applied sciences

MDPI

Article

Improving Conceptual Modeling with Object-Process Methodology Stereotypes

Hanan Kohen * and Dov Dori *

Faculty of Industrial Engineering and Management, Technion—Israel Institute of Technology, Technion City, Haifa 3200003, Israel

* Correspondence: hanank@campus.technion.ac.il (H.K.); dori@technion.ac.il (D.D.)

Featured Application: Introducing into OPM ISO 19450 modeling the ability to define and deploy stereotypes enables improved systems development using standardized complex structures and constrains.

Abstract: As system complexity is on the rise, there is a growing need for standardized building blocks to increase the likelihood of systems' success. Conceptual modeling is the primary activity required for engineering systems to be understood, designed, and managed. Modern modeling languages enable describing the requirements and design of systems in a formal yet understandable way. These languages use stereotypes to standardize, clarify the model semantics, and extend the meaning of model elements. An Internet of things (IoT) system serves as an example to show the significant contributions of stereotypes to model construction, comprehension, error reduction, and increased productivity during design, simulation, and combined hardware–software system execution. This research emphasizes stereotype features that are unique to Object-Process Methodology (OPM) ISO 19450, differentiating it from stereotypes in other conceptual modeling languages. We present the implementation of stereotypes in OPCloud, an OPM modeling software environment, explore stereotype-related problems, propose solutions, and discuss future enhancements.

Keywords: model-based systems engineering (MBSE); object-process methodology (OPM) ISO 19450; stereotypes; conceptual modeling

check for updates

Citation: Kohen, H.; Dori, D. Improving Conceptual Modeling with Object-Process Methodology Stereotypes. *Appl. Sci.* **2021**, *11*, 2301. https://doi.org/10.3390/app11052301

Academic Editor: Stefan Fischer

Received: 5 February 2021
Accepted: 3 March 2021
Published: 5 March 2021

1. Introduction

As system complexity is on the rise, there is a growing need for standardized building blocks to increase the likelihood of these systems to succeed. The realization and recognition in recent years that models can and should become the central artifact in engineered systems' lifecycles has become commonplace, giving rise to model-based systems engineering (MBSE) as an evolving systems engineering (SE) field. Conceptual modeling is the primary activity required for engineering systems to be understood, designed, and managed. Modeling languages enable describing the requirements and design of systems in a formal yet understandable way. Increasingly, the paradigms of Internet of Things (IoT) [1–4], Industry 4.0, [5] System of Systems [6,7], and Internet of Robotic Things (IoRT) [8,9] are gaining momentum. In these systems, software modules are embedded in the hardware components, control them, and communicate their status in near real-time via the Internet, relieving humans from mundane operations and serving their personalized needs accurately and more efficiently. As the trend of weaving software components into hardware ones accelerates, the need for a holistic approach, methodology, and framework for end-to-end systems development becomes ever more apparent.

MBSE has become the leading systems engineering (SE) paradigm, and it has provided SE with a hitherto sorely missing formal facet. Conceptual modeling is the primary activity required for engineering systems to be understood, designed, and managed. This underlying MBSE process precedes, or should precede, mathematical, physical, geometrical, and disciplinary modeling.

Using model-based approaches has numerous benefits, including reduction of risks, early error detection and prevention, team communication enhancement, traceability, and an explicit methodology for reasoning about the modeled system [10–12]. MBSE is not just about modeling; it is SE that produces a formal conceptual model which serves as MBSE's comprehensive underlying blueprint—the reference artifact that constitutes the source of authority of the various system aspects: requirements, performance, functionality, structure, dynamics, and many other physical and informatical (cybernetic) aspects. To achieve this level of authority and trust, MBSE requires a rigorous conceptual modeling methodology, which encompasses a universal ontology, a language, a set of principles and guidelines, and an enterprise-wide supporting modeling software environment.

Object-Process Methodology, OPM [13–15], is a holistic formal yet intuitive approach, language, and methodology for the representation of knowledge and the development of complex systems. The OPM language is bimodal—it uses both graphics and text to represent knowledge at various, interconnected levels of detail about the two major aspects of any system: structure and behavior. This bimodal representation caters to humans' dual channel processing—the first of the three assumptions of the cognitive theory of multimedia learning [16,17], which states that there are two separate channels for processing information: the auditory and the visual. The two other assumptions of the cognitive theory of multimedia learning are that (1) the capacity of the channels is limited, and (2) learning is an active process of filtering, selecting, organizing, and integrating information. OPM is designed with these three cognitive assumptions [18]. It accounts for the second assumption by its top-down hierarchical decomposition via abstraction-refinement mechanisms, reducing cognitive load [17]. The third assumption is taken care of by engaging the modeler in actively organizing and linking model elements, and by visually executing (simulating) the model to gain deep understanding of the dynamic aspect of the knowledge it represents.

In conceptual modeling, a stereotype is a modeling construct that was introduced into the Unified Modeling Language (UML) [19–21] to enable modelers to extend the basic UML metamodel structure without modifying the metamodel itself. Introducing stereotypes has led many modelers to misuse them, particularly where regular domain-level modeling would be more appropriate [22]. In most cases, the OPM equivalent of UML stereotypes could be accomplished by means of OPM itself, without the need to resort to using the stereotype mechanism. However, recent technological developments, such as the proliferation of IoT and industry 4.0, have raised the need to introduce a robust stereotyping mechanism into OPM, which is the focus of this article.

2. Literature Review

2.1. Model-Based Systems Engineering (MBSE)

According to Ramos et al. [11], MBSE is the (then new) standard for product engineering, stemming from improvements in technology and communication, which have led to a massive growth of a disparate mix of corporate engineering and design systems. Solution teams from various companies must work together consistently across corporate and international borders to provide their consumers with top-quality solutions. To gain an increased market share, companies have been creating proprietary solutions that are often incompatible with adjacent systems. As a result, distributed systems, which span various domains and enterprises, become ever more complicated, posing maintenance challenges, and making it difficult to collaborate and exchange data globally.

Modern systems are increasingly an amalgam of hardware and software, with the software acting as the intelligent controller of the hardware, making system-level decisions, and communicating with the rest of the world. This has culminated in the emergence of the Internet of Things (IoT) and its glaring Industry 4.0 manifestation [5], a revolution that is changing dramatically the way systems operate and the modes of human-system interaction.

2.2. Internet of Things (IoT)

The Internet of Things (IoT) is a vision of a ubiquitous Internet that integrates everyday physical objects via information networks. Such objects must also have the capability to communicate, even in environments where fixed network access infrastructure is weak or nonexistent. The increasing role of software in systems calls for a parallel change in the ways new systems are modeled and developed, and in how the software components of these systems are created and tested alongside the hardware they control.

Traditionally, systems engineers have not been software experts, and the development of the software embedded in such systems has been the role of specialized software engineers and programmers. This hardware–software development separation has created a chain of activities with systems engineers first defining the system. Only later, the software development could be delegated to software engineers and programmers, who often lack system-level understanding, resulting in software that does not serve the function of the system in the best possible way. When the software is developed, it has to be tested on the hardware of the system and in its operating environment, revealing bugs and problems that must be resolved, causing major rework, delayed project completions, cost overruns, and damage to corporates' reputation.

The concept of hardware–software co-design is not new [23], but it has been limited almost exclusively to the design of "embedded systems"—a narrowly construed concept of developing the hardware of digital circuits hand-in-hand with their associated software. In this specific context, hardware–software co-design involves such issues as concurrent programming and field-programmable gate arrays [24]. The need for a paradigm shift in the development of software-intensive systems is growing hand-in-hand with the intertwined embedding of software within hardware in current advanced "smart" systems and almost all future systems, which can increasing be classified as IoT systems.

2.3. Stereotypes

A stereotype has been defined as "*A well-formed mechanism for expressing user-definable extensions, refinements or redefinitions of elements of the language without (directly) modifying the meta-model of the language*" [19]. Modeling languages used primarily for MBSE employ stereotypes to empower users to extend, or even modify, the modeling language. Modelers use stereotypes to adapt the language to specific situations or needs, such as creating a specific structure for an airplane in the aerospace domain.

Designers of modeling languages, who introduced stereotypes to overcome certain language limitations, created a well-defined set of extension mechanisms—general-purpose model elements for language customization. Stereotypes respond to local needs and requirements, such as the ones arising in a specific domain, software development process, or problem [19,21,22]. To meet such needs, stereotypes provide means of customizing visual, general purpose, and object-oriented modeling languages, as well as specific constraints.

A stereotype can add new properties to elements of the underlying language or modify existing ones. Stereotypes often express specific properties or add specific semantics to a certain kind of model element. Accordingly, stereotypes range in scope and kind from creating a basic structure, e.g., of a car, or adding a constraint to a system design, all the way to introducing specialized notation for modeling IoT systems security [25].

With both positive and negative effects, a stereotype is like a double-edged sword: When used correctly, stereotypes help balance the model, formalize it, and improve its understandability. However, if stereotypes are misused, they can have adverse effects of distorting the model in which they are used, making it more difficult to understand and implement.

The roles of stereotypes have been investigated mainly for UML and SysML [19–21,26]. Creating a stereotype in UML can be done in different ways by different tools: one can apply stereotypes to a range of model element types, such as Elements (e.g., Classes, Objects), Relationships (e.g., Dependencies, Associations), Association Ends, Attributes, and Operations. A UML stereotype is the only kind of meta-class that cannot be extended by stereotypes [27]. This is not the case with OPM; As we show later, OPM stereotypes can be recursively extended by other OPM stereotypes (see Section 3.5). A UML stereotype is used mostly in cases when a primitive building block is needed frequently, so providing the UML modeler with a stereotype saves the time needed to redesign that building block repeatedly. An example from [25] for a use of a UML stereotype, related to IoT systems, is presented in Figure 1.

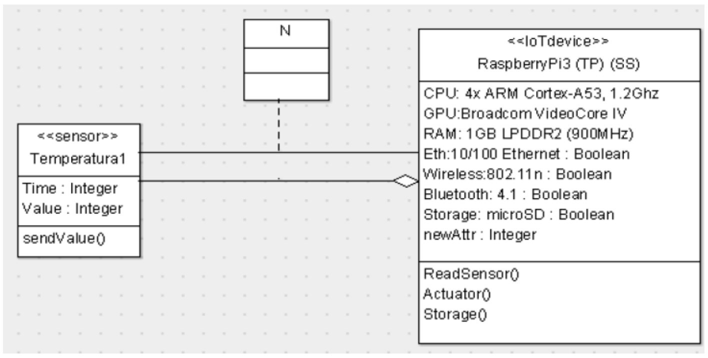


Figure 1. An example of a UML stereotype for an IoT device in a class diagram, taken from [[25], 2021, IEEE].

We have not found a case in which a UML stereotype serves to define a standard building block that includes definition of constrains, such as allowable ranges, which can be then verified in the model statically, let alone in runtime. OPM stereotypes provide such options. Range and value validation are defined in Section 4, and their implementation is demonstrated in Section 5.

