Fraship: A Framework to Support End-User Personalization of Smart Home Services with Runtime Knowledge Graph

Zhiming Huang†
Fuzhou University
China
n180327033@fzu.edu.cn

Jiawen Chen∗
Fuzhou University
China
211027064@fzu.edu.cn

Liwei Shen†
Fudan University
China
shenliwei@fudan.edu.cn

Xing Chen∗‡
Fuzhou University
China
chenxing@fzu.edu.cn

ABSTRACT
With the continuous popularization of smart home devices, people often anticipate using different smart devices through natural language instructions and require personalized smart home services. However, existing challenges include the interoperability of smart devices and a comprehensive understanding of the user environment. This study proposes Fraship, a framework supporting smart home service personalization for end-users. It incorporates a runtime knowledge graph acting as a bridge between users’ language instructions and the corresponding operations of smart devices. The runtime knowledge graph is used to reflect contextual information in a specific smart home, based on which a language-instruction parser is proposed to allow users to manage smart home devices and services in natural language. We evaluated Fraship on a real-world smart home. Our results show that Fraship can effectively manage smart home devices and services based on the runtime knowledge graph, and it recognizes instructions more accurately than other approaches.

KEYWORDS
Internet of Things, knowledge graph, runtime software architecture, smart home, virtual assistant

∗‡Z. Huang, J. Chen and X. Chen are with the College of Mathematics and Computer Science, Fuzhou University, and Fujian Key Laboratory of Network Computing and Intelligent Information Processing, China.
†L. Shen is with the School of Computer Science and Shanghai Key Laboratory of Data Science, Fudan University, China.
∗X. Chen is the corresponding author.

1 INTRODUCTION
With the continuous popularization of smart home devices, human life has gradually entered a new era characterized by device-enabled smart services. Smart home services improve the quality of people’s lives by integrating housing facilities with the technology of the Internet of things (IoT) [9, 33], e.g., facilitating the elderly to turn on or off the light, facilitating the young people to do housework, etc. At present, sizeable smart device manufacturers such as Apple, Huawei, and Samsung are committed to building a device ecosystem covering all aspects of family life. Each manufacturer releases a platform to uniformly access and manage devices in the ecosystem, e.g., Apple Homekit [1], Huawei HiLink [15] and Samsung Smart Home [23]. Users install the platform-specific applications on their smartphones to gain control of the devices. Usually the applications provide the ability to control single devices, and to define rules for scenarios collaborating several devices [26].

In a living space surrounded by smart devices, users often require personalized smart home services [4, 17, 31]. They may need to understand the specific status of a device or directly control it, whether the windows have been closed. In addition, they may specify rules for device collaboration that enable smart devices to be activated under specific environmental conditions. For example, automatically turn on the bathroom light when a sensor in the corridor detects someone passing by at night. The personalized scenarios may be quite complicated. Hence, program-level service implementation is required to support the management and collaboration of smart home devices [10, 13]. It is also necessary to provide users with an easy-to-use interaction mode. Natural language instructions are a reasonable solution to manage smart home devices and services.

However, the personalization of smart home services cannot be fully supported because of challenges related to the interoperability of smart devices and the comprehensive understanding of the user environment. On the one hand, devices of different
brands need to be accessed through different applications. The data format, data communication protocols, and function invocation methods for these devices also differ. Therefore, the interaction and collaboration between devices is difficult in scenarios involving cross-brand devices. On the other hand, personalized smart home services should be linked to contextual knowledge to provide real-time, flexible, and valuable services. Thus, a relationship must exist between the context and smart home services [16, 21]. However, existing smart homes lack a comprehensive knowledge representation at the abstract level [19].

To address these challenges, this paper proposes Fraship, a framework which supports smart home service personalization for end-users (i.e., understanding end-users’ natural language instructions and transforming them into smart home services). We introduce a runtime knowledge graph that acts as a bridge between the instructions and the corresponding operations of smart devices. The knowledge graph is used to reflect contextual information (such as the status of each device or positions of persons) in a specific smart home, and thereby directly operate smart home devices. Based on the runtime knowledge graph, we propose a language-instruction parser to allow users to manage smart home devices and services in natural language.

