NLP-based Generation of Ontological System Descriptions for Composition of Smart Home Devices

Ziyu Zhang
School of Computing Science
University of Glasgow
Glasgow, United Kingdom
ZiyuZhang1998@hotmail.com

Yehia Elkhatib
School of Computing Science
University of Glasgow
Glasgow, United Kingdom
first.lastname@glasgow.ac.uk

Abdessalam Elhabbash
School of Computing and Communications
Lancaster University
Lancaster, United Kingdom
a.elhabbash@lancaster.ac.uk

***Abstract*—With the current rapid development of Internet of Things (IoT) technology and the widespread popularity of smart home devices, wireless technology has made it possible for IoT devices to integrate with each other in a complex system. Previous works have proposed utilizing ontological descriptions, called Holons, of IoT devices and subsystems to reason about the construction of systems. The holonic description, defined by an ontology, includes parameters, services, and properties of the IoT device. However, these previous works assume that Holon descriptions of IoT devices are already provided *e.g.,* by vendors. This assumption requires device vendors and system engineers to manually create descriptions, which is time-consuming and error-prone given the increasing number of IoT devices that are offered in the market. This paper introduces a method for the automatic generation of Holon descriptions of IoT devices. This method uses the brand and model of a device and utilizes knowledge extraction in natural language processing to automatically generate the ontological description of the IoT device. The experimental results show that the proposed method can generate descriptions with a precision of 96.72% and a recall of 87.53% in a practically acceptable time.**

***Index Terms*—System composition, System of systems, Automatic synthesis techniques, Architecture and design**

I. Introduction

With the rapid development of IoT technology and the mainstreaming of *smart life* as a means of creating efficient and comfortable living environments, there has been a continuous increase in the number of deployed IoT devices that share physical spaces. As such, there is a growing need to allow smart home devices to combine opportunistically to form *Systems of Systems* (SoSs), which enables IoT devices to understand each other in terms of services and properties. Such composition is a form of collaborative work between systems to further improve smart home environments in terms of functionality and resilience.

However, IoT vendors constantly produce devices using diverging technology stacks [1]. For this purpose, previous studies proposed the concept of a *Holon* [2], [3] as a self-descriptive semantic representation of a system and its parameters, services, and properties. As a result, IoT devices can exchange and compile *holonic* descriptions based on the CoDAMOS ontology [4] to understand each other. They can also reason for combining efforts (*i.e.,* composing with each other) if certain conditions are satisfied. Other studies (*e.g.,* [5]) have provided specialized architectures that utilize ontological descriptions of systems to facilitate such opportunistic compositions. This provides SoSs with self-aware and self-adaptive capabilities, as they can utilize the descriptions to reason about the evolution of the constituent systems and hence the composite SoS. This allows systems to identify and utilize functionalities offered by other systems, and to reason about composition options based on their analysis. Consequently, SoSs can dynamically adapt to changing conditions and effectively collaborate with other systems to achieve their goals.

However, these studies are all based on the assumption that the Holon descriptions of IoT devices are already provided. In other words, such an approach requires developers to manually generate Holon descriptions in advance, a process that consumes time and human resources and is prone to errors [6].

This motivates us to develop a novel method to automatically generate the Holon description of IoT devices with the aim of improving development efficiency and accuracy. To do this, we exploit device-related information that is readily available on the Internet and extract pertinent information using Natural Language Processing (NLP) techniques. We then use this knowledge to form a holonic description for the device in question in an automated and programmatic manner. This description can then be used to support system discovery and enable spontaneous inter-system composition.

In brief, we first obtain device-related information from the Internet according to known IoT device brands and models. Our method is based on the assumption that it is relatively easy to accurately identify a device's make and model by merely monitoring its DNS network traffic, as reported in recent research [7]. We then use NLP-based knowledge extraction methods to acquire the necessary knowledge (such as device features and capabilities), and use this to automatically generate a Holon description using the Holon ontology structure.

We evaluate our method using 140 different IoT devices from the end-user smart home market. Our approach is able to accurately generate more than 90% of description contents for 134 of these devices, and perfectly and completely generate all description contents for 79 of them.

The main contributions of this study are as follows.

1) An extended Holon ontology structure, particularly for smart home IoT devices, to supplement what has been proposed in the literature (§III).
2) Automated Internet crawlers to obtain detailed specification and capability information about a given IoT device (§IV-A).
3) Methods to extract knowledge about a device from its collected raw specifications (§IV-B).
4) Logic to adapt the collected knowledge and generate the semantic description, *i.e.,* Holon (§IV-C).

II. Background and Related Work

The goal of composing disparate IoT devices into more complex systems involves several research challenges. First, it requires identifying IoT devices and what they do. Second, the ability to extract knowledge about IoT system services is required when a low level of information is available. Third, assuming that the above information is acquired, there is a need for means of composing systems at runtime depending on the context. This section provides an overview of the relevant studies in each of these research domains.

A. Identifying IoT Devices

Thompson et al. [7] introduced a method to identify IoT devices by network data. They trained a neural network using DNS traffic captured from the local network, and were able to identify devices by fitting the model to the first second-worth of DNS traffic after connection. In another study, Ren et al. [8] explored the exposure of IoT device information on the internet by conducting a multidimensional analysis of multiple devices. They showed that the name and manufacturer of an IoT device can be inferred from the owner of the IP address contacted by the IoT device, as well as the type and activity of the device from encrypted traffic patterns and cleartext protocol information.

These works allow us to obtain the brand and model of IoT devices and use them as inputs for our task.

B. Knowledge Extraction in Unstructured Text

Rule-based knowledge extraction is efficient because it only requires a set of rules to be defined and does not require a model to be trained. Bhargavi et al. [9] extracted information from resumes by searching predefined sequences in texts. Miwa et al. [10] constructed a network based on bidirectional sequential and tree-structured LSTM-RNNs, enabling the model to perform both entity- and relation-extraction at the same time.[1] However, these methods have limitations in tasks that have flexible structures and need to extract more than just entities, such as actions.

To avoid restricted templates, Feng et al. [11] proposed a deep reinforcement learning method to extract action sequences from texts by labeling actions and their states as well as their relationship and exclusivity. Sil et al. [12] proposed a structure that extracts action and event semantics from texts while identifying their pre- and post-conditions, thus understanding the relations between actions and events. These state-of-the-art methods achieve efficient action-relation extraction, but are only applicable to declarative texts from the perspective of the object. Text from other perspectives, such as users, causes confusion and affects their efficacy.