No research about the need to introduce OPM stereotypes and how they should be designed and implemented has been carried out. A likely reason for this is that until recently, the need for stereotypes did not arise in OPM, because unlike UML and SysML, OPM uses a single kind of diagram, Object-Process Diagram (OPD), enhanced by its complementary, textual modality—OPL, Object-Process Language (a subset of English or any other natural language), which describes in a subset of natural language precisely what the OPD describes graphically. The combination of a single diagram kind and the bimodal visual-textual model representations has provided a robust capability to model systems of various kinds and domains using a minimal set of elements defined in its core metamodel: stateful objects and processes that transform objects by creating or consuming them, or by changing their states. OPM has an inheritance mechanism, which extends the familiar inheritance of the object-oriented (OO) paradigm: The generalization-specialization relation induces inheritance from the generalizing superclass thing (object or process) to the specialized thing. The inherited elements are not only features (attributes and operations), as in the OO approach, but also of states and of structural and procedural links. Moreover, inheritance is applicable not just to objects but also to processes. This minimal universal ontology of OPM and the robustness of the core OPM is the main reason for the lack of need, until recently, for defining, creating, and using OPM stereotypes. Therefore, there are no examples that can help determine how OPM stereotypes should be developed and deployed, and this is what we present in this work for the first time.

3. Stereotypes in OPM and OPCloud

3.1. OPCloud

OPCloud [28,29] is a cloud-based software environment that utilizes the web for creating and storing OPM models according to OPM ISO 19450:2015. OPCloud supports the construction of correct-by-construction OPM models in a friendly, collaborative way. Correctness-by-construction of OPM models is ensured by guiding the modeler. For example, if the modeler is about to connect two OPM entities (objects, processes, or object states), OPCloud presents all the possible links between them that are syntactically correct, expressing the semantics of their connection via corresponding OPL sentences.

OPCloud operations include layout of stateful objects and processes, routing links, searching the models for entities, easy navigation between OPDs, dragging-and-dropping of OPM things on the model canvas, automatic generation of OPL from the graphics input, collaborative model editing through a web browser, model sharing with transferrable editing rights, commenting, exporting the entire OPM model in PDF or parts of it in JPEG formats, full control over styling, and a modern, slick graphic user interface. OPCloud includes several extensions that enhance OPM, including MAXIM [30]. MAXIM enables introducing into the model quantitative and computational objects and processes, as well as executable code fragments within leaf processes. It provides for communication with external software packages, such as MATLAB, and bidirectional connections to get inputs from hardware components, such as sensors, and to send signals to actuators and servos, such as Arduino [31].

OPM ISO 19450:2015 specifies in Annex C the OPM structure metamodel, depicting the conceptual building blocks of OPM as parallel hierarchies of the graphic and textual OPM modalities, which produce equivalent representations in the visual and verbal modalities. In order to incorporate OPM stereotypes into OPM, in Figure 2 we updated the OPM structure metamodel to reflect the facts that (1) an OPM stereotype is a specialization of an OPM model, and (2) an OPM thing can be anchored to a model. While an OPM model can include multiple OPDs at many detail levels, OPM stereotypes can currently contain only one OPD due to an OPCloud limitation of model merging, which is planned to be removed in a future version.

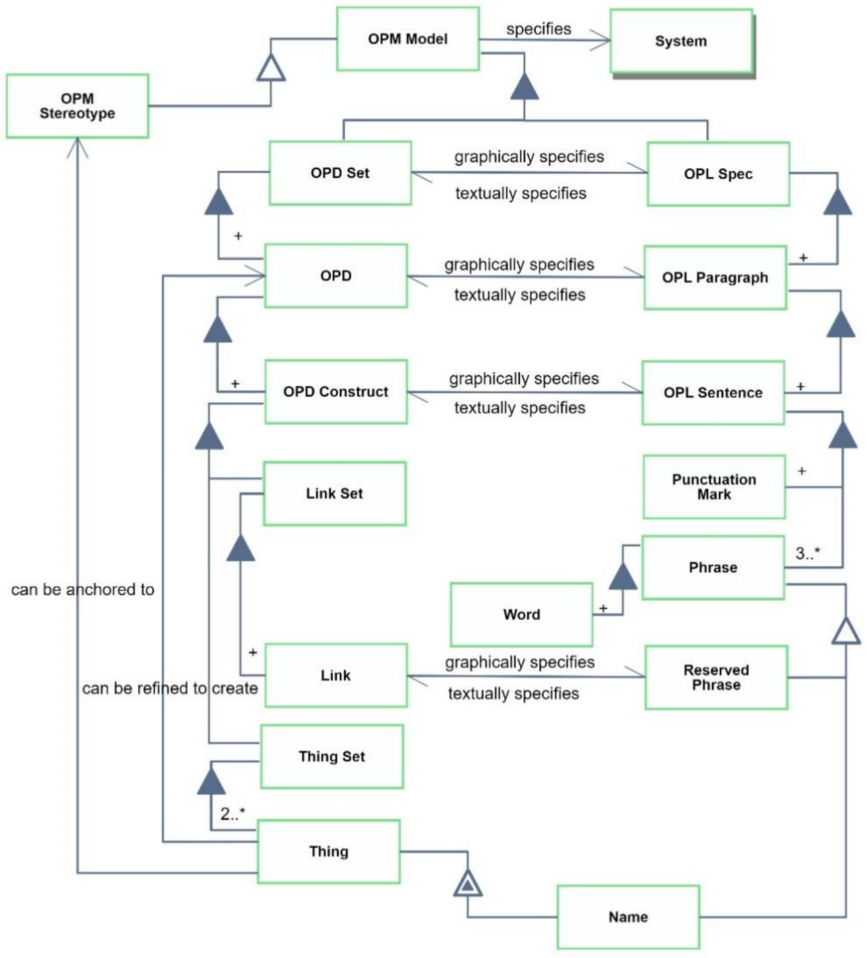


Figure 2. ISO 19450 OPM structure metamodel (ISO 19450:2015 Annex C) updated with OPM Stereotype.

To enhance OPM ISO 19450:2015 with OPM stereotypes, we added to OPCloud a dedicated stereotype extension, which provides for adding stereotypes defined by OPCloud system admins at the global OPCloud level of or by organization admins at the organization level.

3.2. Global and Organization Stereotypes

Two kinds of OPM stereotypes are defined in OPCloud: (1) Global Stereotype, defined by OPCloud system admin, which is a generic stereotype that can be used by OPCloud modelers in any organization, and (2) an organization-specific stereotype, defined by that organization admin, which can be viewed and used only by that organization.

Global stereotypes are available to all OPCloud organizations and individual users. Only OPCloud system admins can add or update a global stereotype, while an organization admin can add and update their own organization stereotypes. At the organization level, an organization admin can create a stereotype for use by all the modelers in that organization. Only the organization admin can create and edit or update organizational stereotypes. The OPD in which the stereotype is defined has a read-only access. A modeler can use a stereotype in the model he develops by selecting it from the global stereotype list or from the organization stereotype list. Once the modeler incorporated a stereotype into her model, she cannot change it, but she can extend it to suit her needs.

An admin who creates a new stereotype must apply a systems thinking approach and exercise great caution [32,33], since a non-admin modeler cannot modify the stereotype, only extend it. Undisciplined use of stereotypes can lead to proliferation of incompatible versions, making the model difficult to understand and maintain. Additions or changes an admin makes to a stereotype do not affect the stereotypes of the same kind that are already anchored in models. Similarly, additions or changes a modeler makes to a stereotype that is linked to an anchor in a model do not affect the stereotypes of the same kind linked to other anchors in the model.

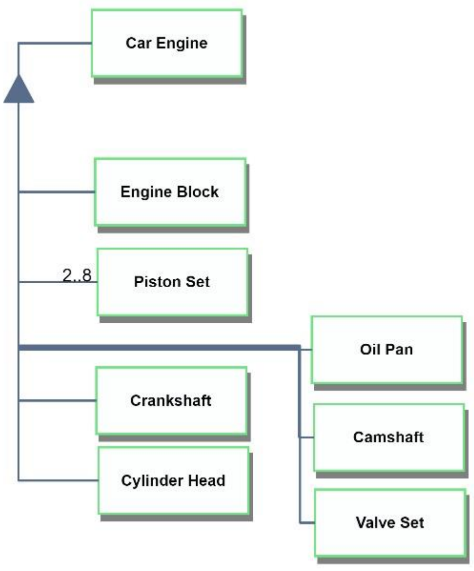
There is no difference in the mechanism of creating stereotypes of the global and organization kinds. The only difference is in that an OPCloud user defined as an organization admin can create and edit stereotypes for only her or his organization, while a user defined as an OPCloud system admin can create or edit global stereotypes in addition for creating and editing stereotypes for his own organization.

3.3. Descriptive and Prescriptive Stereotypes

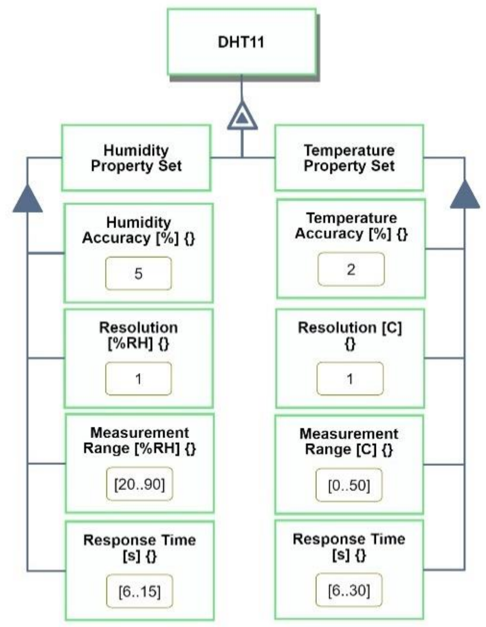
OPM stereotypes can be classified as descriptive and prescriptive, based on their role. A descriptive stereotype specifies the structure of an OPM thing (object or process), aimed to improve the understandability of a model, analogous to the way a good illustration improves the understandability of a textual specification. A prescriptive stereotype is more elaborate, as it can include modeling constrains and validations, which increase the expressive power and verifiability of OPM models that implement them. A prescriptive stereotype adds structured annotations with semantic restrictions, enabling the specification of rules that cannot be directly expressed in the base OPM language and checking for compliance with given semantic restrictions. Both descriptive and prescriptive stereotype kinds are implemented in OPCloud. Examples of descriptive and prescriptive stereotypes are presented in Figure 3.

3.4. Creating an OPM Stereotype

Creating an OPM stereotype starts with creating an OPM model. A properly defined stereotype balances formality and understandability. Initially, the stereotype includes only a base OPM thing (object or process) and its parts. If the stereotype contains one or more computational objects, their default values can be assigned. Currently, the default value for any computational object, for example Object B, is "value", so if we set the stereotype's default value of Object B to "yes", resetting an implemented stereotype to its original will revert the value of Object B to "yes". We elaborate on stereotypes values, default values, and their validation in Section 4.