We applied Fraship to a real-world smart home with 3 areas, 11 devices, and 29 device services. The results proved the effectiveness of the runtime knowledge graph for managing smart home devices and services. In addition, we experimentally evaluated Fraship’s accuracy of instruction recognition using primitive and compound instructions provided by trained workers and ordinary participants. The results showed that Fraship outperformed other methods, including rule-based [6] and machine learning (ML)-based [2] approaches.

The remainder of this paper is organized as follows. Section 2 introduces related work. Section 3 presents an overview of Fraship. Section 4 illustrates the runtime knowledge graph for a smart home. Section 5 describes the language-instruction parser. Section 6 evaluates Fraship on a real-world smart home. Section 7 concludes this paper and discusses future work.

2 RELATED WORK

A smart home is a typical scenario for IoT. The programming for current IoT applications focuses on the operating system level rather than the application logic level [24], which requires developers to have a deep understanding of the underlying system-related technology. Cheng et al. [7] proposed a distributed event-centric collaborative work flow development system for IoT applications called DECW, which supports loosely coupled event-based interaction between processes. Dong et al. [11] proposed a novel adaptive service-oriented paradigm to overcome the disadvantages of the REST architecture style in IoT. Zhang et al. [32] proposed a declarative approach to construct an event-driven IoT service system, where physical devices and systems are explicitly modeled as a service architecture foundation. Chen et al. [5] proposed an IoT application development method based on a runtime software architecture model, which abstracts device capabilities into runtime models and achieves bidirectional synchronization between the application scenario models and the device runtime models through model conversion. Based on device-as-a-software, Hu et al. [14] developed an open IoT system architecture that decoupled upper-level applications from the underlying physical devices using the mechanism of a software-defined device. The above work effectively improved the abstraction level of IoT application development by introducing integrated architectures to manage heterogeneous devices. However, developers needed to manually customize the management logic of IoT applications for specific scenarios.

To support the management logic development for IoT applications, some studies focused on semantic representation. Mallick et al. [18] proposed a data-driven de-multiplexing approach called temporal-sensor-frequency-stitch that disentangled each activity from the sensor stream and thus simplified the activities of daily-living recognition problems. Wang et al. [28] proposed an elegant, natural, and compact data representation model, called a domain-oriented user and service interaction knowledge graph. Xie et al. [29] proposed a knowledge graph-based multilayer IoT middleware, which introduced a new layer to bridge the gap between IoT devices with different communication protocols. The above methods can obtain data from the underlying sensors, reduce the uncertainty of obtaining data, and simplify coding of the management logic.

To support the service assembly and knowledge reasoning of IoT applications, some studies adapted the concept of semantic web to IoT. Tao et al. [27] designed a general domain ontology model of smart home, and studied ontology-based semantic reasoning with a specific semantic matching rule. Rosa et al. [22] presented a knowledge-based recommendation system that included an emotional health monitoring system to detect users with potential psychological disturbances. Yan et al. [30] applied unsupervised learning to extract latent knowledge and embed the activity probability distribution prediction as high-level features to boost real-time activity recognition performance. The above study performed semantic modeling based on sensing data and software services of IoT; however, there is no linkage mechanism between its semantic model and underlying data and devices.

Virtual assistants are widely used in people’s daily lives, including in social situations and smart homes. In the past few years, several studies have investigated how to build virtual assistants. LIA [8] is the first speech-based virtual assistant that can be taught new commands through speech. Almond [2] is the only virtual assistant that allows users to specify trigger-action commands in natural language. Almond can control intelligent devices and create automated tasks. Moreover, an architecture of an open virtual assistant and methodology to acquire training data was proposed. The Genie toolkit [3] is a follow-up work on Almond, in which a semantic parser was trained for a new virtual assistant. The Genie toolkit can handle new compound commands with significantly less manual effort. However, in the case of a limited corpus, the language-instruction parsers obtained by these virtual assistant construction methods are often inaccurate.

