvidedInterface
ProvidedInterface3 : ProvidedInterface

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from GenericTrackableRealization to «interface» IGenericTrackable
Realization from GenericTrackableRealization to iGenericTrackable

A.1.5.4 IGenericTrackable

Class «interface» in package '4.3.5 TrackableObjects'

STRUCTURAL PART OF IGenericTrackable
iGenericTrackable : ProvidedInterface

CONNECTORS
Dependency Source -> Destination From: : IGenericTrackable : Class , Public To: DigestDescriptorDefinition : Class , Public
Dependency Source -> Destination From: : IGenericTrackable : Class , Public To: DigestFunctionType : Enumeration , Public
Dependency Source -> Destination From: : IGenericTrackable : Class , Public To: RecordTracker : Class , Public
Dependency Source -> Destination From: : IGenericTrackable : Class , Public To: GenericIdentityDep : Class , Public

ATTRIBUTES
CreatedBy : **GenericIdentityDep** Public *Details:*
CreatedWhen : **DateTimeStamp** Public *Details:*
CurrentRecord : **RecordTracker** Public *Details:*
DigestDescriptor : **DigestFunctionType** Public *Details:*
hasHASH : **Boolean** Public *Details:*
isDeleted : **Boolean** Public *Details:*

OPERATIONS
_OnCreate () : **void** Public *Details:*
_OnDelete () : **void** Public *Details:*
_OnUpdate () : **void** Public *Details:*
CreateDigest () : **void** Public *Details:*
ValidateDigest () : **void** Public *Details:*

A.1.5.5 OptionallyTrackable

Class in package '4.3.5 TrackableObjects'

Details: This class represents a inheritable class that can enable tracking as an optional service.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from OptionallyTrackable to GenericTrackableRealization

ATTRIBUTES
isTracked : **Boolean** Private

ATTRIBUTES
Details:

OPERATIONS
_OnCreate () : **void** Public *Details:* This is an internal wrapper that can call the parent method only if the *isTracked* flag is set to True.
_OnDelete () : **void** Public *Details:* This is an internal wrapper that can call the parent method only if the *isTracked* flag is set to True.
_OnUpdate () : **void** Public *Details:* This is an internal wrapper that can call the parent method only if the *isTracked* flag is set to True.

A.1.5.6 RecordTracker

Class in package '4.3.5 TrackableObjects'

CONNECTORS
Dependency Source -> Destination From: : RecordTracker : Class , Public To: RecordTracker : Class , Public
Dependency Source -> Destination From: : RecordTracker : Class , Public To: GenericIdentityDep : Class , Public
Dependency Source -> Destination From: : IGenericTrackable : Class , Public To: RecordTracker : Class , Public
Dependency Source -> Destination From: : RecordTracker : Class , Public To: RecordTracker : Class , Public

ATTRIBUTES
DigestDescriptor : **DigestDescriptorDefinition** Private *Details:*
DigestValue : **Bytes** Private *Details:*
pastRecord : **RecordTracker** Private *Details:*
UpdatedBy : **GenericIdentityDep** Private *Details:*
UpdatedWhen : **DateTimeStamp** Private *Details:*

A.1.5.7 SHA256

Class «Realization» in package '4.3.5 TrackableObjects'

Details: This object represents a sample digest class that must implement the function evaluation functions along with the function parameters. Developers need to build realization of FunctionEval and FunctionParameters to enable dynamic checking.

OUTGOING STRUCTURAL RELATIONSHIPS
Realization from «Realization» SHA256 to FunctionParameters
Realization from «Realization» SHA256 to FunctionEval

OPERATIONS
Create (rawData : **Object** , parameters : **Object**) : **void** Public *Details:* This field represents the actual creation logic that a digest must provide. Properties: Implements = FunctionEval.Create ImplementsGuid = {1CBF9EDD-4F74-4537-BD99-7E59D05E18D7}
CreateDigest (rawData : **Object** , parameters : **Object**) : **void** Public *Details:* Properties: Implements = FunctionEval.CreateDigest ImplementsGuid = {1CBF9EDD-4F74-4537-BD99-7E59D05E18D7}
SetParameters (Any : **Object**) : **void** Public *Details:* This field should load/search for the required parameters needed to validate or create a digest. Properties: Implements = FunctionParameters.SetParameters ImplementsGuid = {0904F664-E2E0-4384-B994-B71255974B3A}
Validate (rawData : **Object**) : **Boolean** Public *Details:* This field represents the actual validation logic that a digest must provide. Properties: Implements = FunctionEval.Validate ImplementsGuid = {8FC53EE3-99AE-440a-820F-31AF8180E090}
ValidateDigest (rawData : **Object**) : **Boolean** Public *Details:* Properties: Implements = FunctionEval.ValidateDigest ImplementsGuid = {8FC53EE3-99AE-440a-820F-31AF8180E090}

A.1.5.8 FunctionEval

Interface in package '4.3.5 TrackableObjects'

Details: This interface illustrates the minimal functions that a Digest function must support.

CONNECTORS
Dependency Source -> Destination From: : DigestDescriptorDefinition : Class , Public To: FunctionEval : Interface , Public

OPERATIONS
CreateDigest (rawData : **Object** , parameters : **Object**) : **void** Public *Details:*
ValidateDigest (rawData : **Object**) : **Boolean** Public *Details:*

A.1.5.9 FunctionParameters

Interface in package '4.3.5 TrackableObjects'

Details: This is a generic interface that all Digest functions must implement. Actual field data will be dependent on the function that realizes this interfaces.

CONNECTORS
Dependency Source -> Destination From: : DigestDescriptorDefinition : Class , Public To: FunctionParameters : Interface , Public

ATTRIBUTES
Certificate : **X509Certificate** Public *Details:* This space is reserved for storing certificate data (if supported by the implementation function) Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
PrivateKey : **Bytes** Public *Details:*
PublicKey : **Bytes** Public *Details:*

OPERATIONS
SetParameters (Any : **Object**) : **void** Public *Details:*

A.1.5.10 DigestFunctionType

Enumeration in package '4.3.5 TrackableObjects'

Details: This field represents a subset a of digest or keyed-hashed authentication codes that can provide digital fingerprint services. The system should ideally provide run-time function of these functions to all participants so that all agents can validate data.

CONNECTORS
Dependency Source -> Destination From: : IGenericTrackable : Class , Public To: DigestFunctionType : Enumeration , Public

ENUMERATION:	
SHA256	
SHA3-256	
SHAKE128	
HMAC-SHA1	This represents a digest function, as defined by its standardized name. For additional references, consult RFC4635, FIPS 198.
HMAC-SHA256	
HMAC-MD5.SIG-ALG.REG.INT	

A.1.6 DigitalCertificates

This diagram contains assets that can be used to create a secure, digital representation of a subjects identity. This digital identity relies on the X.509 certificate model described by rfc5280.

The presented interface The presented interface allows static validation (by walking the certificate tree), and an online verification mechanism that checks for revoked certificates using Certificate Revocation Lists.

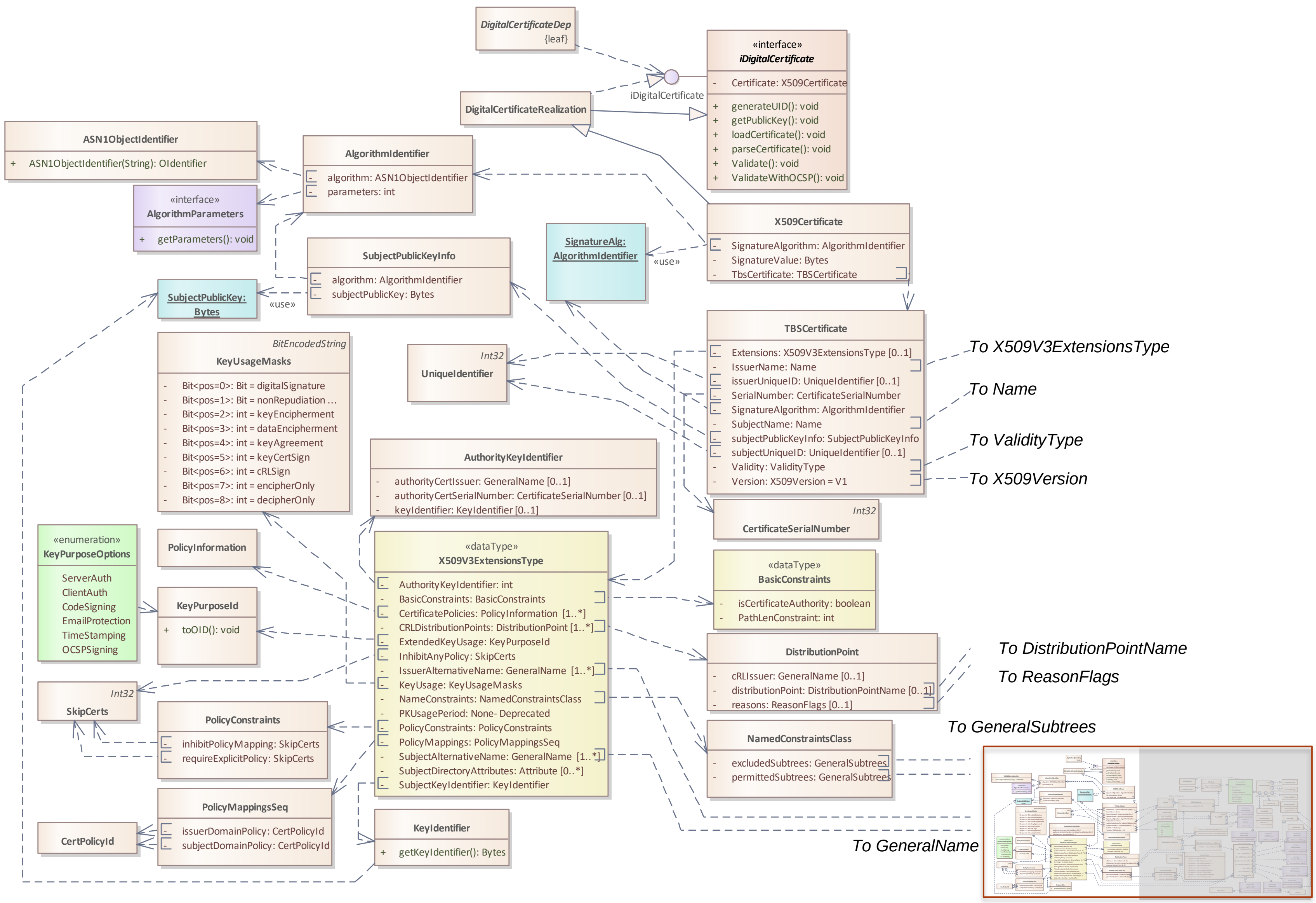


Figure 53. Overview of the DigitalCertificates' package components (Left side).

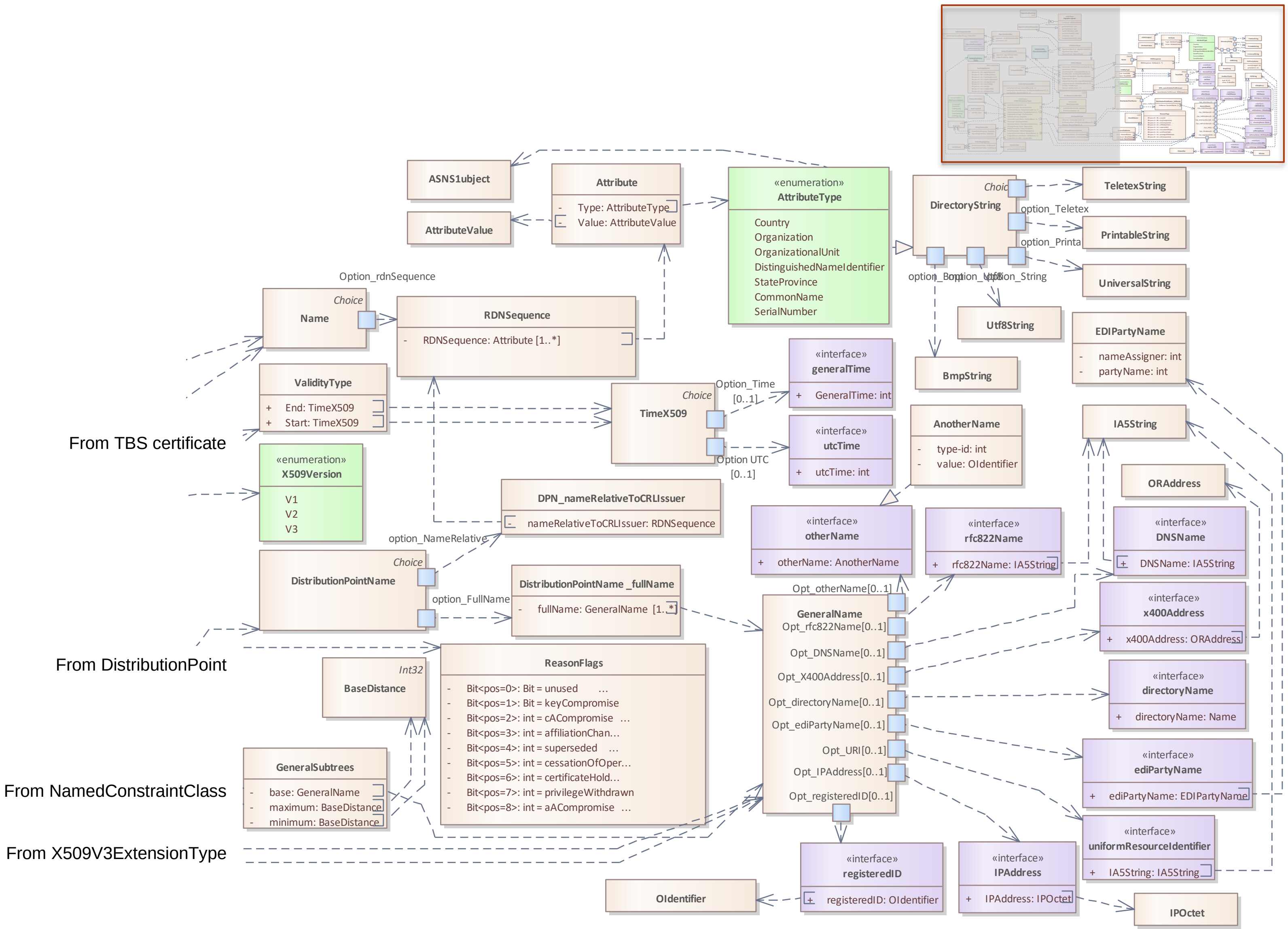


Figure 54. Overview of the DigitalCertificates' package components (Right side).

A.1.6.1 AlgorithmIdentifier

Class in package '4.3.6 DigitalCertificates'

Details: This object serves to encode an encryption algorithm according to rfc3279.

CONNECTORS
Dependency Source -> Destination From: : AlgorithmIdentifier : Class , Public To: AlgorithmParameters : Interface , Public
Dependency Source -> Destination From: : AlgorithmIdentifier : Class , Public To: ASN1ObjectIdentifier : Class , Public
Dependency Source -> Destination From: : X509Certificate : Class , Public To: AlgorithmIdentifier : Class , Public
Dependency Source -> Destination From: : SubjectPublicKeyInfo : Class , Public To: AlgorithmIdentifier : Class , Public

ATTRIBUTES
algorithm : **ASN1ObjectIdentifier** Private *Details:* This field is used to encode the type of signature algorithm used, common examples are: *sha224WithRSAEncryption* *sha256WithRSAEncryption* *sha384WithRSAEncryption* *sha512WithRSAEncryption* For more details read rfc4055. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
parameters : **int** Private *Details:* This field is used to provide additional parameter to the encryption algorithm. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.1.6.2 AnotherName

Class in package '4.3.6 DigitalCertificates'

Details: This is a specialization of a general name which can be be used to provide alternative names.

OUTGOING STRUCTURAL RELATIONSHIPS
Realization from AnotherName to otherName

ATTRIBUTES
type-id : **int** Private *Details:* This value is hard coded according to the OID. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
value : **OIdentifier** Private *Details:* This is the actual value of the field. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.1.6.3 ASN1ObjectIdentifier

Class in package '4.3.6 DigitalCertificates'

Details: This object describes represents an object on OID notation.

CONNECTORS
Dependency Source -> Destination From: : AlgorithmIdentifier : Class , Public To: ASN1ObjectIdentifier : Class , Public

OPERATIONS
ASN1ObjectIdentifier (identifier : **String**) : **OIdentifier** Public *Details:* This function maps a string into an Object Identifier (OID) details can be found in rfc3279. Examples: *id-ecdsa-with-shake128 OBJECT IDENTIFIER ::= { iso(1)* *identified-organization(3) dod(6) internet(1)* *security(5) mechanisms(5) pkix(7) algorithms(6)* *32 }* *id-ecdsa-with-shake256 OBJECT IDENTIFIER ::= { iso(1)* *identified-organization(3) dod(6) internet(1)* *security(5) mechanisms(5) pkix(7) algorithms(6)* *33 }*

A.1.6.4 ASNS1ubject

Class in package '4.3.6 DigitalCertificates'

Details: This represents typical information found on a Subject/Issuer RDNSequence.

CONNECTORS
Dependency Source -> Destination From: : AttributeType : Enumeration , Public To: ASNS1ubject : Class , Public

A.1.6.5 Attribute

Class in package '4.3.6 DigitalCertificates'

Details: This represents attributes expressed via OIDs. This object is often referenced as AttributeTypeValue.

Alias AttributeTypeValue

CONNECTORS
Dependency Source -> Destination From: : Attribute : Class , Public To: AttributeValue : Class , Public
Dependency Source -> Destination From: : Attribute : Class , Public To: AttributeType : Enumeration , Public
Dependency Source -> Destination From: : RDNSequence : Class , Public To: Attribute : Class , Public

ATTRIBUTES
Type : **AttributeType** Private *Details:* This represents the type of the attribute (standardized OID notation). Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Value : **AttributeValue** Private

ATTRIBUTES
Details: This represents the value within the attribute. The encoding will be subject to the OID rules. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.1.6.6 AttributeValue

Class in package '4.3.6 DigitalCertificates'

Details: This can be any value, as long as it fits the type defined by AttributeType

CONNECTORS
Dependency Source -> Destination From: : Attribute : Class , Public To: AttributeValue : Class , Public

A.1.6.7 AuthorityKeyIdentifier

Class in package '4.3.6 DigitalCertificates'

CONNECTORS
Dependency Source -> Destination From: : X509V3ExtensionsType : DataType , Public To: AuthorityKeyIdentifier : Class , Public

ATTRIBUTES
authorityCertIssuer : **GeneralName** Private *Details:* If this field is populated, the *AuthorityCertSerialNumber* should also be present Multiplicity: (0..1, Allow duplicates: 0, Is ordered: False)
authorityCertSerialNumber : **CertificateSerialNumber** Private *Details:* If this field is populated, the *AuthorityCertIssuer* should also be present Multiplicity: (0..1, Allow duplicates: 0, Is ordered: False)
keyIdentifier : **KeyIdentifier** Private *Details:* The value of the keyIdentifier field SHOULD be derived from the public key used to verify the certificate's signature or a cryptographic method. Multiplicity: (0..1, Allow duplicates: 0, Is ordered: False)

A.1.6.8 BaseDistance

Class in package '4.3.6 DigitalCertificates'

Details: Base distance is an integer of size 32.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from BaseDistance to Int32

CONNECTORS
Dependency Source -> Destination From: : GeneralSubtrees : Class , Public To: BaseDistance : Class , Public
Dependency Source -> Destination From: : GeneralSubtrees : Class , Public To: BaseDistance : Class , Public

A.1.6.9 BmpString

Class in package '4.3.6 DigitalCertificates'

Details: This represents a Unicode String encoded in a tag-Length-Value triplet.

CONNECTORS
Dependency Source -> Destination From: : option_Bmp : Port , Public To: BmpString : Class , Public

A.1.6.10 CertificateSerialNumber

Class in package '4.3.6 DigitalCertificates'

Details: This is a positive number usually encoded into an integer that represents the certificate serial number.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from CertificateSerialNumber to Int32

CONNECTORS
Dependency Source -> Destination From: : TBSCertificate : Class , Public To: CertificateSerialNumber : Class , Public

A.1.6.11 CertPolicyId

Class in package '4.3.6 DigitalCertificates'

Details: This is an OID-encoded policy that describes the certificate policies.

CONNECTORS
Dependency Source -> Destination From: : PolicyMappingsSeq : Class , Public To: CertPolicyId : Class , Public
Dependency Source -> Destination From: : PolicyMappingsSeq : Class , Public To: CertPolicyId : Class , Public

A.1.6.12 DigitalCertificateDep

Class in package '4.3.6 DigitalCertificates'

Details: Objects whom reference this class expect an object that realizes the *DigitalCertificate* Interface. This class is an abstract leaf and is only intended to serve as a data type reference.

CONNECTORS
Dependency Source -> Destination From: : DigitalCertificateDep : Class , Public To: iDigitalCertificate : ProvidedInterface , Public
Dependency Source -> Destination From: : AttestationCapabilities : Class , Public To: DigitalCertificateDep : Class , Public
Dependency Source -> Destination From: : GenericIdentity : Class , Public To: DigitalCertificateDep : Class , Public

A.1.6.13 DigitalCertificateRealization

Class in package '4.3.6 DigitalCertificates'

Details: This realization can be used to implement custom functions related to digital certificates (such as establishing, validating and revoking them).

OUTGOING STRUCTURAL RELATIONSHIPS
Realization from DigitalCertificateRealization to iDigitalCertificate
Generalization from DigitalCertificateRealization to «interface» iDigitalCertificate

A.1.6.14 DirectoryString

Class in package '4.3.6 DigitalCertificates'

Details: This class enables to represent a variety of strings in a machine-readable manner. Useful to encode descriptions, values or any other text-based data.

STRUCTURAL PART OF DirectoryString
option_Bmp : Port
option_Printable : Port
option_String : Port
option_Teletex : Port
option_Utf8 : Port

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from DirectoryString to Choice

A.1.6.15 DistributionPoint

Class in package '4.3.6 DigitalCertificates'

Details: This extension provides information about the Certificate Revocation List (CRL) locations. This object is not mandatory but its use is recommended.

CONNECTORS
Dependency Source -> Destination From: : DistributionPoint : Class , Public To: DistributionPointName : Class , Public
Dependency Source -> Destination From: : DistributionPoint : Class , Public To: ReasonFlags : Class , Public
Dependency Source -> Destination From: : X509V3ExtensionsType : DataType , Public To: DistributionPoint : Class , Public

ATTRIBUTES
cRLIssuer : **GeneralName** Private *Details:* This represents the entity that signs and issues the CRL Multiplicity: (0..1, Allow duplicates: 0, Is ordered: False)
distributionPoint : **DistributionPointName** Private *Details:* This field represents a sequence of general names, that can be used to retrieve a Certificate Revocation List, all distribution points must contain the same information. Multiplicity: (0..1, Allow duplicates: 0, Is ordered: False)
reasons : **ReasonFlags** Private *Details:* This field provides the reason for the certificate revocation. Multiplicity: (0..1, Allow duplicates: 0, Is ordered: False)

A.1.6.16 DistributionPointName

Class in package '4.3.6 DigitalCertificates'

Details: This structure represents a sequence of general names, that can be used to retrieve a Certificate Revocation List.

STRUCTURAL PART OF DistributionPointName
option_FullName : Port
option_NameRelative : Port

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from DistributionPointName to Choice

CONNECTORS
Dependency Source -> Destination From: : DistributionPoint : Class , Public To: DistributionPointName : Class , Public

A.1.6.17 DistributionPointName _fullName

Class in package '4.3.6 DigitalCertificates'

Details: This is used to provide a stand-alone reference to a the CLR distribution point.

Either this field or the *nameRelativeToCRLIssuer* field must be populated.

CONNECTORS
Dependency Source -> Destination From: : DistributionPointName _fullName : Class , Public To: GeneralName : Class , Public
Dependency Source -> Destination From: : option_FullName : Port , Public To: DistributionPointName _fullName : Class , Public

ATTRIBUTES
fullName : **GeneralName** Private *Details:*

A.1.6.18 DPN_nameRelativeToCRLIssuer

Class in package '4.3.6 DigitalCertificates'

Details: This is used to provide a reference to a the CLR distribution point, which is dependent on the CRLIssuer location.
Either this field or the *fullName* field must be populated.

CONNECTORS
Dependency Source -> Destination From: : DPN_nameRelativeToCRLIssuer : Class , Public To: RDNSequence : Class , Public
Dependency Source -> Destination From: : option_NameRelative : Port , Public To: DPN_nameRelativeToCRLIssuer : Class , Public

ATTRIBUTES
nameRelativeToCRLIssuer : **RDNSequence** Private *Details:*

A.1.6.19 EDIPartyName

Class in package '4.3.6 DigitalCertificates'

CONNECTORS
Dependency Source -> Destination From: : ediPartyName : Interface , Public To: EDIPartyName : Class , Public

ATTRIBUTES
nameAssigner : **int** Private *Details:*
partyName : **int** Private *Details:*

A.1.6.20 GeneralName

Class in package '4.3.6 DigitalCertificates'

Details: This represents a generic name structure. Specializations can be used to encode data according to the field required parameters.

STRUCTURAL PART OF GeneralName
Opt_directoryName : Port
Opt_DNSName : Port
Opt_ediPartyName : Port
Opt_IPAddress : Port
Opt_otherName : Port
Opt_registeredID : Port
Opt_rfc822Name : Port
Opt_URI : Port
Opt_X400Address : Port

CONNECTORS
Dependency Source -> Destination From: : X509V3ExtensionsType : DataType , Public To: GeneralName : Class , Public
Dependency Source -> Destination From: : X509V3ExtensionsType : DataType , Public To: GeneralName : Class , Public
Dependency Source -> Destination From: : DistributionPointName _fullName : Class , Public To: GeneralName : Class , Public
Dependency Source -> Destination From: : GeneralSubtrees : Class , Public To: GeneralName : Class , Public

A.1.6.21 GeneralSubtrees

Class in package '4.3.6 DigitalCertificates'

Details: This structure can be used to represent any subtree. A subtree such as orgXYZ.com allows to place subjects in these levels:
**.orgXYZ.com*
**.*.orgXYZ.com*
**.*.*orgXYZ.com*
...

CONNECTORS
Dependency Source -> Destination From: : GeneralSubtrees : Class , Public To: BaseDistance : Class , Public
Dependency Source -> Destination From: : GeneralSubtrees : Class , Public To: BaseDistance : Class , Public
Dependency Source -> Destination From: : GeneralSubtrees : Class , Public To: GeneralName : Class , Public
Dependency Source -> Destination From: : NamedConstraintsClass : Class , Public To: GeneralSubtrees : Class , Public

CONNECTORS
Dependency Source -> Destination From: : NamedConstraintsClass : Class , Public To: GeneralSubtrees : Class , Public

ATTRIBUTES
base : **GeneralName** Private *Details:* This field is used to represent the base tree address. In our example this can be *orgXYZ.com*. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
maximum : **BaseDistance** Private *Details:* This field set the minimum level of subdomains that must exist in a tree. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
minimum : **BaseDistance** Private *Details:* This field set the maximum level of subdomains that must exist in a tree. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.1.6.22 IA5String

Class in package '4.3.6 DigitalCertificates'

Details: This is a type of string that contains characters that can be encoded in a URL or a domain name.

CONNECTORS
Dependency Source -> Destination From: : uniformResourceIdentifier : Interface , Public To: IA5String : Class , Public
Dependency Source -> Destination From: : DNSName : Interface , Public To: IA5String : Class , Public
Dependency Source -> Destination From: : rfc822Name : Interface , Public To: IA5String : Class , Public

A.1.6.23 iDigitalCertificate

Class «interface» in package '4.3.6 DigitalCertificates'

STRUCTURAL PART OF iDigitalCertificate
iDigitalCertificate : ProvidedInterface

ATTRIBUTES
Certificate : **X509Certificate** Private *Details:*

OPERATIONS
generateUID () : **void** Public *Details:*
getPublicKey () : **void** Public *Details:*

OPERATIONS
loadCertificate () : **void** Public *Details:*
parseCertificate () : **void** Public *Details:*
Validate () : **void** Public *Details:* This function makes a static validation by walking the certificate chain until the CA is reached. This is done by continuously applying Public-Private key evaluations to ensure validity.
ValidateWithOCSP () : **void** Public *Details:* This function incorporates the Validate function and complements it with an online check to ensure that the certificate has not been revoked yet.

A.1.6.24 Instance:AlgorithmIdentifier

Entity in package '4.3.6 DigitalCertificates'

A.1.6.25 IPOctet

Class in package '4.3.6 DigitalCertificates'

Details: This represents an IP address using an *X* amount of bytes, depending on the protocol.

CONNECTORS
Dependency Source -> Destination From: : IPAddress : Interface , Public To: IPOctet : Class , Public

A.1.6.26 KeyIdentifier

Class in package '4.3.6 DigitalCertificates'

Details: This structure helps to identify the key pair that is applicable to this certificate in case the issuer has multiple public keys.

CONNECTORS
Dependency Source -> Destination From: : KeyIdentifier : Class , Public To: SubjectPublicKey:Bytes : Object , Public
Dependency Source -> Destination From: : X509V3ExtensionsType : DataType , Public To: KeyIdentifier : Class , Public

OPERATIONS
getKeyIdentifier () : **Bytes** Public *Details:* This represents a substring of the *subjectPublicKey*. In this model it is represented as a dynamic function, however in reality this will be a static value.

A.1.6.27 KeyPurposeId

Class in package '4.3.6 DigitalCertificates'

Details: This field can be used to limit the certificate applicability, for example, limiting its use to code signing, client identification, or time stamping. Fields should be encoded using OID.

CONNECTORS
Dependency Source -> Destination From: : KeyPurposeOptions : Enumeration , Public To: KeyPurposeId : Class , Public
Dependency Source -> Destination From: : X509V3ExtensionsType : DataType , Public To: KeyPurposeId : Class , Public

OPERATIONS
toOID () : **void** Public *Details:*

A.1.6.28 KeyUsageMasks

Class in package '4.3.6 DigitalCertificates'

Details: This structure defines the intended purpose/allowed usage for the current certificate.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from KeyUsageMasks to BitEncodedString

CONNECTORS
Dependency Source -> Destination From: : X509V3ExtensionsType : DataType , Public To: KeyUsageMasks : Class , Public

ATTRIBUTES
Bit<pos=0> : **Bit** Private = digitalSignature *Details:* This bitmask indicates that the certificate can be used to digitally sign data. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Bit<pos=1> : **Bit** Private = nonRepudiation *Details:* This bitmask indicates that the certificate can be used for content commitment (verifu digital signatures). Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Bit<pos=2> : **int** Private = keyEncipherment *Details:* This bitmask indicates that the certificate can be used enciphering private or secret keys. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Bit<pos=3> : **int** Private = dataEncipherment *Details:* This bitmask indicates that the certificate can be used to directly provide data encipherment without an intermediary symmetric key. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Bit<pos=4> : **int** Private = keyAgreement *Details:* This bitmask indicates that the subject's public can be used for key exchange. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Bit<pos=5> : **int** Private = keyCertSign *Details:* This bitmask indicates that the certificate can be used to verify other certificates. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Bit<pos=6> : **int** Private = cRLSign *Details:* This bitmask indicates that the certificate can be used to verify signatures from certificate revocation lists (CRL).

ATTRIBUTES
Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Bit<pos=7> : **int** Private = encipherOnly *Details:* This bitmask is undefined if keyAgreement is not set, else the data encipherment is only allowed during key agreement. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Bit<pos=8> : **int** Private = decipherOnly *Details:* This bitmask is undefined if keyAgreement is not set, else the data deciphering is only allowed during key agreement. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.1.6.29 Name

Class in package '4.3.6 DigitalCertificates'

Details: This is a choice-like object where the issuer identity is recorded.

STRUCTURAL PART OF Name
Option_rdnSequence : Port

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from Name to Choice

CONNECTORS
Dependency Source -> Destination From: : TBSCertificate : Class , Public To: Name : Class , Public
Dependency Source -> Destination From: : TBSCertificate : Class , Public To: Name : Class , Public

A.1.6.30 Name_RDNSequence

Class in package '4.3.6 DigitalCertificates'

Details: This represents a sequence of RelativeDistinguishedName, a sequence of properties. In this case it contains a sequence of properties typically observed in an issuer field.

Alias RelativeDistinguishedName

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from Name_RDNSequence to RDNSequence

CONNECTORS
Dependency Source -> Destination From: : Option_rdnSequence : Port , Public To: Name_RDNSequence : Class , Public

ATTRIBUTES
CommonName : **Attribute** Private *Details:* This is the common name that identifies a resource. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Country : **Attribute** Private

ATTRIBUTES
Details: Documents the country, using an ISO standardized 2 letter code Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
DistinguishedNameQualifier : **Attribute** Private *Details:*
Organization : **Attribute** Private *Details:*
OrganizationalUnit : **Attribute** Private *Details:*
SerialNumber : **Attribute** Private *Details:*
StateProvinceName : **Attribute** Private *Details:*

A.1.6.31 NamedConstraintsClass

Class in package '4.3.6 DigitalCertificates'

Details: This class is used to define a set of black-listed and white-listed naming schemes. These are represented using trees that can define multiple paths. Black-listed trees take precedence over white-listed trees.

CONNECTORS
Dependency Source -> Destination From: : NamedConstraintsClass : Class , Public To: GeneralSubtrees : Class , Public
Dependency Source -> Destination From: : NamedConstraintsClass : Class , Public To: GeneralSubtrees : Class , Public
Dependency Source -> Destination From: : X509V3ExtensionsType : DataType , Public To: NamedConstraintsClass : Class , Public

ATTRIBUTES
excludedSubtrees : **GeneralSubtrees** Private *Details:* These represents the tree structures in which the CA is not allowed to place subjects. E.g. a CA may be prohibited from signing subjects under the domain *.com or *.energy.com. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
permittedSubtrees : **GeneralSubtrees** Private *Details:* These represents the tree structures in which the CA is permitted to place subjects. E.g. a CA may place subjects under its own organization *.orgABC.com. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.1.6.32 OIdentifier

Class in package '4.3.6 DigitalCertificates'

Details: This represents an object Identified as standardized by ITU, ISO/IEC.
Example of valid OIDs are:

```
US government:
{joint-iso-itu-t(2) country(16) us(840) organization(1) gov(101)}
Linux syslog:
{iso(1) identified-organization(3) dod(6) internet(1) private(4) enterprise(1) 37476
products(2) oidplus(5) v2(2) plugins(4) logger(7) linux-syslog(100)}
Pacific Northwest National Laboratory:
{iso(1) identified-organization(3) dod(6) internet(1) private(4) enterprise(1) 2325}
```

CONNECTORS
Dependency Source -> Destination From: : registeredID : Interface , Public To: OIdentifier : Class , Public

A.1.6.33 ORAddress

Class in package '4.3.6 DigitalCertificates'

Details: An originator/recipient address within a domain name. Extensive used on email addresses.

CONNECTORS
Dependency Source -> Destination From: : x400Address : Interface , Public To: ORAddress : Class , Public

A.1.6.34 PolicyConstraints

Class in package '4.3.6 DigitalCertificates'

Details: This field is used to encode the start and end depth for which the *policyMapping* attributes can be copied.To remain compliant either the *inhibitPolicyMapping* field or the *requireExplicitPolicy* field MUST be present

CONNECTORS
Dependency Source -> Destination From: : PolicyConstraints : Class , Public To: SkipCerts : Class , Public
Dependency Source -> Destination From: : PolicyConstraints : Class , Public To: SkipCerts : Class , Public
Dependency Source -> Destination From: : X509V3ExtensionsType : DataType , Public To: PolicyConstraints : Class , Public

ATTRIBUTES
inhibitPolicyMapping : **SkipCerts** Private *Details:* This is the maximum number of chained certificates that can use the *policyMappings* information. After this number is reached policyMappings cannot longer be replicated. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
requireExplicitPolicy : **SkipCerts** Private *Details:* This is the minimum number of chained certificates after which the the *policyMappings* information can be used. Before this number is reached policyMappings cannot be replicated. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.1.6.35 PolicyInformation

Class in package '4.3.6 DigitalCertificates'

Details: This object is used to describe the allowed uses of the certificate. Due to its complexity, the attributes of this object have not been defined. Consult RFC 5280 for more details.

CONNECTORS
Dependency Source -> Destination From: : X509V3ExtensionsType : DataType , Public To: PolicyInformation : Class , Public

A.1.6.36 PolicyMappingsSeq

Class in package '4.3.6 DigitalCertificates'

Details: This object maps issuer-domain policies to the subject-domain space. Useful when applications accept inherited permissions.

CONNECTORS
Dependency Source -> Destination From: : PolicyMappingsSeq : Class , Public To: CertPolicyId : Class , Public
Dependency Source -> Destination From: : PolicyMappingsSeq : Class , Public To: CertPolicyId : Class , Public
Dependency Source -> Destination From: : X509V3ExtensionsType : DataType , Public To: PolicyMappingsSeq : Class , Public

ATTRIBUTES
issuerDomainPolicy : **CertPolicyId** Private *Details:* This represents the fields that the issuer wants to map into the subject policies. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
subjectDomainPolicy : **CertPolicyId** Private *Details:* This represents the fields that the subject will present as its policies. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.1.6.37 PrintableString

Class in package '4.3.6 DigitalCertificates'

Details: This represents a string with a limited characther set that was typical of mainframe computers. E.g.,

```
A-Z
a-z
0-9
' ( ) + , - . / : = ? [space]
```

CONNECTORS
Dependency Source -> Destination From: : option_Printable : Port , Public To: PrintableString : Class , Public

A.1.6.38 RDNSequence

Class in package '4.3.6 DigitalCertificates'

Details: This is a generic sequence of attributes that can be used to describe a subject.

CONNECTORS
Dependency Source -> Destination From: : RDNSequence : Class , Public To: Attribute : Class , Public
Dependency Source -> Destination From: : DPN_nameRelativeToCRLIssuer : Class , Public To: RDNSequence : Class , Public
Dependency Source -> Destination From: : Option_rdnSequence : Port , Public To: RDNSequence : Class , Public

ATTRIBUTES
RDNSequence : **Attribute** Private *Details:*

A.1.6.39 ReasonFlags

Class in package '4.3.6 DigitalCertificates'

Details: This provides additional details that led to the certificate being revoked. This is a bitmask that can be used to select multiple option as the same time.

CONNECTORS
Dependency Source -> Destination From: : DistributionPoint : Class , Public To: ReasonFlags : Class , Public

ATTRIBUTES
Bit<pos=0> : **Bit** Private = unused *Details:*
Bit<pos=1> : **Bit** Private = keyCompromise *Details:* The subject's key was compromised. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Bit<pos=2> : **int** Private = cACompromise *Details:* The CA root key was compromised. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Bit<pos=3> : **int** Private = affiliationChanged *Details:* The subject has no longer an affiliation with the reported entities (e.g. country, division) Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Bit<pos=4> : **int** Private = superseded *Details:* A new certificate that replaces this one has been issued. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Bit<pos=5> : **int** Private = cessationOfOperation *Details:* The root CA certificate has been revoked Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Bit<pos=6> : **int** Private = certificateHold *Details:* A temporal hold has been placed on this certificate. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Bit<pos=7> : **int** Private = privilegeWithdrawn *Details:* The priviledges listed in the certificate do not longer hold true and a revocation is requested. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Bit<pos=8> : **int** Private = aACompromise *Details:* This is used to indicate that attribute's aspects have been compromised. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.1.6.40 SkipCerts

Class in package '4.3.6 DigitalCertificates'

Details: This object dictates the number of hops for which constraints are applicable to this certificate.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from SkipCerts to Int32

CONNECTORS
Dependency Source -> Destination From: : PolicyConstraints : Class , Public To: SkipCerts : Class , Public
Dependency Source -> Destination From: : X509V3ExtensionsType : DataType , Public To: SkipCerts : Class , Public
Dependency Source -> Destination From: : PolicyConstraints : Class , Public To: SkipCerts : Class , Public

A.1.6.41 SubjectPublicKeyInfo

Class in package '4.3.6 DigitalCertificates'

Details: This represents the subject's public key algorithm details

CONNECTORS
Usage Source -> Destination From: : SubjectPublicKeyInfo : Class , Public To: SubjectPublicKey:Bytes : Object , Public
Dependency Source -> Destination From: : SubjectPublicKeyInfo : Class , Public To: AlgorithmIdentifier : Class , Public
Dependency Source -> Destination From: : TBSCertificate : Class , Public To: SubjectPublicKeyInfo : Class , Public

ATTRIBUTES
algorithm : **AlgorithmIdentifier** Private *Details:* This field holds the algorithm and parameters used to encode the subject's public key Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
subjectPublicKey : **Bytes** Private *Details:* This field contains the actual public key in raw bytes. The subject keeps its private key in a secure location. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.1.6.42 TBSCertificate

Class in package '4.3.6 DigitalCertificates'

Details: This represents the main structure present on all X509 certificates. Users must implement at least two functions:
Validate: To ensure that a certificate can be traced cryptographically to the source CA.
ValidateWithOCSP: To ensure that the certificate has not been revoked.