(**a**)

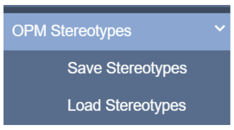


(**b**)

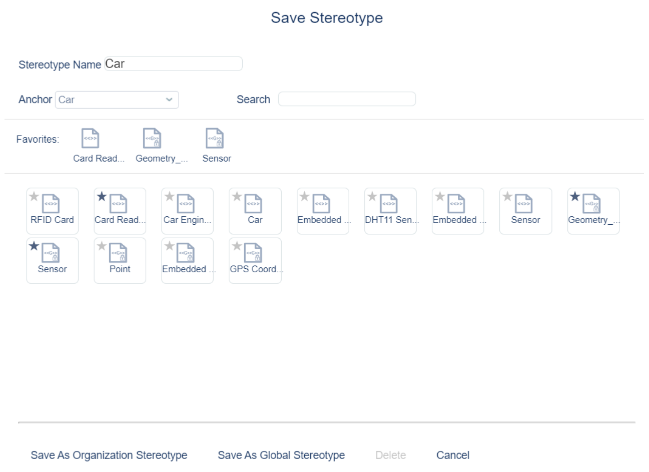
Figure 3. Examples of descriptive (**a**) and prescriptive (**b**) stereotypes.

In each stereotype, a single OPM thing must be defined as the stereotype anchor—the thing that will be anchored to and substitute the model anchor—the thing to which the stereotype is added in the model.

Once an admin created the OPM model designed to become a stereotype, she selects from the main menu the "Save Stereotype" option (see Figure 4a), which opens the Save Stereotype screen (see Figure 4b), enabling the admin to save the model as a stereotype. Here, the admin also has the option to delete an existing stereotype.



(**a**)


Save Stereotype

Stereotype Name Car

Anchor Car

Search

(**b**)

Figure 4. The Save Stereotype menu for OPCloud admins. (**a**) OPCloud main menu OPM Stereotypes options. (**b**) the Save Stereotype screen.

Once saved, the name of the model is changed to include the prefix **<<Stereotype>>**, indicating that this model is a stereotype. As an example, in Figure 5, **Car** is saved as a stereotype, getting the name **<<Stereotype>> Car** and becoming an organizational stereotype.

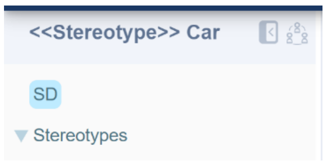


Figure 5. A Stereotype model name example.

As noted, saving and editing a stereotype is available only to admins, because to preserve its utility and value, a stereotype must remain coherent and usable to all the modelers without worrying about changes that might invalidate their model.

3.5. Using Stereotypes in an OPM Model

To use a stereotype in a model under development, the modeler first selects the model anchor—the thing in the model to be linked to the stereotype. Selecting the anchor opens the Set Stereotype menu (see Figure 6), where the available stereotypes are presented (see Figure 7). When the modeler chooses the appropriate stereotype from this list, the stereotype anchor is merged with the model anchor and the stereotype is automatically added to the model as a new read-only OPD view. Having the stereotype available to the modeler in her model enables her to easily review and examine the stereotype and determine how to best utilize it in her model.

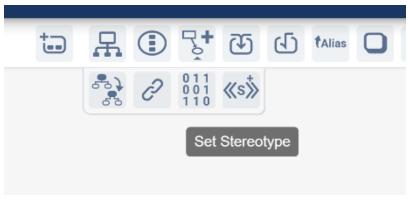


Figure 6. The Set Stereotype button in OPCloud secondary toolbar.

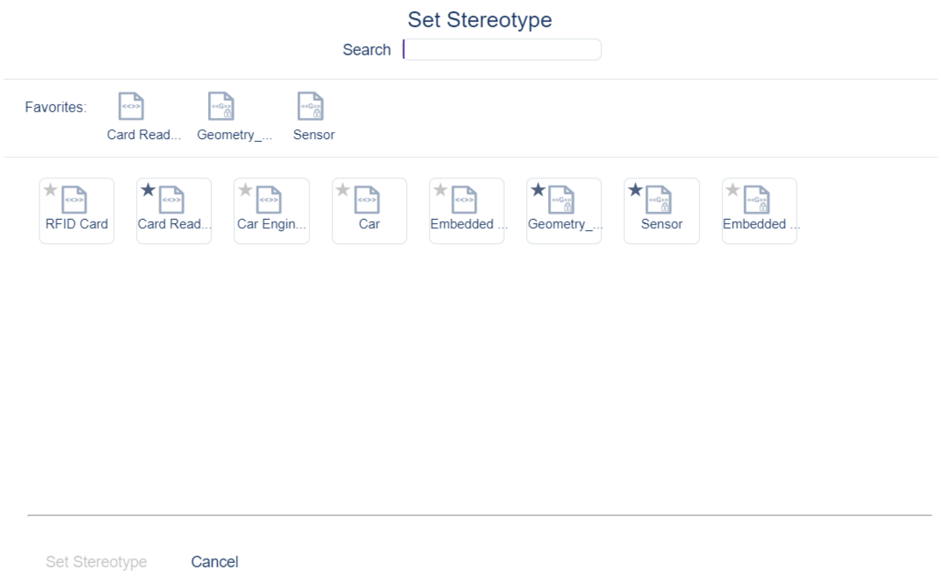


Figure 7. The Set Stereotype screen.

In the following example, we elaborate on the details of this process. Suppose the modeler chooses from her model the object **Car** as the model anchor to be anchored as a stereotype. Clicking on **Car** causes the "Set Stereotype" button with the icon **<<s>>$^+$** to appear in the secondary OPCloud toolbar, as shown in Figure 6.

Clicking this Set Stereotype button, the one with the icon **<<s>>$^+$**, opens the Set Stereotype menu screen, shown in Figure 7.

In the Set Stereotype screen, the modeler can view and select a stereotype. She can use one of her favorite stereotypes, located in the Favorites bar at the top of the screen, or select any other stereotype listed below the Favorites bar. First, the organizational stereotypes are listed, and then the global stereotypes are listed. A modeler cannot add a stereotype to a thing that is already linked as a stereotype. To replace a stereotype, the modeler first needs to remove the previous stereotype and then link the newly selected stereotype.

In the Set Stereotype screen, an organization-specific stereotype icon is designated by the string **<<>>** in it (e.g., **RFID Card** in Figure 7), while an OPCloud global (generic, system level not related to specific organization) stereotype is marked by **<<G>>** (e.g., **Sensor** in Figure 7). A solid (non-blank) star at the top-left corner of the stereotype icon, as in **Sensor** in Figure 7, indicates that the stereotype is favorite. Clicking on a blank star in a stereotype icon will make that star solid and move the stereotype up to the favorite stereotypes list. Conversely, clicking on a solid star makes it blank and removes the stereotype from the favorite stereotypes list.

To anchor a stereotype (which is a single OPD) to the anchor thing—the thing model to which the stereotype is connected, that thing needs to be selected, and then it is anchored as a stereotype. For example, in Figure 8, to anchor **Engine** into the stereotype **<<Car Engine>>**, **Engine** (which was the object at the bottom of Figure 8 before this operation) is selected by clicking on it. Then, the Set Stereotype button at the bottom left of the Set Stereotype screen (see Figure 7) is clicked, causing **Engine** to be anchored to the stereotype and renamed **<<Car Engine>> Engine**.

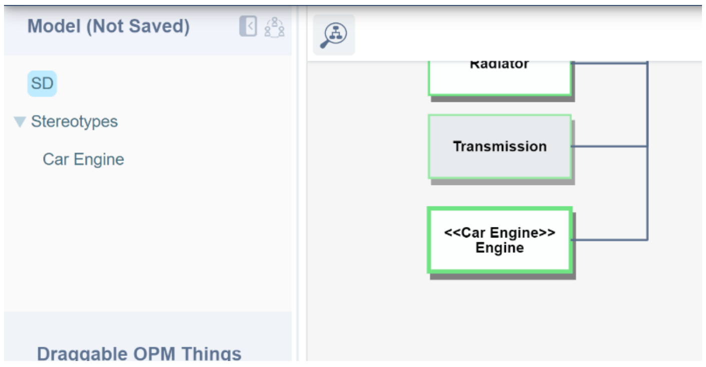


Figure 8. Engine is anchored into the **<<Car Engine>>** stereotype.

When a modeler tries to add to a thing a stereotype that is already anchored to another thing in the model, no new read-only OPD of the stereotype is created. Rather, the modeler is referred to the existing read-only view OPD of that stereotype. For example, if a model contains two **Engine** objects, and both are anchored with the **<<Car Engine>>** stereotype, only one view OPD is added to the OPD tree under the Stereotypes list (see Figure 8 left).

The name of a part of a thing declared and anchored as a stereotype is renamed to reflect the new thing with adding the prefix of the thing's name before the name of the part as it is called in the stereotype. For example, consider the stereotype **<<Car>>** in Figure 9, which has a part **Alternator** linked to it, as denoted by the aggregation-participation link. In the model, the object **Porsche 911 is** anchored to the stereotype **Car**, becoming **<<Car>> Porsche 911**. This object, **<<Car>> Porsche 911**, inherits the parts of **<<Car>>** with adjusted names, as exemplified in Figure 10. Thus, **Alternator** becomes **Alternator of Porsche 911**, and **<<Car Engine>> Engine** becomes **<<Car Engine>> Engine of Porsche 911**.

If one or more parts of the stereotype are themselves stereotypes, they will be added to the Stereotypes list in the FOPD tree. For example, if we create an object **Porsche 911** and anchor it to the **<<Car>>** stereotype, **<<Car>> Porsche 911** will also include the **<<Car Engine>>** stereotype.

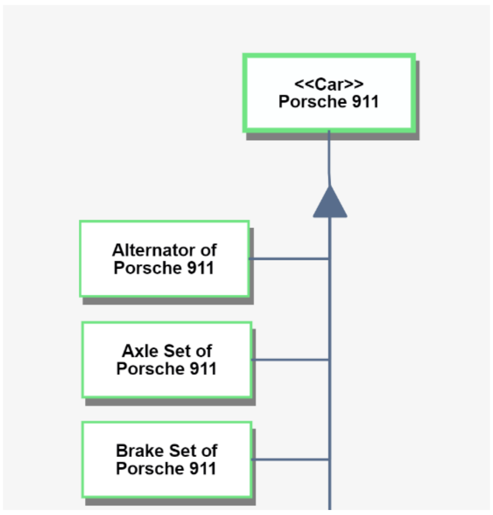


Figure 9. Stereotype naming convention example.

