



OCPP 2.0.1

Part 1 - Architecture & Topology

Edition 4, 2025-12-03

Table of Contents

Disclaimer	1
Version History	2
1. Introduction	3
1.1. Goal of this document	3
1.2. Terms and abbreviations	3
2. 3-tier model	4
3. Information Model	5
4. Device Model: Addressing Components and Variables	6
4.1. Components	6
4.2. Variables	7
4.3. Characteristics and Attributes	7
4.4. Monitoring	9
4.5. Standardized lists of Components and Variables	10
4.6. Minimum Device Model	10
5. Information Model vs. Device Model	12
6. Using OCPP for other purposes than EV charging	13
7. Numbering	14
7.1. EVSE numbering	14
7.2. Connector numbering	14
7.3. Transaction IDs	14
8. Topologies supported by OCPP	15
8.1. Charging Station(s) directly connected to CSMS	15
8.2. Multiple Charging Stations connected to CSMS via Local Proxy	15
8.3. Multiple Charging Stations connected to CSMS via Local Controller	15
8.4. Non-OCPP Charging Stations connected to CSMS via OCPP Local Controller	16
8.5. DSO control signals to CSMS	16
8.6. Parallel control by CSMS and EMS	16
9. Part 1 Appendix: OCPP Information Model	18
9.1. Explanation of UML representation and message generation	18
9.2. Visual Representation of OCPP Information Model	19

Disclaimer

Copyright © 2010 – 2025 Open Charge Alliance. All rights reserved.

This document is made available under the **Creative Commons Attribution-NoDerivatives 4.0 International Public License** (<https://creativecommons.org/licenses/by-nd/4.0/legalcode>).

Version History

Version	Date	Description
2.0.1 Edition 4	2025-12-03	OCPP 2.0.1 Edition 4. All errata from OCPP 2.0.1 Part 1 until and including Errata 2025-11 have been merged into this version of the specification.
2.0.1 Edition 3	2024-05-06	OCPP 2.0.1 Edition 3. All errata from OCPP 2.0.1 Part 1 until and including Errata 2024-04 have been merged into this version of the specification.
2.0.1	2020-03-31	Final version of OCPP 2.0.1
2.0	2018-04-11	OCPP 2.0 April 2018 First release of this Architecture & Topology document

Chapter 1. Introduction

1.1. Goal of this document

The goal of this document is to describe a number of architecture related topics for OCPP 2.0.1 .

OCPP was originally intended for two way communication between a backoffice, in OCPP the *Charging Station Management System* (in this document: CSMS) and a Charging Station. The protocol has become more advanced and with every new revision new functionalities and options are added. It has evolved into a protocol that can be used in different architectures for different types of Charging Stations.

This document describes, in addition to the original "simple" setup CSMS <> Charging Station, a number of topologies as an additional explanation for using OCPP. Furthermore, the Device Management concept to configure and monitor any type of Charging Station, the OCPP Information Model and the 3-tier model are explained.

This document is partially **informative** and partially **normative** and is not intended to limit the use of OCPP. However, it does add an explanation what kind of use of OCPP the creators of OCPP had in mind when creating this version of the specification. This document is therefore also intended to support the reader of the protocol specification in Part 2 of OCPP to understand how it can be used.

1.2. Terms and abbreviations

This section contains the terminology and abbreviations that are used throughout this document.

1.2.1. Terms

Term	Meaning
Charging Station	The Charging Station is the physical system where EVs can be charged. A Charging Station has one or more EVSEs.
Connector	The term Connector, as used in this specification, refers to an independently operated and managed electrical outlet on a Charging Station. In other words, this corresponds to a single physical Connector. In some cases an EVSE may have multiple physical socket types and/or tethered cable/Connector arrangements(i.e. Connectors) to facilitate different vehicle types (e.g. four-wheeled EVs and electric scooters).
EVSE	An EVSE is considered as an independently operated and managed part of the Charging Station that can deliver energy to one EV at a time.
Local port Smart Meter	The local port on a Smart Meter is a port (for example serial) on a digital electricity meter that provides access to information about meter readings and usage.

1.2.2. Abbreviations

Abbreviation	Meaning
DSO	Distribution System Operator
CSO	Charging Station Operator
CSMS	Charging Station Management System
EMS	Energy Management System. In this document this is defined as a device that manages the local loads (consumption and production) based on local and/or contractual constraints and/or contractual incentives. It has additional inputs, such as sensors and controls from e.g. PV, battery storage.
EVSE	Electric Vehicle Supply Equipment
LC	Local Controller. In this document this is defined as a device that can send messages to its Charging Stations, independently of the CSMS. A typical usage for this is the local smart charging case described in the Smart Charging chapter of Part 2 of OCPP, where a Local Controller can impose charge limits on its Charging Stations.
LP	Local Proxy. Acts as a message router.

Chapter 2. 3-tier model

This section is informative.

To understand the terminology in the OCPP specification, it is important to understand the starting point of this specification. The OCPP specification uses the term Charging Station as the physical system where EVs can be charged. A Charging Station can have one or more EVSEs (Electric Vehicle Supply Equipment). An EVSE is considered as a part of the Charging Station that can deliver energy to one EV at a time. The term Connector, as used in this specification, refers to an independently operated and managed electrical outlet on a Charging Station, in other words, this corresponds to a single physical Connector. In some cases an EVSE may have multiple physical socket types and/or tethered cable/connector arrangements to facilitate different vehicle types (e.g. four-wheeled EVs and electric scooters). This setup is referred to as the 3-tier model and visualized in the figure below.

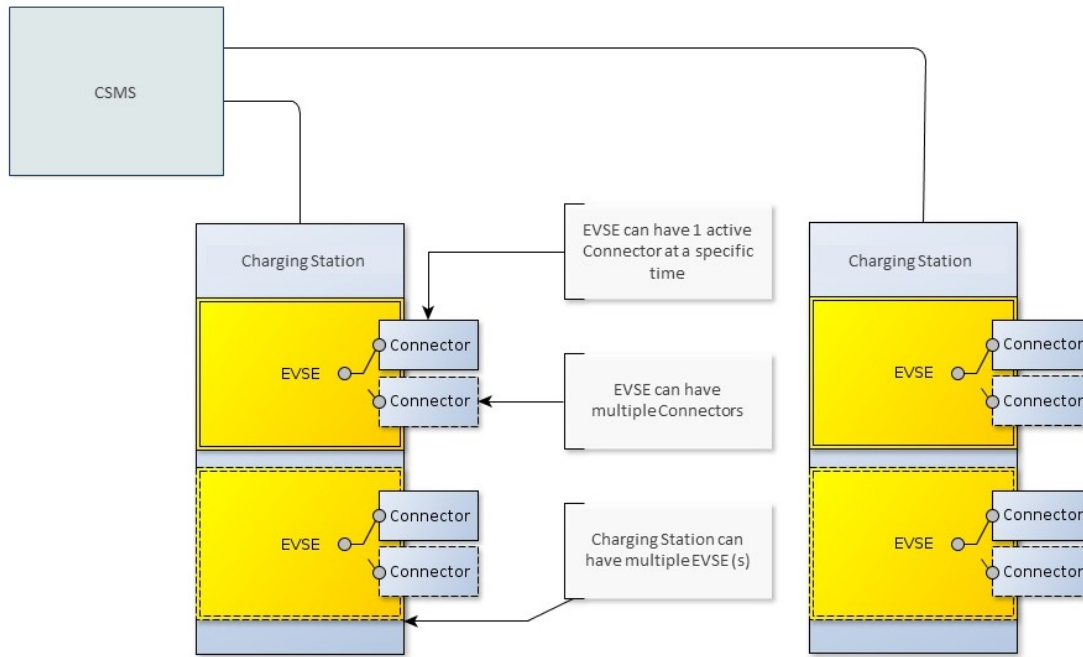


Figure 1. 3-tier model as used in OCPP

NOTE

This section describes the charging infrastructure on a logical level for communication purposes. We do not wish to impose a mapping onto physical hardware. This is a manufacturer's choice. For example, the EVSE might be integrated into a Charging Station and to look as just a part of that device, but it might just as well have its own casing and live outside of the physical entity Charging Station, for example a charging plaza with 20 EVSEs and Connectors which communicates via 1 modem as 1 Charging Station to the CSMS is seen by OCPP as 1 Charging Station.

Chapter 3. Information Model

This section is informative.

Given the growing complexity of the messages of OCPP, OCPP 2.0.1 is based on an *Information Model* as a blueprint for the messages and inherent schemas of OCPP. With an information model, we mean a logical object set, describing real objects with all their properties. This provides an informative representation of information structure in the protocol. Furthermore, it enables making objects within OCPP reusable and enables consistent definition of messages and automatically generated message schemas (Part 3).

The Information Model is a model, also called Domain Model or Core Model, based on which the OCPP messages and datatypes are generated. These datatypes are extracted from the the OCPP 1.6 specification and are named Core DataTypes and Qualified DataTypes. The figure below illustrates how the DataTypes in the information model are built up.