Question-answering-based knowledge extraction treats knowledge as the *answer* to a *question* in a given *context*. It only needs to define *question*s and their *answer*s in the *context*, and does not depend on rules according to the format of knowledge. Hence, it is more flexible in knowledge extraction tasks and can handle the knowledge of various structures effectively. Based on the BERT question-answering model, Zhang et al. [13] developed a framework that performs both parameter candidate extraction and parameter role classification, thereby reducing nested entities being missed or incorrectly predicted in entity extraction. Furthermore, Wang et al. [14] applied knowledge extraction in the form of question-answering to the task of insurance clauses.

[1]Entity-relationship extraction aims to extract triplets (*e.g.,* <`Entity1`, `Relation`, `Entity2`>) from texts.

C. Opportunistic Composition of Systems

The concept of SoSs has been studied for over two decades (*cf.* [15]); however, the challenge of composing an SoS from existing systems at runtime is a much more recent research field [16], [17]. Holons were proposed to allow developers to focus on the overall system behavior rather than on the internal details of the system. Both Blair et al. [2] and Frey et al. [3] defined the concept of Holon as an abstract distributed system that is a hierarchical structure composed of various nodes and a representation of their services. Holons can recursively encapsulate other holons to form complex systems, or Systems of Systems (SoSs).

This semantic framework has been used to enable reasoning about opportunistic composition in IoT systems [4]. It also enables the writing of high-level application workflows to be used to identify and realize compositions for making such logic work in dynamic computing clusters [5] and smart home environments [6], [18].

Self-adaptation is an intrinsic quality of SoSs owing to its complex and opportunistic nature. Weyns et al. [19] discussed the challenges of self-adaptation in three SoS structures. Gomes et al. [20] analyzed existing mechanisms for the composition of SoS using system mapping, providing a new perspective for future research. Other studies have focused on improving the system self-adaptation. Sabatucci et al. [21] proposed a method that injects dynamic services and user requirements into a system as targets, enabling a self-organizing approach at runtime to compose and orchestrate services. Kit et al. [22] put forward a component framework to deal with the dynamics of Cyber-Physical Systems (CPS) that can deal with the environment of the systems and its uncertainty.

III. Design

In a smart home environment, composing IoT systems into a more complex SoS involves a number of tasks: (1) identify the brand and model of the device; (2) understand its specification and capabilities in order to create a Holon description for the device; (3) generate an abstract workflow to accomplish high-level logic; (4) use the Holon descriptions to reason about composing the corresponding IoT systems in order to accomplish the workflow's logic. This process is illustrated in Figure 1.

It is quite important to distinguish that the holonic approach is a **bottom-up** approach that aims to *describe*, as opposed to the numerous **top-down** architectures that pursue to *prescribe*, *e.g.,* ThingML [23], smartCityRA [24], and others.

Currently, step (2) is done manually at a significant cost in terms of developer effort [6], [25] and error margins [6], [18]. Our innovation in this study is a mechanism to automate step (2) such that the Holon descriptions for IoT devices are generated programmatically based on the brand and model of the device, as identified by its network traffic profile.

As an overview, we achieve this by carrying out the following process as described by Algorithm 1. In the remainder of this section, we detail the design concepts and sub-processes that constitute our proposal.

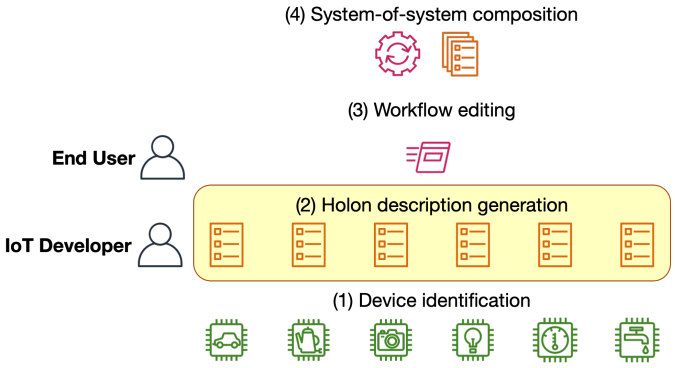

Figure 1: An overview of the steps involved in composing independent IoT systems into a SoS. Our work takes over the task of a developer in step (2) and automates the generation of Holon descriptions that will be used to compose a SoS.

A. Holon Ontology Structure

First, we need to define our Holon ontology structure, as it forms the semantic basis for all subsequent work. To achieve a good coverage of IoT device diversity and the potential for collaborative work of SoSs, we aim to obtain as comprehensive and generic ontology as possible. For example, the specifications of the possible working environment and power supply of a device should be fairly easily deduced, *etc.* Because this is not covered by the original Holon ontology, we expanded it to include additional semantics to cover more aspects of the devices.

The Holon ontology structure is divided into several categories, and in each category, the parameters are related to this category. Some categories are suitable for certain types of IoT devices but not for others. The following is an introduction to the categories, properties of each, and reasons for their inclusion.

General properties. These are the parameters that all IoT devices have.

- id: Contains two parameters `holon_id` and `holon_hash`. `holon_id` is a unique identifier for each Holon description of an IoT device; `holon_hash` is synthesized from combining the device's brand and model, so devices of the same brand and model will have the same `holon_hash`.
- number: The number describes the serial numbers, brand name, manufacturer info, *etc.* of the IoT device as a product.

Algorithm 1 Automatic generation of the Holon description for a device of known brand and model.

Input: <*brand+model*>in String format s
Output: Holon description in JSON d

1: Initialize d in the predefined format
2: Initialize the Holon class object c
3: Crawl from the Internet to get $(para, func\_desc)$ by s
4: Adapt $para$ to d, by which the units are unified and the corresponding values are passed into c by Holon Parser
5: Use knowledge extraction in NLP to get $service$ in the form of the list of Service object from $func\_desc$
6: Pass $service$ to c by Holon Parser
7: Generate Holon description d from c by Holon Creator

- specification: Describes the physical information of the device, such as weight and dimensions. This is useful for inferring device usage scenarios and contexts.
- environment: Describes the ideal environmental conditions for device usage.
- power: Describes the power consumption, such as voltage, wattage and frequency.
- supply: Describes the power supply requirements.
- control: Describes the control method of the device, which is related to the input of the device, such as sound and touch.
- operation: Describes the operation mode, from which we can know how much manpower the device requires. The magnitude and method of human involvement are related to the comfort level of using the device.
- connection: Describes the connection method such as Wi-Fi, Bluetooth, LoRaWAN, *etc.*
- system: Describes the system parameters such as firmware version and hardware instruments.
- compatibility: Describes the compatibility of this device with other system software and hardware. Interfaces or other devices that do not meet compatibility conditions can be ignored directly.
- feature: Describes some functions and uses of the device.