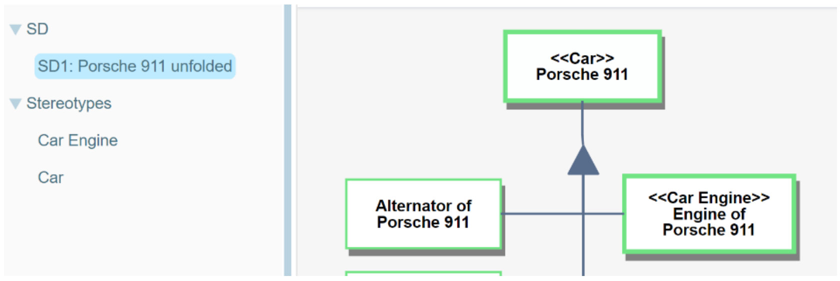


Figure 10. Porsche 911 with <<Car>> and <<Car Engine>> stereotypes.

3.6. Immutable and Mutable Stereotypes

A stereotype has a mutability meta-attribute with two possible values: immutable and mutable. An immutable stereotype that was anchored to a thing in the model is disconnected from its original stereotype. This disconnect ensures model stability: once the immutable stereotype is incorporated into the model, it is not affected by changes done later to its origin stereotype, so updates to the origin stereotype are not reflected in the model to which the stereotype had been added. The modeler should not worry that the stereotype might change.

A mutable stereotype, which is not yet implemented in OPCloud, shall remain linked to the things that anchored to it, so any change to that mutable stereotype shall be reflected in all the OPM models in which it is implemented.

Both the mutable and the immutable stereotypes have advantages and disadvantages, so (when available,) the modeler can choose the kind that best fits her needs. The advantage of an immutable stereotype is that since it is disconnected from its origin, the modeler has a "peace of mind," not having to worry about possible problems that might arise from changes at the stereotype level. this, however, is also the disadvantage of the immutable stereotype: Any improvement at the stereotype level will not propagate to the anchor at the model level.

Conversely, the advantage of a mutable stereotype is the complement of its immutable counterpart: a mutable stereotype is connected to its origin, so any update (that presumably provides some kind of improvement) at the stereotype level will automatically propagate to its anchor at the model level. To ensure that the update does not invalidate the model with the mutable stereotype anchor, the modeler must make sure that no problems would arise from changing the mutable stereotype at the stereotype level. This can be accomplished by careful design of the interface, ensuring that improvements in the mutable stereotype will benefit the model without side effects.

4. OPM Computational Objects: Stereotype-Related Extensions

Having elaborated on OPM stereotypes and their diverse uses and benefits, we turn to describing extensions of OPM computational objects that are related to stereotypes.

4.1. Permissible Value Ranges and Their Validation

In OPM, a value is defined as a state of an attribute. Since an attribute is an object, an attribute value corresponds to an object state. Computational objects can have different values, including numerical and textual ones, but almost invariably, their range is limited. For example, an adult person's weight in kg can be defined to have a range from 40 to 140. Employing features beyond what is defined in ISO 19450:2015, OPM enables expressing and validating such constraints.

Since an attribute is a stateful object, a permissible or legal attribute value is a member of the set of permissible states of that stateful object. The set of permissible states of an OPM computational object (or values of a computational attribute) can be an enumerated list of numbers or character strings, or a set of one or more ranges of numbers or character strings. These constraints on the permissible values of an object class are defined in design time at either the computational object class level or at the stereotype level. Similarly, during runtime, i.e., when the OPM model is executed, the value assigned to a computational object instance of that object class can be validated for compliance with the value ranges defined for that object at the class or stereotype level. The latter level is preferred, as it enforces the same definitions and validations for objects linked to the same stereotype across the entire model, and they cannot be overridden.

Validation requires some software tool in any case, be it OPCloud or any other modelling tool. This is true not only for OPM but also for UML, SysML and any other modelling language. Stereotypes impose additional validations on top of those imposed by the syntax of OPM as a language. In theory, one could do the validations dictated by a stereotype definition manually, but for stereotypes to be of practical use, automated validation by some tool is required.

4.2. Defining Value Range Options

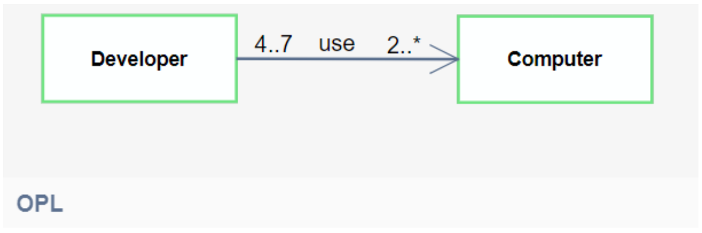
To define a range, we use the OPM symbols in Table 1 that are the same as those used by the standard for OPM links cardinality. Figure 11 is an example of using both source and destination link cardinalities.

Table 1. Computational object value types definition.

Lower & Upper Bounds $q_{min}..q_{max}$	**Abbreviated Symbol**	**Semantics**
0..1	?	One optional thing
0..*	*	Several optional things
1..1	(none)	Exactly one thing
1..*	+	At least one thing

For numerical values, a range is expressed in the following format: [V_{min}..V_{max}] or (V_{min}..V_{max}), where the [and] include the lower and upper boundary values, respectively, while the parentheses, (and), do not include these lower and upper values. The use of brackets and parentheses for lower and upper bounds can be combined. An asterisk, *, marks unlimited lower or upper bound when placed on the left, V_{min}, or right, V_{max}, of the range specification, respectively. For example, the ranges (0..*) and (0..*] express any value greater than 0. If there are N ranges, they shall have the following syntax: [V_{min1}..V_{max1}], [V_{min2}..V_{max2}], ... [V_{minN}..V_{maxN}]. The ranges must not overlap, and

OPCloud checks this at design (modeling) time. One can also combine a range with a nominal (default) value (abbreviated nom) as follows: [V_{min}.. V_{nom}..V_{max}]. Consider, for example, the object **Mountain**, with attribute **Height** [m]. The unit of measurement of **Height** is meters, m, and we set its nominal value to be V_{nom} = **1000**. Hence, since the lowest mountain is 0 m, V_{min}..V_{max} = **0..8900**. With the nominal value, this range is expressed as [V_{min}..V_{nom}..V_{max}) = **[0..1000..8900)**. If **Mountain** is defined as a stereotype, and its model implementation specifies a **Height** of **3450**, resetting **Mountain** in the model will revert its **Height** to its nominal value, **1000**.



4 to 7 Developers use 2 to many Computers.

Figure 11. Source and destination link cardinality example.

If no nominal value is defined, it is set as the average of the minimal and maximal values. If only one value is defined, it is considered the nominal value. If only one boundary is defined, the default is that boundary. For example, in [**0..***], only **0** is defined, so the default is **0**, while for [***..10**], the default is **10**. If the boundary is not included in the range, it will be the value closest to the defined value. Thus, if the range is (**0..***), the default shall be **1** for integer and **0.1** for float, assuming that the number of digits following the decimal points is 1, or **0. 01** if the number of digits following the decimal points is 2. For (***..10**), the default shall be **9** for integer and **9.9** (or **9.99** if the number of digits following the decimal points is 2) for float. If more than one range was defined, it is calculated as the average of V_{min} and V_{max} values of the first range, i.e., the average of V_{min1} and V_{max1}.

Enumerated textual values are marked with quotes, " ". for example, the values **Present** and **Absent** will be recorded as "**Present**" and "**Absent**". The asterisk, *, is a wildcard string. For example, "**Pres***" is any string that starts with "**Pres**". The % symbol is a wildcard for a single character. For example, "**Pr%s**" is any string with four characters that starts with "**Pr**" and ends with "**s**". The default value is denoted by the **$** symbol preceding the default string. For example, ["**Present**", **$**"**Absent**"] means that "**Absent**" is the default value. If the default value is not defined, it is the first value. Only one default value can be defined even for a set with more than one range. Textual ranges are also defined, based on their ASCII values, but not yet implemented.

4.3. OPM Computational Value Types

Five computational value types, listed in Table 2, are defined. For Boolean objects, the value pair true and false can be renamed using many other possibilities, like positive and negative, 0 and 1, non-existent and existent, up and down, black and white, and north and south.

Table 2. Computational object value types: definitions and examples.

Type	Definition	Example
String	Text	"Hello"
Integer	A whole number without decimal point	123 or –1345
Float	A floating-point number, with decimal point	19.99 or –19.99
Char	A character	'a' or 'B'
Boolean	One of the states (values) true or false of an object (attribute) with exactly two states	false

4.4. Designation and Its Values Initial, Final, Default, and Current

At the metamodel level of OPM, object states have an attribute called **Designation**, with values **initial**, **final**, and **default**. An **initial** state is the state at which the object is upon its generation or as the system starts executing. The **final** state is the state at which the object is upon its consumption or as the system finishes executing. The **default** state is the state at which the object is expected to be when its state is not specified. Only one state of an object can be assigned a **final**, **initial** or **default Designation**, but the same state can be in more than one **Designation**. For example, if an object has the states **present** and **absent**, **present** can be both initial and final, but **present** and **absent** cannot both be final.

The three **Designation** values, **initial**, **final**, and **default**, are applicable for design (modeling) time, but for runtime, i.e., during the model execution, a fourth **Designation** value is needed. For run time, we introduce the fourth **Designation** value, **current**, defined as the value at which the object is when inspected at the present time. According to this definition, the **current** value of **Designation** is indeed applicable only at runtime. Since at runtime an existing object is at exactly one state at each point in time. The **Designation** value of that state is **current**. The OPL sentence that specifies the **current Designation** is:

Starting at time <current runtime> s, <Object> is at state <state2>.

For example, in a simulated execution of a car being ignited 3.5 s from the beginning of the execution, the above sentence will become: Starting at time **3.5** s, **Car** is at state **ignited.** If after 4 more seconds the car starts to move, changing its state to **in motion**, the following new sentence shall be created: Starting at time **7.5** s, **Car** is at state **in motion.**

4.5. Meta-Attributes and the Computational Object Type Meta-Attribute

A meta-attribute (also called implicit attribute of property) is an attribute that OPM adds automatically to an OPM element (entity or link). For example, **Designation**, introduced in the previous section, is a meta-attribute of the OPM element **State**. A meta-attribute is thus an attribute that is defined for an OPM element at the OPM metamodel level.

When the modeler defines an object as computational, the meta-attribute **Type** is added to it, as seen in Figure 12. The value of **Type** that the modeler selects is marked by the location symbol, shown in Figure 12 for the value **int**. The default value of **Type** is **int**. The value **current** of **Type** is the one according to which the computational object is validated.

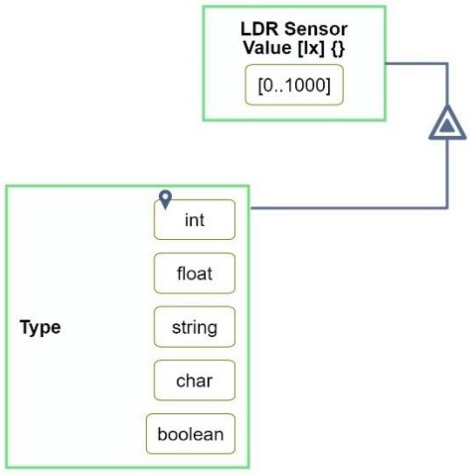


Figure 12. An example of a computational object with the meta-attribute Type.