In part 2 - Specification, chapter Datatypes, some DataTypes have the Common: prefix. This originates from the Information Model. It means that the DataType is able to be shared among other DataTypes and Messages. This has no impact on the OCPP implementation of a device.

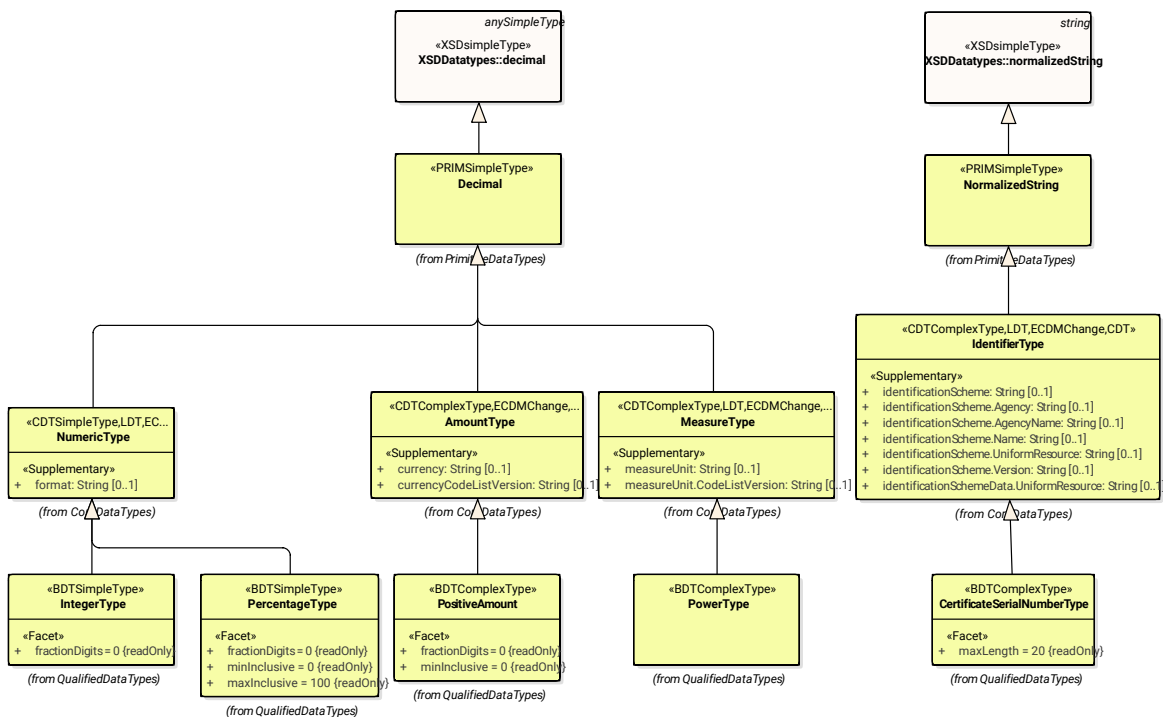


Figure 2. Example datatypes

The Information Model is divided into a number of "functions" to have a better overview of the model (thus for readability):

- Transactions
- SmartCharging
- Metering
- Security (Profiles/Authorization)
- Communication
- SecondaryActorSchedule

For more details about the actual model per function, please refer to the appendix.

Chapter 4. Device Model: Addressing Components and Variables

The Device Model refers to a generalized mechanism within OCPP to enable any model of Charging Station to report how it is build up, so it can be managed from any CSMS. To manage a Charging Station with the Device Model (i.e. "to manage a device") a number of messages and use cases is defined to configure and monitor a Charging Station in detail, without defining the structure of the Charging Station in advance. To be able to do this, OCPP provides a generalized mechanism to allow the exchange of a wide range of information about Charging Station. This version of the Device Model has the 3-tier model (Charging Station, EVSE, Connector) as its starting point, which means that any description created with the Device Model follows these three tiers. The remainder of this chapter describes how the data (and associated meta-data) looks like that can be exchanged between a Charging Station and a CSMS. The use cases and messages that are used to manage a device are *not* described here, but in Part 2 of the specification. This chapter only focuses on the data model.

4.1. Components

In OCPP 2.0.1 , a Charging Station is modelled as a set of "*Components*", typically representing physical devices (including any external equipment to which it is connected for data gathering and/or control), logical functionality, or logical data entities.

Components of different types are primarily identified by a *ComponentName*, that is either the name of a *standardized* component (see OCPP part 2c), or a custom/non-standardized component name, for new, pre-standardized equipment, vendor specific extensions, etc.

ChargingStation (TopLevel), *EVSE*, and *Connector* represent the three major "tiers" of a Charging Station, and constitute an implicit "location-based" addressing scheme that is widely used in many OCPP data structures. Each "tier" has a component of the same name, which represents the tier.

For example, EVSE 1 on a Charging Station is represented by the component named "EVSE" (no instance name) with "evseId = 1". In the same manner, Connector 1 on EVSE 1 is represented by the component named "Connector" (no instance name) with "evseId = 1, connectorId = 1".

By default, all *components* are located at the *ChargingStation* tier, but individual instances of any component can be associated with a specific *EVSE*, or a specific *Connector* (on a specific EVSE) by including EVSE or EVSE and Connector identification numbers as part of a component addressing reference.

Additionally, there can be more than one instance of a component (in the functional dimension), representing multi-occurrence physical or logical components (e.g. power converter modules, fan banks, resident firmware images, etc.).

Each distinct *component* instance is uniquely identified by an (optional) *componentInstance* addressing key. It is allowed for a *component* to exist without an instance and at the same time also exist with one of more instances. When no *componentInstance* is provided, then the *component* without an instance is referenced.

Components do not in themselves hold data: all externally accessible data associated with each component instance is represented by a set of *variables* that can be read, set, and/or monitored for changes. The relationship of a Component with one or more Variables is illustrated in below.

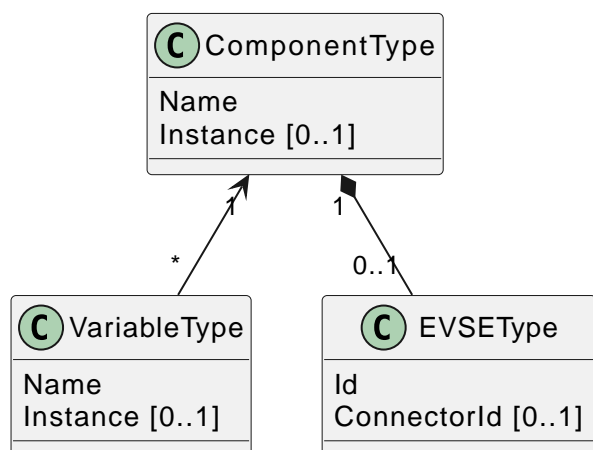


Figure 3. Component and variables

The table below illustrates some common components (by their standardized component-names), and examples of the hierarchical location levels at which they typically occur for a basic home charger and a typical public Charging Station.

Basic home charger example configuration		
ChargingStation tier	EVSE tier	Connector tier
ChargingStation (itself, as a whole)	EVSE (itself, as a whole)	Connector (itself, as a whole)

Basic home charger example configuration		
RadioLink	ControlMetering	PlugRetentionLock
TokenReader	OverCurrentBreaker	
Controller	RCD	
	ChargingStatusIndicator	

Public Charging Station example configuration		
ChargingStation tier	EVSE tier	Connector tier
ChargingStation (itself, as a whole)	EVSE (itself, as a whole)	Connector (itself, as a whole)
ElectricalFeed	ElectricalFeed	AccessProtection
TokenReader	TokenReader	PlugRetentionLock
Display	Display	
FiscalMetering	FiscalMetering	
Clock	ControlMetering	
Controller	OverCurrentBreaker	
	RCD	
	ChargingStatusIndicator	

4.2. Variables

Every *component* has a number of *variables*, that can, as appropriate, be used to hold, set, read, and/or report on all (externally visible) data applicable to that *component*, including configuration parameters, measured values (e.g. a current or a temperature) and/or monitored changes to variable values.

Although many *components* can have associated *variables* that are, by their nature, specific to the component type (e.g. *ConnectorType* for a *Connector* component), there is a minimal set of standardized *variables* that is used to provide standardized high level event notification and state/status reporting (e.g. *Problem*, *Active*) on a global and/or selective basis, and also to report component presence, availability, etc. during the inventorying/discovery process (e.g. *Available*, *Enabled*).

A Charging Station is not required to report the base variables: *Present*, *Available* and *Enabled* when they are readonly and set to *true*. When a Charging Station does not report: *Present*, *Available* and/or *Enabled* the Central System SHALL assume them to be readonly and set to *true*

Variables can be any of a range of common general-purpose data types (boolean, integer, decimal, date-time, string), but also can have their allowable values constrained to particular ranges, enumeration lists, sets, or ordered lists.

