Designing smart home services using machine learning and knowledge-based approaches

Mingming Qiu

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2023IPPAT014

Thèse de doctorat

Designing smart home services using machine learning and knowledge-based approaches

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à Télécom Paris

École doctorale n°626 École doctorale de l'Institut Polytechnique de Paris (ED IP Paris)
Spécialité de doctorat : Informatique

Thèse présentée et soutenue à Palaiseau, le 28 Avril 2023, par

Mingming Qiu

Composition du Jury :

Gérard MEMMI Professeur, Télécom Paris	Examinateur, Président
Alessandro FANTECHI Professeur des universités, Université de Florence	Rapporteur
Patricia PASCAL-STOLF Professeure, Institut de Recherche en Informatique de Toulouse	Rapporteuse
Zoubida KEDAD Maîtresse de conférences, Université de Versailles Saint-Quentin-en-Yvelines	Examinatrice
Elie NAJM Professeur émérite, Télécom Paris	Directeur de thèse
Rémi SHARROCK Maître de conférences, Télécom Paris	Co-encadrant de thèse
Bruno TRAVERSON Ingénieur de recherche, Électricité de France R&D	Invité
Philippe FUTTERSACK Ingénieur de recherche, Électricité de France R&D	Invité

DESIGNING SMART HOME SERVICES USING MACHINE LEARNING AND KNOWLEDGE-BASED APPROACHES

A DISSERTATION PRESENTED
BY
MINGMING QIU

TELECOM PARIS
INSTITUT POLYTECHNIQUE DE PARIS
APRIL 2023

Acknowledgements

The completion of this thesis would not have been possible without the help and support of many people to whom I would like to express my gratitude.

First, I would like to thank my thesis supervisors, Elie Najm, Bruno Traverson, and Rémi Sharrock, for the time they devoted to my work. Elie Najm, my thesis director, helped me a lot in greatly enriching my work with his experience, previous advice, and unique insights, which are particularly essential in achieving the present contributions. Bruno Traverson, my co-supervisor, helped me a lot with the details of my work and gave me unique insights into the problems, which are very important to realize the rigorous work. Rémi Sharrock, my co-supervisor, helped me with the work and software tools and provided me with suggestions to improve my work efficiency.

Next, I would like to thank Alessandro Fantechi and Patricia Pascal-Stolf for agreeing to be the reviewers to review my work and provide insightful comments so that I can improve my thesis and for accepting to be jury members. I would also like to thank Gérard Memmi and Zoubida Kedad for accepting to be jury members and for taking the time to read the work. I would also like to thank Philippe Futtersack for accepting to be invited to my defense, for giving me advice on my work, and for encouraging me throughout my work.

I would also like to thank Amel Bouzeghoub and Maxime Lefrançois for having reviewed my work during my midterm defense. Their insightful comments and suggestions have improved my work and encouraged me in my thesis work.

In addition, I would like to thank the SEIDO laboratory that funded this work. I would also like to thank my colleagues at EDF and Télécom Paris for their help and support. It is a pleasant experience to work with them. I want to thank Drs. Ada Diaconescu, Jan Gugenheimer, Liu Jiali, and Zhuoming Zhang, as well as PhD fellows Wen-Jie Tseng, Elise Bonnail, Yang Liu, and Gaëlle Clavelin. I am so glad to have you in the ACES and DIVA labs.

Moreover, I would like to thank Pierre Banquy for his encouragement, support, and help with my work, and Drs. Sibo Cheng and Yuwei Wang for their encouragement.

Finally, I would like to thank my family, who always support me unconditionally, believe in me, and encourage me. I want to thank all the people I could not mention who have given me valuable help.

Abstract

Smart homes are homes equipped with connected objects (sensors for retrieving state values from the environment, actuators for influencing these state values) and with services that interact with these objects with the aim to provide for the inhabitant more comfort, an improved quality of life, while reducing energy consumption. The objects of a smart home can also interact directly with the inhabitant, resulting in modifications, observable by the services, of the values of the states of the actuators and sensors, thus allowing feedback and adaptation by these services.

The logic of the services deployed in a smart home can be designed by Artificial Intelligence approaches of symbolic type (modeling of state changes in the form of rules) or numerical type (selection of state values by learning techniques).

Nevertheless, neither of the approaches is totally satisfactory since the symbolic approach faces limitations of dynamic adaptability (the rules of behavior are predefined at design time) and the numerical approach faces limitations of explicability (the logic of decision making is not explicit).

This thesis presents four proposals that aim to overcome these limitations by using a hybrid approach combining learning techniques and predefined rules.

The first proposal is the use of reinforcement learning to create a dynamic service in two phases : first a pre-training phase in a simulated environment based on typical state configurations and a rule-based profile of the inhabitant, then a deployment phase in a real environment where the behavior is refined following the feedback from the inhabitant.

The second proposal generalizes the first one to the case of several services having possible conflicts when they try to change the states of shared actuators. Several options are studied depending on whether the learning is done globally for all services or individually per service and then the conflicting decisions are reconciled.

The third proposal is the extraction of rules from created dynamic services by linking an analysis phase (identification of each situation encountered by the service into an instance rule) and a synthesis phase (refinement of the set of extracted rules with the instance rule).

The fourth proposal is the evaluation of the hybrid architecture where the logical behavior of the deployed services comes from either predefined rules, or created dy-

namic services driven by reinforcement learning, or rules extracted from these created dynamic services.

These four proposals have been evaluated in several simulated comparative environments on temperature control, light intensity, and air quality services. In general, the experiment results demonstrate that learning systems can satisfactorily create conflict-free services. In addition, they prove that the proposed rule extraction method can well extract rules from learning systems. Finally, they show the advantages of integrating predefined and extracted rules into the decision-making of learning systems.

This thesis work can be extended according to several perspectives. For example, the co-simulation of physical phenomena would allow a more precise consideration (heat transfer between spaces) and a better anticipation of the evolution of states of the environment. The integration of various inhabitant profiles during the learning process would allow a dynamic adaptation of the services to various inhabitant profiles. Taking into account the energy cost of the deployed services would allow the implementation of energy saving policies.

Résumé

Les maisons intelligentes sont des maisons équipées d'objets connectés (capteurs pour récupérer des valeurs d'état de l'environnement, actionneurs pour influencer ces valeurs d'état) et de services qui interagissent avec ces objets dans le but d'offrir à l'habitant plus de confort, une meilleure qualité de vie, tout en réduisant la consommation d'énergie. Les objets d'une maison intelligente peuvent également interagir directement avec l'habitant, entraînant des modifications, observables par les services, des valeurs des états des actionneurs et des capteurs, permettant ainsi un retour d'information et une adaptation par ces services.

La logique des services déployés dans une maison intelligente peut être conçue par des approches d'Intelligence Artificielle de type symbolique (modélisation des changements d'état sous formes de règles) ou numérique (sélection des valeurs d'état par des techniques d'apprentissage).

Néanmoins, aucune des approches n'est totalement satisfaisante puisque l'approche symbolique se heurte à des limitations d'adaptabilité dynamique (les règles de comportement sont prédéfinies à la conception) et l'approche numérique à des limitations d'explicabilité (la logique de la prise de décision n'est pas explicite).

Ce mémoire de thèse présente quatre propositions qui visent à lever ces limitations en s'inscrivant dans une approche hybride combinant des techniques d'apprentissage et des règles prédéfinies.

La première proposition est l'utilisation de l'apprentissage par renforcement pour créer un service dynamique en deux phases : une phase de pré-entrainement en environnement simulé à partir de configurations d'états types et de profil de l'habitant basé sur des règles, suivie d'une phase de déploiement en environnement réel où le comportement est affiné suite aux retours de l'habitant.

La deuxième proposition généralise la première au cas de plusieurs services ayant éventuellement des conflits quand ils essaient de changer les états des actionneurs partagés. Plusieurs options sont étudiées selon que l'apprentissage s'effectue globalement pour tous les services ou individuellement par service puis les décisions contradictoires sont réconciliées.

La troisième proposition est l'extraction de règles des services dynamiques créés en enchaînant une phase d'analyse (identification de chaque situation rencontrée par le

service en une règle d'instance) et une phase de synthèse (raffinement de l'ensemble de règles extraites avec la règle d'instance).

La quatrième proposition est l'évaluation de l'architecture hybride où le comportement logique des services déployés provient soit de règles prédéfinies, soit de services dynamiques créés et dirigés par l'apprentissage par renforcement, soit de règles extraites à partir de ces services dynamiques créés.

Ces quatre propositions ont été évaluées dans plusieurs environnements comparatifs simulés sur des services de régulation de température, d'intensité lumineuse et de qualité de l'air. En général, les résultats des expériences démontrent que les systèmes d'apprentissage peuvent créer de manière satisfaisante des services sans conflit. De plus, ils prouvent que la méthode d'extraction de règles proposée peut bien extraire les règles des systèmes d'apprentissage. Enfin, ils montrent les avantages de l'intégration de règles prédéfinies et extraites dans la prise de décision des systèmes d'apprentissage.

Ce travail de thèse peut être prolongé selon plusieurs perspectives. Par exemple, la co-simulation des phénomènes physiques permettrait une prise en compte plus précise (transfert de thermique entre espaces) et une meilleure anticipation de l'évolution des états de l'environnement. L'intégration de profils d'habitant divers lors de l'apprentissage permettrait une adaptation dynamique des services à des profils d'habitant variés. La prise en compte du coût énergétique des services déployés permettrait de mettre en place des politiques de sobriété énergétique.

Contents

List of Figures

List of Tables

Part I

Introduction and background

1

Introduction

Contents

1.1 Introduction

From the appearance of home appliances in the early twentieth century to the emergence of smart homes or home automation in the 2000s, smart homes have received increasing attention from research and industry domains. Moreover, the introduction of the Internet of Things (IoT) [107] contributes to creating a smart home system that can be remotely controlled on a large scale.

To realize the intelligence of a smart home and meet the requirements of an inhabitant, we need to create various services [88], such as services to control the room temperature or adjust the room light intensity, and the definition of a smart home service varies depending on the context of the problems involved.

The creation of smart home services depends on various components containing diverse devices, including sensors to collect data and actuators to execute the actions proposed by services, the protocols to realize the data transmission, the inhabitants for whom services are created, and the developers who design services.

Based on these components, developing smart home services involves logical and

physical specifications. The logical specification involves the logical decision making by proposing states to actuators after having considered the environment state values sensed by sensors, and the physical specification concerns the hardware part, which includes how to collect data from the environment, how to transmit data to services using different protocols and how to deploy the smart home system in the real world. In our study, we only focus on the logical specification by developing a logic of decision-making to realize the creation of smart home services. Such logic is mainly based on either knowledge-based or data-driven approaches.

In this chapter, we first present the context of the smart home. Then, we describe the research questions that we try to address in the thesis. Next, we present the principal contributions that we have made during the thesis. Finally, we provide an outline of the entire document.

1.2 Smart home

A smart home was just an idea when it was first mentioned, and only existed in science fiction. With the development of various technologies, this idea became a reality. For example, according to [8], in 1966, Westinghouse engineer Jim Sutherland created ECHO IV. As the first true home automation device, ECHO IV controls temperature and appliances, allowing inputting and retrieving shopping lists, recipes, and other family memos. In 1969, with the introduction of ARPANET, the precursor to the Internet we know today, the truly connected universe was ushered. In 1975, the X10 Home Automation Project was introduced. We got into the territory of practical devices for real homes. In 1980s, motion-sensing lights, automatic garage door openers, programmable thermostats, and security systems were commonplace and affordable. In 1984, the term "smart house" was coined by the American Association of Home Builders. In 1990, John Romkey and Simon Hackett created a toaster connected to and controlled through the Internet. The Internet of Things (IoT) [79] was born. Nevertheless, this term was named by Kevin Ashton another nine years later. Since the 2000s, smart devices and systems have rapidly evolved with the help of the IoT. According to a research report from the IoT analyst firm Berg Insight [9], the number of smart homes in Europe and North America reached 105.0 million in 2021. The most advanced smart home market is North America, having an installed base of 51.3 million smart homes at the end of the year 2021.

With the increasing popularity of a smart home, different definitions are proposed. For example, [85] specifies that a smart home integrates different services using a common communication system, assures an economic, secure, and comfortable home operation, and includes a high degree of intelligent functionality and flexibility. [10] specifies that a smart home is an application of ubiquitous computing where the home environment is monitored by ambient intelligence to provide context-aware services and facilitate remote home control. [19] defines a smart home as a residence equipped with a communication network, linking sensors, domestic appliances, and devices. It can be remotely monitored, accessed, or controlled and provide services that respond to the needs of its inhabitants. [113] emphasizes smart home technologies used to realize a smart home. It describes that smart home technologies refer to devices that provide some degree of digitally connected or enhanced services to occupants, and are often synonymous with "home automation systems". [114] describes a smart home as a private house with many smart home automation devices. The communication of these devices can create new services and additional benefits for an inhabitant. [127] defines a smart home as a home that can proactively change its environment to provide services that promote independent living for the elderly. [89] defines a smart home as a dwelling equipped with smart technologies that provide customized services to the inhabitant.

From the above examples on the definitions of a smart home, we can observe that the intelligence of a smart home is realized by creating various services to meet the needs of an inhabitant.

Depending on contexts and problems, different definitions of a service are proposed. For example, in service-oriented architecture (SOA), a service is a self-contained software unit designed to perform a specific task. In other words, a set of procedures or software components designed to perform a specific task can be regarded as a service [99]. In the field of the Web, [18] defines a Web service as a distributed component that is loosely coupled and reusable, encapsulate discrete functions, and is accessible via standard Internet protocols.

In the field of the IoT, some work also introduces different definitions for a service. For example, in [86], a smart city service can fulfill a goal by performing a set of functions, and its characteristics describe the way the service interacts with urban resources and other services, the impact it has on the environment and people, and

the requirements for the infrastructures it uses. In some knowledge representation models [81], the concept of a service is also included. For example, The SAREF (Smart Applications REFerence) ontology [15] defines that a service is provided by a device and represents certain function of that device. It is used so that the functions of that associated device can be discovered, registered, and remotely controlled by other devices on the same network. In [35], a service is represented as a tuple containing both functional and non-functional properties associated with that service. The same author expresses the same definition of a service in [34], namely that any device can be considered as a service, e.g., a TV set is a TV service. In [115], a service is implicitly expressed as an action that can be performed if certain conditions are met. In [86], a service is the target of an ensemble of functions that can be triggered under certain conditions.

In our study, we define that, to realize the intelligence of a smart home, each service tries to control one specific state, called the monitored state, by proposing states for associated actuators after considering environment states sensed by sensors. For example, a temperature service controls the monitored indoor temperature by adjusting the related curtain, window, and air conditioner after considering the inhabitant's state and the indoor and outdoor temperatures. A light intensity service controls the monitored indoor light intensity by adjusting the associated curtain and lamp after considering the outdoor light intensity and the inhabitant's state.

To create smart home services, both communication and decision-making are involved. Communication concerns collecting data from the environment and transmitting it to services using different protocols. Decision-making involves proposing states for related actuators by considering the environment state values detected by sensors. Our study only focuses on logical decision-making to create smart home services.

Knowledge-based and data-driven approaches are two primary categories of approaches to creating services by designing logical decision-making. In knowledge-based approaches, the created services are explicable. When we say that services are explicable in our study, we means that they can show the inhabitant under which situations they have proposed certain actuators' states. In data-driven approaches, the created services can dynamically propose actuators' states to adapt to the inhabitant's preferences by considering the constantly changing environment states. Furthermore, RL (Reinforcement Learning) [48], whose basic idea is that an artificial agent learns

the system's behavior patterns by interacting with the environment, can create data-driven services that consider the inhabitant's reactions to the actions proposed by the services to find out the inhabitant's preferences. And it is essential to consider the inhabitant's reactions when attempting to design a user-friendly smart home system [36].

1.3 Research questions

Despite some advantages mentioned above for knowledge-based and data-driven approaches, neither of the two types of approaches can create smart home services satisfactorily. For knowledge-based approaches, the services created are usually static and cannot adapt to the changing environment and the changing inhabitant's preference.

For data-driven approaches, however, the created data-driven services are like black boxes that constantly update themselves to adapt to the changing environment states and the changing inhabitant's preference. Therefore, the inhabitant has no idea under which situations certain services have suggested certain actions.

Considering the above brief analysis of using data-driven and knowledge-based approaches to create smart home services, we propose the following research questions (**RQ.1** $\sim$ **RQ.4**) to be addressed to create services. The created services will be explicable and can dynamically propose actuators' states to achieve the inhabitant's target monitored states.

RQ.1 How can we address the drawbacks of existing methods to create a dynamic smart home service to adapt to the changing environment and the changing inhabitant's preference?

Existing methods to create smart home services have several drawbacks. For example, they usually require manual inputs from the inhabitant, and the manual inputs can be complex when the physical phenomena of the environment states are complex and the inhabitant has no idea how to adjust the actuators to achieve his/her target monitored states. In addition, the services are usually created during design time, and cannot evolve to adapt to the changing environment and the changing inhabitant's preference. To address these problems, we start with the simplest smart home

system, where there is only one single service, and try to create a dynamic smart home service that can adapt to the changing environment and the changing inhabitant's preference.

RQ.2 How can we extend the proposal of creating one single dynamic smart home service to a more general situation of creating multiple dynamic smart home services that are adaptive and conflict-free?

There are usually multiple services within a smart home. Including the drawbacks mentioned in **RQ.1**, when there are multiple services, potential conflicts can be generated if these services simultaneously access the same actuators. Nevertheless, it is complex for existing methods to create multiple dynamic smart home services that can adapt to the changing environment and the changing inhabitant's preference while ensuring no conflicts among these services. To address these problems, by extending the proposal to **RQ.1**, we try to realize the creation of multiple dynamic smart home services which are adaptive and conflict-free.

RQ.3 How can we make the created dynamic smart home services explicable?

Existing methods to create dynamic smart home services are like black boxes, and users have no idea of the logic of the behavior of the created services. Nevertheless, knowing the logic of the behavior of the created services is important for system maintenance and security. Therefore, we try to propose methods to make the created dynamic services explicable.

RQ.4 How can we create dynamic smart home services that are adaptive, conflict-free, and explicable?

In a smart home, it is essential that services are both dynamic, to adapt to the changing environment and the changing inhabitant's preference, and explicable, so that users are aware of the logic of the services' behavior. Nevertheless, little existing work is able to create services that are dynamic while explicable. Therefore, we try to propose a solution to create a user-friendly smart home with conflict-free services. These services can dynamically adapt to the changing environment, improve their performance by considering the inhabitant's preferences or reactions, and explain to users in which situations certain actuators' states are proposed.

1.4 Main contributions

To create smart home services which are explicable and can dynamically propose conflict-free actuators' states to adapt to the inhabitant's preferences for monitored states, we try to address the above four research questions by making the following contributions :

1. **Creation of a single dynamic smart home service that is adaptive.**
   Considering the simplest smart home with only one service, we propose an RL-based structure to create the corresponding single smart home service. The created service can dynamically propose actuators' states to adapt to the changing environment and meet the inhabitant's preferences.
2. **Creation of multiple dynamic smart home services which are adaptive and conflict-free.**
   We propose several multi-services based architectures based on the proposed RL-based single service structure. The created multiple services inherit the advantages of an RL-based single service to dynamically propose actuators' states to adapt to the changing environment and the changing inhabitant's preference. Furthermore, they are conflict-free even though they simultaneously work on the same actuators.
3. **Extraction of explicable smart home services from dynamic and adaptive smart home services.**
   We propose an algorithm to extract knowledge-based services from dynamic data-driven services. Therefore, the data-driven services will be explicable using the extracted knowledge-based services to demonstrate to users under which situations certain actuators' states are proposed.
4. **Creation of dynamic smart home services which are adaptive, conflict-free and explicable.**
   We propose a system that combines the proposed multi-services based architectures and the proposed algorithm to extract knowledge-based services. This system can create dynamic services that are conflict-free, explicable, and able to adapt to the changing environment states and the changing inhabitant's preference.

1.5 Outline

The document is divided into three parts. The first part introduces the context, the research questions, the existing solutions to address the questions, and the analysis of the existing solutions. The second part presents the contributions to solve the questions mentioned in the first part. The third part concerns the conclusion, the discussion, and the perspective.

The first part includes chapters 1∼4. Chapter 1 is the current chapter that introduces the context related to the smart home and smart home services, the research questions, the contributions, and the outline of the document. Chapter 2 presents the definition of using knowledge-based approaches for service creation, the components of knowledge-based approaches, and the existing work concerning services based on this category of approaches. Chapter 3 contains the definitions of using data-driven approaches for service creation, the introduction of the data-driven algorithms used in our work, and the existing work concerning service creation and some other applications based on data-driven approaches. Chapter 4 analyzes knowledge-based and data-driven approaches for service creation. Based on this analysis, we consider hybrid systems based on hybrid approaches that combine knowledge-based and data-driven approaches for service creation. In addition, we present existing work on using hybrid approaches for service creation and compare it to the requirements that we believe a hybrid system should have so as to create smart home services in a satisfactory manner.

The second part contains chapters 5∼8. Chapter 5 introduces the concept of a simple smart home system, proposes how to use RL to create a single dynamic smart home service, how to deploy a smart home service in the real world, and how to simulate a smart home system. By reusing the proposed structure of a single dynamic smart home service, several architectures are proposed in Chapter 6 to create multiple services that can dynamically propose conflict-free actuators' states. Several experiments are conducted to evaluate the performance of these architectures and select those with better performance. Chapter 7 presents the definition, the motivation, and the related work on extracting rules from trained learning systems. It also explains our proposed rule extraction method and evaluates and demonstrates its satisfactory performance in several experiments. Moreover, a variant of this method is proposed to dynamically extract rules even when the learning systems are not well trained. Chap-

ter 8 describes the proposed hybrid system based on our proposed hybrid approaches that combines knowledge-based and data-driven approaches to create smart home services that are dynamic, explicable and conflict-free. It also shows the better performance of this hybrid system compared to the other two systems. The first system only contains the data-driven approaches presented in Chapter 6. The second system includes pre-existing rules and the same data-driven approaches.

The third part contains Chapter 9 and Annexes. Chapter 9 describes the main results of our entire work, the discussion of some problems to be solved, and the perspective including interesting research directions to be explored. Annexes contain the acronyms, the variables, and the main glossaries that appear in the document.

2

Knowledge-based approaches for service creation

Contents

2.1 Introduction

Knowledge-based approaches are one of the most important categories of approaches for the development of smart home services. Systems built using this type of approaches include a knowledge representation and an inference engine, where the knowledge representation consists of a set of facts and rules. To create services, knowledge-based approaches can be used. Due to the concise and clear format of the rules, the created services are usually explicable.

In a smart home, there are numerous objects, such as sensors and actuators, and various services. Knowledge-based approaches can express the relationship between objects, between services, and between objects and services. In addition, rules are used to create knowledge-based smart home services, and the created services can

show users in which situations certain states of the actuators are proposed. Moreover, the inhabitant can easily create knowledge-based smart home services if he/she knows how to set the states of the actuators.

Knowledge representation is one of the most important components of knowledge-based approaches. Ontology is one of the most popular methods for knowledge representation. Many ontologies are proposed for different domains. One of the well-known ontologies for the smart home domain is the SAREF (Smart Appliances REFerence) ontology. It allows the separation and recombination of ontologies from different domains. SAREF is also the knowledge representation to be leveraged in our contributions.

We will leverage this chapter to provide the basic principle of knowledge-based approaches and explain the knowledge representation we will use in our work. Therefore, in the rest of the chapter, we first introduce the definition and components of knowledge-based approaches. Then, we describe the SAREF ontology in detail. Next, we present existing work that uses knowledge-based approaches to deal with services related problems and applications. Finally, we conclude this chapter.

2.2 Knowledge-based approaches

A Knowledge representation and an inference engine are two main components of a system implemented by a knowledge-based approach. The knowledge representation contains a set of facts and rules [4]. Facts are known to be true in certain domains, such as the smart home, and can be implemented using, for example, ontologies [80]. Rules can be viewed as extensions of facts with additional conditions that must be satisfied for them to be true [2]. An inference engine, also called a reasoner, applies rules to the set of known facts and derives new facts from it.