Type cannot be deleted, but it can be hidden when its exhibitor (e.g., **LDR Sensor Value** in Figure 12) is folded. **Type** can be hidden or shown at will be toggling it with a click of a button, which shows up only when a computational object is selected.

Duration, with default unit s (second), is another example of a meta-attribute; it is an implicit attribute of **Process**: When a process (be it computational or non-computational) is created, OPM automatically adds **Duration** to it as a meta-attribute.

A value range defined in a stereotype cannot be changed in any one of its implementations during design time. While a value range in a stereotyped thing is visible as it is in a non-stereotyped thing, in the stereotyped thing, the set range option is disabled, preventing it from being editable. When the set range button is hovered over, a tooltip shows up, explaining that the range value cannot be changed because it originates from its stereotype definition. The value of a stereotyped object is validated automatically according to the range defined in its stereotype both in modeling time and runtime, as described next.

4.6. The Hard and Soft Value Validation Policies

Value validation is performed for computational objects during both design time and runtime regarding its type, value range, and other optional constrains. There are two value validation policies: hard validation and soft validation. If the first, hard validation policy is adopted, and at design time the modeler inserts an out-of-range value, an error message appears, indicating the expected range. At runtime, reaching an out-of-range value causes the model execution to stop and issue a message that indicates the reason.

In the second, soft validation policy, during both design time and runtime, the system only warns the modeler when an out-of-range value is reached. In design time, a warning message appears and the color of the exceptional state or value changes to red. When hovering over this red, exceptional value, the warning appears in a tooltip. For runtime, this exception is also highlighted in the simulated execution log file. A model's design time validation policy and its runtime validation policy may be different.

The model validation screen enables highlighting in-range, out-of-range, and undefined values with green, red, and blue colors, respectively. OPCloud enables downloading an MS Excel (or csv) file with all the objects, each with its defined range and actual values marked with the above-defined colors.

To help the modeler during both design and runtime, there is an option to load an Excel file with the object values and validate them against their defined ranges in batch mode, without executing the model.

4.7. General Comparison to UML Stereotypes

UML system model are scattered across different diagram types, each with its idiosyncratic set of graphic alphabet symbols and syntax. That division of the system model into multiple views is a major source of difficulty in capturing the system as a whole, understanding its parts, and being able to coherently follow the functionality it performs. If we will take for example the system and stereotypes defined in [25], the system contains more than 10 diagrams, and at least five are parts with stereotype definition and operation in them. A modeler who uses UML to model the system has to remember many different symbols and to associate each symbol with the correct type of diagram.

In contrast, OPM facilitates on a particular subset of things (objects and/or processes), elaborating on their details by refining them to any desired level of detail. The complexity of the entire system is managed by keeping each OPD at a reasonable size and keeping track of the relationships among the various diagrams. In contrast with UML, OPM stereotypes uses a single reference to the stereotype that makes it easier to understand it and the system as whole, enables the modeler to grasp the system structure, behavior and functionality at the same time, and minimizes the likelihood of making modeling errors.

5. A Stereotype Example: The Optimal Light Power Consumption IoT System

We demonstrate the benefits of using stereotypes in an OPM model with an IoT Optimal Light Power Consumption System aimed at maintaining constant room lighting [34,35]. Light-dependent resistor (LDR) sensors in the system measure the room light intensity and send it to a microcontroller, which runs an algorithm to calculate the needed power, which, in turn, is delivered to the light emitting diode (LED).

5.1. Building the Needed Stereotype

Designing a stereotype must be carried out carefully, so modelers and reviewers can be confident that the stereotype is robust and using it is beneficial. In the stereotype **Embedded Device Attribute Set** for representing IoT embedded devices, we want to include global IoT attributes, which any IoT device can potentially have. One such attribute is its geographic **Location**, which is often needed to know where some signal of an IoT sensor, for example, is coming from. To represent **Location**, we created the stereotype **GPS Coordinate Set**, with **Latitude** and **Longitude** as its parts (see Figure 13a). Both **Latitude** and **Longitude** are numerical values, se we define them as computational objects, and since the values are measured in degrees with decimal points, we set their units to be **deg**. We define the value range of **Latitude** to be −90 to 90 degrees (written **[−90..90]**) and that of **Longitude** to be −180 to 180, **[−180..180]**, degrees.

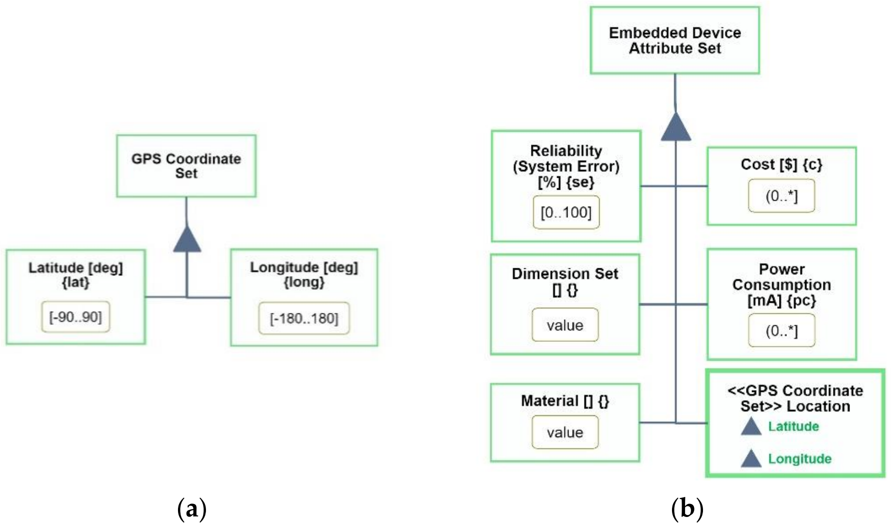


Figure 13. (**a**) The GPS Coordinate Set stereotype (**b**) The Embedded Device Attribute Set model, which serves as a stereotype.

Having defined this base stereotype, we can use it as one of the attributes of the stereotype **Embedded Device Attribute Set**, as shown in Figure 13b. This stereotype consists of **Location**, to which the **GPS Coordinate Set** stereotype is anchored, shown in its semi-folded view. **Embedded Device Attribute Set** has additional five computational attributes: **Cost**, measured in **$**, **Reliability**, measured in **%**, **Material** (a string) and **Power Consumption**, measured in **mA**. **Dimension Set** is similar to the stereotype **GPS Coordinate Set**, as it too bundles the three spatial coordinates, each of which is a computational object. Like **GPS Coordinate Set**, **Dimension Set** could also be defined as a stereotype. Having defined the **GPS Coordinate Set** stereotype, we can anchor it with confidence to the **Location** attribute of the **Embedded Device Attribute Set** without having to spend time and intellectual effort in defining its components (**Latitude** and **Longitude**) and their ranges.

The strings inside curly braces, e.g., **{pc}** for **Power Consumption**, are aliases for use in computational processes. Not all the attributes can have units or value ranges. We may be able to define the cost and power consumption value ranges as greater than zero, but we leave undefined the possible dimensions and materials of our **Embedded Device Attribute Set**.

The next stereotype we define is geared for sensors in IoT systems. **Sensor** is a physical object whose function is the physical process **Sensing**. Since our IoT system and its microcontroller use digitized values and calculation, we define a digital twin process for **Sensing**, called **DT Sensing**. OPM digital twins of their physical counterparts are computational by definition, so **DT Sensing** is computation. As such, it enables defining the specific connection to the hardware and the specific protocol used, e.g., MQTT [2,36–38] or AMQP [2,39].

Since **Sensor** is an embedded device, in Figure 14 we incorporate the **Embedded Device Attribute Set** stereotype into the **Sensor** stereotype, using **Sensor Attribute Set** as the anchor. The result is the attribute **<<Embedded Device Attribute Set>> Sensor Attribute Set**. Using semi-folded object presentation (Figure 14), we can see the parts of this attribute: **Cost**, **Dimension Set**, etc. Additional sensor-specific attributes, including **Resolution** and **Response Time**, have been added, each with a defined value range and measurement units. **Accuracy Set** consists of one or more **Accuracy** objects, each exhibiting the attribute **Accuracy Condition**.