CONNECTORS
Dependency Source -> Destination From: : TBSCertificate : Class , Public To: Name : Class , Public
Dependency Source -> Destination From: : TBSCertificate : Class , Public To: Name : Class , Public
Dependency Source -> Destination From: : TBSCertificate : Class , Public To: UniqueIdentifier : Class , Public
Dependency Source -> Destination From: : TBSCertificate : Class , Public To: CertificateSerialNumber : Class , Public
Dependency Source -> Destination From: : TBSCertificate : Class , Public To: UniqueIdentifier : Class , Public
Dependency Source -> Destination From: : TBSCertificate : Class , Public To: SubjectPublicKeyInfo : Class , Public
Dependency Source -> Destination From: : TBSCertificate : Class , Public To: ValidityType : Class , Public
Dependency Source -> Destination From: : TBSCertificate : Class , Public To: X509Version : Enumeration , Public
Dependency Source -> Destination From: : TBSCertificate : Class , Public To: SignatureAlg:AlgorithmIdentifier : Object , Public
Dependency Source -> Destination From: : TBSCertificate : Class , Public To: X509V3ExtensionsType : DataType , Public
Dependency Source -> Destination From: : X509Certificate : Class , Public To: TBSCertificate : Class , Public

ATTRIBUTES
Extensions : **X509V3ExtensionsType** Private *Details:*
IssuerName : **Name** Private *Details:* This field identifies the entity that has created this certificate. This represents a sequence of RelativeDistinguishedName, a sequence of properties. In this case it contains a sequence of properties typically observed in an issuer/subject field. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
issuerUniqueID : **UniqueIdentifier** Private *Details:*
SerialNumber : **CertificateSerialNumber** Private *Details:* This is a unique serial number that uniquely identifies a subject to the CA. Limited to 20 bytes. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
SignatureAlgorithm : **AlgorithmIdentifier** Private *Details:* This field is a repetition of the information provided in the header of an X509 certificate. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
SubjectName : **Name** Private *Details:*
subjectPublicKeyInfo : **SubjectPublicKeyInfo** Private *Details:* This field stores the subject's public key for which this certificate describes. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
subjectUniqueID : **UniqueIdentifier** Private *Details:*
Validity : **ValidityType** Private

ATTRIBUTES
Details:
Version : **X509Version** Private = V1 *Details:* This field is used to encode the version of the encoded certificate Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.1.6.43 TeletexString

Class in package '4.3.6 DigitalCertificates'

Details: This is a string that precedes the UTF8 standard that allows to encode character data using 1 or 2 bytes.

CONNECTORS
Dependency Source -> Destination From: : option_Teletex : Port , Public To: TeletexString : Class , Public

A.1.6.44 TimeX509

Class in package '4.3.6 DigitalCertificates'

Details: This object represents the choice of time reference in creating the certificate. Certificate validity tests should account for this choice.

STRUCTURAL PART OF TimeX509
Option UTC : Port
Option_Time : Port

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from TimeX509 to Choice

CONNECTORS
Dependency Source -> Destination From: : ValidityType : Class , Public To: TimeX509 : Class , Public
Dependency Source -> Destination From: : DateTimeBound : Class , Public To: TimeX509 : Class , Public
Dependency Source -> Destination From: : ValidityType : Class , Public To: TimeX509 : Class , Public
Dependency Source -> Destination From: : DateTimeBound : Class , Public To: TimeX509 : Class , Public

A.1.6.45 UniqueIdentifier

Class in package '4.3.6 DigitalCertificates'

Details: This object can be used to uniquely identify the subject using a numerical notation. This field can be useful in tying a physical identity to a digital identity.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from UniqueIdentifier to Int32

CONNECTORS
Dependency Source -> Destination From: : TBSCertificate : Class , Public To: UniqueIdentifier : Class , Public
Dependency Source -> Destination From: : TBSCertificate : Class , Public To: UniqueIdentifier : Class , Public

A.1.6.46 UniversalString

Class in package '4.3.6 DigitalCertificates'

Details: This repesents an string encode using the ISO/IEC 10646 rules.

CONNECTORS
Dependency Source -> Destination From: : option_String : Port , Public To: UniversalString : Class , Public

A.1.6.47 Utf8String

Class in package '4.3.6 DigitalCertificates'

Details: This represents a UTF-encoded string.

CONNECTORS
Dependency Source -> Destination From: : option_Utf8 : Port , Public To: Utf8String : Class , Public

A.1.6.48 ValidityType

Class in package '4.3.6 DigitalCertificates'

Details: This structure describes the validity period through which the certificate is considered valid, unless revoked by a CRL.

CONNECTORS
Dependency Source -> Destination From: : ValidityType : Class , Public To: TimeX509 : Class , Public
Dependency Source -> Destination From: : ValidityType : Class , Public To: TimeX509 : Class , Public

CONNECTORS
Dependency Source -> Destination From: : TBSCertificate : Class , Public To: ValidityType : Class , Public

ATTRIBUTES
End : **TimeX509** Public *Details:*
Start : **TimeX509** Public *Details:*

A.1.6.49 X509Certificate

Class in package '4.3.6 DigitalCertificates'

Details: This represents the top-level container of a X509 certificate.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from X509Certificate to DigitalCertificateRealization

CONNECTORS
Dependency Source -> Destination From: : X509Certificate : Class , Public To: TBSCertificate : Class , Public
Dependency Source -> Destination From: : X509Certificate : Class , Public To: AlgorithmIdentifier : Class , Public
Usage Source -> Destination From: : X509Certificate : Class , Public To: SignatureAlg:AlgorithmIdentifier : Object , Public

ATTRIBUTES
SignatureAlgorithm : **AlgorithmIdentifier** Private *Details:* This field encodes the algorithm and parameters used by the CA to sign the certificate. Although V1 is used by default, modern implementations rely on V3. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
SignatureValue : **Bytes** Private *Details:* This field contains the actual signature in raw bytes. The validity of the signature can be accessed base on the *signatureAlgorithm* and the *TBSCertificate* contents. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
TbsCertificate : **TBSCertificate** Private *Details:*

A.1.6.50 AlgorithmParameters

Interface in package '4.3.6 DigitalCertificates'

Details: This is a structure that represents an algorithm's parameters. This is encoded during the certificate creation time and its contents are dependent on the chosen algorithm.

CONNECTORS
Dependency Source -> Destination From: : AlgorithmIdentifier : Class , Public To: AlgorithmParameters : Interface , Public

OPERATIONS
getParameters () : **void** Public *Details:*

A.1.6.51 directoryName

Interface in package '4.3.6 DigitalCertificates'

Details: This represents a valid string that can be used to encode a directory within a server.

CONNECTORS
Dependency Source -> Destination From: : Opt_directoryName : Port , Public To: directoryName : Interface , Public

ATTRIBUTES
directoryName : **Name** Public *Details:*

A.1.6.52 DNSName

Interface in package '4.3.6 DigitalCertificates'

Details: This is a string that can encode any Dynamic Name Server address.

CONNECTORS
Dependency Source -> Destination From: : DNSName : Interface , Public To: IA5String : Class , Public
Dependency Source -> Destination From: : Opt_DNSName : Port , Public To: DNSName : Interface , Public

ATTRIBUTES
DNSName : **IA5String** Public *Details:*

A.1.6.53 ediPartyName

Interface in package '4.3.6 DigitalCertificates'

Details: This represents a string that can encode an Electronic Data Interchange entity. See RFC 5280 for more details.

CONNECTORS
Dependency Source -> Destination From: : ediPartyName : Interface , Public To: EDIPartyName : Class , Public
Dependency Source -> Destination From: : Opt_ediPartyName : Port , Public To: ediPartyName : Interface , Public

ATTRIBUTES
ediPartyName : **EDIPartyName** Public *Details:*

A.1.6.54 generalTime

Interface in package '4.3.6 DigitalCertificates'

Details: This interface is used to encode/decode time data using the time zone offsets. YYYYMMDDHHMMSSZ.

CONNECTORS
Dependency Source -> Destination From: : Option_Time : Port , Public To: generalTime : Interface , Public

ATTRIBUTES
GeneralTime : **int** Public *Details:* Time encoded in local time. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.1.6.55 IPAddress

Interface in package '4.3.6 DigitalCertificates'

Details: This string can encode a IPv6 or IPv4 address using octets.

CONNECTORS
Dependency Source -> Destination From: : IPAddress : Interface , Public To: IPOctet : Class , Public
Dependency Source -> Destination From: : Opt_IPAddress : Port , Public To: IPAddress : Interface , Public

ATTRIBUTES
IPAddress : **IPOctet** Public *Details:*

A.1.6.56 otherName

Interface in package '4.3.6 DigitalCertificates'

CONNECTORS
Dependency Source -> Destination From: : Opt_otherName : Port , Public To: otherName : Interface , Public

ATTRIBUTES
otherName : **AnotherName** Public *Details:*

A.1.6.57 registeredID

Interface in package '4.3.6 DigitalCertificates'

Details: This represents an name which is already contained in the OID database.

CONNECTORS
Dependency Source -> Destination From: : registeredID : Interface , Public To: OIdentifier : Class , Public
Dependency Source -> Destination From: : Opt_registeredID : Port , Public To: registeredID : Interface , Public

ATTRIBUTES
registeredID : **OIdentifier** Public *Details:*

A.1.6.58 rfc822Name

Interface in package '4.3.6 DigitalCertificates'

CONNECTORS
Dependency Source -> Destination From: : rfc822Name : Interface , Public To: IA5String : Class , Public
Dependency Source -> Destination From: : Opt_rfc822Name : Port , Public To: rfc822Name : Interface , Public

ATTRIBUTES
rfc822Name : **IA5String** Public *Details:*

A.1.6.59 uniformResourceIdentifier

Interface in package '4.3.6 DigitalCertificates'

Details: This string can encode a Uniform Resource Identifier, comparable to an WWW address.

CONNECTORS
Dependency Source -> Destination From: : uniformResourceIdentifier : Interface , Public To: IA5String : Class , Public
Dependency Source -> Destination From: : Opt_URI : Port , Public To: uniformResourceIdentifier : Interface , Public

ATTRIBUTES
IA5String : **IA5String** Public *Details:*

A.1.6.60 utcTime

Interface in package '4.3.6 DigitalCertificates'

Details: This interface is used to encode/decode time data using the Coordinated Universal Time reference.

CONNECTORS
Dependency Source -> Destination From: : Option UTC : Port , Public To: utcTime : Interface , Public

ATTRIBUTES
utcTime : **int** Public *Details:* Time encoded in UTC time. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.1.6.61 x400Address

Interface in package '4.3.6 DigitalCertificates'

Details: This is a string that can encode any email address.

CONNECTORS
Dependency Source -> Destination From: : x400Address : Interface , Public To: ORAddress : Class , Public
Dependency Source -> Destination From: : Opt_X400Address : Port , Public To: x400Address : Interface , Public

ATTRIBUTES
x400Address : **ORAddress** Public *Details:*

A.1.6.62 SignatureAlg:AlgorithmIdentifier

Object in package '4.3.6 DigitalCertificates'

Details: This is an specific instance of the AlgorithmIdentifier. It is used to symbolize that both the header and the TBSCertificate SignatureAlgorithm contain the same data.

CONNECTORS
Usage Source -> Destination From: : X509Certificate : Class , Public To: SignatureAlg:AlgorithmIdentifier : Object , Public
Dependency Source -> Destination From: : TBSCertificate : Class , Public To: SignatureAlg:AlgorithmIdentifier : Object , Public

A.1.6.63 SubjectPublicKey:Bytes

Object in package '4.3.6 DigitalCertificates'

Details: This is an instance of the subject's public key

CONNECTORS
Usage Source -> Destination From: : SubjectPublicKeyInfo : Class , Public To: SubjectPublicKey:Bytes : Object , Public
Dependency Source -> Destination From: : KeyIdentifier : Class , Public To: SubjectPublicKey:Bytes : Object , Public

A.1.6.64 AttributeType

Enumeration in package '4.3.6 DigitalCertificates'

Details: These enumeration represents some of the common attributes present in certificates, these must be encoded in OID format.

OUTGOING STRUCTURAL RELATIONSHIPS
Realization from AttributeType to DirectoryString

CONNECTORS
Dependency Source -> Destination From: : AttributeType : Enumeration , Public To: ASNS1ubject : Class , Public
Dependency Source -> Destination From: : Attribute : Class , Public To: AttributeType : Enumeration , Public

ENUMERATION:
Country
Organization
OrganizationalUnit
DistinguishedNameIdentifier
StateProvince
CommonName
SerialNumber

A.1.6.65 BasicConstraints

DataType in package '4.3.6 DigitalCertificates'

Details: This object is used to determine the maximum valid certification depth and to indicate if the certificate corresponds to a CA.

CONNECTORS
Dependency Source -> Destination From: : X509V3ExtensionsType : DataType , Public To: BasicConstraints : DataType , Public

ATTRIBUTES
isCertificateAuthority : **boolean** Private *Details:* THis is a boolean flag used to identify if the certificate corresponds to a CA. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
PathLenConstraint : **int** Private

ATTRIBUTES
Details: This field determines the maximum number of certificates that can be walked during validation. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.1.6.66 KeyPurposeOptions

Enumeration in package '4.3.6 DigitalCertificates'

Details: This enumeration represents the allowed key uses. This object is descriptive, actual implementation use OID-encoded representations.

CONNECTORS
Dependency Source -> Destination From: : KeyPurposeOptions : Enumeration , Public To: KeyPurposeId : Class , Public

ENUMERATION:	
ServerAuth	This certificate can be used for HTTPs server authentication.
ClientAuth	This certificate can be used for HTTPs client authentication.
CodeSigning	This certificate can be used to sign executable code.
EmailProtection	This certificate can be used to provide email protection services.
TimeStamping	This certificate can be used to time stamp objects.
OCSPSigning	This certificate can be used to provide/create Online Certificate Status Protocol (OCSP).

A.1.6.67 Property1

Property in package '4.3.6 DigitalCertificates'

A.1.6.68 X509V3ExtensionsType

DataType in package '4.3.6 DigitalCertificates'

CONNECTORS
Dependency Source -> Destination From: : X509V3ExtensionsType : DataType , Public To: GeneralName : Class , Public
Dependency Source -> Destination From: : X509V3ExtensionsType : DataType , Public To: GeneralName : Class , Public
Dependency Source -> Destination From: : X509V3ExtensionsType : DataType , Public To: KeyIdentifier : Class , Public
Dependency Source -> Destination From: : X509V3ExtensionsType : DataType , Public To: AuthorityKeyIdentifier : Class , Public
Dependency Source -> Destination From: : X509V3ExtensionsType : DataType , Public To: NamedConstraintsClass : Class , Public

CONNECTORS
Dependency Source -> Destination From: : X509V3ExtensionsType : DataType , Public To: PolicyConstraints : Class , Public
Dependency Source -> Destination From: : X509V3ExtensionsType : DataType , Public To: DistributionPoint : Class , Public
Dependency Source -> Destination From: : X509V3ExtensionsType : DataType , Public To: SkipCerts : Class , Public
Dependency Source -> Destination From: : X509V3ExtensionsType : DataType , Public To: BasicConstraints : DataType , Public
Dependency Source -> Destination From: : X509V3ExtensionsType : DataType , Public To: PolicyInformation : Class , Public
Dependency Source -> Destination From: : X509V3ExtensionsType : DataType , Public To: PolicyMappingsSeq : Class , Public
Dependency Source -> Destination From: : X509V3ExtensionsType : DataType , Public To: KeyUsageMasks : Class , Public
Dependency Source -> Destination From: : X509V3ExtensionsType : DataType , Public To: KeyPurposeId : Class , Public
Dependency Source -> Destination From: : TBSCertificate : Class , Public To: X509V3ExtensionsType : DataType , Public

ATTRIBUTES
AuthorityKeyIdentifier : **int** Private *Details:* This field enables to specify the public key that must be used to verify this certificate, useful when the issuer has multiple identities/key pairs. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
BasicConstraints : **BasicConstraints** Private *Details:* This field contains a flag to determine if the public key is a root CA (e.g. it can be used to validate signed certificates). It also sets the maximum path walk in a chain of verification structure. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
CertificatePolicies : **PolicyInformation** Private *Details:* This field can be used to set the optional policy qualifiers, these are used mostly in CA certificates to limit the types of certificates that can be signed. Multiplicity: (1..*, Allow duplicates: 0, Is ordered: False)
CRLDistributionPoints : **DistributionPoint** Private *Details:* This field describes the mechanisms that must be used to access the Certificate Revocation Lists (CRLs). CRLs must be periodically checked to ensure that the provided credentials have not been compromised (i.e. private keys were stolen). Multiplicity: (1..*, Allow duplicates: 0, Is ordered: False)
ExtendedKeyUsage : **KeyPurposeId** Private *Details:* This field indicates additional certificate's purposes. This option is intended for end-user certificates, where no further certificate signing is expected to occur. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
InhibitAnyPolicy : **SkipCerts** Private *Details:* This is used to override the effects of anyPolicy extension on root certificates. If this flag is set then a blanket "Allow all" can only be used by intermediary certificates. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
IssuerAlternativeName : **GeneralName** Private *Details:* This enables to provide multiple names for the issuer's name. Constraints are not enforced in these names Multiplicity: (1..*, Allow duplicates: 0, Is ordered: False)
KeyUsage : **KeyUsageMasks** Private *Details:* This field defines the allowed uses of this certificate.

ATTRIBUTES
Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
NameConstraints : **NamedConstraintsClass** Private *Details:* This field is only used in CA certificates and allows to define the paths or naming schemes for which this certificate can be used to create subjects' certificates. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
PKUsagePeriod : **None- Deprecated** Private *Details:* This field is deprecated Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
PolicyConstraints : **PolicyConstraints** Private *Details:* This field is reserved for CA certificate use. It lists the policy mappings that subjects are allowed to use Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
PolicyMappings : **PolicyMappingsSeq** Private *Details:* This field enables to map issuer properties to the subject. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
SubjectAlternativeName : **GeneralName** Private *Details:* This field enables to provide multiple naming schemes to identify the subject. This may include additional IPs, multiple DNS mails, email addresses. Multiplicity: (1..*, Allow duplicates: 0, Is ordered: False)
SubjectDirectoryAttributes : **Attribute** Private *Details:* This extension enables to provide additional identification attributes to the subject such as country, organization Multiplicity: (0..*, Allow duplicates: 0, Is ordered: False)
SubjectKeyIdentifier : **KeyIdentifier** Private *Details:*

A.1.6.69 X509Version

Enumeration in package '4.3.6 DigitalCertificates'

Details: This enumeration represent the three versions currently in use by the X509 reference implementation.

CONNECTORS
Dependency Source -> Destination From: : TBSCertificate : Class , Public To: X509Version : Enumeration , Public

ENUMERATION:
V1
V2
V3

A.1.7 BlockchainLedger

This diagram provides the basic representation of a blockchain environment. It is intended to serve as a reference for software engineers implementing this technology. Many of the blocks presented in this diagram must be overridden, extended or otherwise adjusted to correctly represent a blockchain environment. Nevertheless, the basic read/write functionalities, and an immutable ledger must continue to hold.

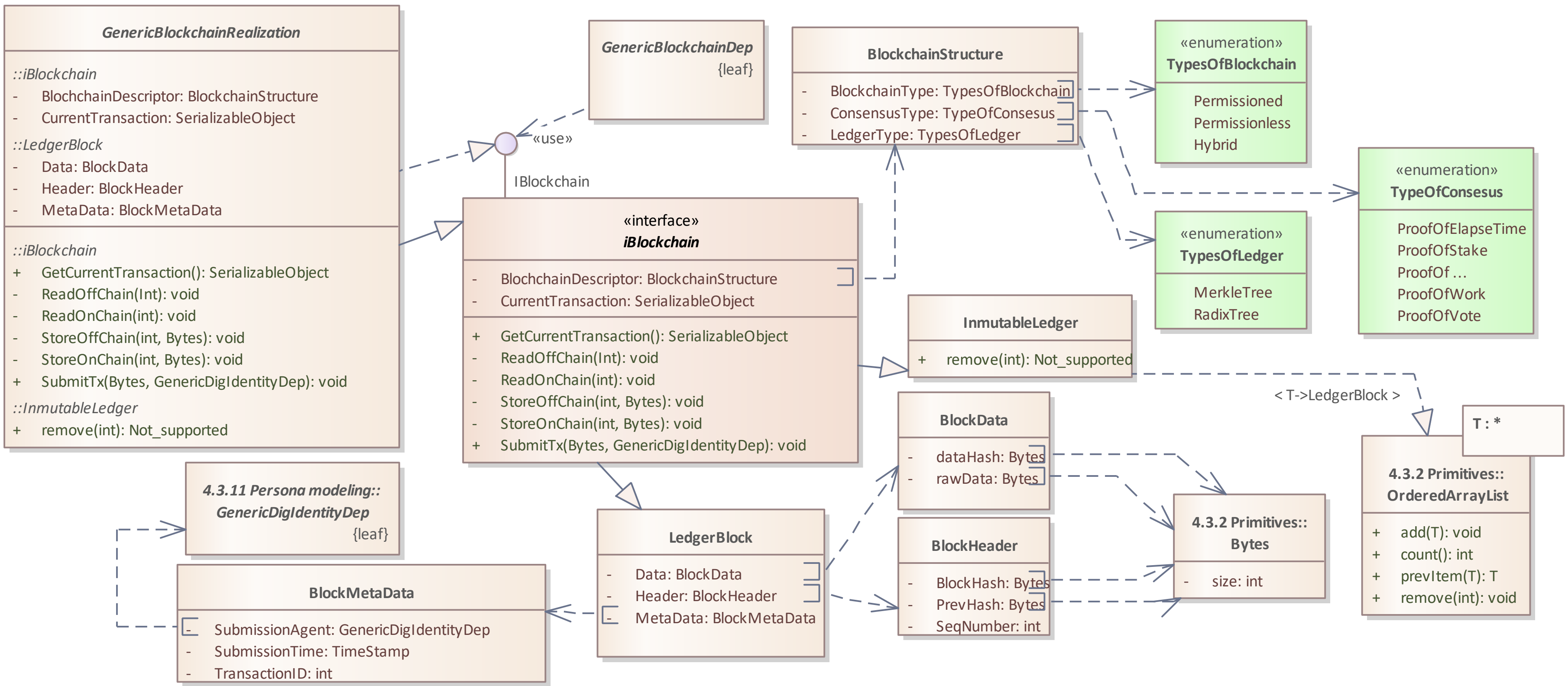


Figure 55. Overview of the BlockchainLedger's package components

A.1.7.1 BlockchainStructure

Class in package '4.3.7 BlockchainLedger'

Details: This structure is used to describe the underlying blockchain characteristics such as the ledger type consensus type, etc.

CONNECTORS
Dependency Source -> Destination From: : BlockchainStructure : Class , Public To: TypesOfLedger : Enumeration , Public
Dependency Source -> Destination From: : BlockchainStructure : Class , Public To: TypesOfBlockchain : Enumeration , Public
Dependency Source -> Destination From: : BlockchainStructure : Class , Public To: TypeOfConsesus : Enumeration , Public
Dependency Source -> Destination From: : iBlockchain : Class , Public To: BlockchainStructure : Class , Public

ATTRIBUTES
BlockchainType : **TypesOfBlockchain** Private *Details:* This entry represent the type of blockchain used in the implementation. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
ConsensusType : **TypeOfConsesus** Private *Details:* This field defines the underlying consensus method. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
LedgerType : **TypesOfLedger** Private *Details:* This field describes the ledger type used in the blockchain implementation. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.1.7.2 BlockData

Class in package '4.3.7 BlockchainLedger'

Details: This structure holds both the raw data (from the transaction) and its corresponding hash.

CONNECTORS
Dependency Source -> Destination From: : BlockData : Class , Public To: Bytes : Class , Public
Dependency Source -> Destination From: : BlockData : Class , Public To: Bytes : Class , Public
Dependency Source -> Destination From: : LedgerBlock : Class , Public To: BlockData : Class , Public

ATTRIBUTES
dataHash : **Bytes** Private *Details:* This is the hash corresponding to the underlying data. Note that the hashing/digest function is abstracted but should be specified by the documentation to ensure agents can perform external verifications Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
rawData : **Bytes** Private

ATTRIBUTES
Details: This is the raw object. For interoperability reasons, objects should be encoded into a universal format that is understood by all parties. Examples include serialization formats such as JSON, protobuf, MessagePack. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.1.7.3 BlockHeader

Class in package '4.3.7 BlockchainLedger'

Details: This object stores information about the linkage of a block, this may include pointer to a parent,or past blocks depending on the blockchain/ledger implementation.

CONNECTORS
Dependency Source -> Destination From: : BlockHeader : Class , Public To: Bytes : Class , Public
Dependency Source -> Destination From: : BlockHeader : Class , Public To: Bytes : Class , Public
Dependency Source -> Destination From: : LedgerBlock : Class , Public To: BlockHeader : Class , Public

ATTRIBUTES
BlockHash : **Bytes** Private *Details:* This represents the fingerprint for the entire block. This will usually contains all the blocks, pointers and transaction information that lead to the stored state. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
PrevHash : **Bytes** Private *Details:* This provides a copy of digest to the previous block. Agents can walk back into the genesis blocks using these digests to validate the integrity of the ledger. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
SeqNumber : **int** Private *Details:* This can be used to uniquely identify a block within a blockchain. This may be sequential or not, as long as it is unique. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.1.7.4 BlockMetaData

Class in package '4.3.7 BlockchainLedger'

Details: This object represents the metadata that some blockchain environments attach to a block to keep track of the transactions details.

CONNECTORS
Dependency Source -> Destination From: : BlockMetaData : Class , Public To: GenericDigIdentityDep : Class , Public
Dependency Source -> Destination From: : LedgerBlock : Class , Public To: BlockMetaData : Class , Public

ATTRIBUTES
SubmissionAgent : **GenericDigIdentityDep** Private

ATTRIBUTES
Details: This contains a reference to the agent/system submitted the transaction. This is mostly applicable to permissioned systems, but can also be used to track the mining/proposing system address if applicable. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
SubmissionTime : **TimeStamp** Private *Details:* This represents the time at which the transaction was initially submitted into the system. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
TransactionID : **int** Private *Details:* This represents a unique identifier, transaction counter, that enables a system to uniquely identify a request. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.1.7.5 GenericBlockchainDep

Class in package '4.3.7 BlockchainLedger'

Details: Objects whom reference this class expect an object that realizes the *Blockchain* Interface. This class is a leaf and is only intended to serve as a data type.

CONNECTORS
Usage Source -> Destination From: : GenericBlockchainDep : Class , Public To: IBlockchain : ProvidedInterface , Public
Dependency Source -> Destination From: : UID : Class , Public To: GenericBlockchainDep : Class , Public
Dependency Source -> Destination From: : iSmartContract : Class , Public To: GenericBlockchainDep : Class , Public
Dependency Source -> Destination From: : BaseClass : Class , Public To: GenericBlockchainDep : Class , Public

A.1.7.6 GenericBlockchainRealization

Class in package '4.3.7 BlockchainLedger'

Details: This abstract class implements the *Blockchain* interface, classes derived from this class should satisfy all of the service and data requirements.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from GenericBlockchainRealization to «interface» iBlockchain
Realization from GenericBlockchainRealization to IBlockchain

A.1.7.7 iBlockchain

Class «interface» in package '4.3.7 BlockchainLedger'

STRUCTURAL PART OF iBlockchain
IBlockchain : ProvidedInterface

STRUCTURAL PART OF iBlockchain
ProvidedInterface1 : ProvidedInterface

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from «interface» iBlockchain to InmutableLedger
Generalization from «interface» iBlockchain to LedgerBlock

CONNECTORS
Dependency Source -> Destination From: : iBlockchain : Class , Public To: BlockchainStructure : Class , Public

ATTRIBUTES
BlochchainDescriptor : **BlockchainStructure** Private *Details:*
CurrentTransaction : **SerializableObject** Private *Details:* Contains a copy of the current transaction. This can be used to inspect the transaction contents, such as the identity of the submitter, calling parameters, endorsers, etc. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

OPERATIONS
GetCurrentTransaction () : **SerializableObject** Public *Details:* This function gets the CurrentTransaction. This function may proof useful when smart contracts are implemented.
ReadOffChain (FQID : **Int**) : **void** Private *Details:* This function should be implemented via a smart contract or other alike logic mechanism. It is the responsibility of the logic to ensure data read is validated before being returned (via a fingerprint/digest comparison
ReadOnChain (FQID : **int**) : **void** Private *Details:* This is a function that should be supported by the underlying Blockchain. This function asynchronous and the data obtained can be from a local peer copy or retrieved from another remote peer via a SC invokatio
StoreOffChain (FQID : **int** , anObject : **Bytes**) : **void** Private *Details:* This function should be implemented via a smart contract or other alike logic mechanism. It is the responsibility of the logic to ensure data is properly secured by storing the data's fingerprint in the ledge
StoreOnChain (FQID : **int** , AnObject : **Bytes**) : **void** Private *Details:* This is a function that should be supported by the underlying Blockchain. This function asynchronous and occurs after consensus and ordering occurs
SubmitTx (rawRequest : **Bytes** , SubmissionAgentIdentity : **GenericDigIdentityDep**) : **void** Public *Details:* This represents a ledger ability to accept transactions that contain within itself the data, parameters or commands required to update the ledger state. Sample requests may include: Putting raw data into the ledger. Calling an non-parameterized SC function. Calling a parameterized SC function.

A.1.7.8 InmutableLedger

Class in package '4.3.7 BlockchainLedger'

Details: This structuture represents an immutable ledger. This particular implementation follows the common "chain of blocks" idea by using an OrderedArrayList minus the remove operation. Note that although most blockchains use a Merkle-tree like structure, the walk from the current state to the genesis block can be represented using an ordered array list.

OUTGOING STRUCTURAL RELATIONSHIPS
Realization from InmutableLedger to OrderedArrayList

OPERATIONS
remove (Index : **int**) : **Not_supported** Public *Details:* This operation represents the lack of data removal capabilities from a ledger.

A.1.7.9 LedgerBlock

Class in package '4.3.7 BlockchainLedger'

Details: This structure represents a data block within the ledger. Strictly speaking a data and signature field are required, this object model presents a more extensible model based on Hyperledger Fabric .

CONNECTORS
Dependency Source -> Destination From: : LedgerBlock : Class , Public To: BlockMetaData : Class , Public
Dependency Source -> Destination From: : LedgerBlock : Class , Public To: BlockHeader : Class , Public
Dependency Source -> Destination From: : LedgerBlock : Class , Public To: BlockData : Class , Public

ATTRIBUTES
Data : **BlockData** Private *Details:* This field represents the raw data appended with a digital fingerprint. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Header : **BlockHeader** Private *Details:* This field contains information about the block, it can be used to map/identify a block within the ledger. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
MetaData : **BlockMetaData** Private *Details:* This field is used to store metadata about the block origins. This may include transaction tracking information, approval information and any other related metadata. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.1.7.10 LedgerStorage

Class in package '4.3.7 BlockchainLedger'

Details: This is an abstract class that represents any DLT/blockchain system that operates over a ledger-like system.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from LedgerStorage to InmutableLedger
Generalization from LedgerStorage to LedgerBlock

A.1.7.11 TypeOfConsesus

Enumeration in package '4.3.7 BlockchainLedger'

Details: This enumeration is used to describe the type of consensus used to propose and append a new block. Usually named *Proof of X*. This is not an exhaustive list and it is only representative.

CONNECTORS
Dependency Source -> Destination From: : BlockchainStructure : Class , Public To: TypeOfConsesus : Enumeration , Public

ENUMERATION:
ProofOfElapseTime
ProofOfStake
ProofOf ...
ProofOfWork
ProofOfVote

A.1.7.12 TypesOfBlockchain

Enumeration in package '4.3.7 BlockchainLedger'

Details: This enumeration is used to indicate the type of blockchain being used.

CONNECTORS
Dependency Source -> Destination From: : BlockchainStructure : Class , Public To: TypesOfBlockchain : Enumeration , Public

ENUMERATION:	
Permissioned	
Permissionless	
Hybrid	This represents a blockchain that has properties of both permissioned/permissionless.

A.1.7.13 TypesOfLedger

Enumeration in package '4.3.7 BlockchainLedger'

Details: This enumeration is used to describe the manner in which data blocks are ordered or structured.

CONNECTORS
Dependency Source -> Destination From: : BlockchainStructure : Class , Public To: TypesOfLedger : Enumeration , Public

ENUMERATION:
MerkleTree
RadixTree

A.1.8 LifecycleManagement

This diagram contains the necessary object templates and interfaces required to track an asset's state across its operational lifecycle. The provided interface is designed to function in an ad-hoc behavior (e.g., on demand). This contrasts with the mechanisms provided by the *trackableInterface* which are designed to be inheritable (and thus always tracking the changes in state).

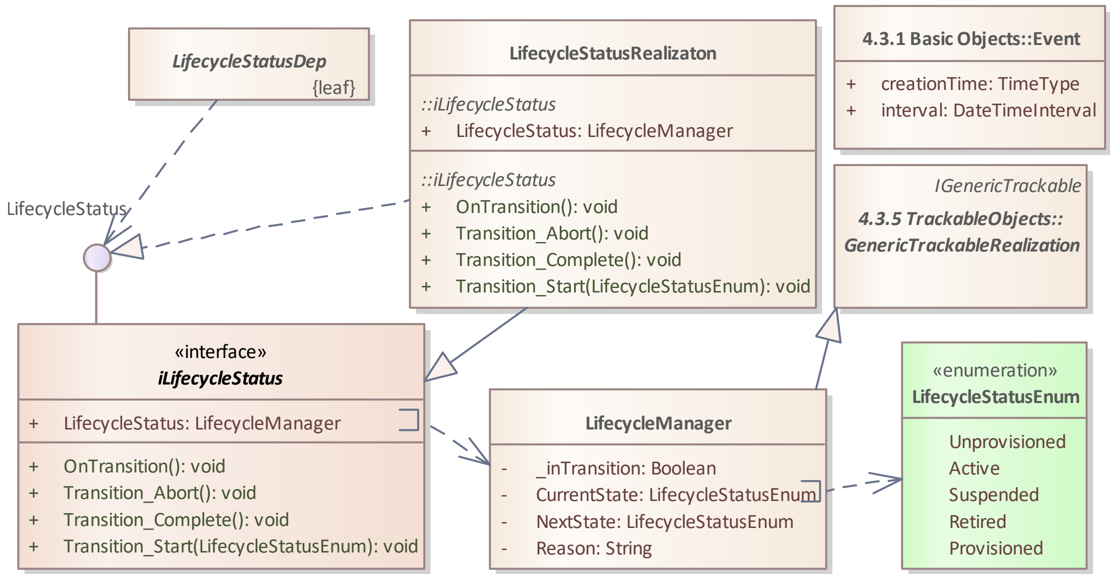


Figure 56. Overview of the LifecycleManagement's package components

A.1.8.1 iLifecycleStatus

Class «interface» in package '4.3.8 LifecycleManagement'

STRUCTURAL PART OF iLifecycleStatus
iLifecycleStatus : ProvidedInterface

CONNECTORS
Dependency Source -> Destination From: : iLifecycleStatus : Class , Public To: LifecycleManager : Class , Public

ATTRIBUTES
LifecycleStatus : **LifecycleManager** Public *Details:*

OPERATIONS
OnTransition () : **void** Public *Details:*
Transition_Abort () : **void** Public *Details:*

OPERATIONS
Transition_Complete () : **void** Public *Details:*
Transition_Start (newState : **LifecycleStatusEnum**) : **void** Public *Details:*

A.1.8.2 LifecycleManager

Class in package '4.3.8 LifecycleManagement'

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from LifecycleManager to GenericTrackableRealization

CONNECTORS
Dependency Source -> Destination From: : LifecycleManager : Class , Public To: LifecycleStatusEnum : Enumeration , Public
Dependency Source -> Destination From: : Individual : Class , Public To: LifecycleManager : Class , Public
Dependency Source -> Destination From: : Address : Class , Public To: LifecycleManager : Class , Public
Dependency Source -> Destination From: : iLifecycleStatus : Class , Public To: LifecycleManager : Class , Public
Dependency Source -> Destination From: : iGridEquipment : Class , Public To: LifecycleManager : Class , Public

ATTRIBUTES
_inTransition : **Boolean** Private *Details:*
CurrentState : **LifecycleStatusEnum** Private *Details:*
NextState : **LifecycleStatusEnum** Private *Details:*
Reason : **String** Private *Details:* This field can be used to provide an explanation on the reasoning behind the current lifecycle status. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.1.8.3 LifecycleStatusDep

Class in package '4.3.8 LifecycleManagement'

Details: Objects whom reference this class expect an object that realizes the *LifeCycle* Interface. This class is a leaf and is only intended to serve as a data type.

CONNECTORS
Dependency Source -> Destination From: : LifecycleStatusDep : Class , Public To: iLifecycleStatus : ProvidedInterface , Public

A.1.8.4 LifecycleStatusRealizaton

Class in package '4.3.8 LifecycleManagement'

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from LifecycleStatusRealizaton to «interface» iLifecycleStatus
Realization from LifecycleStatusRealizaton to iLifecycleStatus

A.1.8.5 LifecycleStatusEnum

Enumeration in package '4.3.8 LifecycleManagement'

CONNECTORS
Dependency Source -> Destination From: : LifecycleManager : Class , Public To: LifecycleStatusEnum : Enumeration , Public

ENUMERATION:
Unprovisioned
Active
Suspended
Retired
Provisioned

A.1.9 Permissions&Qualifications

This diagram contains the basic interfaces for defining access control permissions to resources, as well as mechanisms for assigning qualifications/attributes to entities. These interfaces must be configured to suit the application/use case needs. For example, different types of qualifications may exist within a single TES implementation, each of them applicable to different set of agents, participants or external service providers.

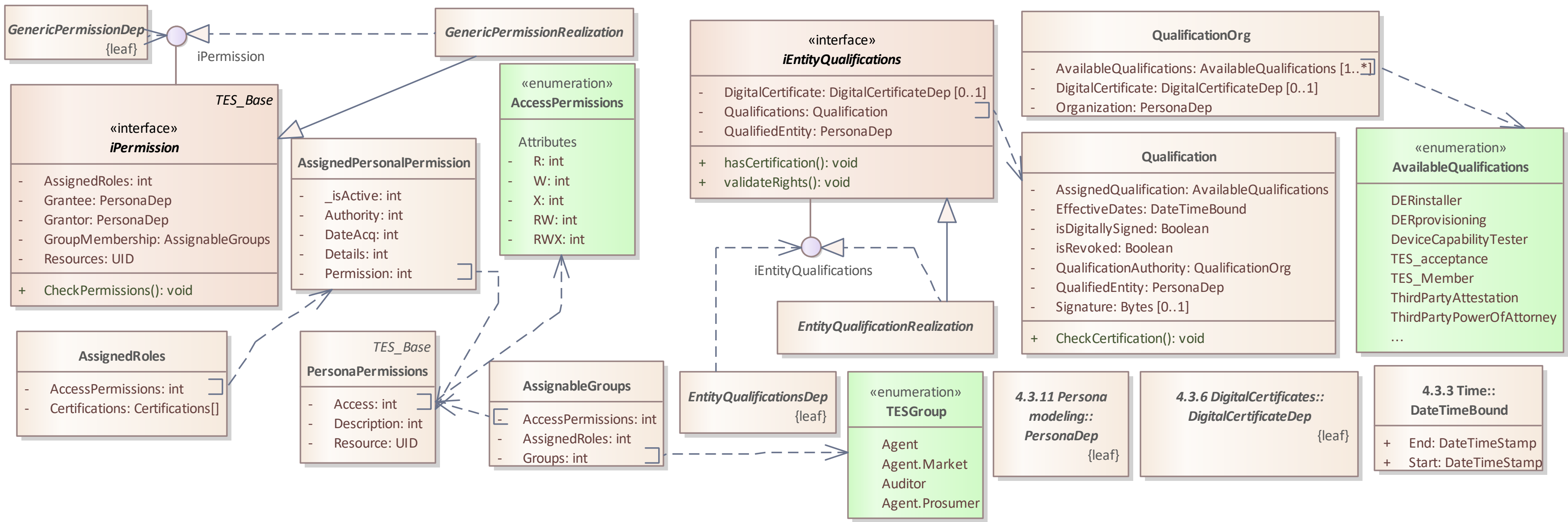


Figure 57. Overview of the Permissions' package components

A.1.9.1 AssignableGroups

Class in package '4.3.9 Permissions&Qualifications'

CONNECTORS
Dependency Source -> Destination From: : AssignableGroups : Class , Public To: PersonaPermissions : Class , Public
Dependency Source -> Destination From: : AssignableGroups : Class , Public To: PersonaPermissions : Class , Public
Dependency Source -> Destination From: : AssignableGroups : Class , Public To: IndustryPersonaCertifications : Class , Public
Dependency Source -> Destination From: : AssignableGroups : Class , Public To: TESGroup : Enumeration , Public

ATTRIBUTES
AccessPermissions : **int** Private *Details:*
AssignedRoles : **int** Private *Details:*
Groups : **int** Private *Details:*

A.1.9.2 AssignedPersonalPermission

Class in package '4.3.9 Permissions&Qualifications'

CONNECTORS
Dependency Source -> Destination From: : AssignedPersonalPermission : Class , Public To: AccessStatus : Enumeration , Public
Dependency Source -> Destination From: : AssignedPersonalPermission : Class , Public To: PersonaPermissions : Class , Public
Dependency Source -> Destination From: : AssignedRoles : Class , Public To: AssignedPersonalPermission : Class , Public

ATTRIBUTES
_isActive : **int** Private *Details:*
Authority : **int** Private *Details:*
DateAcq : **int** Private *Details:*
Details : **int** Private *Details:*
Permission : **int** Private *Details:*

A.1.9.3 AssignedRoles

Class in package '4.3.9 Permissions&Qualifications'

CONNECTORS
Dependency Source -> Destination From: : AssignedRoles : Class , Public To: Assigned_IndustryPersonaCertifications : Class , Public
Dependency Source -> Destination From: : AssignedRoles : Class , Public To: AssignedPersonalPermission : Class , Public

ATTRIBUTES
AccessPermissions : **int** Private *Details:*
Certifications : **Certifications[]** Private *Details:*

A.1.9.4 EntityQualificationRealization

Class in package '4.3.9 Permissions&Qualifications'

Details: This class represents the qualifications that a given entity possesses, which may be composed of a wide variety of individual qualifications obtained from trusted or well known systems.