2.2.1 Ontology

Ontology is a knowledge representation method that tries to realize knowledge exchange between different domains. It describes the semantics of data and provides a unified way of communication through which different parties can understand each other [119]. Thus, an ontology is a shared conceptual model of a formal specification [132] and can facilitate knowledge sharing across different domains by providing a common understanding [119]. As a result, it can be used to solve the interoperability

problem. In addition, it is also used in the Semantic Web [24] to provide additional constraints and annotations during knowledge sharing. An ontology mainly contains 4 elements : classes, instances, relations and axioms [119] where :

1. Classes or concepts are the fundamental elements of the ontology. They can be organized into a hierarchy of superclasses and subclasses, also known as a taxonomy. All classes in the same hierarchy have common properties.
2. Instances or individuals are the representations of certain real objects in practice and belong to certain classes or concepts. For example, "John" can be an instance of the class "Person", and "John" in real life represents a person named "John".
3. Relations, relationships, or properties are binary relations used to describe the relation between two classes or two instances. They can also be used to depict the characteristics of classes or instances. When we apply the cartesian product of two sets on the property, the first element is named a domain, and the second is called a range. We consider two kinds of properties : an object property is a relation having both the domain and the range be both classes or both instances, and a data property is a relation having the domain be an instance or a class and the range be some literal value. For example, "John hasAge 12" is a data property, and "John hasBother Jack" is an object property. In both examples, The domain "John" is an instance of the class "Person", and the range "12" is an integer not belonging to any class. In contrast, the range "Jack" is an instance of the class "Person".
4. An axiom is used to constrain classes, instances, and relations. For example, the subclass axiom can be expressed by an example : class "a :Student" is a subclass of the "class a :Person" to restrict the relationship between the two classes. Axioms are usually expressed in first-order logic (FOL) [112, 38] and are used to check the consistency of the ontology.

Several reference ontologies have been developed to cover the needs of different fields. In this section, we introduce one of the most popular ontologies, the Web Ontology Language (OWL) [93].

OWL

The W3C Web Ontology Language (OWL) is a Semantic Web language [76]. It is part of the W3C's Semantic Web technology stack which also contains RDF (Resource Description Framework), RDFS (RDF Schema), SPARQL, etc. Three increasingly-expressive sublanguages : OWL Lite, OWL DL (Description Logics), and OWL full are included in OWL. Any of them can be used to build an ontology.

OWL Lite

OWL Lite primarily provides classification hierarchy and simple constraints. It is a light version of OWL DL and only contains basic language features. Some requirements need to be met when using some properties in OWL Lite :

1. the subject of "owl :equivalentClass" needs to be class names and the object of "owl :equivalentClass" needs to be class names or restrictions.
2. the subject of "rdfs :subClassOf" should be class names and the object of "rdfs :subClassOf" should be class names or restrictions ;
3. "owl :intersectionOf" needs to be used on lists of length greater than one that contains only class names and restrictions ;
4. the object of "owl :allValuesFrom" and "owl :someValuesFrom" needs to be class names or datatype names ;
5. the object of "rdf :type" needs to be class names or restrictions ;
6. the object of "rdfs :domain" needs to be class names, and the object of "rdfs :range" needs to be class names or datatype names.

OWL DL

OWL DL is a description logic supporting data values, data types, and datatype properties. OWL DL and OWL Lite extend the RDF vocabulary, but also restrict the use of that vocabulary. The restrictions for OWL DL may be the following examples :

1. OWL DL requires a pairwise separation between classes, data types, datatype properties, object properties, annotation properties, ontology properties (i.e., the import and versioning stuff), individuals, data values, and the built-in vocabulary. Therefore, a class cannot be an individual at the same time.

2. OWL DL requires that no cardinality constraints (neither local nor global) can be applied to transitive properties, their inverses, or any of their superproperties.
3. Most RDF(S) vocabulary cannot be used within OWL DL.

OWL Full

OWL Full contains all constructs of the OWL language and allows free, unrestricted use of RDF constructs. In other words, it is designed for maximum RDF compatibility. The compatibility can be illustrated by the following examples :

1. The resource "owl :Class" is equivalent to "rdfs :Class".
2. OWL Full allows classes to be treated as individuals.
3. Object properties and datatype properties are not disjoint, which means that "owl :ObjectProperty" is equivalent to "rdf :Property". Therefore, datatype properties are effectively a subclass of object properties.

OWL Lite is a sublanguage of OWL DL. They have no significant difference. The main difference is between OWL DL and OWL Full. Specifically, OWL Full allows free mixing of OWL with RDF Schema and, like RDF Schema, does not enforce strict separation of classes, properties, individuals, and data values. In contrast, OWL DL restricts mixing with RDF and requires separating classes, properties, individuals, and data values. For example, in OWL Lite and OWL DL, "owl :class" is a separate subclass of "rdf :class", meaning that not all RDF classes are OWL classes. However, in OWL Full, they are identical.

2.2.2 Rules

Rules are usually described by the expression IF-THEN. For example, "if A then B", where "A" contains a set of conditions, also called antecedents, and "B" contains conclusions, also called consequences. Thus, the expression states that "if all the conditions in A are true, then the conclusions in B are true". Since the knowledge-based system is usually applied in the domain of FOL (First Order Logic), each antecedent and consequence in the rule expression is often a FOL formula or sometimes a function such as an n-ary function that accepts many arguments. Introducing rules into the knowledge base helps to improve the ontology. It allows the ontology to describe relationships that cannot be described in DL (Description Logic) [72]. Moreover, rules, just like ontology, allow the exchange and reuse of knowledge between

different systems. Several rule languages have been proposed, such as FOL RuleML (First-Order Logic RuleML), SWRL (Semantic Web Rule Language), Notations3, Jena rules, ECA (Event-Condition-Action) rules, and IFTTT(If-This-Then-That).

(1) FOL RuleML [26] was developed as the central language of RuleML [27]. It combines the Quantifier RuleML to introduce logical quantifiers and the Disjunctive RuleML to introduce logical operators. It is enriched by other connectives, such as classical negation. In addition, FOL RuleML also contains the "query" function. In FOL RuleML, each antecedent is called an atom, consisting of individuals and variables.

(2) SWRL [62] is a markup language based on the combination of two sublanguages of OWL (Web Ontology Language) with Unary/Binary datalog RuleML : OWL DL and OWL Lite [92]. It extends the set of OWL axioms with horn-like rules, which allow the use of horn-like rules in a knowledge base with OWL. An XML syntax is provided to describe these rules based on the syntax of RuleML and OWL XML. In addition, an exchange syntax OWL RDF/XML enables the use of a Resource Description Framework (RDF) syntax in the descriptions of SWRL rules. The logical operators and quantifications in SWRL are the same as in RuleML. Together with SAREF, SWRL is leveraged in our work.

(3) Notation3 [23] or N3 extends the RDF model to include rule formulas, variables, logical implications, and functional predicates, and provides a readable textual syntax as an alternative to RDF/XML. N3 logic has been intentionally restricted to limited expressive power : it currently does not include first-order negation. However, negated forms are available for many of the built-in functions.

(4) The Jena rule [100] can be described in RDF syntax and is designed to work with RDF graphs. Because Jena rules treat a returned parameter as another parameter in the parameter list rather than something applied to the left of the built-in parameter, it has a graph structure instead of the typical tree structures found in programming languages. Also, it can only be used by reasoners in the Jena framework [68].

(5) ECA rules [45] were developed to support the need to react to different kinds of events occurring in active databases [101]. An ECA rule has three parts [32] : An event, a condition, and an action. The event part specifies the signal that triggers the invocation of the rule. The condition part is a logical test that, if satisfied or

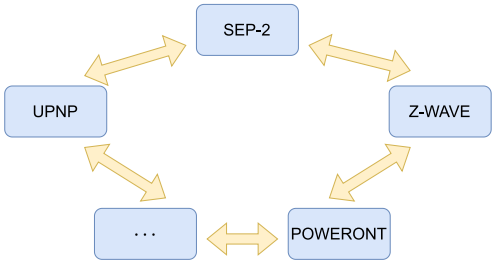

FIGURE 2.1 – Communication between different ontologies without SAREF

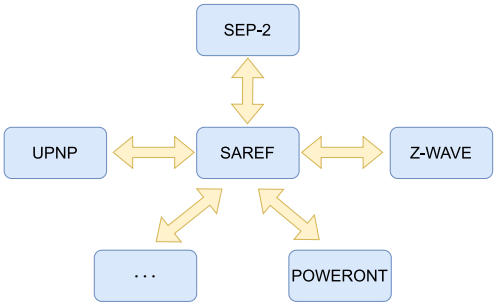

FIGURE 2.2 – Communication between different ontologies with SAREF

evaluated to be true, causes the action to be carried out. The action part specifies the actions to be taken on the data.

(6) IFTTT [5] is a popular trigger-action programming platform [7]. It is created by the IFTTT private commercial company to connect apps, devices, and services from different developers to trigger one or more automation involving those apps, devices and services [7]. The principle of IFTTT is that if certain conditions are met, then something else will happen. The "if this" part is called a trigger, and the "then that" part is called an action [6].

2.3 SAREF

The Smart Appliances REFerence (SAREF) ontology [1] was developed as a consensus model that facilitates the mapping of existing semantic models in the smart appliance domain and reduces the translation effort from one domain to another. It has an explicit definition of classes for the smart appliance, wide application, and large support community. Therefore, It is also the knowledge representation to be used in our work.

According to [15], before the introduction of SAREF, different semantic models have recurring base concepts, but they often use different terminologies and different data models to represent these concepts, as shown in Fig.2.1. With the introduction of SAREF, different semantic models can continue to use their own terminology and data models and relate to each other through the common semantics defined in SAREF, as shown in Fig.2.2. In the following sections, we introduce the important terminologies of SAREF.

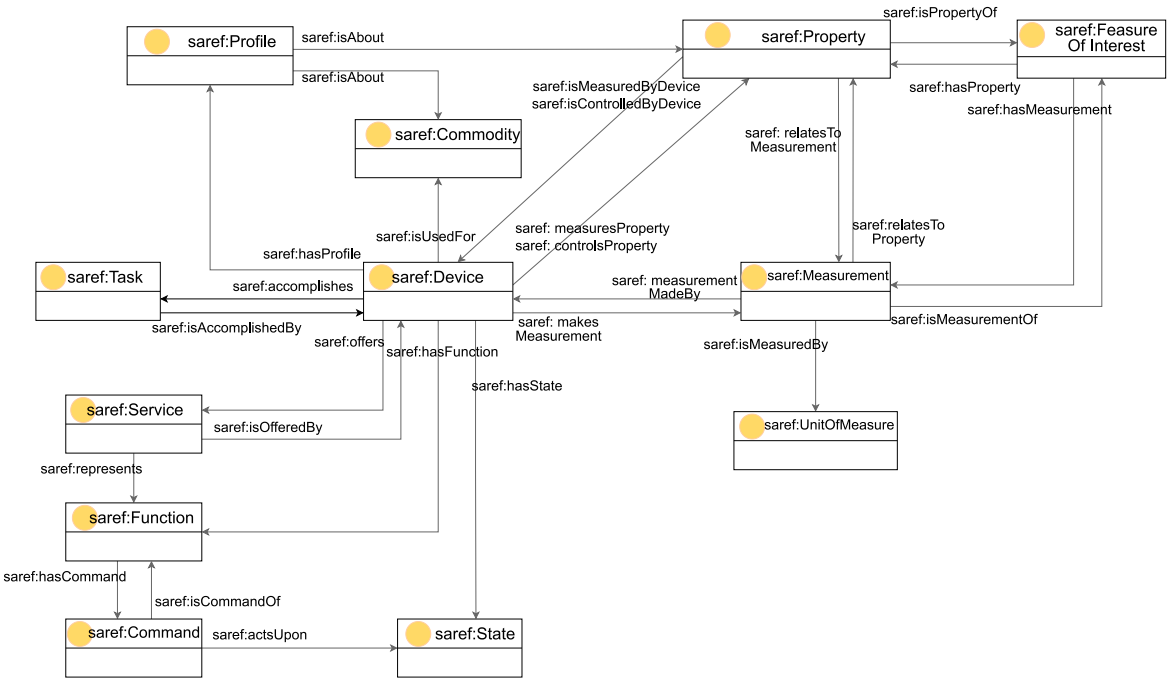

FIGURE 2.3 – General overview of SAREF

SAREF : Classes

The main concepts of the SAREF ontology are shown in Fig.2.3. The starting point of SAREF is the concept of the "Device" (e.g., a lamp). A "Device" is designed to accomplish certain tasks (e.g., increase/decrease the light intensity in a room) and is used to satisfy a marketable good, the "Commodity". In SAREF, the "Commodity" class typically refers to energy commodities (e.g., electricity, gas, coal, and oil). A "Device" may have multiple "Function"s (e.g., turn the lamp on/off) associated with a "Command" (e.g., turn on/off). Different "Function" can cause the "Device" to be in different "State" (e.g., on/off) through the corresponding "Command". In addition, a "Device" may provide a "Service" (e.g., turn on service) that represents the "Function"s of that "Device". The definition of a "Service" does not match the one in Section 1.2, since our definition refers to an attempt to control a particular state by performing various associated functions. Therefore, it is more correct to say that a service can be implemented by leveraging different functions of the associated devices.

Furthermore, a "Device" is used to detect, measure, or control a "Property" (e.g., light intensity) that corresponds to the monitored state controlled by our defined service; the value associated with this "Property" is the "Measurement". When the "Measurement" is described in terms of a unit of power or energy, that property is

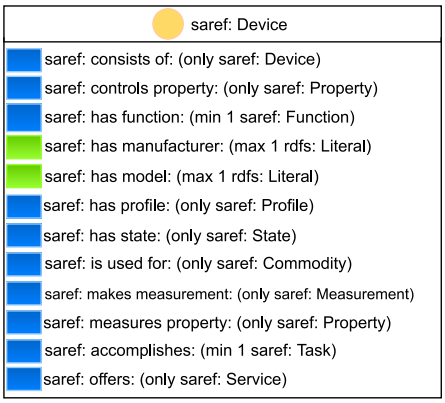

FIGURE 2.4 – Properties of class "Device"

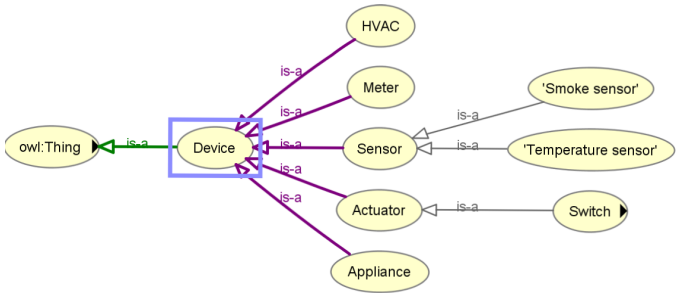

FIGURE 2.5 – SubClasses of class "Device"

a type of power or energy property. A "Device" also has an associated "Profile" that collects information about the "Property" or "Commodity" to optimize its use. Each class can have multiple instances representing the objects in real life or in simulation. In the following sections, we present some important classes in detail.

Device

The "device" is the central class in SAREF. It is a tangible object that, in the context of smart devices, is designed to perform a specific task in households, shared public buildings, or offices. Fig.2.4 shows us the properties of a "Device". The object property is in blue and the data property is in green. In this figure, we can also see the constraints defined for the ranges of some properties. Fig.2.5 shows us the subclasses of the "Device" class. We can see that "Device" has five subclasses : "HVAC", "Metre", "Sensor", "Actuator" and "Appliance". The subclasses "Sensor" and "Actuator" also have subclasses. The relationship between a class and its subclass is expressed by the "is-a" property.

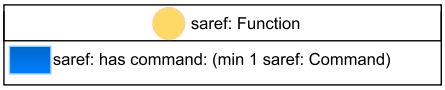

FIGURE 2.6 – Properties of class "Function"

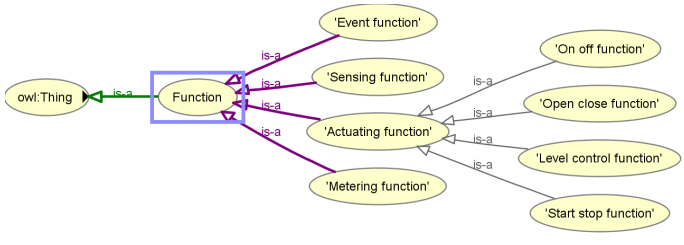

FIGURE 2.7 – SubClasses of class "Function"

Function

The "Function" class contains functionalities required to perform a task for which a "Device" is designed. A "Device" may be designed to perform multiple "Function"s. "Functions"s can be divided into several categories (subclasses) reflecting different viewpoints. For example, the viewpoint of the area for which the function of an application is used (light, temperature, etc.), or the action that the function of an application can support (e.g., receive, reply, notify, etc.), etc. Fig.2.6 shows us the object property of "Function". The subject of "hasCommand" is a Function, and the range is "Command". "hasCommand" shows us which "Function" can be ordered by which "Command". In Fig.2.7 we can find available subclasses of "Function" defined in SAREF, where the subclass "Actuatoring function" has several subclasses.

Command

A "Command" can act on a state, but it does not necessarily have to act on a state. For example, the command ON acts on the state ON/OFF, but the command GET does not act on a state. It merely attempts to retrieve a specific value. A list of relevant "Command"s for SAREF is available and can be expanded. Fig.2.8 shows us

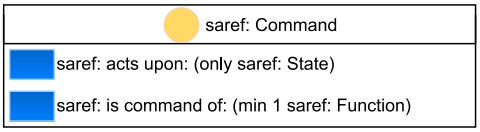

FIGURE 2.8 – Properties of class "Command"

FIGURE 2.9 – SubClasses of class "Command"

the properties of the "Command" class, which contains two object properties. Fig.2.9 shows us all available subclasses of "Command" in SAREF, where "Get command" and "Set level command" also have their subclasses.

Service

A "Service" is a representation of one or more "Function"s that makes those "Function"s discoverable, recordable, and remotely controllable by other devices on the network. It is provided by one or more "Device"s. When describing a "Service", the "Device"s providing the "Service" and the associated "Function" must be specified. In Fig.2.10, the properties of a "Service" are shown. We can see that two object properties are included with the constraints on the ranges. Fig.2.11 shows an example of subclasses of "Service" - "Switch on service". This service can be provided by devices with the "Switch on" functions.

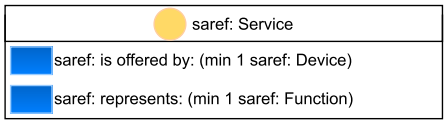

FIGURE 2.10 – Properties of class "Service"

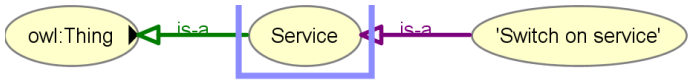

FIGURE 2.11 – SubClasses of class "Service"

State

Depending on the "Function"s performed, a "Device" may be in a particular "State". The subclasses of the "State" can be found in Fig.2.12, from which it can be seen that the "State" has not only binary states, but also n-nary states, e.g. the "Multi level state".

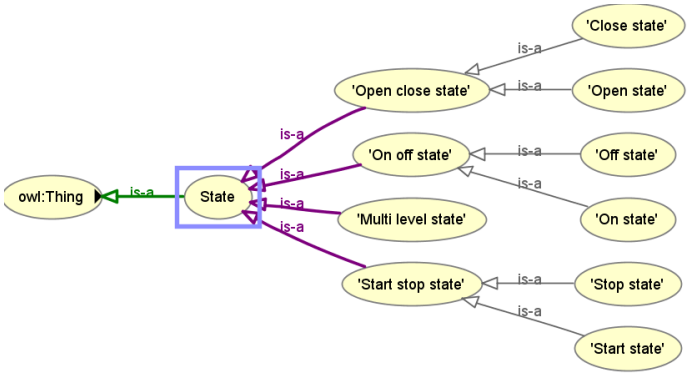

FIGURE 2.12 – Subclasses of class "State"

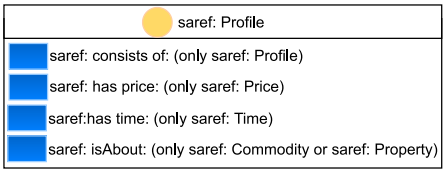

FIGURE 2.13 – Properties of class "Profile"

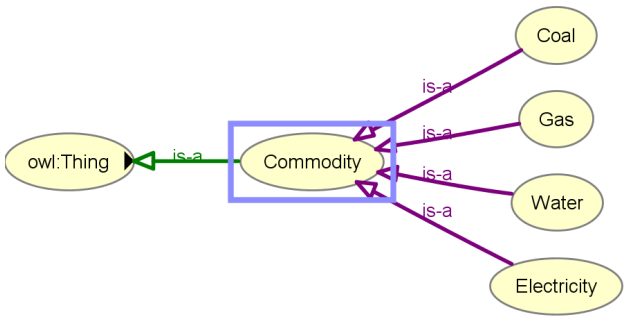

FIGURE 2.14 – Subclasses of class "Commodity"

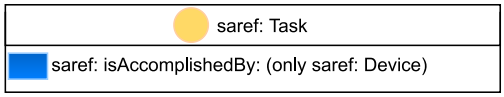

FIGURE 2.15 – Properties of class "Task"

Profile

The "Profile" class is associated with a "Device" to collect information about a particular "Property" or "Commodity" to optimize the use of a device in the home/building. The "Profile" is related to a specific "Property" or "Commodity" and can be calculated over a period of time and be associated with specific costs. Fig.2.13 shows us the available properties of the "Profile".

Commodity

The "Commodity" class can represent a marketable item for which there is demand, and is supplied without difference in quality across a market. SAREF focuses especially on energy commodity such as "Coal", "Electricity", "Gas", and "Water" as shown in Fig.2.14. The "Device" class has a relation with "Commodity" through the object property "isUsedFor" as shown in Fig.2.4.

Task

The "Task" class describes the goal for which a device is designed (from the user's point of view). For example, an air conditioner is designed for the task of changing the temperature of the environment. SAREF has provided instances of the class "Task" instead of subclasses, but we can create the subclasses according to our needs. The "Device" class connects to the "Task" class through "Accomplishes", as shown in Fig.2.4. The "Task" class connects to the "Device" class via the object property "isAccomplishedBy"as shown in Fig.2.15.

Property

The "Property" class represents a quality of a feature of interest that can be measured. Its available subclasses can be found in Fig.2.17. In addition, it uses "isControlled-

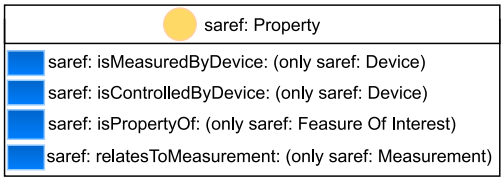

FIGURE 2.16 – Properties of class "Property"

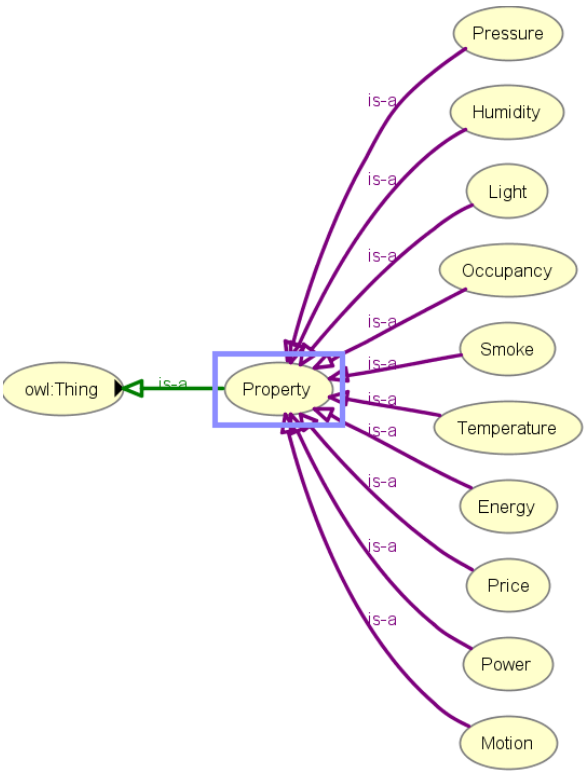

FIGURE 2.17 – Subclasses of class "Property"

ByDevice" and "isMeasuredByDevice" to connect to the class "Device", "isPropertyOf" to connect to the "Feature Of Interest" class, and "relatesToMeasurement" to connect to the "Measurement" Class. These relationships can be seen in Fig.2.16.