To support complex components, there can be more than one instance of any given variable name associated with any component (e.g. power converter modules reporting temperature, current, or voltage at multiple points).

Each distinct *variable* instance is uniquely identified by an (optional) *variableInstance* addressing key string value. It is allowed for a *variable* to exist without an instance and at same time also with one or more instances. When no *variableInstance* is provided, then the *variable* without an instance is referenced.

4.3. Characteristics and Attributes

Each *variable*, in addition to its primary ("Actual") value, can have a set of associated secondary data that is linked to the same primary *variable* name and *variableInstance*.

This greatly avoids cluttering the *variables* namespace with confusing clusters of ancillary variable names (e.g. FanSpeed, FanSpeedUnits, MinimumFanSpeed, BaseFanSpeed) that lack consistence and discoverability. The ancillary variable data includes:

- Variable characteristics meta-data (read-only)
 - Unit of measure (V,W,kW,kWh, etc.)
 - Data type (Integer, Decimal, String, Date, OptionList, etc.)
 - Lower limit
 - Upper limit
 - List of allowed values for enumerated variables

- Variable attributes (read-write):
 - Actual value
 - Target value
 - Configured lower limit
 - Configured upper limit
 - Mutability (whether the value can be altered or not, e.g. ReadOnly or ReadWrite)
 - Persistence (whether the value is preserved in case of a reboot or power loss)

The relationship of a Variable with one or more VariableAttributes is illustrated in the figure below.

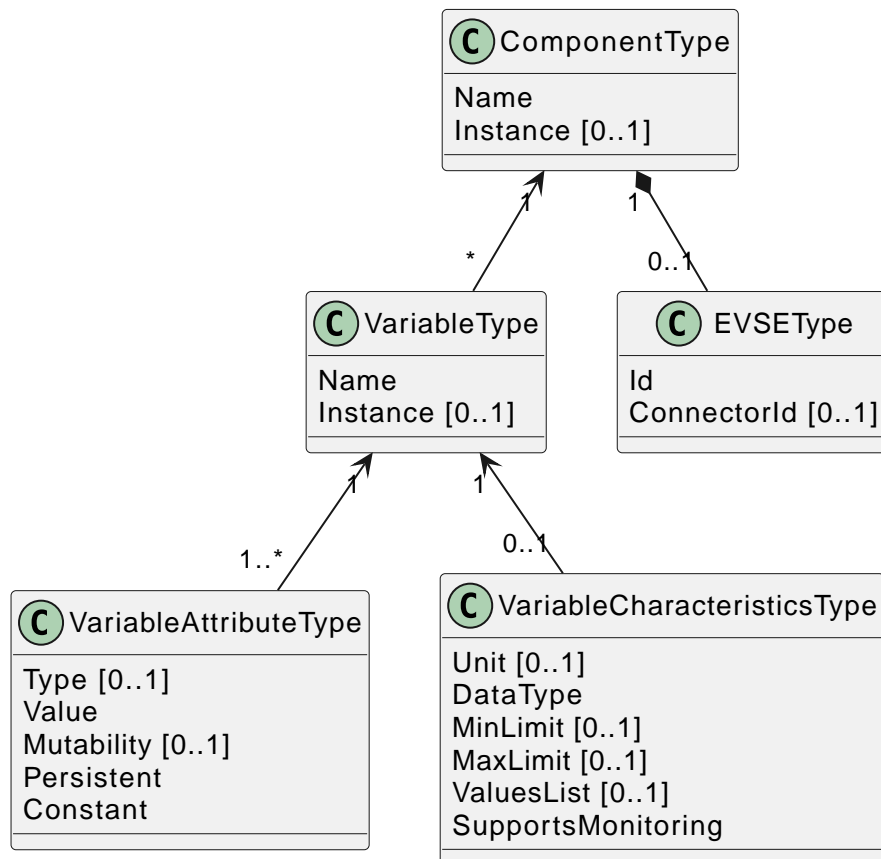


Figure 4. Variable attributes and characteristics

There is a difference between how to implement (physical) devices and (virtual) controller components, using the DeviceModel. A (virtual) controller component has to be implemented as described in part 2 chapter the "Referenced Components and Variables". These kind of components/variables are only using the variableAttribute type 'Actual'. Depending on if this variableAttribute is writable, the CSMS can use this to set a new value.

(Physical) devices are a bit more complex to implement. For example, there is a fan with a fan speed, that has a (physical) limit with a range of 0 - 1000. But it should not be allowed to set the value below 200, because the fan can stop functioning. And it should not be set above 500, because that would be bad for the fan on the long run. When implementing this device using the DeviceModel, it can be defined as follows:

Component	name	Fan	
------------------	-------------	-----	--

Variable	name	FanSpeed	
	variableAttribute 1	type	Actual
		value	<The current fan speed value of the fan.>
		mutability	ReadOnly
	variableAttribute 2	type	Target
		value	<The CSMS can use this value to adjust the fan speed. The Charging Station SHALL try to keep the actual value at the target value.>
		mutability	ReadWrite
	variableAttribute 3	type	MaxSet
		value	<The value '500' from the example. The target may not be set above this value.>
	variableAttribute 4	type	MinSet
		value	<The value '200' from the example. The target may not be set below this value.>
	variableCharacteristics	maxLimit	<The value '1000' from the example. This could be the physical max limit of the fan.>
		minLimit	<The value '0' from the example. This could be the physical min limit of the fan. This could also be -1000, if the fan is also able to rotate in the other direction.>
Description	This is an example of how a fan could be defined using the DeviceModel.		

When trying to set the target with value 600, the Charging Station will first check the allowed min and max values/limits and reject the set. If the target value is set to 500, the value is within range and the Charging Station will allow the set and start to adjust the actual fan speed. If the actual fan speed is measured to be 502, it's out of range. But it should be reported to the CSMS, so the actual value of a physical component should be updated without checking the min and max values/limits.

4.4. Monitoring

Optional monitoring settings can be associated with a variable, that allow changes to *variable (Actual)* values are to be reported to the CSMS as event notifications.

These include:

- Monitoring value
- Monitoring type: upper threshold, lower threshold, delta, periodic
- Severity level when reporting the event

The following table show which MonitorType/dataType combinations are possible.

	string	decimal	integer	dateTime	boolean	OptionList	SequenceList	MemberList
UpperThresh old		X	X					
LowerThresh old		X	X					
Delta	X	X	X	X	X	X	X	X
Periodic	X	X	X		X	X	X	X
PeriodicClockAligned	X	X	X		X	X	X	X

- For *UpperThreshold* and *LowerThreshold* the value represents the to be exceeded value by the actual value of the variable.
- For *Delta* this value represents the change in value compared to the actual value from the moment the monitor was set.
 - When the dataType of the variable is integer or decimal, this value represents the absolute difference to be reached to trigger the monitor.
 - When the dataType of the variable is dateTime the unit of measure will be in seconds.
 - When the dataType of the variable is string, boolean, OptionList, SequenceList or MemberList, this value is ignored. The monitor will be triggered by every change in the actual value.
- When a delta monitor is triggered OR when the Charging Station has rebooted, the Charging Station shall set a new momentary value.

- For *Periodic* and *PeriodicClockAligned* the value represents the interval in seconds.

The relationship between a *Variable* and one or more *VariableMonitoring* elements is illustrated in the figure below.