OUTGOING STRUCTURAL RELATIONSHIPS
Realization from EntityQualificationRealization to iEntityQualifications
Generalization from EntityQualificationRealization to «interface» iEntityQualifications

A.1.9.5 EntityQualificationsDep

Class in package '4.3.9 Permissions&Qualifications'

Details: Objects whom reference this class expect an object that realizes the *EntityQualifications* Interface, an interface that be used to describe an entity qualifications as certified by a trusted/known system. This class is a leaf and is only intended to serve as a data type.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from EntityQualificationsDep to «interface» iEntityQualifications

CONNECTORS
Dependency Source -> Destination From: : EntityQualificationsDep : Class , Public To: iEntityQualifications : ProvidedInterface , Public
Dependency Source -> Destination From: : AttestationCapabilities : Class , Public To: EntityQualificationsDep : Class , Public
Dependency Source -> Destination From: : QualifiedAuditors : Class , Public To: EntityQualificationsDep : Class , Public
Dependency Source -> Destination From: : QualifiedInstallers : Class , Public To: EntityQualificationsDep : Class , Public

A.1.9.6 GenericPermissionDep

Class in package '4.3.9 Permissions&Qualifications'

Details: Objects whom reference this class expect an object that realizes the *GenericPermission* Interface, an interface that be used to store the permissions associated with a given resource.

CONNECTORS
Dependency Source -> Destination From: : GenericPermissionDep : Class , Public To: iPermission : ProvidedInterface , Public
Dependency Source -> Destination From: : SubjectPermissions : Class , Public To: GenericPermissionDep : Class , Public

A.1.9.7 GenericPermissionRealization

Class in package '4.3.9 Permissions&Qualifications'

OUTGOING STRUCTURAL RELATIONSHIPS
Realization from GenericPermissionRealization to iPermission
Generalization from GenericPermissionRealization to «interface» iPermission

A.1.9.8 GivenQualification

Class in package '4.3.9 Permissions&Qualifications'

A.1.9.9 iEntityQualifications

Class «interface» in package '4.3.9 Permissions&Qualifications'

STRUCTURAL PART OF iEntityQualifications
iEntityQualifications : ProvidedInterface

CONNECTORS
Dependency Source -> Destination From: : iEntityQualifications : Class , Public To: Qualification : Class , Public

ATTRIBUTES
DigitalCertificate : **DigitalCertificateDep** Private *Details:* This is an optional parameter that enables to tie the persona to a digital certificate (to prevent identity theft). Multiplicity: (0..1, Allow duplicates: 0, Is ordered: False)
Qualifications : **Qualification** Private

ATTRIBUTES
Details: This represents all qualifications that an entity pertains to have. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
QualifiedEntity : **PersonaDep** Private *Details:* This represents the entity for which this qualifications apply. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

OPERATIONS
hasCertification () : **void** Public *Details:* This property enables external systems to evaluate if the entity satisfies a given qualification.
validateRights () : **void** Public *Details:* Given the current qualifications, and an input request, the entity should be able to determine if it has the correct rights.

A.1.9.10 iPermission

Class «interface» in package '4.3.9 Permissions&Qualifications'

STRUCTURAL PART OF iPermission
iPermission : ProvidedInterface

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from «interface» iPermission to TES_Base

ATTRIBUTES
AssignedRoles : **int** Private *Details:*
Grantee : **PersonaDep** Private *Details:*
Grantor : **PersonaDep** Private *Details:*
GroupMembership : **AssignableGroups** Private *Details:*
Resources : **UID** Private *Details:*

OPERATIONS
CheckPermissions () : **void** Public *Details:*

A.1.9.11 PersonaPermissions

Class in package '4.3.9 Permissions&Qualifications'

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from PersonaPermissions to TES_Base

CONNECTORS
Dependency Source -> Destination From: : PersonaPermissions : Class , Public To: AccessPermissions : Enumeration , Public
Dependency Source -> Destination From: : AssignableGroups : Class , Public To: PersonaPermissions : Class , Public
Dependency Source -> Destination From: : AssignableGroups : Class , Public To: PersonaPermissions : Class , Public
Dependency Source -> Destination From: : AssignedPersonalPermission : Class , Public To: PersonaPermissions : Class , Public

ATTRIBUTES
Access : **int** Private *Details:*
Description : **int** Private *Details:*
Resource : **UID** Private *Details:*

A.1.9.12 Qualification

Class in package '4.3.9 Permissions&Qualifications'

Details: This object holds a single qualification for an individual

STRUCTURAL PART OF Qualification
iPermission : ProvidedInterface

CONNECTORS
Usage «Instantiate» Source -> Destination From: : Actor's_Qualifications : Object , Public To: Qualification : Class , Public
Dependency Source -> Destination From: : iEntityQualifications : Class , Public To: Qualification : Class , Public

ATTRIBUTES
AssignedQualification : **AvailableQualifications** Private *Details:* This represents the qualification being given. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
EffectiveDates : **DateTimeBound** Private *Details:* This field can be used to time-bound a given qualification Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
isDigitallySigned : **Boolean** Private *Details:* This field can be used to establish if the qualification has been digitally signed. It requires the *QualificationOrg* to possess a *digitalCertificate.* Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
isRevoked : **Boolean** Private *Details:* This field can be used to indicate an early revocation. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
QualificationAuthority : **QualificationOrg** Private

ATTRIBUTES
Details: This is a reference to the *QuaficationAuthority*. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
QualifiedEntity : **PersonaDep** Private *Details:* This field represents the entity that has received the qualification. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Signature : **Bytes** Private *Details:* This field can be used to store an optional signature that can be used to provide greater security. Multiplicity: (0..1, Allow duplicates: 0, Is ordered: False)

OPERATIONS
CheckCertification () : **void** Public *Details:* This function can be used to check a certification, unless digitally signed this function may incorrectly return true for the listed qualification.

A.1.9.13 QualificationOrg

Class in package '4.3.9 Permissions&Qualifications'

Details: This structure holds organizations that can issue qualifications to other members.

CONNECTORS
Dependency Source -> Destination From: : QualificationOrg : Class , Public To: AvailableQualifications : Enumeration , Public

ATTRIBUTES
AvailableQualifications : **AvailableQualifications** Private *Details:* This lists all the qualifications that an organization can issue. A system administrator is reponsible for creating this organizations (similar to a CA) Multiplicity: (1..*, Allow duplicates: 0, Is ordered: False)
DigitalCertificate : **DigitalCertificateDep** Private *Details:* This is an optional field that can be used to digitally sign the issued qualifications. Multiplicity: (0..1, Allow duplicates: 0, Is ordered: False)
Organization : **PersonaDep** Private *Details:* This represents an organization or entity responsible for handling these qualifications. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.1.9.14 AccessPermissions

Enumeration in package '4.3.9 Permissions&Qualifications'

Details: This enumeration provides a sample of access permissions that can be given to a resource. The meaning of each flag is:
R - Read is allowed
W - Write is allowed
X - Execution is allowed.

CONNECTORS
Dependency Source -> Destination From: : Policy : Class , Public To: AccessPermissions : Enumeration , Public

CONNECTORS
Dependency Source -> Destination From: : PersonaPermissions : Class , Public To: AccessPermissions : Enumeration , Public

ENUMERATION:
R
W
X
RW
RWX

A.1.9.15 AvailableQualifications

Enumeration in package '4.3.9 Permissions&Qualifications'

CONNECTORS
Dependency Source -> Destination From: : QualificationOrg : Class , Public To: AvailableQualifications : Enumeration , Public

ENUMERATION:	
DERinstaller	e.g., being capable of installing DER equipment
DERprovisioning	e.g. Being capable of provisioning a DER system (setup users, connection profile)
DeviceCapabilityTester	i.e., being able to test a generator output capabilities
TES_acceptance	To represent an entity that can accept a potential agent into a TES.
TES_Member	e.g., to be part of a TES.
ThirdPartyAttestation	Being capable of attesting for another party.
ThirdPartyPowerOfAttorney	Legal power of attorney rights.
...	

A.1.9.16 TESGroup

Enumeration in package '4.3.9 Permissions&Qualifications'

Details: This enumeration represents typical group roles found within a TES. They are specific to the ABAC case being demonstrated.

CONNECTORS
Dependency Source -> Destination From: : PolicyResource : Class , Public To: TESGroup : Enumeration , Public
Dependency Source -> Destination From: : AssignableGroups : Class , Public To: TESGroup : Enumeration , Public

ENUMERATION:	
Agent	This represents any agent that can participate within the deployed TES.
Agent.Market	This represents an specialized agent that has market running duties.
Auditor	This represents an auditor within a TES.
Agent.Prosumer	This represents an specialized agent that has prosumer capabilities.

A.1.10 Grid Object Models

This package provides the basic constructs used to model electrical systems. This includes the ability to store matrix data (for impedance), represent complex power and its direction. The contained classes represent only a subset of electrical-related objects and must be updated depending on the TES' application requirements.

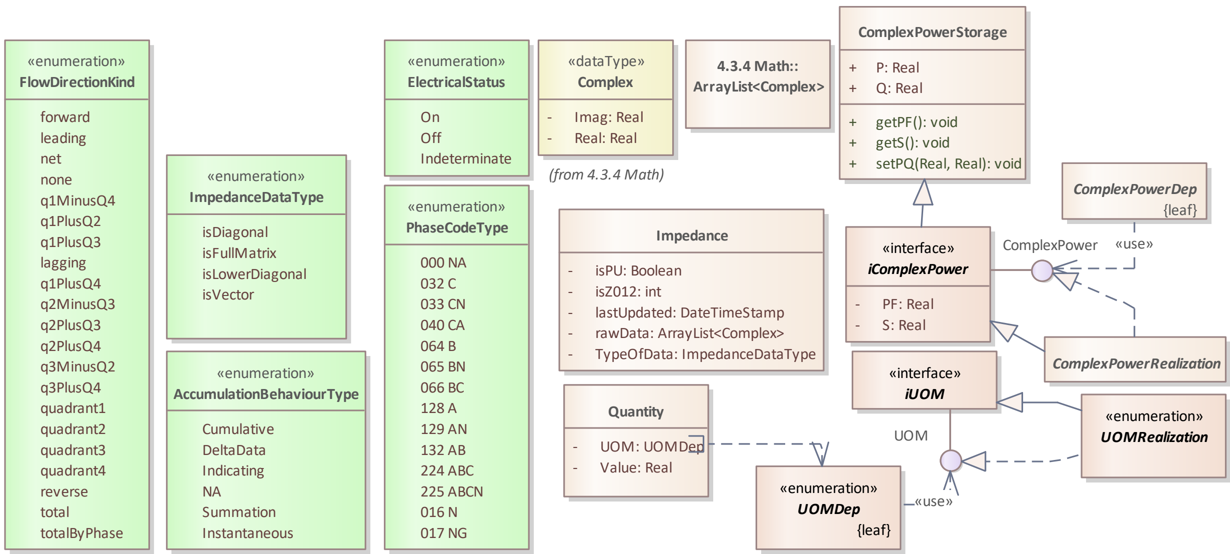


Figure 58. Overview of the GridObjects' package components

A.1.10.1 ComplexPowerDep

Class in package '4.3.10 Grid Objects'

Details: Objects whom reference this class expect an object that realizes the *ComplexPower* Interface. This class is a leaf and is only intended to serve as a data type.

CONNECTORS
Usage Source -> Destination From: : ComplexPowerDep : Class , Public To: ComplexPower : ProvidedInterface , Public

A.1.10.2 ComplexPowerRealization

Class in package '4.3.10 Grid Objects'

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from ComplexPowerRealization to «interface» iComplexPower
Realization from ComplexPowerRealization to ComplexPower

A.1.10.3 ComplexPowerStorage

Class in package '4.3.10 Grid Objects'

ATTRIBUTES
P : **Real** Public *Details:*
Q : **Real** Public *Details:*

OPERATIONS
getPF () : **void** Public *Details:* This function is used to calculate PF on demand.
getS () : **void** Public *Details:* This function is used to calculate S on demand.
setPQ (S : **Real** , PF : **Real**) : **void** Public *Details:* This function sets the P,Q values based on a given S and a power factor.

A.1.10.4 iComplexPower

Class «interface» in package '4.3.10 Grid Objects'

STRUCTURAL PART OF iComplexPower
ComplexPower : ProvidedInterface

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from «interface» iComplexPower to ComplexPowerStorage

ATTRIBUTES
PF : **Real** Private *Details:*
S : **Real** Private *Details:*

A.1.10.5 Impedance

Class in package '4.3.10 Grid Objects'

Details: This object stores the impedance characteristics at a particular node.

CONNECTORS
Dependency Source -> Destination From: : PCCParameters : Class , Public To: Impedance : Class , Public

ATTRIBUTES
isPU : **Boolean** Private *Details:* This field is used to define if the data provided is in per unit. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
isZ012 : **int** Private

ATTRIBUTES
Details: This field is used to indicate if the data is presented in Positive, Negative, Zero sequence. *TypeOfData* must be *isVector*. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
lastUpdated : **DateTimeStamp** Private *Details:* This field is used to indicate the last time this field was last updated. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
rawData : **ArrayList<Complex>** Private *Details:* This field contains the raw data, the order is: From the top, left to right. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
TypeOfData : **ImpedanceDataType** Private *Details:* This specifies the structure of data being provided in the *rawData* array. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.1.10.6 iUOM

Class «interface» in package '4.3.10 Grid Objects'

Details: This interface should be realized by an Enum-like structure. It should contain the units of measurement that are specific to the use domain

STRUCTURAL PART OF iUOM
UOM : ProvidedInterface

A.1.10.7 Quantity

Class in package '4.3.10 Grid Objects'

CONNECTORS
Dependency Source -> Destination From: : Quantity : Class , Public To: UOMDep : Enumeration , Public

ATTRIBUTES
UOM : **UOMDep** Private *Details:* This field encodes the Unit Of Measure. This field should always be redefined according to the use case. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Value : **Real** Private *Details:*

A.1.10.8 UOMDep

Enumeration «enumeration» in package '4.3.10 Grid Objects'

CONNECTORS
Usage Source -> Destination From: : UOMDep : Enumeration , Public To: UOM : ProvidedInterface , Public

CONNECTORS
Dependency Source -> Destination From: : Quantity : Class , Public To: UOMDep : Enumeration , Public

A.1.10.9 UOMRealization

Enumeration «enumeration» in package '4.3.10 Grid Objects'

OUTGOING STRUCTURAL RELATIONSHIPS
Realization from «enumeration» UOMRealization to UOM
Generalization from «enumeration» UOMRealization to «interface» iUOM

A.1.10.10 AccumulationBehaviourType

Enumeration in package '4.3.10 Grid Objects'

Details: This enumeration was taken from IEEE 2030.5, which lists the type of value that is being reported (with respect a measurement).

0 = Not Applicable
3 = Cumulative
4 = DeltaData
6 = Indicating
9 = Summation

CONNECTORS
Dependency Source -> Destination From: : ReadingType : Class , Public To: AccumulationBehaviourType : Enumeration , Public

ENUMERATION:	
Cumulative	"The sum of the previous billing period values".
DeltaData	This number represents the change from the previously reported quantity.
Indicating	Represents an "instantaneous" value which has been subject to "filtering" to obtain a more representative value
NA	Not Applicable
Summation	An accumulation of values with respect to a time reference, e.g., integration.
Instantaneous	A value measured instantaneously, using the minimum amount of time to capture it.

A.1.10.11 ElectricalStatus

Enumeration in package '4.3.10 Grid Objects'

Details: This enumeration is used to indicate an electrical switch state. This enumeration was adapted from IEEE 2030.5

CONNECTORS
Dependency Source -> Destination From: : UsagePointBase : Class , Public To: ElectricalStatus : Enumeration , Public

ENUMERATION:	
On	Switch is closed
Off	Switch is open
Indeterminate	The switch state cannot be determined.

A.1.10.12 FlowDirectionKind

Enumeration in package '4.3.10 Grid Objects'

Details: This enumeration lists the way that a quantity is being measured. This quantity is assumed to be in the direction from the device that is performing the measurement.
E.g. if a generator is doing the measurement, a forward quantity means delivery into the PCC.

CONNECTORS
Dependency Source -> Destination From: : ReadingType : Class , Public To: FlowDirectionKind : Enumeration , Public

ENUMERATION:
forward
leading
net
none
q1MinusQ4
q1PlusQ2
q1PlusQ3
lagging
q1PlusQ4
q2MinusQ3
q2PlusQ3
q2PlusQ4
q3MinusQ2
q3PlusQ4
quadrant1
quadrant2
quadrant3
quadrant4
reverse
total
totalByPhase

A.1.10.13 ImpedanceDataType

Enumeration in package '4.3.10 Grid Objects'

Details: This enumeration is used to characterize the type of data being stored.

ENUMERATION:
isDiagonal
isFullMatrix

ENUMERATION:
isLowerDiagonal
isVector

A.1.10.14 PhaseCodeType

Enumeration in package '4.3.10 Grid Objects'

Details: This enumeration object represents the phases typically found on an electrical power system.
This phase encoding was derived from the one found on IEEE 2030.5 and IEC (using a bitmask-like enumeration).
0 = Not Applicable (default, if not specified)
32 = Phase C (and S2)
33 = Phase CN (and S2N)
40 = Phase CA
64 = Phase B
65 = Phase BN
66 = Phase BC
128 = Phase A (and S1)
129 = Phase AN (and S1N)
132 = Phase AB
224 = Phase ABC
All other values reserved.

Code = Networked × 2^14 + Open × 2^13 + HighLeg × 2^12 + Delta × 2^11 + Wye × 2^10 + S1 × 2^9 + S2 × 2^8 + A1 × 2^7 + B1 × 2^6 + C1 × 2^5 + N1 × 2^4 + A2 × 2^3 + B2 × 2^2 + C1 × 2^1 + N2

CONNECTORS
Dependency Source -> Destination From: : ReadingType : Class , Public To: PhaseCodeType : Enumeration , Public
Dependency Source -> Destination From: : PCCParameters : Class , Public To: PhaseCodeType : Enumeration , Public

ENUMERATION:
000 NA
032 C
033 CN
040 CA
064 B
065 BN
066 BC
128 A
129 AN
132 AB
224 ABC
225 ABCN
016 N
017 NG

A.1.11 Persona modeling

This package groups an assortment of classes that can be used to capture an entity's personal information. In addition, digital certificates and other locational information can be used to support advanced identity services. These identity interfaces can be referenced by other higher-level models to specify participants, and provided a trusted-operational platform.

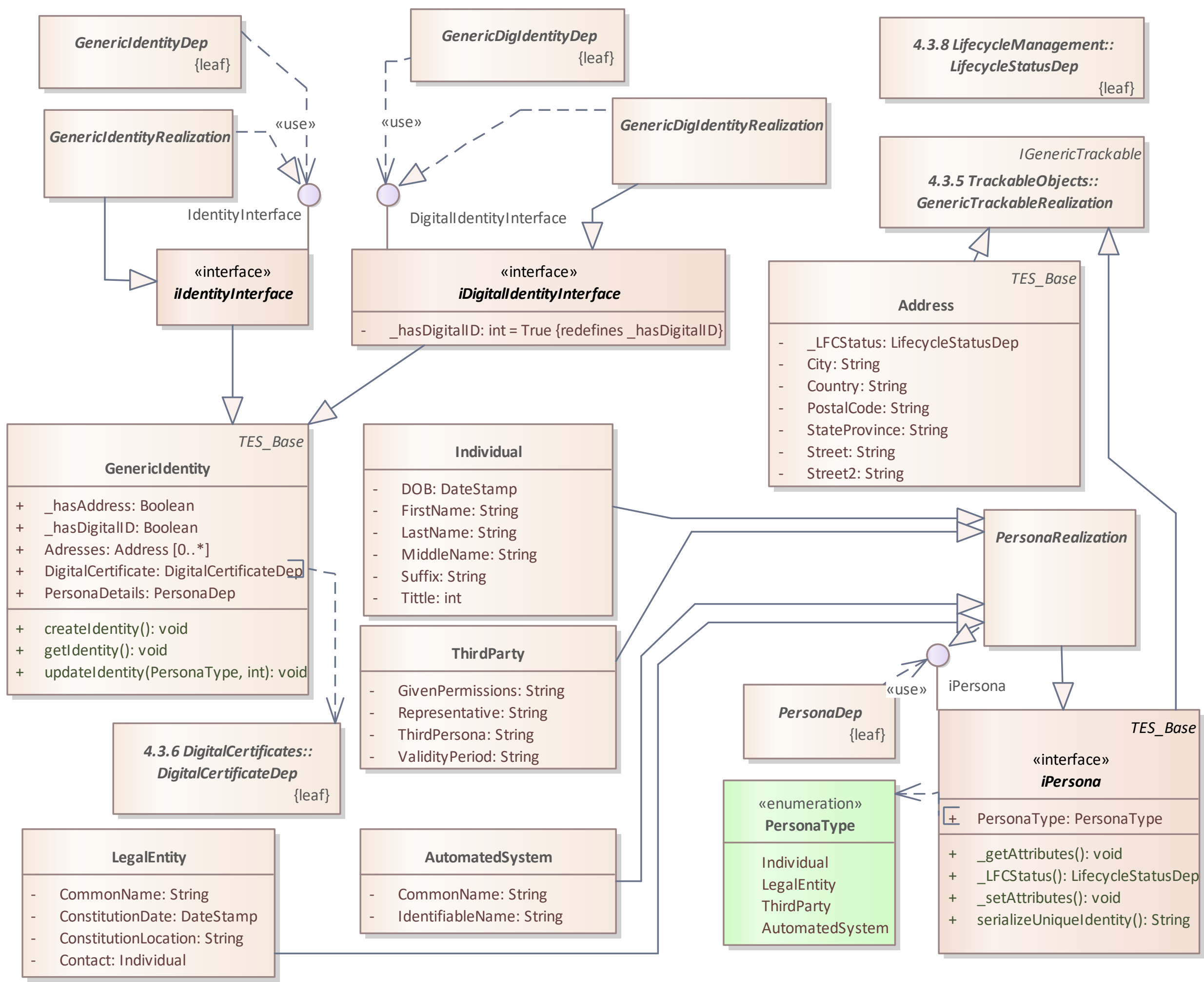


Figure 59. Overview of the Persona's package components

A.1.11.1 Address

Class in package '4.3.11 Persona modeling'

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from Address to TES_Base
Generalization from Address to GenericTrackableRealization

CONNECTORS
Dependency Source -> Destination From: : Address : Class , Public To: LifecycleManager : Class , Public

ATTRIBUTES
_LFCStatus : **LifecycleStatusDep** Private *Details:*
City : **String** Private *Details:*
Country : **String** Private *Details:*
PostalCode : **String** Private *Details:*
StateProvince : **String** Private *Details:*
Street : **String** Private *Details:*
Street2 : **String** Private *Details:*

A.1.11.2 AutomatedSystem

Class in package '4.3.11 Persona modeling'

Details: This is a sample realization of the *Persona* interface. It can be used to store data typically associated with automated agents.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from AutomatedSystem to PersonaRealization

ATTRIBUTES
CommonName : **String** Private *Details:* This field can be used to store a user-defined name that can generically describe the asset. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
IdentifiableName : **String** Private *Details:* This field should be used to uniquely name the device within the system. For example Agent.0002, or in the format Org.Division.Dept.Number. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.1.11.3 GenericDigIdentityDep

Class in package '4.3.11 Persona modeling'

Details: Objects that reference this class expect an object that realizes the *Digital Identity* Interface. This class is a leaf and is only intended to serve as a data type.

CONNECTORS
Usage Source -> Destination From: : GenericDigIdentityDep : Class , Public To: DigitalIdentityInterface : ProvidedInterface , Public
Dependency Source -> Destination From: : BlockMetaData : Class , Public To: GenericDigIdentityDep : Class , Public

A.1.11.4 GenericDigIdentityRealization

Class in package '4.3.11 Persona modeling'

Details: This abstract class implements the *DigitalIdentity* interface, classes derived from this class should satisfy all of the service and data requirements. All realizations of this class must provide access to a digital certificate.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from GenericDigIdentityRealization to «interface» iDigitalIdentityInterface
Realization from GenericDigIdentityRealization to DigitalIdentityInterface

A.1.11.5 GenericIdentity

Class in package '4.3.11 Persona modeling'

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from GenericIdentity to TES_Base

CONNECTORS
Dependency Source -> Destination From: : GenericIdentity : Class , Public To: DigitalCertificateDep : Class , Public

ATTRIBUTES
_hasAddress : **Boolean** Public *Details:*
_hasDigitalID : **Boolean** Public *Details:*
Adresses : **Address** Public *Details:*
DigitalCertificate : **DigitalCertificateDep** Public *Details:*
PersonaDetails : **PersonaDep** Public *Details:*

OPERATIONS
createIdentity () : **void** Public *Details:*
getIdentity () : **void** Public *Details:*
updateIdentity (Type : **PersonaType** , Parameters : **int**) : **void** Public *Details:*

A.1.11.6 GenericIdentityDep

Class in package '4.3.11 Persona modeling'

Details: Objects whom reference this class expect an object that realizes the *Identity* Interface. This class is a leaf and is only intended to serve as a data type.

CONNECTORS
Usage Source -> Destination From: : GenericIdentityDep : Class , Public To: IdentityInterface : ProvidedInterface , Public
Dependency Source -> Destination From: : iMembershipMap : Class , Public To: GenericIdentityDep : Class , Public
Dependency Source -> Destination From: : iMembershipMap : Class , Public To: GenericIdentityDep : Class , Public
Dependency Source -> Destination From: : IGenericTrackable : Class , Public To: GenericIdentityDep : Class , Public
Dependency Source -> Destination From: : RecordTracker : Class , Public To: GenericIdentityDep : Class , Public

A.1.11.7 GenericIdentityRealization

Class in package '4.3.11 Persona modeling'

Details: This class represents an identity, which may be composed of an individual, entity, physical addresses and a digital identity (if applicable).

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from GenericIdentityRealization to «interface» iIdentityInterface
Realization from GenericIdentityRealization to IdentityInterface

A.1.11.8 iDigitalIdentityInterface

Class «interface» in package '4.3.11 Persona modeling'

STRUCTURAL PART OF iDigitalIdentityInterface
DigitalIdentityInterface : ProvidedInterface

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from «interface» iDigitalIdentityInterface to GenericIdentity

ATTRIBUTES
_hasDigitalID : **int** Private = True *Details:*

A.1.11.9 iIdentityInterface

Class «interface» in package '4.3.11 Persona modeling'

STRUCTURAL PART OF iIdentityInterface
IdentityInterface : ProvidedInterface

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from «interface» iIdentityInterface to GenericIdentity

A.1.11.10 Individual

Class in package '4.3.11 Persona modeling'

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from Individual to PersonaRealization

CONNECTORS
Dependency Source -> Destination From: : Individual : Class , Public To: LifecycleManager : Class , Public
Dependency Source -> Destination From: : Individual : Class , Public To: PersonStatus : Enumeration , Public

ATTRIBUTES
DOB : **DateStamp** Private *Details:*
FirstName : **String** Private *Details:*
LastName : **String** Private *Details:*
MiddleName : **String** Private *Details:*
Suffix : **String** Private *Details:*
Tittle : **int** Private *Details:*

A.1.11.11 iPersona

Class «interface» in package '4.3.11 Persona modeling'

Details: This represents a generic interface that offers the ability to retrieve basic info about an individual

STRUCTURAL PART OF iPersona
iPersona : ProvidedInterface

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from «interface» iPersona to TES_Base
Generalization from «interface» iPersona to GenericTrackableRealization

CONNECTORS
Dependency Source -> Destination From: : iPersona : Class , Public To: PersonaType : Enumeration , Public

ATTRIBUTES
PersonaType : **PersonaType** Public *Details:*

OPERATIONS
_getAttributes () : **void** Public *Details:*
_LFCStatus () : **LifecycleStatusDep** Public *Details:*
_setAttributes () : **void** Public *Details:*
serializeUniqueIdentity () : **String** Public *Details:* This function takes any persona-like object and serializes into a string representation that is human readable.

A.1.11.12 LegalEntity

Class in package '4.3.11 Persona modeling'

Details: This sample realization of a persona can serve as the base object for defining a legal entity such as a company, organization.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from LegalEntity to PersonaRealization

ATTRIBUTES
CommonName : **String** Private *Details:*
ConstitutionDate : **DateStamp** Private *Details:*
ConstitutionLocation : **String** Private *Details:*
Contact : **Individual** Private *Details:*

A.1.11.13 OwnerInfo

Class in package '4.3.11 Persona modeling'

A.1.11.14 PersonaDep

Class in package '4.3.11 Persona modeling'

Details: Objects whom reference this class expect an object that realizes the *Persona* Interface. This class is a leaf and is only intended to serve as a data type.

CONNECTORS
Usage Source -> Destination From: : PersonaDep : Class , Public To: iPersona : ProvidedInterface , Public

A.1.11.15 PersonaRealization

Class in package '4.3.11 Persona modeling'

OUTGOING STRUCTURAL RELATIONSHIPS
Realization from PersonaRealization to iPersona
Generalization from PersonaRealization to «interface» iPersona

A.1.11.16 ThirdParty

Class in package '4.3.11 Persona modeling'

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from ThirdParty to PersonaRealization

ATTRIBUTES
GivenPermissions : **String** Private *Details:*
Representative : **String** Private *Details:*
ThirdPersona : **String** Private *Details:*
ValidityPeriod : **String** Private *Details:*

A.1.11.17 PersonaType

Enumeration in package '4.3.11 Persona modeling'

Details: This describes the type of persona that is contained within this object.

CONNECTORS
Dependency Source -> Destination From: : iPersona : Class , Public To: PersonaType : Enumeration , Public

ENUMERATION:	
Individual	This represents a physical person.
LegalEntity	These can be corporations, companies or any other association that has individual-like properties but it is not a person.
ThirdParty	This represents an agent that acts on behalf of another persona.
AutomatedSystem	

A.1.12 Memberships

These classes can be used to establish memberships among two different systems. It is assumed that memberships requests are negotiated internally in between parties. The process assumes that a request-approval process occurs in between a solicitor and a target system, the target agent is responsible for evaluating the impacts/consequences of the relationship.

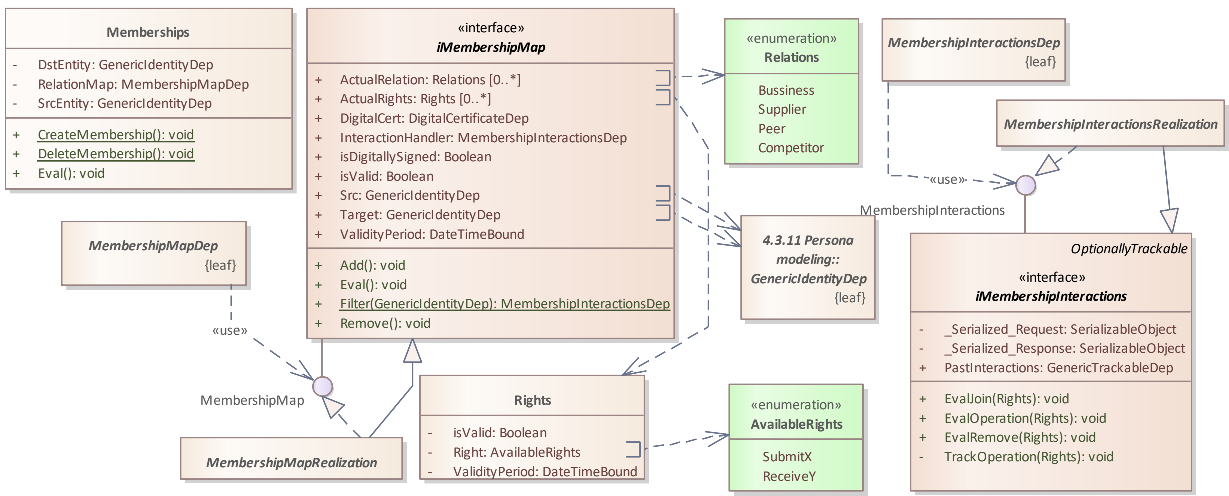


Figure 60. Overview of the Memberships' package components

A.1.12.1 iMembershipInteractions

Class «interface» in package '4.3.12 Memberships'

Details: This inheritable class provides describes the functions that can be executed under the context of both parties. This process is to be performed at the target system side. The target should verify that the presented rights and conditions satisfy its requirements.

STRUCTURAL PART OF iMembershipInteractions
MembershipInteractions : ProvidedInterface

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from «interface» iMembershipInteractions to OptionallyTrackable

ATTRIBUTES
_Serialized_Request : **SerializableObject** Private *Details:* This represents a serialized copy of the membership evaluation request. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
_Serialized_Response : **SerializableObject** Private *Details:*
PastInteractions : **GenericTrackableDep** Public *Details:* This field can be used to track past interactions (if the class is configured to do so, via the inherited *OptionallyTrackable* object. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

OPERATIONS
EvalJoin (existentRights : **Rights**) : **void** Public *Details:* This is a dedicated handler used to evaluate join operations
EvalOperation (existentRights : **Rights**) : **void** Public *Details:* This function can be used to evaluate all requested operations (other than the dedicated leave/join functions). Common functions could include extend membership or live validation. Do not implement complex TES-related actions.
EvalRemove (existentRights : **Rights**) : **void** Public *Details:* This is a dedicated function for handling the removal/revocation of a existing membership
TrackOperation (existentRights : **Rights**) : **void** Private *Details:* This add-on function can be used to track a membership state if required.

A.1.12.2 iMembershipMap

Class «interface» in package '4.3.12 Memberships'

Details: This inheritable class provides the basic mechanisms to track memberships among a fixed set of parties. Specific callbacks can be attached to the *Join/Remove* functions.

STRUCTURAL PART OF iMembershipMap
MembershipMap : ProvidedInterface

CONNECTORS
Dependency Source -> Destination From: : iMembershipMap : Class , Public To: GenericIdentityDep : Class , Public
Dependency Source -> Destination From: : iMembershipMap : Class , Public To: GenericIdentityDep : Class , Public
Dependency Source -> Destination From: : iMembershipMap : Class , Public To: Relations : Enumeration , Public
Dependency Source -> Destination From: : iMembershipMap : Class , Public To: Rights : Class , Public

ATTRIBUTES
ActualRelation : **Relations** Public *Details:* This field can be used to establish the relationship in between parties. This field is mostly for informative purposes. Multiplicity: (0..*, Allow duplicates: 0, Is ordered: False)
ActualRights : **Rights** Public *Details:* This field is used to establish the rights that the SOURCE entity has with the TARGET entity. By default this rights are not bidirectional. Multiplicity: (0..*, Allow duplicates: 0, Is ordered: False)
DigitalCert : **DigitalCertificateDep** Public *Details:* This is an optional certificate that can be attached to support the claimed membership rights. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
InteractionHandler : **MembershipInteractionsDep** Public *Details:* This field is used to define the membership handling functions used for evaluating interactions between both parties. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
isDigitallySigned : **Boolean** Public

ATTRIBUTES
Details: This field can be used to describe if the membership map has been digitally signed by the grantee (the target system/agent). Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
isValid : **Boolean** Public *Details:* This field can be used to evaluate if the membership map as a whole is valid. This may be relevant when credentials are compromised or a party member has reached the end of its lifecycle. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Src : **GenericIdentityDep** Public *Details:* This field is used to define the source party. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Target : **GenericIdentityDep** Public *Details:* This field is used to define the target party. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
ValidityPeriod : **DateTimeBound** Public *Details:* This field can be used to enforce a time validity over all assigned rights. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

OPERATIONS
Add () : **void** Public *Details:* This function can be used to add a membership right /relation.
Eval () : **void** Public *Details:* This function can be used to evaluate a function under the context of all applicable rights.
Filter (SrcDst : **GenericIdentityDep**) : **MembershipInteractionsDep** Public *Details:* This static function can be used to filter memberships based on a source or target party identity.
Remove () : **void** Public *Details:* This function can be used to remove a membership right /relation.

A.1.12.3 MembershipInteractionsDep

Class in package '4.3.12 Memberships'

Details: Objects whom reference this class expect an object that realizes the *MembershipInteractions* Interface. This class is a leaf and is only intended to serve as a data type.

CONNECTORS
Usage Source -> Destination From: : MembershipInteractionsDep : Class , Public To: MembershipInteractions : ProvidedInterface , Public

A.1.12.4 MembershipInteractionsRealization

Class in package '4.3.12 Memberships'

Details: This realization can be used to implement custom functions to handle basic membership interactions (such as establishing, validating and revoking them). This interface should not be used to dictate complex agent behavior, rather to demonstrate that a relation exists.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from MembershipInteractionsRealization to «interface» iMembershipInteractions
Realization from MembershipInteractionsRealization to MembershipInteractions

A.1.12.5 MembershipMapDep

Class in package '4.3.12 Memberships'

Details: Objects whom reference this class expect an object that realizes the *MembershipMap* Interface. This class is a leaf and is only intended to serve as a data type.

CONNECTORS
Usage Source -> Destination From: : MembershipMapDep : Class , Public To: MembershipMap : ProvidedInterface , Public

A.1.12.6 MembershipMapRealization

Class in package '4.3.12 Memberships'

Details: This realization can be used to implement custom membership tracking systems among a fixed set of parties

OUTGOING STRUCTURAL RELATIONSHIPS
Realization from MembershipMapRealization to MembershipMap
Generalization from MembershipMapRealization to «interface» iMembershipMap

A.1.12.7 Memberships

Class in package '4.3.12 Memberships'

Details: This class represents any generic membership that can exist between two parties. The class depends on an dependent interface to map each of the multiple relations that may exist in between a given Destiny/Source pair of parties.

ATTRIBUTES
DstEntity : **GenericIdentityDep** Private *Details:* This represents the source entity to which this memberships apply. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
RelationMap : **MembershipMapDep** Private *Details:* This represent all memberships that exists in between the given parties. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
SrcEntity : **GenericIdentityDep** Private *Details:* This represents the destiny entity to which this memberships apply. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

OPERATIONS
CreateMembership () : **void** Public *Details:* This is a static function that can be used to create a new membership in between the given parties. It internally calls the *Join* function defined by the membership provider method.
DeleteMembership () : **void** Public *Details:* This is a static function that can be used to delete/revoke a membership in between the given parties. It internally calls the *remove* function defined by the membership provider method.
Eval () : **void** Public

OPERATIONS
Details: This function can be used to evaluate an action using the membership interface.

A.1.12.8 Rights

Class in package '4.3.12 Memberships'

CONNECTORS
Dependency Source -> Destination From: : Rights : Class , Public To: AvailableRights : Enumeration , Public
Dependency Source -> Destination From: : iMembershipMap : Class , Public To: Rights : Class , Public

ATTRIBUTES
isValid : **Boolean** Private *Details:*
Right : **AvailableRights** Private *Details:*
ValidityPeriod : **DateTimeBound** Private *Details:*

A.1.12.9 AvailableRights

Enumeration in package '4.3.12 Memberships'

CONNECTORS
Dependency Source -> Destination From: : Rights : Class , Public To: AvailableRights : Enumeration , Public

ENUMERATION:
SubmitX
ReceiveY

A.1.12.10 Relations

Enumeration in package '4.3.12 Memberships'

CONNECTORS
Dependency Source -> Destination From: : iMembershipMap : Class , Public To: Relations : Enumeration , Public

ENUMERATION:
Bussiness
Supplier
Peer
Competitor

A.1.13 Summary of basic interfaces

This section summarizes the basic interfaces provided by this report, the interfaces are generic and can be used to support most of the data and communication needs of grid applications (and specifically TES systems). The provided interfaces remain at the high-level, but still capture most of the common requirements and functionalities that will require different modules to communicate, creating a highly-interoperable network.