3 SYSTEM ARCHITECTURE

Fig. 1 presents an overview of the Fraship supporting end-user personalization of smart home services. It takes the end-user’s language instruction as input, matches it to a use-case scenario,
4 RUNTIME KNOWLEDGE GRAPH FOR SMART HOME

This section describes the runtime knowledge graph for smart homes, which conforms to the concept model, as shown in Fig. 2. The core concepts are Location, Context, Device, Service, and User, represented as ellipses in Fig 2. We define each concept as follows:

- **Location** is a specific area in a home. We use a single attribute LName to represent the identifier of the location.
- **Context** is a specific environment index that can be perceived in a smart home, e.g., brightness and temperature. It is defined as a triple-tuple (CType, LName, CValue), which includes the index name, perception location, and the value of the context.
- **Device** is a concept for the smart devices at home. A device is represented by a tuple (DName, LName, CType, Effect, Status, CValue). The first three attributes correspond to the name, deployment location, and the enabled status of the device. The fourth attribute is an embedded array with configuration parameters or system indicators of the device (such as brightness and humidity).
- **Service** is the ability provided by a device to affect the environment index by manipulating the device. A service is defined as a tuple (DName, LName, CType, Effect, Status, CValue). The first three attributes describe the name of the device that provides the service, the area where the service operates, and the controlled environment index. The fourth attribute indicates the type of effect on the context when the service is applied: Monitor, Increase, Reduce, or Assign. The fifth attribute denotes the enabled status of the service. The last attribute records the value of the environment index that is monitored by the service or indicates the configuration parameter that is assigned by the service to impact the environment index.
- **User** is involved in the concept model as an object of the service. We use a tuple (UName, LName) to indicate the name and current location of the user.

The concept model defines the relationships between the concepts (directed dotted lines with text labels in Fig. 2):

- **Located in** (Located in) brings the spatial semantics or constraints to the concepts of Device, Context, Service, and User, denoting the deployment location of a device, the perceived location of a context, the operation area of a service, and the current location of a user, respectively.
- **Sense** (Sense) is a relationship between User and Context that indicates the context to which the user is sensitive.
- **Provide** (Provide) indicates the services a device can offer.
- **Monitor** (Monitor), **Increase** (Increase), and **Reduce** (Reduce) are four types of relationships between Service and Context, which indicate...
the different types of effects on the context when the service is applied. Among them, Monitor means tracking the status of a context. Increase and Reduce imply increasing or decreasing the context value by a unit. Assign sets a specific value to the context.

To maintain the bidirectional synchronization between the knowledge graph and the real world, the runtime software architecture model [6] [5] [25] is used to construct a runtime model of the knowledge graph. It operates devices at the model level, and the values of attributes related to situational knowledge are not assigned but obtained in runtime through mapping operations. Thus, the runtime knowledge graph may evolve owing to changes in the situational context. Hence, the personalization of smart home services can be implemented by model operations on the runtime knowledge graph.

5 LANGUAGE-INSTRUCTION PARSER

Based on the runtime knowledge graph, we present the language-instruction parser in this section. First, we describe the classification and generation of use-case scenarios. Subsequently, we explain the matching between end-user language instructions and a ranked list of use-case scenarios. Importantly, the execution of services depends on the runtime knowledge graph.

5.1 Classification of Use-Case Scenarios

We define a use-case scenario as a capability description that consists of the services provided by one or several smart home devices. The scenarios are divided into three categories based on their capabilities.

- **Context control.** A scenario belonging to this category aims to change the state of a context by adjusting the attribute value of a device or service in the knowledge graph. For example, "Turn on the lights in the bedroom." and "Increase the temperature in the living room." are two model operations generated for this type of scenario.

- **Quantitative query.** A scenario belonging to this category aims to obtain the state of a device or context by retrieving the corresponding attribute of a device or context instance from the runtime knowledge graph. For example, "What is the PM 2.5 in the living room?" indicates invoking the monitoring capability of a device that can monitor the PM 2.5 in the living room.

- **Rule setting.** A scenario belonging to this category can trigger a context-control scenario according to a specific environmental state value by setting an execution inference rule. For example, a plant in the balcony can survive in a situation where the humidity is greater than 20%. In this case, a rule is set to monitor the humidity in the balcony. When the humidity is lower than 20%, the smart water pump is opened.

5.2 Generation of Use-Case Scenarios

We generate in advance all possible use-case scenarios from the runtime knowledge graph to improve the efficiency of matching language instructions. Use-case scenarios are generated based on the corresponding model operations and their text patterns as follows.