Medium of input. This describes how the IoT device interacts with its environment, from which we can deduce information types and formats.

- camera: Includes the relevant parameters such as resolution, field of view, *etc.*
- tv: Parameters such as port, network service, *etc.*
- sensor: Different types of sensors, such as temperature, humidity, *etc.*
- speaker: Pertinent parameters such as type, woofers, tweeters, *etc.*

Output form. These outline the form in which data is stored or produced by the device. This relates to the following peripherals: video, photo, screen, audio, light, and bulb.[2]

Services. These are the provided services.

- input: The information that can be obtained by or be formed inside the device. This would obviously vary according to the service.
- function: The function of the service, this is typically expressed as an action.
- save: The location where the *input* is stored on the device.
- condition: The conditions for implementing the service.

The Holon ontology described above has a tree-like hierarchical structure. Listing 1 shows the format we expect for a Holon description.

Although a vast amount of information can be obtained from the Internet, only a minute fraction can be used as accurate parameters of the Holon ontology structure. In the current smart home market, "high efficiency" and "high comfort" are two of the most important consumer needs [26]; therefore, we include all data around these two aspects in our scope.

[2]If the light bulb base becomes smart, it can also be considered an IoT device system in its own right and can, thus, be combined with other devices.

```
{
   "id": {
      "holon_id": "the string of holon_id",
      "holon_hash": "the string of holon_hash"
   },
   "number": {
      "brand": "TP-link",
      "model_name": "P100",
      "#": "..."
   },
   "specification": {
      "width": 12,
      "#": "..."
   },
   "#": "...",
   "service": [
      {
         "input": "the input of the device",
         "#": "..."
      }
   ]
}
```

Listing 1: The expected format of a Holon description for an IoT device, expressed in JSON.

B. Collection of Raw Data

Once a device is identified in terms of brand and model, as implemented in recent research works [7], [8], we can obtain related information by crawling the Internet.

1) Crawling: A crawler is a program that simulates web browser activity by sending network requests for web resources (*e.g.,* pages) and receiving responses. Crawler technology can replace the manual search for information on the web, drastically reducing the required effort and time.

We used certain tools and methods to extract the content we needed from HTML documents. The format of the HTML structure is generally the same for websites on the same path with the same domain name.

2) Webpage Selection: To design the crawling method, we had two approaches: obtaining data from the device manufacturer or from retailers that sell the devices. Each approach has its advantages and disadvantages.

Obtaining data from official webpages. The first is to scrape the official website of the device manufacturer. For example, for a Xiaomi speaker, we used the product page dedicated to this device under the `mi.com` domain name. The advantage of this approach is that the information is more reliable. Therefore, this approach is our initial design choice. However, the large variance between the webpage structures of different manufacturers (and even device types from a single manufacturer) meant incredibly time- and energy-consuming efforts to implement crawlers for a handful of device manufacturers. Even so, the coverage is limited by the number of such tailored crawlers implemented.

Obtaining data from e-commerce websites. An alternative is to obtain data from online retailers. The key advantage of this method is that it only requires the design of a single crawler corresponding to the chosen website, which is extremely time-efficient and allows us to focus on perfecting this single crawler. However, this approach has some disadvantages. For example, the data may not be as comprehensive as official websites. Moreover, information could be unavailable if the online retailer does not offer the product for sale (*i.e.,* no longer available due to being superseded by a newer model, being out of stock, *etc.*).

Nevertheless, we decided to use this approach to obtain data from online shopping retailers because of its superior time efficiency. We are cognizant of the potential drawbacks highlighted above and consider them during our evaluation (see §Vfor details).

After careful inspection of different shopping websites (namely Amazon, eBay, Shopee, and Taobao), we found that Amazon covers a much wider range of IoT devices, especially in European and US markets. We also found product information pages to be more comprehensive and provide detailed product specifications. Hence, we designed our crawler for Amazon product webpages.

C. Knowledge Extraction

The device information that is retrieved from Amazon is divided into two parts:

- Parameter: These are key-value pairs presented in tabular format. They include various parameters and device properties.
- Function description: This contains several paragraphs of unstructured text. Often, each paragraph is a description of a function of the device.

For the parameters, we need to match the key-value pairs to the designed Holon ontology structure. If the parameters are described using units, we must specify such units. For the function description, we must extract the corresponding service from each paragraph as unstructured text. NLP methods have been used for this purpose. This process is now described in detail.

Knowledge to extract from unstructured text. Each paragraph is in the form of declarative sentences that describe how functions act and how they can be used from the perspective of the device and/or user. We need to extract the following four elements to populate the Holon description:

- **input**: The information that the IoT device can obtain from the outside world, or form and possess within the device. For example, *motion* is the input form of a motion-sensing smart camera. This is generally a noun in the phrases representing the action of *obtaining* or *detecting*. For example in "detects weather" and "motion detection", "weather" and "motion" are the *input*s, respectively.
- **function**: The function the device can implement is generally a combination of a verb, a noun, and the conditions (if any) required to implement the function, or a phrase that expresses an action. For example, "control your appliances from anywhere at any time" or "set schedules for each of your appliances".
- **save**: The place where the *input* is stored, such as an SD card, the cloud, *etc.* This is usually a noun after a verb that denotes *storing* or *holding* data. For example, in the sentence "store video clips and photos in the cloud", 'cloud' is the *save* parameter.
- **condition**: The condition that the device needs to meet in order to implement a specific function. For instance, in the sentence "the user needs to purchase the X service to enjoy unlimited cloud storage", the 'X service' is the content we need to extract as the *condition*.

Key challenges. Since the intended audience of the text describing the device functions is the retail user, the narrative tends

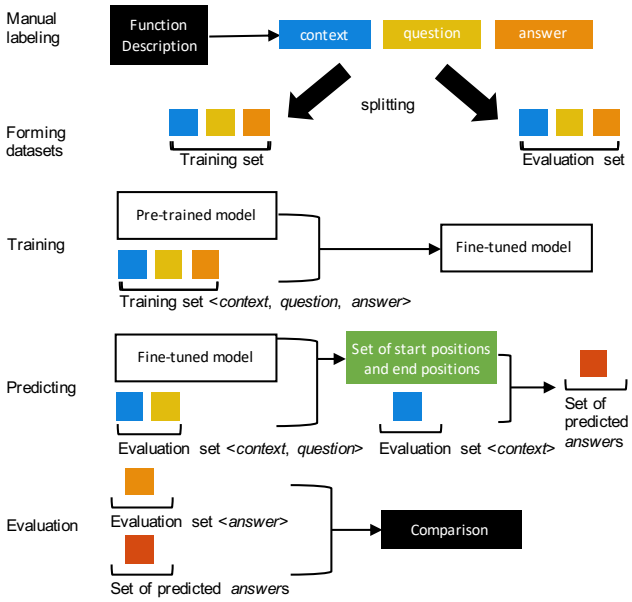

Figure 2: An overview of the process used to extract knowledge from a device's function description.

to be more natural, and the sentence structure is more flexible. Consequently, it is unsuitable to use a set of static rules to extract this information. Moreover, the knowledge structure that needs to be extracted is flexible. For example, we must extract entities to populate the *input* and *save* information elements. However, for the *function*, we must extract phrases that represent actions.

