information

Article

Physical Device Compatibility Support for Implementation of IoT Services with Design Once, Provide Anywhere Concept

Mohd Anuaruddin Bin Ahmadon [1,*], Shingo Yamaguchi [1,*], Abd Kadir Mahamad [2] and Sharifah Saon [2]

[1] Graduate School of Sciences and Technology for Innovation, Yamaguchi University, Tokiwadai 2-16-1, Japan
[2] Faculty of Electrical and Electronic Engineering, Universiti Tun Hussein Onn Malaysia, Batu Pahat 86400, Malaysia; kadir@uthm.edu.my (A.K.M.); sharifa@uthm.edu.my (S.S.)
* Correspondence: anuar@yamaguchi-u.ac.jp (M.A.B.A.); shingo@yamaguchi-u.ac.jp (S.Y.)

Abstract: This paper proposes a method to ensure compatibility between physical devices for implementing a service design. The method supports the relaxation of strict implementation. It allows a set of compatible devices to be implemented instead of specifying specific devices in the service design. This paper's main contribution is the formalization of device constraints using the device's attributes and a method to check the compatibility between devices. The method's characteristic is that a service designer can decide the level of strictness and abstractness of the design by adjusting the compatibility rate. We show the feasibility of the proposed method to achieve the goal of "Design Once, Provide Anywhere" with an application example. We also evaluated the quality of service of the implemented IoT service in a different environment using a platform called Elgar.

Keywords: IoT; device compatibility; orchestration; device constraint; service design; SOA; smart object

Citation: Bin Ahmadon, M.A.; Yamaguchi, S.; Mahamad, A.K.; Saon, S. Physical Device Compatibility Support for Implementation of IoT Services with Design Once, Provide Anywhere Concept. *Information* **2021**, *12*, 30. https://doi.org/10.3390/info12010030

Received: 30 November 2020
Accepted: 5 January 2021
Published: 12 January 2021

1. Introduction

The realization of Industry 4.0 [1] and Digital Transformation [2] relies heavily on the Internet of Things (IoT). The IoT has undeniably matured and developed into a more sophisticated and complex technology. It is not just a state-of-the-art concept that leads to the idea of connecting physical objects called 'things' but it is also to bring business and industries to a whole new level. In recent years, physical IoT devices are being manufactured to adapt to the requirements of communication protocols and, in some cases, for a specific environment. Generally, IoT systems are built tailor-made to certain environments and applications. Therefore, support from IoT platforms is required for the implementation of IoT devices. Along with the development of new standards and protocols [3], IoT technologies moved one step further.

Many IoT platforms were introduced to accelerate the development of IoT infrastructures and services. Some of the platforms are cloud-based or premise-based, which adopt popular protocols such as Message Query Telemetry Transport (MQTT) [4] and Constrained Application Protocol (CoAP) [5]. The IoT platform such as Thingsworx [6] and Thingspeak [7] supports the integration of MQTT and CoAP protocols and targets both industries and consumers. Although these platforms support the design and implementation of IoT services, they specifically require devices that are tailor-made to the environment. In other words, the device's specifications, such as data range and data type, must adhere to the requirement or restrictions of the environments. Industrial IoT platforms such as NEC WISE [8] were designed for industries and require specialists from the developer's company to implement. Another flexible IoT platform such as Bosch IoT Suite [9] provides hubs for large scale device communications that focus on message telemetry. MindSphere [10] focuses on the storage of IoT data on the cloud so that they can be analyzed for factual information. However, the end-users, either the industries or

consumers, will have difficulties in connecting suitable devices when restrictions on data type and capabilities are different. Most of these platforms nowadays limit the compatible devices to a certain type of protocols, making the end-users have difficulties deciding on whether the devices satisfy the services' requirements.

This paper proposes a method to ensure compatibility between physical devices for implementing a service design. The method supports the relaxation of strict implementation. It allows a set of compatible devices to be implemented instead of specifying specific devices. This paper contributes to (i) formalization of device constraint using device's attributes; and (ii) a method to check the compatibility between devices. The characteristic of the proposed method is that a service designer can decide the level of strictness of the design by adjusting the tolerance between the subclasses of attributes in the ontology tree and the range of capabilities of each device. We show the feasibility of using the proposed method in order to achieve the goal of "Design Once, Provide Anywhere" (DOPA) with an application example. We also evaluated the quality of service of the implemented IoT service in a different environment using a platform called Elgar.

2. Preliminary

2.1. Service-Oriented-Architecture

Service-Oriented-Architecture (SOA) is regarded as an approach that enables heterogeneous components to connect and scale flexibly. Our research's main goal is to achieve "Design Once, Provide Anywhere" that embraces SOA as the base of the concept. Figure 1 shows the positioning of our research in the SOA's layer. Our research focuses on the process and orchestration layer. We utilize the process model representing the tasks and attributes owned by logical devices and physical devices in the data-control flow model. The process model represents the service design. A logical device can be assigned with any compatible physical devices if they satisfy the constraint between devices described using an ontology model. The study by Niknejad et al. [11] stated the characteristics of SOA in order to enhance service delivery performance of existing conventional systems while preserving essential features. Many design approaches and patterns adopt SOA because it can be defined as an architectural concept that allows loose coupling, reusability, interoperability, agility, and efficiency. It focuses on separating the process into components of tasks and functions such as services. Based on this concept, we propose a model with loose coupling using a Data Petri net, where we specify only the logical devices. On top of the process model, we specify the constraint of the attributes of the compatible devices. Then, we can check the compatibility of the physical devices implemented on the logical devices using the ontology tree. Kohar et al. [12] published a study of utilizing SOA architecture in IoT systems. Therefore, it is well-known that SOA is applicable to IoT systems. SOA is suitable for IoT systems because SOA architectural design allows heterogeneous components to communicate at the lower level and allows loose coupling, enabling scalability. To achieve "Design Once, Provide Anywhere," we also allow heterogeneous devices (physical devices) to be implemented without specifying a specific device in the design. This relaxation method can ease the strict implementation of service design using different devices in different environments. In Ref. [13], Azzedin et al. gave the overview of resource management in P2P systems. Since IoT is similar to P2P systems, which are distributed, there is a need to manage resources. Meanwhile, Washizaki et al. [14] proposed a landscape of architecture and design patterns for IoT systems. From the literature, we can see that the structure that operates IoT systems is essential. While embracing the SOA concept, "Design Once, Provide Anywhere" can be achieved by allowing compatible physical devices to be used. However, service design based on SOA does not focus mainly on the constraint of compatibility.

2.2. Data Petri Net

A Data Petri Net (DPN for short) [15] is a 6-tuple $DPN = (N, V, U, R, W, G)$. It consists of a Petri net [16] $N = (P, T, A)$ (where P, T, A ($\subseteq (P \times T) \cup (T \times P)$) are finite

sets of *places*, *transitions*, and *arcs*, respectively. Let x be a place or transition, a set of data variables V, a function U that defines the values admissible for each variable $v \in V$, a read function R that labels each transition with the set of variables that it reads, a write function W that labels each transition with the set of variables that it writes, a guard function G that denotes a guard with each transition. A transition $t \in T$ is enabled if its guard $G(t)$ returns true. If t has no guard, then $G(t) = true$. We define a subclass of DPN called Data Workflow Net (DWF-net for short). A DPN is a DWF-net if N is a WF-net. A trace of transition bindings' execution sequences of DPN is denoted by a pair (t, φ) where $t \in T$ and φ is value assignment function for some variables $v \in V$, i.e., $v^w = x, v^r = y$, where $v^w = x$ ($v^r = y$) denotes writing (reading) value x (y) to (from) variable $v \in V$. We can write the trace such as $\sigma = \langle t_1\{\varphi_1\}, t_2\{\varphi_2\}, \cdots, t_n\{\varphi_n\}\rangle$. A valid firing of (t, φ) satisfies the state transition rule in Def. 4 of Ref. [15].

Let F be the set of functions $f_k(v_1, v_2, \cdots, v_i)$ for DPN with input $v_i \in R$ and output $w_j \in W$, where ($k = 1, 2, 3...$). L is a labeling function of f_k such that $L : f_k \rightarrow \Sigma$ where Σ assigns to each transition's label in DPN. The transition labels represent the task or function executable by a logical device or physical device. A labeled Data Petri Net is a 2-tuple $DPNL = (DPN, \ell)$, where ℓ is the set of labels for vertex E in an ontology tree $\mathscr{L}$ and all labels ℓ in L correspond to the transitions in T, where $\ell : T \rightarrow L \subset \mathscr{L}$.

2.3. Ontology Tree

An ontology tree $\mathscr{L} = (E, L)$ is a tree with a set of vertex E and a set of label L. Each vertex is a pair of node n and label $\ell \in L$ such that (n, ℓ). In this paper, we use an ontology tree to store the classes and attributes of logical devices or physical devices and their relationship.