First, the model operations for context-control scenarios include operations on the device and service instances of the runtime knowledge graph.

- The model operations for device instances are Set $D_i.Status$ to on/off and Set $D_i.Key_{ym}$ to $X$.
- The model operations for service instances are Set $S_i.Status$ to on/off and Set $S_i.Value$ to $X$.

Table 1 lists the generation rules for context-control use-case scenarios. Each row indicates a pattern for generating a scenario with a textual description converted from a model operation to the concept instance of a runtime knowledge graph. For example, we assign $D_i.DName$ as “air conditioner” and $D_i.Status$ as “on” for the model operation “Set $D_i.Status$ to on.” The corresponding generated use-case scenario is “Turn on the air conditioner in the living room.” Another example is that if we set $S_i.CType$ as “brightness,” $S_i.Effect$ as “reduce,” and $S_i.LName$ as “living room,” a scenario “Reduce the brightness in the living room.” can be generated according to the model operation “Set $S_i.Status$ to off.”

Second, the model operations for quantitative-query scenarios include operations on the device and service instances of the runtime knowledge graph.

- The model operations for device instances are as follows: Get $D_i.Status$ and Get $D_i.Key_{ym}$.
- The model operations for context instances are: Get $C_i.Value$.

Similarly, Table 2 lists the generation rules for quantitative-query scenarios. Each row of the table presents a pattern for a querying event, including the object and the specific control. For example, to query whether the smart light is on, we assign $D_i.DName$ as “smart light” for the model operation “Get $D_i.Status$.” In this case, a use-case scenario “Is the smart light on?” is generated.

Third, the model operation for a rule-setting scenario is a compound operation composed of a decision part and an action part. The decision part is a Boolean expression related to the context value. When the Boolean expression is satisfied, the corresponding action is triggered. The model operation is defined as follows: $C_i.CValue=X \ | \ C_i.CValue=X \ | \ Y < \ C_i.CValue < X \Rightarrow$ Set $D_i.Key_{ym}$ to $X \ | \ Set\ D_i.Status\ to\ on/off \ | \ Set\ S_i.Status\ to\ on/off \ | \ Set\ S_i.Value\ to\ X$.

To convert the model operation to the use-case scenario, we list the conditions in Table 3, in which each row denotes a pattern for generating the Boolean expression part of the scenario. Consider a rule that raises the temperature in the living room when it is lower than 16°C. We take “$C_i.CValue < X$” as the decision part. We further assign $C_i.LName$ as “living room,” $C_i.CValue$ as “temperature,” and $X$ as “16°C.” In addition, we choose the model operation “Set $S_i.Status$ to on” as the action part. In this pattern, we assign $S_i.CType$ as “temperature,” $S_i.Effect$ as “increase,” and $S_i.LName$ as “living room.” The generated use-case scenario can be “If the temperature in the living room is less than 16°C, increase the temperature of the living room.”

5.3 Language Instruction to Use-Case Scenario

When a user inputs a language instruction, the instruction is matched according to the similarity between the language instruction and each
### Table 1: Generation rules for context-control scenarios

<table>
<thead>
<tr>
<th>Object</th>
<th>Given condition</th>
<th>Model operation</th>
<th>Corresponding use-case scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device</td>
<td>∀ D&lt;sub&gt;i&lt;/sub&gt;</td>
<td>Set D&lt;sub&gt;i&lt;/sub&gt; to on</td>
<td>Turn on/Open the D&lt;sub&gt;i&lt;/sub&gt; in the D&lt;sub&gt;i&lt;/sub&gt; LName</td>
</tr>
<tr>
<td>Service</td>
<td>∀ D&lt;sub&gt;i&lt;/sub&gt;, Key&lt;sub&gt;m&lt;/sub&gt;</td>
<td>Set D&lt;sub&gt;i&lt;/sub&gt;, Key&lt;sub&gt;m&lt;/sub&gt; to X</td>
<td>Set/Turn &quot;Key&lt;sub&gt;m&lt;/sub&gt;&quot; of the D&lt;sub&gt;i&lt;/sub&gt; in the S&lt;sub&gt;i&lt;/sub&gt; LName to X</td>
</tr>
</tbody>
</table>