Question-answering Using BERT. The Bidirectional Encoder Representations from Transformers (BERT) model [27], developed by Google, is a transformer-based deep bidirectional encoder representation that is pretrained using plain text corpora. Word embedding is a real-value vector that encodes the meaning of a word. Words closer to this vector space should have similar meanings. Each word in the vocabulary is mapped into a word embedding through the language model. For BERT, word embeddings are related to both sequence and context. For context-free models, such as word2vec, each word in the vocabulary will have the same embedding. However, two identical words in English can have different meanings, depending on the context. For example, the word "bark" in "a loud bark" has a very different meaning from that in "the tree's bark". Thus, BERT can extract more features, express semantic understanding more accurately, and achieve state-of-the-art performance on many natural language understanding tasks, such as question-answering tasks.

In the field of NLP, the method of using pretrained models and learning by fine-tuning models based on custom datasets is widely used. BERT models have fine-tuning flexibility that can be fine-tuned for different question-answering tasks to adapt to different datasets and requirements, and they perform well for tasks that require answers in the given context. Therefore, we selected a question-answering model based on BERT and used a custom dataset as a downstream task to fine-tune it to meet the requirements of our knowledge-extraction task.

Basic Method. First, the text is manually labeled in the form of <`context, question, answer`> to form a *question* and *answer* dataset. Among them, the *answer* is a segment of the corresponding *context* according to the *question*, so it is also necessary to indicate its offset in the *context*. Then the dataset is manually labeled to fine-tune the BERT pre-trained model.

Manual labeling is one of the most important tasks in our work, because we need to find a large amount of data (*i.e.,* information about IoT device products) that fits our application scenarios and manually label within the four knowledge points mentioned above (*i.e., input*, *function*, *save*, and *condition*). However, the knowledge points in paragraphs are not always necessarily obvious, and thus cannot be immediately identified and marked by us. Therefore, we also set up a set of labeling rules, typically judging whether a word or phrase is the knowledge point we need through the sentence structure. For example, the *input* is generally a noun in the phrases representing *obtaining* or *receiving*, like "obtain X", "create X", "detect X" and even "get X" and "see X", or some phrases like "X detection", "X sensor", where 'X' is the *input* we need. And for the *save* parameter, it is usually a noun after a verb which denotes *storing* or *holding* data. Also, a *function* is often a structure of <verb, noun, conditions (if any)>that implements a function, or just a phrase with verbs as nouns that represents an action like "motion detection".

Through these specifications, our manual annotation achieves higher efficiency. The labeling process is roughly shown in the following example: "Supports local video storage for up to 10 Outdoor, Indoor, Video Doorbell and Mini devices. Record and store motion clips when you insert a USB flash drive (up to 256 GB – sold separately) into the Blink Sync Module 2." where the pink, blue, yellow and green parts represent the *input*, *function*, *save* and *condition* knowledge elements, respectively.

After the manual labeling, the resulting model can be used to extract the information we need in our application scenario. For model testing and application, we only need to input <`context, question`> pairs to the model, while the predicted *answer*s (in the form of the start and end positions in the *context*) is the knowledge we need to extract. This process is depicted in Figure 2.

D. Authoring Data

Finally, the extracted knowledge is parsed into the Holon class and Holon description, which can be communicated to and used by other Holons. The Holon description can be in JSON, XML, or other formats that describe a Holon ontology. We chose the JSON format because the hierarchical Holon ontology structure can be clearly and intuitively displayed for results and evaluation purposes. In addition, it is relatively convenient to convert the Holon description from JSON to other formats if necessary.

IV. Implementation

So far, we have designed a process for the automatic generation of a Holon description of IoT devices. This section details our implementation of this design.

A. Crawling

We assume that the information is the brand and model of the IoT device. Our task was to crawl the Amazon website to obtain information about the device. For this purpose, we implemented two layers of crawlers: an outer crawler and an inner crawler. The outer crawler uses the brand and model to query the Amazon homepage and retrieve all the related products. The inner crawler

accesses the contents retrieved by its outer counterpart to collect product specifications. This is analogous to a typical user access pattern when a user uses a browser to obtain the required information, although our process is more methodical and comprehensive.

The outer crawler. We set the outer crawler to issue HTTP queries, specifying the product brand and model as query parameters and used the POST method to access the URL. The returned result is an Amazon retrieval page consisting of a list of search results of links to pages describing related devices. Each link contains an ASIN code, or the Amazon Standard Identification Number, which is a unique code given to each product on the Amazon website. Essentially, the outer crawler returns a list of ASINs.

The inner crawler. The inner crawler parses the results retrieved by the outer crawler and accesses the product page that best matches the search query. Matching is performed based on the device brand and model information as well as page access on ASIN and basic URL splicing. The crawler then examines the webpage content to obtain the raw product data.

B. Knowledge Extraction

From the raw data, we used question-answering-based methods to extract the information needed for the Holon ontology structure.

The selection of a pre-trained model. The *answer*s to the *question*s are used as the results to generate the service of Holon description. We chose to fine-tune the pre-trained model to generate a model suitable for the knowledge extraction scenario in the function description section.

For the initial pre-trained question-answering model, we chose RoBERTa-base-Squad2[3] by deepset. This is a pretrained model using the Robustly optimized BERT approach (RoBERTa) [28] which is also fine-tuned using the Stanford Question Answering Dataset (SQuAD.0) [29]. RoBERTa uses dynamic masking and other techniques to improve the downstream task performance of BERT. SQuAD2 is an English reading comprehension dataset consisting of a set of Wikipedia articles and questions about these articles. Squad2 intentionally mixes answerable questions with similar-looking unanswerable questions to test the ability of NLP models to answer reading comprehension questions and identify those that cannot be answered. Since the articles in the SQuAD dataset involve a wide range of topics and questions, and the RoBERTa-base-Squad2 model has achieved good performance on this dataset, it can be inferred that the model has a strong ability to recognize sentence structure and grammar features in English, and can be used to fit in our dataset and adapt to our application scenario.