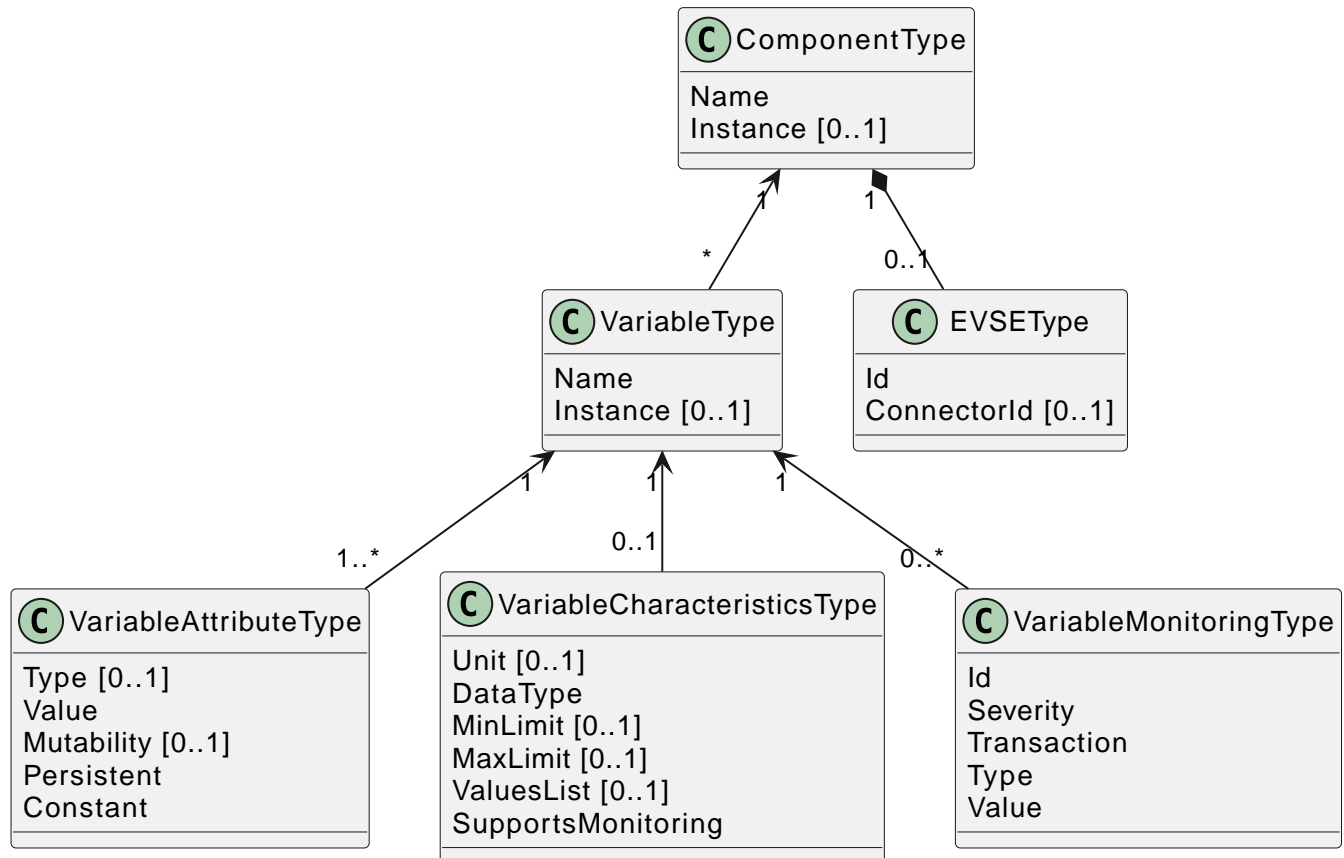


Figure 5. Variables and monitoring

4.5. Standardized lists of Components and Variables

To provide some level of interoperability between different Charging Stations and CSMSs, besides the above defined model of *Components* and *Variables*, part 2 - appendices of the OCPP specification provides a list of standardized names for Components and Variables. The idea of this lists is to make sure that *if* a Charging Station and CSMS want to exchange information about a component, they both use the same name and description *if* it is listed in the OCPP specification. For names of a *Components* or *Variables* that are not listed in the specification, bilateral appointments between Charging Station manufacturer and CSMS are to be made. In these cases it is advised to provide feedback to the Open Charge Alliance to be able to include new/additional *Components* and *Variables* in new versions of OCPP.

4.6. Minimum Device Model

Since the Device Model is a *generalized* mechanism which can be applied to any model of Charging Station, the complexity of different implementations can vary. It consists of a number of use cases and messages that are not all required. This section describes the minimum part of the Device Model that needs to be implemented to create a working implementation of OCPP 2.0.1.

The Device Model introduces Components and Variables that can be used for configuring and monitoring a Charging Station. A number of these Components and Variables are included in the list of *Referenced Components and Variables* (grouped by Functional Block) in Part 2 of the specification. When implementing a Functional Block, ALL required Configuration Variables that belong to a Functional Block SHALL be implemented. The required Configuration Variables from the *General* section SHALL also be implemented for all implementations of OCPP 2.0.1.

The following table describes which messages are required to implement for use cases that are part of the Device Model implementation.

Use cases / messages that are part of a minimum Device Model implementation	
Use case	Messages
B05 Set Variables	SetVariables message MUST be implemented
B06 Get Variables	GetVariables message MUST be implemented.

Use cases / messages that are part of a minimum Device Model implementation	
<i>B07 Get Base Report</i>	GetBaseReport message MUST be implemented and MUST support ConfigurationInventory and FullInventory. The content of these reports depends on the implementation of the Charging Station. It is up to the implementer to decide which components and variables exist in the implementation.
Additional use cases / messages that are <i>not</i> part of a minimum Device Model implementation	
Use case	Messages
<i>B08 Get Custom Report</i>	GetCustomReport message is optional.
<i>N02 Get Monitoring Report</i>	GetMonitoringReportRequest message is optional.
<i>N03 Set Monitoring Base</i>	SetMonitoringBaseRequest message is optional.
<i>N04 Set Variable Monitoring</i>	SetVariableMonitoringRequest message is optional.
<i>N05 Set Monitoring Level</i>	SetMonitoringLevelRequest message is optional.
<i>N06 Clear/Remove Monitoring</i>	ClearVariableMonitoringRequest message is optional.
<i>N07 Alert Event</i>	it is RECOMMENDED that NotifyEventRequest is implemented in the Charging Station even when monitoring is not implemented, so that this can be used to report built-in monitoring events.
<i>N08 Periodic Event</i>	see N07.

Chapter 5. Information Model vs. Device Model

As described above, the terms Information Model and Device Model refer to different concepts. The Information Model refers to a model of the information structure upon which the messages and datatypes in OCPP are based, whereas the Device Model refers to a generalized mechanism within OCPP to enable any model of Charging Station to report how it is build up so, it can be managed from any CSMS without defining the structure of the Charging Station in advance.

Chapter 6. Using OCPP for other purposes than EV charging

As indicated in the introduction of this document, OCPP is primarily intended for two way communication between a CSMS and a Charging Station. However, with the addition of the Device Model as described in the chapter [Device Model](#), OCPP can additionally be used for other purposes. For example, the reporting of Events or Status changes in transformers or stand-alone battery packs might also be useful for companies that are rolling out EV charging infrastructure. In this example, a BootNotification could be used to connect these devices to a management system. In the device model a device that is not a Charging Station, can be recognized by the fact that the component Charging Station is not present at the top level. At the moment the OCPP specification does not provide use cases for non Charging Station devices. However, they may be added in a future version of OCPP.

Chapter 7. Numbering

This section is normative.

7.1. EVSE numbering

To enable the CSMS to address all the EVSEs of a Charging Station, EVSEs MUST always be numbered in the same way.

EVSEs numbering (evselds) MUST be as follows:

- The EVSEs MUST be sequentially numbered, starting from 1 at every Charging Station (no numbers may be skipped).
- evselds MUST never be higher than the total number of EVSEs of a Charging Station
- For operations initiated by the CSMS, evseld 0 is reserved for addressing the entire Charging Station.
- For operations initiated by the Charging Station (when reporting), evseld 0 is reserved for the Charging Station main controller.

Example: A Charging Station with 3 EVSEs: All EVSEs MUST be numbered with the IDs: 1, 2 and 3. It is advisable to number the EVSEs of a Charging Station in a logical way: from left to right, top to bottom incrementing.

7.2. Connector numbering

To enable the CSMS to address all the Connectors of a Charging Station, Connectors MUST always be numbered in the same way.

Connector numbering (connectorIds) MUST be as follows:

- The connectors are numbered (increasing) starting at connectorId 1 on every EVSE
- Every connector per EVSE has a unique number
- ID of the first Connector of an EVSE MUST be 1
- Additional Connectors of the same EVSE MUST be sequentially numbered (no numbers may be skipped)
- connectorIds MUST never be higher than the total number of connectors on that EVSE

Example: A Charging Station with 3 EVSEs that each have 2 connectors, is numbered as follows:

- EVSE 1 has connectors with connectorId 1 and 2
- EVSE 2 has connectors with connectorId 1 and 2
- EVSE 3 has connectors with connectorId 1 and 2

7.3. Transaction IDs

TransactionIds are now generated by the Charging Station and MUST be unique on this Charging Station for every started transaction.

In OCPP 1.x this was done by the CSMS.

The format of the transaction ID is left to implementation. This MAY for example be an incremental number or an UUID.

Chapter 8. Topologies supported by OCPP

This chapter shows a number of topologies for using OCPP. As indicated in the introduction, OCPP was originally used for a setup where each Charging Station communicates directly with the CSMS. It is important to keep in mind that OCPP has no knowledge of the topology of the Charging Station network. The following figure shows the possible components in a setup using OCPP and the relations between these components:

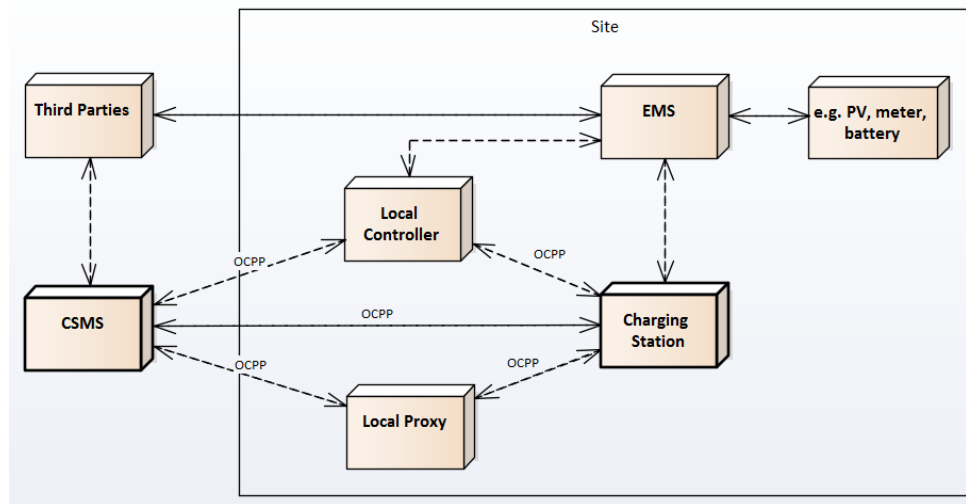


Figure 6. Possible components in a setup using OCPP

8.1. Charging Station(s) directly connected to CSMS

Description

This is the basic setup for using OCPP.