3. Related Work

IPSO Smart Object [17] provides a common design pattern and an object model to provide high-level interoperability between Smart Objects such as devices and software applications. IPSO compliant smart objects were defined using a knowledge-based model. In this paper, we extended the ontology to support object definitions such as capability, quantity kind, and unit. The protocol of OMA Lightweight M2M [18] sends messages over MQTT and HTTP. Our method utilizes ontology to describe components or smart objects such as logical devices and physical devices that utilize MQTT and HTTP. The attributes such as resource name, i.e., units and range value of these devices, are similar to the IPSO object. Some objects, such as sensors and actuators, were defined in the guideline of IPSO. In our work, we utilize ontology to define these objects (devices), which follows the guideline by IPSO. Our method's advantage is that we utilize ontology to define the attributes of devices such as quantity kind, unit, and range. Then, we bind the attributes to the tasks defined in the service design, which is the process model. The process model represents the data-control flow of the system. Therefore, in our approach, we have both the process and ontology model that describe the objects and their attributes in the service design.

In 2012, Compton et al. proposed a semantic sensor Ontology model called a Semantic Sensor Network (SSN) [19] as a structured form for linking sensors to the web and enable them to be monitored on any IoT platform. In 2017, Bermudez-Edo et al. proposed a lightweight semantic model for IoT called IoT-Lite [20] based on SSN. The semantic model is highly scalable and suitable for a dynamic environment, such as an IoT system. They showed that IoT-Lite is effective in getting high-performance query-response times. Moreover, it is highly extensible depending on the requirements of the sensor network. International Telecommunication Union (ITU) published technical specifications on a framework to support data interoperability in IoT environments [21]. The technical specification addressed the technology and requirements to enable data interoperability to support semantic and syntactic functions and web-based objects. The specifications roughly stated the models and architecture to support semantic data interoperability. One

of them is SSN models, which can be linked to Web of Object (WoB). Later in 2020, Elsaleh et al. proposed another ontology based on IoT-Lite for data streams in the IoT ecosystem called IoT-Stream [22]. They showed that data stream is usually scarce and deserved to be handled in a more structured form. Usually, in a dynamic system such as IoT, different devices handle and exchange data in various ways depending on the environments' requirements. We know that the semantic model plays an important part in IoT ecosystems based on these related works. However, interoperability cannot be assured by the only good form of semantic models. As the next stage of using a semantic model, we propose a method that utilizes ontology for binding physical devices to service design to implement an IoT service.

Based on Figure 2, our research focuses on the process orchestration. Some related works regarding the process or service composition focus on constraints such as quality-of-service, context, and service tasks. In Ref. [23], Javaid et al. proposed a process model for Web of Things (WoT) called CATSWoTS. The description utilized JSON for defining the WoT components. The context was used as a constraint related to creating a reliable context-aware trustworthy WoT system. Kwei et al. [24] proposed a method using a function graph for orchestration based on the SOA concept. The description used is a suffix tree that sets the quality of service (QoS) as its constraint for creating a process model. Wang et al. [25] proposed a composite service model and utilized Semantic Annotations for WSDL (SWSDL) and consider services as constraints for the composition of service model. In our work, we utilize Data Petri Net (DPN) and ontology tree described in web ontology language (OWL) by taking device attributes as constraints. We also show that our method can be evaluated quantitatively when considering the trade-off of the constraint.

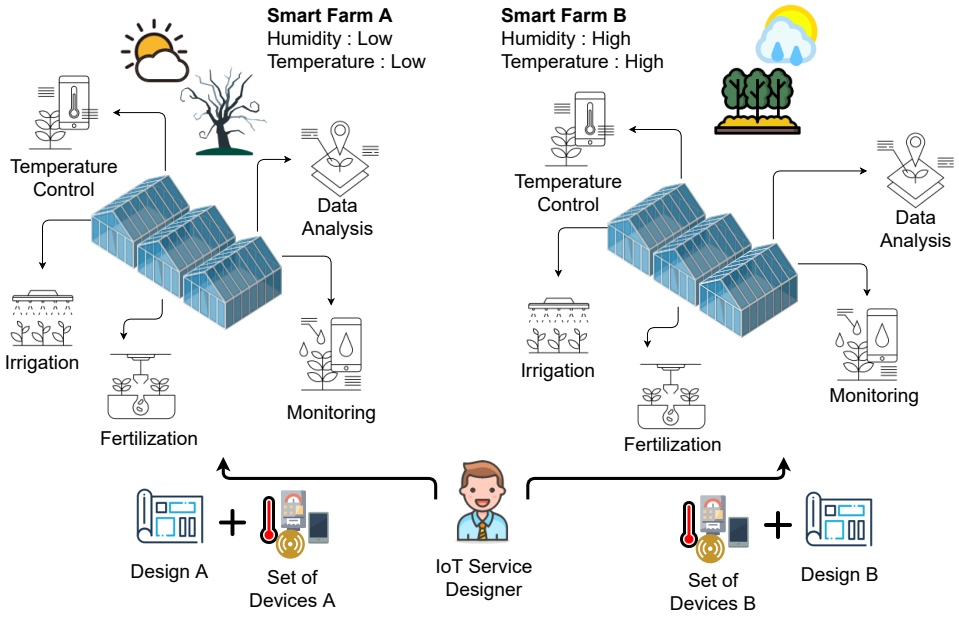

Figure 1. Example of the problem of IoT service implementation in a different environment. Two different environments but with the same IoT service, i.e., a smart farm, require specific devices for each environment due to different capabilities and availability. Therefore, the designer will have to repeat the design process and implementation process for each environment.

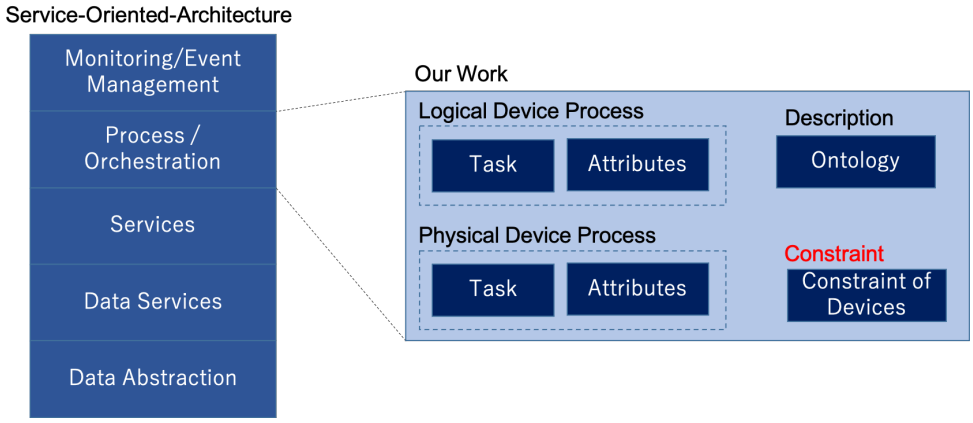

Figure 2. Overview of the positioning of our research.

4. Problem Statement

This paper focuses on designing a single service design and implementing the same design into different environments using different devices. There are two major phases within the scope of this paper. The first phase is the design phase, and the second phase is the implementation phase. We can give the scenario for the two major phases where there is a need to implement the same service in a different environment.

First, we give the definition of a logical device, physical device, attribute, device constraint, and service design. Then, with the given definitions, we define the problem of physical device compatibility.

We give the definition of the attribute. Attributes represent the property of a device that can be used for the compatibility check.

Definition 1 (Attribute). *An attribute α is defined as an entity represented in the vertex $e \in E$ of ontology $\mathscr{L}$. The set of attributes is given as $\mathscr{A} = \{\alpha_1, \alpha_2, \cdots, \alpha_i\}$.*

Next, we define logical device and physical device as follows:

Definition 2 (Physical Device). *A physical device $p = (DPN, L, \mathscr{A})$ is a 3-tuple with DPN, label L assigned for the set of tasks T in DPN and a set of attributes $\mathscr{A}$.*

We define logical device as follows:

Definition 3 (Logical Device). *A logical device $d = (T, L)$ is a 2-tuple with a set of tasks T assigned with label L.*

A logical device is an abstract representation of an actual device where some parts of the device are hidden, such as the control-flow. In this phase, we need to distinguish the physical devices that are compatible or not to implement the service design S_{spec}. Therefore, we restrict the logical device with constraints. In this paper, we focus on the attributes of the device for our restriction. We can formalize the constraint of logical devices using attributes as follows:

Definition 4 (Device Attribute). *A device attribute δ is an n-tuple of attribute α such that $\delta = (c_{\alpha_1}, c_{\alpha_2}, \cdots, c_{\alpha_n})$ where c_{α_n} for task $t_n \in T$ of a logical device.*

Definition 5 (Device Capability). *A device capability r is a range of value between $[a, b]$. The range of capability r is denoted as $r : [a, b]$, where a is the minimum value and b is the maximum value.*

A given device constraint is configured by the service designer depending on the application of the service. Based on this constraint, a physical device can be checked for compatibility during a service design implementation. A logical device with a set of constraints is denoted as $d = (T, L, \delta)$.