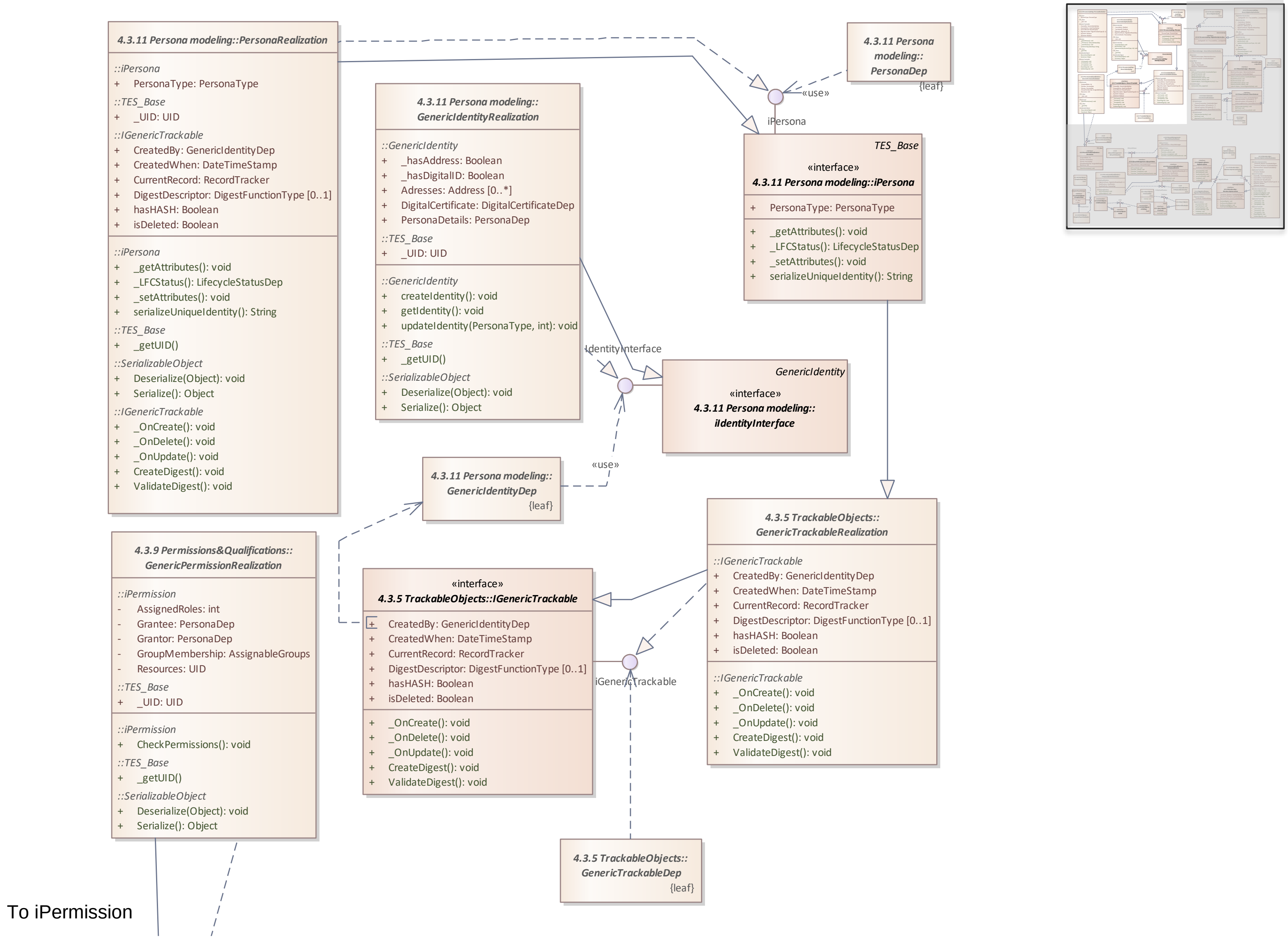
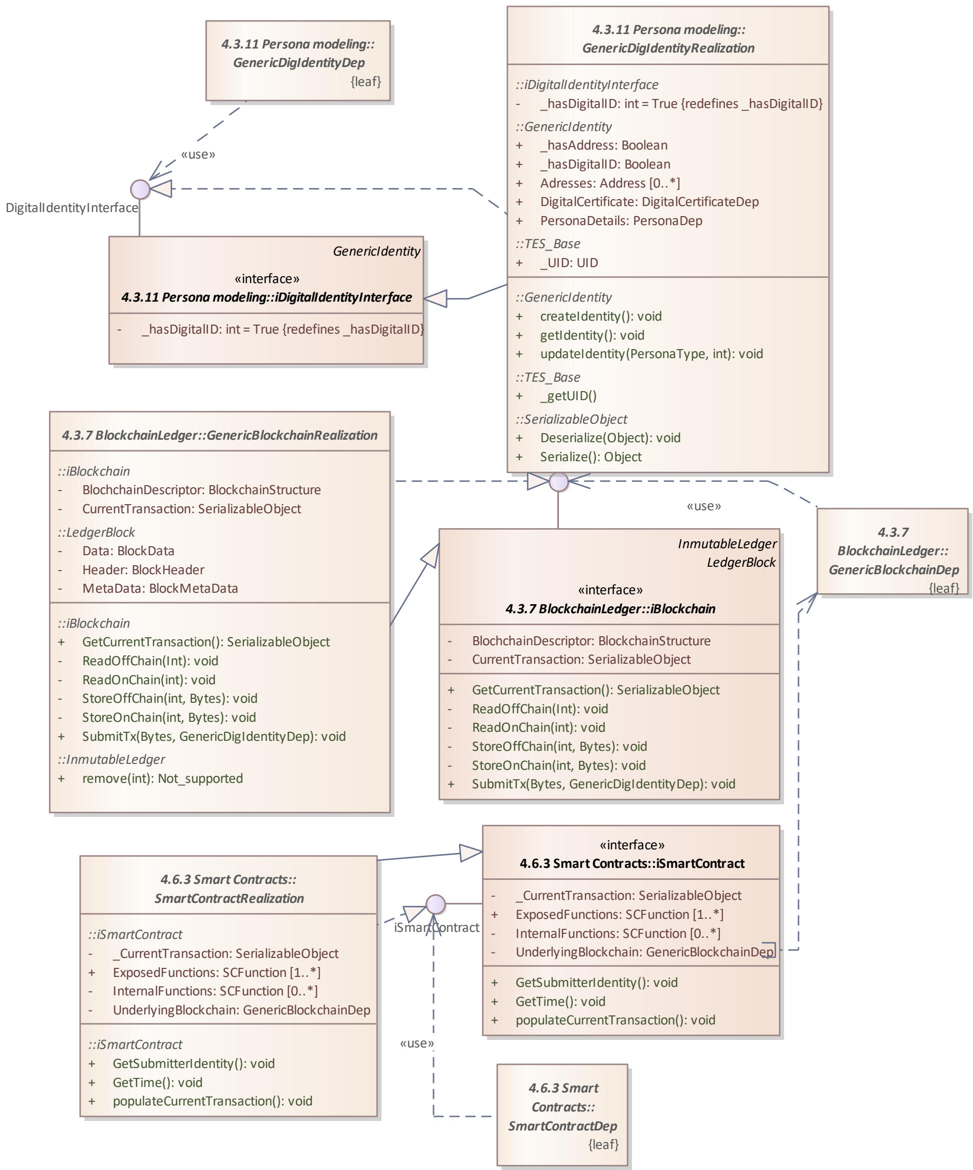


Figure 61. Overview of the BasicInterface’s package components (Top-left view).



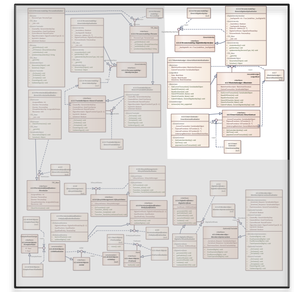
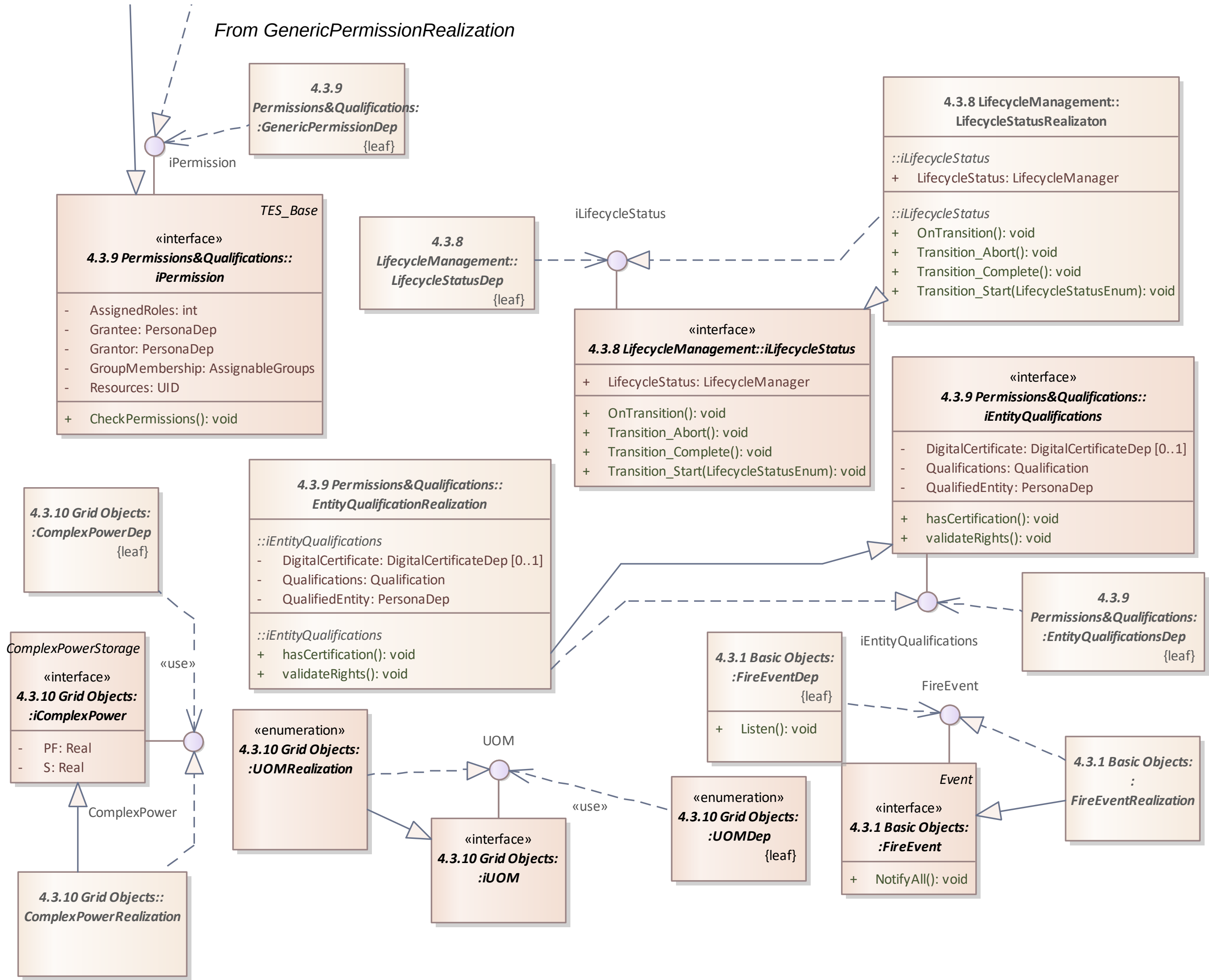


Figure 62. Overview of the BasicInterface's package components (Top-right view).



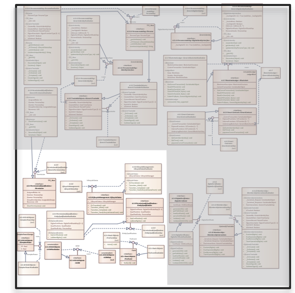
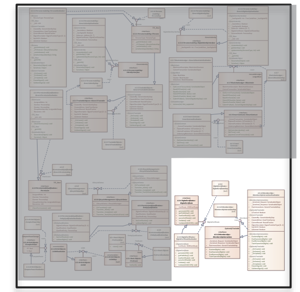


Figure 63. Overview of the BasicInterface's package components (Bottom-left view).



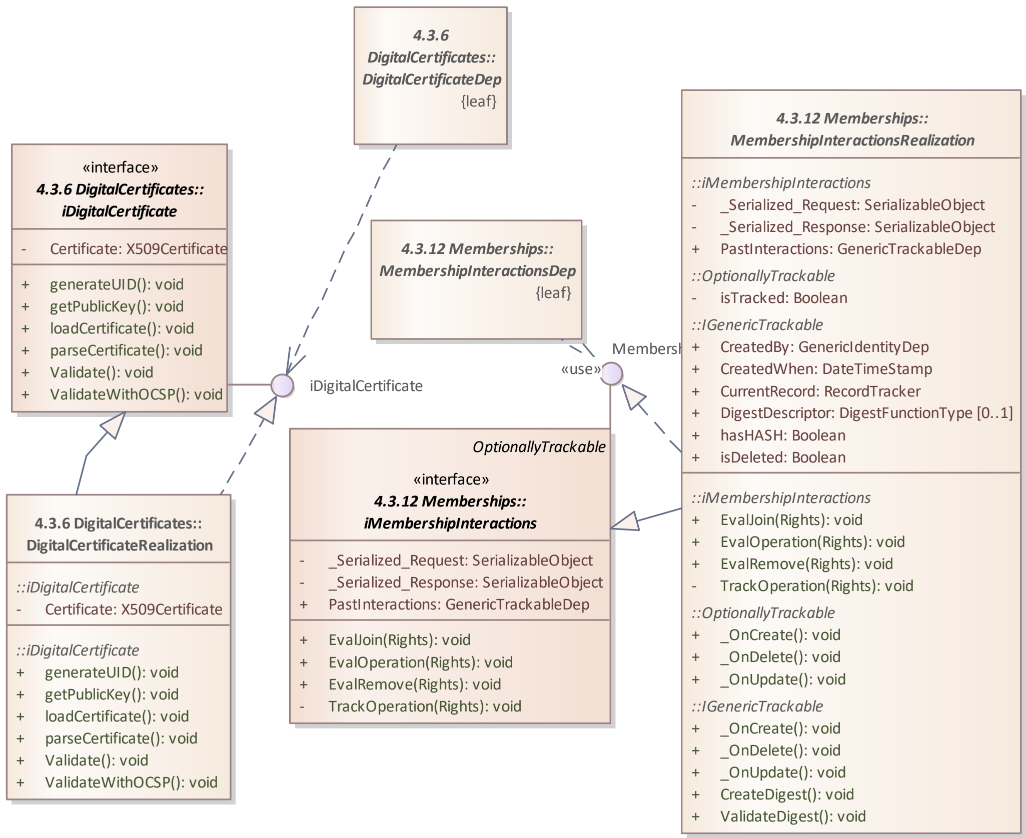


Figure 64. Overview of the BasicInterface's package components (Bottom-right view).

A.2 Resources and Participants modeling

In this section an overview of templates that can be used to model a large variety of grid devices and participants is introduced. These resources are the main building block of any TES-based application and represent the main contribution of the presented work.

A.2.1 Resources

This package documents a variety of classes that work together to represent grid equipment and expose it to a transactive system. The provided interfaces enable to abstract the different levels of interactions that are expected to occur within a TES. The proposed models have the ability to account for active as well as "dumb" devices, exposing only capable equipment as a grid resource. From this point forward, grid resources can be controlled and managed by dedicated transactive agents.

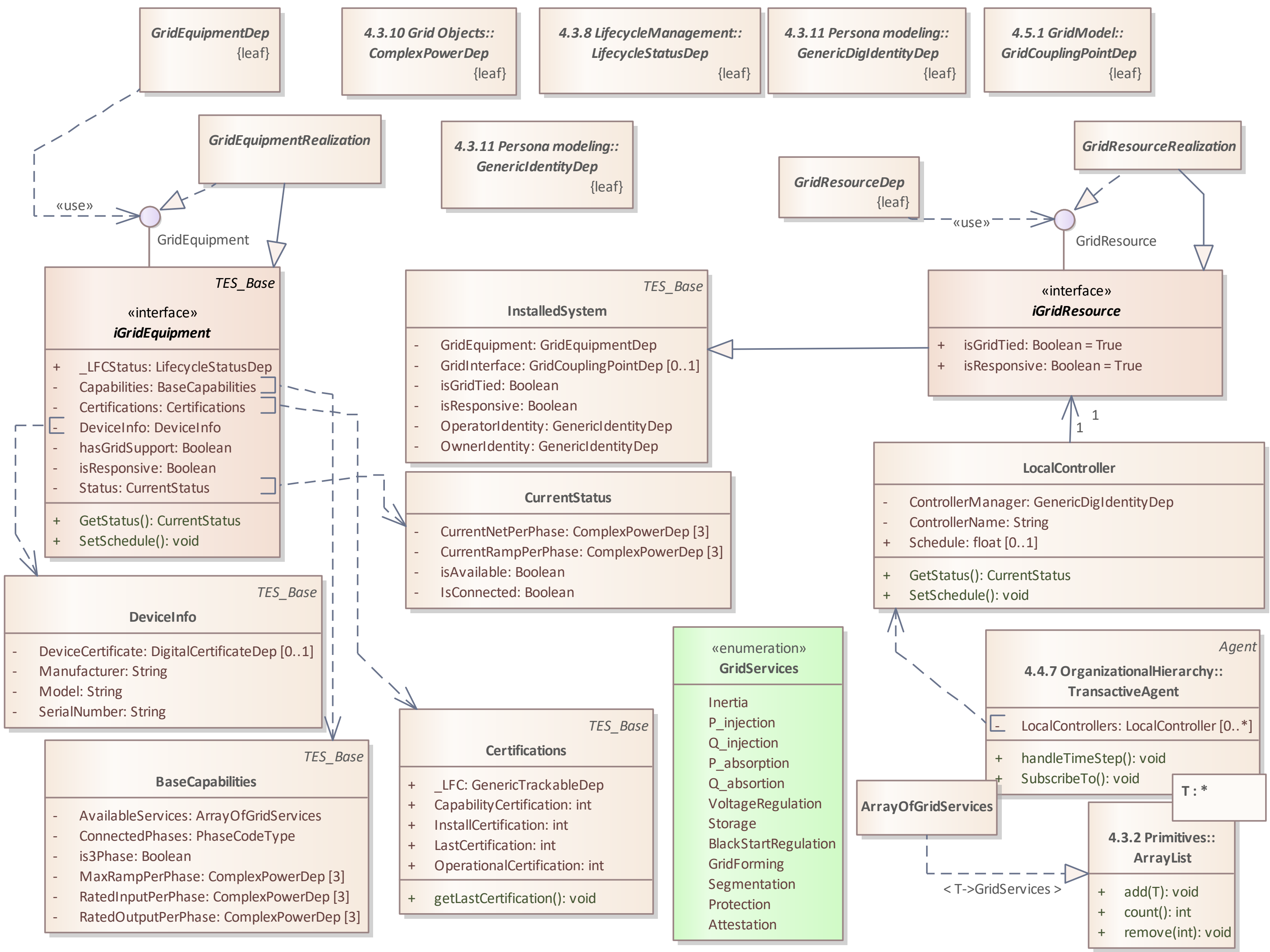


Figure 65. Overview of the Resources' package components

A.2.1.1 ArrayOfGridServices

Class in package '4.4.1 Resources'

OUTGOING STRUCTURAL RELATIONSHIPS
Realization from ArrayOfGridServices to ArrayList

A.2.1.2 BaseCapabilities

Class in package '4.4.1 Resources'

Details: This object captures the basic attributes of a device that has an energy consumption/generation interface. This object has been expanded from the definition given in the "tiger" model.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from BaseCapabilities to TES_Base

CONNECTORS
Dependency Source -> Destination From: : iGridEquipment : Class , Public To: BaseCapabilities : Class , Public

ATTRIBUTES
AvailableServices : **ArrayOfGridServices** Private *Details:* This field encodes all grid services that a device can provide. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
ConnectedPhases : **PhaseCodeType** Private *Details:* This field can be used to indicate the phases that are actually connected to an electrical network. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
is3Phase : **Boolean** Private *Details:* This flag can be used to determine if the equipment under question is intended to operate on 3 phases. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
MaxRampPerPhase : **ComplexPowerDep** Private *Details:* This field can be used to determine the maximum ramping rate for the resource. More complex devices can provide an XY curve to capture more dynamic ramping characteristics. Multiplicity: (3, Allow duplicates: 0, Is ordered: False)
RatedInputPerPhase : **ComplexPowerDep** Private *Details:* This represents the amount of complex power the device can absorb per unit of time (e.g., per hour) Multiplicity: (3, Allow duplicates: 0, Is ordered: False)
RatedOutputPerPhase : **ComplexPowerDep** Private *Details:* This field can be used to indicate the maximum output capabilities of the equipment being captured. This quantity is assumed to be per unit of time (e.g., and hour). Multiplicity: (3, Allow duplicates: 0, Is ordered: False)

A.2.1.3 Certifications

Class in package '4.4.1 Resources'

Details: This object can be used to capture an agents certification/inspection results. A device should refrain from operate unless an operational certification has been issued.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from Certifications to TES_Base

CONNECTORS
Dependency Source -> Destination From: : iGridEquipment : Class , Public To: Certifications : Class , Public

ATTRIBUTES
_LFC : **GenericTrackableDep** Public *Details:* This field can be used to track the certifications history. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
CapabilityCertification : **int** Public *Details:* This field is intended to hold a capability certification. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
InstallCertification : **int** Public *Details:* This field is intended to hold a install certification. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
LastCertification : **int** Public *Details:* This field is intended to hold a reference to the latest certification. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
OperationalCertification : **int** Public *Details:* This field is intended to hold an operational certification. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

OPERATIONS
getLastCertification () : **void** Public *Details:* This field is intended to return the latest certification available.

A.2.1.4 CurrentStatus

Class in package '4.4.1 Resources'

Details: This class represents the current device status. It should not be committed to the blockchain, rather an "on-demand" callback should be implemented.

CONNECTORS
Dependency Source -> Destination From: : iGridEquipment : Class , Public To: CurrentStatus : Class , Public

ATTRIBUTES
CurrentNetPerPhase : **ComplexPowerDep** Private *Details:* This field represents the current net power output (- for loads). Multiplicity: (3, Allow duplicates: 0, Is ordered: False)
CurrentRampPerPhase : **ComplexPowerDep** Private *Details:* This represents the current ramping rate that the device is executing either to a scheduled event or due to capacity constraints. Multiplicity: (3, Allow duplicates: 0, Is ordered: False)
isAvailable : **Boolean** Private *Details:* This field represents the dynamic ability of some systems to momentarily stop participating as a responsive system. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
IsConnected : **Boolean** Private *Details:* This field can be used to determine if the device is connected to the grid or has been isolated. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.2.1.5 DeviceInfo

Class in package '4.4.1 Resources'

Details: This object can be used to describe the basic properties of a device/equipment.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from DeviceInfo to TES_Base

CONNECTORS
Dependency Source -> Destination From: : iGridEquipment : Class , Public To: DeviceInfo : Class , Public
Dependency Source -> Destination From: : PVCell : Class , Public To: DeviceInfo : Class , Public

ATTRIBUTES
DeviceCertificate : **DigitalCertificateDep** Private *Details:* This optional field can be used to store a device digital certificate (if provided/generated). Multiplicity: (0..1, Allow duplicates: 0, Is ordered: False)
Manufacturer : **String** Private *Details:* This represents the manufacturer/vendor that developed, assembled or sold the equipment. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Model : **String** Private *Details:* This field represents the model, models are manufacturer's specific. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
SerialNumber : **String** Private *Details:* This field represents a unique number that represents a device. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.2.1.6 GridEquipmentDep

Class in package '4.4.1 Resources'

Details: Objects whom reference this class expect an object that realizes the *GridEquipment* Interface. This class is a leaf and is only intended to serve as a data type.

CONNECTORS
Usage Source -> Destination From: : GridEquipmentDep : Class , Public To: GridEquipment : ProvidedInterface , Public

A.2.1.7 GridEquipmentRealization

Class in package '4.4.1 Resources'

Details: This abstract class implements the *GridEquipment* interface which allows to implement grid devices that rely on power delivery. Classes derived from this class should satisfy all of the service and data requirements.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from GridEquipmentRealization to «interface» iGridEquipment
Realization from GridEquipmentRealization to GridEquipment

A.2.1.8 GridResourceDep

Class in package '4.4.1 Resources'

CONNECTORS
Usage Source -> Destination From: : GridResourceDep : Class , Public To: GridResource : ProvidedInterface , Public

A.2.1.9 GridResourceRealization

Class in package '4.4.1 Resources'

Details: This realization can serve as a base to implement different types of grid resources that are either responsive or can provide on-demand support services.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from GridResourceRealization to «interface» iGridResource
Realization from GridResourceRealization to GridResource

A.2.1.10 iGridEquipment

Class «interface» in package '4.4.1 Resources'

Details: This is the base class for representing grid equipment, it can hold capability information, generic device info, as well as *CurrentStatus* variables.

STRUCTURAL PART OF iGridEquipment
GridEquipment : ProvidedInterface

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from «interface» iGridEquipment to TES_Base

CONNECTORS
Dependency Source -> Destination From: : iGridEquipment : Class , Public To: DeviceInfo : Class , Public
Dependency Source -> Destination From: : iGridEquipment : Class , Public To: Certifications : Class , Public
Dependency Source -> Destination From: : iGridEquipment : Class , Public To: BaseCapabilities : Class , Public
Dependency Source -> Destination From: : iGridEquipment : Class , Public To: CurrentStatus : Class , Public
Dependency Source -> Destination From: : iGridEquipment : Class , Public To: LifecycleManager : Class , Public

ATTRIBUTES
_LFCStatus : **LifecycleStatusDep** Public *Details:* This field can be used to track an asset lifecycle status. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Capabilities : **BaseCapabilities** Private *Details:* This field describes the electrical capabilities of the device in question. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Certifications : **Certifications** Private *Details:* This field contains the authorizations or certifications necessary for operating the device. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
DeviceInfo : **DeviceInfo** Private *Details:* This field contains the device's basic identifying information. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
hasGridSupport : **Boolean** Private *Details:* This flag is used to indicate that a resource can provide some type of grid support services to the electrical system. This may include demand response, voltage regulation, remote measuring capabilities, etc. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
isResponsive : **Boolean** Private *Details:* This flag is used to indicate that a resource is responsive in its energy demand. Most agents will require a responsive device to operate. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Status : **CurrentStatus** Private *Details:* This field contains a reference to t Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

OPERATIONS
GetStatus () : **CurrentStatus** Public *Details:*
SetSchedule () : **void** Public *Details:* This function represents a very powerful function that can set the output power levels within the bounds of *RatedInput* and *RatedOutput*, at the *MaxRamp* rate.

A.2.1.11 InstalledSystem

Class in package '4.4.1 Resources'

Details: This object helps to map a device to a grid interconnection point once it gets installed. Notice that some devices can be installed but remain off-grid.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from InstalledSystem to TES_Base

CONNECTORS
Dependency Source -> Destination From: : InstalledSystemIdentity : Class , Public To: InstalledSystem : Class , Public

ATTRIBUTES
GridEquipment : **GridEquipmentDep** Private *Details:* This is a reference to the equipment being considered as installed. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
GridInterface : **GridCouplingPointDep** Private *Details:* This is an OPTIONAL argument that can be used to describe the interconnection point. Multiplicity: (0..1, Allow duplicates: 0, Is ordered: False)

ATTRIBUTES
isGridTied : **Boolean** Private *Details:* This flag can be used to determine if the equipment is connected to the grid. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
isResponsive : **Boolean** Private *Details:* This flag can be used to determine if the device is considered responsive. Note that resource availability does not affect the responsive state in this context. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
OperatorIdentity : **GenericIdentityDep** Private *Details:* This field can be used to define the identity of the entity responsible for its operation. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
OwnerIdentity : **GenericIdentityDep** Private *Details:* This field can be used to define the identity of the entity that owns the device, which may not be the entity responsible for its operation. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.2.1.12 LocalController

Class in package '4.4.1 Resources'

Details: This local controller model provide local intelligence to a resource. such intelligence is not limited to load/generation controllers but also to voltage and protection devices, this object was adapted from the model found in the Tiger's team report.

OUTGOING STRUCTURAL RELATIONSHIPS
Realization from LocalController to iResourcePhysicalStatus
Realization from LocalController to iLocalControl

CONNECTORS
Dependency Source -> Destination From: : LocalController : Class , Public To: PowerGen : Interface , Public
Dependency Source -> Destination From: : TransactiveAgent : Class , Public To: LocalController : Class , Public

ATTRIBUTES
ControllerManager : **GenericDigIdentityDep** Private *Details:* This fields represents the identity responsible for managing this controller. Due to its operational nature, it is expected that this identity has a digital representation. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
ControllerName : **String** Private *Details:* This field represents the controller's name Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Schedule : **float** Public *Details:* This field can be used to deploy an schedule of tasks. Multiplicity: (0..1, Allow duplicates: 0, Is ordered: False) Constraints: The "quantity" that makes up the PowerRealQuantity of the PowerMeasurementsSet shall be greater than or equal to 0. : Invariant

ASSOCIATIONS	
Association (direction: Source -> Destination)	
Source: Public (Class) LocalController	Target: Public upRamp (Class) PowerRampSegmentType

ASSOCIATIONS	
Association (direction: Source -> Destination)	
Source: Public (Class) LocalController	Target: Public downRamp (Class) PowerRampSegmentType
Association (direction: Source -> Destination)	
Source: Public (Class) LocalController Cardinality: [1]	Target: Public (Class) iGridResource «interface» Cardinality: [1]
Association (direction: Source -> Destination)	
Source: Public (Class) LocalController	Target: Public demandLimits (Class) PowerRatings
Association (direction: Source -> Destination)	
Source: Public (Class) SupervisoryController	Target: Public (Class) LocalController Cardinality: [1..*]

OPERATIONS
GetStatus () : **CurrentStatus** Public *Details:* This function is intended to provide details on behalf of the resource to higher hierarchy systems such as a transactive agent or a system-level controller.
SetSchedule () : **void** Public *Details:*

A.2.1.13 iGridResource

Class «interface» in package '4.4.1 Resources'

Details: This interface can be used to indicate an equipment that is connected to the grid and can provide responsive features. Such intelligence is not limited to load/generation elements but also to devices that can provide ancillary services or help in grid operations (e.g. protection devices). This object was adapted from the model found in the Tiger's team report.

STRUCTURAL PART OF iGridResource
GridResource : ProvidedInterface

OUTGOING STRUCTURAL RELATIONSHIPS
Realization from «interface» iGridResource to iResourceControl
Generalization from «interface» iGridResource to InstalledSystem
Realization from «interface» iGridResource to iResourcePhysical
Realization from «interface» iGridResource to iWeatherData

ATTRIBUTES
isGridTied : **Boolean** Public = True

ATTRIBUTES
Details: This field is used to indicate a requirement for devices to be connected, before being considered as a grid resource. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
isResponsive : **Boolean** Public = True *Details:* This field is used to indicate a requirement for devices to be responsive, before being considered as a grid resource. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

ASSOCIATIONS	
Association (direction: Source -> Destination)	
Source: Public (Class) iGridResource «interface»	Target: Public power (Class) Power
Association (direction: Source -> Destination)	
Source: Public (Class) iGridResource «interface»	Target: Public voltage (Class) Voltage
Association (direction: Source -> Destination)	
Source: Public (Class) iGridResource «interface»	Target: Public impedance (Class) Impedance
Association (direction: Bi-Directional)	
Source: Public (Class) iGridResource «interface»	Target: Private (Class) Grid
Association (direction: Source -> Destination)	
Source: Public (Class) iGridResource «interface»	Target: Public current (Class) Current
Association (direction: Source -> Destination)	
Source: Public (Class) Weather	Target: Public (Class) iGridResource «interface»
Association (direction: Source -> Destination)	
Source: Public (Class) LocalController Cardinality: [1]	Target: Public (Class) iGridResource «interface» Cardinality: [1]
Association (direction: Source -> Destination)	
Source: Public (Class) SupervisoryController	Target: Public resources (Class) iGridResource «interface»

A.2.1.14 GridServices

Enumeration in package '4.4.1 Resources'

Details: This enumeration contains a sample of grid services that can be provided by grid resources. Individual equipment can select the services that it can provide, the quality of such services can be implemented by using a dedicated *GridServices* Interface (future work).

ENUMERATION:
Inertia
P_injection
Q_injection
P_absorption
Q_absortion
VoltageRegulation
Storage
BlackStartRegulation
GridForming
Segmentation
Protection
Attestation

A.2.2 Load Resources

This package contains an specialization of a grid resource, it illustrates the two main types of loads that are present on the grid. Both load models present a conformant interface to the *GridEquipment* requirements.

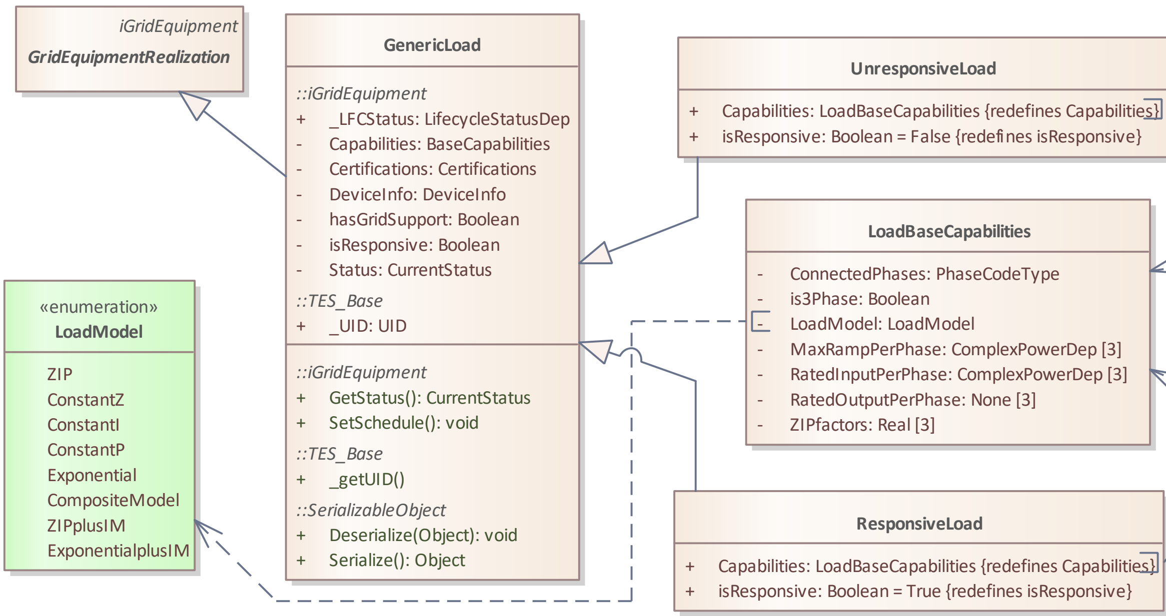


Figure 66. Overview of the LoadResources' package components

A.2.2.1 GenericLoad

Class in package '4.4.2 LoadResources'

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from GenericLoad to GridEquipmentRealization

A.2.2.2 LoadBaseCapabilities

Class in package '4.4.2 LoadResources'

Details: This object has been expanded from the definition given in the "tiger" model.

CONNECTORS
Dependency Source -> Destination From: : LoadBaseCapabilities : Class , Public To: LoadModel : Enumeration , Public
Dependency Source -> Destination From: : ResponsiveLoad : Class , Public To: LoadBaseCapabilities : Class , Public
Dependency Source -> Destination From: : UnresponsiveLoad : Class , Public To: LoadBaseCapabilities : Class , Public

ATTRIBUTES
ConnectedPhases : **PhaseCodeType** Private *Details:* This field can be used to indicate the phases that are actually connected to an electrical network. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
is3Phase : **Boolean** Private *Details:* This flag can be used to determine if the equipment under question is intended to operate on 3 phases. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
LoadModel : **LoadModel** Private *Details:*
MaxRampPerPhase : **ComplexPowerDep** Private *Details:* This field can be used to determine the maximum ramping rate for the resource. More complex devices can provide an XY curve to capture more dynamic ramping characteristics. Multiplicity: (3, Allow duplicates: 0, Is ordered: False)
RatedInputPerPhase : **ComplexPowerDep** Private *Details:* This represents the amount of complex power the device can absorb per unit of time (e.g., per hour) Multiplicity: (3, Allow duplicates: 0, Is ordered: False)
RatedOutputPerPhase : **None** Private *Details:* This field can be used to indicate the maximum output capabilities of the equipment being captured. This quantity is assumed to be per unit of time (e.g., and hour). Multiplicity: (3, Allow duplicates: 0, Is ordered: False)
ZIPfactors : **Real** Private *Details:* This field can be used to provide a custom mixture of ZIP components. The *LoadModel* should be set to ZIP if this field must be used. The sum of this factors must be equal to 1.0 Multiplicity: (3, Allow duplicates: 0, Is ordered: False)

A.2.2.3 ResponsiveLoad

Class in package '4.4.2 LoadResources'

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from ResponsiveLoad to GenericLoad

CONNECTORS
Dependency Source -> Destination From: : ResponsiveLoad : Class , Public To: LoadBaseCapabilities : Class , Public

ATTRIBUTES
Capabilities : **LoadBaseCapabilities** Public *Details:*
isResponsive : **Boolean** Public = True *Details:*

A.2.2.4 UnresponsiveLoad

Class in package '4.4.2 LoadResources'

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from UnresponsiveLoad to GenericLoad

CONNECTORS
Dependency Source -> Destination From: : UnresponsiveLoad : Class , Public To: LoadBaseCapabilities : Class , Public

ATTRIBUTES
Capabilities : **LoadBaseCapabilities** Public *Details:*
isResponsive : **Boolean** Public = False *Details:*

A.2.2.5 LoadModel

Enumeration in package '4.4.2 LoadResources'

Details: This enumeration represents the different types of loads that can be present on the system.

CONNECTORS
Dependency Source -> Destination From: : LoadBaseCapabilities : Class , Public To: LoadModel : Enumeration , Public

ENUMERATION:	
ZIP	A mixture of constant Impedance, Current and Power.
ConstantZ	
ConstantI	
ConstantP	
Exponential	
CompositeModel	
ZIPplusIM	
ExponentialplusIM	

A.2.3 IBR-Based Generation Resources

This diagram is an specialization of a grid resource, it enables end users to model the features of an Inverter-Based generator. It documents specific examples to model PV-based and wind-based resources which can further refined to satisfy the data capturing needs of the end use.