### Table 2: Generation rules for quantitative-query scenarios

<table>
<thead>
<tr>
<th>Query the status and properties of the device</th>
<th>∀ D&lt;sub&gt;i&lt;/sub&gt;</th>
<th>Get D&lt;sub&gt;i&lt;/sub&gt;, Status</th>
<th>Is the D&lt;sub&gt;i&lt;/sub&gt;, DName on/off?</th>
<th>What is the status of the D&lt;sub&gt;i&lt;/sub&gt;, DName?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query the status of the context</td>
<td>∀ C&lt;sub&gt;i&lt;/sub&gt;</td>
<td>Get C&lt;sub&gt;i&lt;/sub&gt;, CValue</td>
<td>What is the C&lt;sub&gt;i&lt;/sub&gt;, CType in the C&lt;sub&gt;i&lt;/sub&gt;, LName?</td>
<td></td>
</tr>
</tbody>
</table>

### Table 3: Generation rules for rule-setting scenarios

<table>
<thead>
<tr>
<th>Given condition</th>
<th>Condition statement</th>
<th>Corresponding use-case scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>∀ C&lt;sub&gt;i&lt;/sub&gt;</td>
<td>C&lt;sub&gt;i&lt;/sub&gt;, CValue ▷ X</td>
<td>If the C&lt;sub&gt;i&lt;/sub&gt;, CType in the C&lt;sub&gt;i&lt;/sub&gt;, LName is higher than/above/higher than X</td>
</tr>
<tr>
<td>∀ C&lt;sub&gt;i&lt;/sub&gt;</td>
<td>C&lt;sub&gt;i&lt;/sub&gt;, CValue ▷ X</td>
<td>If the C&lt;sub&gt;i&lt;/sub&gt;, CType in the C&lt;sub&gt;i&lt;/sub&gt;, LName is lower than/below/low than X</td>
</tr>
<tr>
<td>∀ C&lt;sub&gt;i&lt;/sub&gt;</td>
<td>Y ▷ C&lt;sub&gt;i&lt;/sub&gt;, CValue ▷ X</td>
<td>If the C&lt;sub&gt;i&lt;/sub&gt;, CType in the C&lt;sub&gt;i&lt;/sub&gt;, LName is between Y and X</td>
</tr>
</tbody>
</table>

### Scenario’s Description

In the matching process, a simple sentence is directly matched with the context-control and quantitative-query scenarios. In contrast, a compound sentence with an If clause is first divided into two simple sentences, each of which is matched separately. The overall similarity is then calculated based on the two sub-similarities. In addition, we consider the user’s language instruction to be location-sensitive; that is, users intend to invoke the services of devices near their location when the instruction does not mention a precise location. Therefore, under these circumstances, we append the obtained user location to the language instruction.

Our method first leverages word embedding to transform a sentence composed of a set of words into a distributed representation, i.e., a sentence vector. A sentence vector is synthesized by the word vectors of the contained words depending on the Word2Vec model [20], which is trained on a large-scale training corpus. By setting the number of dimensions of the word vector (i.e., the length of the vector), the vector of a sentence is expressed as the average of all the word vectors in the sentence. Therefore, each sentence can be transformed into a vector of the same length.

Equation (1) shows a sentence X consisting of n words and represented as an m-dimensional sentence vector. The word vector for a contained word x<sub>i</sub> is an m-dimensional vector, as shown in (2). Each dimensional value of sentence X is calculated using (3), which is the average value of the word vectors of a specific dimension.

\[
X = (x_1, x_2, ..., x_n) \quad (1)
\]

\[
x_i = (x_{1i}, x_{2i}, ..., x_{ni}) \quad (2)
\]

\[
X_j = \frac{\sum_{i=1}^{n} x_{ij}}{n} \quad (j = 1, 2, ..., m) \quad (3)
\]

Given two sentence vectors, which denote a language instruction and a scenario description, respectively, our method evaluates their cosine similarity. Suppose a is a sentence whose vector can be expressed as \((x_1, x_2, ..., x_m)\) and b is another sentence whose vector can be expressed as \((y_1, y_2, ..., y_m)\). The similarity of two multidimensional vectors can be calculated using (4). A value closer to 1 indicates a higher similarity.