Next, we give the definition of service design as follows:

Definition 6 (Service Design). *A service design is a 3-tuple $S_{spec} = (D, V, C, R, U)$, where D is a set of logical devices, a set of data variables V, a set of attributes $C = \{\alpha_1, \alpha_2, \ldots, \alpha_m\}$, a set of device capability R, and U which is the relation of input/output between logical devices and data variables such that $U \subseteq (D \times V) \cup (V \times D)$. A constraint c_m with a pair such that $c_m = (v, r)$ denotes that a data variable $v \in V$ has a constraint with device capability $r \in R$.*

A service designer makes a service design S_{spec} and specifies the set of logical devices D with a set of device constraints $\Sigma = \{\delta_1, \delta_2, \cdots, \delta_n\}$. An end-user implements the service design S_{spec} by assigning the logical devices D with physical devices P that satisfy the set of constraints Σ as shown in Figure 3. We can give the definition of compatible device as follows:

Definition 7 (Compatible Device). *For a given service design, $S_{spec} = (D, V, C, R)$, physical device with attributes $p = (DPN, L_p, \mathscr{A})$ is said to be compatible for logical device $d = (T_d, L_d, C_d)$ if:*

(i) *Physical device p satisfies the constraint C of a logical device d i.e., for all tasks $t \in T_p$ of DPN, the attributes $\alpha_{t_1}, \alpha_{t_2}, \cdots, \alpha_{t_k}$, satisfy the attribute conditions C_d; and*

(ii) *For all relation of input/output between logical devices and data variables such that $U \subseteq (D_I \times V) \cup (V \times D_O)$, where $d_I \in D_I$ and $d_O \in D_O$ are logical devices that output and input variables V, when $d_I \rightarrow p_I$ and $d_O \rightarrow p_O$ the set of capabilities R_I, and R_O satisfies the constraint $R_I \subseteq R_O$*

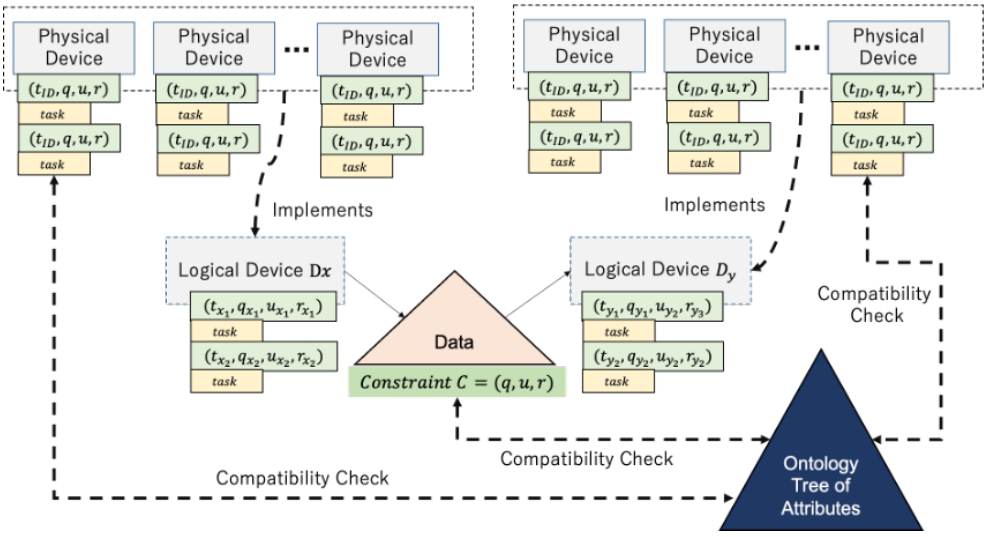

Figure 3. Overview of the proposed method.

To verify whether the devices are implementable or not, we need to check the type of devices and their functions. Here, we formalize the problem of device compatibility.

Problem 1 (Problem of Device Compatibility).
Input: Service Design S_{spec} with logical devices $D = \{d_1, d_2, \cdots, d_n\}$, set of physical devices $P = \{p_1, p_2, \cdots, p_m\}$, set of device assignment $\Sigma = \{\sigma_1, \sigma_2, \cdots, \sigma_k\}$ of physical devices to logical devices such that $\sigma : D \rightarrow P$
Output: Is the set of physical devices P compatible for S_{spec}?

For example, we consider the service design shown in Figure 4, an air conditioning controller for a Heating, ventilation, and Air Conditioning (HVAC) system for a smart home. The HVAC system requires an air thermometer and air conditioner that sends and receives data of indoor temperatures. To decide the type of data handled between the devices, we consider looking at an ontology tree for the related attributes. In this paper, we consider three general types of attributes, i.e., quantity kind, unit, and capability. The ontology tree is shown in Figures 5–7. Quantity kind is the type of data handling, for example, *Temperature* or *Humidity*. Unit is the unit of the quantity kind such as *Degree Celsius*, *Kelvin*, or *Fahrenheit*. Capability is the range between the minimum (*min*) and maximum (*max*) value of the data, distinguishing between measurement and actuation range.

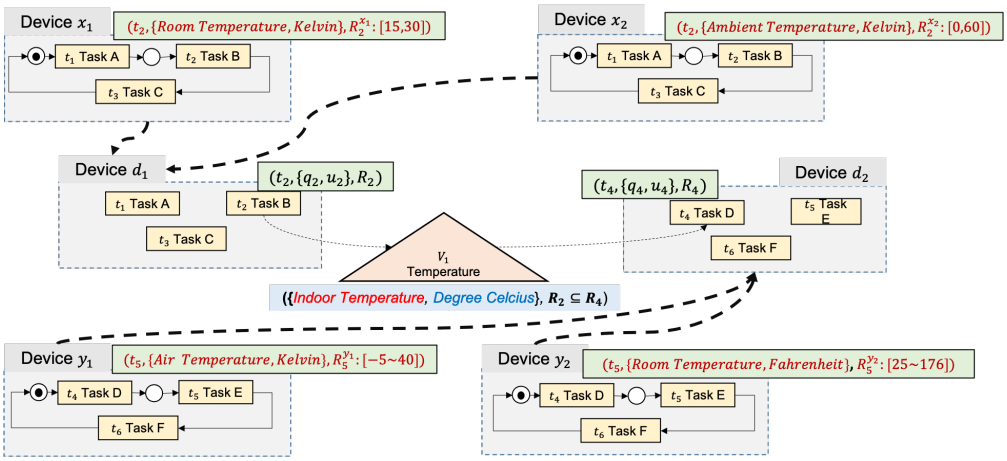

Figure 4. Example of a service design with logical devices and constraints.

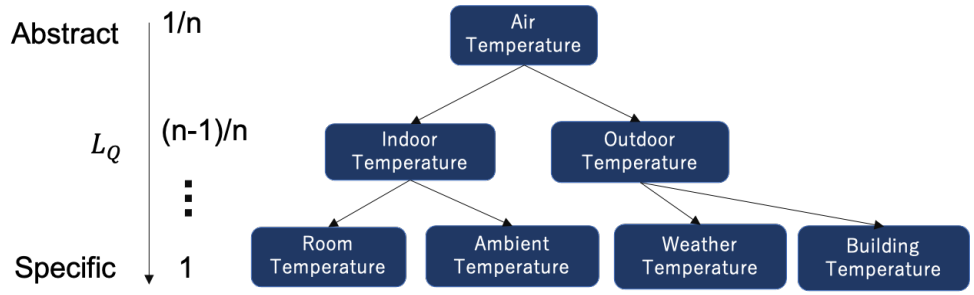

Figure 5. A subtree of quantity kind. The level of compatibility is calculated by calculating the ratio of the quantity class and its level in the ontology tree. The lower the subclass level, the device will be more specific, but the implementation will be stricter. The higher the level, the implementation will be more flexible.

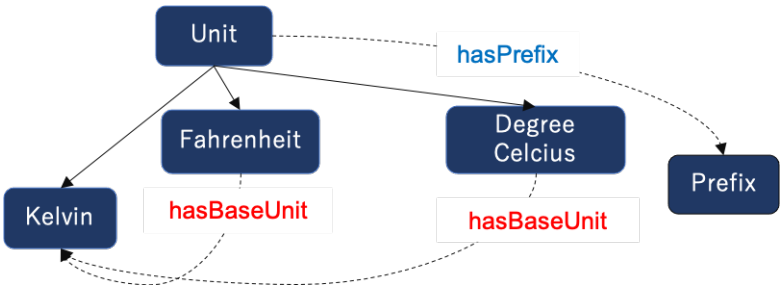

Figure 6. A part of subtree of unit attribute.