Feature Of Interest

The "Feature Of Interest" class represents a real world entity from which a property is measured. For example, a lamp can be defined as a feature of interest, and its property can be the light which will be measured by the light intensity belonging to the "Measurement" class. In SAREF, there are no predefined subclasses for "Feature Of Interest. However, the "Feature Of Interest" links to the "Property" through

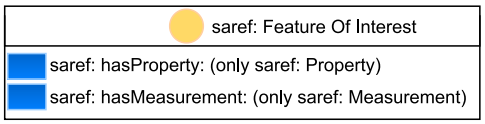

FIGURE 2.18 – Properties of class "Feature Of Interest"

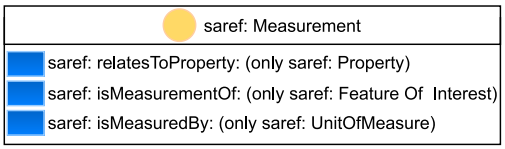

FIGURE 2.19 – Properties of class "Measurement"

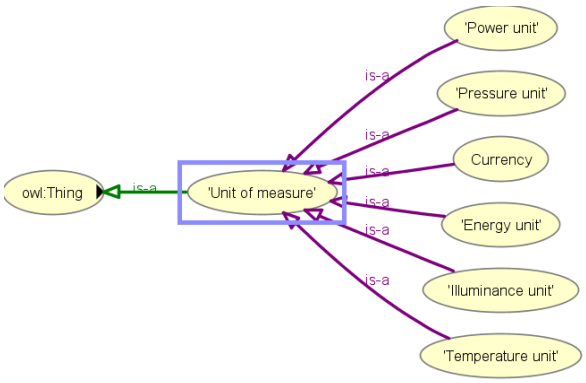

FIGURE 2.20 – Subclasses of class "Unit Of Measure"

"hasProperty", and links to the "Measurement" through "hasMeasurement" as shown in Fig.2.18

Measurement

The "Measurement" class measures the value of a "Property". In SAREF, there are no subclasses predefined for the "Measurement" class. Nevertheless, the "Measurement" class has relationships with several other classes. For example, it uses "relatesToProperty" to link to the "Property" class, "isMeasurementOf" to link to the "Feature Of Interest", and "isMeasuredBy" to link to the "Unit Of Measure" as shown in Fig.2.19.

Unit Of Measure

The "Unit Of Measure" class is a standard for measurement of a quantity, such as a "Property". For example, Light intensity is a "Property" and illuminance (lux) is a unit of light intensity that represents a definite predetermined light intensity. When we say 100 lux, we actually mean 100 times the definite predetermined light intensity called "illuminance (lux)". SAREF has defined several subclasses for the "Unit Of Measure" class as shown in Fig.2.20. Nevertheless, there is only one relationship to connect the "Measurement" class to "Unit Of Measure" class, while no relationship coming out from the "Unit Of Measure" class.

2.4 Existing work

Several knowledge-based work concerning services has already been carried out. Some of the existing work aims to solve problems that may exist when creating ser-

vices. For example, [17] proposes the concept of "cluster" to group services that share the same actuators, or services that share actuators with the same services aiming at avoiding potential conflicts. Within each cluster, some priorities are defined for different services, and the determination of the states of the shared actuators depends on these priorities and the objective to maximize the number of services to be satisfied. [98] proposes a tool called TrigGen to avoid errors caused by too few triggers in rules manually created by users. Using this way, we can avoid unexpected behavior and security vulnerabilities caused by custom rules with missing triggers, for example, during the service development. [134] proposes a layered architecture of smart home that combines ontology and multiagent technologies to automatically acquire semantic knowledge and support heterogeneity and interoperability services to address several issues, such as device heterogeneity, composite activities recognition, and providing appropriate services. In such architecture, a generic inference algorithm is presented based on unordered actions and temporal property of activity for inferring both continuous composite activity and personalized service in real time. Then a novel idea is introduced for agent to learn the knowledge of human activity (HA) autonomously and translate into itself knowledge, the purpose of which is to guide agent in performing services in a way compatible with HA.

Some other existing work tries to use knowledge-based approaches to realize the service creation or service-based applications. For example, [51] proposes a platform called Midgar consisting of several layers to create services. First, the inhabitant defines his/her requirements in the process definition layer in a graphical format, which are then transformed into an XML-based DSL (Domain Specific Language). Then, in the service generation layer, a Java application is generated, compiled and executed based on the DSL specification. During compilation, the Java application communicates with the servers and database in the processor and object manager layer to find available objects that can satisfy the requirements. Finally, in the object layer, the objects execute the commands received from the servers. [87] proposes a smart home system containing two levels to create services : a low level and a high level. The low level includes the execution of various devices, while the high level includes system modeling, rule transformation, and system reasoning. System modeling uses the semantic web to model the environment and ECA-based descriptions to simulate the operation of services. Rule transformation is about converting ECA into executable semantic rules, e.g., SWRL (Semantic Web Rule Language). And reasoning is about

deriving new facts based on SWRL. These new facts are converted into commands and sent to the devices for execution. [31] respects the following procedures to create new services. First, it describes each device with Vorto-DSL containing functional and non-functional features. Then, the matcher analyzes the functionalities and interfaces of the devices based on the Vorto-DSL to determine if they can be integrated. After the matching process, the functionalities of the matched devices are merged to assign the trigger and action. This process is performed by the Mapper. Finally, the integrated service is translated into a programming language for execution. [47] aims to renew the role of the station by improving the passenger experience by providing access to real-time train information and external mobility services through STINGRAY project to overcome the disadvantage that traditional station information systems tend to be closed to the outside world. For the provision of external mobility services, It presents a distributed software architecture that can integrate interfaces from heterogeneous infomobility service providers using an ontology describing the main characteristics of mobility domains.

2.5 Conclusion

Knowledge-based approaches are one of the most important approaches for the development of smart home services. A knowledge-based system usually consists of a knowledge representation and an inference engine. The knowledge representation contains various facts and rules. The creation of smart home services using knowledge-based approaches is to establish a set of rules. There are several types of knowledge representations, of which ontology is one of the most well-known, and OWL is one of the well-known ontologies designed for the Semantic Web. Since our study focuses on the smart home, despite the numerous ontologies, we choose to use the SAREF ontology in our work, which facilitates the mapping of existing semantic models in the smart appliance domain and reduces the effort to convert one model to another. In the next chapter, we present another category of approaches for creating smart home services.

3

Data-driven approaches for service creation

Contents

3.1 Introduction

Data-driven approaches are another primary category of approaches to creating smart home services. The services created suggest states for actuators through data analysis and interpretation. Machine learning methods that learn from data and make predictions based on it are at the forefront of data-driven decision-making [30]. They seek to automatically discover patterns in systems by analyzing the datasets provided. Therefore, machine learning methods in a smart home can create services that can dynamically propose states for actuators by considering the environment states to adapt to the inhabitant's preferences.

User-friendliness is important when creating services for a smart home, and interaction is one of the essential factors in creating user-friendly services [59]. Some

machine learning methods, such as Reinforcement Learning (RL) [48], can learn to adjust actuators by interacting with the inhabitant, and the interaction of RL is implemented by expressing the (dis)satisfaction of the inhabitant and can be realized in two ways. The first way is that the inhabitant can change the states of the actuators to express his/her (dis)satisfaction. The second way is that the inhabitant can define the target monitored states so that the service can compute the difference between the updated monitored states and their corresponding target values. The difference can represent the (dis)satisfaction of the inhabitant.

RL is used to learn closed-loop control policy [58]. It is different from closed-loop control systems in several aspects. For example, it is not mandatory to define the target values for the RL algorithm as the RL algorithm can interact with the environment or with the inhabitant to find out the patterns of the system. However, target values or references are usually obligatorily required in closed-loop control systems. Second, one RL algorithm can make propositions to adapt to the inhabitant's multiple preferences regarding different inhabitant states. In other words, one RL algorithm can learn multiple targets simultaneously. For example, an RL algorithm can propose actuators' states to adapt to the preferences specifying that if the inhabitant is reading, then the indoor light intensity should be 200 lux ; if the inhabitant is working, then the indoor light intensity should be 250 lux ; if the inhabitant is seeing a movie, then the indoor light intensity should be 50 lux ; and so on. However, one closed-loop control system can only meet one target at any time by tuning its parameters. If it wants to meet another target, it needs to retune its parameters, and the updated closed-loop control system can no longer meet the previous target. For example, instead of proposing actuators' states to meet the above multiple preferences simultaneously, a closed-loop control system can only meet one preference at any time. Third, the "feedback" of the RL algorithm is the reward that can be in different forms. For example, it can be the difference between the sensed monitored state and its corresponding target value. It can also be the reactions of the inhabitant, e.g., changing the actuators' states proposed by the RL algorithm by the inhabitant to express the negative reward. Nevertheless, the feedback of closed-loop control systems can only be the difference between the sensed monitored state and its corresponding target value.

Because of the capability of interacting with the inhabitant, the support of multiple target values, and the existence of different ways, especially the simple and natural

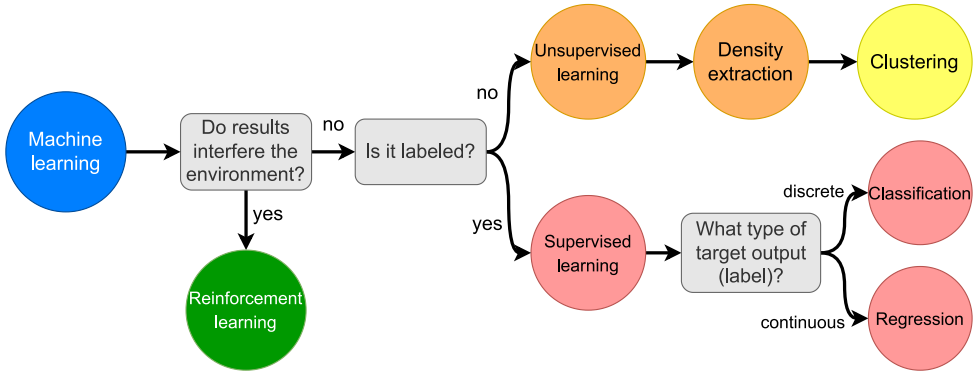

FIGURE 3.1 – Machine learning classification

way to provide the "feedback", RL is chosen as the data-driven approach in our study for creating smart home services.

In this chapter, we first introduce the existing classification of machine learning algorithms. Then, we present the classification of RL that we use in our study. After that, we introduce the principle of RL. Next, we present existing work on using data-driven algorithms to realize service creation and some other applications. Finally, we present the conclusion of this chapter.

3.2 Machine learning

Machine learning is a subfield of artificial intelligence [75]. It aims to develop methods for discovering patterns in provided data and then using those patterns to automatically make predictions about future data from the same source or with the same distributional characteristics [97]. Therefore, machine learning is about predicting the future based on the past [42].

Several classification methods have been proposed to categorize machine learning [11]. One of the most common categories is to divide machine learning into supervised learning, unsupervised learning, and reinforcement learning [41] as shown in Fig.3.1 [106] :

(1) **Supervised learning** : According to [82], supervised learning, also known as density estimation in statistics, attempts to determine the input-output relationship of a system based on a given set of paired input-output training samples. Leveraging the found relationship, it can suggest the output of the system when a new input is given. Depending on the nature of the target output data, su-

pervised learning can be divided into classification and regression : If the target output consists of category data or a finite set of discrete values that indicate the class labels of the input, then supervised learning is a type of classification. On the other hand, if the output consists of continuous values, then supervised learning is a regression problem. For example, given the inputs about the animals' appearance, such as hair color, weight, and height, and the outputs of the animal categories, a classification-based supervised learning algorithm can find the relationship between the animals' appearance and their categories. In addition, we can use regression learning to find the relationship between the description of some houses, such as the areas, the lighting conditions and the number of rooms, and the prices required to buy the corresponding houses.

(2) **Unsupervised learning** : According to [43], unsupervised learning studies how systems can learn to represent certain input patterns in a way that reflects the statistical structure of the entire collection of input patterns. Clustering and dimensionality reduction are two classic examples of unsupervised learning [53]. As an example of a clustering problem, suppose that the inputs are photoreceptor activities generated by different images of a cat and a dog. Unsupervised learning first characterizes these photoreceptor activities. Then it groups them into the appropriate number of clusters.

(3) **Reinforcement learning** : Reinforcement learning [70] is related to control theory. Here, an agent learns to propose actions that can influence the environment through trial-and-error interactions with the dynamic environment. The goal of reinforcement learning is to maximize cumulative rewards throughout its lifetime. A reward is generated by a predefined reward function with respect to the updated states of the environment and sent back to the agent to guide its learning direction, which serves a similar function as target values in supervised learning. An example of reinforcement learning is the robot. In this example, an agent is used to make decisions for the robot, such as go straight, turn left, turn right, or turn around to reach a certain position. A reward is generated when the robot approaches the target position ; otherwise, a penalty or negative reward is suggested. The reward is sent back to the agent so that it can update its decisions for the next time step.

As mentioned in Chapter 2, existing methods for creating smart home services often require manual input from the inhabitant. These operations can be very complex

if the the physical phenomena of the environment states are complex and the inhabitant has no idea how to adjust the actuators to achieve his/her target monitored states. Therefore, in this chapter, we focus on improving the creation of smart home services aiming at reducing the inhabitant's manual interventions or making the interaction simpler and more natural. Since RL can automatically discover the patterns of the system by interacting with the environment, we can use it to create smart home services that dynamically propose actuators' states.

RL-based smart home services can propose actuators' states to adapt to the inhabitant's preferred monitored states by interacting with the environment. To this end, RL-based services take the environment states as input and suggest the states of the actuators to update the corresponding monitored states, e.g., indoor temperature. The inhabitant expresses either satisfaction or dissatisfaction with the updated monitored states, and this reaction is converted into rewards that are sent back to the services. The way the reaction is expressed should be as simple as possible. For example, in the real world, the inhabitant can directly change the states of the actuators to express his/her dissatisfaction, or he/she can change nothing about the actuators' states to express the satisfaction. Otherwise, the inhabitant can define the target monitored states. In this case, to calculate the rewards, services compare the updated monitored states with their target values. They then update their associated parameters by considering the original environment states, the rewards received, and the updated environment states to ensure that positive rewards are generated in the next time step.

Therefore, RL-based smart home services do not require complex or technical operations from the inhabitant. They can consider the inhabitant's reactions, including satisfaction or dissatisfaction, and can dynamically and continuously learn the inhabitant's target monitored states by interacting with the environment. In the following sections of this chapter, the principle of RL is explained in more detail.

3.3 Reinforcement learning classification

Reinforcement learning (RL) algorithms can be broadly categorized into two main types : Model-Based Reinforcement Learning (MBRL) [94] and Model-Free Reinforcement Learning (MFRL) [33].

(1) **MBRL** : In MBRL, a model is used to mimic the environment and decide

the next states after the system performs certain actions. The environment dynamic characteristics are represented by a set of transition probabilities, with each describing the probability of changing from certain environment states to some new ones.

(2) **MFRL** : On the other hand, MFRL does not require transition probabilities to describe the dynamics of the environment. Since it is difficult and complex to obtain transition probabilities between environment states in the real world, we choose RL algorithms belonging to MFRL algorithms in our work.

In addition, MFRL is further divided into Monte Carlo (MC) learning and Temporal Difference (TD) learning [117] :

(1) **MC** : MC only works for episodic tasks and updates action quality values at the end of the episode, making the learning process of MC slower

(2) **TD** : TD works not only for episodic tasks, but also continuing tasks that update action quality values at each time step [65].

Since it makes more sense to allow the inhabitant to express his/her (dis)satisfaction at each time step rather than at the end of an episode, TD learning is preferred in our study.

Depending on how the action quality values are updated, TD learning is further divided into several categories, such as **Q-learning** [126], **SARSA** [117], **Double deep Q-learning (DDQN)** [124], and **Dueling Double Deep Q-learning** [125]. In our studies, Q-learning is chosen. Therefore, the principles of RL to be presented in the next section is based on the Q-learning.

3.4 Reinforcement learning principles

The detailed principle of RL is shown in Fig.3.2, with the associated variables being explained as follows :

1. t at the subscript is the time step, where a time step is an iteration from the acquisition of the observable states from the environment to the acquisition of the new observable states.
2. Az represent the states proposed by the agent for the related actuators.

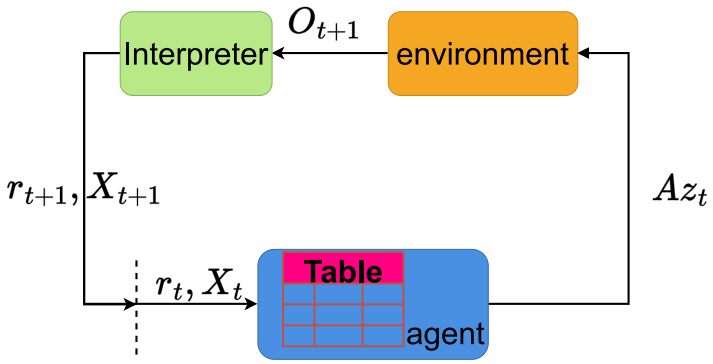

FIGURE 3.2 – Principle of RL based on table

3. O is the observable states that represent states that can be accessed by the agent, thus including the states of actuators and sensors.
4. X represent the states extracted from O and used as input to the agent.
5. r represent the reward computed by a given reward function, and tells the agent whether it is good or not to propose Az.

The "agent" is usually a table that stores the mapping between the input states X at different time steps and the action quality values of all states of all actuators, as shown in Fig.3.3. The action quality value of a particular state of an actuator at

environment states	Q: actuator's state / actuator	state 0	state 1	state 2	⋯
X_0	Act 1	$Q^1_{0,0}$	$Q^1_{1,0}$	$Q^1_{2,0}$	⋯
	Act 2	$Q^2_{0,0}$	$Q^2_{1,0}$	$Q^2_{2,0}$	⋯
	⋯				
X_1	Act 1	$Q^1_{0,1}$	$Q^1_{1,1}$	$Q^1_{2,1}$	⋯
	Act 2	$Q^2_{0,1}$	$Q^2_{1,1}$	$Q^2_{2,1}$	⋯
	⋯				
⋯		⋯			
X_t	Act 1	$Q^1_{0,t}$	$Q^1_{1,t}$	$Q^1_{2,1}$	⋯
	Act 2	$Q^2_{0,t}$	$Q^2_{1,t}$	$Q^2_{2,t}$	⋯
	⋯				

FIGURE 3.3 – Q-learning table storing action quality values

certain X is denoted as $Q^a_{b,c}$, where a represents the actuator to which the state is associated, b indicates the state of the actuator a to which the current action quality value Q belongs, and c is the time step of the current environment states X_c. An action quality value Q represents the expected long-term reward the agent could receive if it selects some state as the new state of the associated actuator at the current time step.

At time step t, the agent uses a certain policy, for example, the greedy-policy, to select the state whose action quality value is the maximum for each actuator at time step t :

$$\begin{aligned} &\text{state for actuator Act 1 is the state with index :} \\ &argmax(Q^1_{0,t}, Q^1_{1,t}, Q^1_{2,t}, \cdot) \\ &\text{state for actuator Act 2 is the state with index :} \\ &argmax(Q^2_{0,t}, Q^2_{1,t}, Q^2_{2,t}, \cdot) \\ &\cdots, \end{aligned} \tag{3.1}$$

where $argmax$ means to select the index of the action quality value whose value is the maximum.

We use A_t to represent the updated actuators' states after the actuators change their states from A_{t-1} to Az_t (therefore, A_t has the same value with Az_t). The state changes of the actuators cause the observable states to change from O_t to O_{t+1}. Based on the updated monitored state contained in the updated observable states, a reward r_{t+1} is calculated. Therefore, r_{t+1} is calculated at time step $t+1$ to represent the satisfaction of the proposed actuators' states by the agent at time step t. Moreover, the states X_{t+1}, which serve as the input of the RL agent, are extracted from O_{t+1} by an "interpreter". Then, the next time step $t+1$ becomes the current time step t, and X_{t+1} and the reward r_{t+1} are used as the current X_t and r_t and sent to the agent.

Next, based on X_t, A_t, X_{t+1}, and r_{t+1}, the "agent" updates the action quality values for the updated A_t within the table with :

$$Q^{new}(X_t, A_t) \leftarrow (1-\alpha) \cdot Q_t(X_t, A_t) + \alpha \cdot (r_{t+1} + \gamma \cdot max_A Q(X_{t+1}, A)), \tag{3.2}$$

where $max_A Q(X_{t+1}, A)$ is the maximum action quality values that can be obtained when the agent is at states X_{t+1}. Finally, the updated "agent" repeats the above process.

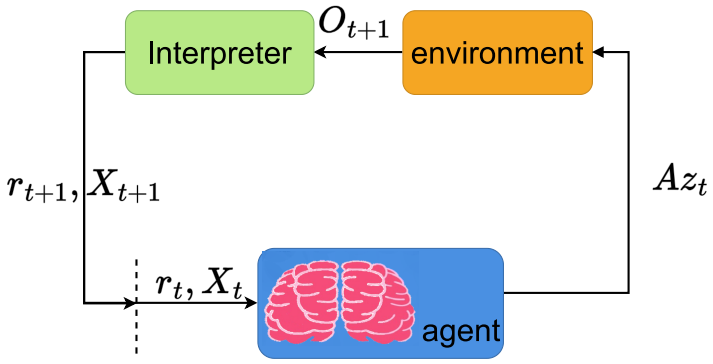

FIGURE 3.4 – Principle of RL based on machine learning system

The agent continues the iteration until it reaches the desired state or a maximum number of time steps has elapsed. This series of time steps is called one episode. In our study, we denote an episode when the maximum number of time steps T is reached. We use $Epoch$ to denote the number of episodes, which means that the agent repeats the iterations $Epoch$ times, where each iteration starts with the initial time step $t = 0$ and lasts until the maximum time step $t = T$.

However, systems in the real world, e.g., a smart home system, are usually high-dimensional by containing diverse environment states. Therefore, a table is not sufficient to establish the mapping between the input states and the corresponding action quality values of all possible states of all actuators, since storing such a table can require a large amount of memory.

To solve this problem, as shown in Fig.3.4, Deep Reinforcement Learning (DRL), which uses deep learning algorithms that include MultiLayer Perceptions (MLP) and Long Short Term Memory (LSTM) cells to replace the table, is used in our study. The agent based on MLP and LSTM cells is shown in Fig.3.5. The input X_t is passed through several LSTM cells and an MLP to obtain the corresponding action quality values Q_t. Some policy, such as the ϵ-greedy policy [121], is used to select states for each actuator. The principle of the LSTM cell and MLP is presented in subsections 3.4.1 and 3.4.2, respectively.

Therefore, the agent contains a regression function $f_{DRL}(X)$ based on the DRL model that, given knowledge of the input states X, can compute the action quality values for all possible states of all actuators. For example, knowing the input states X_t at time step t, we obtain a vector of action quality values Q for all possible states

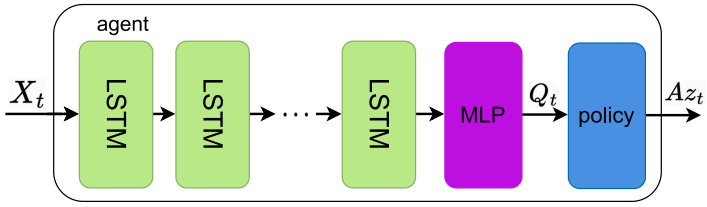

FIGURE 3.5 – MLP and LSTM cells-based RL agent

of all actuators :

$$Q \leftarrow f_{DRL}(X_t). \tag{3.3}$$

The process of calculating the action quality values without updating the regression function by updating the parameters contained in it is also called the testing process of the DRL model.

Assuming DRL contains parameters θ that include weights and bias within related deep learning algorithms, such as the weights and bias of LSTM cells and MLP to be presented in subsections 3.4.1 and 3.4.2, f_{DRL} can be written as $f_{DRL}(X\,;\theta)$, meaning that f_{DRL} has parameters θ and takes states X as input. To update the agent, instead of updating the action quality values as a table did, the DRL model tries to update its parameters θ so that the difference between the original action quality values $Q(X_t)$ and the updated action quality values $Q^{new}(X_t)$ are minimized. Therefore, we need to first compute $Q^{new}(X_t)$ using f_{DRL} by rewriting Eq.3.2 as :

$$Q^{new} \leftarrow (1-\alpha)\cdot f_{DRL}(X_t\,;\theta) + \alpha \cdot (r_{t+1} + \gamma \cdot max(f_{DRL}(X_{t+1}\,;\theta'))), \tag{3.4}$$

where a copy of f_{DRL} is defined with possibly different values θ' for the same kind of parameters. In this case, $f_{DRL}(X\,;\theta)$ is called the main DRL, which is the model we are trying to update. $f_{DRL}(X\,;\theta')$ is called the target DRL. It is used as the target that the main DRL is trying to adapt to, and its parameters are assigned to the parameters of the main DRL every a predetermined number of episodes, avoiding the problem of target shifting.