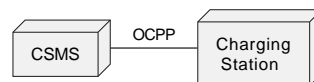


Figure 7. Charging Station directly connected to CSMS

8.2. Multiple Charging Stations connected to CSMS via Local Proxy

Description

In some situations it is desirable to route all communications for a group of Charging Stations through a single network node (i.e. modem, router, etc.). A typical example is the situation where a number of a Charging Stations are located in an underground parking garage with little or no access to the mobile network. In order to provide access to mobile data the Charging Stations are linked to a central data communications unit over a LAN. This central unit connects to the mobile network and acts as a proxy between CSMS and Charging Stations. Such a unit is called a "local proxy" (LP) in OCPP. A local proxy acts as a message router. Neither the CSMS nor the Charging Stations are aware of the topology of the network. For the Charging Stations in the group the local proxy "is" the CSMS. Similarly, for the CSMS the local proxy "is" the Charging Station. The diagram below illustrates this configuration.

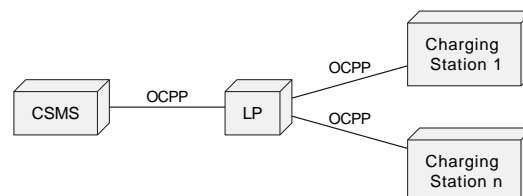


Figure 8. Multiple Charging Stations connected to CSMS via Local Proxy

8.3. Multiple Charging Stations connected to CSMS via Local Controller

Description

Whereas a [local proxy](#) does little more than route OCPP messages, a Local Controller can send messages to its Charging Stations,

independently of the CSMS. A typical usage for this is the local smart charging case described in the Smart Charging chapter of Part 2 of OCPP, where a Local Controller can impose charge limits on its Charging Stations. In order for a Local Controller to be addressed by the CSMS, it needs to have its own Charging Station identity. From the point of view from OCPP, the Local Controller will just be a Charging Station (without any EVSEs/Connectors). The CSMS will possess the logic to deal with the Local Controller in order to support, for example, local smart charging. It is up to the implementation of the CSMS, whether the group topology is manually configured or deduced from the network based on IP addresses and information in BootNotifications. The diagram below illustrates this configuration.

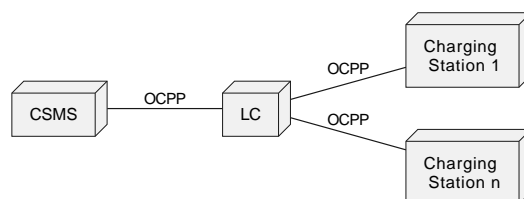


Figure 9. Multiple Charging Stations connected to CSMS via Local Controller

NOTE

Technically this topology can be realized in multiple ways. When using this setup with websockets, this implies that when a Charging Station connects to the Local Controller, it should open a websocket connection with the same address to the CSMS. The advantages of this approach is that the Local Controller can see all the messages and act on it, messages don't have to wait, firmware updates etc. on the Charging Stations are possible and the CSMS does not need special software. It could (in big installations) lead to a lot of websocket connections between CSMS and LC needed. For further information, please refer to OCPP implementation guide in Part 4.

8.4. Non-OCPP Charging Stations connected to CSMS via OCPP Local Controller

Description

This setup has multiple non-OCPP Charging Stations that are abstracted away using a OCPP enabled Local Controller. When applying OCPP in this situation, the LC should be considered as a Charging Station with many EVSEs or the LC should act as multiple OCPP Charging Stations (having their own Charging Station Identity).

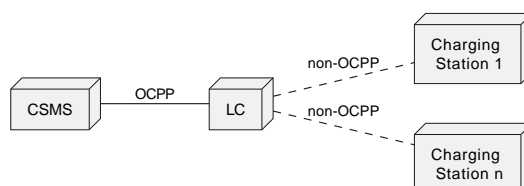


Figure 10. Multiple non-OCPP Charging Stations connected to CSMS via Local Controller

8.5. DSO control signals to CSMS

Description

This is a set-up in which the CSMS is the only application sending signals to its Charging Stations, but the CSMS receives smart charging signals from a DSO based on (most likely) grid constraints. This means that a non-OCPP signal such as OpenADR or OSCIP is received and based on this signal, the CSMS limits charging on its Charging Stations. CSOs that want full control over their Charging Station use this architecture, this way they are in control of the amount of energy being used by their Charging Stations. This can be done by sending charging profiles / charging schedules to Charging Stations.

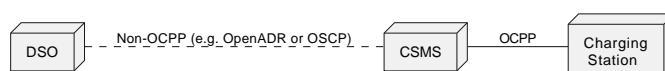


Figure 11. Smart Charging - DSO control signals to CSMS

8.6. Parallel control by CSMS and EMS

Description

In a (semi-)private situation where a Charging Station is not only connected to the CSMS, but also to an Energy Management System, some form of parallel control should be supported. OCPP should at least be used for Charging Station maintenance, but OCPP 2.0.1 also supports reporting external smart charging control limits. So if the Energy Management System decides that

charging at a later time is "better", the Energy Management System can impose an external limit (e.g. 0) to a Charging Station, which the Charging Station in turn can report to the CSMS via OCPP. The Energy Management System might get input from e.g. Local port of Smart Meter to prevent overloading connection but can also have other reasons for not charging (e.g. weather conditions).

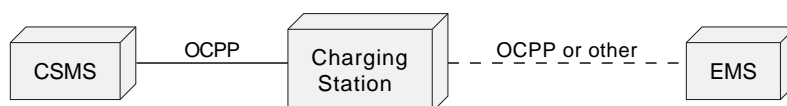


Figure 12. Parallel control by CSMS and EMS

Chapter 9. Part 1 Appendix: OCPP Information Model

9.1. Explanation of UML representation and message generation

In the next paragraph, the UML schemes of the OCPP Information Model are shown. The model is based on the Common Information Model (CIM) and to some extent to the CEFAC naming standards (only part of the standard). The objects in the model are named *BusinessComponents* and inherit properties from the CIM *IdentifiedObject*, such as MRID and Name. In the UML diagrams the attributes that are inherited from *IdentifiedObject* are shown under the *IdentifiedObject* stereotype (between < > >).

Other attributes are listed under the stereotype < < Content > >.

The messages in OCPP are derived from the model represented in the next paragraph, in a 3 step process:

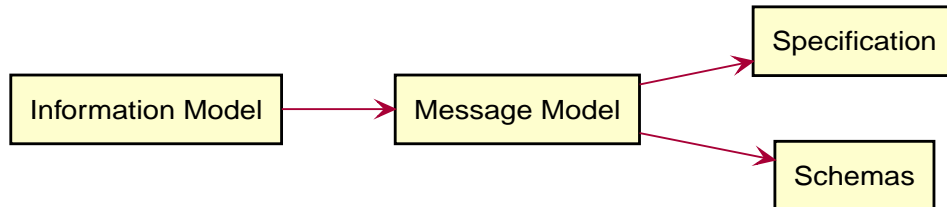


Figure 13. Process from information Model to Messages / schemes

After creating the Information Model, the messages are created based on the Information Model. However, in this transition (first arrow), some rules are (manually) applied for modelling messages. The most important rule that is applied, is that messages containing a reference to a <class> with only one <field>, are replaced by a field with the name <class><field>. For example, if a message contains a Transaction, with only an Id, this is replaced by a transactionId.

In the next step, when generating the messages and datatypes section of Part 2 of the specification, for readability, all Core DataTypes such as *CounterType*, are replaced by the Primitive DataType they refer to (except for enumerations) in this example *integer*.