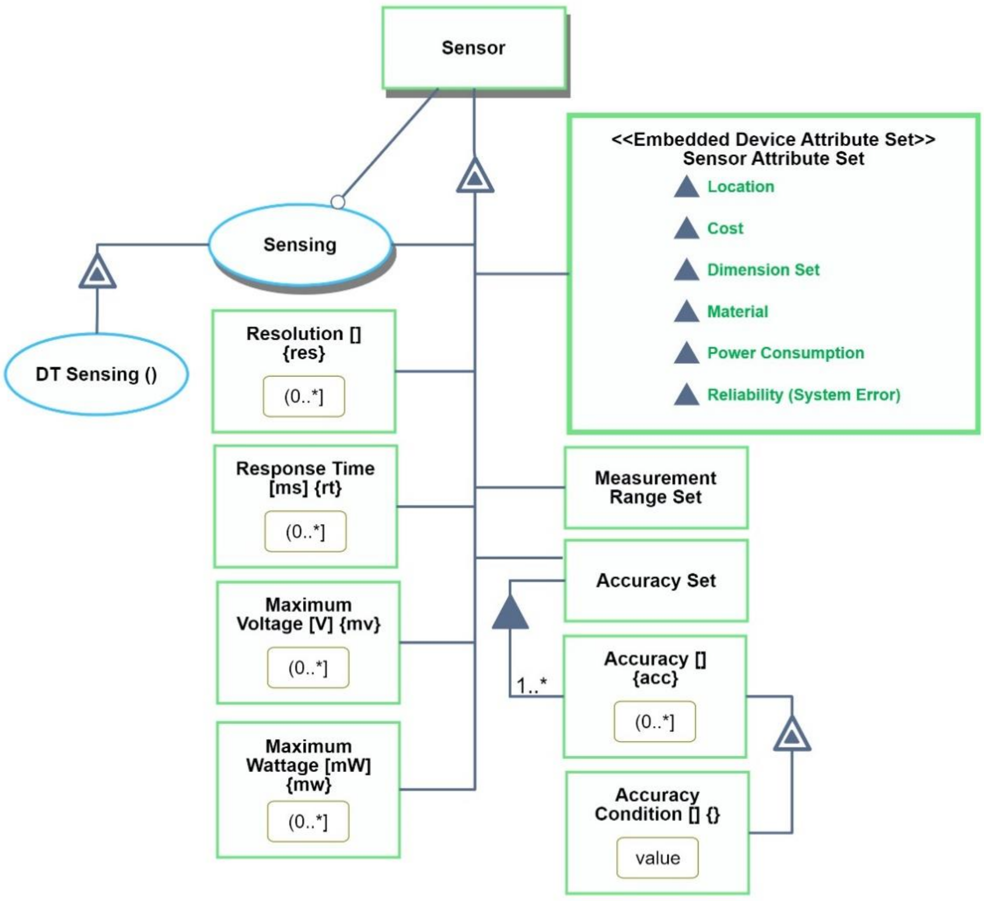


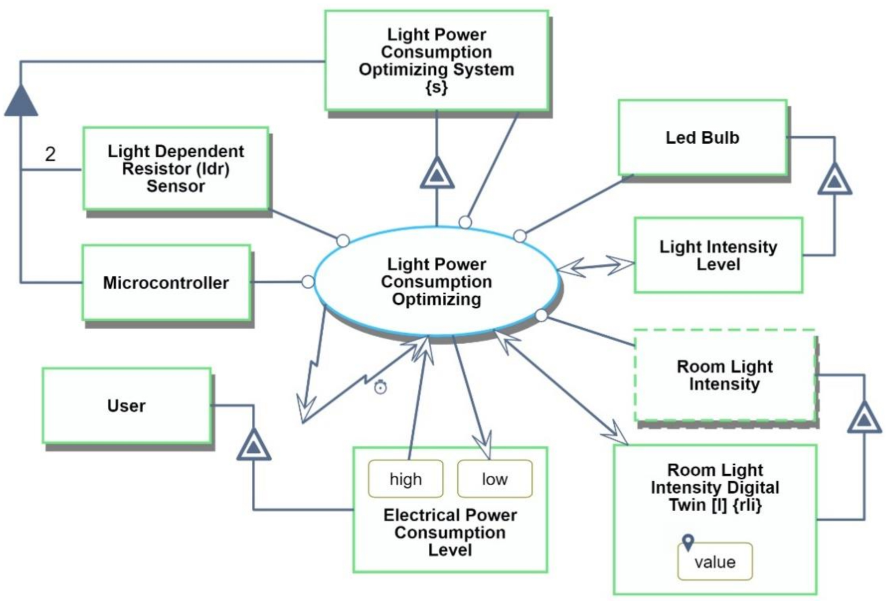
Figure 14. The Sensor stereotype.

5.2. The Optimal Light Power Consumption System—A Conceptual Model

Conceptual modeling is the first stage in creating a complete conceptual-computational OPM mode. At this stage, we define the beneficiary of the system, the system's main function, and its value to the beneficiary. The main function of our system is to handle the system operation with focus on power consumption optimization: **Light Power Consumption Optimizing**. The instrument enabling this process is **Light Power Consumption Optimizing System**. The system, shown in Figure 15, comprises two **Light Dependent Resistor Sensors** and one **Microcontroller**. This is reflected in the following OPL sentence:

Optimal Light Power Consumption System, s, consists of 2 Light Dependent Resistor Sensors and Microcontroller.

The beneficiary of the system is the **User**, whose **Electric Power Consumption Level** changes from **high** to **low**. As the system is completely automatic, the **User** does not need to be involve in its operation. The system is affected by **Room Light Intensity—**a physical and environmental object. In order to represent that our IoT system does not interact directly with the analog signal, we added its digital twin **Room Light Intensity Digital Twin**. The system runs in a loop periodically, e.g., every five seconds, as expressed by the self-invocation of the main process with the clock symbol.


Light Power Consumption Optimizing System {s}
Light Dependent Resistor (ldr) Sensor
2
Microcontroller
User
Light Power Consumption Optimizing
Led Bulb
Light Intensity Level
Room Light Intensity
high
low
Electrical Power Consumption Level
Room Light Intensity Digital Twin [l] {rli}
value

(**a**)

Light Power Consumption Optimizing System, s, is physical.
User is physical.
Electrical Power Consumption Level of User can be high or low.
Led Bulb is physical.
Room Light Intensity is physical and environmental.
Room Light Intensity Digital Twin, rli, of Room Light Intensity is value l.
  Room Light Intensity Digital Twin, rli, of Room Light Intensity is currently at state value l
Microcontroller is physical.
Light Dependent Resistor Sensor is physical.
User exhibits Electrical Power Consumption Level.
Led Bulb exhibits Light Intensity Level.
Room Light Intensity exhibits Room Light Intensity Digital Twin, rli,.
Light Power Consumption Optimizing System, s, exhibits Light Power Consumption Optimizing.
Light Power Consumption Optimizing System, s, consists of 2 Light Dependent Resistor Sensors and Microcontroller.
Light Power Consumption Optimizing of Light Power Consumption Optimizing System, s, is physical.
Light Power Consumption Optimizing of Light Power Consumption Optimizing System, s, changes Electrical Power Consumption Level of User from high to low.
Light Power Consumption Optimizing of Light Power Consumption Optimizing System, s, requires Led Bulb, Light Dependent Resistor Sensor, Light Power Consumption Optimizing System, s,, Microcontroller, and Room Light Intensity.
Light Power Consumption Optimizing of Light Power Consumption Optimizing System, s, affects Light Intensity Level of Led Bulb and Room Light Intensity Digital Twin, rli, of Room Light Intensity.
Light Power Consumption Optimizing of Light Power Consumption Optimizing System, s, invokes itself.

(**b**)

Figure 15. Optimal Light Power Consumption System. (**a**) System Diagram (SD). (**b**) The OPL paragraph of SD.

Having created the system diagram, in Figure 16 we refine it by zooming into its function, the synchronous process **Light Power Consumption Optimizing**, exposing three consecutive subprocesses: **Light Intensity Sensing**, **Power Supply Calculating**, and lastly, **Power Delivering**, each with its instruments and resultees (objects resulting from processes). **Power Supply Calculating** and **Power Delivering** are computational processes. The system is executed initially with simulated values, and when satisfactory results are achieved, it is gradually connected to the actual hardware components for additional testing and optimization, until the system is completely operational.

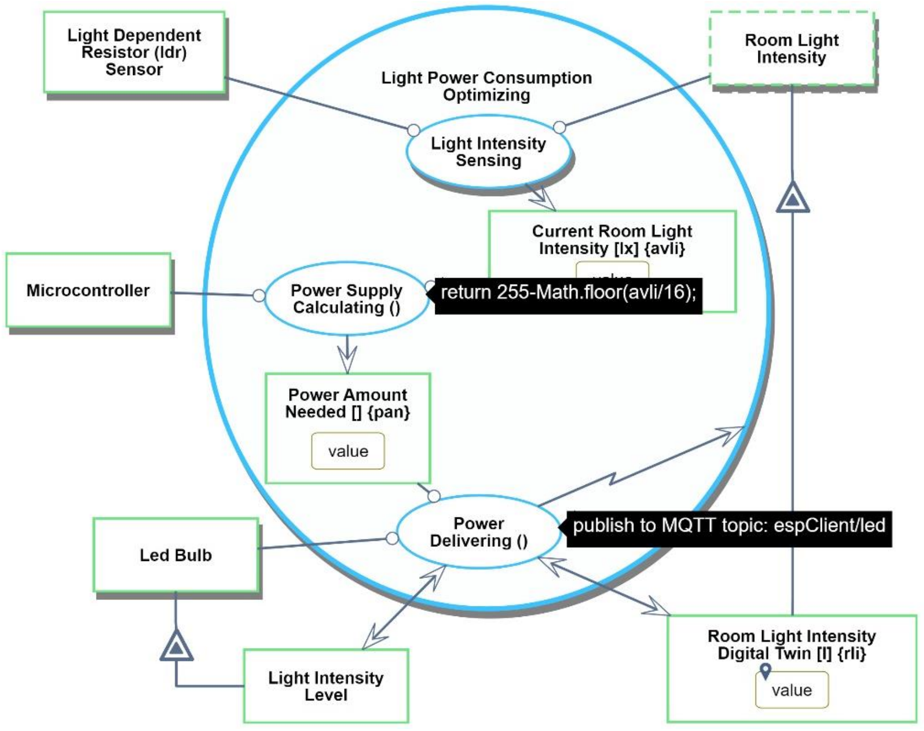

Light Dependent Resistor (ldr) Sensor
Room Light Intensity
Light Power Consumption Optimizing
Light Intensity Sensing
Current Room Light Intensity [lx] {avli}
Microcontroller
Power Supply Calculating ()
return 255-Math.floor(avli/16);
Power Amount Needed [] {pan}
value
Power Delivering ()
publish to MQTT topic: espClient/led
Led Bulb
Light Intensity Level
Room Light Intensity Digital Twin [l] {rli}
value

Figure 16. SD1, in which Light Power Consumption Optimizing is zoomed into.

5.3. Adding the Stereotypes to the Model

We now enhance our model example with the predefined OPM stereotypes. To simplify the example, we consider only the IoT system cost and power consumption. Once some OPD in the model is updated with the stereotyped objects, they automatically propagate to all the model OPDs.

In Figure 17, our system is defined with two **Light Dependent Resistor Sensor** instances: **GI5516 LDR** and **GI5528 LDR**. The system has a total of two sensors, yielding three possible configurations: (1) two instances of **GI5516 LDR**, (2) two instances of **GI5528 LDR**, and (3) a combination of one instance of **GI5516 LDR** and one instance of **GI5528 LDR**. The system can also have two possible microcontroller instances: **ESP32-WROOM-32SE** and **ESP32-SOLO-1**, but only one microcontroller is needed, so in total there are 2 × 3 = 6 configurations.

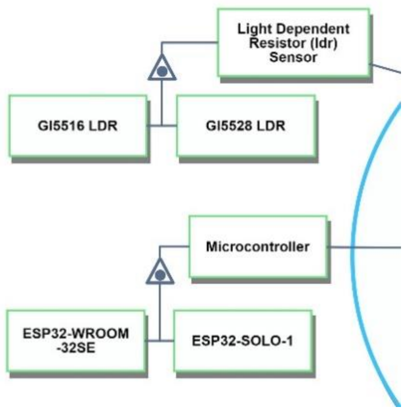


Figure 17. The system IoT devices instances.

In Figure 18, each of the instances is anchored using its corresponding stereotype: Each microcontroller is anchored to its **Embedded Device Attribute Set** stereotype, and the **Sensor** becomes the stereotype for each of the two **Light Dependent Resistor Sensors**.

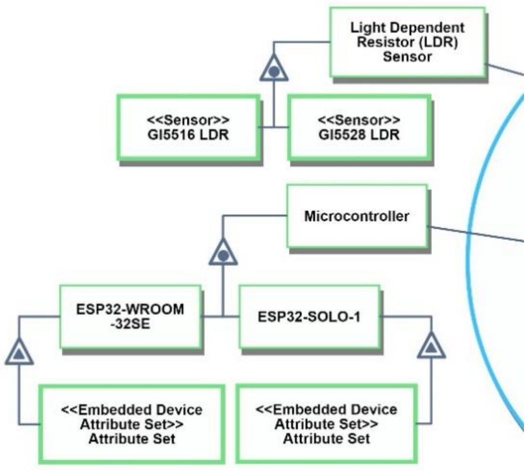

Light Dependent Resistor (LDR) Sensor
<<Sensor>> GI5516 LDR
<<Sensor>> GI5528 LDR
Microcontroller
ESP32-WROOM -32SE
ESP32-SOLO-1
<<Embedded Device Attribute Set>> Attribute Set
<<Embedded Device Attribute Set>> Attribute Set

Figure 18. The system sensor and microcontroller instances with their stereotypes.