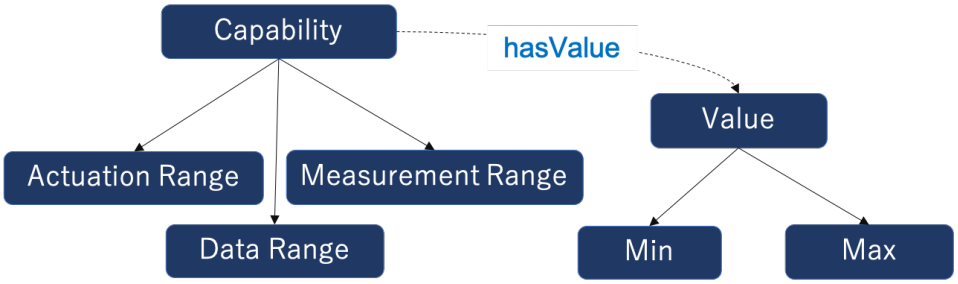

Figure 7. A subtree of capability.

Given two logical devices d_1 , d_2 and data v_1, we need to assign the pair of logical devices d_1, d_2 with physical devices x_1, x_2 and y_1, y_2. Note that d_1 handles v_1 as output and d_2 handles v_1 as input. In the design, first, the service designer will specify the attributes of the data such as quantity kind (q), unit (u), and capability (r). The attributes is attached independently to the task $t \in T$ of the logical and physical devices. V_1 is specified with constraint $(Indoor\ Temperature, Degree\ Celsius, R_2 \subseteq R_4)$. $Indoor\ Temperature, Degree\ Celsius$ are the attribute constraints, and $R_2 \subseteq R_4$ is the capability constraint. We need to check if x_1, x_2, y_1, and y_2 are compatible for implementation into d_1 and d_2 or not.

5. Concept and Approach for Design Once, Provide Anywhere

5.1. Concept Overview

We considered the service design S_{spec} shown in Figure 4. In general, the service design that specifies essential attributes and constraints for an IoT service allows re-usability to a different environment. The overview is shown in Figure 8. We can specify a single design by allowing logical devices to be assigned with a range of physical devices.

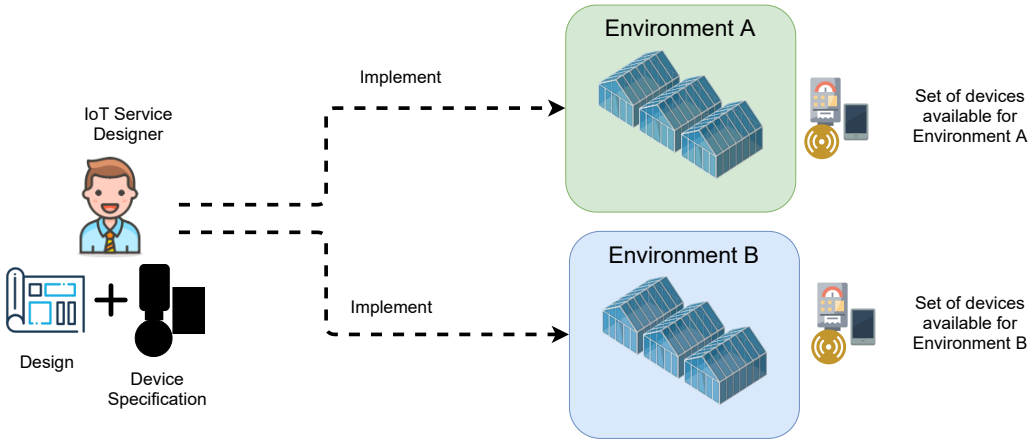

Figure 8. The concept of *Design Once, Provide Anywhere*. The service can be implemented with one service design and only specify the required devices' specifications.

To achieve "Design Once, Provide Anywhere", this method allows compatible devices that satisfy the constraint instead of specifying specific devices. For example, given two logical devices d_1 and d_2 in a service design, what is the set of devices compatible for each d_1 and d_2 that satisfies the constraint of devices with quantity kind, unit, and range attributes represented by (q, u, r)? The attributes can be checked using an ontology tree by verifying the superclass and subclass of the attributes owned by each task.

5.2. Our Approach

In this section, we give our approach. Definition 7 stated that, in Cond. (i), if all physical device attributes satisfy the constraint of attributes C, then the physical device is compatible with the target logical device. In Cond. (ii), for two physical devices, p_I and p_O, the device capabilities must satisfy the constraint R. Figure 3 shows that a device which

is the subclass of a device in the ontology tree is said to provide similar functionality and capability as specified in the design. This is because a subclass under the ontology tree is a child in the device family such that $\alpha_n \subseteq c_n$. In any way, the service designers and device manufacturers should manage the ontology tree so that the service design is valid in general. In Cond. (ii), the logical device's capability needs to match the constraint R.

Next, we give the following procedure to solve the given problem. We check the design S_{spec}. If S_{spec} is implementable, the IoT service can run according to the specification. We give the procedure as follows:

≪Device Compatibility Check≫

Input: Service Design S_{spec} with logical devices $D = \{d_1, d_2, \cdots, d_n\}$, set of physical devices $P = \{p_1, p_2, \cdots, p_m\}$, set of device assignment $\Sigma = \{\sigma_1, \sigma_2, \cdots, \sigma_k\}$ of physical devices to logical devices such that $\sigma : D \rightarrow P$

Output: Is the set of physical devices P compatible for S_{spec}?

1° Check the compatibility based on Definition 7 for all $p \in P$.

- 1-1 Check if the constraint of a logical device i.e., for all tasks $t \in T$ of DPN, the attributes $\alpha_{t_1}, \alpha_{t_2}, \cdots, \alpha_{t_k}$, satisfy the attribute conditions C_d. If no, then, output 'no' and stop.
- 1-2 Check for all relations of input/output between logical devices and data variables such that $U \subseteq (D_I \times V) \cup (V \times D_O)$, where $d_I \in D_I$ and $d_O \in D_O$ are logical devices that output and input variables V, when $d_I \rightarrow p_I$ and $d_O \rightarrow p_O$, the set of capabilities R_I and R_O satisfies the constraint $R_I \subseteq R_O$. If no, then output 'no' and stop.

2° Output 'yes' and stop.

Our approach relaxes the implementation by allowing the devices with similar attributes and capabilities. For example, *Ambient Temperature* and *Room Temperature* are both the subclass of *Air Temperature*, so we implement any Thermometer that can measure *Air Temperature*. In the evaluation, we consider the number of sensors and actuators in the ontology, including the number of its subclasses.

The results of the procedure are shown in Tables 1–3. Table 1 shows that physical devices x_1 and x_2 are compatible for logical device d_1. Table 2 shows that physical device y_2 is compatible for logical device d_2. Finally, Table 3 shows that physical devices x_1 and y_2 are compatible with each other. Therefore, we can use x_1 and y_2 to implement S_{spec}.

Table 1. Compatibility check result for logical device d_1.

Device		Physical Device x_1		Physical Device x_2	
		Room Temperature	**Kelvin**	**Ambient Temperature**	**Kelvin**
Logical Device d_1	**Indoor Temperature**	Yes	-	Yes	-
	Degree Celsius	-	Yes	-	Yes

Table 2. Compatibility check result for logical device d_2.

Device		Physical Device y_1		Physical Device y_2	
		Air Temperature	**Kelvin**	**Room Temperature**	**Fahrenheit**
Logical Device d_2	**Indoor Temperature**	No	-	Yes	-
	Degree Celsius	-	Yes	-	Yes

Table 3. Compatibility check result between input/output of physical devices.

Device		Physical Device x_1			Physical Device x_2		
		Room Temperature	**Kelvin**	**15 ∼ 30**	**Ambient Temperature**	**Kelvin**	**0∼60**
Physical Device y_2	**Room Temperature**	Yes	-	-	No	-	-
	Fahrenheit	-	Yes	-	-	Yes	-
	25∼176 (-5-40)	-	-	Yes	-	-	No

During the design process, the designer can see how strict or abstract the design is by testing it on a set of devices by calculating the compatibility rate. The user can also assign a set of physical devices and evaluate the compatibility using the compatibility rate. The attributes decide the compatibility rate of a device. In the ontology tree, the level of subclass represents the abstractness of the device. For example, for a given physical device p with attribute α of a task t, the compatibility rate when assigned to a logical device d with constraint c_α, where c_α is the constraint of α can be given as L_α, and the constraint which satisfies a true or false statement of satisfying a subset equation can be given as L_β.