9.2. Visual Representation of OCPP Information Model

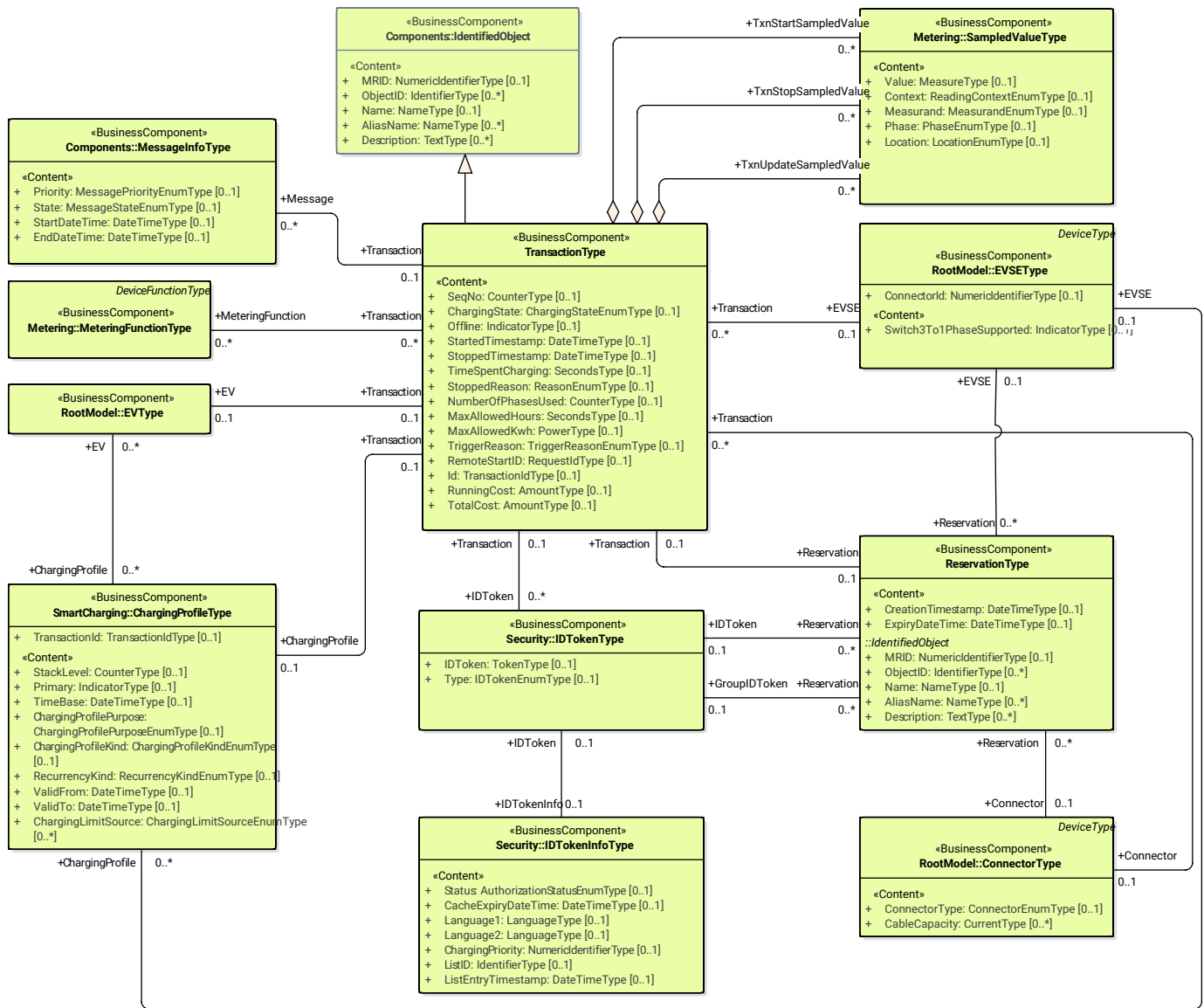


Figure 14. OCPP Information Model: Transactions

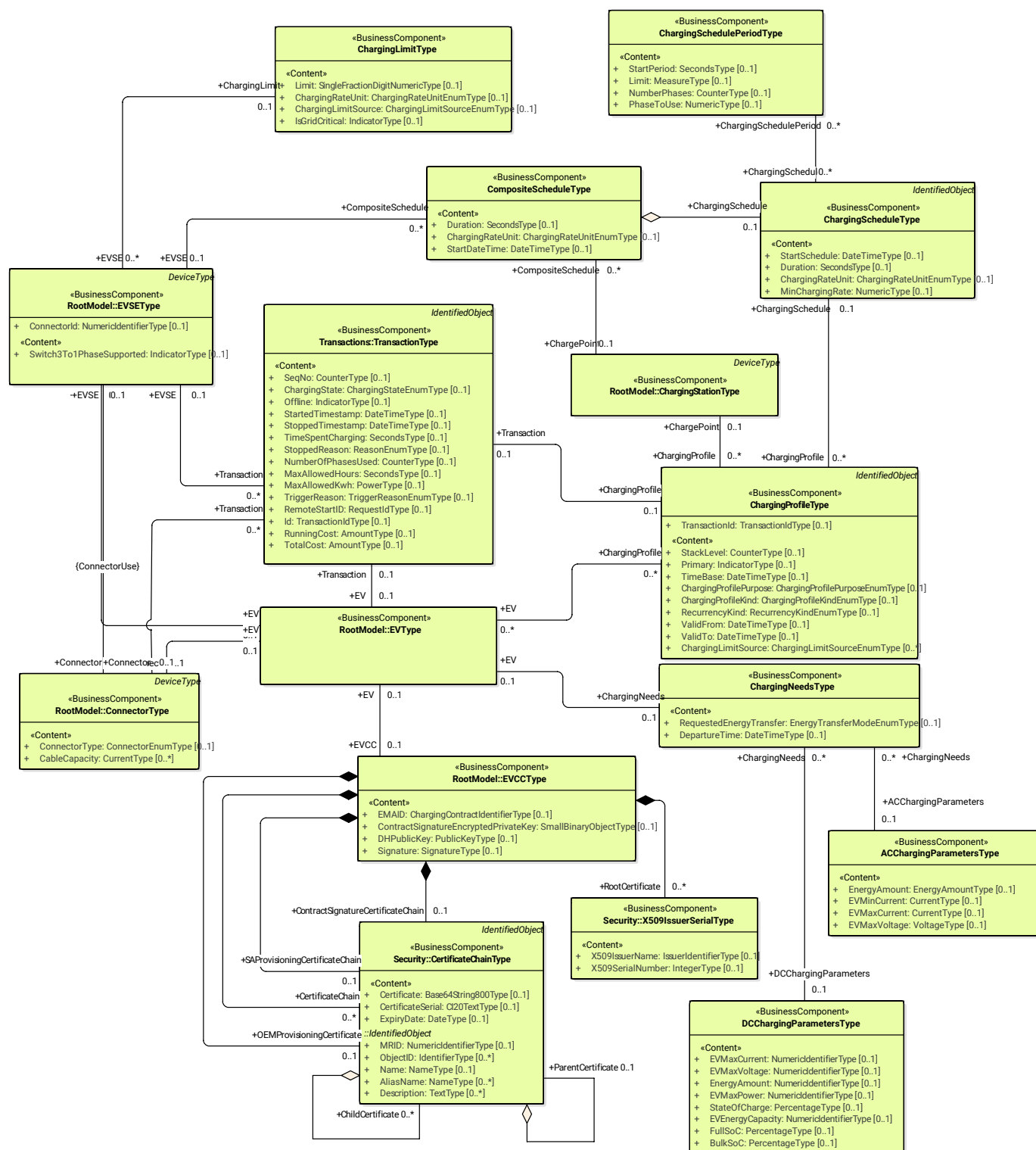


Figure 15. OCPP Information Model: SmartCharging