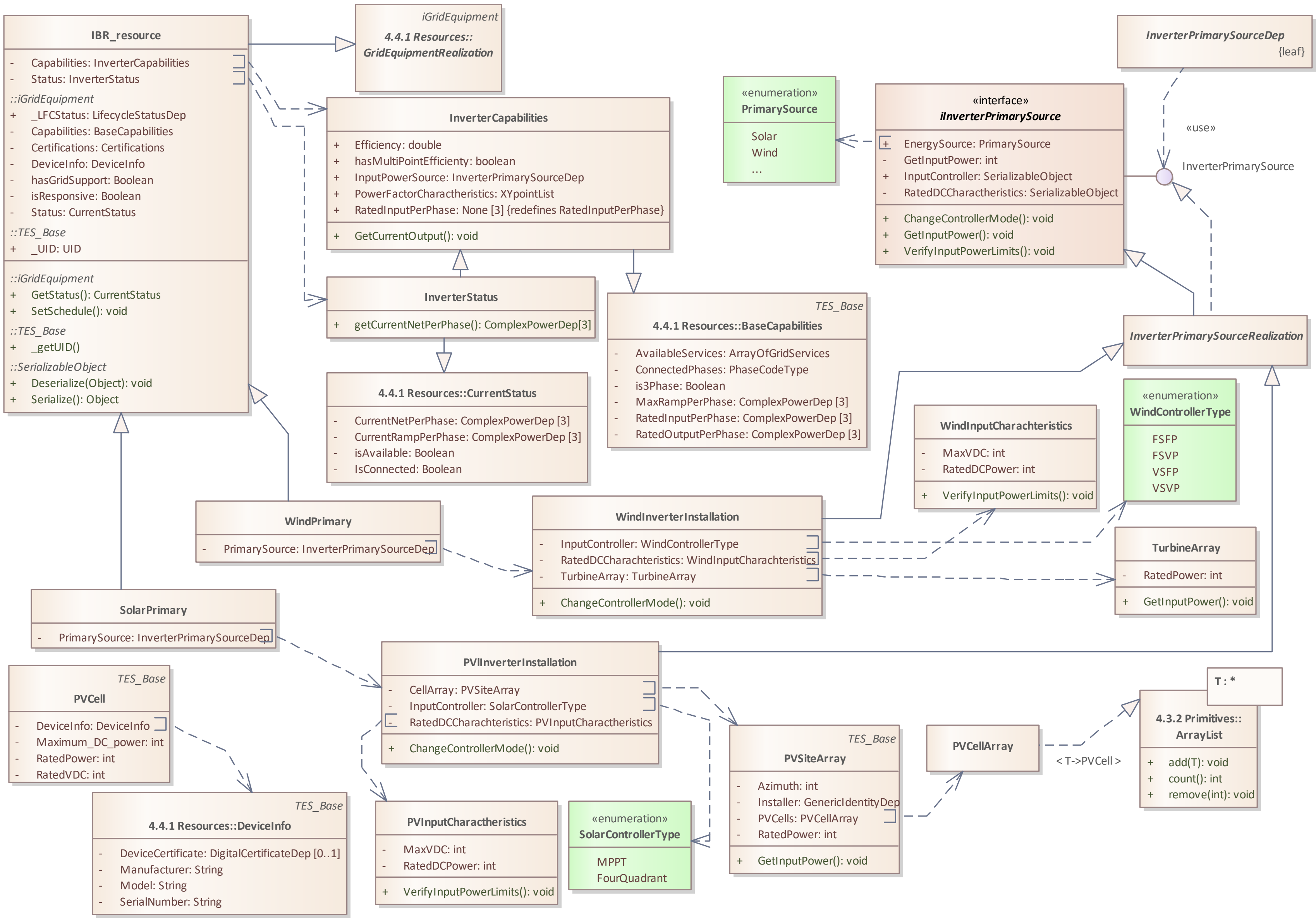


Figure 67. Overview of the IBR-BasedGenerationResources' package components

A.2.3.1 IBR_resource

Class in package '4.4.3 IBR-BasedGenerationResources'

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from IBR_resource to GridEquipmentRealization

CONNECTORS
Dependency Source -> Destination From: : IBR_resource : Class , Public To: InverterCapabilities : Class , Public
Dependency Source -> Destination From: : IBR_resource : Class , Public To: InverterStatus : Class , Public

ATTRIBUTES
Capabilities : **InverterCapabilities** Private *Details:* This field overrides the default capability model, exposing more low-level details to the local controller. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Status : **InverterStatus** Private *Details:* This field overrides the default status object, exposing more low-level details to the local controller. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.2.3.2 iInverterPrimarySource

Class «interface» in package '4.4.3 IBR-BasedGenerationResources'

Details: This abstract class describes the interface requirements for an inverter-based primary energy source. The interface provides standard interfaces for obtaining the power characteristics and defining the operational mode of the DC-side components

STRUCTURAL PART OF iInverterPrimarySource
InverterPrimarySource : ProvidedInterface

CONNECTORS
Dependency Source -> Destination From: : iInverterPrimarySource : Class , Public To: PrimarySource : Enumeration , Public

ATTRIBUTES
EnergySource : **PrimarySource** Public *Details:*
GetInputPower : **int** Private *Details:*
InputController : **SerializableObject** Public *Details:*
RatedDCCharactheristics : **SerializableObject** Private *Details:*

OPERATIONS
ChangeControllerMode () : **void** Public *Details:*
GetInputPower () : **void** Public *Details:*
VerifyInputPowerLimits () : **void** Public *Details:*

A.2.3.3 InverterCapabilities

Class in package '4.4.3 IBR-BasedGenerationResources'

Details: This inverter model capability description is based on *InverterModel* introduced by the NIST-Challenge, within the "Tiger" model.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from InverterCapabilities to BaseCapabilities

CONNECTORS
Dependency Source -> Destination From: : IBR_resource : Class , Public To: InverterCapabilities : Class , Public

ATTRIBUTES
Efficiency : **double** Public *Details:* Efficiency of the inverter. This is assigned by inverter_type and cannot be overridden at this time. Unit: unit Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
hasMultiPointEfficiency : **boolean** Public *Details:* This is use multipoint efficiency. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
InputPowerSource : **InverterPrimarySourceDep** Public *Details:*
PowerFactorCharactheristics : **XYpointList** Public *Details:*
RatedInputPerPhase : **None** Public *Details:* This property refers to a system that cannot absorb power from the AC/grid side. Multiplicity: (3, Allow duplicates: 0, Is ordered: False)

ASSOCIATIONS	
Association (direction: Source -> Destination)	
Source: Public (Class) InverterCapabilities	Target: Public (Class) Solar
Association (direction: Source -> Destination)	
Source: Public (Class) Triplex_meter	Target: Public invertor (Class) InverterCapabilities
Association (direction: Source -> Destination)	
Source: Public (Class) House	Target: Public (Class) InverterCapabilities

ASSOCIATIONS

OPERATIONS
GetCurrentOutput () : **void** Public *Details:*

A.2.3.4 InverterPrimarySourceDep

Class in package '4.4.3 IBR-BasedGenerationResources'

Details: Objects whom reference this class expect an object that realizes the *InverterPrimarySource* Interface. This class is a leaf and is only intended to serve as a data type.

CONNECTORS
Usage Source -> Destination From: : InverterPrimarySourceDep : Class , Public To: InverterPrimarySource : ProvidedInterface , Public

A.2.3.5 InverterPrimarySourceRealization

Class in package '4.4.3 IBR-BasedGenerationResources'

Details: This abstract class implements the *InverterPrimarySource* interface which allows to model the DC power source of an inverter. Classes derived from this class should satisfy all of the service and data requirements.

OUTGOING STRUCTURAL RELATIONSHIPS
Realization from InverterPrimarySourceRealization to InverterPrimarySource
Generalization from InverterPrimarySourceRealization to «interface» iInverterPrimarySource

A.2.3.6 InverterStatus

Class in package '4.4.3 IBR-BasedGenerationResources'

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from InverterStatus to InverterCapabilities
Generalization from InverterStatus to CurrentStatus

CONNECTORS
Dependency Source -> Destination From: : IBR_resource : Class , Public To: InverterStatus : Class , Public

OPERATIONS
getCurrentNetPerPhase () : **ComplexPowerDep[3]** Public *Details:*

A.2.3.7 PVCell

Class in package '4.4.3 IBR-BasedGenerationResources'

Details: This sample object illustrates the interface abilities to capture low-level details of an inverter-based, PV system, while offering an standardized interface that can enable interoperability.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from PVCell to TES_Base

CONNECTORS
Dependency Source -> Destination From: : PVCell : Class , Public To: DeviceInfo : Class , Public

ATTRIBUTES
DeviceInfo : **DeviceInfo** Private *Details:* This field captures the solar cell manufacturer, model and serial number. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Maximum_DC_power : **int** Private *Details:* This field captures the power characteristics of a PV module. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
RatedPower : **int** Private *Details:* This field captures the power characteristics of a PV module. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
RatedVDC : **int** Private *Details:* This field captures the power characteristics of a PV module. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.2.3.8 PVCellArray

Class in package '4.4.3 IBR-BasedGenerationResources'

OUTGOING STRUCTURAL RELATIONSHIPS
Realization from PVCellArray to ArrayList

CONNECTORS
Dependency Source -> Destination From: : PVSiteArray : Class , Public To: PVCellArray : Class , Public

A.2.3.9 PVInputCharactheristics

Class in package '4.4.3 IBR-BasedGenerationResources'

Details: This sample object provides sample power characteristics that are applicable to PV-based systems.

CONNECTORS
Dependency Source -> Destination From: : PVIInverterInstallation : Class , Public To: PVInputCharactheristics : Class , Public

ATTRIBUTES
MaxVDC : **int** Private *Details:*
RatedDCPower : **int** Private *Details:*

OPERATIONS
VerifyInputPowerLimits () : **void** Public *Details:*

A.2.3.10 PVIInverterInstallation

Class in package '4.4.3 IBR-BasedGenerationResources'

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from PVIInverterInstallation to InverterPrimarySourceRealization

CONNECTORS
Dependency Source -> Destination From: : PVIInverterInstallation : Class , Public To: PVInputCharactheristics : Class , Public
Dependency Source -> Destination From: : PVIInverterInstallation : Class , Public To: PVSiteArray : Class , Public
Dependency Source -> Destination From: : PVIInverterInstallation : Class , Public To: SolarControllerType : Enumeration , Public
Dependency Source -> Destination From: : SolarPrimary : Class , Public To: PVIInverterInstallation : Class , Public
Dependency Source -> Destination From: : PowerGen : Interface , Public To: PVIInverterInstallation : Class , Public

ATTRIBUTES
CellArray : **PVSiteArray** Private *Details:* This is a interface-specific field that can be used to describe the underlying power generating source Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
InputController : **SolarControllerType** Private *Details:* This field is used to define the available control modes for this device. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
RatedDCCharachteristics : **PVInputCharactheristics** Private *Details:* This field is used to describe the technical capabilities of the DC power source, along with methods to validate their limits. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

OPERATIONS
ChangeControllerMode () : **void** Public *Details:* This function can be invoked by the local controller to change the operational mode of the inverter.

A.2.3.11 PVSiteArray

Class in package '4.4.3 IBR-BasedGenerationResources'

Details: This object can be used to describe a PV-based installation from a physical perspective. A system like this could enable a centralized data-store to track installation permits as well as enable TES participation.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from PVSiteArray to TES_Base

CONNECTORS
Dependency Source -> Destination From: : PVSiteArray : Class , Public To: PVCellArray : Class , Public
Dependency Source -> Destination From: : PVIInverterInstallation : Class , Public To: PVSiteArray : Class , Public

ATTRIBUTES
Azimuth : **int** Private *Details:* This represents the horizontal angle of a PV solar array Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Installer : **GenericIdentityDep** Private *Details:* This field can be used to track the individual/company that performed the solar installation. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
PVCells : **PVCellArray** Private *Details:*
RatedPower : **int** Private *Details:*

OPERATIONS
GetInputPower () : **void** Public *Details:*

A.2.3.12 SolarPrimary

Class in package '4.4.3 IBR-BasedGenerationResources'

Details: This object represents a solar-based, inverter-based resource. It relies on custom interface to specify the primary energy source of the inverter.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from SolarPrimary to IBR_resource

CONNECTORS
Dependency Source -> Destination From: : SolarPrimary : Class , Public To: PVIInverterInstallation : Class , Public

ATTRIBUTES
PrimarySource : **InverterPrimarySourceDep** Private *Details:*

A.2.3.13 TurbineArray

Class in package '4.4.3 IBR-BasedGenerationResources'

Details: This structure can be used to define the power ratings of the device, along with installation-specific details (id desired).

CONNECTORS
Dependency Source -> Destination From: : WindInverterInstallation : Class , Public To: TurbineArray : Class , Public

ATTRIBUTES
RatedPower : **int** Private *Details:*

OPERATIONS
GetInputPower () : **void** Public *Details:*

A.2.3.14 WindInputCharachteristics

Class in package '4.4.3 IBR-BasedGenerationResources'

Details: This structure represents a sample construct that can be used to define a wind-based system capabilities.

CONNECTORS
Dependency Source -> Destination From: : WindInverterInstallation : Class , Public To: WindInputCharachteristics : Class , Public

ATTRIBUTES
MaxVDC : **int** Private *Details:*
RatedDCPower : **int** Private *Details:*

OPERATIONS
VerifyInputPowerLimits () : **void** Public *Details:*

A.2.3.15 WindInverterInstallation

Class in package '4.4.3 IBR-BasedGenerationResources'

Details: This sample object implements the InverterPrimarySource Interface to provide support for wind-based generation.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from WindInverterInstallation to InverterPrimarySourceRealization

CONNECTORS
Dependency Source -> Destination From: : WindInverterInstallation : Class , Public To: WindInputCharachteristics : Class , Public
Dependency Source -> Destination From: : WindInverterInstallation : Class , Public To: TurbineArray : Class , Public
Dependency Source -> Destination From: : WindInverterInstallation : Class , Public To: WindControllerType : Enumeration , Public
Dependency Source -> Destination From: : WindPrimary : Class , Public To: WindInverterInstallation : Class , Public

ATTRIBUTES
InputController : **WindControllerType** Private *Details:* This field is used to define the available control modes for this device. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
RatedDCCharachteristics : **WindInputCharachteristics** Private *Details:* This field is used to describe the technical capabilities of the DC power source, along with methods to validate their limits. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
TurbineArray : **TurbineArray** Private *Details:* This is a interface-specific field that can be used to describe the underlying power generating source Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

OPERATIONS
ChangeControllerMode () : **void** Public *Details:* This function can be invoked by the local controller to change the operational mode of the inverter.

A.2.3.16 WindPrimary

Class in package '4.4.3 IBR-BasedGenerationResources'

Details: This object represents a wind-based inverter-based resource. It relies on custom interface to specify the primary energy source of the inverter.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from WindPrimary to IBR_resource

CONNECTORS
Dependency Source -> Destination From: : WindPrimary : Class , Public To: WindInverterInstallation : Class , Public

ATTRIBUTES
PrimarySource : **InverterPrimarySourceDep** Private *Details:*

A.2.3.17 PrimarySource

Enumeration in package '4.4.3 IBR-BasedGenerationResources'

Details: This enumeration can be used to describe the primary energy source. This list is illustrative and can be expanded to suit the end-user needs.

CONNECTORS
Dependency Source -> Destination From: : iInverterPrimarySource : Class , Public To: PrimarySource : Enumeration , Public

ENUMERATION:
Solar
Wind
...

A.2.3.18 SolarControllerType

Enumeration in package '4.4.3 IBR-BasedGenerationResources'

Details: This object offers a PV-specific definition of the controller

CONNECTORS
Dependency Source -> Destination From: : PVlInverterInstallation : Class , Public To: SolarControllerType : Enumeration , Public

ENUMERATION:
MPPT
FourQuadrant

A.2.3.19 WindControllerType

Enumeration in package '4.4.3 IBR-BasedGenerationResources'

Details: This enumeration lists the common types of controllers available in wind-based generators.

CONNECTORS
Dependency Source -> Destination From: : WindInverterInstallation : Class , Public To: WindControllerType : Enumeration , Public

ENUMERATION:	
FSFP	Fixed-speed fixed-pitch
FSVP	Fixed-speed variable-pitch
VSFP	Variable-speed fixed-pitch
VSVP	Variable-speed variable-pitch

A.2.4 Rotational Generation Resources

This diagram is an specialization of a grid resource, it enables end users to capture the components of a traditional power plant. The interface can be used to provide (and extract) data from both the electromechanical energy conversion process, as well as the mechanism used to capture the mechanical energy. By including the most common energy generation processes (DER, Bulk, and storage) the provided templates can be applicable a wide variety of application scenarios. Potentially enable the participation of a wide variety of systems.

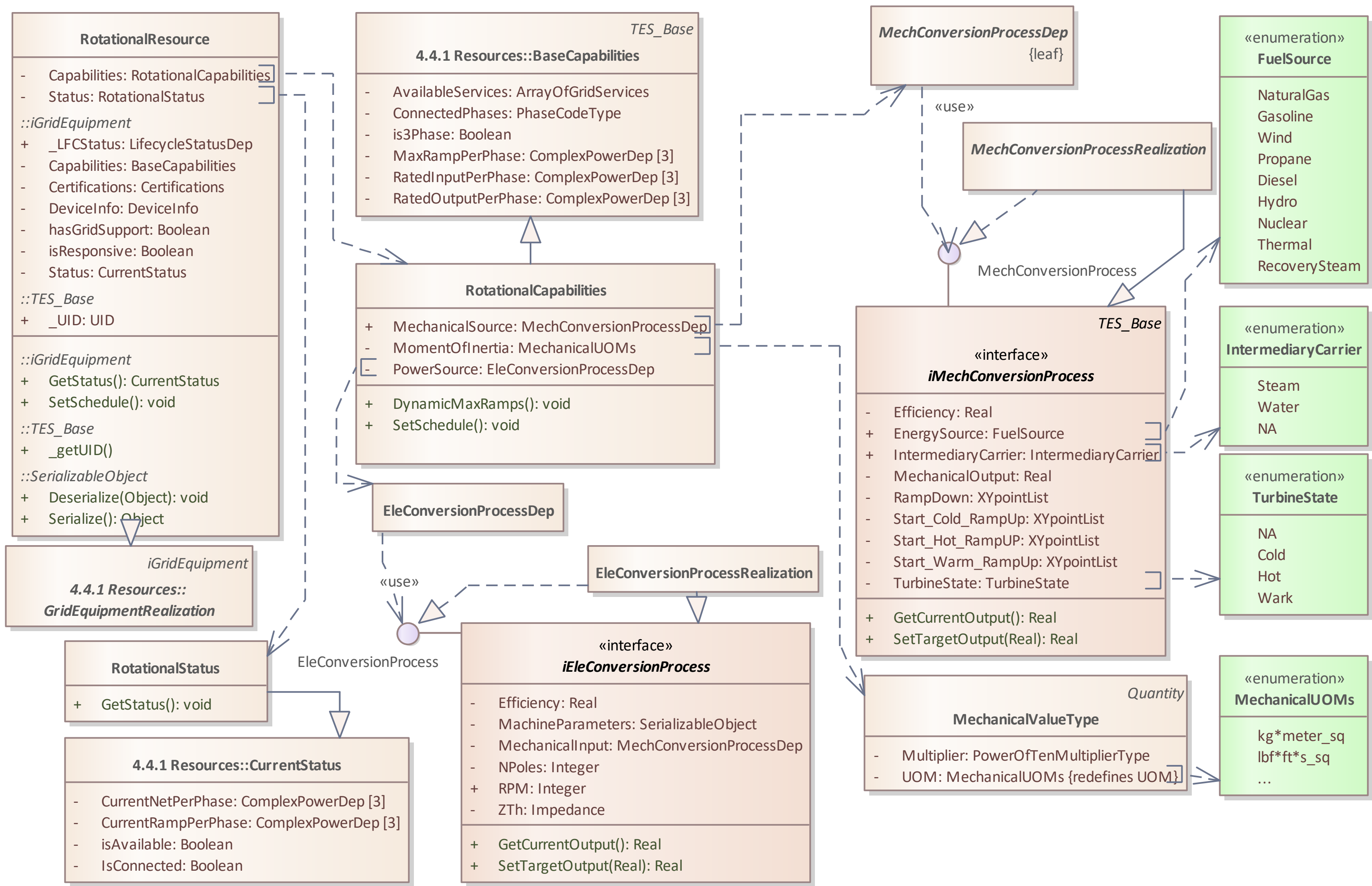


Figure 68. Overview of the RotationalGenerationResources' package components

A.2.4.1 EleConversionProcessDep

Class in package '4.4.4 RotationalGenerationResources'

Details: Objects that reference this class expect an object that realizes the *EleConversionProcessDep* Interface. This class is a leaf and is only intended to serve as a data type.

CONNECTORS
Usage Source -> Destination From: : EleConversionProcessDep : Class , Public To: EleConversionProcess : ProvidedInterface , Public
Dependency Source -> Destination From: : RotationalCapabilities : Class , Public To: EleConversionProcessDep : Class , Public

A.2.4.2 EleConversionProcessRealization

Class in package '4.4.4 RotationalGenerationResources'

Details: This abstract class implements the *EleConversionProcess* interface which allows to transform a mechanical rotational force into electricity. Classes derived from this class should satisfy all of the service and data requirements.

OUTGOING STRUCTURAL RELATIONSHIPS
Realization from EleConversionProcessRealization to EleConversionProcess
Generalization from EleConversionProcessRealization to «interface» iEleConversionProcess

A.2.4.3 iEleConversionProcess

Class «interface» in package '4.4.4 RotationalGenerationResources'

Details: This interface aggregates the necessary properties to define an electromechanical system. It is assumed that the machine operates as a generator but by setting the TargetOutput to a negative number a machine-like behavior can be achieved.

STRUCTURAL PART OF iEleConversionProcess
EleConversionProcess : ProvidedInterface
MechConversionProcess : ProvidedInterface

ATTRIBUTES
Efficiency : **Real** Private *Details:* This represents the systems efficiency. This number is only to serve as a general reference, since actual conversion rates can be variable across the operational range Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
MachineParameters : **SerializableObject** Private *Details:* This field is used to encode the machine characteristics so accurate electrical models can be build (e.g., to model a synchronous machine). Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
MechanicalInput : **MechConversionProcessDep** Private

ATTRIBUTES
Details: This field references the mechanical input that will be used to provide the input power. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
NPoles : **Integer** Private *Details:* This field can be used to store the number of poles on the machine. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
RPM : **Integer** Public *Details:* This field documents the nominal Revolutions Per Minute that the machine performs. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
ZTh : **Impedance** Private *Details:* This field stores the equivalent Thevenin impedance (from the internal components). Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

OPERATIONS
GetCurrentOutput () : **Real** Public *Details:* This function can be used to determine the current mechanical output.
SetTargetOutput (ValuekW : **Real**) : **Real** Public *Details:* This function can be used to schedule a system generation capacity (from a mechanical standpoint), it returns the number of hours that it will take to reach the requested state.

A.2.4.4 iMechConversionProcess

Class «interface» in package '4.4.4 RotationalGenerationResources'

STRUCTURAL PART OF iMechConversionProcess
MechConversionProcess : ProvidedInterface

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from «interface» iMechConversionProcess to TES_Base

CONNECTORS
Dependency Source -> Destination From: : iMechConversionProcess : Class , Public To: FuelSource : Enumeration , Public
Dependency Source -> Destination From: : iMechConversionProcess : Class , Public To: IntermediaryCarrier : Enumeration , Public
Dependency Source -> Destination From: : iMechConversionProcess : Class , Public To: TurbineState : Enumeration , Public

ATTRIBUTES
Efficiency : **Real** Private *Details:* This represents the systems efficiency. This number is only to serve as a general reference, since actual conversion rates can be variable across the operational range Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
EnergySource : **FuelSource** Public *Details:* This field encode the primary energy source. Multiple conversion objects must be instantiated if recovery mechanism are installed in series. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

ATTRIBUTES
IntermediaryCarrier : **IntermediaryCarrier** Public *Details:* This field identifies the intermediary mechanism used to move a turbine. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
MechanicalOutput : **Real** Private *Details:* This value represents the rated mechanical output. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
RampDown : **XYpointList** Private *Details:* This field can be used to encode the ramping down characteristics of a turbine-based system. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Start_Cold_RampUp : **XYpointList** Private *Details:* This field is used to encode the ramp up characteristics when the turbine is in a cold state. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Start_Hot_RampUP : **XYpointList** Private *Details:* This field is used to encode the ramp up characteristics when the turbine is in a hot state. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Start_Warm_RampUp : **XYpointList** Private *Details:* This field is used to encode the ramp up characteristics when the turbine is in a warm state. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
TurbineState : **TurbineState** Private *Details:* This field stores the current turbine state. This state along with the associated ramp up capabilities determines the time to reach a new state. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

OPERATIONS
GetCurrentOutput () : **Real** Public *Details:* This function can be used to determine the current mechanical output.
SetTargetOutput (ValuekW : **Real**) : **Real** Public *Details:* This function can be used to schedule a system generation capacity (from a mechanical standpoint), it returns the number of hours that it will take to reach the requested state.

A.2.4.5 MechanicalValueType

Class in package '4.4.4 RotationalGenerationResources'

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from MechanicalValueType to Quantity

CONNECTORS
Dependency Source -> Destination From: : MechanicalValueType : Class , Public To: MechanicalUOMs : Enumeration , Public
Dependency Source -> Destination From: : RotationalCapabilities : Class , Public To: MechanicalValueType : Class , Public

ATTRIBUTES
Multiplier : **PowerOfTenMultiplierType** Private *Details:*
UOM : **MechanicalUOMs** Private *Details:*

A.2.4.6 MechConversionProcessDep

Class in package '4.4.4 RotationalGenerationResources'

Details: Objects that reference this class expect an object that realizes the *MechConversionProcess* Interface. This class is a leaf and is only intended to serve as a data type.

CONNECTORS
Usage Source -> Destination From: : MechConversionProcessDep : Class , Public To: MechConversionProcess : ProvidedInterface , Public
Dependency Source -> Destination From: : RotationalCapabilities : Class , Public To: MechConversionProcessDep : Class , Public

A.2.4.7 MechConversionProcessRealization

Class in package '4.4.4 RotationalGenerationResources'

Details: This abstract class implements the *MechConversionProcess* interface which allows to capture a source of energy and translate it into a mechanical rotational force. Classes derived from this class should satisfy all of the service and data requirements.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from MechConversionProcessRealization to «interface» iMechConversionProcess
Realization from MechConversionProcessRealization to MechConversionProcess

A.2.4.8 RotatingMachineModel

Class in package '4.4.4 RotationalGenerationResources'

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from RotatingMachineModel to ManufacturerInfo

CONNECTORS
Dependency Source -> Destination From: : RotatingMachineModel : Class , Public To: FuelSource : Enumeration , Public
Dependency Source -> Destination From: : RotatingMachineModel : Class , Public To: FuelSource : Enumeration , Public
Dependency Source -> Destination From: : PowerGen : Interface , Public To: RotatingMachineModel : Class , Public

ATTRIBUTES
Efficiency : **int** Private *Details:*
Fuel1 : **int** Private *Details:*
Fuel1Mix : **int** Private *Details:*

ATTRIBUTES
Fuel2 : **int** Private *Details:*
UID : **int** Private *Details:*

A.2.4.9 RotationalCapabilities

Class in package '4.4.4 RotationalGenerationResources'

Details: This structure aggregates the mechanical energy capture process and the electromechanical generation process. The object provides functions that can dynamically update the ramping characteristics to help the local controller make informed decisions.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from RotationalCapabilities to BaseCapabilities

CONNECTORS
Dependency Source -> Destination From: : RotationalCapabilities : Class , Public To: MechConversionProcessDep : Class , Public
Dependency Source -> Destination From: : RotationalCapabilities : Class , Public To: MechanicalValueType : Class , Public
Dependency Source -> Destination From: : RotationalCapabilities : Class , Public To: EleConversionProcessDep : Class , Public
Dependency Source -> Destination From: : RotationalResource : Class , Public To: RotationalCapabilities : Class , Public

ATTRIBUTES
MechanicalSource : **MechConversionProcessDep** Public *Details:* This object represents the mechanical, energy-capturing process. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
MomentOfInertia : **MechanicalUOMs** Private *Details:* This value captures the rotational inertia that is available to the system. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
PowerSource : **EleConversionProcessDep** Private *Details:* This field references the electromechanical process used to transform mechanical energy into electrical energy. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

OPERATIONS
DynamicMaxRamps () : **void** Public *Details:* This function can automatically update the ramping rate based on the internal state of the mechanical interface.
SetSchedule () : **void** Public *Details:* This function can "schedule" a resources output capabilities across time.

A.2.4.10 RotationalResource

Class in package '4.4.4 RotationalGenerationResources'

Details: This class realizes a rotation machine specialization of the *GridEquipment* interface by extending the *GridEquipmentRealization* class. It overrides the *Capabilities* and *Status* fields.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from RotationalResource to GridEquipmentRealization

CONNECTORS
Dependency Source -> Destination From: : RotationalResource : Class , Public To: RotationalStatus : Class , Public
Dependency Source -> Destination From: : RotationalResource : Class , Public To: RotationalCapabilities : Class , Public

ATTRIBUTES
Capabilities : **RotationalCapabilities** Private *Details:* This field overrides the Capabilities definition to provide an in-depth model of a typical rotation machine, focusing on capturing the properties of large-scale, bulk generator. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Status : **RotationalStatus** Private *Details:* This field overrides the *Status* type to provide a specialized version that can account for the properties of a rotational machine. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.2.4.11 RotationalStatus

Class in package '4.4.4 RotationalGenerationResources'

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from RotationalStatus to CurrentStatus

CONNECTORS
Dependency Source -> Destination From: : RotationalResource : Class , Public To: RotationalStatus : Class , Public

OPERATIONS
GetStatus () : **void** Public *Details:*

A.2.4.12 FuelSource

Enumeration in package '4.4.4 RotationalGenerationResources'

CONNECTORS
Dependency Source -> Destination From: : iMechConversionProcess : Class , Public To: FuelSource : Enumeration , Public
Dependency Source -> Destination From: : RotatingMachineModel : Class , Public To: FuelSource : Enumeration , Public

CONNECTORS
Dependency Source -> Destination From: : RotatingMachineModel : Class , Public To: FuelSource : Enumeration , Public

ENUMERATION:
NaturalGas
Gasoline
Wind
Propane
Diesel
Hydro
Nuclear
Thermal
RecoverySteam

A.2.4.13 IntermediaryCarrier

Enumeration in package '4.4.4 RotationalGenerationResources'

CONNECTORS
Dependency Source -> Destination From: : iMechConversionProcess : Class , Public To: IntermediaryCarrier : Enumeration , Public

ENUMERATION:
Steam
Water
NA

A.2.4.14 MechanicalUOMs

Enumeration in package '4.4.4 RotationalGenerationResources'

Details: This enumeration is used to document the units on which the *MomentOfInertia* is being reported.

CONNECTORS
Dependency Source -> Destination From: : MechanicalValueType : Class , Public To: MechanicalUOMs : Enumeration , Public

ENUMERATION:
*kg*meter_sq*
*lbf*ft*s_sq*
...

A.2.4.15 TurbineState

Enumeration in package '4.4.4 RotationalGenerationResources'

CONNECTORS
Dependency Source -> Destination From: : iMechConversionProcess : Class , Public To: TurbineState : Enumeration , Public

ENUMERATION:
NA
Cold
Hot
Wark

A.2.5 Storage Resources

This diagram presents an overview of a storage-based resource. It expands on the interfaces presented earlier and divides the charging/delivery process into two dedicated systems, which can be specified independently.

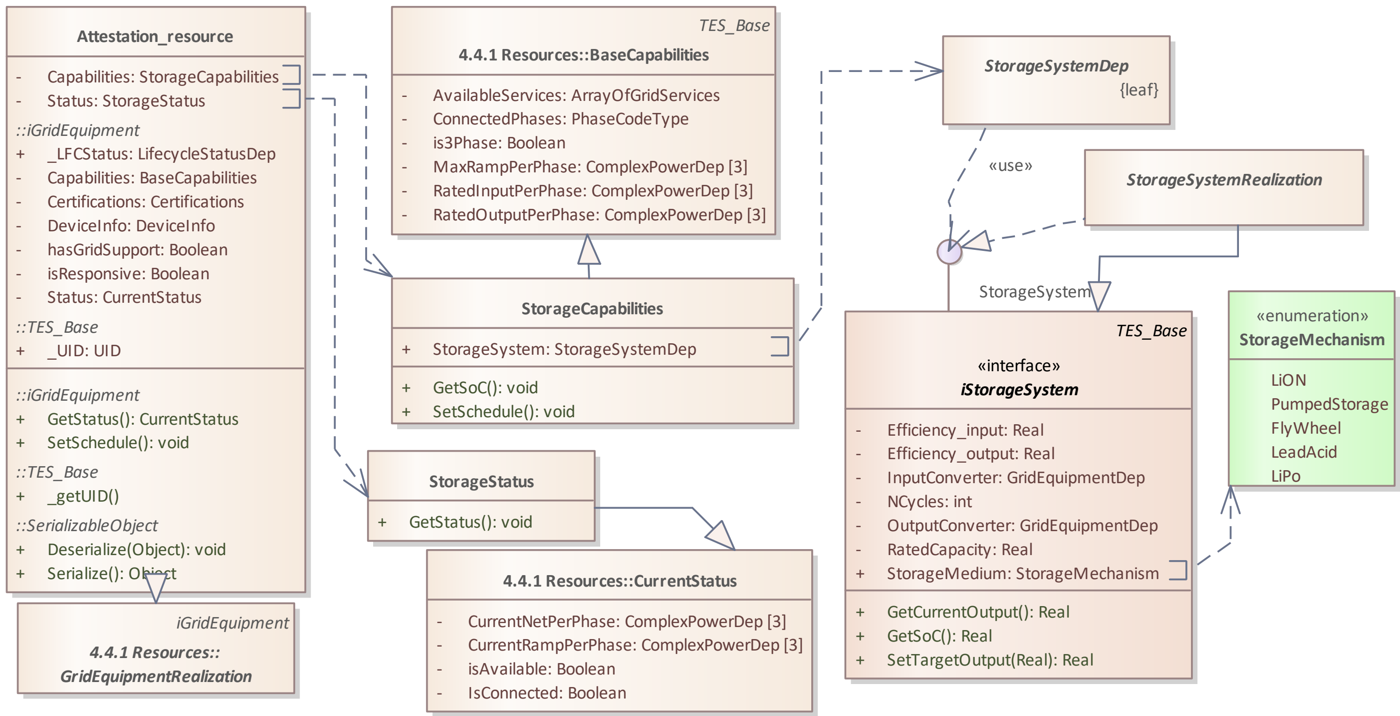


Figure 69. Overview of the StorageResources' package components

A.2.5.1 Attestation_resource

Class in package '4.4.5 StorageResources'

Details: This class realizes a rotation machine specialization of the *GridEquipment* interface by extending the *GridEquipmentRealization* class. It overrides the *Capabilities* and *Status* fields.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from Attestation_resource to GridEquipmentRealization

CONNECTORS
Dependency Source -> Destination From: : Attestation_resource : Class , Public To: StorageStatus : Class , Public

ATTRIBUTES
Capabilities : **StorageCapabilities** Private *Details:* This field overrides the Capabilities definition to represent the qualifications of a storage-like system, exposing specialized properties such as the state of charge. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Status : **StorageStatus** Private *Details:* This field overrides the standard *Status* type to provide a specialized version that can retrieve properties typically associated with storage systems. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.2.5.2 iStorageSystem

Class «interface» in package '4.4.5 StorageResources'

Details: This base class defines the typical properties associated with storage systems. The interface considers that two separate systems are responsible for putting and retrieving energy from the energy storage mechanism. It exposes certain system-specific properties such as the system's state of charge (SoC).

STRUCTURAL PART OF iStorageSystem
StorageSystem : ProvidedInterface

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from «interface» iStorageSystem to TES_Base

CONNECTORS
Dependency Source -> Destination From: : iStorageSystem : Class , Public To: StorageMechanism : Enumeration , Public

ATTRIBUTES
Efficiency_input : **Real** Private *Details:* This represents a system's efficiency as measured from the input side (charging) to the storage medium. This number is only to serve as a general reference, since actual conversion rates can vary depending on the level-of-charge, consumption/charging currents. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Efficiency_output : **Real** Private

ATTRIBUTES
Details: This represents a system's efficiency as measured from the storage point to the output point (delivery). This number only serves as a general reference, since actual conversion rates can vary depending on the level-of-charge, consumption/charging currents. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
InputConverter : **GridEquipmentDep** Private *Details:* This field is used to document the input side equipment. This may typically be a rectifier, but also a mechanical system (for pumped storage or a flywheel). Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
NCycles : **int** Private *Details:* This property can be used to store the number of charge/discharge cycles. Useful for estimating degradation, or actual capacity. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
OutputConverter : **GridEquipmentDep** Private *Details:* This field can be used to define the output conversion equipment, which takes the stored energy and delivers it to the grid. Typical equipment may be an inverter or a mechanical system (for a pumped storage system). Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
RatedCapacity : **Real** Private *Details:* This value represents the rated power capacity (in kWH). Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
StorageMedium : **StorageMechanism** Public *Details:* This field is used to define the storage medium used by the system. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

OPERATIONS
GetCurrentOutput () : **Real** Public *Details:* This function can be used to determine the current output (from the output converter).
GetSoC () : **Real** Public *Details:* This function gets the system's state of charge (if applicable).
SetTargetOutput (ValuekW : **Real**) : **Real** Public *Details:* This function can be used to schedule the device power demand behavior. Positive numbers imply extracting energy from the storage, negative numbers imply charging.

A.2.5.3 StorageCapabilities

Class in package '4.4.5 StorageResources'

Details: This structure aggregates functions and properties that are typical of a storage system. The object provides access to the SoC function, which can be used by a TES agent to optimize resource usage.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from StorageCapabilities to BaseCapabilities

CONNECTORS
Dependency Source -> Destination From: : StorageCapabilities : Class , Public To: StorageSystemDep : Class , Public
Dependency Source -> Destination From: : Attestation_resource : Class , Public To: StorageCapabilities : Class , Public

ATTRIBUTES
StorageSystem : **StorageSystemDep** Public *Details:* This object represents the storage system, which may be an aggregation of multiple equipment Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

OPERATIONS
GetSoC () : **void** Public *Details:* This function can automatically update the ramping rate based on the internal state of the mechanical interface.
SetSchedule () : **void** Public *Details:* This function can "schedule" a resources output capabilities across time.

A.2.5.4 StorageStatus

Class in package '4.4.5 StorageResources'

Details: This object holds functions that are specific to a storage system

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from StorageStatus to CurrentStatus

CONNECTORS
Dependency Source -> Destination From: : Attestation_resource : Class , Public To: StorageStatus : Class , Public

OPERATIONS
GetStatus () : **void** Public *Details:* This function will return the output/input values, and the SoC.

A.2.5.5 StorageSystemDep

Class in package '4.4.5 StorageResources'

Details: Objects that reference this class expect an object that realizes the *StorageSystem* Interface. This class is a leaf and is only intended to serve as a data type.

CONNECTORS
Usage Source -> Destination From: : StorageSystemDep : Class , Public To: StorageSystem : ProvidedInterface , Public
Dependency Source -> Destination From: : StorageCapabilities : Class , Public To: StorageSystemDep : Class , Public

A.2.5.6 StorageSystemRealization

Class in package '4.4.5 StorageResources'

Details: This abstract class implements the *StorageSystem* interface which allows to model a storage-like system. Classes derived from this class should satisfy all of the service and data requirements.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from StorageSystemRealization to «interface» iStorageSystem
Realization from StorageSystemRealization to StorageSystem

A.2.5.7 StorageMechanism

Enumeration in package '4.4.5 StorageResources'

Details: This enumeration is used to describe the storage medium used. The listed mediums are intended for reference and should be expanded to suit the TES needs.

CONNECTORS
Dependency Source -> Destination From: : iStorageSystem : Class , Public To: StorageMechanism : Enumeration , Public

ENUMERATION:
LiON
PumpedStorage
FlyWheel
LeadAcid
LiPo

A.2.6 Attestation Resources

This diagram presents an overview of an attestation-capable resource. In this case, it is assumed that an attestation device only exists on the digital domain (although signal sampling can occur on the physical side). The proposed model ties the attestation device to another's device sampling/measurement interface, and can choose to digitally sign/protect data if desired.