\[
\cos(\theta) = \frac{a \cdot b}{\|a\| \|b\|} = \frac{\sum_{i=1}^{m} (x_i \cdot y_i)}{\sqrt{\sum_{i=1}^{m} (x_i)^2} \times \sqrt{\sum_{i=1}^{m} (y_i)^2}} \quad (4)
\]

For each language instruction that is a simple sentence, we calculate its corresponding vector with all use-case scenario vectors, and we reserve the top five scenarios with the highest similarity value as the matching results. In addition, for a compound sentence with an If clause, two simple sentences are first extracted as the condition and action sentences. Each sentence is matched separately, and the top five results are reserved. Subsequently, we combine both parts using the Cartesian product and obtain 25 rule settings. Each rule setting is transformed into a sentence vector, and the similarity between the rule and the language instruction is evaluated again. The top five results from the further similarity evaluation are taken as the matching results for the compound instruction.

### 6 Evaluation

We implemented Fraship and evaluated it to answer the following research questions:

**RQ1:** Can Fraship effectively manage smart home devices and services (Section 6.1)?

**RQ2:** What is Fraship’s accuracy of instruction recognition (Section 6.2)?
To answer RQ1, we applied Fraship to a real-world smart home. The details are described in [12].

6.1 Example of Industrial Application

6.1.1 Setting. The scenario includes 3 areas: living room, bedroom, and balcony. Smart devices are arranged in each area, as shown in Fig. 3. The scenario includes four types of contexts: temperature, humidity, brightness, and PM 2.5. Smart devices provide services such as Monitor, Increase, Reduce and Assign for different contexts. For example, air conditioners can monitor, increase, decrease, or assign the temperature. Air purifiers can provide services that decrease the PM 2.5.

6.1.2 Results. As shown in Fig. 4, Fraship created 3 location instances (L1), 12 context instances (C1-l), 11 device instances (D1) and 29 service instances (S1-d, l). Fraship further specifies two users Alice and Ken, whose instances are created (U1 and U2).

Based on the basic knowledge of smart device deployment, Fraship initially established 54 Located in, 29 Provide, 9 Monitor, 8 Increase, 5 Reduce and 7 Assign relationships. The Sense relationship is determined by the actual location of the user. Fraship assumes that user U2 is currently in the bedroom (L1), while user U1 is not yet at home. Therefore, Fraship establishes an additional Located in and 4 Sense relationships starting from U2. Finally, a runtime knowledge graph is constructed based on the above concept and relationship instances.

The runtime knowledge graph can effectively express contextual information and establish consistent relationships between the concept instances of a smart home. Based on this, Fraship can recognize natural language instructions.

We use an example to illustrate this process. Suppose there is a natural language command “Turn up the brightness.” We first improve the command by appending the location information, and hence, the matching similarity can be increased. For example, if a user proposing the command is currently located in the living room, the sentence can be modified to “Turn up the brightness in the living room.” Next, the sentence is transformed into a vector, and the top five results are listed as follows:

1. Increase the brightness of the living room.
2. Reduce the brightness of the living room.
3. Monitor the brightness of the living room.
4. Turn on the smart light in the living room.
5. Turn off the smart light in the living room.

The first result is prioritized over the others, with a similarity value of 0.98767. The underlying model operation is “Set Si.Status on,” where Si CType is “brightness,” and Si LName is “living room.” According to the runtime knowledge graph, a smart light is determined to support the command. Finally, the model operation on the device is executed based on the runtime knowledge to achieve the smart home service personalization.

In total, we simulated 2304 language instructions, and more than 85% of them could be executed correctly. The results are detailed in Part B of this section.

6.2 Accuracy of Instruction Recognition

To answer RQ2, we experimentally evaluated the accuracy of instruction recognition based on the above smart home.

6.2.1 Setting. We used three datasets: a base set, a paraphrase set, and a scenario set. In total, 2304 sentences were collected, including 1424 primitive and 680 compound sentences.

The language instructions in the base set were generated using the rules described in Section 5.2. The base set provides basic instructions for the smart home service, which enables basic control of various devices. Our base set included 200 instructions (121 primitive and 79 compound sentences).