The process of updating DRL parameters is called the training process of the DRL model. A replay memory [84] is used to train DRL. A replay memory M stores the transition e of each time step. Assuming that the current time step is t, the training process of DRL can be represented in Algo.1.

Algorithm 1: train DRL model

Result: reward
1 M : replay memory
2 L : the minibatch (minimum numbers) of transitions used to train DRL
3 C : certain predefined number of time steps
4 $Epoch$: total episodes
5 T : total time steps
6 ep : current episode
7 $f_{DRL}(X;\theta)$: main DRL
8 $f_{DRL}(X;\theta')$: target DRL, where θ' has the same value with θ before starting to train.
9 **Function** $training()$ **:**
10 **for** *each episode* ep *in* $Epoch$ **do**
11 **for** *each time step* t *in* T **do**
12 $Q_t \leftarrow f_{DRL}(X_t;\theta)$
13 Use ϵ-greedy policy to select states Az_t for actuators based on Q_t
14 State changes of the actuators ($A_t \leftarrow Az_t$) cause states O_t to change to O_{t+1}
15 Calculate r_{t+1} using predefined reward function by considering O_{t+1}
16 Acquire the transition $e_{t+1} \leftarrow \{O_t, A_t, O_{t+1}, r_{t+1}\}$
17 Update the replay memory M by concatenating e_{t+1} to M
18 Randomly sample L transitions from M, and denote these transitions as E.
19 **for** *each transition* e *in* E **do**
20 knowing $e \rightarrow O, A, O^{new}, r$
21 $X \leftarrow$ extract from O by the interpreter
22 $X^{new} \leftarrow$ extract from O^{new} by the interpreter
23 $Q \leftarrow f_{DRL}(X;\theta)$
24 $Q^{new} \leftarrow (1-\alpha)\cdot Q + \alpha \cdot (r + \gamma \cdot max(f_{DRL}(X^{new};\theta'))$
25 $f_{loss}(Q, Q^{new};\theta)$ mean square error loss function
26 $\{\theta \leftarrow \theta - \alpha \frac{\partial f_{loss}}{\partial \theta}\}$
27 **end**
28 **end**
29 **if** $ep\%C$ $equals$ 0 **then**
30 $\theta' \leftarrow \theta$
31 **end**
32 **end**

Knowing that the total number of episodes is $Epoch$, and the total number of time steps is T, suppose that the current episode is ep (line 10) and the current time step is t (line 11). The main DRL calculates the action quality values for all states of all actuators using Eq.3.3 (line 12). Then, the ϵ-greedy policy is used to select the states Az_t for all associated actuators (13). The actuators change their states from A_{t-1} to A_t, where A_t equals to the proposed Az_t, and the state changes of the actuators lead to the changes of the observable states O_t to new values O_{t+1} (line 14). Afterwards, a predefined reward function is used to calculate reward r_{t+1} by considering O_{t+1} (line 15). At this point, a transition e_{t+1} is obtained that has the value $e_{t+1} = \{O_t, A_t, O_{t+1}, r_{t+1}\}$ (line 16), and concatenated in the replay memory M (line 17).

Next, we randomly sample a minibatch transitions E with size L from the replay

memory M (line 18). For each transition e in E (line 19), knowing that the transition e contains O, A, O^{new} and r (line 20). we use the interpreter to respectively extract input states X and X^{new} from O and O^{new} (lines 21$\sim$ 22). Next, we calculate the action quality values Q for the initial states X in e using Eq.3.3 (line 23). Then we use Eq.3.4, and leveraging main DRL $f_{DRL}(X;\theta)$ and target DRL $f_{DRL}(X;\theta')$ to update the action quality values Q so as to obtain Q^{new} (line 24). Before starting to train $f_{DRL}(X;\theta)$, θ' has the same value with θ. Afterwards, we calculate the mean square error between Q and Q^{new} (line 25), and use gradient descent to update the value of θ (line 26). The updating process of parameters θ is performed at each time step. After iterating all transitions, at the end of the current time step t, if the current episode ep is an integer multiple of C (line 29), we assign the values of θ to θ'.

The f_{DRL} with updated parameter θ can be used to do the testing process using Eq.3.3 to propose action quality values for the new arriving input X_{t+1}. In the following sections, we present the principle of MLP and that of LSTM cell.

3.4.1 Multilayer perceptron

Neural networks (NNs) are computing systems inspired by biological neural networks that constitute animal brains. They are based on a collection of connected units or nodes called artificial neurons. In each neuron, as shown in Fig.3.6, a multiple linear regression is first performed by multiplying multi-variable inputs $x_0, x_1, \cdots, x_n$, such as the inhabitant's states, the outdoor environment state, and the indoor monitored states (e.g., the indoor temperature), by the corresponding bias b and weights $w_1, w_2, \cdots, w_n$ of the connections :

$$\mathsf{z}(x_0, x_1, \cdots, x_n) = x_0 \cdot b + x_1 \cdot w_1 + x_2 \cdot w_2 + \cdots + x_n \cdot w_n, \tag{3.5}$$

The result is then used as input to a linear or nonlinear activation function to obtain the output y [52] :

$$y = \mathsf{f}(x_0 \cdot b + x_1 \cdot w_1 + x_2 \cdot w_2 + \cdots + x_n \cdot w_n) \tag{3.6}$$

The nonlinear activation functions allow the neural networks to recognize complex nonlinear patterns of the system under study.

An artificial neural network (ANN), as shown in Fig.3.7, consists of three layers : an input layer, a hidden layer, and an output layer. The input layer contains inde-

pendent variables that help determine the outputs in the output layer. The hidden layer consists of several artificial neurons that allow ANN to recognize the complex patterns of the system under study.

When the depth of the hidden layers is more than one, the system is called MLP (Multi-Layered Perceptron) [78]. An MLP example containing two hidden layers is shown in 3.8. The input layer and the output layer contain the same thing as ANN : the input layer collects input patterns, and the output layer has classifications or output signals to which the input patterns can be assigned. For example, the input patterns may contain a set of descriptions about an animal, and the output is the category of the animal. The hidden layers fine-tune their input weights on the connections until the cumulative error of the samples, i.e., the difference between the targets and the outputs of the MLP, is the minimum.

3.4.2 Long short term memory

Recurrent neural networks (RNNs) [108], are a class of neural networks that use the outputs of the previous time step as inputs of the current time step. The principle of RNNs is shown in Fig.3.9. The structure on the left shows the general principle of RNN. The "RNN" block in the middle represents an RNN cell, which is the backbone of the RNN. It corresponds to an artificial neuron in the MLP and fine-tune their input weights until the cumulative error between the targets and the outputs of the MLP is the minimum.

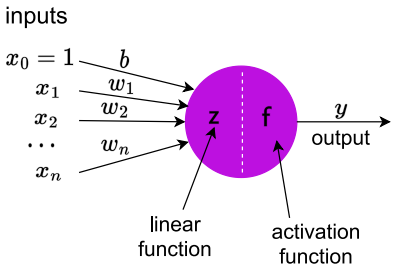

FIGURE 3.6 – An artificial neuron

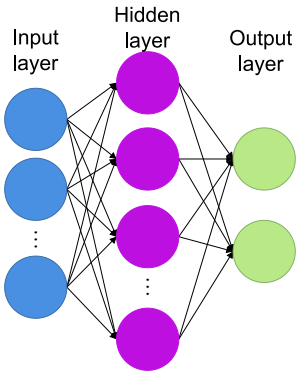

FIGURE 3.7 – An aritificial neural network (ANN)

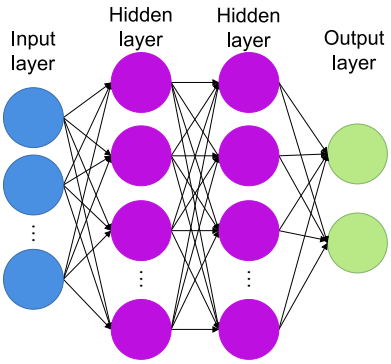

FIGURE 3.8 – A multi-layer perceptron (MLP)

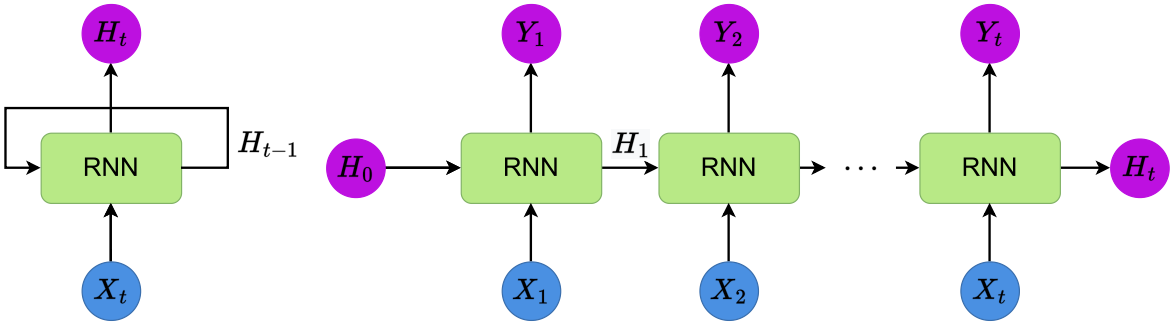

FIGURE 3.9 – Principle of RNN

In addition, we can see that the RNN cell contains two inputs : Input states X_t, which represent the current information from the environment, and hidden states H_t, which are used to encode the characterization of the information from the previous time step. The output of the RNN at the current time step are the hidden states of that time step.

The specific procedure of an RNN containing one RNN cell is explained on the right in Fig.3.9 : first, we initialize the hidden states H_0. Then, in the first time step, H_0 and the environment states X_1 in the current time step are used as input to the RNN cell, resulting in the hidden states H_1 and the output Y_1, where Y_1 and H_1 have the same value. Subsequently, H_1 and X_2 are used as input to the same RNN cell state, producing H_2, which is also used as Y_2. The process repeats until the current time step t, where the same RNN cell takes the current environment states X_t and the hidden states H_{t-1} of the previous time step as input and generates the output value Y_t and the hidden states H_t for the current time step t.

There are several types of RNN. The one we use is Long Short Term Memory (LSTM), first introduced in [61]. Its general principle is shown on the left in Fig.3.10. Compared to the left part in Fig.3.9, in addition to the hidden state H_{t-1}, there is an additional variable C_{t-1} that encodes the aggregation of information from all previous time steps that were processed. The right part is the structure of the LSTM cell, which distinguishes the LSTM from other RNN learning systems.

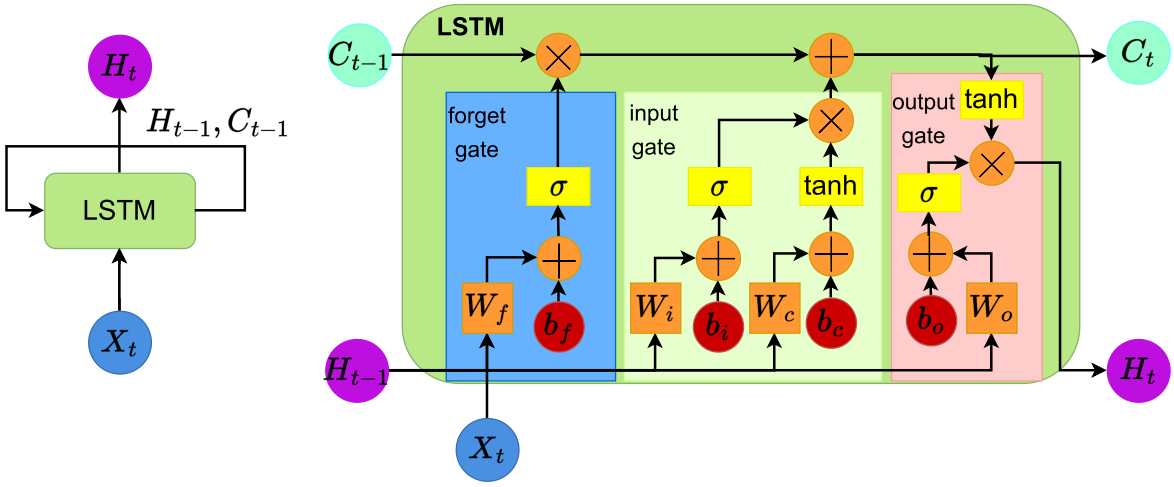

FIGURE 3.10 – Principle of LSTM

From the right part of Fig.3.10, we can see that an LSTM cell contains three gates : a forget gate, an input gate and an output gate to store, write or read the information. A forget gate decides which information is important and which can be ignored by using a sigmoid function $\frac{1}{1+e^{-x}}$, represented by σ. The sigmoid function takes the hidden states H_{t-1} of the previous time step and the current environment states X_t as inputs. If the output of the sigmoid function is close to 1, the corresponding inputs are important ; on the other hand, if the output is close to 0, the forget gate ignores the corresponding inputs. Thus, the output of the forget gate is :

$$f_t = \sigma(W_f \cdot [H_{t-1}, X_t] + b_f), \tag{3.7}$$

where $[H_{t-1}, X_t]$ denotes the concatenation of H_{t-1} and X_t to obtain a multivariable matrix, where each input sample studied contains states belonging to H_{t-1} or X_t.

The second gate is the input gate that decides what new information is added into the previous cell state C_{t-1}. There are several operations in this gate : first, a sigmoid function is applied to the output of the linear regression function on the current states X_t and the previous hidden states H_{t-1} :

$$i_t = \sigma(W_i \cdot [H_{t-1}, X_t] + b_i). \tag{3.8}$$

Similar to the forget gate, this operation selects the important information with value 1 from the inputs and ignores the unimportant information with value 0. Then, a tanh function $\frac{e^x - e^{-x}}{2}$ is used to encode the states X_t and the hidden states H_{t-1} between

-1 and 1 :

$$\tilde{C}_t = tanh(W_c \cdot [H_{t-1}, S_t] + b_c). \tag{3.9}$$

This new information is then multiplied by the output of the sigmoid function to select the more important information among the encoded new information, where $\odot$ means element-wise multiplication :

$$\tilde{C}_t = i_t \odot \tilde{C}_t. \tag{3.10}$$

Next, the cell state updates its value by first multiplying the output of the forget gate and then adding the additional new cell state that comes from the input gate. The whole process can be expressed as follows :

$$C_t = C_{t-1} \odot f_t + \tilde{C}_t. \tag{3.11}$$

The cell state C_t is firstly used as the updated cell state at the current time step t. Secondly, it is used in the output gate to multiply the selected important information from the input states to obtain the updated hidden state, which is also the output value Y_t proposed by the LSTM to approximate the target value :

$$H_t = tanh(C_t) \odot o_t, \tag{3.12}$$

where o_t is the output of the output gate and is calculated by :

$$o_t = \sigma(W_o \cdot [H_{t-1}, S_t]). \tag{3.13}$$

3.5 Existing work

Machine learning systems have become increasingly popular in recent years. More and more smart home systems are using these systems to implement different services within a smart home. For example, [28] proposes a framework to learn a situation model to free inhabitants from manually designing smart home services. This model aims to provide context-aware services in a smart environment : User activities are learned and recognized using support vector machines (SVMs) based on collected data labeled by an expert. Then, unsupervised learning is used to extract situations from the observed data based on Jeffrey divergence. Finally, the extracted situation segments can be used to learn situation labels with user or expert input. The resul-

ting situation model can be further developed based on the resident's feedback using the situation partitioning. [96] proposes a system called ACHE that controls temperature, heating, and lighting without prior programming by the inhabitant. ACHE aims to save energy resources while respecting the inhabitant's habitual behavior. Specifically, the system continuously monitors the environment and observes inhabitant's activities. Then, it uses reinforcement learning-based neural networks to extract the patterns of inhabitant's habitual lifestyle. Finally, it implements smart home control. [130] uses machine learning, sensing and networking technology, and eco-feedback features to regulate indoor temperature while adapting to the inhabitant's habitual behaviors. [22], using Batch Reinforcement Learning, proposes an energy management system called RLbEMS to autonomously define a strategy for selling or storing excess energy in a smart home. [131] proposes an energy management algorithm based on deep deterministic policy gradients to schedule HVAC (Heating, Ventilation, and Air Conditioning) and energy storage in smart home. [95] uses Q-learning to adapt to an inhabitant's habitual behavior and implement lighting control. [77, 129] use DRL to study a user's preferred indoor temperature and use this information to adjust temperature-related actuators to ultimately achieve the goal of saving energy while meeting the user's comfort requirements

In addition, there is also other work that uses RL to address different problems and realize various applications. For example, [44] proposes an RL-based automatic scaling algorithm called HSLinUCB to address the horizontal auto-scaling problem, which consists in creating or deleting some complete resources for Web applications in the Cloud. HSLinUCB aims to optimize resource allocation by finding the smallest number of containers required to satisfy SLA (Service Level Agreement). To do this, HSLinUCB proposes the number of containers by considering the contextual information. The proposed number of containers, together with the new contextual information contributes to the acquisition of the latency at the new time step. A predefined reward function based on the latency is leveraged to calculate the reward to update the parameters of HSLinUCB so as to guide its learning direction. [109] proposes a framework to realize the autonomous driving. This framework uses the attention models to extract relevant information from the collected raw data of sensors, then applies RNN for information integration, and finally uses end-to-end DRL pipeline for autonomous driving. [54] demonstrates that a recent DRL algorithm based on off-policy training of DQN can scale to complex 3D manipulation tasks and can

learn deep neural network policies efficiently enough to train on real physical robots. It also demonstrate that the training times can be further reduced by parallelizing the algorithm across multiple robots which pool their policy updates asynchronously.

3.6 Conclusion

Data-driven approaches are another main category of approaches used for the development of smart home services. In this type of approaches, smart home services are created by proposing states to the corresponding actuators after considering the current states of the environment. Our study mainly focuses on data-driven approaches based on machine learning. This approach can dynamically create services by learning from available datasets. Depending on whether the suggestions affect the environment, the current machine learning algorithms are divided into RL when the response is positive and other algorithms when the response is negative. Depending on whether there is a target or label for each sample during the training process, these other algorithms can be divided into supervised learning when there are targets or labels for the corresponding samples and unsupervised learning when there are no targets or no labels for these samples. Since RL can dynamically create services by interacting with the environment, we tend to use it to create smart home services that can dynamically suggest the state of actuators considering the satisfaction or dissatisfaction of the inhabitant. It is also proposed to integrate RL with other machine learning algorithms such as MLPs and LSTM cells so that RL can learn the patterns of more complex systems.

4

Hybrid approaches for service creation

Contents

4.1 Introduction

Knowledge-based and data-driven approaches are two main approaches to smart home service development. Nevertheless, neither of them can satisfactorily realize service creation. For example, although the knowledge-based approach can create services that show the inhabitant in which situations certain actuators' states are suggested, these services have difficulty in adapting to the changing environment states. As for the data-driven approach, despite its ability to create dynamic services, it is usually like a black box, and the inhabitant cannot understand in which situation certain services proposed certain actuators' states. To solve the problems of the two approaches while preserving their advantages, we think of using hybrid systems based on both approaches and propose some requirements that a hybrid system should

satisfy when creating services. For example, the services created by the hybrid system should be able to dynamically propose actuators' states to adapt to the changing environment states and the changing inhabitant's preference. Moreover, the created services should be explicable by showing the inhabitant in which situations certain states of the actuators are suggested. Even though the rules in the knowledge representation cannot ensure to cover all possible environment states, the hybrid system should still be able to create dynamic services

In this chapter, we first discuss the advantages and disadvantages of knowledge-based and data-driven approaches. Based on the discussion, we propose hybrid systems based on knowledge-based and data-driven approaches. Afterward, we present existing work about using hybrid systems to create services. Next, considering the existing hybrid systems, we formulate requirements that a hybrid system should have to enable the satisfying creation of smart home services. Finally, we provide the conclusion of the chapter.

4.2 Discussion of existing service creation approaches

As shown in the previous chapters, knowledge-based and data-driven approaches are the main approaches to create smart home services. In this section, we describe the advantages and disadvantages of the two approaches for smart home service creation.

4.2.1 Knowledge-based approaches for service creation

The knowledge-based approaches have several advantages and disadvantages in the development of smart home services. The advantages of the knowledge-based approaches for the development of smart home services can be enumerated as follows :

(1) **Explicability** : Due to the clear structure of rules, the created knowledge-based services can show users in which situations certain services have suggested certain states for the actuators.

(2) **Manual creation of simple services** : If the inhabitant knows his/her preferences and how to set the actuators to achieve the preferences, he/she can manually create services by describing a set of rules that guarantee the inhabitant's satisfaction if the inhabitant's preferences do not change.

(3) **Time saving** : The time required to create services with knowledge-based approaches is small if they are simple.

(4) **Learning guide** : The created knowledge-based services can be used to determine the learning direction of the data-driven services.

However, knowledge-based approaches also have disadvantages in the development of smart home services :

(1) **Difficulty of manual creation of complex services** : The inhabitant often has to create knowledge-based services manually or make some mandatory inputs. The process of manually creating services or making required inputs can be complex. This is because if only the inhabitant's target monitored states are known, the developers or inhabitant must calculate the inverse equations to figure out how to set the actuators to obtain the target monitored states. However, these inverse equation calculations can be difficult if the physical phenomena of the environment states are complex and there are numerous actuators being involved.

(2) **Static services** : The services created are often static and cannot evolve to adapt to the changing environment and the changing inhabitant's preference.

(3) **Difficulty in ensuring to cover all environment states** : Knowledge-based services cannot ensure to cover all possible situations of the environment states while satisfying the changing inhabitant's preference for certain monitored state, especially when there are multiple services, therefore, multiple monitored states.

4.2.2 Data-driven approaches for service creation

When data-driven approaches are used to create smart home services, the advantages are as follows :

(1) **Relatively easy service creation** : The data-driven services can be relatively easily created for any environment states without considering the explicit physical phenomena of the associated environment states.

(2) **Dynamic services creation with inhabitant interaction** : Some data-driven approaches, such as RL, can create services that dynamically adapt to the inhabitant's preferences by taking into account inhabitant's satisfaction or dissatisfaction.

(3) **Coverage of all environment states** : The created data-driven services can propose actuators' states for any possible environment states.

However, data-driven approaches also have some drawbacks when creating dynamic services, for example :

(1) **Lack of explicability** : Unlike explicable knowledge-based services, the inhabitant does not know in which situations the created data-driven services suggested certain actuators' states.

To address the problems of both knowledge-based and data-driven approaches in creating dynamic smart home services while preserving their advantages, we consider hybrid approaches.

4.2.3 Hybrid approaches for service creation

Hybrid approaches combine knowledge-based and data-driven approaches, where knowledge-based and data-driven approaches are used to propose the states of actuators. Depending on how the two approaches are combined, there are different hybrid systems based on hybrid approaches, as shown in the next section.

4.3 Existing work

Some hybrid solutions have been studied in the domain of the Internet of Things (IoT) or control problems. Two existing projects have been found concerning the service creation using hybrid systems. For example, [128] proposes a neural network and rules based approach for the smart home system, using predefined rules to control the system while collecting data and training the neural network. Once the neural network can generate new knowledge that guarantees higher efficiency, the system switches to using the neural network.

[57] proposes a hybrid system that integrates knowledge-based and neural network systems. At the beginning of the control problem, the knowledge-based system achieves a specific control objective based on predefined rules in the knowledge base. Then, the rules teach the neural network to make the same decision through reinforcement learning. Finally, the neural network takes over the decision-making once it is optimized. However, if errors occur in the neural network's decisions, the knowledge-based system again takes responsibility for decision-making and teaching the neural network. This work does not only use the rules because, in its opinion, with the integration of highly parallel hardware architectures for the neural network,

the performance, such as the execution time, of this neural network would vastly improve.

4.4 Requirements for an hybrid system

Both [128] and [57] only consider the situation that at the beginning of the training process the machine learning method-based system is not well trained. Therefore, knowledge-based services are used to control the system. They could be used to control the learning direction of the data-driven services. Once the data-driven services are well-trained, the data-driven services take control of the system instead of continuing to use the knowledge-based services.

However, both failed to take into account several situations. First, knowledge-based services have difficulty in covering all possible situations while adapting to the inhabitant's preferences. Second, they are usually created manually, and the creation process can be complicated if the physical phenomena of the environment states are complex and the inhabitant has no idea how to adjust the actuators to achieve his/-her target monitored states. Third, data-driven services are like black boxes, and the inhabitant has no idea in which situations the data-driven services suggested certain actuators' states.