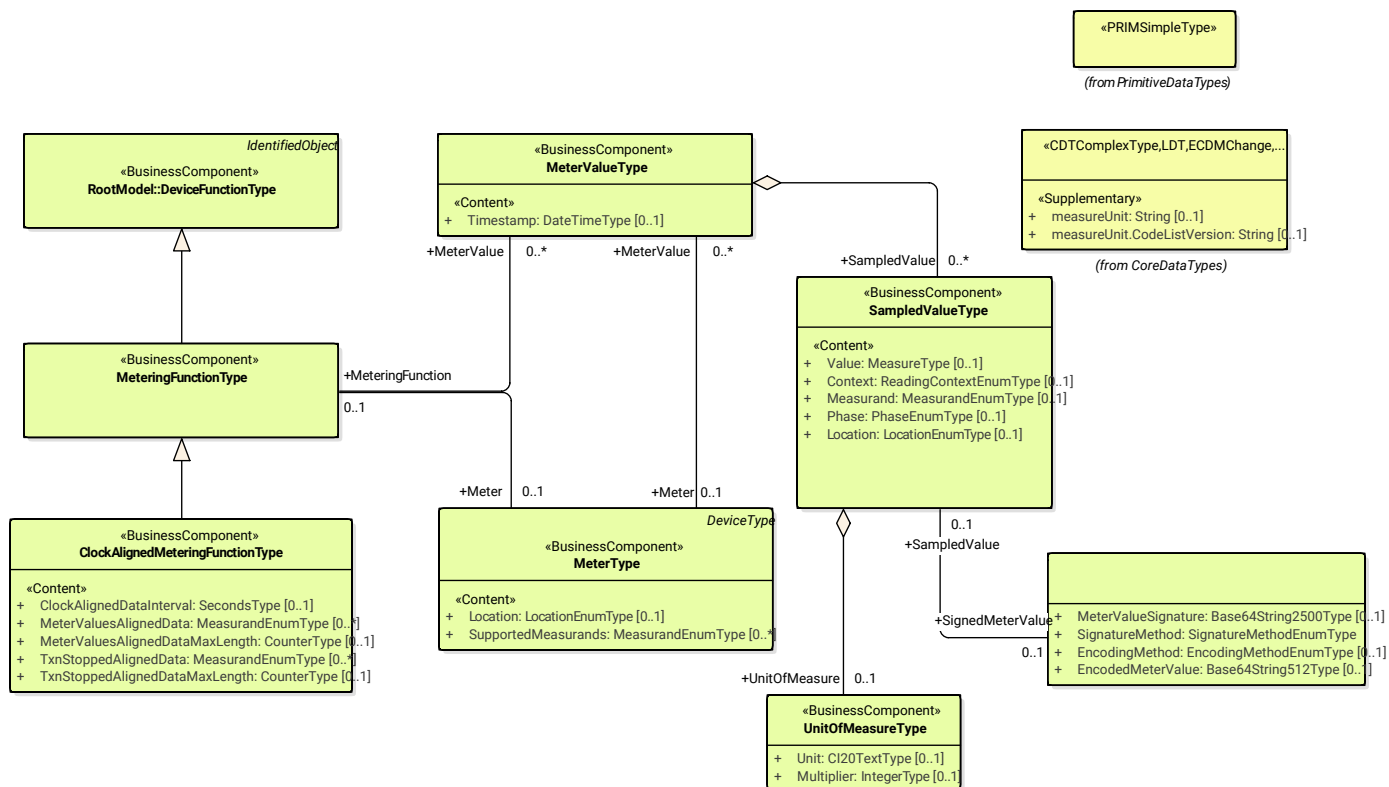


Figure 16. OCPP Information Model: Metering



Figure 17. OCPP Information Model: Device Model

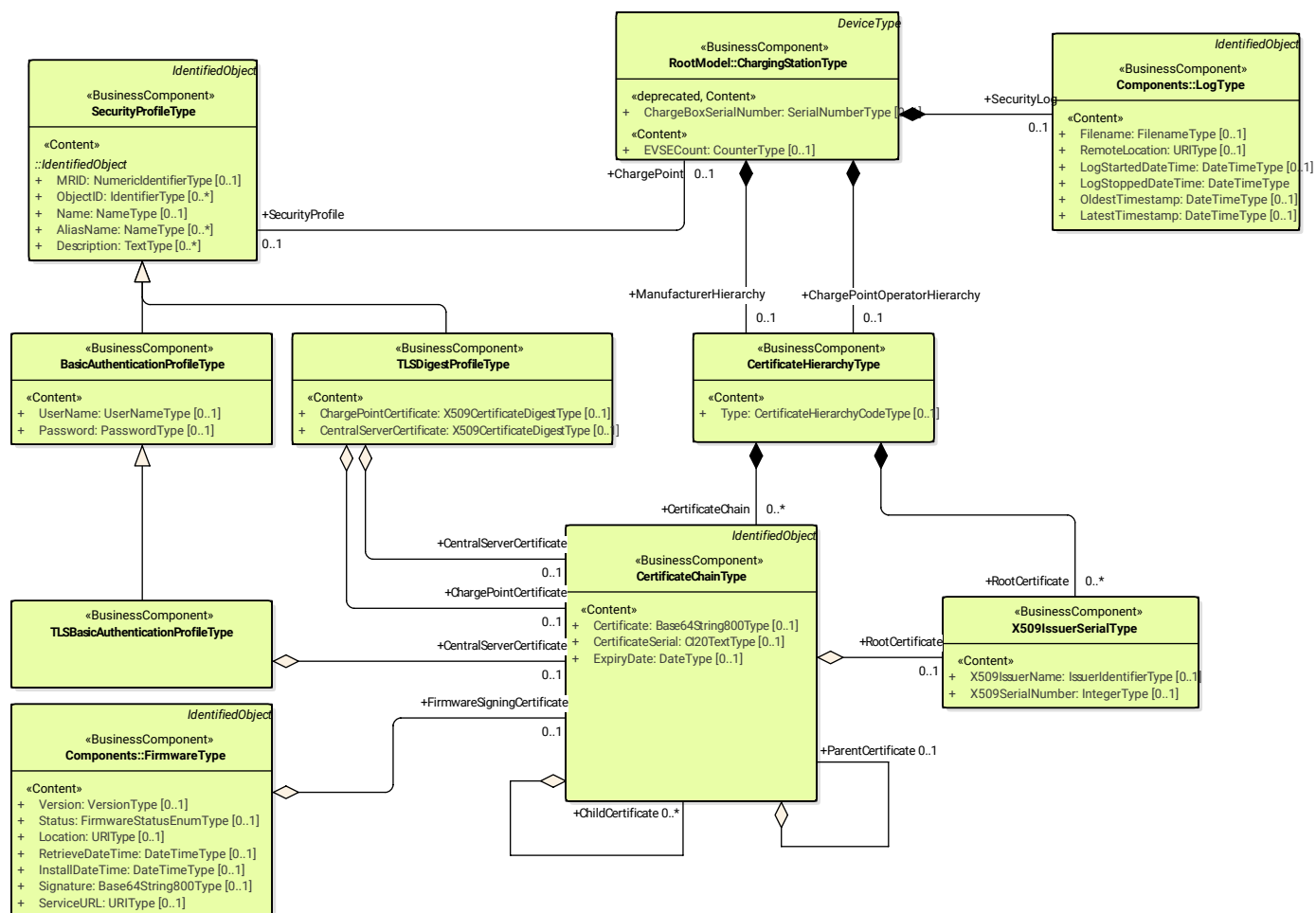


Figure 18. OCPP Information Model: Security-Profiles

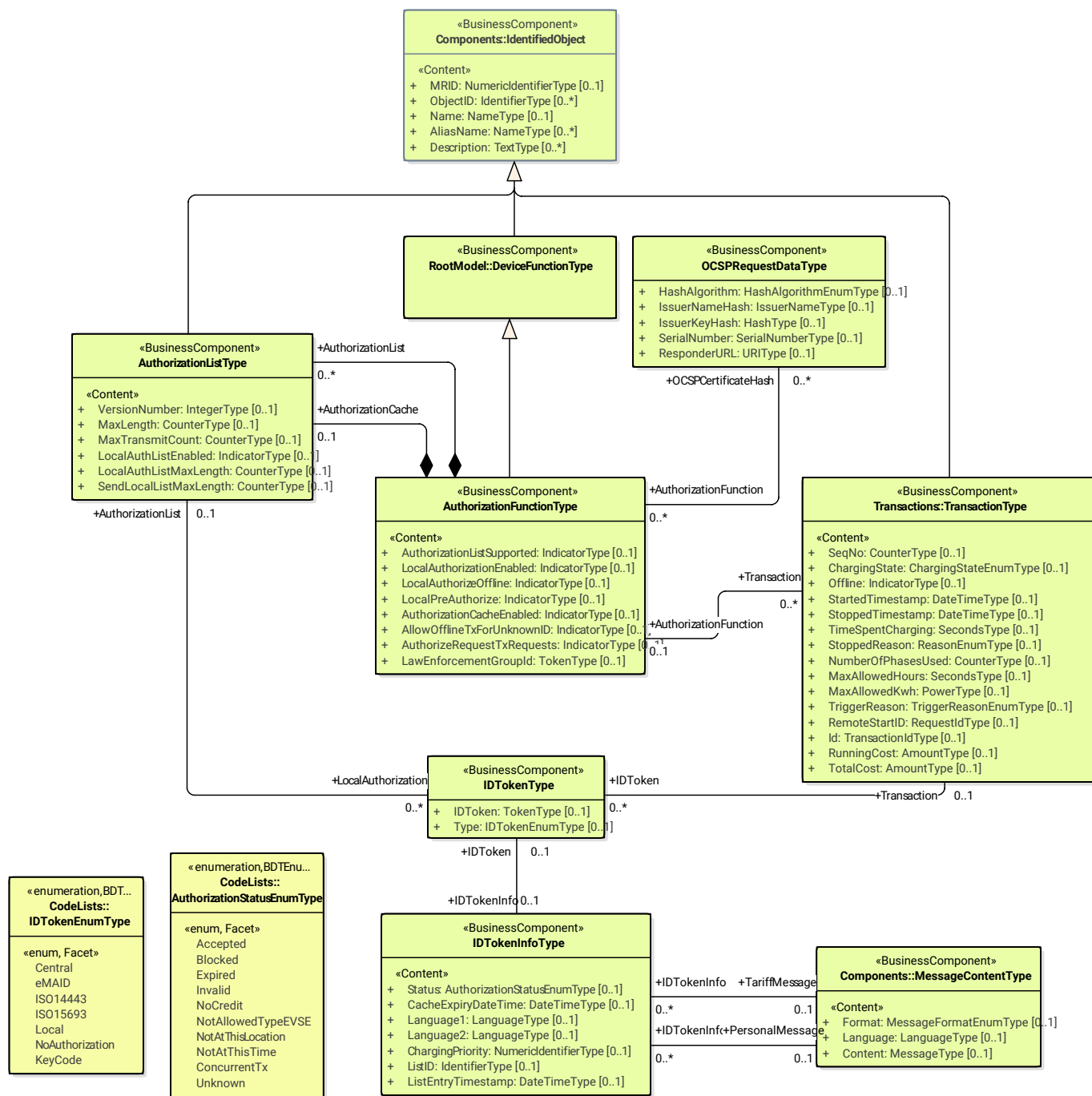