Dataset for fine-tuning. We then set up a custom dataset for fine-tuning and testing the model. The data format should be <`context`, `question`, `answer`>, and we need to label each element in the training dataset manually. The data is split 80%-20% between training and testing.

Since the function description consists of several paragraphs, each of which usually describing one of the functions of the IoT device, we divide the function description into paragraphs and use each paragraph as a separate *context*.

Next is the setting for the *question*s. For any *context*, we need to extract the required knowledge (if it exists in the paragraph) to generate services. Given that the service includes *input*, *function*, *save* and *condition*, it is necessary to generate *question*s according to these four knowledge points respectively and find the *answer*s corresponding to the *question*s in the *context*.

We carried out a series of comparative analyses by listing multiple *question*s corresponding to obtain each of the four knowledge points. We used these *question*s in the dataset to conduct a whole-set process of training and evaluating the model, and wanted to know which set of questions eventually performed better. We measured the performance in terms of average accuracy, precision, recall, and F1 score (these metrics will be introduced in detail in §V-B) for each question corresponding to different knowledge points on the evaluation set after model training. The results are summarized in Table I. The performance of the different questions is, on the whole, relatively similar. However, some stand out above others. These are highlighted in bold font for each knowledge point.

Finally, we selected the set of *question*s that had the relatively best performance on the evaluation set. These *question*s are as follows:

- 'What can be got?'
- 'What is the action?'
- 'Where is the input saved?'
- 'What else is necessary?'

As long as the list of *question*s is defined, the *answer*s to them can be labeled in the *context*. For a paragraph as a *context*, the corresponding *answer* and the offset are marked in the *context* according to each *question*.

In terms of data collection, we choose to collect data according to the type of smart home devices including cameras, smart speakers, sensors (*e.g.,* temperature, humidity, motion sensors), smart TVs, smart bulbs, smart cookers, *etc.* As the services of smart home devices vary according to their type, collecting data by type helps to extract the features of the *answer*s of different types of devices, and also allows the trained model to be used in a wider range of IoT devices.

C. Data Parsing and Holon Creation

Once all the required IoT device information is obtained, it can be used in the creation of the Holon descriptions. We defined the data structure of the Holon class and created two units of logic: the Holon Parser and Holon Creator. The Holon Parser generates Holon class objects from the information that we obtained, whereas the Holon Creator uses Holon class objects to construct Holon descriptions in JSON format. The process is illustrated by an example in Figure 3.

1) Holon Parser: The Holon Parser implements three functions, **Collate**, **Adapt** and **Pass**.

Collate information. This function collates the acquired information and stores them in JSON format. The function first initializes a blank Holon description in JSON format and then adapts it with the data obtained by the crawler. The key name settings of the product parameters on the Amazon website are relatively uniform, but there are cases where the content of multiple keys is repeated. For this, we set up a dictionary that ignores duplicate keys.

Adapt the Holon description. This function adapts the Holon description in JSON format to the Holon class object, and unifies the units of parameters. Since the data type of the

[3]https://huggingface.co/deepset/roberta-base-squad2

Table I: The performance of questions in relation to each knowledge point.

Knowledge Point	Question	Average Metrics			
		Accuracy	*Precision*	*Recall*	*F1*
input	"What can be obtained?"	40.56%	20.01%	22.58%	19.64%
	"What is the input?"	40.86%	20.34%	23.72%	20.09%
	"What can be got?"	**43.45%**	**24.30%**	**28.28%**	**23.74%**
function	"What is the function?"	49.14%	64.35%	69.38%	63.17%
	"What is the action of the function?"	49.48%	64.08%	69.60%	63.35%
	"What is the action?"	**49.90%**	**65.25%**	**71.27%**	**64.39%**
save	"Where is the input stored?"	37.38%	29.58%	37.03%	29.96%
	"Where is the input preserved?"	**43.96%**	**37.51%**	37.78%	35.61%
	"Where is the input saved?"	**43.96%**	37.39%	**47.87%**	**38.99%**
condition	"What is the condition?"	59.89%	66.48%	68.52%	65.28%
	"What else is needed?"	62.01%	66.39%	**69.12%**	65.31%
	"What else is necessary?"	**64.08%**	**67.79%**	68.99%	**66.17%**

Figure 3: An example showing the process of parsing the information of an IoT device (in this case, a motion-activated security camera) into intermediate JSON and eventually a Python object.

parameter in the object of the Holon class is set according to the value of the parameter, the Holon class has a suitable data type (*e.g.,* double) for the parameters that represent numerical values (such as physical dimensions), so we need to extract the numerical value and unit, respectively, then unify the unit.

Consider, for example:

```
"width": "12 inches"
```

We need to extract the value 12 and the unit "inches", respectively, through regular expressions. If we have unified all the units of length into "inches", the value "12" can be directly passed into the Holon class. Otherwise, *i.e.,* if the expressed unit is inconsistent with the unified one, we need to convert the numerical value to a value in the unified one and then pass the converted value into the Holon class.

In order to achieve unit consistency, we have determined unified units of various measurements, as listed in Table II. We then defined a set of methods that are used to convert the units of measurement.

In addition, considering the convenience of transferring values between IoT devices, we also set some special unit conversions, such as resolution conversions. For instance, "1080p" is a video display format that represents a resolution of 1920×1080. However, if only the value of "1080p" is stored in the Holon structure, it is unable to make sure that other devices can use the value "1080p" directly. Therefore, we use the method of resolution conversion to convert such video display format into a resolution of [Integer, Integer] format, which is convenient for sharing parameters and cooperating between IoT devices.

Table II: Unified units for all measurement units involved in our application scenario.

Measurement	Unit
Weight	Ounce
Temperature	Degree Celsius
Wattage	Watt
Voltage	Volt
Frequency	Hertz
Memory unit	GB
Dimensions	Inch
Luminous flux	Lumen
Time	Hour

Pass the list of services. This function passes the list of services as a parameter to the Holon class. For the service part, the function organizes the variable keys and the *answer*s from the question-answering model into services in JSON format in turn. The function then generates a Service class object for each service. Since an IoT device can have multiple services, we define the services in the Holon class as a list, which stores all Service class objects. Finally, the list of services will be passed to the Holon class object as a parameter.

Table III: The number of devices used in our experiments, categorized by type.