Therefore, we consider another hybrid system to overcome the above disadvantages. The desired hybrid system should meet the following requirements :

(1) **Simultaneous control by knowledge-based and data-driven services** : Knowledge-based and data-driven services should control the system simultaneously. It is because if the knowledge-based services cannot propose actuators' states for certain situations, the data-driven services can make the proposition to control the system. The simultaneous working process allows the system to overcome the disadvantage that the knowledge-based services cannot ensure to cover all possible situations of the environment states while adapting to the inhabitant's preferences.

(2) **Predefined priorities for knowledge-based and data-driven services** : Some priorities should be predefined for knowledge-based and data-driven services when these services attempt to adjust the same actuators. This avoids potential conflicts when they propose different states for identical actuators.

(3) **Dynamic creation of knowledge-based services** : In addition, knowledge-based services should be dynamically generated to adapt to the changing environment states and the changing inhabitant's preference, eliminating the necessity for the inhabitant to manually create complex knowledge-based services.

(4) **Manual creation of simple knowledge-based services** : Manual creation of knowledge-based services is always possible for the inhabitant, so the inhabitant can manually create simple smart home services. For example, the inhabitant wants to turn off the lamp when he/she is sleeping. The manual creation of knowledge-based services saves the time spent by data-driven approaches to learning the same services.

(5) **Explicability of data-driven services** : Finally, the data-driven services should be able to indicate to the inhabitant in which environment states they suggested certain states of the actuators, which can be essential for the maintenance and security of the hybrid system.

To realize this hybrid system, we try to solve the following problems in the following chapters of the second part "Contribution" :

(1) We start from a simple smart home system in which there is only one service, and try to propose a method for creating a single dynamic smart home service.

(2) Since a smart home system usually contains multiple services, we should find a way to create multiple dynamic smart home services which are conflict-free based on the solution to the (1).

(3) Since it is important to show the inhabitant in which situations certain services proposed certain states of actuators, we should propose a method to make the created dynamic smart home services explicable.

(4) We should propose a system that improves the solution to the (2) while integrating the solution to the (3) to create dynamic smart home services that are explicable and conflict-free .

4.5 Conclusion

Knowledge-based approaches can be used to create smart home services by defining a list of rules. Based on the concise and clear structure of rules, the created

knowledge-based services can show the inhabitant in which situations certain services proposed certain states for the actuators. Moreover, the inhabitant can manually create services if he/she knows his/her preferences and how to set the actuators to achieve the target monitored states, which ensures the inhabitant's satisfaction if his/her preferences do not change. Creating services with knowledge-based approaches will take little time if the services are easy to create. Finally, the created knowledge-based services can be used to determine the learning direction of the data-driven services.

Nevertheless, the inhabitant is required to manually create services or make certain entries. These manual creation processes can become complex if the desired services are complex. For example, the physical phenomena of the environment states are complex and the inhabitant has no idea how to adjust the actuators to achieve his/her target monitored states. As a result, the developers or the inhabitant need to calculate the inverse equations to know which actuators' states to be set when the system is at certain environment states. This process becomes even more complex when there are numerous actuators and multiple monitored states. Moreover, the created services are often static and cannot adapt to the changing environment and the changing inhabitant's preference. Furthermore, knowledge-based services usually cannot cover all possible situations, which means that there are situations where existing knowledge-based services cannot propose states to actuators.

As for data-driven approaches, they can relatively easily create smart home services without requiring explicit physical phenomena of the environment states. They can create dynamic services that adapt to the changing inhabitant's preference and environment by considering the inhabitant's satisfaction or dissatisfaction. The created data-driven services can propose actuators' states under any environment states. However, unlike knowledge-based services, the inhabitant does not know in which situations the data-driven services have proposed certain actuators' states.

To overcome the drawbacks of these two approaches, some work has proposed hybrid systems that combine both approaches : They use the knowledge-based services to control the system until the data-driven services are well-trained, and to control the learning direction of the data-driven services. However, the knowledge-based services cannot ensure to cover all possible situations of the constantly changing environment. Thus, when only the knowledge-based services control the system, there can be some situations where no services are available to propose states for the actuators. In ad-

dition, the manual creation of the knowledge-based services may become complex when the the physical phenomena of the environment states are complex and the inhabitant has no idea how to adjust the actuators to achieve his/her target monitored states. Finally, the data-driven services are like black boxes, and the inhabitant does not know in which situations certain services have proposed certain actuators' states.

Therefore, we consider another hybrid system to overcome the problems of the existing hybrid system, and propose several requirements that the hybrid system should meet. First, the knowledge-based and data-driven approaches should control the system simultaneously to avoid the disadvantage that the knowledge-based services cannot ensure to cover all possible situations. Second, knowledge-based and data-driven services should be prioritized to avoid conflicts when they simultaneously propose different states for the same actuators. Third, knowledge-based services should be generated dynamically. Meanwhile, the manual creation of knowledge-based services should be maintained so that the inhabitant can create his/her desired smart home services when these services are simple to be created. Finally, data-driven services should indicate to the inhabitant in which situations they have suggested certain actuators' states.

Part II

Contribution

5

Creation of a smart home service using Reinforcement Learning

Contents

5.1 Introduction

Smart homes are homes that deliver smart services to their inhabitants. As shown in Fig.5.1, each smart home is equipped with sensing and actuating devices capable of capturing (represented by a sparse dotted arrow in Fig.5.1) the environment states and of performing actions (represented by a z-shaped arrow in Fig.5.1) that affect these states. The intelligence of smart homes is realized by deploying services. The role of a service is to communicate with (represented by a solid arrow from devices to services in Fig.5.1) and act upon (represented by a solid arrow from services to

actuators in Fig.5.1) these devices in order to produce the appropriate changes of the state of the environment so as to comply with the inhabitant's needs (represented by a dense dotted arrow in Fig.5.1). For instance, an air temperature service may regulate indoor air temperature of a home by acquiring the current room temperature from a smart thermometer (sensor device) and by appropriately acting upon the heater (actuator device) present in the room. The environment represents all physical phenomena that can be measured by sensors and that are of interest to the inhabitant. The states that are of interest to the inhabitant are also called the monitored states. The inhabitant may also be part of the environment as his/her status (reading, watching TV, sleeping, ...) may be sensed and used by services to make the corresponding changes in the environment.

Hence, the inhabitant relies on services to produce the needed changes in the environment. But, in fact, he/she may also take an active part in making these changes. The inhabitant may act in two ways. For one, he/she may define the preferred values for the monitored states, values that will be considered as targets to be met by services. Alternatively, the inhabitant may directly act on actuator devices to directly provoke the changes in the monitored states. For instance, he/she may set the preferred indoor air temperature to 19°C, and also, switch on a lamp with the desired light intensity to read a book. The inhabitant's actions, i.e., setting target values or acting on actuators, are observable by the services and hence influence their future behavior.

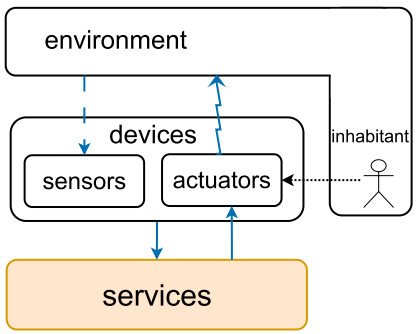

FIGURE 5.1 – Smart home system based on services

Furthermore, a smart home is intrinsically dynamic. On the one hand, the physical environment is itself continuously changing, and, on the other hand, the inhabitant's preference may differ depending on his/her state. For instance, indoor light intensity in a room may be influenced by the outdoor light intensity which varies according

to the time of the day and to weather conditions. This indoor light intensity should also adapt to inhabitant's current activity, bright when reading a book and dim when watching television.

As seen in Chapter 2 and Chapter 3, two main approaches exist to develop smart home services, knowledge-based approach and data-driven approach. The knowledge-based approach consists in specifying rules that are fed to a reasoner which decides what actions need to be performed depending on the current states sensed in the environment. The advantage of this approach is its simplicity when the services to be deployed are simple. But it fails short when multiple services are jointly considered, especially when these services share actuators with possible conflicts in the actions to be applied to these actuators. In addition, knowledge-based services are usually static and cannot evolve to adapt to the dynamic smart home. Furthermore, the knowledge-based approach may be more challenging to use even for some simple services where physical phenomena need to be known by the service. For instance, in a light intensity service where the light intensity of a lamp in a room needs to be set to a certain level so as to have the light in the room comply with a predefined intensity. The level of the light intensity to be set to the lamp will in fact depend on the light intensity provided by external light sources, but the relationship between these different light sources are governed by laws of physics that need to be made explicit in order to be used in a knowledge-based approach.

Data-driven approaches do not have these shortcomings. They rely on having a training period where the inhabitant has access to actuators and has the initiative to take the appropriate actions, while having the services observe and learn from the inhabitant's behavior. Then, the service gradually takes over and becomes more and more active in making decisions. The present chapter is dedicated to presenting a data-driven approach for service creation in a smart home. More specifically, it presents the application of the RL method for creating one single service, the simplest situation of a smart home system. Chapter 6 is devoted to the case of smart homes with multiple services, a general situation of a smart home system.

The data-driven approach, compared to the knowledge-based approach, has some disadvantages. First, it is not transparent to the inhabitant as it does not provide the right explanations about how some decisions have been made by the service. Methods to overcome the defect of explicability are presented in chapter 7. Second, the training period may be long and annoying to the inhabitant. We present means to overcome

this problem at the end of this chapter.

The remainder of this chapter is structured as follows. First, we present the general concepts underlying a simple smart home system, i.e., a smart home with one single service. Then we present the use of an RL method and associated algorithms for the creation of a single service. Next, we address the issue of the long training phase when the service is deployed in the real environment by presenting a simulation based pre-training phase. After that, we provide some simple smart home systems applications, each with a different smart home service. Finally, we conduct two simulated experiments to prove that a simple smart home system can adapt to the inhabitant's target monitored state when he/she uses different ways (either defining the target value for the monitored state or directly changing the actuators' states) to express his/her reaction to the updated monitored state.

5.2 A simple smart home system

We start with the simplest situation, namely a single smart home service. A smart home system that contains only a single service is called a simple smart home system. In this section, we introduce the structure, related concepts, and some examples of such a system.

The principle of a simple smart home system is shown in Fig.5.2, where the service module z can be implemented using either knowledge-based or data-driven approaches. The variables in this figure are explained as follows :

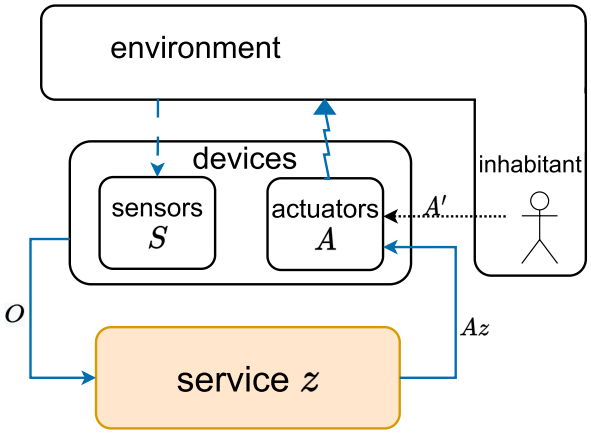

FIGURE 5.2 – simple smart home system

1. $S = \{s_0, s_1, s_2, \cdots\}$ is a set of sensor variables. They capture the current states of

the sensors representing the values of the environment states. s_0 is the monitored variable. Its value is monitored by the service to meet the target defined by the inhabitant.

2. $A = \{a_0, a_1, a_2, \cdots, \}$ is a set of actuator variables. They capture the current states of the actuators. a_0 is the target variable. It holds the target value to be met by the service in s_0. Its value can only be modified by the inhabitant (using a'_0). The other actuator variables ($a_1, a_2, \cdots$) can be modified by the inhabitant (using A'), or by the service (using Az).
3. $A' = \{a'_0, a'_1, a'_2, \cdots, \}$ is a set of variables that are used to assign values to actuator variables ($a_0, a_1, \cdots$) by the inhabitant. a'_0 is the target assignment variable.
4. $Az = \{az_1, az_2, \cdots, \}$ is a set of variables that are used to assign values to actuator variables ($a_1, a_2, \cdots$) by the service. They reflect the actions performed by the service. Az does not include the variable az_0 as the target value is only defined by the inhabitant using a'_0.

Depending on the value set in a_0, we consider two kinds of services :

1. target-defined service : when $a_0 \neq \perp$, i.e., a_0 is defined.
2. target-undefined service : when $a_0 = \perp$.

Target-defined smart home service

A target-defined smart home service is a service where the target variable a_0 has a value defined either initially or dynamically by the inhabitant using a'_0. The inhabitant may also still directly act on the other actuators using $a'_1, a'_2, \cdots$.

Target-undefined smart home service

A target-undefined smart home service is a service where the target variable a_0 is undefined. So the service does not have a target to comply with, the satisfaction of the inhabitant is obtained when the service provides the right actions to be performed on the actuators. When the inhabitant is not satisfied, he/she cannot use a'_0, and can only act on the other actuators using $a'_1, a'_2, \cdots$.

The way inhabitant satisfaction is expressed differs in target-defined and target-undefined services. In target-defined services, satisfaction is measured by the difference between a_0, the target defined by the inhabitant, and s_0, the sensor variable

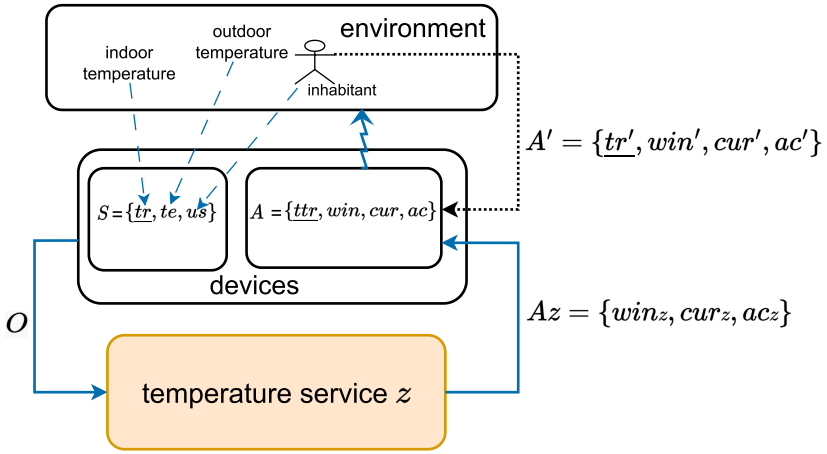

FIGURE 5.3 – Simple smart home system with one temperature service

monitored by the service. In target-undefined service (i.e., the target a_0 is not defined), the inhabitant's satisfaction is evaluated by measuring the distance between Az, the actions proposed by the service, and A', the reactions of the inhabitant to these actions.

Example 1 : simple smart home system with one temperature service

Consider a simple smart home system with only one temperature service, as shown in Fig.5.3. The temperature service sets the temperature in a room by adjusting the states of the window, the curtain, and the air conditioner. The monitored state is the indoor temperature. The temperature service is a target-defined service, which means that the inhabitant defines the preferred indoor temperature. The inhabitant may still act on the states of the actuators. From Fig.5.3 we can see that the components of the temperature service are as follows :

1. $S = \{tr, te, us\}$, where tr is the sensor variable capturing the indoor temperature, te is the sensor variable capturing the outdoor temperature, and us is the sensor variable capturing the inhabitant state (e.g., reading, working,$\cdots$).
2. $A = \{ttr, win, cur, ac\}$, where ttr is the target variable holding the target value that tr should meet. win is the actuator variable containing the state of the window, cur is the actuator variable containing the state of the curtain, and ac is the actuator variable containing the state of the air conditioner.
3. $A' = \{tr', win', cur', ac'\}$, where tr' is the target assignment variable containing the target value for tr, and $\{win', cur', ac'\}$, which are optional, are the actuator

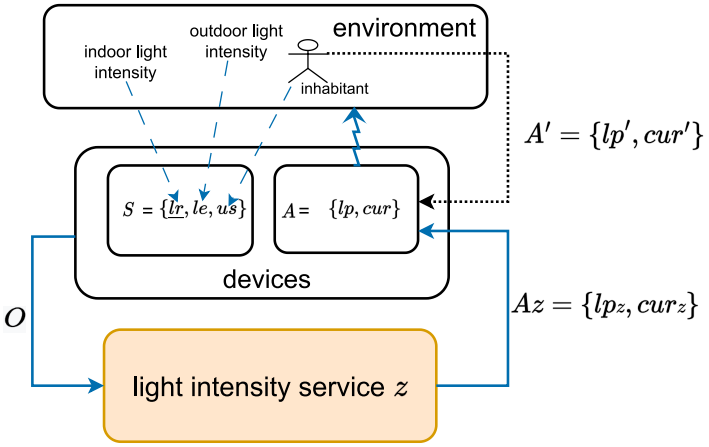

FIGURE 5.4 – Simple smart home system with one light intensity service

variables containing state values that the inhabitant wants the related actuators to have.

4. $Az = \{win_z, cur_z, ac_z\}$, where win_z is an actuator variable containing the state value of the window proposed by the service, cur_z is the actuator variable containing the state value of the curtain proposed by the service, and ac_z is the actuator variable containing the state of the air conditioner proposed by the service.
5. O is a set of states observable by the service, where $O = S \cup A = \{tr, te, us, ttr, win, cur, ac\}$.

Example 2 : simple smart home system with one light intensity service

We continue another simple smart home system with only one light intensity service, as shown in Fig.5.4. The light intensity service adjusts the light intensity in a room by acting on the state of a lamp and a curtain. Therefore, the monitored state for this light intensity service is the indoor light intensity. The light intensity service is a target-undefined service, meaning that the inhabitant only changes the states of the actuators to express his/her dissatisfaction. This allows the service to adapt to the inhabitant's states. From Fig.5.4, it can be seen that the components of the light intensity service are as follows :

1. $S = \{lr, le, us\}$, where lr is the sensor variable capturing the indoor light intensity, le is the sensor variable capturing the outdoor light intensity, and us is the sensor variable capturing the inhabitant state.
2. $A = \{lp, cur\}$, where lp is the actuator variable containing the state value for the lamp, and cur is the actuator variable containing the state value for the curtain.

3. $A' = \{lp', cur'\}$, where lp' is the actuator variable containing the state value for the lamp set by the inhabitant, and cur' is the actuator variable containing the state value for the curtain set by the inhabitant.
4. $Az = \{lp_z, cur_z\}$, where lp_z is the actuator variable containing the state value for the lamp proposed by the service, and cur_z is the actuator variable containing the state value for the curtain proposed by the service.
5. O is a set of observable states, where $O = S \cup A = \{lr, le, us, lp, cur\}$.

5.3 RL-based simple smart home system

Reinforcement learning (RL) [48] has a basic idea that an artificial agent learns the behavioral patterns of the system by interacting with the environment. It is well suited for both target-defined and target-undefined services. When the target values for the monitored states are explicitly defined by the inhabitant, the RL algorithm proposes the states of the actuators considering the environment states and the target values to realize the adaptation. However, if the target values for the monitored states are not explicitly defined by the inhabitant, the RL algorithm proposes the states of the actuators considering the environment states and the new states of the actuators defined by the inhabitant. In this section, we use RL to model a service (target-defined or target-undefined) that can dynamically propose the states of the actuators to adapt to the inhabitant's target monitored state in a simple smart home system.

Fig.5.5 shows the RL-based service within a simple smart home system. It can be seen that the smart home service z consists of three modules : an interpreter, an RL al-

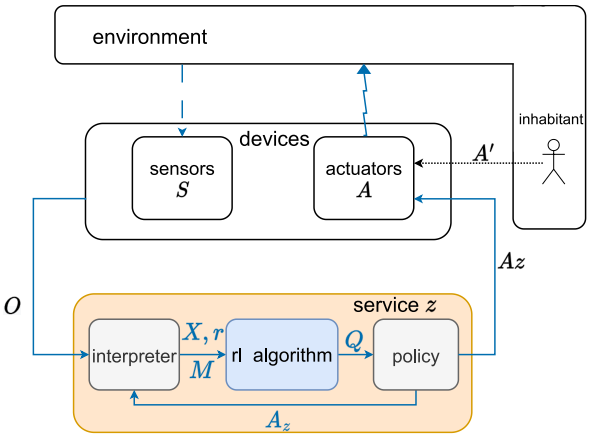

FIGURE 5.5 – RL-based simple smart home system

gorithm, and a policy. An interpreter is responsible for first, extracting the states used as input to the RL algorithm ; second, calculating the reward representing the inhabitant's (dis)satisfaction to the updated monitored state ; third, updating the replay memory storing the transition of one historical time step, where one time step is an iteration from capturing the environment states by the smart home system following the calculation of the reward function by the interpreter, to the new environment states acquisition followed by the reward function calculation. An RL algorithm is used to calculate the action quality values for all possible states of all actuators. A policy selects the final states to which actuators will change their states. The variables mentioned in this figure are shown as follows :

1. X is the states extracted from O and used as input to the RL algorithm.
2. r is the reward calculated by the interpreter representing the inhabitant's (dis)satisfaction to the updated monitored state.
3. M is the replay memory storing the transition e of each past time step.
4. Q is the action quality values denoting the long term reward that the smart home system can obtain when the actuators change their states to certain values.

Fig.5.6 shows the time diagram of one time step for a service with the structure shown in Fig.5.5. A time step is a time interval during which data is examined and analyzed. In this figure, the representation of each variable not presented previously is shown as follows :

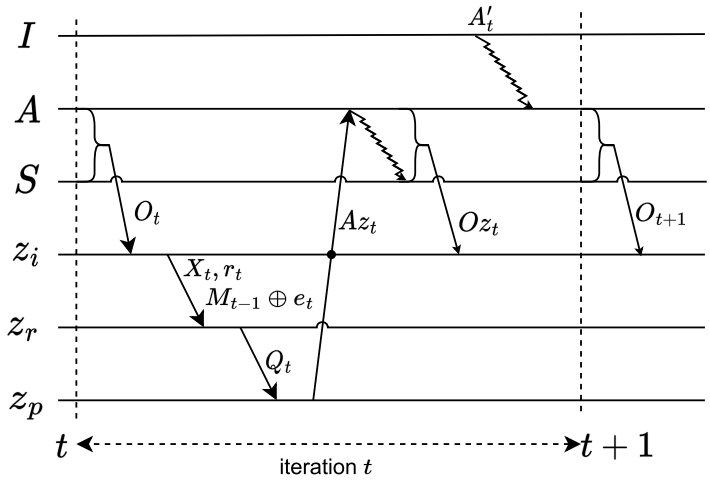

FIGURE 5.6 – Time diagram for RL-based simple smart home system

1. I : inhabitant.
2. z_i : interpreter within the model of the service z.
3. z_r : reward function within the model of the service z.
4. z_p : policy within the model of the service z.
5. e : transition of one time step, where a transition is a vector storing the states related with the service z during the associated time step.

Suppose that currently the system is at time step t, the working process of this time diagram is shown as follows :

1. At the beginning of t, the observable states O_t that include the actuators' states A and the sensors' states S at time step t are acquired by the interpreter z_i. In z_i, O_t will be included into the transition e_t and e_{t+1}.
2. a reward r_t is calculated by z_i to express the inhabitant's (dis)satisfaction to the updated monitored state at time step $t-1$. Therefore, r_t represents the reward calculated at time step t for the actions at time step $t-1$.
3. The transition e_t has its whole values where $e_t = \{O_{t-1}, Az_{t-1}, sz_{0,t-1}, O_t, r_t\}$. Therefore, e_t is calculated at time step t for the actions taken place at time step $t-1$.
4. The replay memory M_{t-1} updates its value to M_t where $M_t = M_{t-1} \oplus e_t$. Therefore, M_t at time step t contains past information before the time step t, and has value $M_t = \{e_1, e_2, \cdots, e_t\}$.
5. The RL algorithm z_r updates its parameters using M_t through training process as to be introduced in Algo.4.
6. Including to calculating r_t, the interpreter z_i also extracts X_t that is used as input to the updated RL algorithm z_r.
7. The updated RL algorithm z_r then proposes the action quality values Q_t to the policy z_p.
8. The policy z_p then selects the states Az_t for the variables A associated with the actuators' states. Az_t will also be passed to the interpreter z_i so that its value can be included into e_{t+1}.
9. The state changes of the actuators result in the changes of the monitored state with a new value $sz_{0,t}$. The sensors capture the values of the environment states

and assign these values to the variables S that are associated with the environment states.