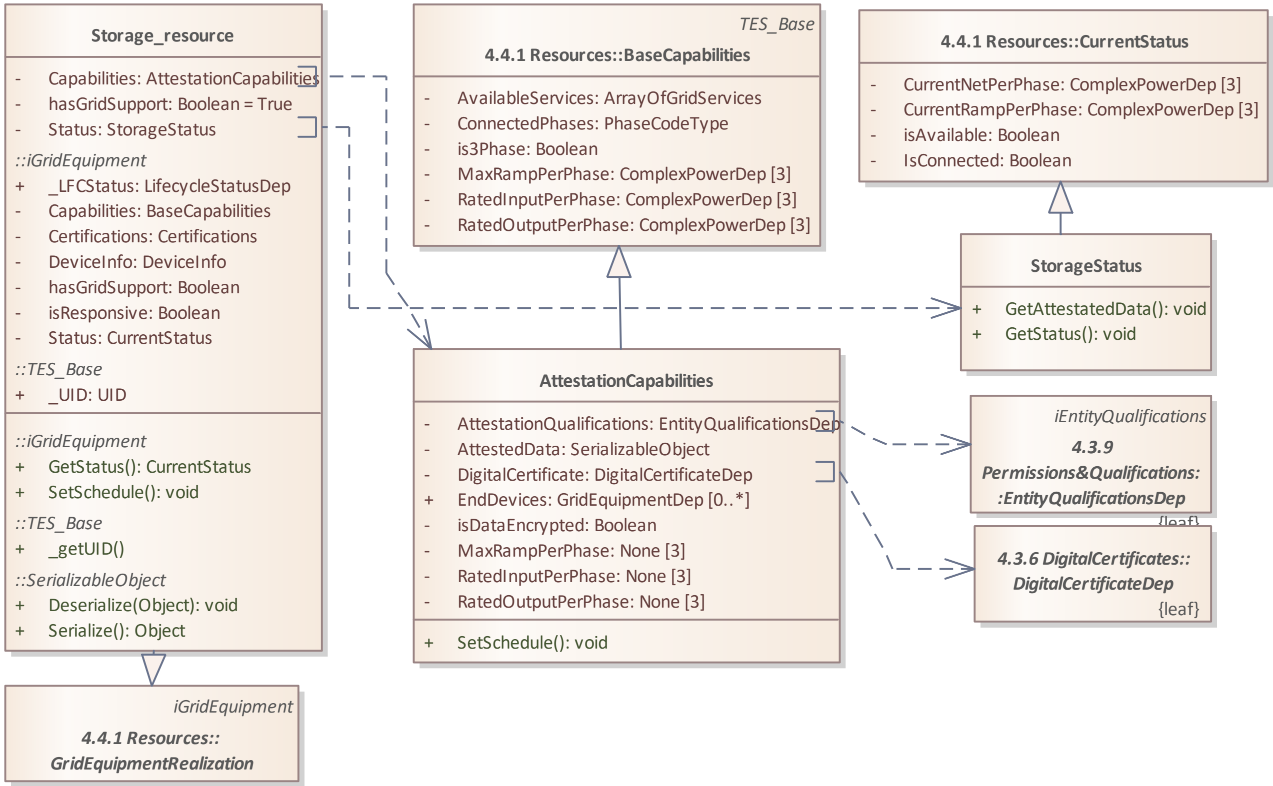


Figure 70. Overview of the AttestationResources' package components

A.2.6.1 AttestationCapabilities

Class in package '4.4.6 AttestationResources'

Details: This structure aggregates functions and properties that are typical of a storage system. The object provides access to the SoC function, which can be used by a TES agent to optimize resource usage.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from AttestationCapabilities to BaseCapabilities

CONNECTORS
Dependency Source -> Destination From: : AttestationCapabilities : Class , Public To: DigitalCertificateDep : Class , Public
Dependency Source -> Destination From: : AttestationCapabilities : Class , Public To: EntityQualificationsDep : Class , Public
Dependency Source -> Destination From: : Storage_resource : Class , Public To: AttestationCapabilities : Class , Public

ATTRIBUTES
AttestationQualifications : **EntityQualificationsDep** Private *Details:* This field can be used to document a device's attestation rights/permissions. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
AttestedData : **SerializableObject** Private *Details:* This field contains the serialized data being attested. The data format must be defined at runtime in order for producers and consumers to communicate appropriately. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
DigitalCertificate : **DigitalCertificateDep** Private *Details:* This field can be used to provide the certificate details that is used to sign or encrypt the attested data. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
EndDevices : **GridEquipmentDep** Public *Details:* This object represents the systems for which digital attestation is being provided Multiplicity: (0..*, Allow duplicates: 0, Is ordered: False)
isDataEncrypted : **Boolean** Private *Details:*
MaxRampPerPhase : **None** Private *Details:* This field overrides the ramping properties to assert that no power flexibility is offered. Multiplicity: (3, Allow duplicates: 0, Is ordered: False)
RatedInputPerPhase : **None** Private *Details:* This field overrides the input power properties to assert that no power flexibility is offered. Multiplicity: (3, Allow duplicates: 0, Is ordered: False)
RatedOutputPerPhase : **None** Private *Details:* This field overrides the output properties to assert that no power output is being offered. Multiplicity: (3, Allow duplicates: 0, Is ordered: False)

OPERATIONS
SetSchedule () : **void** Public *Details:* This function can be used to configured the attestation periodicity or end-points.

A.2.6.2 Storage_resource

Class in package '4.4.6 AttestationResources'

Details: This class realizes a rotation machine specialization of the *GridEquipment* interface by extending the *GridEquipmentRealization* class. It overrides the *Capabilities* and *Status* fields.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from Storage_resource to GridEquipmentRealization

CONNECTORS
Dependency Source -> Destination From: : Storage_resource : Class , Public To: AttestationCapabilities : Class , Public
Dependency Source -> Destination From: : Storage_resource : Class , Public To: StorageStatus : Class , Public

ATTRIBUTES
Capabilities : **AttestationCapabilities** Private *Details:* This field overrides the Capabilities definition to represent the qualifications of a storage-like system, exposing specialized properties such as the state of charge. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
hasGridSupport : **Boolean** Private = True *Details:*
Status : **StorageStatus** Private *Details:* This field overrides the standard *Status* type to provide a specialized version that can retrieve properties typically associated with storage systems. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.2.6.3 StorageStatus

Class in package '4.4.6 AttestationResources'

Details: This object holds functions that are specific to a storage system

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from StorageStatus to CurrentStatus

CONNECTORS
Dependency Source -> Destination From: : Storage_resource : Class , Public To: StorageStatus : Class , Public

OPERATIONS
GetAttestatedData () : **void** Public *Details:*
GetStatus () : **void** Public *Details:* This function will return the output/input values, and the SoC.

A.2.7 OrganizationalHierarchy

This package introduces a reference hierarchy that can be used to map the different types of actors/systems that may be present on a typical TES system where a wide variety of participants may interact. This diagram is only intended to be illustrative and can be adjusted to suit the application needs.

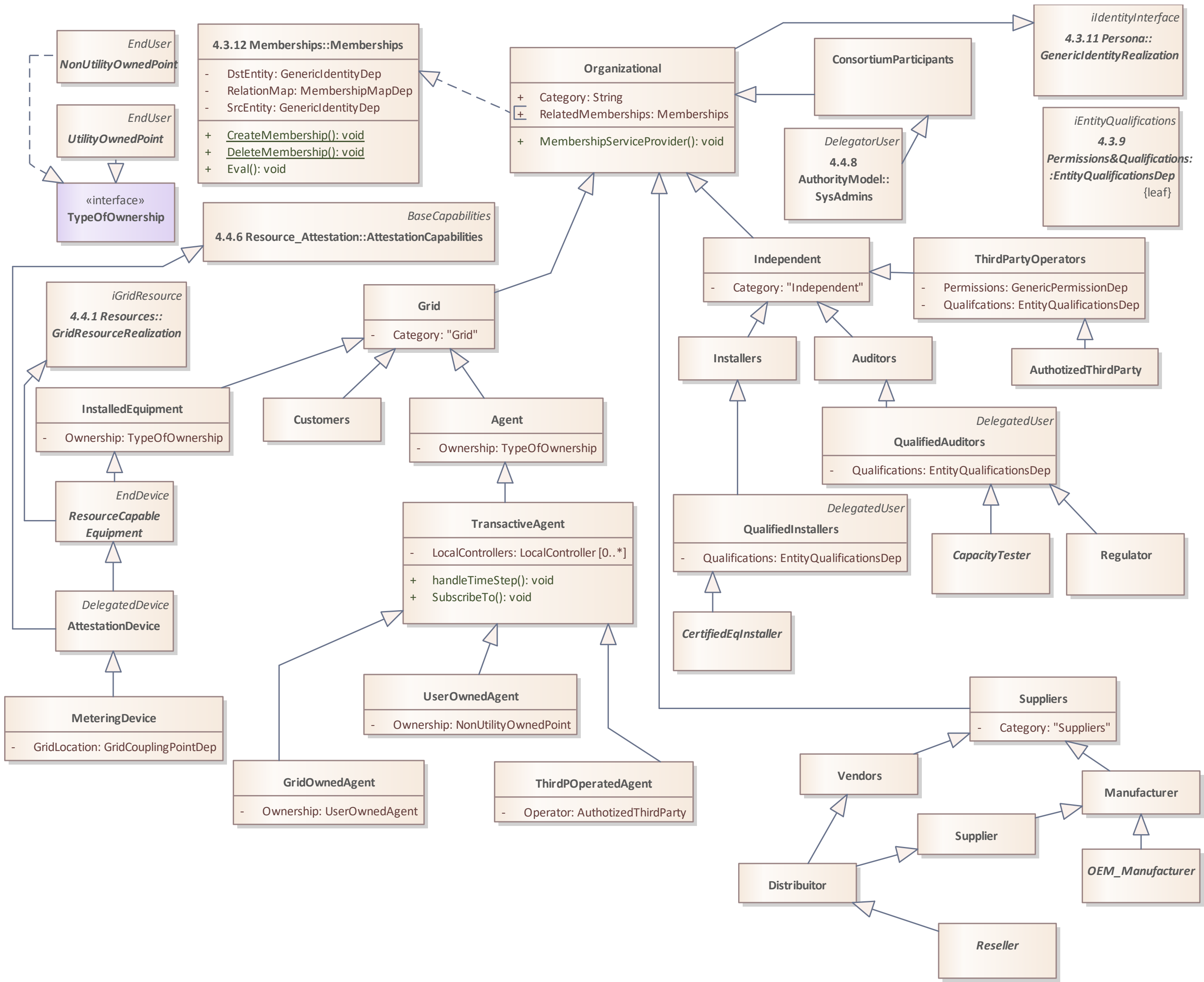


Figure 71. Overview of the OrganizationalHierarchy' package components

A.2.7.1 Agent

Class in package '4.4.7 OrganizationalHierarchy'

Details: This class encompasses all agents that are either tied or have an effect on the grid state.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from Agent to Grid

ATTRIBUTES
Ownership : **TypeOfOwnership** Private *Details:*

A.2.7.2 AttestationDevice

Class in package '4.4.7 OrganizationalHierarchy'

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from AttestationDevice to ResourceCapable Equipment
Generalization from AttestationDevice to AttestationCapabilities
Generalization from AttestationDevice to DelegatedDevice

A.2.7.3 Auditors

Class in package '4.4.7 OrganizationalHierarchy'

Details: This class encompasses independent entities that can audit/inspect grid related aspects. This may include equipment, process flows or smart contract enforcement.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from Auditors to Independent

A.2.7.4 AuthorizedInstallers

Class in package '4.4.7 OrganizationalHierarchy'

A.2.7.5 AuthotizedThirdParty

Class in package '4.4.7 OrganizationalHierarchy'

Details: This class represents third-party systems. In specific, entities that operate/represent another entity.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from AuthotizedThirdParty to ThirdPartyOperators

CONNECTORS
Usage «Instantiate» Source -> Destination From: : AuthorizedThirdParty : Actor , Public To: AuthotizedThirdParty : Class , Public

A.2.7.6 CapacityTester

Class in package '4.4.7 OrganizationalHierarchy'

Details: This class represents a set of specialized agents than can assess a resource actual capabilities and can enforce compliance requirements.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from CapacityTester to QualifiedAuditors

CONNECTORS
Usage «Instantiate» Source -> Destination From: : CapacityTester_Actor : Actor , Public To: CapacityTester : Class , Public
Dependency Source -> Destination From: : TESResourceQuaification : Class , Public To: CapacityTester : Class , Public

A.2.7.7 CertifiedEqInstaller

Class in package '4.4.7 OrganizationalHierarchy'

Details: This class represents entities that are certified to install/commission certain types of devices (e.g., an inverter-based resource).

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from CertifiedEqInstaller to QualifiedInstallers

CONNECTORS
Usage «Instantiate» Source -> Destination From: : CertifiedInstaller_Actor : Actor , Public To: CertifiedEqInstaller : Class , Public

A.2.7.8 ConsortiumParticipants

Class in package '4.4.7 OrganizationalHierarchy'

Details: This class represents all entities that have decided to participate on a TES-BC based system.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from ConsortiumParticipants to Organizational

CONNECTORS
Usage «Instantiate» Source -> Destination From: : TES_Consortium : Actor , Public To: ConsortiumParticipants : Class , Public

A.2.7.9 Customers

Class in package '4.4.7 OrganizationalHierarchy'

Details: Within this example, this class represents all customers with no specific role. A customer can own/operate devices or transactive devices and become more specialized.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from Customers to Grid

CONNECTORS
Usage «Instantiate» Source -> Destination From: : Customer_Actor : Actor , Public To: Customers : Class , Public

A.2.7.10 Distribuitor

Class in package '4.4.7 OrganizationalHierarchy'

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from Distribuitor to Supplier
Generalization from Distribuitor to Vendors

A.2.7.11 Grid

Class in package '4.4.7 OrganizationalHierarchy'

Details: This class groups all objects that are connected, interact or have a relation with the grid.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from Grid to Organizational

ATTRIBUTES
Category : **"Grid"** Private *Details:*

A.2.7.12 GridOwnedAgent

Class in package '4.4.7 OrganizationalHierarchy'

Details: This class represents transactive agents that are owned/operated by a utility (or other fully trusted organization that responds to the utility)

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from GridOwnedAgent to TransactiveAgent
Generalization from GridOwnedAgent to UtilityOwnedPoint

CONNECTORS
Usage «Instantiate» Source -> Destination From: : UtilityAgent : Actor , Public To: GridOwnedAgent : Class , Public

ATTRIBUTES
Ownership : **UserOwnedAgent** Private *Details:*

A.2.7.13 Independent

Class in package '4.4.7 OrganizationalHierarchy'

Details: This class represents actors who are independent of the grid but can participate as observers or indirect-system providers.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from Independent to Organizational

CONNECTORS
Usage «Instantiate» Source -> Destination From: : NonQualified_Actor : Actor , Public To: Independent : Class , Public

ATTRIBUTES
Category : **"Independent"** Private *Details:*

A.2.7.14 InstalledEquipment

Class in package '4.4.7 OrganizationalHierarchy'

Details: This object model can be used to represent equipment that is installed/connected to the grid.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from InstalledEquipment to Grid

ATTRIBUTES
Ownership : **TypeOfOwnership** Private *Details:*

A.2.7.15 Installers

Class in package '4.4.7 OrganizationalHierarchy'

Details: This class encompasses independent service providers that perform device installation/commissioning.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from Installers to Independent

A.2.7.16 Manufacturer

Class in package '4.4.7 OrganizationalHierarchy'

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from Manufacturer to Suppliers

A.2.7.17 MeteringDevice

Class in package '4.4.7 OrganizationalHierarchy'

Details: This object represents a trusted metering agent. This object has been adapted from IEEE 2030.5

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from MeteringDevice to AttestationDevice

CONNECTORS
Dependency Source -> Destination From: : MeteringDevice : Class , Public To: GridCouplingPointDep : Class , Public
Usage «Instantiate» Source -> Destination From: : Meter_Actor : Actor , Public To: MeteringDevice : Class , Public
Dependency Source -> Destination From: : UsagePoint : Class , Public To: MeteringDevice : Class , Public

ATTRIBUTES
GridLocation : **GridCouplingPointDep** Private *Details:*

A.2.7.18 NonUtilityOwnedPoint

Class in package '4.4.7 OrganizationalHierarchy'

Details: This class realizes the properties of non-utility owned devices. This construct can be expanded to support cases when multiple utilities are integrated into a single TES solution.

OUTGOING STRUCTURAL RELATIONSHIPS
Realization from NonUtilityOwnedPoint to TypeOfOwnership
Generalization from NonUtilityOwnedPoint to EndUser

A.2.7.19 OEM_Manufacturer

Class in package '4.4.7 OrganizationalHierarchy'

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from OEM_Manufacturer to Manufacturer

CONNECTORS
Usage «Instantiate» Source -> Destination From: : Manufacturer_Actor : Actor , Public To: OEM_Manufacturer : Class , Public

A.2.7.20 Organizational

Class in package '4.4.7 OrganizationalHierarchy'

Details: This class represents a typical organization's member. All members have an identity, roles/attributes, and in this case memberships to support member to member relationships.

OUTGOING STRUCTURAL RELATIONSHIPS
Realization from Organizational to Memberships
Generalization from Organizational to GenericIdentityRealization

ATTRIBUTES
Category : **String** Public *Details:* This field can be broadly describe the relationship with the organization/consortium. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
RelatedMemberships : **Memberships** Public *Details:*

OPERATIONS
MembershipServiceProvider () : **void** Public *Details:*

A.2.7.21 QualifiedAuditors

Class in package '4.4.7 OrganizationalHierarchy'

Details: These class represents a category of entities that can audit grid operations. This may include market monitoring agents, capacity evaluators and any other agent that can perform unbiased grid assessments.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from QualifiedAuditors to DelegatedUser
Generalization from QualifiedAuditors to Auditors

CONNECTORS
Dependency Source -> Destination From: : QualifiedAuditors : Class , Public To: EntityQualificationsDep : Class , Public

ATTRIBUTES
Qualifications : **EntityQualificationsDep** Private *Details:*

A.2.7.22 QualifiedInstallers

Class in package '4.4.7 OrganizationalHierarchy'

Details: These class represents a category of installers that are qualified to perform certain types of installations or perform device commissioning. Examples may include those entities that have sufficient expertise to enroll a new *ResourceCapableEquipment* or can deploy new agents on behalf of end users.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from QualifiedInstallers to Installers
Generalization from QualifiedInstallers to DelegatedUser

CONNECTORS
Dependency Source -> Destination From: : QualifiedInstallers : Class , Public To: EntityQualificationsDep : Class , Public

ATTRIBUTES
Qualifications : **EntityQualificationsDep** Private *Details:*

A.2.7.23 Regulator

Class in package '4.4.7 OrganizationalHierarchy'

Details: This class represents a set of specialized agents in charge of enforcing compliance requirements. Examples include local energy commissions, market monitors, etc.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from Regulator to QualifiedAuditors

CONNECTORS
Usage «Instantiate» Source -> Destination From: : Regulator_Actor : Actor , Public To: Regulator : Class , Public

A.2.7.24 Reseller

Class in package '4.4.7 OrganizationalHierarchy'

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from Reseller to Distribuitor

CONNECTORS
Usage «Instantiate» Source -> Destination From: : Reseller_Actor : Actor , Public To: Reseller : Class , Public

A.2.7.25 ResourceCapable Equipment

Class in package '4.4.7 OrganizationalHierarchy'

Details: This class represents any equipment that can provide grid services.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from ResourceCapable Equipment to InstalledEquipment
Generalization from ResourceCapable Equipment to GridResourceRealization
Generalization from ResourceCapable Equipment to EndDevice

A.2.7.26 Supplier

Class in package '4.4.7 OrganizationalHierarchy'

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from Supplier to Manufacturer

A.2.7.27 Suppliers

Class in package '4.4.7 OrganizationalHierarchy'

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from Suppliers to Organizational

ATTRIBUTES
Category : **"Suppliers"** Private *Details:*

A.2.7.28 ThirdPartyOperators

Class in package '4.4.7 OrganizationalHierarchy'

Details: This class represents third-party system-wide operators. In specific, entities that are allowed to act on behalf of another entity or service. It also includes dedicated contractors.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from ThirdPartyOperators to Independent

ATTRIBUTES
Permissions : **GenericPermissionDep** Private *Details:*
Qualifcations : **EntityQualificationsDep** Private *Details:*

A.2.7.29 ThirdPOperatedAgent

Class in package '4.4.7 OrganizationalHierarchy'

Details: This class represents transactive agents that are operated by a third party.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from ThirdPOperatedAgent to TransactiveAgent

CONNECTORS
Usage «Instantiate» Source -> Destination From: : ThirdPAgent : Actor , Public To: ThirdPOperatedAgent : Class , Public

ATTRIBUTES
Operator : **AuthotizedThirdParty** Private *Details:*

A.2.7.30 TransactiveAgent

Class in package '4.4.7 OrganizationalHierarchy'

Details: This object represents a high-level representation of a transactive agent. Such an agent may depend on a series of local controllers to offer flexibility.
Under the TES-BC template approach, agents are not required to have a physical counterpart. Examples of agents that may operate only in the digital domain include market, supervisory and data aggregation agents

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from TransactiveAgent to Agent

CONNECTORS
Dependency Source -> Destination From: : TransactiveAgent : Class , Public To: LocalController : Class , Public
Usage «Instantiate» Source -> Destination From: : ThirdPAgent : Actor , Public To: TransactiveAgent : Class , Public

ATTRIBUTES
LocalControllers : **LocalController** Private *Details:*

OPERATIONS
handleTimeStep () : **void** Public *Details:* This function can be configured to listen for time-step changes or dynamic events.
SubscribeTo () : **void** Public *Details:* This function can be used to subscribe to events, or any semi-automatic polling mechanism.

A.2.7.31 UserOwnedAgent

Class in package '4.4.7 OrganizationalHierarchy'

Details: This class represents transactive agents that are owned/operated by private entities.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from UserOwnedAgent to TransactiveAgent
Generalization from UserOwnedAgent to NonUtilityOwnedPoint

CONNECTORS
Usage «Instantiate» Source -> Destination From: : UserOwnedAgent : Actor , Public To: UserOwnedAgent : Class , Public

ATTRIBUTES
Ownership : **NonUtilityOwnedPoint** Private *Details:*

A.2.7.32 UtilityOwnedPoint

Class in package '4.4.7 OrganizationalHierarchy'

Details: This class represents objects that are not owned/managed by the electrical utilities. These devices may have limited control and visibility properties.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from UtilityOwnedPoint to EndUser
Realization from UtilityOwnedPoint to TypeOfOwnership

A.2.7.33 Vendors

Class in package '4.4.7 OrganizationalHierarchy'

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from Vendors to Suppliers

A.2.7.34 TypeOfOwnership

Interface in package '4.4.7 OrganizationalHierarchy'

Details: This interface can be used to describe the different types of ownership. In addition it may contain properties, attributes and functions that are only found on certain types of ownership.

A.2.8 Authority Model

These group of classes represents the subset of participants that have administrator-like rights over other participants. These participants can manage other participants (such as dictating a role or permissions) as well as defining processes and setting rules.

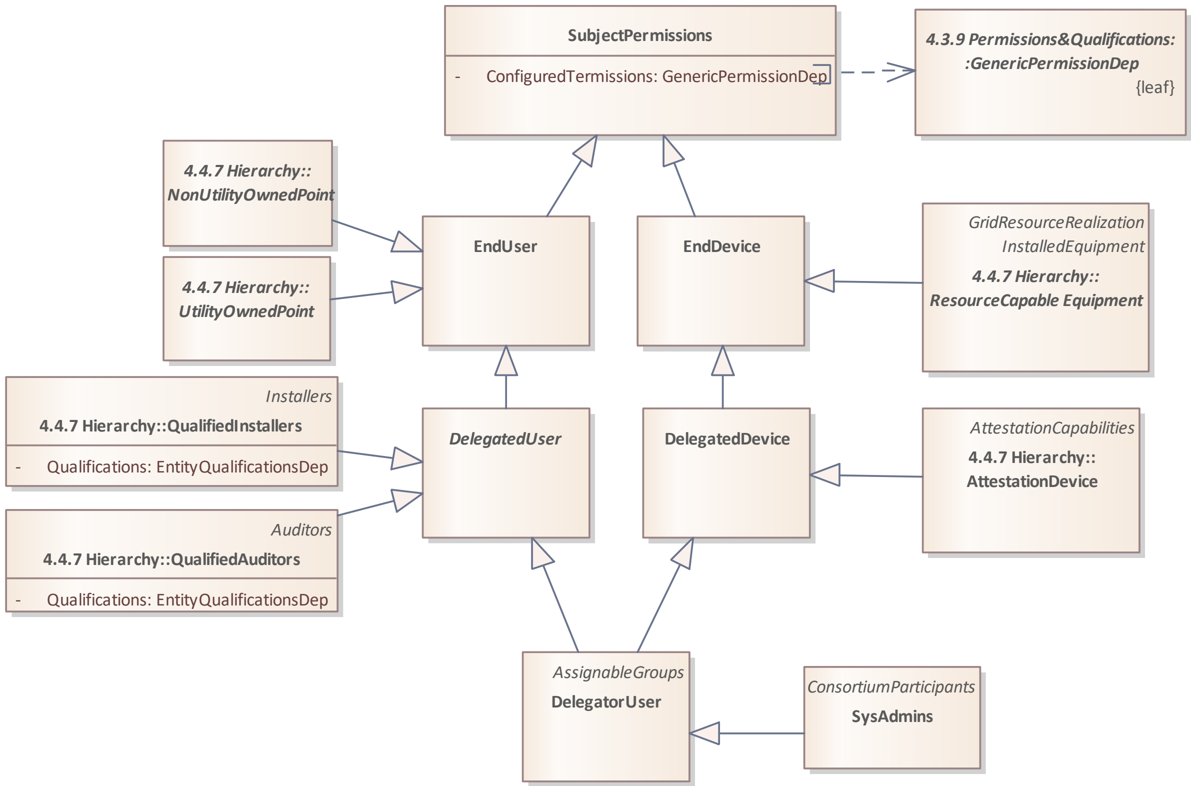


Figure 72. Overview of the AuthorityModel' package components

A.2.8.1 DelegatedDevice

Class in package '4.4.8 AuthorityModel'

Details: This class can be used to assign permissions to devices/systems that have active participation roles within the system. These may include managing or administering other systems' properties. This class may also be used to classify services that have higher trust levels than end-point service providers.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from DelegatedDevice to EndDevice

A.2.8.2 DelegatedUser

Class in package '4.4.8 AuthorityModel'

Details: This class can be used to assign permissions to users that have active participation roles within the system. These may include managing or administering other user's properties. This class may also be used to classify agents that have higher trust levels than other peers.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from DelegatedUser to EndUser

A.2.8.3 DelegatorUser

Class in package '4.4.8 AuthorityModel'

Details: This class models entities that have strong management roles. This may include creating, updating and removing entities, as well as managing their properties/permissions. These entities are fully trusted by the consortium.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from DelegatorUser to DelegatedUser
Generalization from DelegatorUser to DelegatedDevice
Generalization from DelegatorUser to AssignableGroups

A.2.8.4 EndDevice

Class in package '4.4.8 AuthorityModel'

Details: This class can be used to assign permissions to devices/systems that have limited participation scope. These may include systems that have limited operational capacities, or that have no influence over other agents.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from EndDevice to SubjectPermissions

A.2.8.5 EndUser

Class in package '4.4.8 AuthorityModel'

Details: This class can be used to assign permissions to users that have limited participation scope. These may include users that have limited operational capacities, or that have no influence over other agents.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from EndUser to SubjectPermissions

A.2.8.6 SubjectPermissions

Class in package '4.4.8 AuthorityModel'

Details: This class can be used to configure an entity permissions.

CONNECTORS
Dependency Source -> Destination From: : SubjectPermissions : Class , Public To: GenericPermissionDep : Class , Public

ATTRIBUTES
ConfiguredTermissions : **GenericPermissionDep** Private *Details:*

A.2.8.7 SysAdmins

Class in package '4.4.8 AuthorityModel'

Details: This class represents the subset of participants that have administrator-like rights over other participants. These participants can manage other participants (such as dictating a role or permissions) as well as defining processes and setting rules.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from SysAdmins to ConsortiumParticipants
Generalization from SysAdmins to DelegatorUser

CONNECTORS
Usage «Instantiate» Source -> Destination From: : TES_Admin : Actor , Public To: SysAdmins : Class , Public

A.2.9 Sample Hierarchy With Actors

This diagram represents a reference hierarchy that can be used to map the different types of actors/systems that may be present on a typical TES system where a wide variety of participants may interact. This diagram is only intended to be illustrative and can be adjusted to suit the application needs. The diagram has been populated with actors that will be used to demonstrate potential use cases.

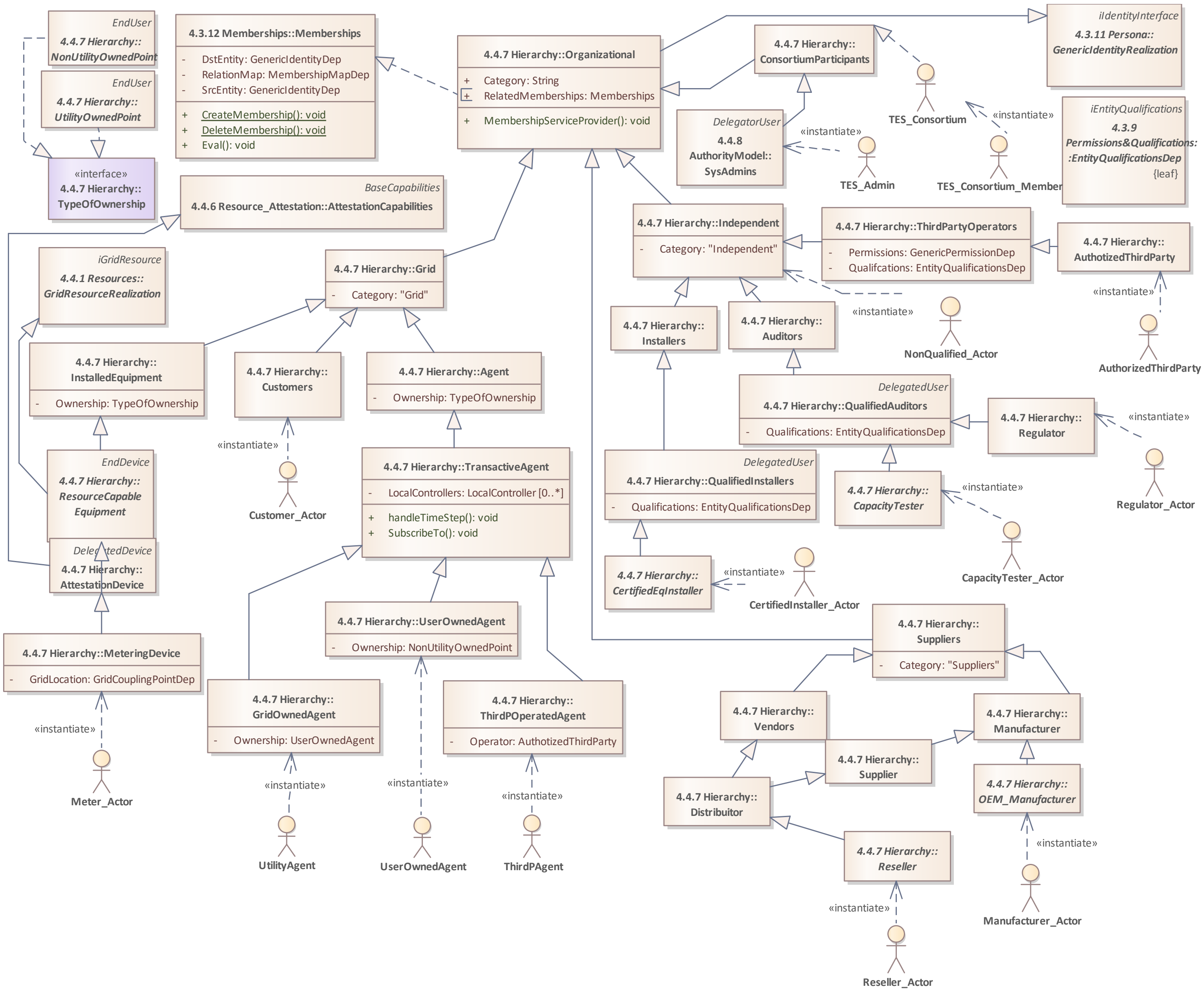


Figure 73. Overview of the SampleHierarchyWithActors' package components

A.2.9.1 AuthorizedThirdParty

Actor in package '4.4.9 SampleHierarchyWithActors'

Details: This actor represents an entity that has been authorized to operate other's systems/devices.

CONNECTORS
Usage «Instantiate» Source -> Destination From: : AuthorizedThirdParty : Actor , Public To: AuthotizedThirdParty : Class , Public

A.2.9.2 CapacityTester_Actor

Actor in package '4.4.9 SampleHierarchyWithActors'

Details: This actor represents an individual that can assess an equipment/service qualities, or its actual capabilities based on standardized tests.

CONNECTORS
Usage «Instantiate» Source -> Destination From: : CapacityTester_Actor : Actor , Public To: CapacityTester : Class , Public

A.2.9.3 CertifiedInstaller_Actor

Actor in package '4.4.9 SampleHierarchyWithActors'

Details: This actor represents an instance of a certified installer.

CONNECTORS
Usage «Instantiate» Source -> Destination From: : CertifiedInstaller_Actor : Actor , Public To: CertifiedEqInstaller : Class , Public

A.2.9.4 Customer_Actor

Actor in package '4.4.9 SampleHierarchyWithActors'

Details: This actor represents a customer with no specific attributes other than being connected to the grid. In this sample architecture, a customer can become an specialized agent by installing/operating equipment or a transactive agent.

CONNECTORS
Usage «Instantiate» Source -> Destination From: : Customer_Actor : Actor , Public To: Customers : Class , Public

A.2.9.5 Manufacturer_Actor

Actor in package '4.4.9 SampleHierarchyWithActors'

CONNECTORS
Usage «Instantiate» Source -> Destination From: : Manufacturer_Actor : Actor , Public To: OEM_Manufacturer : Class , Public

A.2.9.6 Meter_Actor

Actor in package '4.4.9 SampleHierarchyWithActors'

Details: This actor represents an instance of an electrical meter, installed on the grid.

CONNECTORS
Usage «Instantiate» Source -> Destination From: : Meter_Actor : Actor , Public To: MeteringDevice : Class , Public

A.2.9.7 NonQualified_Actor

Actor in package '4.4.9 SampleHierarchyWithActors'

Details: This represents a non-qualified actor that has access to the TES-BC platform.

CONNECTORS
Usage «Instantiate» Source -> Destination From: : NonQualified_Actor : Actor , Public To: Independent : Class , Public

A.2.9.8 Regulator_Actor

Actor in package '4.4.9 SampleHierarchyWithActors'

Details: This actor represents a regulator body, usually in charge of compliance reviews.

CONNECTORS
Usage «Instantiate» Source -> Destination From: : Regulator_Actor : Actor , Public To: Regulator : Class , Public

A.2.9.9 Reseller_Actor

Actor in package '4.4.9 SampleHierarchyWithActors'

CONNECTORS
Usage «Instantiate» Source -> Destination From: : Reseller_Actor : Actor , Public To: Reseller : Class , Public

A.2.9.10 TES_Admin

Actor in package '4.4.9 SampleHierarchyWithActors'

CONNECTORS
Usage «Instantiate» Source -> Destination From: : TES_Admin : Actor , Public To: SysAdmins : Class , Public

A.2.9.11 TES_Consortium

Actor in package '4.4.9 SampleHierarchyWithActors'

OUTGOING STRUCTURAL RELATIONSHIPS
Realization from TES_Consortium to ConsortiumParticipants

CONNECTORS
Usage «Instantiate» Source -> Destination From: : TES_Consortium : Actor , Public To: ConsortiumParticipants : Class , Public
Usage «Instantiate» Source -> Destination From: : TES_Consortium_Member : Actor , Public To: TES_Consortium : Actor , Public

A.2.9.12 TES_Consortium_Member

Actor in package '4.4.9 SampleHierarchyWithActors'

CONNECTORS
Usage «Instantiate» Source -> Destination From: : TES_Consortium_Member : Actor , Public To: TES_Consortium : Actor , Public

A.2.9.13 ThirdPAgent

Actor in package '4.4.9 SampleHierarchyWithActors'

Details: This actor represents a transactive agent that is operated/managed by a third party.

CONNECTORS
Usage «Instantiate» Source -> Destination From: : ThirdPAgent : Actor , Public To: ThirdPOperatedAgent : Class , Public
Usage «Instantiate» Source -> Destination From: : ThirdPAgent : Actor , Public To: TransactiveAgent : Class , Public

A.2.9.14 UserOwnedAgent

Actor in package '4.4.9 SampleHierarchyWithActors'

Details: This actor represents a transactive agent that is owned and operated by a private user.

CONNECTORS
Usage «Instantiate» Source -> Destination From: : UserOwnedAgent : Actor , Public To: UserOwnedAgent : Class , Public

A.2.9.15 UtilityAgent

Actor in package '4.4.9 SampleHierarchyWithActors'

Details: This actor represents a transactive agent that is owned and operated by a utility.

CONNECTORS
Usage «Instantiate» Source -> Destination From: : UtilityAgent : Actor , Public To: GridOwnedAgent : Class , Public

A.3 Grid Components

This section presents a grid modeling proposal that aims to retain the electrical topological hierarchy of power systems while at the same time enabling grid support services to attach to virtual grid points. This grid-resource modeling is expected to enable an efficient mapping in between a traditional grid operation and a TES-enabled one.

A.3.1 GridModel

This diagram provides a reference architecture for modeling grid connectivity on a relational database format. The proposed design exposes a grid coupling interface that serves as a bridge to other TES components. The classes used to represent this grid model were adapted from IEEE 2030.5 and the Common Smart Inverter Profile V2.0.

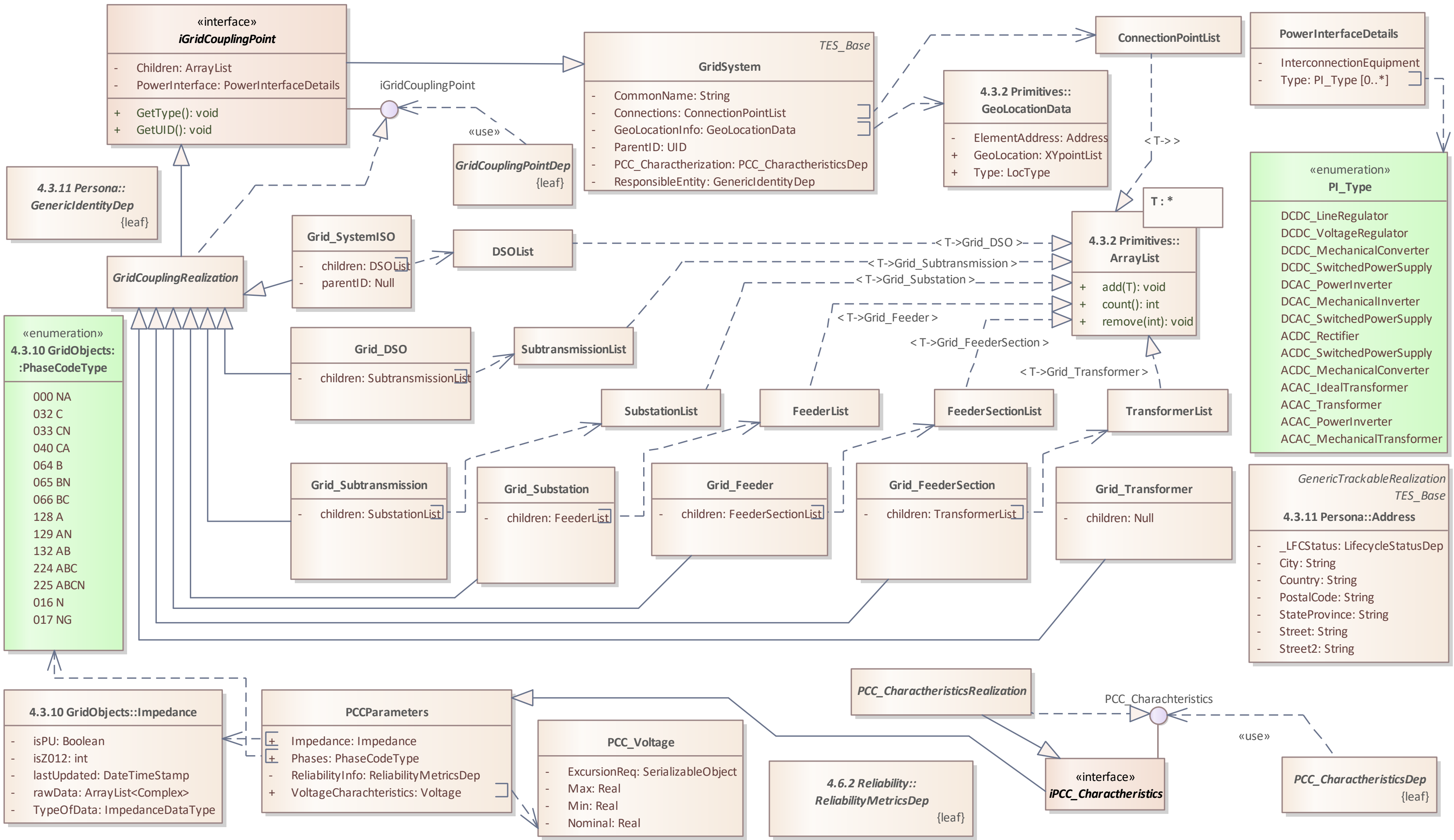


Figure 74. Overview of the GridModel' package components

A.3.1.1 ConnectionPointList

Class in package '4.5.1 GridModel'

OUTGOING STRUCTURAL RELATIONSHIPS
⇐ Realization from ConnectionPointList to ArrayList

CONNECTORS
Dependency Source -> Destination From: : GridSystem : Class , Public To: ConnectionPointList : Class , Public

A.3.1.2 DSOList

Class in package '4.5.1 GridModel'

Details: This represents a list of DSOs that are managed by the ISO.

OUTGOING STRUCTURAL RELATIONSHIPS
⇐ Realization from DSOList to ArrayList

CONNECTORS
Dependency Source -> Destination From: : Grid_SystemISO : Class , Public To: DSOList : Class , Public

A.3.1.3 FeederList

Class in package '4.5.1 GridModel'

Details: This represents a list of feeders that are connected to this grid subsystem.