#	Device Category	Number of Devices
1	Smart camera	28
2	Speaker	17
3	Smart plug	16
4	Smart bulb	15
5	TV	15
6	Sensor	11
7	Smart coffee maker	8
8	Smart cooker	8
9	Smart washer	7
10	Smart refrigerator	5
11	Smart vacuum cleaner	5
12	Smart toilet	5

2) Holon Creator: As above, we matched the device information from the Internet to the Holon class object according to the defined methods. We defined the Holon Creator to convert the Holon class object into a Holon description in JSON format, which is the final output.

Execution of the Holon Creator generates an ontological description of the device.

V. Evaluation

In this section, we describe an evaluation of the proposed method. Specifically, we assess the performance of the crawlers, fine-tuned question-answering model, and overall method to gain insight into the performance of our automatic Holon description generation method. All experiments were carried out on a MacBook Pro with a 3.1 GHz Dual-Core Intel Core i5 processor, 8GB of memory, and running macOS 10.15.7.

A. Evaluation of Crawler

We evaluated the percentage of correct webpages that the crawler obtains for the input IoT device brand and model. We find 241 of the popular smart home devices in Europe and the US from the Internet, and define a list of string combinations of <*brand+model*>, such as "TP-link P100". We use these 241 strings as input to determine if the crawler can successfully retrieve the relevant webpages of the corresponding devices.

Among these 241 brands and models, the crawler correctly returned 226 pages, giving a correctness percentage of 93.36%.

The failure to retrieve the other 15 pages was due to the unavailability of the corresponding devices on Amazon. In addition, all other product pages were successfully and accurately returned.

B. Evaluation of Knowledge Extraction

1) Experiment Settings: We used a dataset containing 1272 entries of <`context, question, answer`> triples for 140 devices. The devices span 12 different smart home categories, and are shown in Table III.

The selected Roberta-base-squad2 pretrained model was fine-tuned and tested. In terms of parameter settings, the initial learning rate was 5e-5, batch size was 32, and epoch was 20.

2) Results: We evaluated the performance of the fine-tuned model in terms of the accuracy, precision, recall, and F1 score. We abstract the predicted *answer* and real *answer* into two token sequences. Accuracy represents the proportion of exact matches between two sequences. Precision represents the percentage of the number of shared tokens in the two sequences relative to the number of tokens in the predicted *answer* sequence. Recall represents the percentage of shared tokens in the two sequences in the number of tokens in the real *answer* sequence. The F1 score is the harmonic mean of the precision and recall.

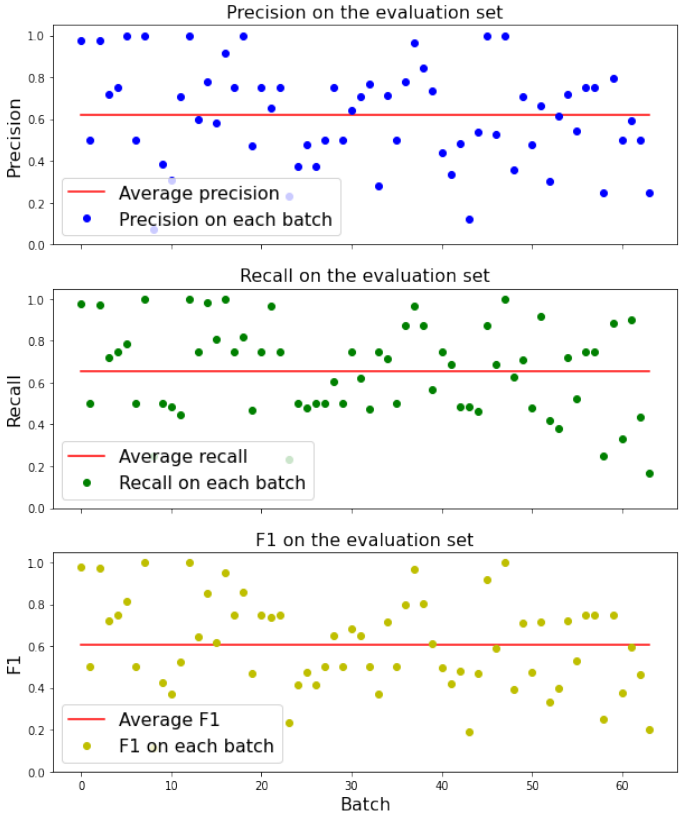

Figure 4: The performance of the model in returning *answer*s to *question*s on the evaluation set.

We measured the above metrics for each batch and report them in Figure 4. We observed that the average value of the metrics for each batch fluctuated, but most of the data points were concentrated in the position of the average value in all batches (as shown by the red line). In the process of multiple training and evaluation, it was also found that the average recall was higher than the average precision. This is because the *answer*s did not quite meet our expectations (*i.e.,* did not fully match our manual labeling strategy). For example, we omitted insignificant adjectives or supplementary phrases during manual labeling, but the model sometimes returns these adjectives or phrases together with the required *answer* during evaluation, which leads to a higher recall and lower precision.

Table IV summarizes the performance of the fine-tuned and original pretrained models on the evaluation set. We observe that the fine-tuned model has a significant improvement in *all* metrics in the knowledge extraction task of the function description over the original pre-trained model.

C. Overall Outcome Assessment

This section evaluates the performance of the overall method of automatic generation of Holon descriptions. We compared the

Table IV: Summary performance metrics of the original pre-trained model against the fine-tuned one.

Model	Metrics			
	Accuracy	*Precision*	*Recall*	*F1*
Pre-trained	0.51%	0.17%	1.02%	0.29%
Fine-tuned	60.74%	61.87%	62.81%	59.92%

generated ontological descriptions with the expected descriptions to gain insight into the accuracy of the method in obtaining correct and relevant information.

We evaluated the performance of the method in generating the Holon description in the parameter part and the function description part separately, and then combined them for the overall performance. In this part of the evaluation, we used the metrics of precision, recall, and f1. However, these metrics have different meanings from those in §V-B; here, they evaluate the ability of the method to retrieve the information. The following paragraphs explain the meanings of the evaluation metrics in each part.

Parameter. For parameter extraction, precision represents the percentage of the correct key-value pairs in all of the obtained pairs, while recall represents the percentage of the key-value pairs obtained in all the pairs that are supposed to be obtained by a well-performing method.