10. The observable states Oz_t containing the values of the actuators and the sensors, and are sent to the interpreter z_i so that $s_{z0,t}$ can be included into e_{t+1}.
11. If the inhabitant I is not satisfied with the updated monitored state $s_{z0,t}$, by defining A'_t, he/she can change the states of the actuators or he/she can define the target value for the monitored state. It is supposed that the definition of A'_t should not before the interpreter z_i receiving Oz_t, and should not be too late so that the changes of the monitored state could be finished before the arrival of time step $t+1$.
12. Afterwards, the system comes to the next time step $t+1$, and repeats the above process including capturing O_{t+1} and calculate r_{t+1}. The two values will be included into e_{t+1} so that e_{t+1} has its whole values : $e_{t+1} = \{O_t, Az_t, s_{z0,t}, O_{t+1}, r_{t+1}\}$. Therefore, the same with e_t, e_{t+1} is calculated at time step $t+1$ for the actions taken place at time step t.

In the following paragraphs, we respectively present in more detail the three modules of a service model : an interpreter, an RL algorithm, and a policy.

Interpreter

The interpreter contains three functions. The first function is presented as follows :

$$X \leftarrow f_x(\{S, A\}) = f_x(O), \tag{5.1}$$

where f_x is a function that selects the subset X from the observable states O, and X is used as input to the RL algorithm.

Second, the interpreter contains a reward function f_r that generates a reward r given the latest input transition e in the replay memory M. The Algo.2 shows the detail of this reward function f_r when the service is a target-defined service, meaning that the inhabitant directly specifies the target value $a'_0 \in A'$ for the monitored state : Knowing the updated monitored state s_{z0} resulting from the actuators changing their states to Az, the target monitored state a'_0, and that s_{z0} and a'_0 are included in the latest transition of M (line 7), if s_{z0} complies with a'_0, then a constant positive reward is returned by the reward function f_r. Otherwise, a negative reward will be returned.

Algorithm 2: calculate the reward when the service is a target-defined service

Result: reward
M : replay memory
s_{z_0} : updated monitored state, resulting from the actuators changing their states to the states proposed by the service
$a'_0 \in A'$: target monitored state defined by the inhabitant
r : reward representing the (dis)satisfaction of the inhabitant to s_{z_0}
c : certain predefined positive constant value
Function $f_r(M)$ **:**
    knowing that s_{z_0} and a'_0 are contained in the latest transition of M
    if s_{z_0} *complies with* a'_0 **then**
        return $r = c$
    else
        return $r = -c$ // or $r = -|s_{z_0} - a'_0|$, or $r = -|s_{z_0} - median(a'_0)|$ if a'_0 is a range
    end

This negative reward can be a constant negative value, or the negative difference between s_{z_0} and a'_0 if a'_0 is a scalar, or the negative difference between s_{z_0} and the median of a'_0 if a'_0 is a domain value. Using the median value as a target allows the monitored state to approach the median value and stay away from the two boundary values. Besides using the constant value as a reward, other reward functions that provide more varied rewards can be used.

However, the inhabitant can also express his/her dissatisfaction by changing the actuators' states from Az to A', in this case, the service is a target-undefined service. The corresponding reward function is shown in Algo.3, knowing the actuators' states Az proposed by the service, the updated monitored state s_{z_0} resulting from the actuators changing their states to Az, the actuators' states A' proposed by the inhabitant,

Algorithm 3: calculate the reward when the service is a target-undefined service

Result: reward
M : replay memory
Az : actuators' states proposed by the service
s_{z_0} : updated monitored state, resulting from the actuators changing their states to the states proposed by the service
A' : actuators' states defined by the inhabitant
r : reward representing the (dis)satisfaction of the inhabitant to s_{z_0}
c : certain predefined positive constant value
Function $f_r(M)$ **:**
    knowing that the latest Az, s_{z_0}, and A' are contained in the latest transition in the replay memory M
    if Az *equals* A' **then**
        return $r = c$
    else
        return $r = -c$
    end

and that Az, s_{z_0}, and A' are in the latest transition of M (line 8). The reward function contained in the interpreter tells whether Az and A' are equal. If they are equal, a positive constant reward value is returned, as shown in line 10. Otherwise, a negative constant reward value is returned, as shown in line 12. Similarly to Algo.2, the reward function f_r in Algo.3 can also define other algorithms which can generate more varied reward values.

RL algorithm - training process

The RL algorithm proposes action quality values for each possible state of each actuator. In our study, DQN (Deep Q learning) [60], where Q-learning is based on deep learning algorithms including MLPs and LSTM cells as presented in Chapter 3, is used as the RL algorithm. The policy is applied to select the state that each actuator will take. The RL algorithm contains two phases : the training process and the testing process.

The training phase is about how the RL algorithm updates its parameters based on the replay memory. The detailed process of the training phase is shown in Algo.4. To train the RL algorithm, as shown in line 19, we first sample a subset of replay memory called E from M. E contains L transitions. Then for each transition e in E (line 20), we first use the interpreter (Eq.5.1) to select the RL algorithm's input state X (line 22). Then the RL algorithm RL$(X;\theta)$ with the parameters θ generates the action quality values Q_o for all states of all actuators by considering the states X (line 23).

Next, after the actuators have changed states to Az, using Eq.5.1, the interpreter is used another time to select the updated input X_u of the target RL algorithm. The input of Eq.5.1 contains the updated monitored state s_{z_0} and other observable states O_{-s_0} that maintain unchanged except the monitored state, where $O_{-s_0} = O - s_0$ (line 24). The target RL algorithm RL$(X_u;\theta')$ with the parameters θ' generates the new action quality values for all states of all actuators considering the updated states X_u (line 25). Then, using Eq.3.4 in Chapter 3 and considering the original Q_o and the updated Q_u, Q_o is changed to Q'_o (line 26).

Then, a mean square error loss function is attained by considering Q_o and Q'_o (line 27). We use the back gradient descent of the loss function on the parameters θ to update θ (line 28). We perform the above operation for each transition e in E. If

Algorithm 4: training process of RL algorithm

Result: updated RL algorithm
1 M : replay memory
2 e : one transition in the replay memory
3 S : states captured by sensors
4 s_0 : monitored state before the actuators change their states
5 sz_0 : updated monitored state, resulting from the actuators changing their states to the states proposed by the service
6 s'_0 : updated monitored state, resulting from the actuators changing their states to the states proposed by the inhabitant
7 A : actuators' states
8 Az : actuators' states proposed by the service z
9 X : states considered by the service
10 X_u : updated states considered by the service
11 r : reward representing the (dis)satisfaction of the inhabitant to the updated monitored state
12 $\mathsf{RL}(X;\theta)$: main RL algorithm whose input is X, and its parameters to be updated are θ
13 $\mathsf{RL}'(X_u;\theta')$: target RL algorithm whose input is X_u, and its parameters θ' is set to be θ every constant C time steps
14 L : batch size representing the number of transitions used to train the RL algorithm
15 α : the learning rate representing the gradient descent speed
16 ep : current epoch
17 C : certain predefined integer number
18 **Function** $f_{training}(M)$ **:**
19 randomly sample L transitions from M and name them as E
20 **for** e *in* E **do**
21 select O, A_z, sz_0, r from e
22 $X \leftarrow f_x(\{O\})$
23 $Q_o \leftarrow \mathsf{RL}(X;\theta)$
24 $X_u \leftarrow f_x(\{O_{-s_0}, sz_0\})$
25 $Q_u \leftarrow \mathsf{RL}'(X_u;\theta')$
26 $Q'_o \leftarrow$ run Eq.3.4
27 $f_{loss}(Q_o, Q'_o;\theta) \leftarrow$ mean square error loss function
28 $\{\theta \leftarrow \theta - \alpha \frac{\partial f_{loss}}{\partial \theta}\}$
29 **end**
30 **if** $ep\%C \; equals \; 0$ **then**
31 $\theta' \leftarrow \theta$
32 **end**

the current epoch ep is an integer multiple of C as shown in line 30, we update the parameters θ' of the target RL algorithm to the parameters θ of the main RL algorithm, as shown in line 31.

RL algorithm - testing process

After obtaining the updated main RL algorithm with parameter θ, we can predict the action quality values of all possible states of all actuators. As shown in Algo.5, to predict the action quality values of all possible states of all actuators, we first collect the observable states O. Then, based on Eq.5.1, we use the interpreter to select the input state of the RL algorithm X from O as shown in line 5. Finally, considering the

Algorithm 5: testing process of RL algorithm

Result: action quality values
O : observable states
X : input states considered by the service
RL(X ;θ) : main RL algorithm whose input is X, and its parameters to be updated are θ
Function $f_{evaluation}$ (O) **:**
    $X \leftarrow f_x(\{O\})$
    $Q \leftarrow$ RL(X ;θ))
    return Q

selected X, the RL algorithm RL(X ;θ) yields the action quality values of all possible states of all actuators as shown in line 6. The generated Q is returned and sent to the policy module to select new states for the associated actuators.

Policy

The policy we use is the ϵ-greedy policy. Its principle can be found in Algo.6. We first define a constant C_p, which is between 0 (exclusive) and 1 (exclusive), as shown in line 7. Then, we uniformly randomly generate a number ϵ ranging between 0 (inclusive) and 1 (inclusive), as shown in line 8. If ϵ is not less than C_p (line 9), the policy selects the states whose action quality value is the maximum for each actuator (line 11). Otherwise, if ϵ is less than C_p, the policy randomly selects states for each actuator by following the uniform distribution. The ϵ-greedy policy ensures a balance between exploitation (selecting the states with the maximum action quality values) and exploration (randomly selecting states for actuators) [46].

Algorithm 6: ϵ-greedy policy for selecting the states of the actuators

Result: states of the actuators
Q : action quality values of all states of each actuator, where $Q = \{Q^1, Q^2, \cdots, Q^k, \cdots\}$
Q^k : action quality value of all states of the K_{th} actuator, where $Q^k = \{q_0^k, q_1^k, \cdots\}$
S : environment states
A : states of actuators, where $A = \{A^1, A^2, \cdots\}$
A^k : possible states of k_{th} actuator, where $A^k = \{a_0^k, a_1^k, \cdots, a_i^k, \cdots\}$
Function f_{policy} (Q) **:**
    define a constant number C_p, where $0 < C_p < 1$
    uniformly randomly generate a real number ϵ, where $0 \leq \epsilon \leq 1$
    if $\epsilon \geq C_p$ **then**
        for Q^k *in* Q **do**
            select a_i^k whose q_i^k is the maximum in Q^k
        end
    else
        uniformly randomly select a state a_i^k from all available states of A^k
    end
    return selected states of each actuator

Dealing with min-max constraints

Some target-defined services may also need to satisfy another secondary requirement in addition to reaching the target value. This is the case, for example, of temperature services where the secondary requirement is energy saving. The present document does not cover this kind of services. However, we present a partial solution for a service that constrains the use of the actuator with lower priority. To that end, we define an alternative reward function which adapts the reward to the actions that satisfy this constraint. This alternative reward function is presented in the section dealing with service pretraining which will be introduced in a subsequent section.

Deployment of an RL-based service in the real world

To deploy the above process in the real world, we need the service module, the devices and the environment. Since the devices and the environment are provided by the real world, only the service with its three modules : interpreter, RL algorithm, and policy, needs to be implemented by the developer. The implementation of the three modules in the real world can be seen in the presentations of the above sections "Interpreter", "RL algorithm - training process", "RL algorithm - testing process", and "policy".

However, the service needs time to be well-trained. In addition, the process of proposing actuators' states by the service is almost random at the beginning of the training process. Moreover, the training process of this system involves interacting with the inhabitant. As a result, the inhabitant can be frequently interrupted to express his/her (dis)satisfaction with the updated monitored state. The frequent interruption may disturb the inhabitant, so a good user experience cannot be guaranteed.

To address these issues, we propose to pretrain this system in a simulated environment. When the system is well-trained, it is deployed in the real world and continues its training process. Pretraining in the simulation allows the service to gain some knowledge of the preferences of an inhabitant, reducing service interruption to the inhabitant when the service is deployed in the real world. To further reduce the number of interruptions, it is recommended that the inhabitant's profile in the simulated experiment matches the preferences of the real inhabitant as closely as possible. However, our current study does not address how to make the simulated inhabitant's profile as close as possible to the real inhabitant's preferences. Instead, we view this

as a promising perspective for future studies. Since we do not have access to a real smart home, our study only focuses on the logic for creating one (or multiple) smart home service in the simulation.

5.4 Simulated simple smart home system

When deploying the system in the real world, the developer only needs to design the service module. However, in the simulated experiment, the developer must simulate the environment in addition to implementing the service module. Theoretically, in the simulation, the service can be a target-defined or target-undefined service, where a simple simulated experiment is conducted at the end of this chapter (Section 5.6) to prove that the RL-based target-defined and target-undefined service can adapt to the simulated inhabitant's profile.

When we try to simulate the environment, it is easy to simulate the inhabitant profile by specifying the values for the target assignment variable a'_0 associated with the corresponding monitored state for a target-defined service. Nevertheless, it is complex to simulate the environment for a target-undefined service, as it is difficult to compute inverse equations to obtain the states of the actuators that should be set for the target variables $\{a'_1, a'_2, \cdots\}$ within the inhabitant profile. Therefore, in the rest of the document, we mainly focus on the target-defined services in the simulation.

Designing the service module in the simulated experiment for a target-defined service is the same as in the real world. The implementation details can be found in the "Interpreter", "RL algorithm - training process", "RL algorithm - testing process", and "policy" in Section5.2. As for the simulated environment, different services correspond to different environments. Therefore, to simulate the environment of a specific service, we should study the existing work on the physical phenomena of the environment states associated with that service. The simulated environments of some services are provided in the next section. Regardless of the type of service, in our work, we divide the simulated environment states into three categories :

Definition 5.4.1: Simulated environment

A simulated environment for a given service is composed of :

- an input scenario, which is a collection of time functions that produce values for each sensor variable in S_{-0} at each time step, where $S_{-0} = S - \{s_0\}$.

- a transfer function which calculates the value of the variable representing the monitored state based on the values of the sensor and actuator variables.
- an inhabitant profile, which is a function defining the target value of the variable a'_0 with respect to the inhabitant states, with some additional constraint if necessary.

Fig.5.7 shows us the simulated simple smart home system. We can see that the simulated environment contains an input scenario, a transfer function, and an inhabitant profile. The sensor variables S capture the values of the environment states simulated by the input scenario (simulating the inhabitant state and the physical environment states other than the monitored state) and the transfer function (simulating the monitored state). The actuator variables A represent the states of the actuators. Their values and the values of S serve as input for the transfer function. The inhabitant profile specifies the target value of the monitored state when the inhabitant is in a particular state.

Compared to Fig.5.5 in the real world, the simulation in the rest of the document (except the simple experiment conducted in Section 5.6 to prove that theoretically the RL-based service can be both target-defined and target-undefined) only simulates how to select the target variable $A' = \{a'_0\}$ of the monitored state for the inhabitant profile, and not how to choose $A' = \{a'_1, a'_2, \cdots\}$. This is because it is difficult to dynamically solve the inverse equations to obtain the value of $A' = \{a'_1, a'_2, \cdots\}$ that leads to the target value of the monitored state, given the values of the other environment

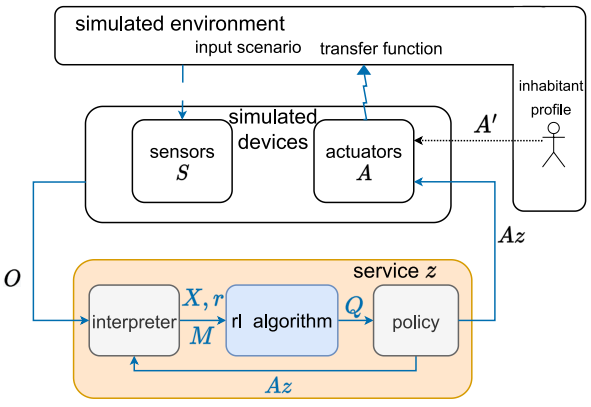

FIGURE 5.7 – Simulated RL-based simple smart home system

states.

We present how the three components of a simulated environment can be simulated in a general way. First, the input scenario F_{-0} containing a set of functions can be represented as :

$$S_{-0,t} \leftarrow F_{-0}(t), \tag{5.2}$$

where $S_{-0,t}$ is a set of sensor variables capturing the values of the environment states except the monitored state at time step t.

Then, the transfer function f_m can be represented as follows :

$$s_{0,t} \leftarrow f_m(S_t, A_t) = f_m(O_t), \tag{5.3}$$

where S_t are the sensor variables capturing the values of available environment states at time step t, A_t are the actuator variables having values of the states of the actuators, $O_t = S_t \cup A_t$ are the observable states, and $s_{0,t}$ is a sensor variable capturing the value of the monitored state at time step t.

Finally, we need to simulate the inhabitant profile $H(us)$ that describes the variable representing the target value a_0 for the monitored state s_0. The simulated $H(us)$ depends on the simulated inhabitant state variable us. It corresponds to the preferences for the monitored state of the real inhabitant when the service is in the real environment. To simulate $H(us)$, we use a rules based approach. Therefore, $H(us)$ can be expressed as follows :

$$\begin{aligned}
&\textit{if the } us \textit{ is } us_0\textit{, then the target monitored state } a_0 \textit{ is } a_{0,0}\textit{;}\\
&\textit{if the } us \textit{ is } us_1\textit{, then the target monitored state } a_0 \textit{ is } a_{0,1}\textit{;}\\
&\textit{if the } us \textit{ is } us_2\textit{, then the target monitored state } a_0 \textit{ is } a_{0,2}\textit{;}\\
&\textit{if the } us \textit{ is } us_3\textit{, then the target monitored state } a_0 \textit{ is } a_{0,3}\textit{.}\\
&\dots
\end{aligned} \tag{5.4}$$

where $a_{0,0} \cdots a_{0,3}$ can be scalar values or domain values. When we try to simulate the inhabitant's profile for different services, the target values for the corresponding monitored states are defined considering the experience of a realistic inhabitant. Since H_{us} is simulated by knowledge, the smart home system shown in Fig.5.7 is a hybrid system that combines the simulated knowledge-based inhabitant profile and the data-driven RL algorithm-based service.

5.5 Simulated simple smart home system examples

In this section, we present several simple smart home systems, each containing a different service : a light intensity service, a temperature service, and an air quality service. Moreover, some constraint may be added to each of these services. In our study, the one that constrain the use of the actuators with lower priority, is added to each of these services, assuming that each of these services contains one lower prioritized actuator. This constraint can be considered as the composition of the inhabitant profile. The constraint and the target monitored state contribute to calculating a reward using an ad-hoc reward function. This ad-hoc reward function differs from the one in Algo.2 for the general situation without constraint.

As described in Section 5.4, the service module design process for a simple smart home system in the simulation is the same as in the real world. The details can be found in Section 5.2 of the"Interpreter", "RL algorithm - training process", "RL algorithm - testing process", and "policy". Therefore, we only simulate the environment for each associated service. When simulating the environment, we neglect effects with low impact. For example, we do not consider the heat generated by turning on the lamp. In addition, we consider two types of inhabitant profiles : The first is the inhabitant profile specifying the target values for the monitored state depending on the inhabitant's state. The second profile is based on the first one and constrain the use of the actuator with lower priority. The reward function in Algo.7 corresponding to the first type of the inhabitant profile is the application of Algo.2, while the reward function for the second type of the inhabitant profile is an ad-hoc function and described in Algo.8.

5.5.1 Simulated environments for services

The involved variables and the states they are associated with are as follows : (1) us : inhabitant state. (2) le : outdoor light intensity. (3) te : outdoor temperature. (4) ae : outdoor air quality. (5) lr : indoor light intensity. (6) tr : indoor temperature. (7) ar : indoor air quality. (8) lp : state of the lamp. (9) cur : the curtain state. (10) ac : state of the air conditioner. (11) win : the window state. (12) ap : state of the air purifier. (13) wct : working duration for windows and curtain. (14) act : air conditioner working duration. (15) apt : air purifier working duration. lr, tr and ar are the monitored states that light, temperature and air quality services attempt to

adjust.

Light intensity service

The light intensity service takes us and le as input and uses a policy to select lp and cur as output. The selected lp and cur are used to change lr, the monitored state of the light intensity service. A predefined reward function generates a reward r_{light} to describe whether the updated value of lr corresponds to the inhabitant profile. Then, the RL algorithm is trained on the replay memory containing a set of transitions. Each transition contains the current environment states us and le, the proposed states by the service for the actuators lp and cur, the updated monitored state for lr, and the reward r_{light}. The updated RL algorithm is used to repeat the above process. The components of the corresponding simulated environment are as follows :

(1) the input scenarios contains the functions to produce values for us and le.

(2) the transfer function is the function to calculate the value for lr considering le, lp and cur.

(3) the inhabitant profile H_{light} is the function to define the target value for lr.

us_t is randomly generated at time t by following the uniform distribution : $us_t = U_{int}(0, n_{us})$, where n_{us} is the total number of possible states of the inhabitant and $U_{int}(0, n_{us})$ is a uniform distribution that randomly generates an integer between 0 inclusive and n_{us} exclusive.

le within a day is simulated with a Gaussian distribution [67, 69] : $le_t = \mathcal{N}(amplititude = 600, mean = 12, stddev = 3) + 5 \cdot U(0, 1)$, where $\mathcal{N}$ denotes the Gaussian distribution. Also, some noise is added to le_t, which is simulated using a uniform distribution with a maximum value of 5. To simplify the experiment, le_t is generated every 5 minutes each day.

The output of the RL-simulated light intensity service is lp_t and cur_t. lp_t can be chosen among multiple levels represented by integers, with level 0 indicating that the lamp is off and other levels indicating that the lamp is on. The light intensity that lp_t can provide is $\beta \cdot lp_t$, where $\beta = 100$ is the light intensity that one level can provide. cur_t has three possible values : 0, $1/2$,1. They mean that the curtain is closed, half open, and fully open, respectively. The calculation of the indoor light intensity simplifies the complex interactions between natural and artificial lighting in the indoor space, for example, forgetting the size and orientation of the window and

the presence of shading devices. Therefore, in our study, the indoor light intensity lr_t at time step t is expressed as :

$$lr_t = \beta \times lp_t + le_t \times cur_t \tag{5.5}$$

The inhabitant profile H_{light}, which describes the target indoor light intensity with respect to the inhabitant state us, is as follows :

if the inhabitant is absent, then the indoor light intensity is 0 lux;
if the inhabitant is working, then the indoor light intensity is between 250 lux to 350 lux;
if the inhabitant is seeing a movie, then the indoor light intensity is between 350 lux to 450 lux;
if the inhabitant is sleeping, then the indoor light intensity is 0 lux.

The inhabitant profile H_{light} constraining the use of the actuator with lower priority is :

constraining the use of the actuator (the lamp) with lower priority,
if the inhabitant is absent, then the indoor light intensity is 0 lux;
if the inhabitant is working, then the indoor light intensity is between 250 lux to 350 lux;
if the inhabitant is seeing a movie, then the indoor light intensity is between 350 lux to 450 lux;
if the inhabitant is sleeping, then the indoor light intensity is 0 lux.

Temperature service

The temperature service is responsible for adjusting tr to the target values set by the inhabitant. To do this, the temperature service learns to propose ac, win, cur and the corresponding working duration act, wct considering us and te. A reward r_{temp} is then generated by the predefined reward function to show whether the updated value of tr satisfies the preferred value of the inhabitant. The DQN uses the transitions, each of which contains the current environment states us, te, the proposed values by the service for ac, win, cur, act, wct, the updated monitored state for tr, and the reward r_{temp}. Once the RL is updated, it is used to restart the above procedure. The

components of the corresponding simulated environment are as follows :

(1) the input scenarios contain the functions to generate values for us and te.

(2) the transfer function is the function to calculate the value for tr considering $ac, act, cur, win, wct, tr$ and te.

(3) the inhabitant profile H_{temp} is the function to define the target value for tr.

The detailed description of the simulation of the associated environment state is : us_t is simulated in the same way as in the light intensity service. te_t is simulated with trigonometric functions [21] and expressed as :

$$te_t = A \cdot cos(B \cdot (x - D)) + C \tag{5.6}$$

where $A = -7, B = \pi/12, C = 19$ and $D = 4$ and x is the corresponding time with unit $hour$ at time step t ; the relation between x and t is : $x = t \cdot 5/60$ because the time interval between two time steps t and $t+1$ is 5 minutes. Since $ac_t, win_t, cur_t, act_t, wct_t$ are suggested by the temperature service, they need not be simulated. Here we give their possible values : (1) $ac_t \in \{0, -1, 1\}$ with 0 for off, 1 for heating, and -1 for cooling ; (2) $cur_t \in \{0, 1/2, 1\}$ with 0 for closed, $1/2$ for half open, and 1 for open. It has the same range as in the light intensity service. (3) $win_t \in \{0, 1\}$ with 0 for closed, and 1 for open ; (4) $act_t, wct_t \in \{i/10 \text{ for } i \in \{0, 50\}\}$.