For each L_α, we can calculate it based on three types of constraints, i.e., (i) constraint of subclass in the ontology tree, (ii) constraint of ontology class relation, and (iii) constraint of a value range. First, the constraint of subclass in the ontology can be specified by setting a class of an attribute. For example, in Figure 5, if we set $c_\alpha = Indoor\ Temperature$, only the subclasses of *Indoor Temperature* satisfy the constraint C_α. In the ontology tree, let the number of levels in the subtree in which the label of the root is α be n. The level of the attribute of physical device is k where $k = 0, 1, 2, \cdots, (n-1)$. Therefore, the level of compatibility for quantity kind $L_Q = L_\alpha$ is given as follows:

$$L_\alpha = \frac{n-k}{n} \ (k = 0, 1, 2, \cdots, (n-1)) \tag{1}$$

Second, the constraint of class relation can be calculated by checking if the relation is true or not. This can be calculated when α is a true or false question. For example, as shown in Figure 6, if *hasBaseUnit* is true or *hasPrefix* is true, then $L_U = 1$, otherwise $L_U = 0$. Another example is when considering the range of value for capability r. If Cond. (ii) of Definition 7 is satisfied, then $L_R = 1$, otherwise $L_R = 0$. Here, we decide $L_R = L_\beta$ as 0 or 1:

$$L_\beta = = \begin{cases} 1, & \text{if } R_I \subseteq R_O \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

Therefore, we can give the compatibility rate $\mathscr{H}$ for a physical device p_I and p_O that takes a common data V as inputs and outputs with a set of attributes $\alpha_1, \alpha_2, \cdots, \alpha_n$. For constraints $c_{\alpha_1}, c_{\alpha_2}, \cdots, c_{\alpha_n}$, we give the rate of compatibility using L_α and L_β for physical devices p_I and p_O as follows:

$$\mathscr{H}(p_I, p_O) = \frac{\Sigma_{i=1}^{n} L_{\alpha_n}^{p_I} + \Sigma_{i=1}^{m} L_{\alpha_m}^{p_O}}{n+m} \times \prod_{i=1}^{k} L_{\beta_k}^{(p_I, p_O)} \tag{3}$$

For example, we can calculate the compatibility rate of a physical device x_1 and y_2 in service design S_{spec} as with quantity kind $L_Q^{x_1}$ and $L_Q^{y_1}$, unit $L_U^{x_1}$, $L_U^{y_2}$, and the range of capability $L_{R_1}^{(x_1, y_2)}$. We get the compatibility rate for x_1 and y_2 as $\mathscr{H}(x_1, y_2) = \frac{L_Q^{x_1} + L_Q^{y_1}}{2} \times L_U^{x_1} \times L_U^{y_2} \times L_{R_1}^{(x_1, y_2)} = \frac{0.5+0.5}{2} \times 1 \times 1 \times 1 = 0.5$. From this example, the compatibility rate is decided by the level L_α of superclass in the ontology tree.

It also depends on the constraint of L_β of the true or false question which all of the L_β must be true. In the design stage, the compatibility rate can be used when a service designer specifies a service design and tests the abstractness of the service design to a sample group of physical devices. In the implementation stage, the user can use it to confirm the accuracy of implementation when using a certain set of devices. The rate is given from 0 to 1, where a value of 1 indicates the highest compatibility rate.

6. Application Example

We demonstrate our approach to the problem with an application example of a service design and its implementation. Figure 9 shows a service design for an HVAC system. The service requires an air thermometer, humidity sensor, and an HVAC controller. Each logical device is represented as d_1, d_2, and d_3. The constraints for each device and between devices are shown in Figure 9. Figure 10 shows the candidates for each logical device. The physical devices for air thermometer are p_1^x and p_2^x, humidity sensors are p_1^y and p_2^y, and the HVAC controllers are p_1^z and p_2^z.

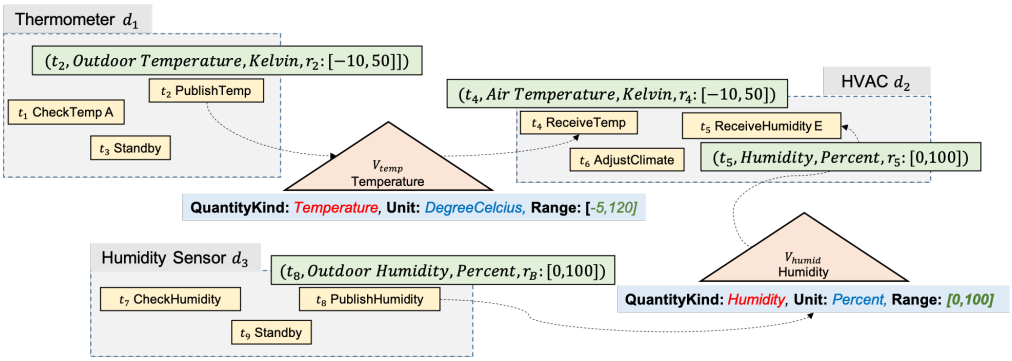

Figure 9. Service Design S_{HVAC} for an HVAC system of a smart farm.

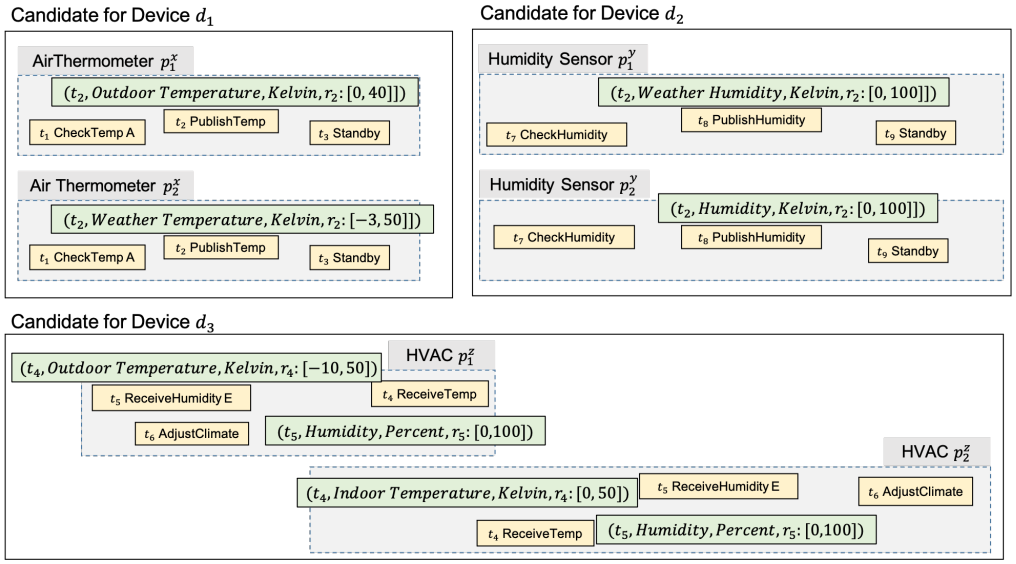

Figure 10. Example group of physical devices for S_{HVAC}.

A service designer can check the design by checking the compatibility rate for a sample group of available devices. The service designer utilizes the ≪Device Compatibility Check≫ procedure to check the compatibility of each physical device. Then, the service designer can evaluate the compatibility rate for the design using Equation (3). The result is shown in Table 4. From the result in Table 4, the service designer can conclude that the service design's overall performance is 0.69, which is quite good for the first version of the

design. The compatibility rate can be improved by increasing or reducing the abstractness of the design. Concretely, the service designer should focus on $L_{\alpha}^{d_2}$ and $L_{\alpha}^{d_3}$, which shows a quite low compatibility rate for the whole group of sample devices. Therefore, the service designer should recheck the abstractness of the humidity sensor and the HVAC controller.

Table 4. Evaluation of compatibility rate for a sample group of physical devices.

Device Assignment			**Device Compatibility**			**Device-to-Device Compatibility**		**Compatibility Rate**
d_1	d_2	d_3	$L_{\alpha}^{d_1}$	$L_{\alpha}^{d_2}$	$L_{\alpha}^{d_3}$	$\mathscr{H}_{(d_1,d_2)}$	$\mathscr{H}_{(d_3,d_2)}$	**Average** $\bar{\mathscr{H}}$
p_1^x	p_1^y	p_1^z	0.50	1.00	0.66	0.75	0.83	0.72
p_1^x	p_2^y	p_1^z	0.50	0.25	0.66	1.00	1.00	0.47
p_1^x	p_1^y	p_2^z	0.50	1.00	0.66	0.75	0.83	0.72
p_1^x	p_2^y	p_2^z	0.50	0.25	0.66	0.38	0.46	0.47
p_2^x	p_1^y	p_1^z	**1.00**	**1.00**	**0.71**	**1.00**	**0.86**	**0.90**
p_2^x	p_2^y	p_1^z	1.00	0.25	0.71	0.63	0.48	0.65
p_2^x	p_1^y	p_2^z	**1.00**	**1.00**	**0.71**	**1.00**	**0.86**	**0.90**
p_2^x	p_2^y	p_2^z	1.00	0.25	0.71	0.63	0.48	0.65
Average Performance			**0.75**	**0.63**	**0.69**	**0.77**	**0.72**	**0.69**

In terms of compatibility between devices, $\mathscr{H}(d_1, d_2)$ and $\mathscr{H}(d_3, d_2)$ show a good performance which achieved an average of 0.77 and 0.72. Finally, between all candidates of physical devices, a combination of devices p_2^x, p_1^y and p_1^z or p_2^x, p_1^y and p_2^z have a compatibility rate of 0.9, which is very good for implementation of the devices. Therefore, a service designer can refer to specific products as a recommendation to implement the design.