The results of these metrics are summarized in Table V, where we observe that the overall performance of our method is relatively high and acceptable. There was a slight drop in the precision and recall metrics. As for precision, it is because some values do not conform to the regular expression format and, as a result, cannot be obtained. Because the main audience of the data on Amazon product pages are human shoppers, it is generally acceptable for shoppers even if there is some inconsistency between the format of the data and the regulations for some specific keys. However, these data are not recognized by strict, regular expression rules. Take the following key-value pair from the table in one webpage as an example:

```
'product_dimensions': '1"L x 1"W'
```

Shoppers would understand that the value represents both length and width as 1 inch. But the Holon Parser cannot 'understand' this value if the regular expression for `product_dimension` is to get three doubles (*i.e.,* <`length, width, height`>, which is the format for all other devices) in the string of the value.

There were two reasons for the slight drop in the recall metric. The first is the webpage structure. Some product pages have different HTML structures from other products, so the data in these webpages are not equally successfully accessed by the crawlers. For example, the specifications of devices produced by Amazon use a table structure that is different from that of devices of other manufacturers; therefore, Amazon-produced devices will have a lower recall value if the expression of crawlers does not correctly match the table structure and fails to obtain data in it.

Second, owing to the variety of IoT devices, there will be some key names that we did not expect for some devices, and thus, we failed to properly match these key-value pairs onto our Holon ontology structure. Continuing with the above example of Amazon-produced devices, even if we define separate

Table V: The performance in obtaining data representing parameter values.

	Metrics (Average)		
	Precision	*Recall*	*Macro F1*
Parameter extraction	98.17%	94.81%	96.11%

Table VI: The performance in obtaining function description data.

	Metrics (Average)		
	Precision	*Recall*	*Macro F1*
Function description extraction	94.64%	78.00%	84.19%

Table VII: The performance of obtaining data on the overall Holon description.

	Metrics (Average)		
	Precision	*Recall*	*Macro F1*
Overall Holon description]	96.72%	87.53%	91.90%

expressions for tables of these devices, the key names in these tables are relatively flexible and changeable, whereas the key names are generally uniform for other manufacturer devices. Occasionally, there are some 'unexpected keys'.

Function description. We measured the cosine similarity of sentences[4] to determine whether the *answer* to a *question* was successfully retrieved. If the cosine similarity between the returned *answer* and the real *answer* reaches a certain threshold (set to 0.5), the model is considered to have correctly retrieved the *answer*. The reason the cosine similarity is applied is to understand the method's ability to obtain information by evaluating the semantic similarity between the returned *answer* and the real one.

The precision represents the percentage of returned *answer*s that are semantically similar to the actual *answer* to all the returned *answer*s, indicating how accurately the model returns an *answer* to the *question*. The recall represents the percentage of all the returned *answer*s in the *answer*s that should be returned, indicating the effectiveness of the model in returning an *answer* for the *question*.

The performance summary is presented in Table VI. A high precision of 94.64% means that the *answer* is fairly accurate and semantically similar to the real *answer* if one is returned by the model. The slight drop in precision is because sometimes some insignificant adjectives and phrases are returned together with the expected *answer* (as discussed in §V-B), causing semantic similarity to be lower than the threshold. The recall values indicate that the model can only return *answer*s for an average of 78.00% of the *question*s that have *answer*s. The reason no *answer* is returned for a *question* is that the highest-scoring *answer* returned by the model in the *context* is still far below the threshold. Combined with the content in §V-B, the ability of our question-answering model to recognize features still needs to be improved.

Overall Holon description. Finally, we combined the parameter and function description parts and evaluated the performance of the method to generate the overall Holon description.

[4]https://huggingface.co/tasks/sentence-similarity

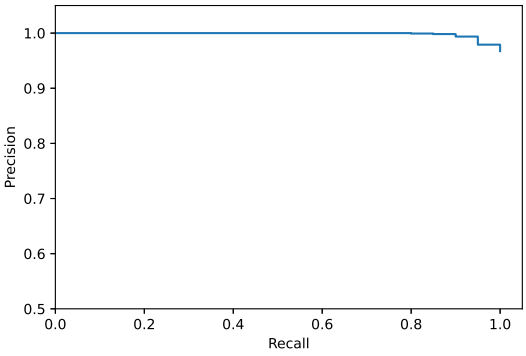

Figure 5: The precision-recall graph.

In this case, precision represents the proportion of the obtained data (on the leaf node of the Holon structure) being correct and relevant among all the data in the final Holon description, and recall represents the proportion of the obtained data among all the data that should be obtained. The results are presented in Table VII.

The precision-recall curve based on the performance results of the 140 devices is plotted in Figure 5. Since the method obtains each piece of data described by Holon as order-independent, the precision rate is initially kept at 1.0, but inaccurate data (if any) emerge as data are gradually recalled to generate full Holon descriptions. So the precision gradually decreases under these circumstances.

Finally, the accuracy of the method in generating Holon descriptions is evaluated to obtain an intuitive understanding of the performance of the method. Of all the devices evaluated, every device generated more than 80% of its Holon description data. Across all 140 devices, 79 accurately generated all Holon description data, accounting for 56.43% of all devices; 134 devices accurately generated more than 90% of Holon description data, accounting for 95.71% of all devices.

D. Summary and Improvement

Evaluation of the above three aspects shows that our method of automatically generating Holon descriptions can efficiently and accurately return the relevant descriptions of the parameters, services, and properties of IoT devices.

VI. Concluding Remarks

A. Summary

In IoT environments where smart homes are widely popular, the collaborative work of smart home IoT devices is just around the corner. As such, composition between IoT systems should be carried out in an effective and automated manner based on accurate and comprehensive descriptions. One means of achieving this is through using Holons to provide an ontological description of systems. However, manually generating Holons is a cumbersome process. Our proposal is to automatically generate Holon (*i.e.,* system) descriptions based on NLP methods and scraping product information from the Amazon website. Our technique achieves accurate and reliable system descriptions, which greatly reduces developer overhead and room for error in building a self-adaptive system of systems.

B. Limitations

Since the main audience of the function descriptions on the Amazon website is human shoppers, some descriptions are colloquial and flexible in structure. Nevertheless, these are not common in our application scenario and only a single digit number of instances of colloquial and flexible-structured paragraphs appear in our whole dataset. In such rare cases, our knowledge extraction model slightly struggles to extract the knowledge points of the various structures, affecting its accuracy.

C. Future Work

We plan to enhance our approach in various ways. First, we will expand the number and types of devices in our experiments to improve the coverage of the method, as well as improve and refine its implementation.

Second, we plan to expand the Holon ontology structure so that it can be applied to the vast majority of smart home IoT devices and eventually has a potential impact as a mature product targeted at the smart home market.

Third, we aim to expand beyond the smart home context to other IoT environments – such as hospitals, airports and transport stations, farmlands, and forests – where there is a higher demand for equipment interconnection and system composition.