The calculation of the indoor temperature is simplified by forgetting the complex interaction between the indoor and outdoor temperatures, for example, the insulation and ventilation of the building. Therefore, in our study, the calculation of the indoor temperature is as follows : The indoor temperature is affected by the outdoor temperature and the temperature supplied by the air conditioner. Assume that energy consumed by the air conditioner for one hour is $\phi^{ac} = 20 \cdot 735$ watt-hour ($W \cdot h$), where this value is to ensure an obvious temperature change after having the air conditioner work for one minute ; the specific heat of the air is constant and is $C_p = 1.005$; the air density is constant on average and has the value $\rho = 1.205(kg/m^3)$; and the room under study has the volume $V = 60(m^3)$. Thus, the energy produced after the operation of the air conditioner for the duration of $\Delta act_t(h)$ is :

$$Q_{ac,t} = \phi_{ac} \cdot \Delta act_t \tag{5.7}$$

Assuming that the resulting indoor temperature at time $t+1$ is tr_{t+1}, the total energy

that should be provided to maintain tr_{t+1} is thus equal :

$$Q_{heat,t} = C_p \cdot \rho \cdot V \cdot |tr_{t+1} - tr_t| \tag{5.8}$$

Besides, there is always the air circulation between the indoor and outdoor through the window and curtain, so the heat loss due to the air exchange is :

$$Q^{heat}_{loss,t} = L_t \cdot wct_t \cdot \rho \cdot C_p \tag{5.9}$$

where $L_t(m^3/s)$ is the air flow rate for indoor and outdoor air circulation and can be calculated as follows [3] :

$$L_t = d_h \cdot d_l \cdot \sqrt{\frac{2 \cdot g_r \cdot (\rho^e_t - \rho^r_t) \cdot h}{\lambda \cdot d_w \cdot \rho^r_t / d_l + \sum \varsigma \cdot \rho^r_t}} \tag{5.10}$$

where $d_h = 2(m)$, $d_l = 2(m)$, and $d_w = 0.2(m)$ are the height, length, and width of the window, respectively ; $g_r = 9.81(m/s^2)$ is the acceleration rate due to the gravity ; $\lambda = 0.019$ is the Darcy-Weisbach friction coefficient ; $\sum \varsigma$ is the summarized minor loss coefficient ; and ρ^e_t and ρ^r_t are the indoor and outdoor air densities as a function of the corresponding air temperature :

$$\rho^e_t, \rho^r_t = 1.293(kg/m^3) \cdot 273(K)/(273(K) + (te_t, tr_t)(^\circ C)) \tag{5.11}$$

As a result, we have the relation :

$$Q^{heat}_t = Q_{ac,t} + Q^{heat}_{loss,t} \tag{5.12}$$

According to Eq.5.7,Eq.5.8 and Eq.5.9, we can acquire the resulted indoor temperature :

$$tr_{t+1} = \frac{Q_{ac,t} + Q^{heat}_{loss,t}}{C_p \cdot \rho \cdot V} \pm tr_t \tag{5.13}$$

where "+" represents that the air conditioner is heating, and "-" denotes that it is cooling.

The simulated inhabitant profile H_{temp} describing the target indoor temperature when the inhabitant is in different state is :

if the inhabitant is absent, then the indoor temperature is constant ;
if the inhabitant is working, then the indoor temperature is between $23°$ *and* $25°$ *;*
if the inhabitant is seeing a movie, then the indoor temperature is between $20°$ *and* $22°$ *;*
if the inhabitant is sleeping, then the indoor temperature is between $17°$ *to* $19°$.

The simulated inhabitant profile H_{temp} that constrains the use of the actuator with lower priority is :

constraining the use of the actuator (the air conditioner) with lower priority,
if the inhabitant is absent, then the indoor temperature is constant ;
if the inhabitant is working, then the indoor temperature is between $23°$ *and* $25°$ *;*
if the inhabitant is seeing a movie, then the indoor temperature is between $20°$ *and* $22°$ *;*
if the inhabitant is sleeping, then the indoor temperature is between $17°$ *to* $19°$.

Air quality service

The air quality service attempts to control the monitored indoor air quality by adjusting ap, win, cur of the actuators and their working duration apt and wct. The air quality service takes the state us and ae as inputs and propose states for the actuators ap, win, cur, apt, wct. A reward r_{air} is generated by the predefined reward function to describe whether the updated value for ar matches the preferences of the inhabitant. The RL algorithm then uses the transitions to train itself, with each transition containing the current states us, ae, the proposed values for ap, win, cur, apt, wct, the updated value for ar, and the received reward r_{air}. Once the RL algorithm is updated, it is used to repeat the above process. The components of the corresponding simulated environment are as follows :

(1) the input scenarios contain the functions to generate values for us and ae.

(2) the transfer function is the function to calculate the value for ar considering $ap, apt, win, cur, wct, te, tr$ and ar.

(3) the inhabitant profile H_{air} is the function to define the target value for ar.

The detailed descriptions for simulating the above states are as follows : us_t is simulated in the same way as in the light intensity and temperature services. To simulate

ae_t, we construct a model to calculate ae_t at each time step from the atmospheric carbon dioxide dataset [1] from quasi-continuous measurements on American Samoa [74]. Since this is a real dataset, it can better reflect the changes of the carbon dioxide in the atmosphere. To build the model, we first do the imputation by replacing the anomalous data with surrogate data. In this study, we use the average of the corresponding data as the surrogate data. Then, an interpolation (a Python built-in interpolation function called interp1d within the Python library scipy.interpolate) is used to obtain a dataset sampled every 5 minutes instead of every hour. We will use this new dataset as the ae_t. The states proposed for the actuators are adjusted by the RL algorithm. The possible values for the actuators involved are : (1) $s_{ap,t} \in \{0, 60, 170, 280, 390, 500\}$ with 0 being turning off, and other numbers representing the values of the cubic meter air flow rate (m^3/h or CMH) of the air purifier ; (2) cur_t has the same range as in the example of light intensity and temperature services ; (3) win_t has the same range as in the example of temperature service ; (4) $apt_t, wct_t \in \{i/10 \textit{ for } i \in \{0, 50\}\}$.

The calculation of the monitored indoor air quality is simplified by forgetting several complex factors, for example, the ventilation of the building, indoor air pollutants, indoor humidity, and indoor temperature. The monitored indoor air quality is influenced by the outdoor air quality and the air purifier and can be calculated as follows [14] :

$$\begin{aligned} ar_{t+1} =& ar_t \cdot \left(1 - \frac{ap_t \cdot apt_t}{V} - \frac{L_t \cdot wct_t}{V} - \frac{n_{us,t} \cdot b_{us,t} \cdot \Delta x_{us}}{V}\right) + ae_t \cdot \frac{L_t \cdot wct_t}{V} \\ &+ s_{exha,t} \cdot \frac{n_{us,t} \cdot b_{us,t} \cdot \Delta x_{us}}{V} \end{aligned} \tag{5.14}$$

where V and L_t represent the same value as in the temperature service environment setting ; $s_{exha,t} = 38000(ppm)$ is a constant representing the CO_2 content in the exhaled air ; $n_{us,t}$ is the number of inhabitants in the room. In this study, $n_{us,t}$ is constant and has the value 1 ; $b_{us,t}$ is the CO_2 breathing rate of the inhabitant, whose value depends on the inhabitant's activity and can be found in Table 3 and Table 4 in [102]. In this article, we assume that the inhabitant is between 21 and 30 years old, so the physical activity level B corresponding to us is $B = \{0, 1.4, 4, 1\}$. The CO_2 breathing rate $b_{us,t}$ associated with us is $b_{us,t} \in \{0, 11.004(mg/s), 31.44(mg/s), 7.6635(mg/s)\}$. Δx_{us} is the inhabitant's breathing time, which is a constant value and is 5 minutes between two time steps.

1. The dataset can be downloaded from : https://we.tl/t-Lbj5PxK2bF

The inhabitant's profile H_{air} describing the target indoor air quality when the inhabitant is in different state for the air quality service is as follows with the unit being $\mu mol/mol$ or $parts\ per\ million\ (ppm)$:

if the inhabitant is absent, then the indoor air quality is constant ;
if the inhabitant is working, then the indoor air quality is between 100 ppm and 300 ppm ;
if the inhabitant is seeing a movie, then the indoor air quality is between 200 ppm and 400 ppm ;
if the inhabitant is sleeping, then the indoor air quality is between 100 ppm and 200 ppm.

Meanwhile, the inhabitant profile H_{air} constraining the use of the actuator with lower priority is :

Constraining the use of the actuator (the air purifier) with lower priority,
if the inhabitant is absent, then the indoor air quality is constant ;
if the inhabitant is working, then the indoor air quality is between 100 ppm and 300 ppm ;
if the inhabitant is seeing a movie, then the indoor air quality is between 200 ppm and 400 ppm ;
if the inhabitant is sleeping, then the indoor air quality is between 100 ppm and 200 ppm.

5.5.2 Design of reward functions

In this section, we design reward functions by comparing the updated monitored state with the target monitored state specified in the inhabitant profile for experiments without and with constraining the use of the actuator with lower priority in the inhabitant profile. To simplify the presentation, we show how to design the reward function for the temperature service. The reward functions for the light intensity and air quality services can be designed using the same principle.

Algorithm 7: Reward function without constraint

Result: Reward functions without constraint
1 us_t : inhabitant state at time step t
2 tr_t : indoor temperature at time step t
3 tr_t^* : the target indoor temperature at time step t
4 r_t : reward obtained at time step t
5 v : certain predefined positive numerical value
6 μ : certain predefined negative numerical value
7 **Function** `Reward_no_constraint`(us_t, tr_t, tr_t^*) :
8 **if** us_t *equals to certain state* **then**
9 **if** tr_t *satisfies the target indoor temperature* **then**
10 $r_t = v$
11 **else**
12 $r_t = \mu$ `// or` $r = -|tr_t - tr_t^*|$ `if` tr_t^* `is a scalar , or` $r = -|tr_t - median(tr_t^*)|$ `if` tr_t^* `is a range`
13 **end**
14 **end**
15 return r_t

Reward function for the service without constraint

When there is no constraint, we use the reward function as shown in Algo.7. It is an application of Algo.2 on the temperature service.

In our study, the target indoor temperature in the inhabitant's profile depends on the inhabitant's state. According to Algo.7, if the inhabitant is in a particular state (line 8), e.g. "working", and the updated indoor temperature tr_t complies with the inhabitant's target indoor temperature, then the reward at time step t is a some positive numerical value v (lines 9∼10) ; otherwise, the reward is a certain negative numerical value μ, or the negative Manhattan distance between tr_t^* and tr_t if tr_t^* is a scalar value, or the negative Manhattan distance between $median(tr_t^*)$ and tr_t if tr_t^* is a domain value (lines 11∼12).

Reward function for the service that constrain the use of the actuator with lower priority

If constraining the use of the actuator with lower priority is required in the inhabitant profile, we can express the new reward function in Algo.8 as follows :

If the inhabitant is in a some state (line 13), such as the inhabitant is "working", if the indoor temperature complies with the target indoor temperature (line 14), under this condition, if the air conditioner is off, or its working duration is zero, then the reward is a certain predefined positive numerical value v (lines 15∼16) ; otherwise, the reward is $v/2$ (lines 17∼18). However, if the indoor temperature does not comply with the target indoor temperature, then the reward is some negative value μ, or the

Algorithm 8: Reward function with constraint

Result: Reward function constraining the use of the actuator with lower priority

1 us_t : inhabitant state at time step t
2 tr_t : indoor temperature at time step t
3 tr_t^* : the target indoor temperature at time step t
4 ac_t : air conditioner state at time step t
5 act_t : work duration of air conditioner at time step t
6 win_t : window state at time step t
7 cur_t : curtain state at time step t
8 wct_t : window and curtain working duration at time step t
9 r_t : reward obtained at time step t
10 v : certain predefined positive numerical value
11 μ : certain predefined negative numerical value
12 **Function** `Reward_with_constraint`$(us_t, tr_t, ac_t, act_t, win_t, cur_t, wct_t, tr_t^*)$:
13 **if** us_t *equals to some state* **then**
14 **if** tr_t *satisfies the target indoor temperature* **then**
15 **if** ac_t *is off or* act_t *is 0* **then**
16 $r_t = v$
17 **else**
18 $r_t = v/2$
19 **end**
20 **else**
21 $r_t = \mu$ `// or` $r = -|tr_t - tr_t^*|$ `if` tr_t^* `is a scalar , or` $r = -|tr_t - median(tr_t^*)|$ `if` tr_t^* `is a range`
22 **end**
23 **end**
24 return r_t

negative Manhattan distance between tr_t^* and tr_t if tr_t^* is a scalar value, or the negative Manhattan distance between tr_t and the median of tr_t^* if tr_t^* is a domain value (lines 20∼21).

5.6 Adaptation to a simple target-undefined service and a simple target-defined service

As we pointed out in Section 5.4, theoretically, we can simulate both target-defined and target-undefined services. To prove this, we perform a simple simulated experiment to show that both RL-based target-defined and target-undefined services can adapt to the simulated inhabitant profile.

Simulated simple smart home system with a target-defined service

The inhabitant profile for a target-defined service only specifying the target values for the monitored state is as follows, where the profile is the same as that shown in Section 5.5.1 :

if the inhabitant is absent, then the indoor light intensity is 0 lux;
if the inhabitant is working, then the indoor light intensity is between 250 lux to 350 lux;
if the inhabitant is seeing a movie, then the indoor light intensity is between 350 lux to 450 lux;
if the inhabitant is sleeping, then the indoor light intensity is 0 lux.

We first train this RL-based system using Algo.4 on a training dataset randomly sampled from the replay memory storing the transitions of each historical time step. After the system is well trained, we use the RL-based light intensity service to propose actuators' states using Algo.5 for a test dataset where each data can be a transition sampled from the replay memory and different from those that are already within training dataset, or a transition of the newest time step. The corresponding average accuracy is 90.5%. The accuracy indicates the number of samples within the testing dataset for which the service can correctly propose actuators' states to satisfy the target monitored state, as a percentage of the total number of samples. The result shows that the structure in Fig.5.7 can create the light intensity service that meets the inhabitant's profile consisting of the target values for the monitored state.

Simulated simple smart home system with a target-undefined service

When the service is a target-undefined service, we suppose that we have the following inhabitant profile :

if the inhabitant is absent, then the lamp should be off, and the curtain should be closed ;
if the inhabitant is working, then the lamp should be at level 3, and the curtain should be off ;
if the inhabitant is seeing a movie, then the lamp should be at level 4, and the curtain should be fully open ;
if the inhabitant is sleeping, then the lamp should be off, and the curtain should be half open.

We use an RL-based system, as explained in Fig.5.7, to create a light intensity service that adapts to the inhabitant's profile defined above. We first train this RL-based

system on a training dataset. After the system is well trained, we use the service to propose actuators' states for a testing dataset. The corresponding accuracy is 90.75%. Therefore, we can conclude that the RL-based service can meet the inhabitant's profile specifying the preferred actuators' states.

From the above two experiments, we can conclude that the RL-based service can be used to simulate both target-defined and target-undefined services to adapt to the corresponding inhabitant profiles. Nevertheless, the inhabitant is not interested in the details of how to adjust the actuators. Moreover, in most situations, it is difficult for the inhabitant to calculate the inverse equations to find solutions for setting the actuators. This analysis is especially complex when the dynamics of the monitored state is complex and the number of associated actuators is large. Therefore, our study will only focus on the target-defined service, where the inhabitant profile specifies the target values for the monitored state when the inhabitant is in different states, as shown in Section 5.5.

5.7 Conclusion

We define a smart home as a home where the inhabitant's preferences are met by leveraging various services. Smart home services can propose actuators' states after considering the environment states sensed by sensors. Each smart home service aims to adjust a particular monitored state so that the value of the monitored state matches its target value explicitly or implicitly defined by the inhabitant.

To create services, instead of considering a general smart home, we start from the simplest situation of a simple smart home system containing only one service. This service can propose states for associated actuators to change the monitored state so that its value can adapt to the inhabitant's preferences. A smart home service can be target-defined or target-undefined. In a target-defined service, to express the (dis)satisfaction of the inhabitant to the updated monitored state, the inhabitant can define its target value. The service compares the updated monitored state value with its target to determine the (dis)satisfaction of the inhabitant. In a target-undefined service, if the inhabitant is unsatisfied with the updated monitored state, he/she directly changes the actuators' states. Otherwise, he/she does nothing.

Since it is important to consider the inhabitant's reactions and reduce his/her manual operations to build a user-friendly smart home, we propose an RL-based struc-

ture for the simple smart home system. Therefore, the home system contains a dynamic service that can dynamically propose actuators' states after considering the environment states detected by the sensors and the inhabitant's (dis)satisfaction. In this structure, each service can be considered as an RL agent that includes an interpreter, an RL algorithm, and a policy. The interpreter has three functions. First, it selects the states that serve as input to the RL algorithm. Second, it computes the reward regarding the inhabitant's reaction to the updated monitored state. Third, it stores the transition in a replay memory that is used to train the RL algorithm. The RL algorithm is used to calculate action quality values for all states of all actuators. The policy selects the state with the maximum action quality value for each actuator.

Such an RL-based service can already be deployed in the real environment. However, in this case, it will take time for the service to be well-trained. Moreover, frequent interruptions may occur during the adaptation to the inhabitant's profile, since proposing the states of the actuators at the beginning of the learning process can be considered as a random process. Therefore, we propose to pretrain this system using a hybrid knowledge-based/data-driven approach in simulation. The inhabitant profile is simulated by the rules based knowledge, and the service is modeled by an RL-based data-driven algorithm. The pretrained system is then deployed in the real world and continues its training with the real inhabitant. Our study focuses solely on developing the logic to create dynamic services in simulation. Furthermore, it is difficult for the inhabitant to manually compute the inverse equation to obtain the states that the actuators should have to satisfy the inhabitant's preferred monitored state. Therefore, in the simulation, the inhabitant profile only contains the target values of the monitored state, with some additional constraint. Thus, only the target-defined service is considered in the simulation.

This chapter also presents the process of simulating a simple smart home system, with each smart home system containing a specific service. In the simulation of a simple smart home system, the design process of the service modules for different services is the same as in the real environment. Therefore, we only simulate the environment that is different between services. Each simulated environment for each service includes an input scenario, a transfer function, and an inhabitant profile. The input scenario consists of functions that generate values for sensor variables other than the monitored state variable. The transfer function calculates the value for the monitored state variable considering sensor and actuator variables. The inhabitant

profile describes the target values for the monitored state variable when the inhabitant is in different states. It is optional with some additional constraint.

Finally, we present several simulated simple smart home system examples where each smart home system contains respectively a light intensity service, a temperature service, and an air quality service. For each service, we define two types of inhabitant profiles : The first one defines the target values for the monitored state variable in terms of different inhabitant states, and the second one is based on the first one and constrain the use of the actuator with lower priority. An ad-hoc reward function is defined for the second case where the inhabitant profile has the constraint.

6

Creation of multiple smart home services

Contents

6.1 Introduction

In a smart home, there are usually multiple services. Nevertheless, it is challenging for an inhabitant or developer to manually create these services. Several reasons exist for this difficulty. For example, the inhabitant or developers must manually deduce the actuators' states to adapt to the inhabitant's preferred monitored states while ensuring that no potential conflicts occur when these services act on the same actuators. Nevertheless, it is complex when the physical phenomena of the environment states are complex and the number of related actuators is large.

In Chapter 5, we proposed a smart home system based on RL,where each service is regarded as an agent and contains three modules : an interpreter, an RL algorithm,

and a policy. The created service is dynamic that can dynamically propose actuators' states by considering the environment states. In this chapter, extending the proposed single service based structure, we propose several multi-services based architectures with a collective name SHOMA (Smart HOme-based Multi-services Architectures). Services in each SHOMA architecture can dynamically propose conflict-free actuators' states by considering the environment states. They are also adaptive by changing their behavior to match the inhabitant's preferences after considering the inhabitant's reaction.

The process of deploying SHOMA architectures in the real world is the same as deploying the system with one single smart home service. It also involves two steps. The first step is to train the SHOMA architecture-based system in simulation, which can reduce the frequency of requiring the inhabitant's reactions to the updated monitored states and increase the training rate of the system in the real world. The second step is continuously training the SHOMA architecture-based system in the real world. Our study only focuses on the simulation part.

For the rest of the chapter [1], we first introduce the proposed architectures for creating multiple dynamic smart home services. Then, we run several simulated environments to evaluate the proposed architectures, analyze the corresponding results, and select the architectures with better performance. Next, we briefly present how the selected architectures can be deployed in the real world. Finally, we summarize the chapter.

6.2 Proposed architectures for creating multiple dynamic smart home services

To better illustrate the principle of SHOMA architectures, we first present a smart home system with three services. Then, we introduce how to use SHOMA architectures to create this system. Creating a smart home system with more or less than three services using SHOMA architectures is the same as the process with three services. In our study, the time interval of a time step is the same for all services. We do not consider the situation where the time interval of a time step is different in different services. Instead, solutions to such a situation are a promising perspective.

1. The original version of this chapter was published in 2022 IEEE International Conference on Machine Learning and Applications (ICMLA'22) and can be found in [104]

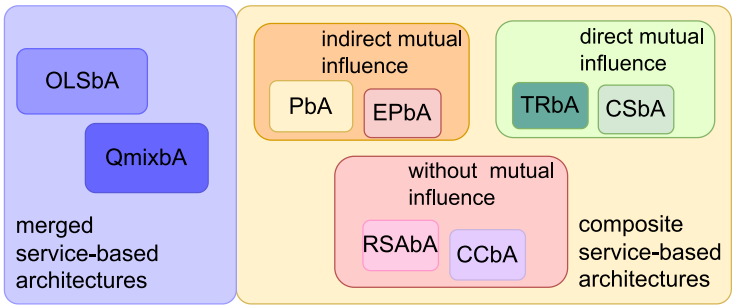

FIGURE 6.1 – Classification of SHOMA architectures

We denote the three services as z_1, z_2, and z_3. They belong to different types of services, and each can represent any specific smart home service. The actuators associated with each of them are : (1) $\{d_1, d_2, d_3\}$ for z_1 ; (2) $\{d_2, d_4, d_5\}$ for z_2 ; (3) $\{d_2, d_3, d_6\}$ for z_3, where d_2 is shared by the three services and d_3 is shared by z_1 and z_3. In addition, the states that each service considers are : (1) $O^{z_1} = \{o_1^{z_1}, o_2^{z_1}, \cdots\}$ for z_1 ; (2) $O^{z_2} = \{o_1^{z_2}, o_2^{z_2}, \cdots\}$ for z_2 ; (3) $O^{z_3} = \{o_1^{z_3}, o_2^{z_3}, \cdots\}$ for z_3. The states converted by the interpreter and used as input to the RL algorithm in each service are expressed as X^{z_1}, X^{z_2}, and X^{z_3}, where $X = X^{z_1} \cup X^{z_2} \cup X^{z_3}$. The corresponding target values defined by the inhabitant are denoted as $A'^{,z_1}, A'^{,z_2}$, and $A'^{,z_3}$, where $A' = A'^{,z_1} \cup A'^{,z_2} \cup A'^{,z_3}$. Services z_1, z_2, and z_3 are individually implemented by respecting the principle shown in Fig.5.5 in Chapter 5. Therefore, we have the following modules : interpreter IP^{z_1}, rl algorithm RL^{z_1} and policy PO^{z_1} for z_1 ; interpreter IP^{z_2}, RL algorithm RL^{z_2} and policy PO^{z_2} for z_2. The same is true for z_3.

Using the three services and their associated modules in different ways, we define eight architectures in SHOMA based on the concept of multi-agent RL (MARL) [133] to model a smart home system to create dynamic services : One Learning System-based Architecture (OLSbA), Qmix-based Architecture (QmixbA), Remove Shared Actuators-based Architecture (RSAbA), Common Controller-based Architecture (CCbA), Priority-based Architecture (PbA), Equal Priority-based Architecture (EPbA), Total Reward-based Architecture (TRbA), and Context Sharing-based Architecture (CSbA). As shown in Fig.6.1, these architectures can be divided into two types : "merged service-based architectures" and "composite service-based architectures". The difference between them depends on whether all services are merged as one single service when modeling the smart home system. For "composite service-based architectures", the architectures can be further divided into three subtypes : "indirect mutual influence", "direct mutual influence" and "without mutual influence".