A service designer can check the compatibility rate and evaluate the design using a group of candidate devices. The compatibility rate represents the abstractness level of a service design. Therefore, the service designer can consider increasing or decreasing the service design by configuring the device's constraint. There exists a trade-off between the level of strictness and abstractness in specifying the service design. Therefore, a service designer can leverage the trade-off to create a specific design or an abstract design that can support more devices. However, note that a higher abstraction level can lower the accuracy of the device specification. This shows the limitation in determining the level of abstractness of a service design.

7. Implementation of Elgar Platform for Design Once, Provide Anywhere

The concept considers the following three perspectives: [26,27].

- **Designer**: The service designer role is to design the specification of the service. Specification includes a list of required logical devices and services, a description of the service design, and the service's data and control flow model. The designer selects the logical device or service required for the service and designs the data control flow. Register the designed documents in the marketplace and make them available to manufacturers and users.
- **Manufacturer**: The device manufacturer produces Elgar Platform-compatible devices. The compatible devices must be able to communicate with the platform through the Elgar API. In order to manufacture a connectable logical device, an API connecting the logical device and the platform is used.
- **End-User**: The end-user can purchase both service design and devices required for the implementation of the IoT services from the marketplace. Using the Elgar Platform, they can implement the in-a-box solution for the required IoT service. When implementing the service, they download the necessary service design documents

from the marketplace and connect the purchased logical device using the platform. The user can adapt the service by inputting environmental parameters.

We developed a tool from three perspectives revolving around this platform. The first tool is the designer's tool, and the second is the manufacturer's tool, and the third is the end user's tool.

7.1. Designer's Tools

There are three tools that designers use. The first tool is the DPN tool for designing service designs. The designer designs the service behavior with a control flow called Data Petri Net (DPN) using a DPN tool that represents the input or output of data for DPN. The designer can set the environmental parameters in the DPN using data nodes. Environment parameters are entered by the user and set environment variables in the logical device. Figure 11 shows the DPN tool. The DPN tool is saved in the graph display format GraphViz and loaded into the Elgar platform. The devices in the service design are selected from the list of registered devices input by the manufacturer.

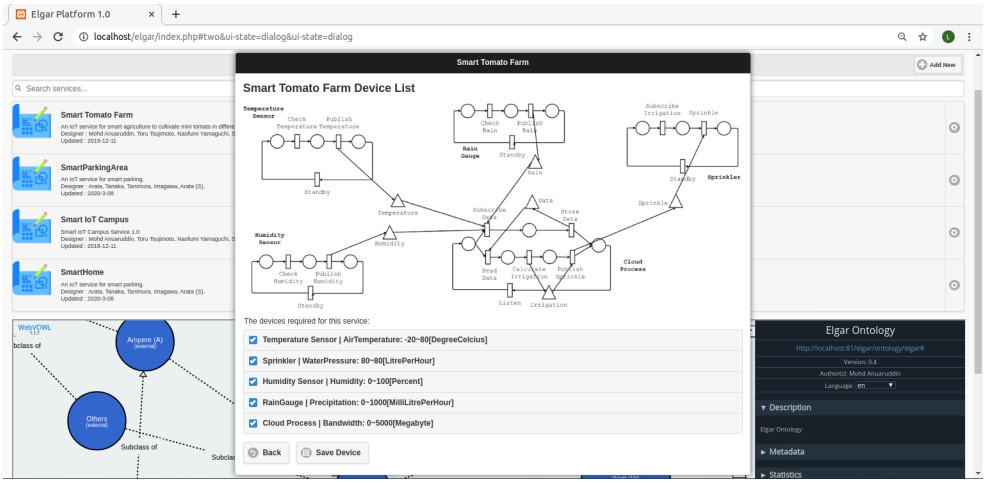

Figure 11. Service Design and service implementation screen.

The description includes the device name, type of resource, quantity kind, unit, and value of measurement range. The sample of Resource Description File (RDF) description for the device list and its requirement is shown in the following listing:

```
<?xml version="1.0" encoding="UTF-8"?>
<service>
    <info>
     <creator xml:lang="en
    ">M.Anuaruddin, T.Tsujimoto, N.Yamaguchi, S.Yamaguchi</creator>
     <date xml:lang="en">2018-12-11</date>
     <description xml:lang="en
    ">An IoT service for smart tomato farm.</description>
     <title xml:lang="en">Smart Tomato Farm Service</title>
     <version xml:lang="en">0.4</version>
    </info>
  <device name="Temperature Sensor">
      <type resource="TemperatureSensor" />
      <hasQuantityKind resource="AirTemperature" />
      <hasUnit resource="DegreeCelsius" />
      <hasValue min="-20" max="200"/>
  </device>
  <device name="Sprinkler">
      <type resource="IrrigationActuator" />
      <hasQuantityKind resource="WaterPressure" />
      <hasUnit resource="LitrePerHour" />
      <hasValue min="0" max="80"/>
  </device>
</service>
```

7.2. Manufacturer's Tool

A manufacturer tool is a tool for making and referencing ontology used by the device manufacturer. The ontology is called Elgar Ontology. Elgar Ontology [27] is displayed in the ontology design tool. Elgar Ontology is based on the IoT-Lite Ontology registered in W3C [28]. The difference from IoT-Lite is that the logical device function *method* that can be expressed by DPN has been expanded. In addition, the entities of *QuantityKind* and *Unit* have been expanded in accordance with this concept. Figure 12 shows the sample ontology of the IoT system with attributes. The devices own their own attributes specified in the ontology subtrees. If logical devices are chosen from the subclass of a superclass, it will be a strict implementation of the device. In the case of a superclass, more physical devices will be able to fit into the class for implementation.

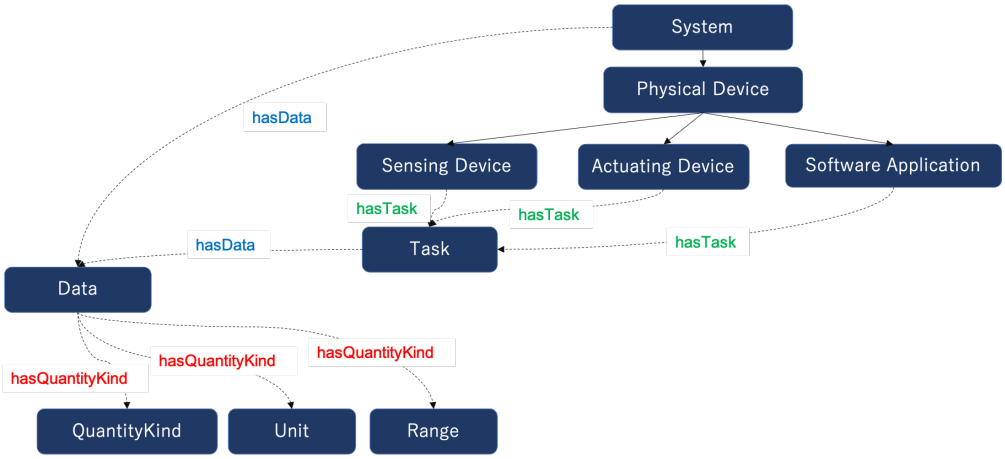

Figure 12. Ontology tree of the IoT system with attributes.

The manufactured devices utilize an API called Elgar API. The API separates protocols from the control flow, where the protocols are wrapped in the API wrapper. Therefore, the manufacturer is free to use any protocols. The API is a library that invokes the basic function and classes of Elgar API. We can configure the variable settings for specific devices such as device name, model, device type, quantity type and unit based on the ontology tree. The address of the broker (central platform server), port number, and time-to-live should also be defined. The rest of the API components contains the executable function of the device which are represented by the tasks specified in the service design S_{spec} configured to the applications. For example, for an irrigation system, *SprinkleWater* specifies the function to execute the sprinkle command based on *Irrigation* value received from the message passing. The arguments of the function such as *SprinkleWater* take the input from a message passing with a certain label, i.e., *Irrigation*. This means that *Irrigation* is received from a different device that processes the value of *Irrigation*. *Irrigation* is specified as input or output of devices in the service design. The default message passing protocol is based on the MQTT protocol implementing the publish–subscribe protocol.

7.3. End-User's Tool

An end-user tool is a tool for registering service design documents. A registration screen is displayed on the platform designer screen. A Resource Description File (RDF) file described in Extensible Markup Language (XML) is registered on this screen. The registered design will be published in the marketplace. The end user's screen is shown in Figure 11.

8. Case Study and Discussion

In this section, we give the case study and discussion on the proposed method. We focus on the following:

(i) Implementation of service in different environments and evaluating the quality of service.
(ii) Evaluating the rate of implementable 'things' for a service design and comparing it to the conventional method.