OUTGOING STRUCTURAL RELATIONSHIPS
⇐ Realization from FeederList to ArrayList

CONNECTORS
Dependency Source -> Destination From: : Grid_Substation : Class , Public To: FeederList : Class , Public

A.3.1.4 FeederSectionList

Class in package '4.5.1 GridModel'

Details: This represents a list of branches/feeder sections that are connected to this grid subsystem.

OUTGOING STRUCTURAL RELATIONSHIPS
⇐ Realization from FeederSectionList to ArrayList

CONNECTORS
Dependency Source -> Destination From: : Grid_Feeder : Class , Public To: FeederSectionList : Class , Public

A.3.1.5 GenericObject

Class in package '4.5.1 GridModel'

Details: Hash
TableName
UID

CONNECTORS
Dependency Source -> Destination From: : GridSystem : Class , Public To: GenericObject : Class , Public

ATTRIBUTES
Hash : **int** Private *Details:*
TableName : **int** Private *Details:*
UID/UUID : **int** Private *Details:*

A.3.1.6 Grid_DSO

Class in package '4.5.1 GridModel'

Details: This sample implementation of the *GridCouplingPoint* represents a DSO-scale system.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from Grid_DSO to GridCouplingRealization

CONNECTORS
Dependency Source -> Destination From: : Grid_DSO : Class , Public To: SubtransmissionList : Class , Public

ATTRIBUTES
children : **SubtransmissionList** Private *Details:*

A.3.1.7 Grid_Feeder

Class in package '4.5.1 GridModel'

Details: This sample implementation of the *GridCouplingPoint* represents a Feeder-scale system.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from Grid_Feeder to GridCouplingRealization

CONNECTORS
Dependency Source -> Destination From: : Grid_Feeder : Class , Public To: FeederSectionList : Class , Public

ATTRIBUTES
children : **FeederSectionList** Private *Details:*

A.3.1.8 Grid_FeederSection

Class in package '4.5.1 GridModel'

Details: This sample implementation of the *GridCouplingPoint* represents a feeder's branch-scale system.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from Grid_FeederSection to GridCouplingRealization

CONNECTORS
Dependency Source -> Destination From: : Grid_FeederSection : Class , Public To: TransformerList : Class , Public

ATTRIBUTES
children : **TransformerList** Private *Details:*

A.3.1.9 Grid_Substation

Class in package '4.5.1 GridModel'

Details: This sample implementation of the *GridCouplingPoint* represents a substation-scale system.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from Grid_Substation to GridCouplingRealization

CONNECTORS
Dependency Source -> Destination From: : Grid_Substation : Class , Public To: FeederList : Class , Public

ATTRIBUTES
children : **FeederList** Private *Details:*

A.3.1.10 Grid_Subtransmission

Class in package '4.5.1 GridModel'

Details: This sample implementation of the *GridCouplingPoint* represents a Subtransmission-scale system.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from Grid_Subtransmission to GridCouplingRealization

CONNECTORS
Dependency Source -> Destination From: : Grid_Subtransmission : Class , Public To: SubstationList : Class , Public

ATTRIBUTES
children : **SubstationList** Private *Details:*

A.3.1.11 Grid_SystemISO

Class in package '4.5.1 GridModel'

Details: This sample implementation of the *GridCouplingPoint* represents a ISO-scale system.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from Grid_SystemISO to GridCouplingRealization

CONNECTORS
Dependency Source -> Destination From: : Grid_SystemISO : Class , Public To: DSOList : Class , Public

ATTRIBUTES
children : **DSOList** Private *Details:*
parentID : **Null** Private *Details:*

A.3.1.12 Grid_Transformer

Class in package '4.5.1 GridModel'

Details: This sample implementation of the *GridCouplingPoint* represents a distribution-scale transformer that serves low-voltage customers.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from Grid_Transformer to GridCouplingRealization

ATTRIBUTES
children : **Null** Private *Details:*

A.3.1.13 GridCouplingPointDep

Class in package '4.5.1 GridModel'

Details: Objects that reference this class expect an object that realizes the *GridCouplingPoint* Interface. This class is a leaf and is only intended to serve as a data type.

CONNECTORS
Usage Source -> Destination From: : GridCouplingPointDep : Class , Public To: iGridCouplingPoint : ProvidedInterface , Public
Dependency Source -> Destination From: : MeteringDevice : Class , Public To: GridCouplingPointDep : Class , Public

A.3.1.14 GridCouplingRealization

Class in package '4.5.1 GridModel'

Details: This abstract class implements the *GridCouplingPoint* interface which enables resources to connect to the grid, classes derived from this class should satisfy all of the service and data requirements.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from GridCouplingRealization to «interface» iGridCouplingPoint
Realization from GridCouplingRealization to iGridCouplingPoint

A.3.1.15 GridSystem

Class in package '4.5.1 GridModel'

Details: This class represents a generic grid system. This system has a geographical area that serves a list of customers. Its voltage level remains the same within the service region (unless a child/parent is navigated)

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from GridSystem to TES_Base

CONNECTORS
Dependency Source -> Destination From: : GridSystem : Class , Public To: ConnectionPointList : Class , Public
Dependency Source -> Destination From: : GridSystem : Class , Public To: GeoLocationData : Class , Public
Dependency Source -> Destination From: : GridSystem : Class , Public To: GenericObject : Class , Public

ATTRIBUTES
CommonName : **String** Private *Details:*
Connections : **ConnectionPointList** Private *Details:*
GeoLocationInfo : **GeoLocationData** Private *Details:*
ParentID : **UID** Private *Details:*
PCC_Charactherization : **PCC_CharactheristicsDep** Private *Details:*

ATTRIBUTES
ResponsibleEntity : **GenericIdentityDep** Private *Details:*

A.3.1.16 iGridCouplingPoint

Class «interface» in package '4.5.1 GridModel'

Details: This object represents a generic electrical interface. It is assumed that this interconnection point can occur at any point of the hierarchical grid structure through the use of an intermediary device.

STRUCTURAL PART OF iGridCouplingPoint
iGridCouplingPoint : ProvidedInterface

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from «interface» iGridCouplingPoint to GridSystem

ATTRIBUTES
Children : **ArrayList** Private *Details:* This field can be used to map lower hierarchy systems. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
PowerInterface : **PowerInterfaceDetails** Private *Details:* This field defines the power interface used to connect different systems. Note that no implicit system scale is assumed (e.g., a microgrid can connect to another microgrid). Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

OPERATIONS
GetType () : **void** Public *Details:* This field defines the coupling type from the grid perspective. This might proof useful when devices require to know the "scale" of the interconnection on the other side.
GetUID () : **void** Public *Details:* This field gets the coupling UID from the grid perspective.

A.3.1.17 iPCC_Charactheristics

Class «interface» in package '4.5.1 GridModel'

STRUCTURAL PART OF iPCC_Charactheristics
PCC_Charachteristics : ProvidedInterface

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from «interface» iPCC_Charactheristics to PCCParameters

A.3.1.18 PCC_CharactheristicsDep

Class in package '4.5.1 GridModel'

Details: Objects that reference this class expect an object that realizes the *PCC_Charactheristics* Interface. This class is a leaf and is only intended to serve as a data type.

CONNECTORS
Usage Source -> Destination From: : PCC_CharactheristicsDep : Class , Public To: PCC_Charachteristics : ProvidedInterface , Public

A.3.1.19 PCC_CharactheristicsRealization

Class in package '4.5.1 GridModel'

Details: This abstract class implements the *PCC_Charactheristics* interface which enables agents to retrieve grid properties at the specified location, classes derived from this class should satisfy all of the service and data requirements.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from PCC_CharactheristicsRealization to «interface» iPCC_Charactheristics
Realization from PCC_CharactheristicsRealization to PCC_Charachteristics

A.3.1.20 PCC_Voltage

Class in package '4.5.1 GridModel'

Details: This object is used to define the typical voltage characteristics for the PCC.

CONNECTORS
Dependency Source -> Destination From: : PCCParameters : Class , Public To: PCC_Voltage : Class , Public

ATTRIBUTES
ExcursionReq : **SerializableObject** Private *Details:*
Max : **Real** Private *Details:* This field represents the maximum voltage that is considered normal within this grid system (in kV). Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Min : **Real** Private *Details:* This field represents the minimum voltage that is considered normal within this grid system (in kV). Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Nominal : **Real** Private *Details:* This field represents the nominal voltage that is experienced by this grid system (in kV). Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.3.1.21 PowerInterfaceDetails

Class in package '4.5.1 GridModel'

Details: This class is used to describe the coupling type, along with the equipment used to facilitate such interconnection (i.e. a cable or a transformer).

CONNECTORS
Dependency Source -> Destination From: : PowerInterfaceDetails : Class , Public To: PI_Type : Enumeration , Public

ATTRIBUTES
InterconnectionEquipment : Private *Details:*
Type : **PI_Type** Private *Details:*

A.3.1.22 SubstationList

Class in package '4.5.1 GridModel'

Details: This represents a list of substations that are connected to this grid subsystem.

OUTGOING STRUCTURAL RELATIONSHIPS
Realization from SubstationList to ArrayList

CONNECTORS
Dependency Source -> Destination From: : Grid_Subtransmission : Class , Public To: SubstationList : Class , Public

A.3.1.23 SubtransmissionList

Class in package '4.5.1 GridModel'

Details: This represents a list of sub transmissions systems that are managed by the DSO.

OUTGOING STRUCTURAL RELATIONSHIPS
Realization from SubtransmissionList to ArrayList

CONNECTORS
Dependency Source -> Destination From: : Grid_DSO : Class , Public To: SubtransmissionList : Class , Public

A.3.1.24 TransformerList

Class in package '4.5.1 GridModel'

Details: This represents a list of transformers that are connected to this grid subsystem.

OUTGOING STRUCTURAL RELATIONSHIPS
Realization from TransformerList to ArrayList

CONNECTORS
Dependency Source -> Destination From: : Grid_FeederSection : Class , Public To: TransformerList : Class , Public

A.3.1.25 PCCParameters

Class in package '4.5.1 GridModel'

Details: This class lists typical PCC parameters that can be used to decide the suitability of an interconnection point.

CONNECTORS
Dependency Source -> Destination From: : PCCParameters : Class , Public To: Impedance : Class , Public
Dependency Source -> Destination From: : PCCParameters : Class , Public To: PCC_Voltage : Class , Public
Dependency Source -> Destination From: : PCCParameters : Class , Public To: PhaseCodeType : Enumeration , Public

ATTRIBUTES
Impedance : **Impedance** Public *Details:* This field represents the electrical impedance present at the PCC. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Phases : **PhaseCodeType** Public *Details:* This field represents the electrical phases present at the PCC. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
ReliabilityInfo : **ReliabilityMetricsDep** Private *Details:* This field captures the reliability info at the specified PCC. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
VoltageCharachteristics : **Voltage** Public *Details:* This field captures the typical voltage characteristics at this PCC Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.3.1.26 PI_Type

Enumeration in package '4.5.1 GridModel'

Details: This enumeration lists possible interconnection types according to the type of systems being connected (AC/DC).

CONNECTORS
Dependency Source -> Destination From: : PowerInterfaceDetails : Class , Public To: PI_Type : Enumeration , Public

ENUMERATION:
DCDC_LineRegulator
DCDC_VoltageRegulator
DCDC_MechanicalConverter
DCDC_SwitchedPowerSupply
DCAC_PowerInverter
DCAC_MechanicalInverter

ENUMERATION:
DCAC_SwitchedPowerSupply
ACDC_Rectifier
ACDC_SwitchedPowerSupply
ACDC_MechanicalConverter
ACAC_IdealTransformer
ACAC_Transformer
ACAC_PowerInverter
ACAC_MechanicalTransformer

A.4 Smart Contract Modeling and Support Services

In this section a series of auxiliary grid monitoring services will be introduced (from a modeling perspective). These services are expected to assist with the measurement, verification and eventual settlement of TES-based transactions.

A.4.1 Reliability

This diagram provides a reference implementation of a data interface that can be used to capture grid reliability data. This interface can be leveraged to provide additional details to market and monitoring applications that run on top of the TES stack.

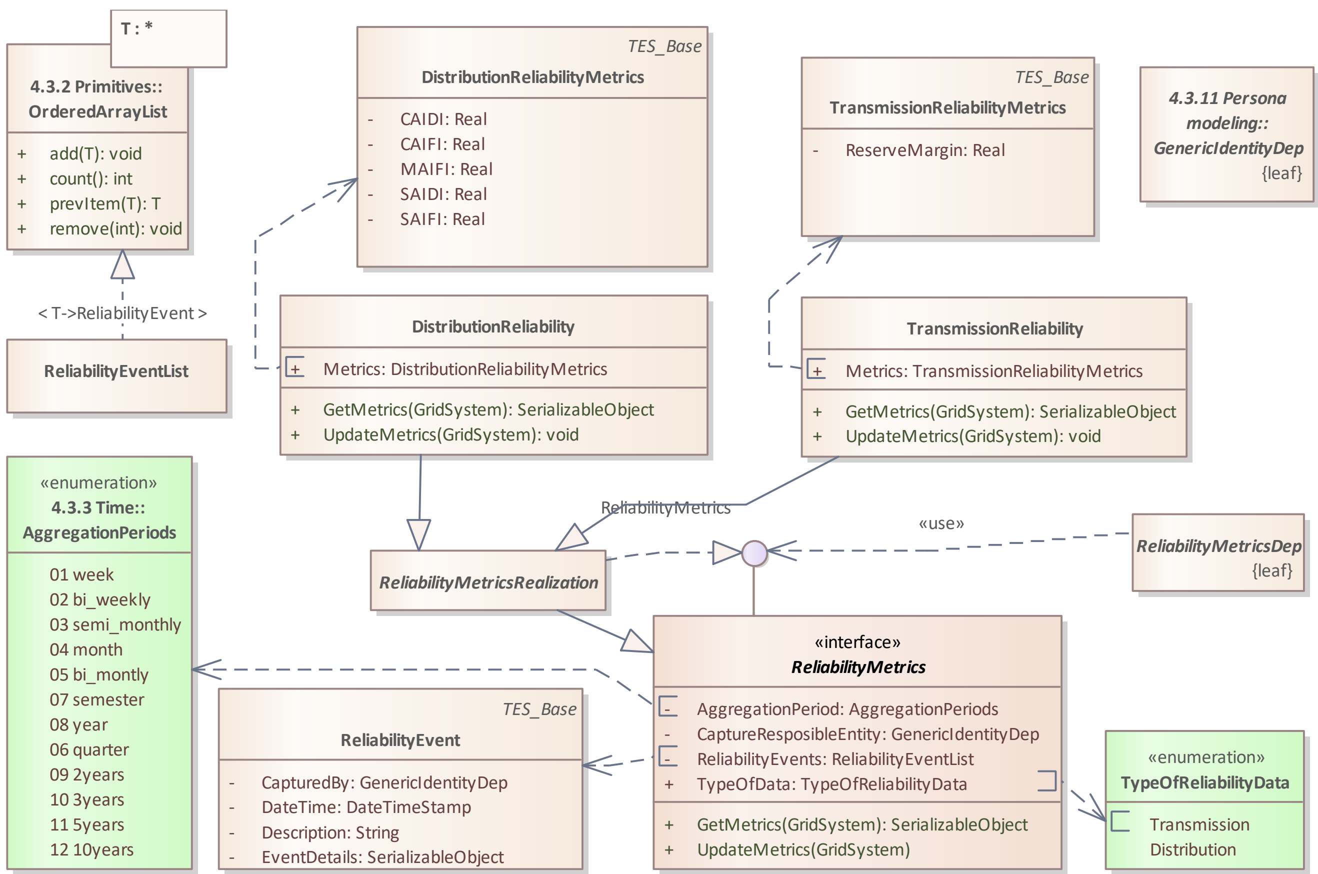


Figure 75. Overview of the Reliability' package components

A.4.1.1 DistributionReliability

Class in package '4.6.2 Reliability'

Details: This object serves as a sample for realizing the *ReliabilityMetrics* interface in distribution systems. In this case, both the underlying metrics and the update/reporting functions are customized to suit the needs of a typical DSO.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from DistributionReliability to ReliabilityMetricsRealization

CONNECTORS
Dependency Source -> Destination From: : DistributionReliability : Class , Public To: DistributionReliabilityMetrics : Class , Public

ATTRIBUTES
Metrics : **DistributionReliabilityMetrics** Public *Details:*

OPERATIONS
GetMetrics (GridPoint : **GridSystem**) : **SerializableObject** Public *Details:* This function encodes the distribution metrics into a *SerializableObject* that the dependent assembly can parse.
UpdateMetrics (GridPoint : **GridSystem**) : **void** Public *Details:* This function can be periodically called to update the system metrics.

A.4.1.2 DistributionReliabilityMetrics

Class in package '4.6.2 Reliability'

Details: This object represents the typical distribution-scale reliability metrics.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from DistributionReliabilityMetrics to TES_Base

CONNECTORS
Dependency Source -> Destination From: : DistributionReliability : Class , Public To: DistributionReliabilityMetrics : Class , Public

ATTRIBUTES
CAIDI : **Real** Private *Details:* Customer Average Interruption Duration Index. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
CAIFI : **Real** Private *Details:* Customer Average Interruption Frequency Index. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
MAIFI : **Real** Private *Details:* Momentary Average Interruption Frequency Index. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
SAIDI : **Real** Private *Details:* System Average Interruption Duration Index. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
SAIFI : **Real** Private *Details:* System Average Interruption Frequency Index. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.4.1.3 ReliabilityEvent

Class in package '4.6.2 Reliability'

Details: This represents a single reliability event, the event has enough metadata to trace the source and event time (along with the event data).

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from ReliabilityEvent to TES_Base

CONNECTORS
Dependency Source -> Destination From: : ReliabilityMetrics : Class , Public To: ReliabilityEvent : Class , Public

ATTRIBUTES
CapturedBy : **GenericIdentityDep** Private *Details:* This field is used to indicate the identity of the agent that has reported this event. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
DateTime : **DateTimeStamp** Private *Details:* This field encodes the date/time at which this event was captured. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Description : **String** Private *Details:* This field can be used to provide a text-based description of the event. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
EventDetails : **SerializableObject** Private *Details:* This field contains the actual event data. It is expected that the *UpdateMetrics* function can interpret the data as intended. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.4.1.4 ReliabilityEventList

Class in package '4.6.2 Reliability'

OUTGOING STRUCTURAL RELATIONSHIPS
Realization from ReliabilityEventList to OrderedArrayList
Realization from ReliabilityEventList to ArrayList

A.4.1.5 ReliabilityMetrics

Class «interface» in package '4.6.2 Reliability'

Details: This object represents the interface requirements that a reliability data producer/consumer must implement. It abstracts the type of system by relying on a serialized object to provide data exchanges, along with a data descriptor (*TypeOfReliabilityData*).

STRUCTURAL PART OF ReliabilityMetrics
ReliabilityMetrics : ProvidedInterface

CONNECTORS
Dependency Source -> Destination From: : ReliabilityMetrics : Class , Public To: ReliabilityEvent : Class , Public

CONNECTORS
Dependency Source -> Destination From: : ReliabilityMetrics : Class , Public To: TypeOfReliabilityData : Enumeration , Public
Dependency Source -> Destination From: : ReliabilityMetrics : Class , Public To: AggregationPeriods : Enumeration , Public

ATTRIBUTES
AggregationPeriod : **AggregationPeriods** Private *Details:* This represents the aggregation period over which statistical records are computed. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
CaptureResposibleEntity : **GenericIdentityDep** Private *Details:* This field defines the entity that is responsible for capturing the reliability metrics. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
ReliabilityEvents : **ReliabilityEventList** Private *Details:* This ordered, arrayList can be used to store past reliability events. This type of field can be thought as a historical data store. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
TypeOfData : **TypeOfReliabilityData** Public *Details:* This field describes the data being provided to the reliability subscriber. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

OPERATIONS
GetMetrics (GridPoint : **GridSystem**) : **SerializableObject** Public *Details:* This function can be used to retrieve the system reliability metrics using a serializable data container.
UpdateMetrics (GridPoint : **GridSystem**) : Public *Details:* This function enables subscribers to request data updates on-demand.

A.4.1.6 ReliabilityMetricsDep

Class in package '4.6.2 Reliability'

Details: Objects that reference this class expect an object that realizes the *ReliabilityMetrics* Interface. This class is a leaf and is only intended to serve as a data type.

CONNECTORS
Usage Source -> Destination From: : ReliabilityMetricsDep : Class , Public To: ReliabilityMetrics : ProvidedInterface , Public

A.4.1.7 ReliabilityMetricsRealization

Class in package '4.6.2 Reliability'

Details: This abstract class implements the *ReliabilityMetrics* interface which enables agents to get reliability information, independent from the system or point of interconnection. Classes derived from this class should satisfy all of the service and data requirements.

OUTGOING STRUCTURAL RELATIONSHIPS
Realization from ReliabilityMetricsRealization to ReliabilityMetrics
Generalization from ReliabilityMetricsRealization to «interface» ReliabilityMetrics

A.4.1.8 TransmissionReliability

Class in package '4.6.2 Reliability'

Details: This object serves as a sample for realizing the *ReliabilityMetrics* interface in transmission systems. In this case, both the underlying metrics and the update/reporting functions are customized to suit the needs of NERC reporting requirements.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from TransmissionReliability to ReliabilityMetricsRealization

CONNECTORS
Dependency Source -> Destination From: : TransmissionReliability : Class , Public To: TransmissionReliabilityMetrics : Class , Public

ATTRIBUTES
Metrics : **TransmissionReliabilityMetrics** Public *Details:* Percentage of additional capacity over load Reserve Margin (%) = (Capacity – Load)/Load X 100 Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

OPERATIONS
GetMetrics (GridPoint : **GridSystem**) : **SerializableObject** Public *Details:* This function encodes the transmission metrics into a *SerializableObject* that the dependent assembly can parse.
UpdateMetrics (GridPoint : **GridSystem**) : **void** Public *Details:* This function can be periodically called to update the system metrics.

A.4.1.9 TransmissionReliabilityMetrics

Class in package '4.6.2 Reliability'

Details: This object represents the typical transmission-scale reliability metrics.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from TransmissionReliabilityMetrics to TES_Base

CONNECTORS
Dependency Source -> Destination From: : TransmissionReliability : Class , Public To: TransmissionReliabilityMetrics : Class , Public

ATTRIBUTES
ReserveMargin : **Real** Private *Details:* This number indicates the amount of reserves according to NERC Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.4.1.10 TypeOfReliabilityData

Enumeration in package '4.6.2 Reliability'

Details: This enumeration can be used to identify the type of reliability data being reported.

CONNECTORS
Dependency Source -> Destination From: : ReliabilityMetrics : Class , Public To: TypeOfReliabilityData : Enumeration , Public

ENUMERATION:
Transmission
Distribution

A.4.2 Measurement and Verification

This diagram contains a variety of data models that can be used to record a variety of commodities, quantities, using a wide variety of data aggregation methods. It is expected that attestation-capable resources will be responsible for capturing this data, while a mixture of on-chain and of-chain methods will be used to capture the streams of data.

Most of the data models introduced by this section are based on the models contained in IEEE 2030.5.

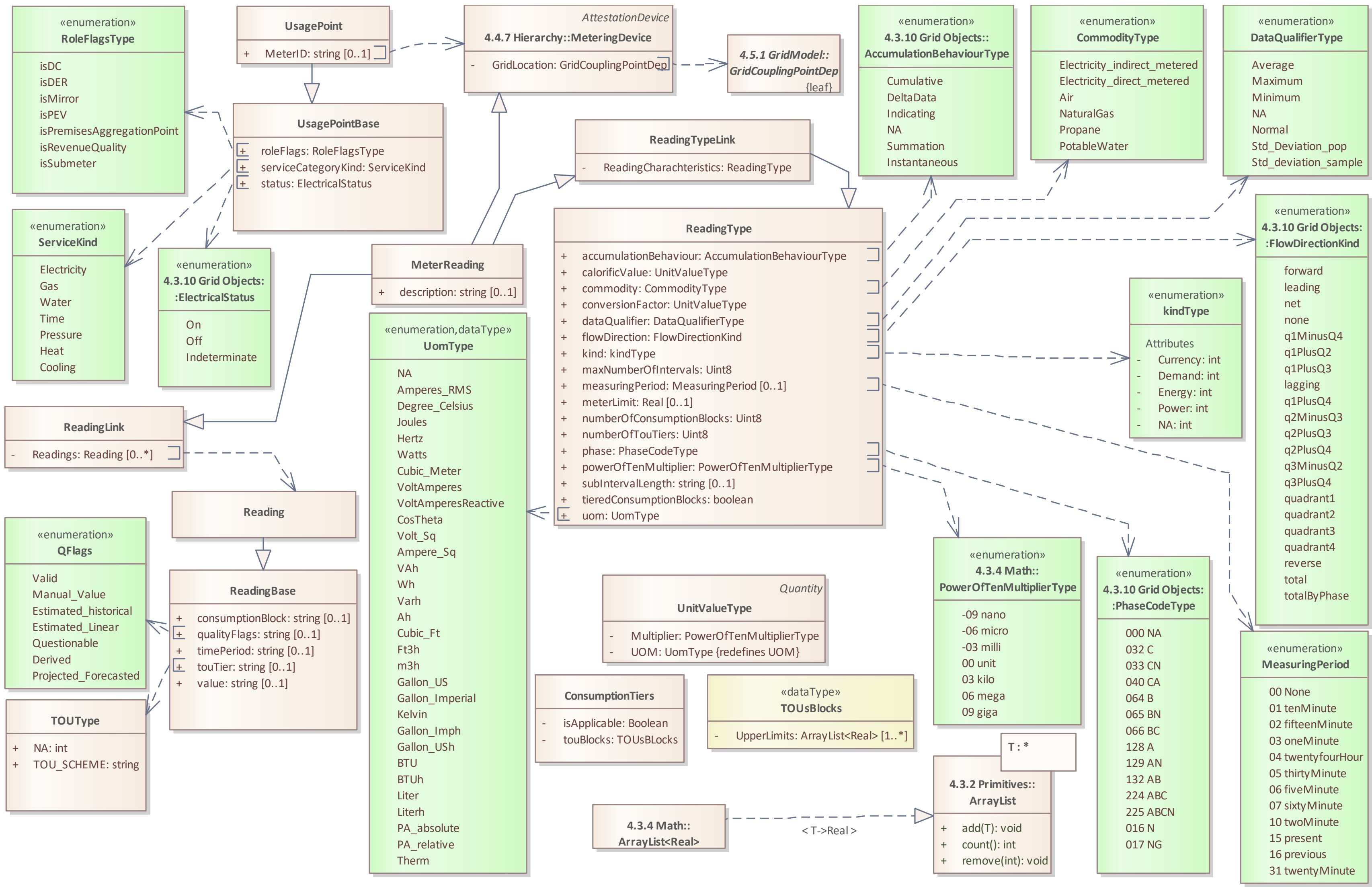


Figure 76. Overview of the MeasurementandVerification' package components

A.4.2.1 ConsumptionTiers

Class in package '4.6.1 Measurement and Verification'

Details: This class allows the TES market operator to define the different pricing blocks that are in use. These are usually broken according to the consumption over a period of time. This object model was taken from IEEE 2030.5

ATTRIBUTES
isApplicable : **Boolean** Private *Details:*
touBlocks : **TOUsBLocks** Private *Details:*

A.4.2.2 IdentifiedObject

Class in package '4.6.1 Measurement and Verification'

Details: This is a root class to provide common naming attributes for all classes needing naming attributes

ATTRIBUTES
description : **string** Public *Details:* The description is a human readable text describing or naming the object. Multiplicity: (0..1, Allow duplicates: 0, Is ordered: False) Properties: maxOccurs = 1 minOccurs = 0
mRID : **string** Public *Details:* The global identifier of the object. Multiplicity: (1, Allow duplicates: 0, Is ordered: False) Properties: maxOccurs = 1 minOccurs = 1
version : **string** Public *Details:* Contains the version number of the object. See the type definition for details. Multiplicity: (0..1, Allow duplicates: 0, Is ordered: False) Properties: maxOccurs = 1 minOccurs = 0

A.4.2.3 MeterReading

Class in package '4.6.1 Measurement and Verification'

Details: This class holds the quantities, and associated properties obtained from the meter. It is recommended that these data readings are stored outside the blockchain network, although periodic checkpoints can be stored or hashed into the ledger.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from MeterReading to ReadingTypeLink
Generalization from MeterReading to MeteringDevice
Generalization from MeterReading to ReadingLink

ATTRIBUTES
description : **string** Public *Details:* The description is a human readable text describing or naming the object. Multiplicity: (0..1, Allow duplicates: 0, Is ordered: False) Properties: maxOccurs = 1 minOccurs = 0

A.4.2.4 Reading

Class in package '4.6.1 Measurement and Verification'

Details: This class contains the specific value measured by a meter or other recording asset. Adapted from IEEE 2030.5

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from Reading to ReadingBase

CONNECTORS
Dependency Source -> Destination From: : ReadingLink : Class , Public To: Reading : Class , Public

A.4.2.5 ReadingBase

Class in package '4.6.1 Measurement and Verification'

Details: This class stores the actual reading. The class augments a captured value by adding metadata related to its quality and time of acquisition.

CONNECTORS
Dependency Source -> Destination From: : ReadingBase : Class , Public To: TOUType : Class , Public
Dependency Source -> Destination From: : ReadingBase : Class , Public To: QFlags : Enumeration , Public

ATTRIBUTES
consumptionBlock : **string** Public *Details:* Indicates the consumption block related to the reading. REQUIRED if ReadingType numberOfConsumptionBlocks is non-zero. If not specified, is assumed to be "0 - N/A". Multiplicity: (0..1, Allow duplicates: 0, Is ordered: False) Properties: maxOccurs = 1 minOccurs = 0
qualityFlags : **string** Public *Details:* List of codes indicating the quality of the reading, using specification: Bit 0 - valid: data that has gone through all required validation checks and either passed them all or has been verified Bit 1 - manually edited: Replaced or approved by a human Bit 2 - estimated using reference day: data value was replaced by a machine computed value based on analysis of historical data using the same type of measurement. Bit 3 - estimated using linear interpolation: data value was computed using linear interpolation based on the readings before and after it Bit 4 - questionable: data that has failed one or more checks Bit 5 - derived: data that has been calculated (using logic or mathematical operations), not necessarily measured directly Bit 6 - projected (forecast): data that has been calculated as a projection or forecast of future readings Multiplicity: (0..1, Allow duplicates: 0, Is ordered: False) Properties: maxOccurs = 1 minOccurs = 0
timePeriod : **string** Public *Details:* The time interval associated with the reading. If not specified, then defaults to the intervalLength specified in the associated ReadingType.

ATTRIBUTES
Multiplicity: (0..1, Allow duplicates: 0, Is ordered: False) Properties: maxOccurs = 1 minOccurs = 0
touTier : **string** Public *Details:* Indicates the time of use tier related to the reading. REQUIRED if ReadingType numberOfTouTiers is non-zero. If not specified, is assumed to be "0 - N/A". Multiplicity: (0..1, Allow duplicates: 0, Is ordered: False) Properties: maxOccurs = 1 minOccurs = 0
value : **string** Public *Details:* Value in units specified by ReadingType Multiplicity: (0..1, Allow duplicates: 0, Is ordered: False) Properties: maxOccurs = 1 minOccurs = 0

A.4.2.6 ReadingLink

Class in package '4.6.1 Measurement and Verification'

Details: A Link to a list of readings. These readings should be stored off the blockchain.

CONNECTORS
Dependency Source -> Destination From: : ReadingLink : Class , Public To: Reading : Class , Public

ATTRIBUTES
Readings : **Reading** Private *Details:*

A.4.2.7 ReadingType

Class in package '4.6.1 Measurement and Verification'

Details: This structure serves to define a reading's characteristics. These characteristics are set once per type of reading. The base class was adopted from IEEE 2030.5

CONNECTORS
Dependency Source -> Destination From: : ReadingType : Class , Public To: AccumulationBehaviourType : Enumeration , Public
Dependency Source -> Destination From: : ReadingType : Class , Public To: UomType : DataType , Public
Dependency Source -> Destination From: : ReadingType : Class , Public To: DataQualifierType : Enumeration , Public
Dependency Source -> Destination From: : ReadingType : Class , Public To: UomType : DataType , Public

CONNECTORS
Dependency Source -> Destination From: : ReadingType : Class , Public To: kindType : Enumeration , Public
Dependency Source -> Destination From: : ReadingType : Class , Public To: FlowDirectionKind : Enumeration , Public
Dependency Source -> Destination From: : ReadingType : Class , Public To: MeasuringPeriod : Enumeration , Public
Dependency Source -> Destination From: : ReadingType : Class , Public To: PhaseCodeType : Enumeration , Public
Dependency Source -> Destination From: : ReadingType : Class , Public To: PowerOfTenMultiplierType : Enumeration , Public
Dependency Source -> Destination From: : ReadingType : Class , Public To: CommodityType : Enumeration , Public

ATTRIBUTES
accumulationBehaviour : **AccumulationBehaviourType** Public *Details:* The "accumulation behaviour" indicates how the value is represented to accumulate over time. Multiplicity: (1, Allow duplicates: 0, Is ordered: False) Properties: maxOccurs = 1 minOccurs = 0
calorificValue : **UnitValueType** Public *Details:* The amount of heat generated when a given mass of fuel is completely burned. The CalorificValue is used to convert the measured volume or mass of gas into kWh. The CalorificValue attribute represents the current active value. Multiplicity: (1, Allow duplicates: 0, Is ordered: False) Properties: maxOccurs = 1 minOccurs = 0
commodity : **CommodityType** Public *Details:* Indicates the commodity applicable to this ReadingType. Multiplicity: (1, Allow duplicates: 0, Is ordered: False) Properties: maxOccurs = 1 minOccurs = 0
conversionFactor : **UnitValueType** Public *Details:* Accounts for changes in the volume of gas based on temperature and pressure. The ConversionFactor attribute represents the current active value. The ConversionFactor is dimensionless. The default value for the ConversionFactor is 1, which means no conversion is applied. A price server can advertise a new/different value at any time. Multiplicity: (1, Allow duplicates: 0, Is ordered: False) Properties: maxOccurs = 1 minOccurs = 0
dataQualifier : **DataQualifierType** Public *Details:* The data type can be used to describe a salient attribute of the data. Possible values are average, absolute, and etc. Multiplicity: (1, Allow duplicates: 0, Is ordered: False) Properties: maxOccurs = 1 minOccurs = 0
flowDirection : **FlowDirectionKind** Public *Details:* Anything involving current might have a flow direction. Possible values include forward and reverse. Multiplicity: (1, Allow duplicates: 0, Is ordered: False) Properties: maxOccurs = 1 minOccurs = 0
kind : **kindType** Public *Details:* Compound class that contains kindCategory and kindIndex Multiplicity: (1, Allow duplicates: 0, Is ordered: False) Properties: maxOccurs = 1 minOccurs = 0

ATTRIBUTES
maxNumberOfIntervals : **Uint8** Public *Details:* To be populated for mirrors of interval data to set the expected number of intervals per ReadingSet. Servers may discard intervals received that exceed this number. Multiplicity: (1, Allow duplicates: 0, Is ordered: False) Properties: maxOccurs = 1 minOccurs = 0
measuringPeriod : **MeasuringPeriod** Public *Details:* Default interval length specified in seconds. Eq. to the Measuringperiod in IEC Multiplicity: (0..1, Allow duplicates: 0, Is ordered: False) Properties: maxOccurs = 1 minOccurs = 0
meterLimit : **Real** Public *Details:* SupplyMeter->meterLimit. Reflect the max amount of X that can reliably be measured. Reflects the supply limit set in the meter. This value can be compared to the Reading value to understand if limits are being approached or exceeded. Units follow the same definition as in this ReadingType. Multiplicity: (0..1, Allow duplicates: 0, Is ordered: False) Properties: maxOccurs = 1 minOccurs = 0
numberOfConsumptionBlocks : **Uint8** Public *Details:* Number of consumption blocks. 0 means not applicable, and is the default if not specified. The value needs to be at least 1 if any actual prices are provided. Multiplicity: (1, Allow duplicates: 0, Is ordered: False) Properties: maxOccurs = 1 minOccurs = 0
numberOfTouTiers : **Uint8** Public *Details:* The number of TOU tiers that can be used by any resource configured by this ReadingType. Servers SHALL populate this value with the largest touTier value that will ever be used while this ReadingType is in effect. Servers SHALL set numberOfTouTiers equal to the number of standard TOU tiers plus the number of CPP tiers that may be used while this ReadingType is in effect. Servers SHALL specify a value between 0 and 255 (inclusive) for numberOfTouTiers (servers providing flat rate pricing SHOULD set numberOfTouTiers to 0, as in practice there is no difference between having no tiers and having one tier). Multiplicity: (1, Allow duplicates: 0, Is ordered: False) Properties: maxOccurs = 1 minOccurs = 0
phase : **PhaseCodeType** Public *Details:* Contains phase information associated with the type. Multiplicity: (1, Allow duplicates: 0, Is ordered: False) Properties: maxOccurs = 1 minOccurs = 0
powerOfTenMultiplier : **PowerOfTenMultiplierType** Public *Details:* Indicates the power of ten multiplier applicable to the unit of measure of this ReadingType. Multiplicity: (1, Allow duplicates: 0, Is ordered: False) Properties: maxOccurs = 1 minOccurs = 0
subIntervalLength : **string** Public *Details:* Default sub-interval length specified in seconds for Readings of ReadingType. Some demand calculations are done over a number of smaller intervals. For example, in a rolling demand calculation, the demand value is defined as the rolling sum of smaller intervals over the intervalLength. The subintervalLength is the length of the smaller interval in this calculation. It SHALL be an integral division of the intervalLength. The number of sub-intervals can be calculated by dividing the intervalLength by the subintervalLength. Multiplicity: (0..1, Allow duplicates: 0, Is ordered: False) Properties: maxOccurs = 1 minOccurs = 0
tieredConsumptionBlocks : **boolean** Public *Details:* Specifies whether or not the consumption blocks are differentiated by TOUTier or not. Default is false, if not specified. true = consumption accumulated over individual tiers

ATTRIBUTES
false = consumption accumulated over all tiers Multiplicity: (1, Allow duplicates: 0, Is ordered: False) Properties: maxOccurs = 1 minOccurs = 0
uom : **UomType** Public *Details:* Indicates the measurement type for the units of measure for the readings of this type. Multiplicity: (1, Allow duplicates: 0, Is ordered: False) Properties: maxOccurs = 1 minOccurs = 0

A.4.2.8 ReadingTypeLink

Class in package '4.6.1 Measurement and Verification'

Details: This class is used as a placeholder for describing the reading type (as defined by IEEE 2030.5)

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from ReadingTypeLink to ReadingType

ATTRIBUTES
ReadingCharachteristics : **ReadingType** Private *Details:*

A.4.2.9 TOUType

Class in package '4.6.1 Measurement and Verification'

Details: This class can be used to define the Time Of Use scheme to be used in price calculation.

CONNECTORS
Dependency Source -> Destination From: : ReadingBase : Class , Public To: TOUType : Class , Public

ATTRIBUTES
NA : **int** Public *Details:* Properties: maxOccurs = 1 minOccurs = 1
TOU_SCHEME : **string** Public *Details:* Properties: maxOccurs = 1 minOccurs = 1

A.4.2.10 UnitValueType

Class in package '4.6.1 Measurement and Verification'

Details: This class is an specialization of the quantity data type. It introduces a power of ten multiplier and re-defines the UOM field.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from UnitValueType to Quantity

ATTRIBUTES
Multiplier : **PowerOfTenMultiplierType** Private *Details:* This field contains the power of ten multiplier by which the value must be multiplied to obtain the actual value Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
UOM : **UomType** Private *Details:* This field can be used to identify the Unit Of Measure that is being used to report the value. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.4.2.11 UsagePoint

Class in package '4.6.1 Measurement and Verification'

Details: This class is used to document the measuring point identifier. This identifier allows end-users to abstract the grid model from the measuring device data feed.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from UsagePoint to UsagePointBase

CONNECTORS
Dependency Source -> Destination From: : UsagePoint : Class , Public To: MeteringDevice : Class , Public

ATTRIBUTES
MeterID : **string** Public *Details:* The UID of the source device. This attribute SHALL be present when mirroring. Multiplicity: (0..1, Allow duplicates: 0, Is ordered: False) Properties: maxOccurs = 1 minOccurs = 0

A.4.2.12 UsagePointBase

Class in package '4.6.1 Measurement and Verification'

Details: This class is used to define the characteristics of the metering point.