The paraphrase set is used to increase the diversity of language instructions and simulate users’ usage of smart home devices in real-world scenarios as much as possible. Therefore, we invited several trained workers who were familiar with the devices and corresponding services of the smart home to customize the instructions. Thus, we collected 1057 sentences, including 714 primitive and 343 compound sentences.

The language instructions in the scenario set were provided by volunteers who were unfamiliar with the various devices and services. Based on our brief introduction to the smart home, these volunteers were asked to give instructions according to their experience and understanding of smart home scenarios. The volunteers provided 1047 instructions, including 710 primitive and 337 compound instructions.

6.2.2 Comparison Approaches. We compared the recognition accuracy of the proposed approach with that of the rule-based approach [6] and the ML-based approach called Almond [2]. The rule-based approach [6] establishes a series of instruction recognition rules and determines the sentence pattern according to the syntax structure of each instruction. It uses the syntax dependency relationship to identify the information, including the actions, device names, location names, and context properties in the instructions. According to the predefined knowledge inference rules, it matches the device, location, and context in the runtime knowledge graph. Thus, it maps the instruction to a specific service that can implement the instruction requirement. On the other hand, Almond [2] obtains information through Thingpedia entries and leverages machine learning to train the Thingtalk program and its paraphrase to construct a semantic parser. The semantic parser is used to match
the users’ instructions to the codes. Only one result is generated using the rule-based approach. In contrast, top-\(n\) results are generated by Fraship and Almond. In this experiment, we measured whether the correct answer appears in the single result of the rule-based approach and in the top-1, top-3, and top-5 results of Fraship and Almond.

6.2.3 Results. The details of the collected instructions and recognition results can be found in [12]. The experimental results are shown in Fig. 5. The left part shows the recognition accuracy rates of primitive instructions by the rule-based approach, Fraship, and Almond for the three datasets. The right part shows the recognition accuracy rates of compound instructions. In particular, the recognition accuracy for matching the instructions in the base set is 100% for all three approaches; therefore, we do not include these results in the figure. The reason for the full match is that the instructions from the base set are generated by the rules, and the vocabulary is limited. Therefore, all three approaches can accurately recognize the words in the instructions and map them to the correct use-case scenarios.

We first analyze the recognition accuracy of the instructions from the paraphrase set. The accuracy of the rule-based approach for primitive instructions was 77%. The rule-based approach only recognizes the sentence structure and specification defined by rules in the text. Therefore, this approach cannot understand the instructions when the sentence structure or lexical representation is not predefined. The accuracy of the rule-based approach for the recognition of compound sentences was 61%. In this case, the structure of compound instructions is more complex and diverse. In this approach, many rules for each type of sentence must be predefined to increase the accuracy; however, the complete rule definition is unlikely to be implemented. For example, if the leading word of the condition in the compound sentence (e.g., “while” or “when”) is undefined, the sentence structure is changed and cannot be recognized.

Using Almond, the parser obtained a top-1 accuracy of 71%, top-3 of 88%, and top-5 of 89% for primitive instructions, and 51% for top-1, 61% for top-3, and 63% for top-5 for compound instructions. The Thingpedia of Almond is an open-domain-based knowledge base. Therefore, the recognition accuracy depends on the training result for a particular field, such as a smart home, which requires additional effort. In comparison, the top-1, top-3, and top-5 accuracies of primitive instruction recognition using Fraship were 86%, 91%, and 94%, respectively, which were higher than in the two previous approaches. Fraship can solve the problem of diversity of sentence structure and leading words. However, the applied natural language processing method has limitations, and some groups of synonyms were not recognized. For example, the device operation “turn down” should be matched with “reduce”; however, it was matched with “turn off” because of the higher similarity of the word vectors. Fraship obtained a top-1 accuracy of 74%, top-3 of 84%, and top-5 of 89% for the compound instructions, which were approximately 5% to 12% lower than the accuracy for the primitive instructions. The reduction in accuracy is reasonable because the probability of successful recognition of a compound sentence is affected by two factors, i.e., a compound sentence can be correctly recognized only when the two split primitive sentences are recognized.

Further, we analyze the matching results for the instructions from the scenario set. This dataset contains more complex and colloquial instructions than the paraphrase set, which complicates their recognition.