With the IoT device instances modeled, in Figure 19 we proceed to define for the **Sensor** stereotype the **Sensing** process and its digital twin, **DT Sensing**, which is executed initially during the simulated model execution and later on during the execution with the embedded hardware, at which point it is no longer a model, but an operational system prototype.

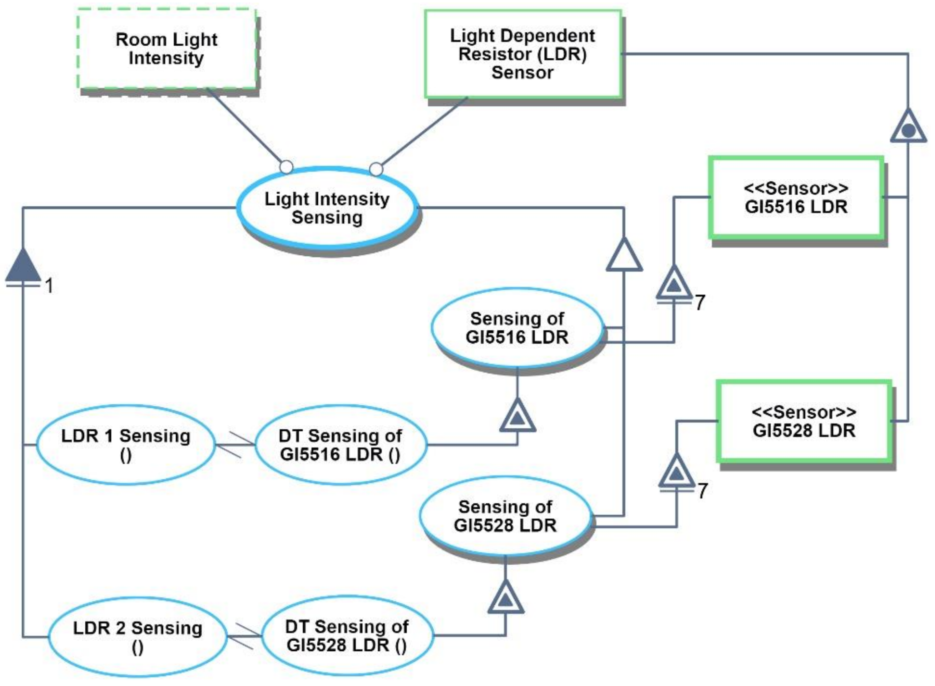


Figure 19. The **Sensing** processes connected to their corresponding **Sensing** stereotype processes.

In Figure 20, the **Light Intensity Sensing** process is zoomed into, exposing the computational stereotyped subprocesses, where the actual data processing takes place. In this case, the MQTT protocol is used for both **LDR 1 Sensing** and **LDR 2 Sensing**, as shown in the black tooltip to the right of each subprocess in Figure 20.

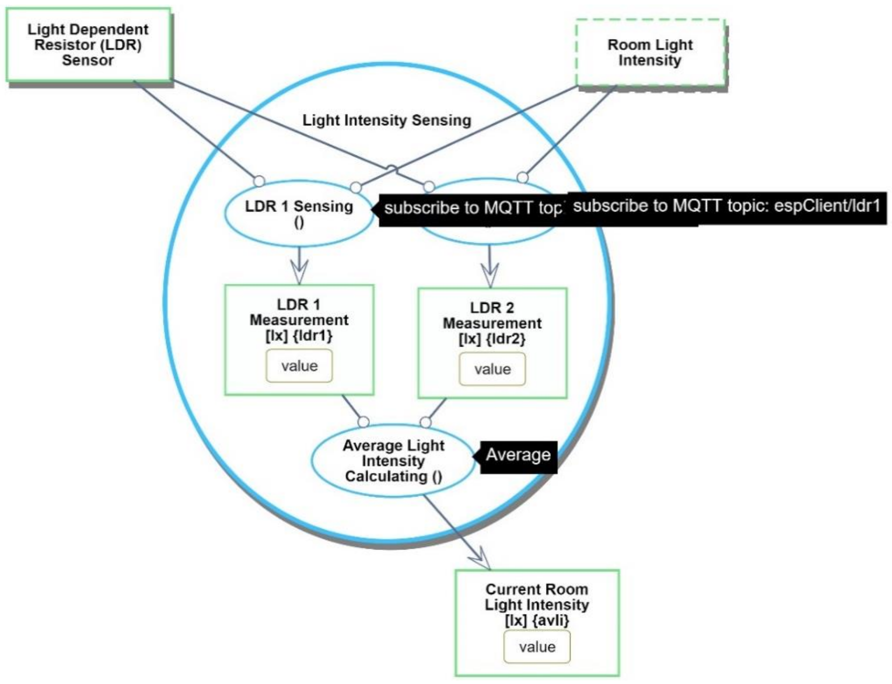

Light Dependent Resistor (LDR) Sensor
Room Light Intensity
Light Intensity Sensing
LDR 1 Sensing ()
subscribe to MQTT top
subscribe to MQTT topic: espClient/ldr1
LDR 1 Measurement [lx] {ldr1}
value
LDR 2 Measurement [lx] {ldr2}
value
Average Light Intensity Calculating ()
Average
Current Room Light Intensity [lx] {avli}
value

Figure 20. SD1.1 inzooming into the Light Intensity Sensting process.

5.4. Using the Value Validation

Having modeled the system with its stereotypes, we can apply the value validation described in Section 4. During modeling time, the value validation kicks in when we specify values for the IoT device instances. Specifying a value for the **Cost** attribute or the **Power Consumption** attribute of an instance of **Light Dependent Resistor (LDR) Sensor** or the **Microcontroller** that is out of the defined range, a warning is issued according to the validation policy defined. In Figure 21, with soft validation as the selected policy, assigning the value 0 for **Cost** and **Power Consumption** causes their value slots to become red, indicating that they are both out of range, because as shown in Figure 13b, none of them can be exactly zero.

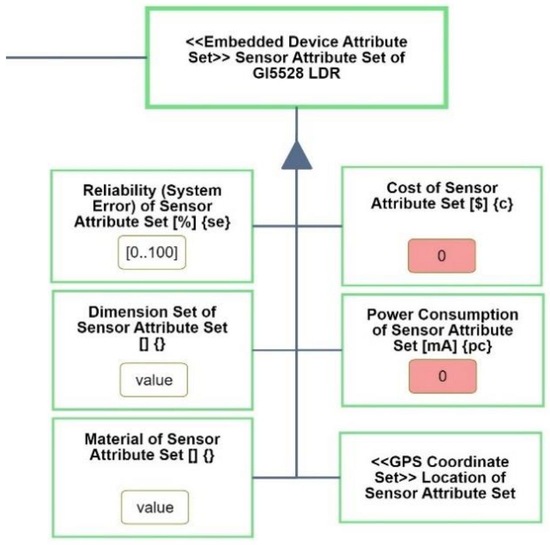

<<Embedded Device Attribute Set>> Sensor Attribute Set of GI5528 LDR
Reliability (System Error) of Sensor Attribute Set [%] {se}
[0..100]
Cost of Sensor Attribute Set [$] {c}
0
Dimension Set of Sensor Attribute Set [] {}
value
Power Consumption of Sensor Attribute Set [mA] {pc}
0
Material of Sensor Attribute Set [] {}
value
<<GPS Coordinate Set>> Location of Sensor Attribute Set

Figure 21. The sensor **GI5528 LDR** attribute set in soft validation marked red.

Since this example does not include any computational process that updates an attribute value at runtime, no runtime value validation needs to be performed.

6. Conclusions and Future Work

Modelers use stereotypes to adapt the language to specific situations or needs, such as hypersonic and suborbital transportation systems in the aerospace domain [40]. Designers of modeling languages, who introduced stereotypes to overcome certain language limitations, have created a well-defined set of general-purpose model elements called extension mechanisms, which allow for customization of the language.

The focus of this paper is to define stereotypes in OPM and demonstrate their deployment in OPM models. We show how OPM stereotypes and their OPCloud implementation can improve the modeling process and enhance the model quality, making it more robust and amenable to validation. We have used the new OPM stereotype feature in an example of an IoT system, where value ranges are defined and validated during the modeling. While the main example in this paper is stereotypes for IoT systems, the use of stereotype is by no means specific to IoT systems. For example, we have shown in Sections 3.3, 3.5 and 5.1 That stereotypes are useful in transportation, GPS, and other kinds of systems.

Aside from improving the modeling process, stereotypes have several other benefits, but using stereotypes also involves potential risks. Stereotypes can help both in modeling time and execution time, providing the modeler with the ability to use properly defined and well-crafted model building blocks. Stereotypes can include semantic and value validations, optimally balancing model formality and understandability. The unique ability of OPM stereotypes to be global or organizational makes OPM especially suitable and appealing for organizations. However, inappropriate use of stereotypes may lead to misuse, causing more harm than good. Excessive use of unsystematic stereotypes, creation of inconsistent and contradicting stereotypes will cause bad models and misunderstanding of the modeled system. Indeed, OPM stereotypes have limitations that stem primarily from misuse: While the technical mechanism has been implemented successfully in OPCloud, creating a stereotype for a specific purpose is challenging; It requires a cognitive effort to design the stereotype so as to make its use in a model correct, effective, and efficient. Therefore, in our OPCloud implementation of stereotypes, only organizational admins can create or edit organizational stereotypes, and only OPCloud admins can create or edit global stereotypes. This is done deliberately to ensure that great care is taken while creating and updating stereotypes, and to prevent abuse of the stereotype mechanism, which would flood the stereotypes library with numerous stereotypes, most of which are superfluous and confusing.

As future work, we plan to extend stereotypes to processes and to multi-OPD models and provide for stereotype inheritance. With stereotype inheritance formally specified and implemented, we will be able to define a hierarchy of stereotypes that inherit attributes from their ancestors. Thus, in our example, instead of defining the stereotype **Embedded Device Attribute Set**, we will be able to define **Embedded Device**, which exhibits **Attribute Set**. Defining **Sensor** as a specialization of **Embedded Device**, **Attribute Set** shall be inherited to **Sensor** and added to its set of sensor-specific attributes. We also plan to enhance the stereotypes to include constraining capability to the modeling process beyond value range validation. For example, if a process is modeled such that two agents are needed to handle it, OPM and its OPCloud modeling tool will be able to enforce this.