CONNECTORS
Dependency Source -> Destination From: : UsagePointBase : Class , Public To: ElectricalStatus : Enumeration , Public
Dependency Source -> Destination From: : UsagePointBase : Class , Public To: RoleFlagsType : Enumeration , Public
Dependency Source -> Destination From: : UsagePointBase : Class , Public To: ServiceKind : Enumeration , Public

ATTRIBUTES
roleFlags : **RoleFlagsType** Public *Details:* Specifies the roles that apply to the usage point. Multiplicity: (1, Allow duplicates: 0, Is ordered: False) Properties: anonymousRole = true default = fixed = form =

ATTRIBUTES
maxOccurs = 1 minOccurs = 1
serviceCategoryKind : **ServiceKind** Public *Details:* The kind of service provided by this usage point. Multiplicity: (1, Allow duplicates: 0, Is ordered: False) Properties: maxOccurs = 1 minOccurs = 1
status : **ElectricalStatus** Public *Details:* Specifies the current status of the service at this usage point. 0 = off 1 = on Multiplicity: (1, Allow duplicates: 0, Is ordered: False) Properties: maxOccurs = 1 minOccurs = 1

A.4.2.13 CommodityType

Enumeration in package '4.6.1 Measurement and Verification'

Details: This enumeration was taken from IEEE 2030.5. It can be used to identify the type of commodity being measured.

CONNECTORS
Dependency Source -> Destination From: : ReadingType : Class , Public To: CommodityType : Enumeration , Public

ENUMERATION:
Electricity_indirect_metered
Electricity_direct_metered
Air
NaturalGas
Propane
PotableWater

A.4.2.14 DataQualifierType

Enumeration in package '4.6.1 Measurement and Verification'

Details: This enumeration can be used to specify the data sampling mechanism used to capture the data (if applicable). This enumeration was taken from IEEE 2030.5.

CONNECTORS
Dependency Source -> Destination From: : ReadingType : Class , Public To: DataQualifierType : Enumeration , Public

ENUMERATION:	
Average	Data readings are averaged.
Maximum	Data reading is the maximum value observed.
Minimum	Data reading is the minimum value observed.
NA	
Normal	The value reported is the actual value.
Std_Deviation_pop	

ENUMERATION:
Std_deviation_sample

A.4.2.15 kindType

Enumeration in package '4.6.1 Measurement and Verification'

Details: This enumeration is used to specify the type of measurement that is being reported. This enumeration was taken from IEEE 2030.5.

CONNECTORS
Dependency Source -> Destination From: : ReadingType : Class , Public To: kindType : Enumeration , Public

ENUMERATION:
Currency
Demand
Energy
Power
NA

A.4.2.16 MeasuringPeriod

Enumeration in package '4.6.1 Measurement and Verification'

Details: This enumeration can be used to describe the aggregation time over which the measurement is reported. This enumeration was obtained from IEEE 2030.5

CONNECTORS
Dependency Source -> Destination From: : ReadingType : Class , Public To: MeasuringPeriod : Enumeration , Public

ENUMERATION:
00 None
01 tenMinute
02 fifteenMinute
03 oneMinute
04 twentyfourHour
05 thirtyMinute
06 fiveMinute
07 sixtyMinute
10 twoMinute
15 present
16 previous
31 twentyMinute

A.4.2.17 QFlags

Enumeration in package '4.6.1 Measurement and Verification'

Details: This enumeration can be used to describe the quality properties of an individual reading.

CONNECTORS
Dependency Source -> Destination From: : ReadingBase : Class , Public To: QFlags : Enumeration , Public

ENUMERATION:
Valid
Manual_Value
Estimated_historical
Estimated_Linear
Questionable
Derived
Projected_Forecasted

A.4.2.18 RoleFlagsType

Enumeration in package '4.6.1 Measurement and Verification'

Details: This enumeration can be used to describe the meter type. This can be a standalone system or be integrated into another device. This enumeration was taken from IEEE 2030.5

CONNECTORS
Dependency Source -> Destination From: : UsagePointBase : Class , Public To: RoleFlagsType : Enumeration , Public

ENUMERATION:
isDC
isDER
isMirror
isPEV
isPremisesAggregationPoint
isRevenueQuality
isSubmeter

A.4.2.19 ServiceKind

Enumeration in package '4.6.1 Measurement and Verification'

Details: This enumeration can be used to describe the type of service that is being measured, for the B-A TES framework it is assumed that electricity is the primary kind. This enumeration was taken from IEEE 2030.5

CONNECTORS
Dependency Source -> Destination From: : UsagePointBase : Class , Public To: ServiceKind : Enumeration , Public

ENUMERATION:
Electricity
Gas
Water
Time
Pressure
Heat

ENUMERATION:
Cooling

A.4.2.20 TOUsBlocks

DataType in package '4.6.1 Measurement and Verification'

Details: This data type can be used to define multiple TimeOfUse data blocks, useful for calculating total costs in markets with fixed tariffs.

ATTRIBUTES
UpperLimits : **ArrayList<Real>** Private *Details:*

A.4.2.21 UomType

DataType «dataType» in package '4.6.1 Measurement and Verification'

Details: The enumeration provides unit of measurement that are intended for electricity applications. The values are listed in IEEE 2030.5, and are themselves sourced from IEC 61968-9 [61968]. Other case-specific units of measure may be added.

CONNECTORS
Dependency Source -> Destination From: : ReadingType : Class , Public To: UomType : DataType , Public

ENUMERATION:
NA
Amperes_RMS
Degree_Celsius
Joules
Hertz
Watts
Cubic_Meter
VoltAmperes
VoltAmperesReactive
CosTheta
Volt_Sq
Ampere_Sq
VAh
Wh
Varh
Ah
Cubic_Ft
Ft3h
m3h
Gallon_US
Gallon_Imperial
Kelvin
Gallon_Imph
Gallon_USh
BTU
BTUh
Liter
Literh
PA_absolute

ENUMERATION:
PA_relative
Therm

A.4.2.22 UomType

DataType in package '4.6.1 Measurement and Verification'

OUTGOING STRUCTURAL RELATIONSHIPS
Realization from UomType to «dataType» UomType

CONNECTORS
Dependency Source -> Destination From: : ReadingType : Class , Public To: UomType : DataType , Public

A.4.2.23 CommodityType - Copy

Enumeration in package '4.6.1 Measurement and Verification'

Details: 0 = Not Applicable (default, if not specified)
1 = Electricity secondary metered value (a premises meter is typically on the low voltage, or secondary, side of a service transformer)
2 = Electricity primary metered value (measured on the high voltage, or primary, side of the service transformer)
4 = Air
7 = NaturalGas
8 = Propane
9 = PotableWater
10 = Steam
11 = WasteWater
12 = HeatingFluid
13 = CoolingFluid
All other values reserved.

ENUMERATION:
Electricity_indirect_metered
Electricity_direct_metered
Air
NaturalGas
Propane
PotableWater

A.4.3 Smart Contracts

This diagram presents an overview of the components found within a smart contract. Most of the information of this model is abstract, and its functionality must be defined by the underlying blockchain and unique application requirements.

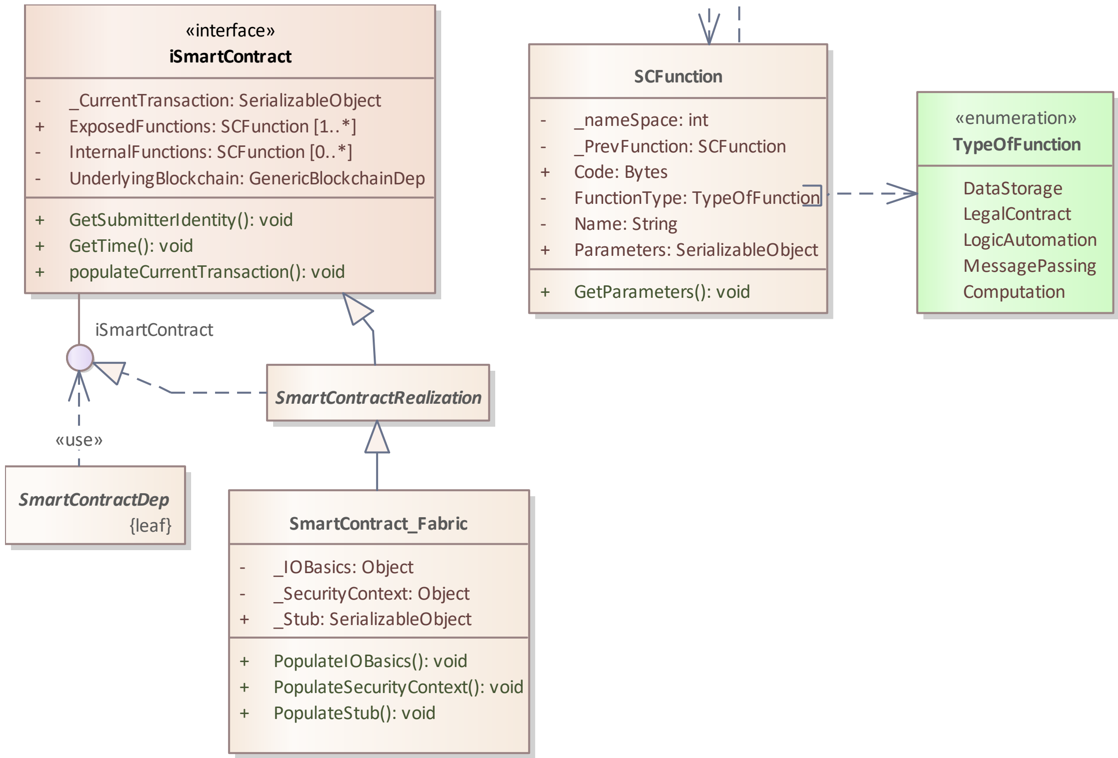


Figure 77. Overview of the SmartContracts' package components

A.4.3.1 iSmartContract

Class «interface» in package '4.6.3 Smart Contracts'

Details: This interface summarizes the properties and capabilities of any smart contract. A smart contract current world-state depends on the underlying blockchain implementation, while modifications are dictated by the functions/procedures stored within.
It is likely that this basic properties can be extended or redefined depending on the actual blockchain being used.

STRUCTURAL PART OF iSmartContract
iSmartContract : ProvidedInterface

CONNECTORS
Dependency Source -> Destination From: : iSmartContract : Class , Public To: GenericBlockchainDep : Class , Public

ATTRIBUTES
_CurrentTransaction : **SerializableObject** Private *Details:* This is a serialized version of the transaction. This info can be used to extract other properties such as the agent that submitted the transaction and the time at which was created. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
ExposedFunctions : **SCFunction** Public *Details:* This represents functions which can be called by the blockchain network via a transaction request. Multiplicity: (1..*, Allow duplicates: 0, Is ordered: False)
InternalFunctions : **SCFunction** Private *Details:* These are internal functions who remain hidden to the blockchain network but can be called by *exposed* or *internal* functions. Multiplicity: (0..*, Allow duplicates: 0, Is ordered: False)
UnderlyingBlockchain : **GenericBlockchainDep** Private *Details:* This contains a reference to the underlying blockchain implementation. It contains its description, underlying ledger and the functions needed to access it. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

OPERATIONS
GetSubmitterIdentity () : **void** Public *Details:* This function parses the transaction request to determine the identity of the submission agent.
GetTime () : **void** Public *Details:* This function parses the transaction request to determine the time at which the current transaction was submitted/received.
populateCurrentTransaction () : **void** Public *Details:* This function loads the transaction request into a local object. The result should be put in *_CurrentTransaction.*

A.4.3.2 SCFunction

Class in package '4.6.3 Smart Contracts'

Details: This represents a Smart Contract function. This construct should be generic enough to be applicable to most common blockchain implementations

CONNECTORS
Dependency Source -> Destination From: : SCFunction : Class , Public To: SCFunction : Class , Public
Dependency Source -> Destination From: : SCFunction : Class , Public To: TypeOfFunction : Enumeration , Public
Dependency Source -> Destination From: : Policy : Class , Public To: SCFunction : Class , Public
Dependency Source -> Destination From: : SCFunction : Class , Public To: SCFunction : Class , Public
Usage «Instantiate» Source -> Destination From: : CapacityTester_RegistrationFN : Object , Public To: SCFunction : Class , Public
Dependency Source -> Destination From: : BaseClass : Class , Public To: SCFunction : Class , Public

ATTRIBUTES
_nameSpace : **int** Private *Details:* This is an internal reference to the namespace on which the function is currently being executed. Named spaces enable developers to gain control over the properties and methods that are visible. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
_PrevFunction : **SCFunction** Private *Details:* This pointer can be used to assemble a virtual *callstack*, the *callstack* can be used to determine the original function invocation/context. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Code : **Bytes** Public *Details:* This field represents the logical code contained within a function. This code may be interpreted, compiled binary or a mixture of them. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
FunctionType : **TypeOfFunction** Private *Details:* This field can be used to classify the function type, this is informative and the actual usage will be dependent on the code logic, provided parameters and execution context. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Name : **String** Private *Details:* This field represents the function name. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Parameters : **SerializableObject** Public *Details:* This field represents the parameters passed to this function. The parameters should be encoded on a manner that it supports the reconstruction of the *callstack* (see *_PrevFunction*). Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

OPERATIONS
GetParameters () : **void** Public *Details:*

A.4.3.3 SmartContract_Fabric

Class in package '4.6.3 Smart Contracts'

Details: This class represents a reference implementation of *SmartContracts* within Hyperledger Fabric. The class inherits the *SmartContractRealization*, thereby realizing the *iSmartContract* interfacee.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from SmartContract_Fabric to SmartContractRealization

ATTRIBUTES
_IOBasics : **Object** Private *Details:*
_SecurityContext : **Object** Private *Details:*
_Stub : **SerializableObject** Public *Details:*

OPERATIONS
PopulateIOBasics () : **void** Public *Details:*
PopulateSecurityContext () : **void** Public *Details:*
PopulateStub () : **void** Public

OPERATIONS
Details:

A.4.3.4 SmartContractDep

Class in package '4.6.3 Smart Contracts'

Details: Objects whom reference this class expect an object that realizes the *SmartContract* Interface. This class is a leaf and is only intended to serve as a data type.

CONNECTORS
Usage Source -> Destination From: : SmartContractDep : Class , Public To: iSmartContract : ProvidedInterface , Public

A.4.3.5 SmartContractRealization

Class in package '4.6.3 Smart Contracts'

Details: This abstract class implements the *SmartContract* interface, classes derived from this class should satisfy all of the service and data requirements.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from SmartContractRealization to «interface» iSmartContract
Realization from SmartContractRealization to iSmartContract

A.4.3.6 TypeOfFunction

Enumeration in package '4.6.3 Smart Contracts'

Details: This list provides a set of examples that can be used to describe a smart contract funtion.

CONNECTORS
Dependency Source -> Destination From: : SCFunction : Class , Public To: TypeOfFunction : Enumeration , Public

ENUMERATION:
DataStorage
LegalContract
LogicAutomation
MessagePassing
Computation

A.5 Operations-Structural components

In this section, a sample set of structural components that may be relevant to a TES five-stage operational model are presented. These structural components are intended to serve as a reference and application developers will need to build their processes based on their needs and templates introduced in the previous sections.

A.5.1 Qualification&Registration

Package in package '4.7 Operations-Structural components'

Qualification diagram

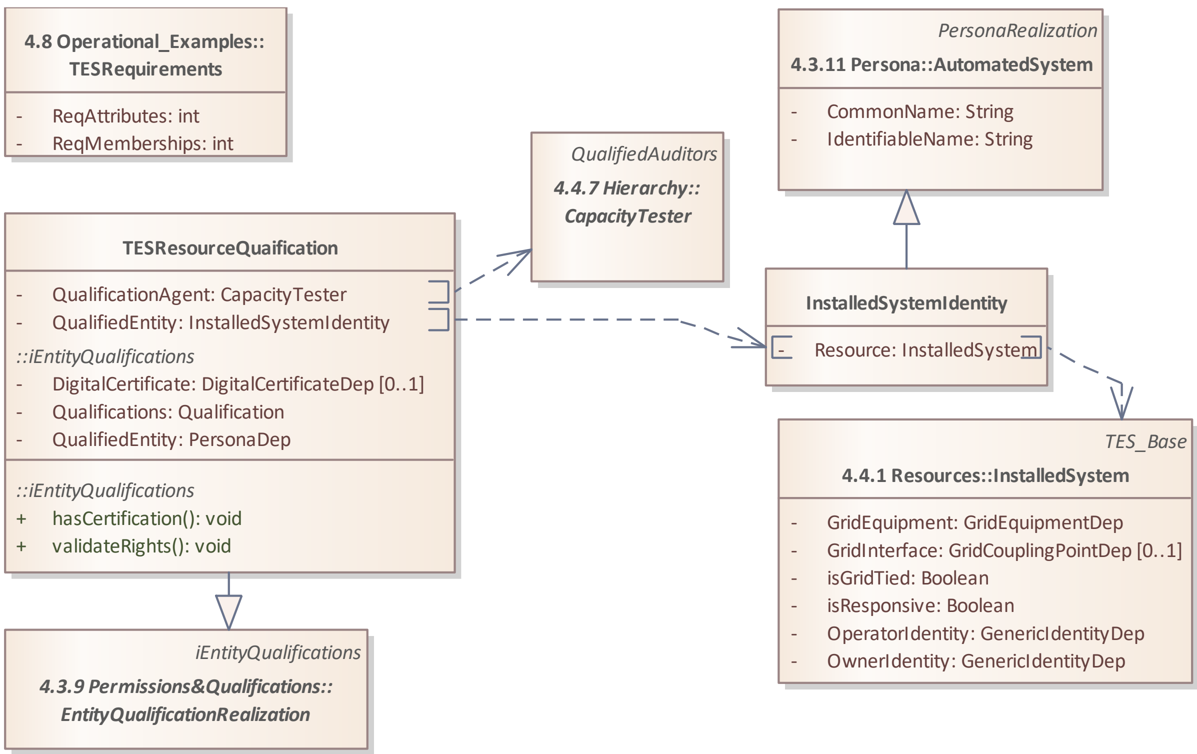


Figure 78. Overview of the Qualification' package components

A.5.1.1 Equipment

Class in package '4.7.1 Qualification&Registration'

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from Equipment to GenericIdentityRealization

CONNECTORS
Dependency Source -> Destination From: : Equipment : Class , Public To: InstalledSystemIdentity : Class , Public

ATTRIBUTES
PersonaDetails : **InstalledSystemIdentity** Public *Details:*

A.5.1.2 InstalledSystemIdentity

Class in package '4.7.1 Qualification&Registration'

Details: This object

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from InstalledSystemIdentity to AutomatedSystem

CONNECTORS
Dependency Source -> Destination From: : InstalledSystemIdentity : Class , Public To: InstalledSystem : Class , Public
Dependency Source -> Destination From: : Equipment : Class , Public To: InstalledSystemIdentity : Class , Public
Dependency Source -> Destination From: : TESResourceQuaification : Class , Public To: InstalledSystemIdentity : Class , Public

ATTRIBUTES
Resource : **InstalledSystem** Private *Details:*

A.5.1.3 TESResourceQuaification

Class in package '4.7.1 Qualification&Registration'

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from TESResourceQuaification to EntityQualificationRealization

CONNECTORS
Dependency Source -> Destination From: : TESResourceQuaification : Class , Public To: InstalledSystemIdentity : Class , Public
Dependency Source -> Destination From: : TESResourceQuaification : Class , Public To: CapacityTester : Class , Public

ATTRIBUTES
QualificationAgent : **CapacityTester** Private *Details:*
QualifiedEntity : **InstalledSystemIdentity** Private *Details:*

A.5.1.4 TESQualification

Enumeration in package '4.7.1 Qualification&Registration'

A.6 Operations-Examples

This section contains examples that may serve as a reference for building more complex systems. This examples only list the main steps and will need to be adapted to suit an application's needs.

A.6.1 Agent qualification

In this demo we assume that a non-qualified actor is interested in becoming a qualified DER installer. To achieve this state the actor must first get a copy of the terms and conditions (requirements), followed by getting all the documentation ready. Finally, the agent submits this documentation (e.g., proof of courses taken) and its case gets evaluated in a transparent, equitable manner by the blockchain-based solution.

A.6.1.1 Qualification Use Case diagram

This diagram presents the data dependencies needed to transition a non-qualifier actor into a qualified actor. For example a company may want to gain qualifications as a DER-system capacity tester.

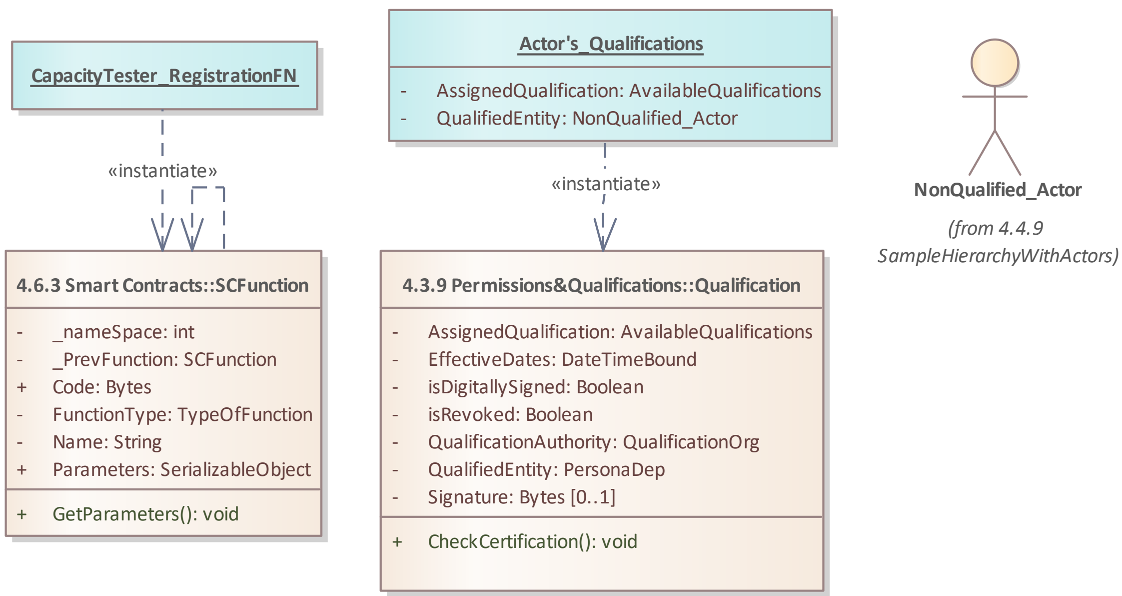


Figure 79. Overview of the QualificationUseCase' package components

A.6.1.2 Actor's_Qualifications

Object in package '4.8.1.A Agent Qualification Demonstration-Structural side'

CONNECTORS
Usage «Instantiate» Source -> Destination From: : Actor's_Qualifications : Object , Public To: Qualification : Class , Public

ATTRIBUTES
AssignedQualification : **AvailableQualifications** Private *Details:*
QualifiedEntity : **NonQualified_Actor** Private *Details:*

A.6.1.3 CapacityTester_RegistrationFN

Object in package '4.8.1.A Agent Qualification Demonstration-Structural side'

Details: This object represents an instance of the *SCFunction* class. The body of the function should enable non-qualified actors to become qualified by providing the correct arguments (such as training requirements). The function should be responsible for updating the actor's qualifications.

CONNECTORS
Usage «Instantiate» Source -> Destination From: : CapacityTester_RegistrationFN : Object , Public To: SCFunction : Class , Public

A.6.1.4 Agent Qualification Demonstration-Structural side

This diagram presents a sequence diagram for transitioning a non-qualifier actor into a qualified actor.
It is assumed that a consortium has already decided on the terms and conditions and the registration process for becoming a capacity tester has been outlined.

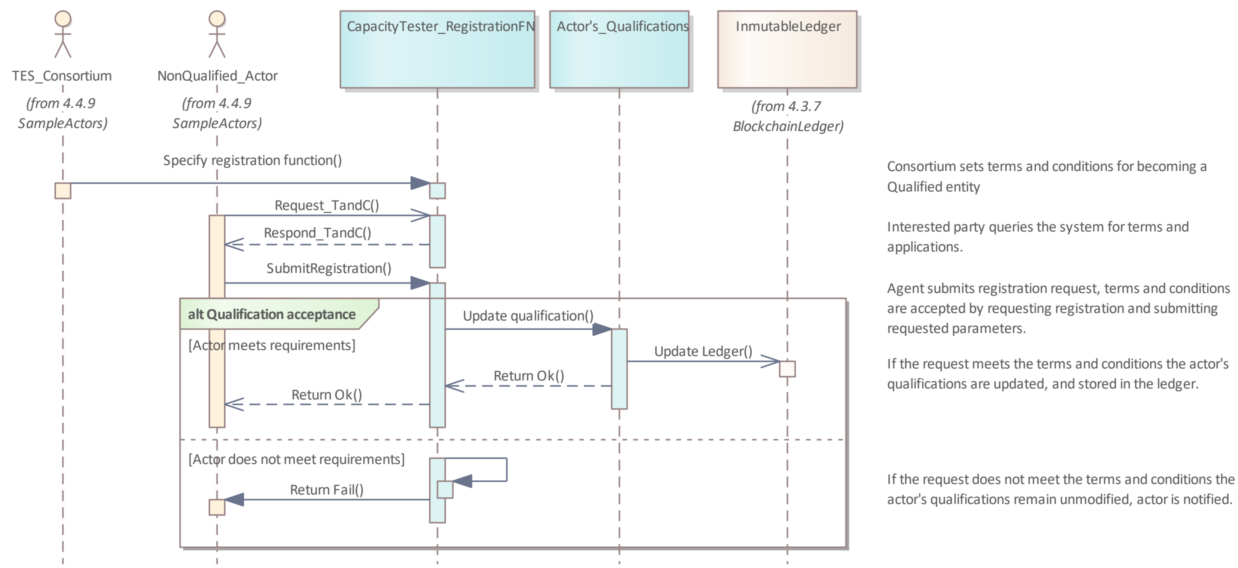


Figure 80. Overview of the Registration&Qualification' package components

INTERACTION MESSAGES
1.0 'Specify registration function' from 'TES_Consortium' sent to 'CapacityTester_RegistrationFN'. Synchronous Call. Returns void.
1.1 'Request_TandC' from 'NonQualified_Actor' sent to 'CapacityTester_RegistrationFN'. Asynchronous Call. Returns void.
1.2 'Respond_TandC' from 'CapacityTester_RegistrationFN' sent to 'NonQualified_Actor'. Asynchronous Call. Returns void.
1.3 'SubmitRegistration' from 'NonQualified_Actor' sent to 'CapacityTester_RegistrationFN'. Synchronous Call. Returns void.
1.4 'Update qualification' from 'CapacityTester_RegistrationFN' sent to 'Actor's_Qualifications'. Synchronous Call. Returns void.

1.5 'Update Ledger' from 'Actor's_Qualifications' sent to 'InmutableLedger'. Asynchronous Call. Returns void.
1.6 'Return Ok' from 'Actor's_Qualifications' sent to 'CapacityTester_RegistrationFN'. Synchronous Call. Returns void.
1.7 'Return Ok' from 'CapacityTester_RegistrationFN' sent to 'NonQualified_Actor'. Synchronous Call. Returns void.
1.8 '' from 'CapacityTester_RegistrationFN' sent to 'CapacityTester_RegistrationFN'. Synchronous Call. Returns void.
1.9 'Return Fail' from 'CapacityTester_RegistrationFN' sent to 'NonQualified_Actor'. Synchronous Call. Returns void.

A.7 Sample: Developing a Smart Contract-Based Permission Solution

This section presents the low-level details of a TES-based Attribute Based Access Control mechanism that leverages the objects/constructs introduced in the previous sections.

A.7.1 Base TES execution model

This model represents an abstract representation of the base-class used to interface any object-oriented class with a Blockchain-based ledger, it contains all the bootstrap functions to streamline the creation, loading, updating of any object.

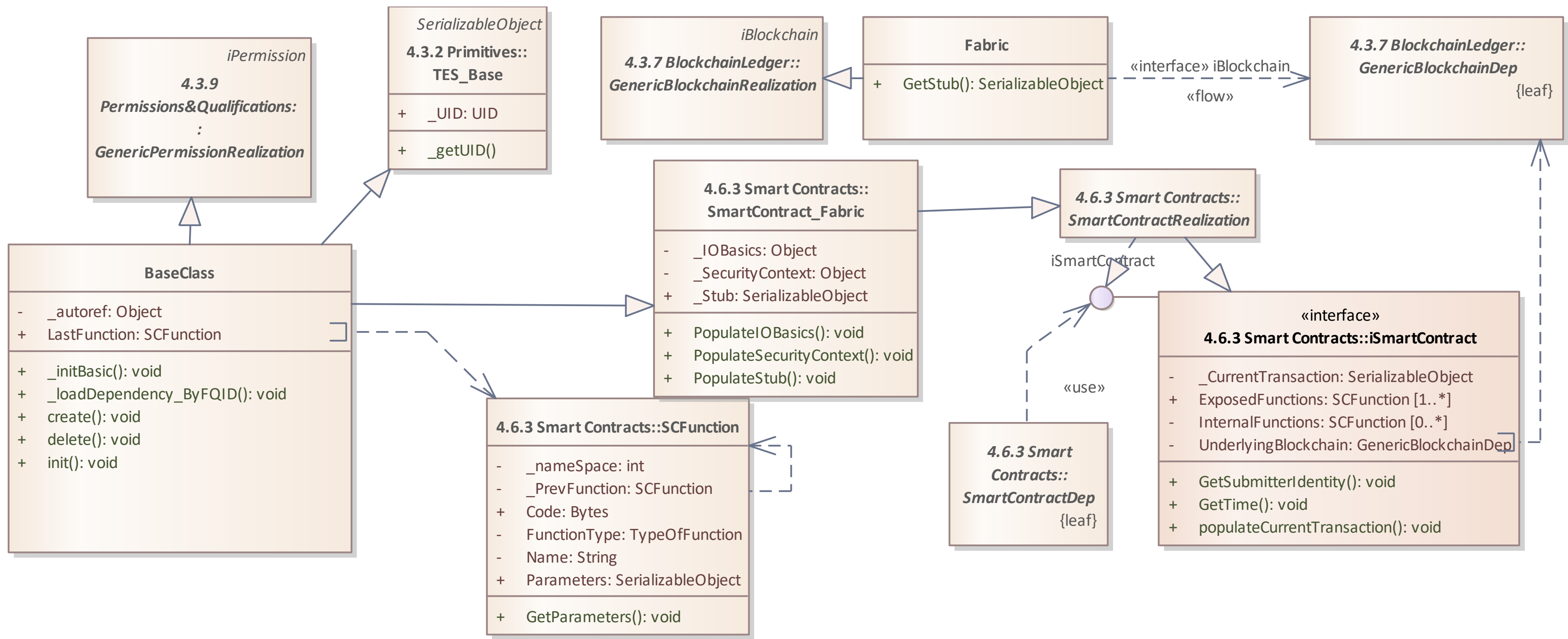


Figure 81. Overview of the BaseTES' package components

A.7.1.1 BaseClass

Class in package '4.9.1 Base TES execution model'

Details: This base class contains all the bootstrap functions to streamline the creation, loading, updating of any object into the ledger. In addition, it contains certain properties that can be used to support an ABAC system, such as providing information about the blockchain transaction.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from BaseClass to TES_Base
Generalization from BaseClass to GenericPermissionRealization
Generalization from BaseClass to SmartContract_Fabric

CONNECTORS
Dependency Source -> Destination From: : BaseClass : Class , Public To: GenericBlockchainDep : Class , Public
Dependency Source -> Destination From: : BaseClass : Class , Public To: SCFunction : Class , Public

ATTRIBUTES
_autoref : **Object** Private *Details:* This field represents a reference to itself. Commonly known as this in programming languages. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
LastFunction : **SCFunction** Public *Details:* This pointer references the last function which loaded or initialized this object. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

OPERATIONS
_initBasic () : **void** Public *Details:* This is a generic function that can be used to bootstrap functions before an object becomes fully initialized. This bootstrap is exploited to implement the ABAC system.
_loadDependency_ByFQID () : **void** Public *Details:* This function loads an object dependency based on a known FQID.
create () : **void** Public *Details:* This is a bootstrap function that can be used to populate attributes that define an objects birth-right attributes, such as owner, time of creation, etc.
delete () : **void** Public *Details:* This is a bootstrap function that can be used to populate attributes or mark an object as inactive. Real data deletion is an unsupported function of DLT-based technologies.
init () : **void** Public *Details:* This is a bootstrap function that can be used to dynamically obtain parameters upon object initialization.

A.7.1.2 Fabric

Class in package '4.9.1 Base TES execution model'

STRUCTURAL PART OF Fabric
ProvidedInterface1 : ProvidedInterface

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from Fabric to GenericBlockchainRealization

OPERATIONS
GetStub () : **SerializableObject** Public *Details:* This is a Hyperledger Fabric-specific function that provides additional details about the execution context. It extends the realization provided by *iBlockchain* interface

A.7.2 Attribute Base Access control (ABAC) implementation

This model represents the structural requirements for implementing an Attribute Based Access Control mechanism that can be used to implement and enforce access controls over a resource. The proposed use case overrides the role-based permission mechanism introduced by the *GenericPermission* interface. Demonstrating once more the templates ability to adapt to the needs of an application.

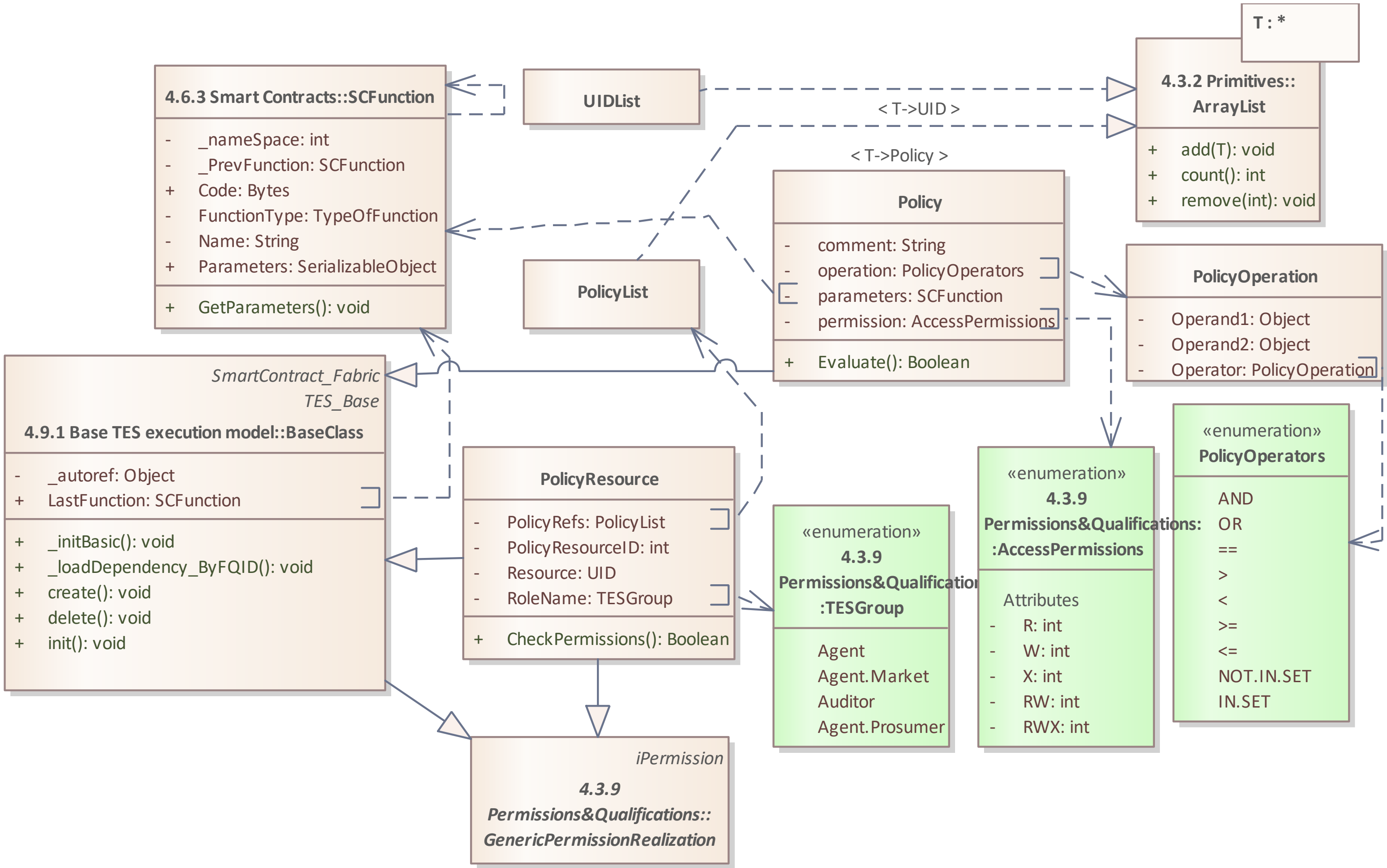


Figure 82. Overview of the ABAC' package components

A.7.2.1 Policy

Class in package '4.9.2 Attribute Base Access control (ABAC) implementation'

Details: This class represents a single policy. Its run-time evaluation determines the actual access permissions. These permissions are dynamic and represent the core functionality of an ABAC system

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from Policy to BaseClass

CONNECTORS
Dependency Source -> Destination From: : Policy : Class , Public To: AccessPermissions : Enumeration , Public
Dependency Source -> Destination From: : Policy : Class , Public To: SCFunction : Class , Public
Dependency Source -> Destination From: : Policy : Class , Public To: PolicyOperation : Class , Public

ATTRIBUTES
comment : **String** Private *Details:*
operation : **PolicyOperators** Private *Details:*
parameters : **SCFunction** Private *Details:*
permission : **AccessPermissions** Private *Details:*

OPERATIONS
Evaluate () : **Boolean** Public *Details:* This evaluation runs under the assumption of an AND operator. This means that all policy operators must return True.

A.7.2.2 PolicyList

Class in package '4.9.2 Attribute Base Access control (ABAC) implementation'

Details: This class represents a list of policies.

OUTGOING STRUCTURAL RELATIONSHIPS
Realization from PolicyList to ArrayList

CONNECTORS
Dependency Source -> Destination From: : PolicyResource : Class , Public To: PolicyList : Class , Public

A.7.2.3 PolicyOperation

Class in package '4.9.2 Attribute Base Access control (ABAC) implementation'

Details: This object represents a comparison in between two objects. Each object can contain nested PolicyOperations to create complex rule sets.

CONNECTORS
Dependency Source -> Destination From: : PolicyOperation : Class , Public To: PolicyOperators : Enumeration , Public
Dependency Source -> Destination From: : Policy : Class , Public To: PolicyOperation : Class , Public

ATTRIBUTES
Operand1 : **Object** Private *Details:* This operand can be a static value, another *policyOperation* or an object which can be dynamically evaluated. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Operand2 : **Object** Private *Details:* This operand can be a static value, another *policyOperation* or an object which can be dynamically evaluated. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Operator : **PolicyOperation** Private *Details:* This field represents an operator that will be applied between both operands. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

A.7.2.4 PolicyResource

Class in package '4.9.2 Attribute Base Access control (ABAC) implementation'

Details: This object represents an intermediary agent that can relate both a set of policies and a resource.

OUTGOING STRUCTURAL RELATIONSHIPS
Generalization from PolicyResource to GenericPermissionRealization
Generalization from PolicyResource to BaseClass

CONNECTORS
Dependency Source -> Destination From: : PolicyResource : Class , Public To: TESGroup : Enumeration , Public
Dependency Source -> Destination From: : PolicyResource : Class , Public To: PolicyList : Class , Public

ATTRIBUTES
PolicyRefs : **PolicyList** Private *Details:*
PolicyResourceID : **int** Private *Details:* This field can be used as an identifier that can be used to track a decision. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
Resource : **UID** Private *Details:* This represents a resource for which access is being restricted. Individual components of the UID can be left blank to cover all objects, all versions or all instances with the same ID. Multiplicity: (1, Allow duplicates: 0, Is ordered: False)
RoleName : **TESGroup** Private *Details:* This field can be used to determine the current role/context of the agent that is requesting access.

ATTRIBUTES
Multiplicity: (1, Allow duplicates: 0, Is ordered: False)

OPERATIONS
CheckPermissions () : **Boolean** Public *Details:* This function evaluates all policies using an OR scheme (e.g. any policy that grants access will be followed).

A.7.2.5 UIDList

Class in package '4.9.2 Attribute Base Access control (ABAC) implementation'

Details: This represents a list of UID numbers, It can be used to reference multiple objects at once.

OUTGOING STRUCTURAL RELATIONSHIPS
Realization from UIDList to ArrayList

A.7.2.6 PolicyOperators

Enumeration in package '4.9.2 Attribute Base Access control (ABAC) implementation'

Details: This enumeration provides samples of policy operators that can be evaluated by the ABAC platform

CONNECTORS
Dependency Source -> Destination From: : PolicyOperation : Class , Public To: PolicyOperators : Enumeration , Public

ENUMERATION:
AND
OR
==
>
<
>=
<=
NOT.IN.SET
IN.SET

Pacific Northwest National Laboratory

902 Battelle Boulevard
P.O. Box 999
Richland, WA 99354
1-888-375-PNNL (7665)

www.pnnl.gov