Acknowledgment

This work was partly supported by the UK EPSRC under grant number EP/R010889/2.

References

[1] T. Yamakami, "A social dimension view model of divergence of IoT standardization," in *Innovative Mobile and Internet Services in Ubiquitous Computing*, L. Barolli, F. Xhafa, N. Javaid, and T. Enokido, Eds. Cham: Springer International Publishing, 2019, pp. 738–747.

[2] G. S. Blair, Y.-D. Bromberg, G. Coulson, Y. Elkhatib, L. Réveillère, H. B. Ribeiro, E. Rivière, and F. Taïani, "Holons: Towards a systematic approach to composing systems of systems," in *Workshop on Adaptive and Reflective Middleware (ARM)*, 2015, pp. 5:1–5:6.

[3] S. Frey, A. Diaconescu, D. Menga, and I. Demeure, "A generic holonic control architecture for heterogeneous multiscale and multiobjective smart microgrids," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 10, no. 2, Jun. 2015.

[4] V. Nundloll, Y. Elkhatib, A. Elhabbash, and G. S. Blair, "An ontological framework for opportunistic composition of IoT systems," in *International Conference on Informatics, IoT, and Enabling Technologies (ICIoT)*. IEEE, Feb. 2020.

[5] A. Elhabbash, V. Nundloll, Y. Elkhatib, G. S. Blair, and V. Sanz Marco, "An ontological architecture for principled and automated system of systems composition," in *International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, Oct. 2020.

[6] A. Elhabbash, Y. Elkhatib, G. Bouloukakis, and M. Salama, "A middleware for automatic composition and mediation in IoT systems," in *International Conference on the Internet of Things (IoT)*. ACM, Nov. 2022, p. 127–134.

[7] O. Thompson, A. M. Mandalari, and H. Haddadi, "Rapid IoT device identification at the edge," in *Workshop on Distributed Machine Learning (DistributedML)*. ACM, 2021, p. 22–28.

[8] J. Ren, D. J. Dubois, D. Choffnes, A. M. Mandalari, R. Kolcun, and H. Haddadi, "Information exposure from consumer IoT devices: A multidimensional, network-informed measurement approach," in *Internet Measurement Conference (IMC)*. ACM, 2019, p. 267–279.

[9] P. B. B. Jyothi, S. Jyothi, and K. Sekar, "Knowledge extraction using rule based decision tree approach," *International Journal of Computer Science and Network Security (IJCSNS)*, vol. 8, no. 7, p. 296, 2008.

[10] M. Miwa and M. Bansal, "End-to-end relation extraction using LSTMs on sequences and tree structures," *CoRR*, vol. abs/1601.00770, 2016. [Online]. Available: http://arxiv.org/abs/1601.00770

[11] W. Feng, H. H. Zhuo, and S. Kambhampati, "Extracting action sequences from texts based on deep reinforcement learning," *CoRR*, vol. abs/1803.02632, 2018. [Online]. Available: http://arxiv.org/abs/1803.02632

[12] A. Sil, F. Huang, and A. Yates, "Extracting action and event semantics from web text," in *AAAI Fall Symposium Series*, 2010.
[13] Y. Zhang, G. Xu, Y. Wang, D. Lin, F. Li, C. Wu, J. Zhang, and T. Huang, "A question answering-based framework for one-step event argument extraction," *IEEE Access*, vol. 8, pp. 65 420–65 431, 2020.
[14] W. Zijia, L. Ye, and Z. Zhongkai, "BERT-based knowledge extraction method of unstructured domain text," *CoRR*, vol. abs/2103.00728, 2021. [Online]. Available: https://arxiv.org/abs/2103.00728
[15] M. W. Maier, "Architecting principles for systems-of-systems," *Systems Engineering*, vol. 1, no. 4, pp. 267–284, 1998.
[16] C. Ncube, P. Oberndorf, and A. W. Kark, "Opportunistic software systems development: Making systems from what's available," *IEEE Software*, vol. 25, no. 6, pp. 38–41, 2008.
[17] Z. Fang, "System-of-Systems Architecture Selection: A survey of issues, methods, and opportunities," *IEEE Systems Journal*, vol. 16, no. 3, pp. 4768–4779, 2022.
[18] Z. Wang, Y. Elkhatib, and A. Elhabbash, "HolonCraft – an architecture for dynamic construction of smart home workflows," in *Conference on Future Internet of Things and Cloud (FiCloud)*. IEEE, Aug. 2022.
[19] D. Weyns and J. Andersson, "On the challenges of self-adaptation in systems of systems," in *Workshop on Software Engineering for Systems-of-Systems (SESoS)*. ACM, 2013, p. 47–51.
[20] P. Gomes, E. Cavalcante, P. Maia, T. Batista, and K. Oliveira, "A systematic mapping on discovery and composition mechanisms for systems-of-systems," in *Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2015, pp. 191–198.
[21] L. Sabatucci, C. Lodato, S. Lopes, and M. Cossentino, "Highly customizable service composition and orchestration," in *European Conference on Service-Oriented and Cloud Computing (ESOCC)*. Springer, 2015.
[22] M. Kit, I. Gerostathopoulos, T. Bures, P. Hnetynka, and F. Plasil, "An architecture framework for experimentations with self-adaptive cyber-physical systems," in *IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 2015, pp. 93–96.
[23] N. Harrand, F. Fleurey, B. Morin, and K. E. Husa, "ThingML: A language and code generation framework for heterogeneous targets," in *ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS)*. ACM, 2016, p. 125–135.
[24] M. Abu-Matar, "Towards a software defined reference architecture for smart city ecosystems," in *International Smart Cities Conference (ISC2)*. IEEE, 2016.
[25] G. Uddin, F. Sabir, Y.-G. Guéhéneuc, O. Alam, and F. Khomh, "An empirical study of IoT topics in IoT developer discussions on stack overflow," *Empirical Software Engineering*, vol. 26, no. 6, 2021.
[26] S. Team, "Smart home consumer trends and shopping insights," https://www.security.org/smart-home/consumer-shopping-insights/, Apr. 2021.
[27] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018. [Online]. Available: http://arxiv.org/abs/1810.04805
[28] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "RoBERTa: A robustly optimized BERT pretraining approach," *CoRR*, vol. abs/1907.11692, 2019. [Online]. Available: http://arxiv.org/abs/1907.11692
[29] P. Rajpurkar, R. Jia, and P. Liang, "Know what you don't know: Unanswerable questions for SQuAD," *CoRR*, vol. abs/1806.03822, 2018. [Online]. Available: http://arxiv.org/abs/1806.03822