8.1. Implementation of IoT System with Environment Parameters

We implemented the Smart Tomato Farm service on the prototype platform and applied them in different environments. We created a service instance, "Smart Tomato Farm" from the service design.

For a tomato plant, the amount of water required per day is 3.4 L per plant (area with 0.5 m^2). Since we are targeting 20 plants in an area with 10 m^2, a total of 68 L will be required per day. The devices used in this implementation are a temperature sensor, humidity sensor, single-type sprinkler, and double-type sprinkler. This includes a custom scheduler for controlling the sprinkler based on the required amount of irrigation per day.

As an evaluation, we consider the platform in two smart farm environments located in Japan, as shown in Figure 13. We take Akita Prefecture with low rain volume (shown in Figure 13a,b) and Kagoshima Prefecture with high rain volume (shown in Figure 13c,d). The precipitation and humidity data were obtained from the live weather forecasting service OpenWeather [29]. First, we study the irrigation service of Akita Prefecture. Although rainfall is low in this region, as shown in the graph on the left, the conventional schedule does not consider rainfall and device performance, so the average daily irrigation rate was often above 68 L when it rained. On the other hand, this platform's scheduling method was able to achieve an average of 68 L, although the types of devices and their capabilities are different.

We tested an environment with a lot of precipitation, such as Kagoshima Prefecture. Although rainfall is low in this region, the average daily irrigation rate was often well above 68 L. In this implementation, we use a custom scheduler for the irrigation controller. The traditional scheduler does not consider rainfall and device performance, as shown in Figure 13a,c. On the other hand, after adapting the schedule on this platform, we achieved an average of about 70 L, which are as close as the ideal volume required by the tomato plants as shown in Figure 13b,d. We can see that the water sprinkled value (blue line) does not exceed the average value (yellow line). Therefore, as long as the required minimum irrigation is achieved, the high cost of water can be reduced, thus preserving plants' quality and minimizing the cost.

8.2. Evaluation of the Design Once, Provide Anywhere Concept

We evaluated our approach by showing the effectiveness of using ontology to ease the implementation of devices into service design. In our evaluation, we give the rate of implementable devices using conventional verification and compare the result with the implementation rate of our approach.

We considered a service design that requires 35 actuators, 80 sensors, 18 tag devices, and 20 software and services. We first evaluate the number of devices which is implementable without using our approach. The evaluation is based on the ontology of 120 classes of actuators and 150 classes of sensors, 60 tag devices, and 145 software and services, which in total has 475 classes of devices. We use all 35 actuators and 80 sensors, 18 tag devices, and 20 software/services for the evaluation. Therefore, in the conventional ways where the implementation is strict, the implementation rate of the actuators, sensors, tag devices and software/services are 28%, 53%, 30%, 69%, and 43% each. The implementation rate improved to 56%, 75%, 82%, 68.9% and 69.2%. The results are shown in Figures 14 and 15.

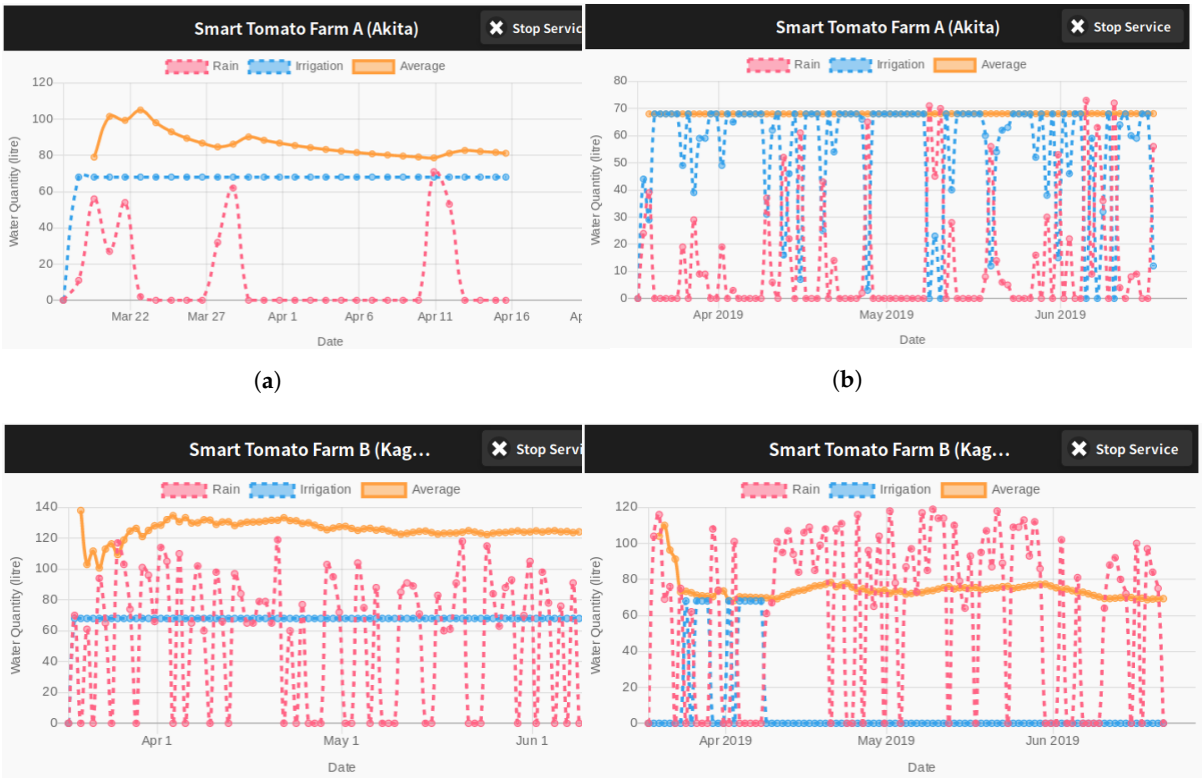

Figure 13. Evaluation on the quality of adaptability for different environments. (**a**) Smart Farm A implemented without Elgar Platform; (**b**) Smart Farm A implemented with Elgar Platform; (**c**) Smart Farm B implemented without Elgar Platform; (**d**) Smart Farm B implemented with Elgar Platform.

Our verification method eases devices' implementation by checking the subclass in the ontology and checking the implementation rate. In the evaluation, we assume that all classes and their subclasses inherit the devices' behavior in the design. Therefore, we only check the number of devices that exist in the ontology tree. The device's behavior is programmed by the manufacturer and made accessible by the platform through the API that wraps the embedded program in the devices.

In the case study, we implemented the smart farm service in different environments and evaluated its quality. We showed that the designed service could adapt to a different environment where devices with different capabilities but in the same subclass are used. This shows that the concept allows adaptability to different environmental parameters. Next, we evaluated the rate of implementable 'things' for a service design and compared it to the conventional method. By utilizing ontology, we can create a knowledge-base for compatible devices. The service designer can decide the implementation based on the problem stated in Problem 1. By carefully selecting the devices for a design, the designer can control the trade-off of implementation rate between the number of devices and type of devices. The implementation rate will increase if the designer selects the higher class, i.e., the main class. However, selecting the most general classes will make it harder to define the design for the specific application. In this case, the designer should consider the scope of the IoT service and carefully review the capability and compatibility during implementation. On top of that, the ease of implementation of the end-user should also be considered.

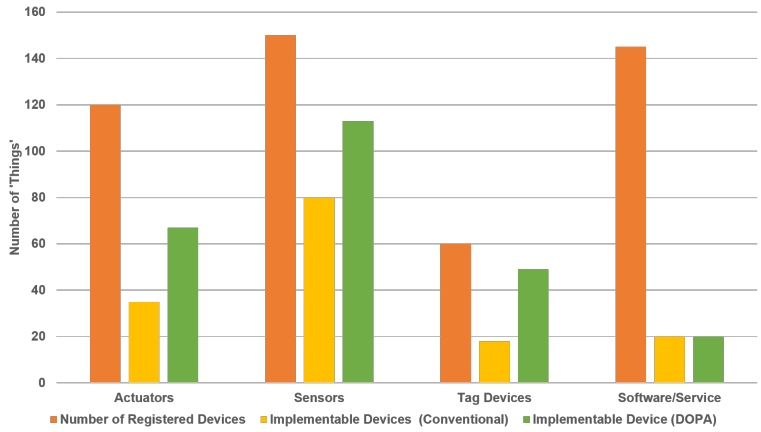

Figure 14. Evaluation for implementable devices.