To examine the usability aspect of stereotypes, we plan to carry out an experiment with two different student groups. In this experiment, we will assess the impact of using OPM stereotypes on several factors, including modeling rate, model understandability, and model quality. The experimental group will use OPM stereotypes, while the control group will not. The experiment participants of both groups will have basic OPM knowledge. The experimental group will be introduced by the researchers to OPM stereotypes during a two-hour session and will be able to use the stereotype documentation in the user manual. The participants in the experimental and control groups will be provided with OPM models with and without stereotypes, respectively. They will be asked to respond to a set of closed and open questions aimed at assessing the factors being researched.

Author Contributions: Conceptualization, H.K. and D.D.; Methodology, H.K.; Software, H.K.; Validation, H.K.; Formal Analysis, H.K.; Investigation, H.K.; Resources, H.K.; Data Curation, H.K.; Writing-Original Draft Preparation, H.K.; Writing—Review and Editing, H.K. and D.D.; Visualization, H.K.; Supervision, D.D.; Project Administration, H.K. and D.D.; Funding Acquisition, H.K. and D.D. All authors have read and agreed to the published version of the manuscript.

Funding: The research in this paper was partially funded by the Gordon Center for Systems Engineering at the Technion, Israel Institute for Technology. We thank Gordon Center for supporting this research.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: The research in this paper was partially funded by the Gordon Center for Systems Engineering at the Technion, Israel Institute for Technology under grant number 2029853. Authors are thankful for this support.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Yousuf, O.; Mir, R.N. A survey on the Internet of Things security. *Inf. Comput. Secur.* **2019**, *27*, 292–323. [CrossRef]
2. Yassein, M.B.; Shatnawi, M.Q.; Aljwarneh, S.; Al-Hatmi, R. Internet of Things: Survey and open issues of MQTT protocol. In Proceedings of the 2017 International Conference on Engineering & MIS (ICEMIS), Monastir, Tunisia, 8–10 May 2017; IEEE: Piscataway, NJ, USA, 2017; Volume 2018, pp. 1–6.
3. Serpanos, D.; Wolf, M. *Internet-of-Things (IoT) Systems*; Springer International Publishing: Cham, Switzerland, 2018; ISBN 978-3-319-69714-7.
4. Papke, B.L. Enabling design of agile security in the IOT with MBSE. In Proceedings of the 2017 12th System of Systems Engineering Conference (SoSE 2017), Waikoloa, HI, USA, 18–21 June 2017; pp. 1–6. [CrossRef]
5. Lee, J.; Bagheri, B.; Kao, H.-A. A cyber-physical systems architecture for industry 4.0-based manufacturing systems. *Manuf. Lett.* **2015**, *3*, 18–23. [CrossRef]
6. Tannahill, B.K.; Jamshidi, M. System of Systems and Big Data analytics–Bridging the gap. *Comput. Electr. Eng.* **2014**, *40*, 2–15. [CrossRef]
7. Nilsson, R.; Dori, D.; Jayawant, Y.; Petnga, L.; Kohen, H.; Yokell, M. Towards an Ontology for Collaboration in System of Systems Context. *INCOSE Int. Symp.* **2020**, *30*, 666–679. [CrossRef]
8. Ray, P.P. Internet of Robotic Things: Concept, Technologies, and Challenges. *IEEE Access* **2016**, *4*, 9489–9500. [CrossRef]
9. Afanasyev, I.; Mazzara, M.; Chakraborty, S.; Zhuchkov, N.; Maksatbek, A.; Yesildirek, A.; Kassab, M.; Distefano, S. Towards the Internet of Robotic Things: Analysis, Architecture, Components and Challenges. In Proceedings of the 2019 12th International Conference on Developments in eSystems Engineering (DeSE), Kazan, Russia, 7–10 October 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 3–8.
10. Dori, D. Object-Process Methodology for Structure-Behavior Co-Design. In *Handbook of Conceptual Modeling*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 209–258.
11. Ramos, A.L.; Ferreira, J.V.; Barcelo, J. Model-Based Systems Engineering: An Emerging Approach for Modern Systems. *IEEE Trans. Syst. Man, Cybern. Part C* **2012**, *42*, 101–111. [CrossRef]
12. Jensen, J.C.; Chang, D.H.; Lee, E.A. A model-based design methodology for cyber-physical systems. In Proceedings of the 2011 7th International Wireless Communications and Mobile Computing Conference, Istanbul, Turkey, 4–8 July 2011; pp. 1666–1671.
13. Dori, D. *Model-Based Systems Engineering with OPM and SysML*; Springer: New York, NY, USA, 2016; ISBN 9781493932955.
14. Dori, D. *Object-Process Methodology: A Holistic Systems Paradigm*, 1st ed.; Springer: Berlin, Germany, 2002; ISBN 978-3-540-65471-1.
15. ISO. *ISO/PAS 19450:2015—Automation Systems and Integration—Object-Process Methodology*; ISO: Geneva, Switzerland, 2015.
16. Mayer, R.E.; Fiorella, L. 12 Principles for Reducing Extraneous Processing in Multimedia Learning: Coherence, Signaling, Redundancy, Spatial Contiguity, and Temporal Contiguity Principles. In *Cambridge Handbook of Multimedial Learning*; Cambridge University Press: Cambridge, UK, 2014; p. 279.
17. Mayer, R.E.; Moreno, R. Nine ways to reduce cognitive load in multimedia learning. *Educ. Psychol.* **2003**, *38*, 43–52. [CrossRef]
18. Dori, D. Words from pictures for dual-channel processing. *Commun. ACM* **2008**, *51*, 47–52. [CrossRef]
19. Berner, S.; Glinz, M.; Joos, S. A classification of stereotypes for object-oriented modeling languages. *Lect. Notes Comput. Sci.* **1999**, *1723*, 249–264. [CrossRef]
20. Wichmann, A.; Maschotta, R.; Bedini, F.; Zimmermann, A. Model-driven development of UML-based domain-specific languages for system architecture variants. In Proceedings of the 2019 IEEE International Systems Conference (SysCon 2019), Orlando, FL, USA, 8–11 April 2019; pp. 1–8. [CrossRef]

21. Kuzniarz, L.; Staron, M.; Wohlin, C. An empirical study on using stereotypes to improve understanding of UML models. *Progr. Compr. Work. Proc.* **2004**, *12*, 14–23. [CrossRef]
22. Gogolla, M.; Henderson-Sellers, B. Analysis of UML stereotypes within the UML metamodel. *Lect. Notes Comput. Sci.* **2002**, *2460 LNCS*, 84–99. [CrossRef]
23. De Michell, G.; Gupta, R.K. Hardware/software co-design. *Proc. IEEE* **1997**, *85*, 349–365. [CrossRef]
24. Staunstrup, J.; Wolf, W. *Hardware/Software Co-Design: Principles and Practice*; Springer Science & Business Media: Berlin, Germany, 2013.
25. Robles-Ramirez, D.A.; Escamilla-Ambrosio, P.J.; Tryfonas, T. IoTsec: UML extension for internet of things systems security modelling. In Proceedings of the 2017 International Conference on Mechatronics, Electronics and Automotive Engineering (ICMEAE 2017), Cuemavaca, Mexico, 21–24 November 2017; pp. 151–156. [CrossRef]
26. Maschotta, R.; Wichmann, A.; Zimmermann, A.; Gruber, K. Integrated Automotive Requirements Engineering with a SysML-Based Domain-Specific Language. In Proceedings of the 2019 IEEE International Conference on Mechatronics (ICM), Ilmenau, Germany, 18–20 March 2019; Volume 1, pp. 402–409. [CrossRef]
27. The Object Management Group OMG. *Unified Modeling Language Superstructure*; OMG: Needham, MA, USA, 2011; p. 732.
28. Dori, D.; Kohen, H.; Jbara, A.; Wengrowicz, N.; Lavi, R.; Soskin, N.L.; Bernstein, K.; Shani, U. OPCloud: An OPM Integrated Conceptual-Executable Modeling Environment for Industry 4.0. In *Systems Engineering in the Fourth Industrial Revolution*; Wiley: Hoboken, NJ, USA, 2019; pp. 243–271.
29. Model-Based System Engineering OPCloud OPM. Available online: https://www.opcloud.tech/ (accessed on 17 December 2020).
30. Levi-Soskin, N.; Shaoul, R.; Kohen, H.; Jbara, A.; Dori, D. Model-Based Diagnosis with FTTell: Assessing the Potential for Pediatric Failure to Thrive (FTT) During the Perinatal Stage. In Proceedings of the EuroSymposium on Systems Analysis and Design, Gdansk, Poland, 19 September 2019; pp. 37–47.
31. Kohen, H.; Dori, D. Incorporating Hardware-in-the-Loop Simulation into Object-Process Methodology. In Proceedings of the 2020 IEEE International Systems Conference (SysCon), Montreal, QC, Canada, 24 August–20 September 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 1–8.
32. Lavi, R.; Dori, Y.J.; Dori, D. Assessing Novelty and Systems Thinking in Conceptual Models of Technological Systems. *IEEE Trans. Educ.* **2020**. [CrossRef]
33. Aubrecht, K.B.; Dori, Y.J.; Holme, T.A.; Lavi, R.; Matlin, S.A.; Orgill, M.; Skaza-Acosta, H. Graphical Tools for Conceptualizing Systems Thinking in Chemistry Education. *J. Chem. Educ.* **2019**, *96*, 2888–2900. [CrossRef]
34. Salim, G.M.; Ismail, H.; Debnath, N.; Nadya, A. *Optimal Light Power Consumption Using LDR Sensor*; IEEE: Piscataway, NJ, USA, 2016; pp. 144–148.
35. Han, S.; Zhong, X.; Ding, Y.; Li, W.; Liu, S.; Liu, P. Intelligent dimming LED for moonlight simulation. In Proceedings of the 2015 2nd International Conference on Information Science and Control Engineering, ICISC, Shanghai, China, 24–26 April 2015; IEEE: Piscataway, NJ, USA, 2015; pp. 778–782.
36. Hunkeler, U.; Truong, H.L.; Stanford-Clark, A. MQTT-S—A publish/subscribe protocol for wireless sensor networks. In Proceedings of the 3rd IEEE/Create-Net International Conference on Communication System Software and Middleware, COMSWARE, Bangalore, India, 6–10 January 2008; IEEE: Piscataway, NJ, USA, 2008; pp. 791–798.
37. ISO. *ISO/IEC 20922:2016 Information technology—Message Queuing Telemetry Transport (MQTT) v3.1.1.*; ISO: Geneva, Switzerland, 2016.
38. Banks, A.; Briggs, E.; Borgendale, K.R.G. MQTT Version 5.0. *OasisOpen* **2017**, *5*, 137.
39. Naik, N. Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. In Proceedings of the 2017 IEEE International Systems Engineering Symposium (ISSE 2017), Vienna, Austria, 11–13 October 2017; pp. 1–7. [CrossRef]
40. Fusaro, R.; Ferretto, D.; Viola, N. MBSE approach to support and formalize mission alternatives generation and selection processes for hypersonic and suborbital transportation systems. In Proceedings of the 2017 IEEE International Systems Engineering Symposium (ISSE 2017), Vienna, Austria, 11–13 October 2017.