As for the recognition results for the primitive instructions, the accuracy of the rule-based approach was 64%. The accuracy of Almond was 34% for top-1, 54% for top-3, and 65% for top-5. Fraship achieved 74% for top-1, 81% for top-3, and 86% for top-5. As for the recognition results for the compound instructions, the accuracy of the rule-based approach was 41%. The accuracy of Almond was 22%, 31%, and 33% for top-1, top-3, and top-5, respectively, whereas Fraship achieved 56%, 66%, and 70%, respectively.

The accuracy was low mainly because of the diverse instruction formats and words provided by the volunteers based on their life experience and the given smart home description. For example, a volunteer may provide an abstract condition such as “if the living room is well lighted” to set room brightness. In this case, none of the three approaches can understand the semantics, and hence, the instruction is not correctly recognized. In comparison, Fraship yielded better results than the other two approaches. We define a series of Context instances in the knowledge graph so that our approach
can recognize some of the instructions with high-level semantics that are not processed by the other two approaches. For example, “How is the air quality of the living room?” is a quantitative-query instruction. Both the rule-based approach and Almond cannot understand the specific meaning of “air quality.” Fraship defines the context of PM 2.5 in the living room. Therefore, we know that this query instruction is asking about the air quality, and we correctly match the instruction to the use-case scenario “What is the PM 2.5 in the living room?”.

Next, we compare the accuracy of recognition between the two datasets. As for the primitive instructions, the accuracy for the scenario set is reduced by 14% using the rule-based approach, by 24% to 43% using Almond, and by 8% to 12% using Fraship. The reduction is 23%, 29% to 30%, and 18% to 19% for the compound instructions, respectively. The comparison shows that Fraship can avoid a sharp reduction in recognition accuracy when encountering highly abstract semantic instructions in smart home service scenarios. In summary, Fraship not only recognizes instructions generated by rules but also has a high recognition accuracy for instructions provided by trained workers. For instructions with complex and abstract semantics provided by volunteers, Fraship can also achieve acceptable recognition results.

6.3 Discussion

Users of smart home services benefit from the framework because it performs end-to-end control of smart devices. Taking natural language instructions as input, Fraship attempts to understand the instructions by matching them to prespecified use-case scenarios, and the devices are then manipulated based on the runtime model mechanism. Users can use cross-brand and cross-type devices anytime and anywhere when a unified smart home service is built in the form of a knowledge graph.

Moreover, developers of smart home services benefit from the framework because of the low requirements for development. Instead of hard-coded service programs, Fraship depends on scenario configurations to construct the knowledge graph. Those without software development skills can provide these configurations after being trained on the configuration specifications.

We believe that Fraship can be applied as a core component of smart home assistants. The input natural language instructions can be obtained through a smart speaker deployed at home or the user’s personal smartphone. The derived operations drive the execution of the corresponding devices when the devices are integrated using the runtime model architecture. Furthermore, Fraship provides top-n matching results to improve usability. Therefore, an interactive process can be offered to the user to determine the anticipated operation when the similarities of the top-n results are comparable. The interaction can be implemented using a smart speaker, which can read aloud the operation options and obtain feedback. In addition, feedback can also be achieved through text or voice interactions between the smartphone and the user.

7 CONCLUSION AND FUTURE WORK

This study proposed Fraship, a framework supporting smart home service personalization for end-users. We introduced a runtime knowledge graph that acted as a bridge between instructions and the corresponding operations of smart home devices. Based on this, we proposed a language-instruction parser to allow users to manage smart home devices and services in natural language. We evaluated the Fraship on a real-world smart home. The results show that Fraship can effectively manage smart home devices and services and has a high accuracy of instruction recognition.

Future work will focus on two aspects. First, we plan to further employ natural language processing and sophisticated semantic analysis techniques to improve the accuracy of instruction recognition. Second, we plan to use pattern recognition and reinforcement learning techniques to automatically generate trigger-action tasks for smart homes by analyzing huge amounts of data on user behavior.

ACKNOWLEDGMENTS

This work is supported by the National Key R&D Program of China under Grant No. 2018YFB1004800, and the Natural Science Foundation of Fujian Province for Distinguished Young Scholars under Grant No. 2020J06014.
REFERENCES