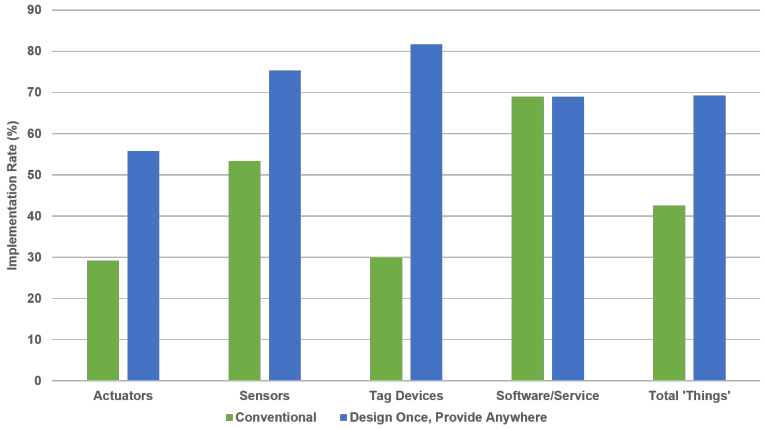

Figure 15. Evaluation for implementation rate.

9. Conclusions

This paper proposes a method to ensure compatibility between physical devices for implementing a service design. The method supports the relaxation of strict implementation. It allows a set of compatible devices to be implemented instead of specifying specific devices in the service design. This paper's main contribution is the formalization of device constraints using the device's attributes and a method to check the compatibility between devices. The method's characteristic is that a service designer can decide the level of strictness and abstractness of the design by adjusting the compatibility rate. We showed the feasibility of the proposed method to achieve the goal of "Design Once, Provide Anywhere" with an application example. We also evaluated the quality of service of the implemented IoT service in a different environment using a platform called Elgar.

We showed that we could increase the implementation rate of devices by adjusting the compatibility rate for a design. We also found that, by abstracting the class of device types, we can increase the number of implementable devices, making the service design more universal. Therefore, by allowing more devices to be used in different deployment environments, the designer does not repeat the design stage. Therefore, this method reduces the effort and time cost required for designing an IoT service.

Author Contributions: Conceptualization, M.A.B.A. and S.Y.; methodology, M.A.B.A., S.Y.; software, M.A.B.A.; validation, M.A.B.A.; formal analysis, M.A.B.A.; investigation, M.A.B.A.; resources, M.A.B.A.; data curation, M.A.B.A.; writing—original draft preparation, M.A.B.A.; writing—review and editing, M.A.B.A., S.Y., A.K.M., S.S.; visualization, M.A.B.A.; supervision, S.Y.; project administration, S.Y.; funding acquisition, S.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This research was partially supported by Interface Corporation, Japan.

Data Availability Statement: The ontology used in the Elgar platform was based on IoT-Lite (https://www.w3.org/Submission/iot-lite/) ontology. The extended ontology file can be accessed from https://github.com/anuaruddin/elgar.

Acknowledgments: The authors would like to thank Interface Corporation, Japan for providing the funding for this research since 2017 and Yamaguchi University's Young Researcher Paper Submission Support Project. In addition, the authors thank Naofumi Yamaguchi and Toru Tsujimoto for their contributions.

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

References

1. Kagermann, H.; Lukas, W.; Wahlster, W. *Germany: Industrie 4.0*; 2017. Available online: https://ec.europa.eu/growth/tools-databases/dem/monitor/sites/default/files/DTM_Industrie%204.0.pdf (accessed on 30 November 2020).
2. Matt, C.; Hess, T.; Benlian, A. Digital Transformation Strategies. *Bus. Inf. Syst. Eng.* **2015**, *57*, 339–343. [CrossRef]
3. Ponnusamy, K.; Rajagopalan, N. Internet of Things: A Survey on IoT Protocol Standards. In *Progress in Advanced Computing and Intelligent Engineering, Advances in Intelligent Systems and Computing*; Springer: Berlin/Heidelberg, Germany, 2017; Volume 564.
4. MQTT. Available online: http://mqtt.org/ (accessed on 30 November 2020).
5. CoAP. Available online: https://coap.technology/ (accessed on 30 November 2020).
6. Thingsworx. Available online: https://www.ptc.com/en/products/iiot (accessed on 30 November 2020).
7. Thingspeak. Available online: https://thingspeak.com/ (accessed on 30 November 2020).
8. NEC WISE. Available online: https://www.nec.com/en/global/solutions/iot/iotplatform/index.html (accessed on 30 November 2020).
9. Bosch IoT Suite. Available online: https://www.bosch-iot-suite.com/ (accessed on 30 November 2020).
10. Mindsphere. Available online: https://new.siemens.com/global/en/products/software/mindsphere.html (accessed on 30 November 2020).
11. Niknejad, N.; Ismail, W.; Ghani, I.; Nazari, B.; Bahari, M.; Hussin, A.R.B.C. Understanding Service-Oriented Architecture (SOA): A systematic literature review and directions for further investigation. *Inf. Syst.* **2020**, *91*, 101491. [CrossRef]
12. Kohar, R. IoT systems based on SOA services: Methodologies, Challenges and Future directions. In Proceedings of the 2020 Fourth International Conference on Computing Methodologies and Communication (ICCMC), Erode, India, 11–13 March 2020; pp. 556–560.
13. Azzedin, F.; Eltoweissy, M.; Khwaja, S. Overview of service oriented architecture for resource management in p2p systems. In *The Handbook of Research on P2P and Grid Systems for Service-Oriented Computing: Models, Methodologies and Applications*; IGI Global: Hershey, PA, USA, 2010.
14. Washizaki, H.; Ogata, S.; Hazeyama, A.; Okubo, T.; Fernandez, E.B.; Yoshioka, N. Landscape of Architecture and Design Patterns for IoT Systems. *IEEE Internet Things J.* **2020**, *7*, 10091–10101. [CrossRef]
15. de Leoni, M.; van der Aalst, W.M.P. Data-aware process mining: discovering decisions in processes using alignments. In Proceedings of the 28th Annual ACM Symposium on Applied Computing, SAC '13, Coimbra, Portugal, 18–22 March 2013; pp. 1454–1461.
16. Murata, T. Petri nets: Properties, analysis and applications. *Proc. IEEE* **1989**, *77*, 541–580. [CrossRef]
17. IPSO Smart Object Guideline. 2017. Available online: https://omaspecworks.org/develop-with-oma-specworks/ipso-smart-objects/guidelines/ (accessed on 30 November 2020).
18. OMA LightweightM2M (LwM2M) Object and Resource Registry. Available online: http://openmobilealliance.org/wp/OMNA/LwM2M/LwM2MRegistry.html (accessed on 30 November 2020).
19. Compton, M.; Barnaghi, P.; Bermudez, L.; García-Castro, R.; Corcho, O.; Cox, S.; Graybeal, J.; Hauswirth, M.; Henson, C.; Herzog, A.; et al. The SSN ontology of the W3C semantic sensor network incubator group. *J. Web Semant.* **2012**, *17*, 25–32. [CrossRef]
20. Bermudez-Edo, M.; Elsaleh, T.; Barnaghi, P. IoT-Lite: a lightweight semantic model for the internet of things and its use with dynamic semantics. *Pers. Ubiquitous Comput.* **2017**, *21*, 475–487. [CrossRef]
21. Chong, I.; Ali, S. *Technical Specification D3.3 Framework to Support Data Interoperability in IoT Environments*; International Telecommunication Union (ITU): Geneva, Switzerland, 2019.
22. Elsaleh, T.; Enshaeifar, S.; Rezvani, R.; Acton, S.T.; Janeiko, V.; Bermudez-Edo, M. IoT-Stream: A Lightweight Ontology for Internet of Things Data Streams and Its Use with Data Analytics and Event Detection Services. *Sensors* **2020**, *20*, 953. [CrossRef] [PubMed]
23. Javaid, S.; Afzal, H.; Arif, F.; Iltaf, N.; Abbas, H.; Iqbal, W. CATSWoTS: Context Aware Trustworthy Social Web of Things System. *Sensors* **2019**, *19*, 3076. [CrossRef] [PubMed]
24. Lin, K.-J.; Zhang, J.; Zhai, Y.; Xu, B. The design and implementation of service process reconfiguration with end-to-end QoS constraints in SOA. *Serv. Oriented Comput. Appl.* **2010**, *4*, 157–168. [CrossRef]

25. Wang, P.; Ding, Z.; Jiang, C.; Zhou, M. Constraint-Aware Approach to Web Service Composition. *IEEE Trans. Syst. Man Cybern. Syst.* **2014**, *44*, 770–784. [CrossRef]
26. Ahmadon, M.A.B.; Yamaguchi, S. On service orchestration of cyber physical system and its verification based on Petri net. In Proceedings of the 2016 IEEE 5th Global Conference on Consumer Electronics, Kyoto, Japan, 11–14 October 2016; pp. 1–4.
27. Ahmadon, M.A.B.; Yamaguchi, S. Ontology-Supported Verification Method for Implementation of IoT Service Design with Petri Net. In Proceedings of the 2018 IEEE 8th International Conference on Consumer Electronics—Berlin (ICCE-Berlin), Berlin, Germany, 2–5 September 2018.
28. World Wide Web Consortium. Available online: https://www.w3.org/ (accessed on 30 November 2020).
29. OpenWeather API. Available online: https://openweathermap.org/ (accessed on 30 November 2020).