Figure 19. OCPP Information Model: Security-Authorization

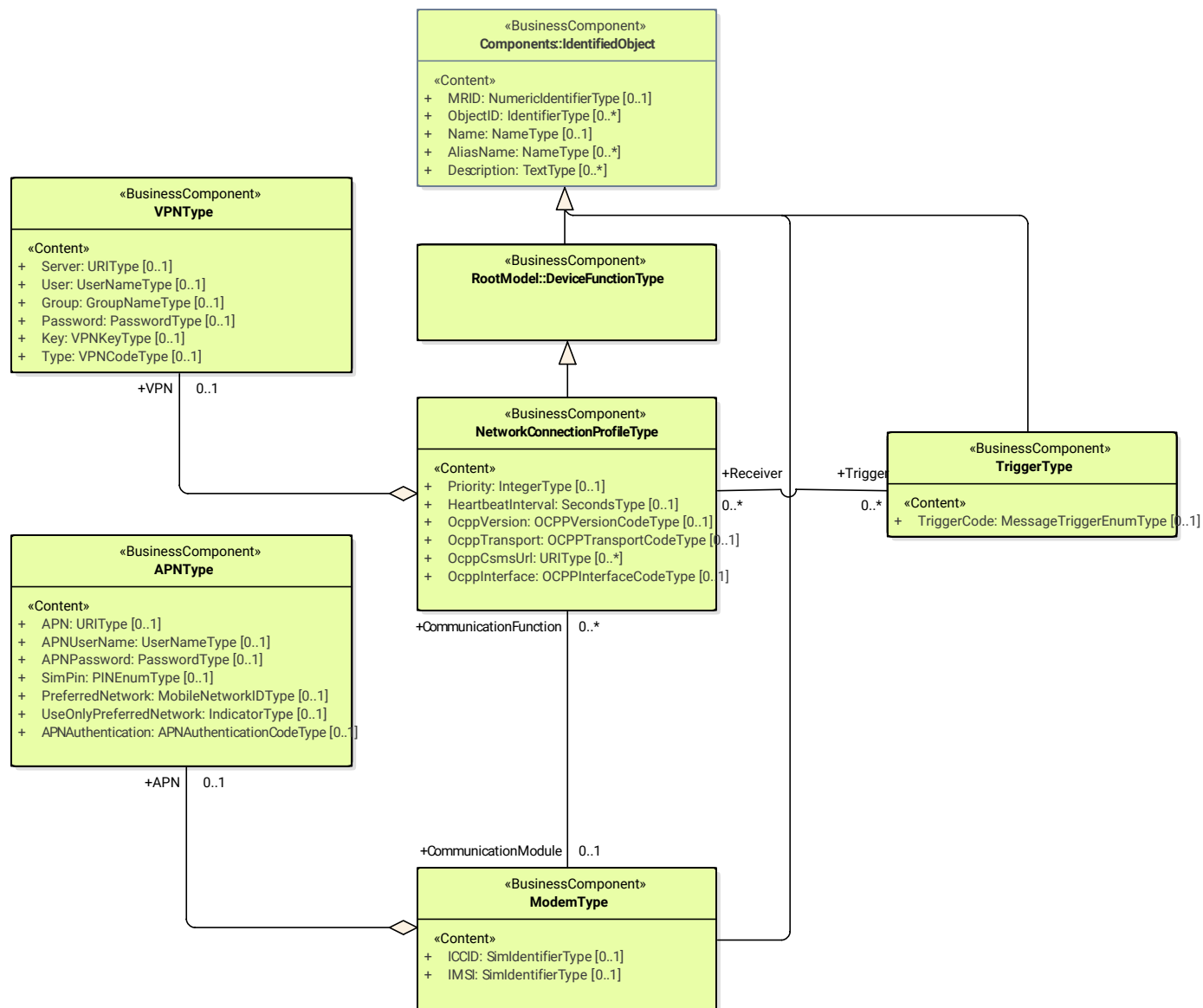


Figure 20. OCPP Information Model: Communication

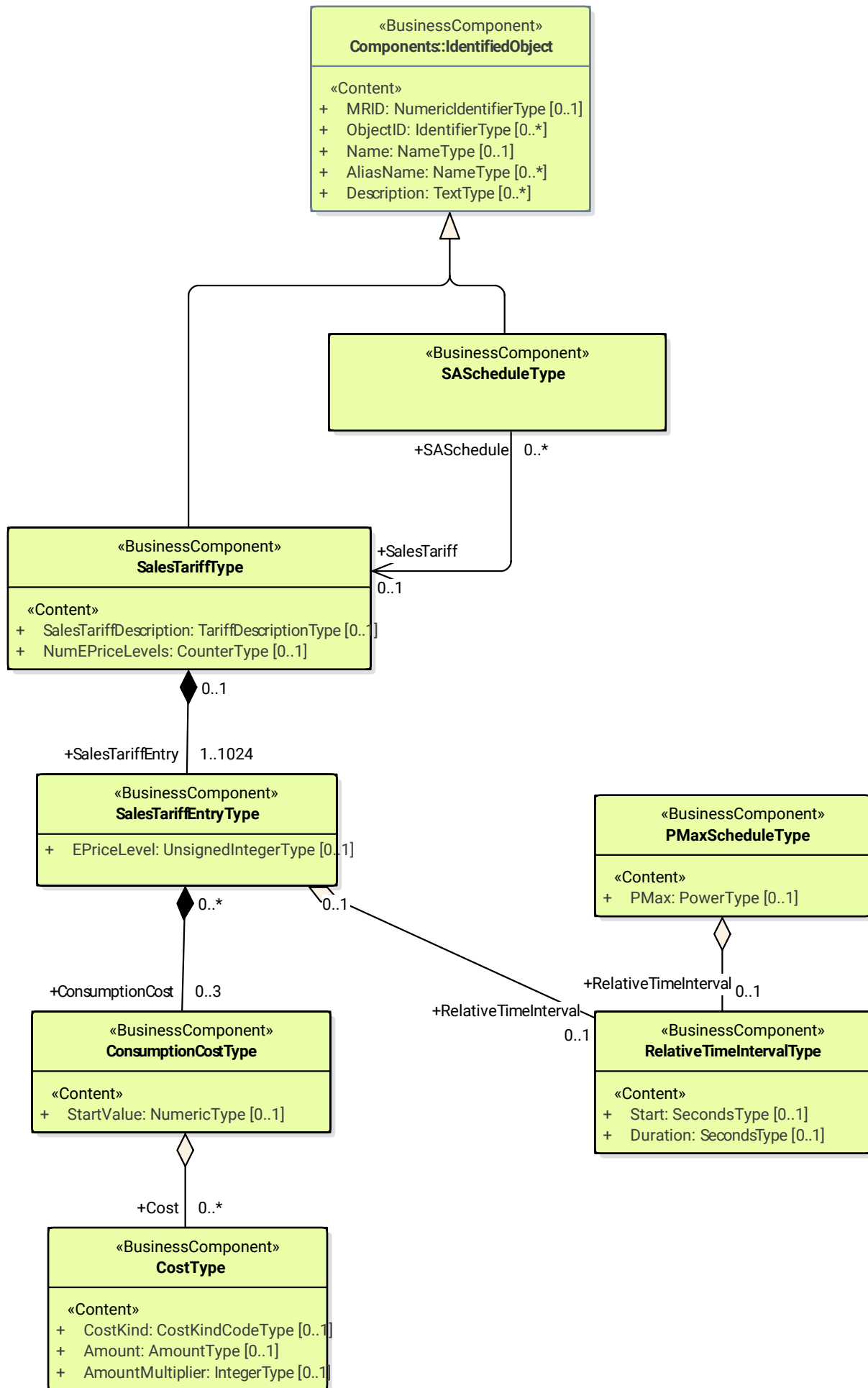


Figure 21. OCPP Information Model: SecondaryActorSchedule