The difference between them lies in how the shared actuators' states are determined. For "indirect mutual influence" and "direct mutual influence", each service proposes states for the shared actuators under the indirect or direct influence of other services. In "without mutual influence", the states of the shared actuators are determined by one service or controller, thus eliminating the influence between services.

6.2.1 Merged service-based architectures

The first architecture category is the merged service-based architecture, where all services of the smart home system are merged as one single service to regulate multiple monitored states. Two SHOMA architectures belong to this category : OLSbA and QmixbA.

One learning system based architecture

The OLSbA shown in Fig.6.2 models the entire smart home system as a single service z. In the initial time step, the interpreters $\mathsf{IP}^{z_1}, \mathsf{IP}^{z_2}$ and IP^{z_3} transform the observable environment states O^{z_1}, O^{z_2} and O^{z_3} into states X^{z_1}, X^{z_2} and X^{z_3}, which are used as input to the RL algorithm. The algorithm generates action quality values for all possible states of all available actuators. Then the policy PO^z uses a function to decide the state for each actuator, e.g., the greedy policy selects the state with the

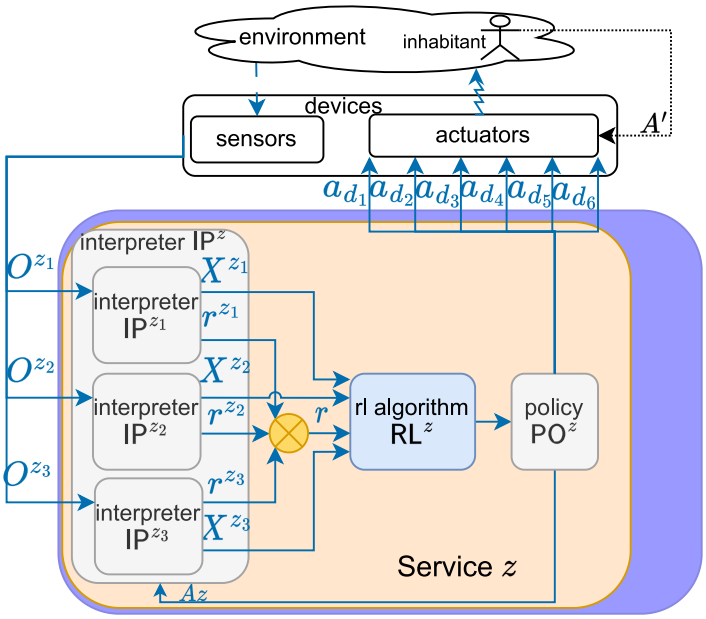

FIGURE 6.2 – Architecture of OLSbA

highest action quality value for the actuator. The monitored states update their values as the actuators change their states to the proposed ones. Based on the updated monitored states and the inhabitant's reactions A', IP^{z_1}, IP^{z_2} and IP^{z_3} respectively calculate the reward r^{z_1}, r^{z_2} and r^{z_3}. The original environment states, the proposed actuators' states $Az = Az^{z1} \cup Az^{z2} \cup Az^{z3}$ by the service, the updated environment states after actuators having changed their states to Az, and the total reward $r = r^{z_1} + r^{z_2} + r^{z_3}$ are collected as one transition and stored in a replay memory. The replay memory is used to train the RL algorithm in z. Then, the updated RL algorithm or the updated service z is used to repeat the above process.

Qmix based architecture

Based on the principle of [105], we propose QmixbA shown in Fig.6.3. To determine the states of the actuators, QmixbA introduces a learning system called Qmix that takes the action quality value outputs $Q^{z_1}, Q^{z_2}, Q^{z_3}$ from the RL algorithms RL^{z_1}, RL^{z_2} and RL^{z_3} as inputs and a hyper neural network [55] to determine the values of its parameters. Moreover, the total reward $r = r^{z_1} + r^{z_2} + r^{z_3}$ is used to determine the learning direction of Qmix and the hyper neural network. Finally, Qmix generates the action quality values for all possible states of each actuator, and a policy is used to select the final states to which each actuator switches.

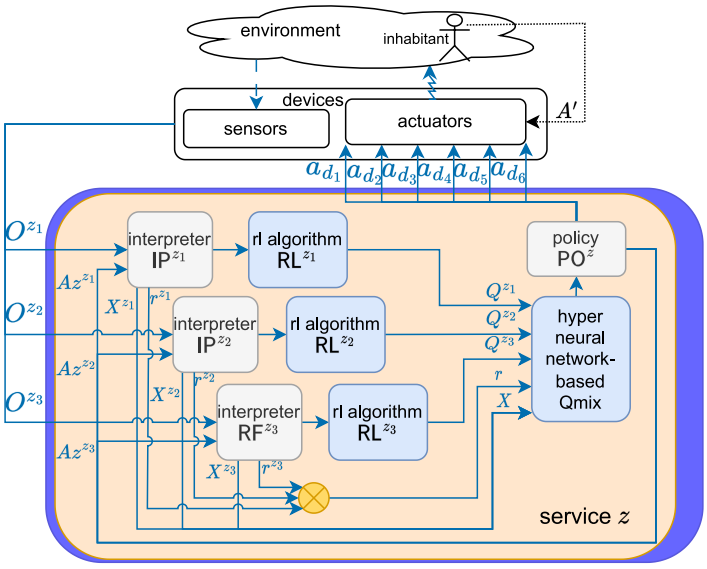

FIGURE 6.3 – Architecture of QmixbA

6.2.2 Composite service-based architectures

In composite service-based architectures, each monitored state is controlled by a service. Since a service controls one or more actuators, certain actuators may be shared by multiple services. Therefore, conflicts can be generated when these services simultaneously propose different states for the same actuators. Based on the mechanisms used to determine the states of these shared actuators, "composite service-based architectures" can be further divided into three subcategories : "without mutual influence", "indirect mutual influence", and "direct mutual influence".

Without mutual influence

"without mutual influence" refers to architectures where shared actuators are controlled by only one service or controller, which includes two architectures : RSAbA and CCbA.

Remove shared actuators based architecture In RSAbA, only one sharing service is maintained to determine the states of the shared actuators. The service that is maintained is the one whose corresponding dynamic characteristic about the monitored state is the simplest. In the given example with multiple services, we assume that the order of complexity of each service is : $complexity(z_1) < complexity(z_2) < complexity(z_3)$,

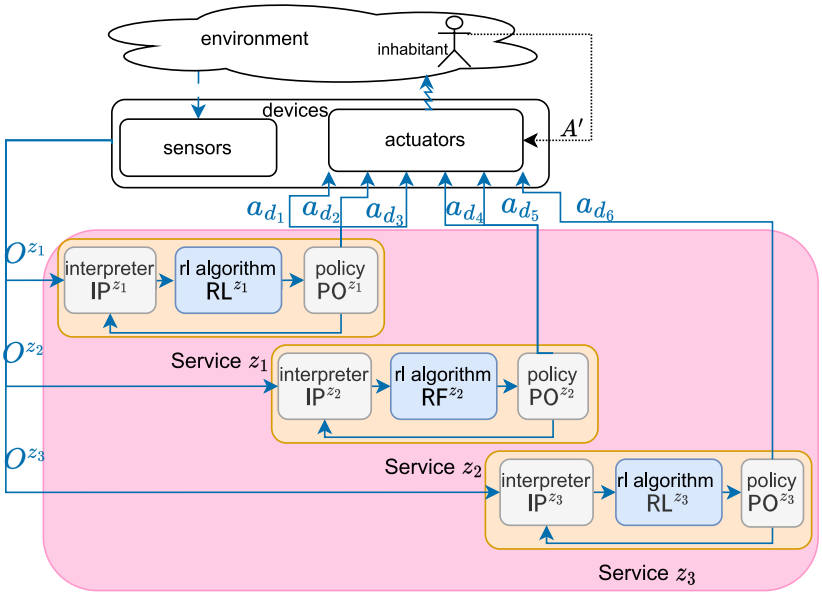

FIGURE 6.4 – Architecture of RSAbA

therefore, d_2 and d_3 are controlled by z_1. Thus, RSAbA can be expressed in Fig.6.4.

Common controller based architecture CCbA defines a common controller modeled by an RL algorithm for each shared actuator. As shown in Fig.6.5, because d_2 is shared by z_1, z_2, and z_3, a common controller CC_{d_2} is defined for d_2. CC_{d_2} takes the inputs of the RL algorithms of the sharing services $X = X^{z_1} \cup X^{z_2} \cup X^{z_3}$ and the total reward $r_{d_2} = r^{z_1} + r^{z_2} + r^{z_3}$ as its input, and proposes the state for the actuator d_2. The same principle can be applied for the actuator d_3. Because it is shared by z_1 and z_3, a common controller CC_{d_3} is defined. CC_{d_3} takes the inputs of the RL algorithm of the corresponding sharing services $X^{z_1,z_3} = X^{z_1} \cup X^{z_3}$ and the total reward $r_{d_3} = r^{z_1} + r^{z_3}$ as input, and proposes the state for the actuator d_3.

Indirect mutual influence

The category "indirect mutual influence" includes architectures where each service proposes states of the shared actuators under the indirect influence from other services. The final states of the shared actuators are determined by considering the propositions of all services. They can influence the states of the shared actuators proposed by each service in the next time step. Therefore, the final states of these shared actuators can be considered as the indirect influence of other services on each service.

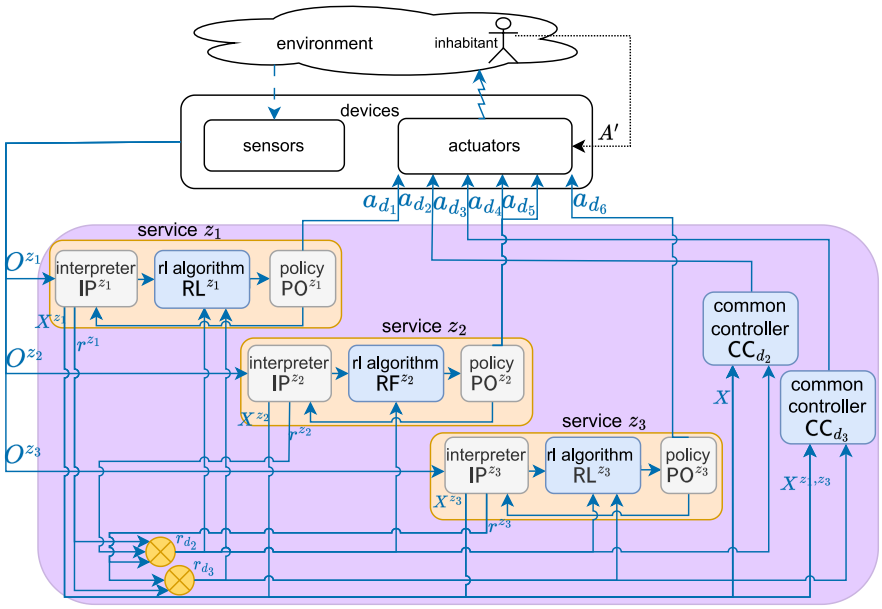

FIGURE 6.5 – Architecture of CCbA

PbA and EPbA are two architectures of this type.

Priority-based architecture In PbA, a priority calculator modeled by an RL algorithm is defined for each shared actuator. Each priority calculator proposes priorities for services that share the same actuator. Fig.6.6 in yellow shows how z_1, z_2 and z_3 can be modeled with PbA : Each service receives its associated observable states as input. Then, for non-shared actuators, each service directly proposes the states. For each of the shared actuators, the action quality values of all possible states proposed by the sharing services are multiplied by the priorities of the corresponding sharing services to obtain the weighted action quality values. These priorities are proposed by the corresponding priority calculator. The state with the highest action quality value is then selected for this shared actuator.

For example, d_2 is shared by z_1, z_2 and z_3. Each service uses an RL algorithm to propose action quality values $Q_{d_2}^{z_1}, Q_{d_2}^{z_2}$, and $Q_{d_2}^{z_3}$ for d_2. A priority calculator PC_{d_2} is defined for d_2. It takes the inputs of all RL algorithms $X = X^{z_1} \cup X^{z_2} \cup X^{z_3}$ as input and proposes three priorities $p_{d_2}^{z_1}$, $p_{d_2}^{z_2}$, $p_{d_2}^{z_3}$ whose sum is one. A policy based on the value function addition principle [116] (abbreviated as VFAP in this document) introduced

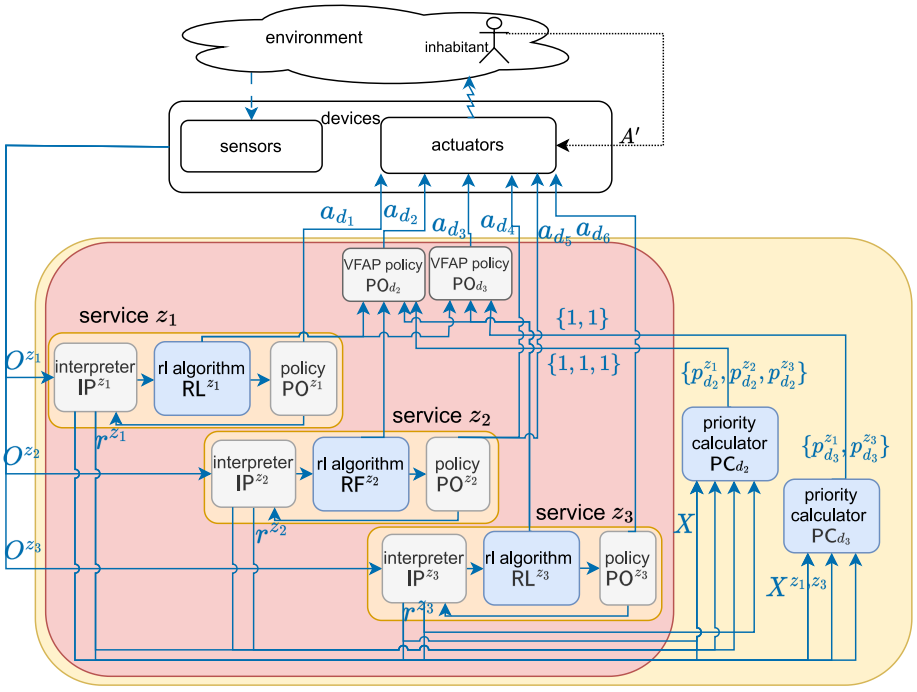

FIGURE 6.6 – Architectures of PbA (in yellow) and EPbA (in red)

in [116] is used to compute the final action quality value Q_{d_2} for d_2 :

$$Q_{d_2} = Q_{d_2}^{z_1} \cdot p_{d_2}^{z_1} + Q_{d_2}^{z_2} \cdot p_{d_2}^{z_2} + Q_{d_2}^{z_3} \cdot p_{d_2}^{z_3}, \tag{6.1}$$

It then selects the state with the highest action quality value in Q_{d_2} or randomly select a state as the new state to which d_2 will switch.

Equal priority based architecture The principle of EPbA is shown in red in Fig.6.6. Instead of computing the priorities of the services sharing the same actuators, EPbA directly summarizes the action quality values of the shared actuators and selects the states with the highest action quality values according to VFAP policy. We can also say that all sharing services have the same priority with value 1 to associate EPbA with PbA. The example of using VFAP policy to calculate the final action quality values for the shared actuator d_2 is :

$$Q_{d_2} = Q_{d_2}^{z_1} + Q_{d_2}^{z_2} + Q_{d_2}^{z_3}. \tag{6.2}$$

Then the state with the highest action quality value in Q_{d_2} or a random state is selected as the new state of d_2.

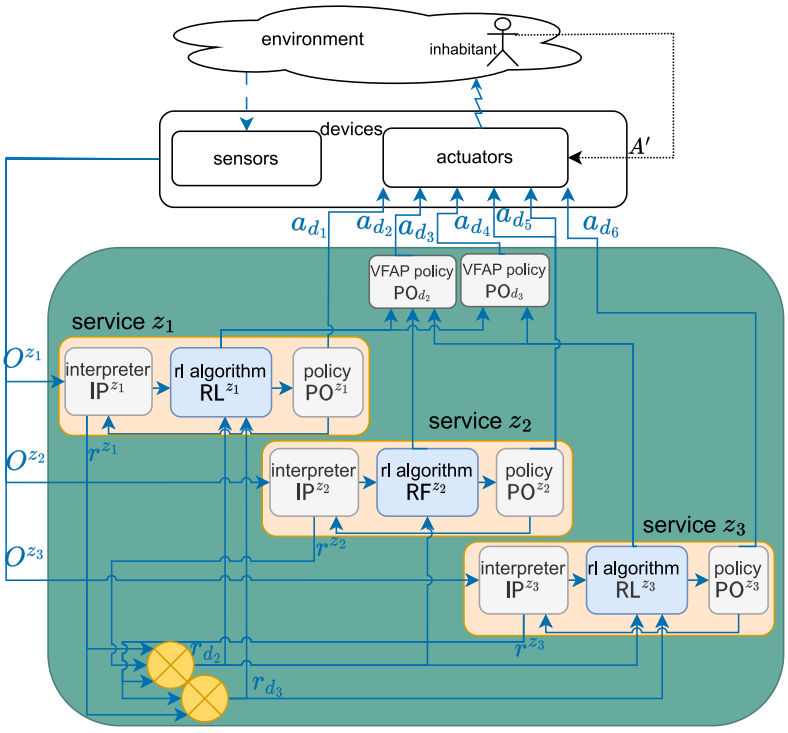

FIGURE 6.7 – Architecture of TRbA

Direct mutual influence

The direct mutual influence category includes architectures where each service proposes the states of shared actuators under the direct influence of other services. For example, each service that shares the same actuators considers the total reward that results from summing the rewards of these services. The total reward then directly influences the proposition that each service makes for the state of the shared actuator. Therefore, for each service sharing the same actuators, the total reward can be considered as a direct influence from the other services on that service. TRbA and CSbA are two architectures of this type of SHOMA.

Total reward based architecture TRbA, as shown in Fig.6.7, is a variant of EPbA. Unlike EPbA, TRbA adds the total rewards of all services sharing the same actuator to each service's RL algorithm as an additional reward to determine the states of that shared actuator. For example, d_3 is shared by z_1 and z_3. Therefore, the total reward $r_{d_3} = r^{z_1} + r^{z_3}$ is used as an additional reward and sent to RL^{z_1} and RL^{z_3}. By introducing the total rewards, the states of the shared actuators can be determined by ensuring that all sharing services receive high rewards simultaneously.

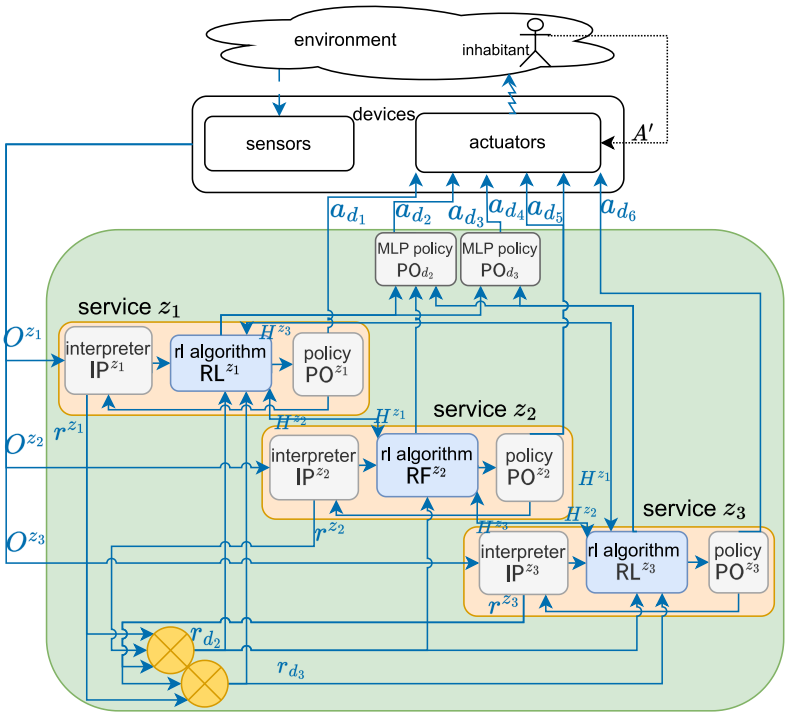

FIGURE 6.8 – Architecture of CSbA

Context sharing based architecture In CSbA, each service takes its associated environment states, its hidden states, and the hidden states of other services that use the same actuators as input. Each hidden state contains the information of the previous time step about the associated service. As shown in Fig.6.8, service z_1 considers its environment states O^{z_1}, its hidden states H^{z_1}, and the hidden states H^{z_2} and H^{z_3} of z_2 and z_3 ; while z_2 receives its environment state O^{z_2}, its hidden state H^{z_2}, and the hidden states H^{z_1} and H^{z_3} of z_1 and z_3 as input. The same principle applies to the z_3. In addition, as in TRbA, the total reward is used. For each service, the hidden states and total rewards are used as a direct influence from other sharing services that share the same actuator with it. The states of the non shared actuators depend only on the output of the action quality values of the associated services. The states of the shared actuators depend on the action quality values generated by a MultiLayer Perceptron (MLP)-based policy which uses the action quality values coming from the sharing services as its input.

6.3 Comparative experiment

To compare the performance of SHOMA architectures for creating smart home services, we conduct several simulated experiments based on three simulated target-defined services : a light intensity service, a temperature service, and an air quality service. Each service constrains the use of the actuator with lower priority. The simulated environments of the three services and their associated reward functions are presented in Section 5.5.

Among simulated light intensity, temperature and air quality services, the curtain is shared by the three services, where $cur^{light}, cur^{temp}, cur^{air}$ are the curtain state values respectively proposed by the three services. In addition, the window is shared by the temperature and air quality services, which respectively propose win^{temp}, win^{air} values for the window state. The same is true for the working duration of the curtain and the window. The temperature and air quality services share this working duration and respectively suggest wct^{temp}, wct^{air} values for this working duration.

6.3.1 Experiment metrics

To evaluate the SHOMA architectures and select the architectures with the best performance, we first compare all architectures with the temperature and air quality

services, with and without constraining the use of the actuator with lower priority in the simulated inhabitant profile. From this comparison, we select the architectures with better performance. Then, we evaluate the selected architectures with three services with and without constraining the use of the actuator with lower priority in the inhabitant profile, and determine the best performing architectures.

Following metrics are defined to evaluate the performance of the architectures : (1) Accuracy of each service to propose the states of the actuators correctly. (2) Accuracy of all services to propose the states of the actuators simultaneously correctly, which describes the inhabitant's satisfaction to the architecture. (3) Average of all accuracy, which denotes the general performance of an architecture. The above accuracy indicates the number of samples for which each service or all services correctly propose actuators' states so that the updated monitored states match the inhabitant's target values, as a percentage of the total number of samples.

6.3.2 Evaluation results

Evaluation results with two services

Fig.6.9 shows the result of the predefined metrics for the experiment without the constraint. It can be seen that EPbA generally performs better on all four metrics : the accuracy of the temperature service (red legend), the air quality service (yellow legend), the two services that correctly suggest the state of the actuators simultaneously (blue legend), and the general performance (green legend). RSAbA and CCbA perform best after EPbA. PbA is the fourth, followed by TRbA. OLSbA, CSbA, and QmixbA perform worst.

We then apply these architectures to the same test dataset but with the constraint. The result is shown in Fig.6.10. The result is almost the same as in Fig.6.9, except

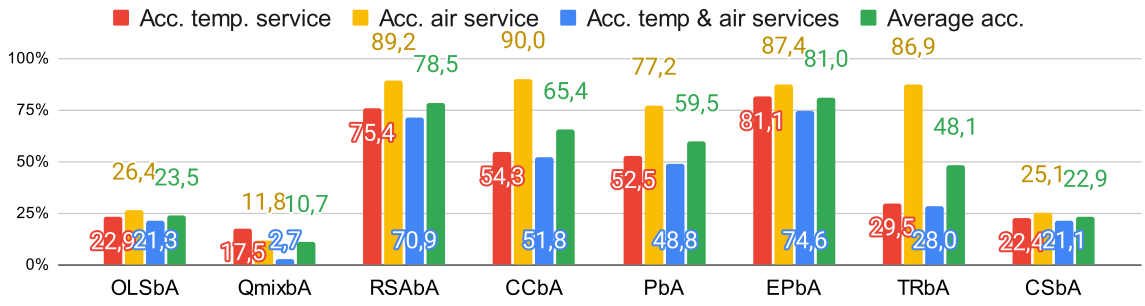

FIGURE 6.9 – Architecture evaluations without constraint and with two services

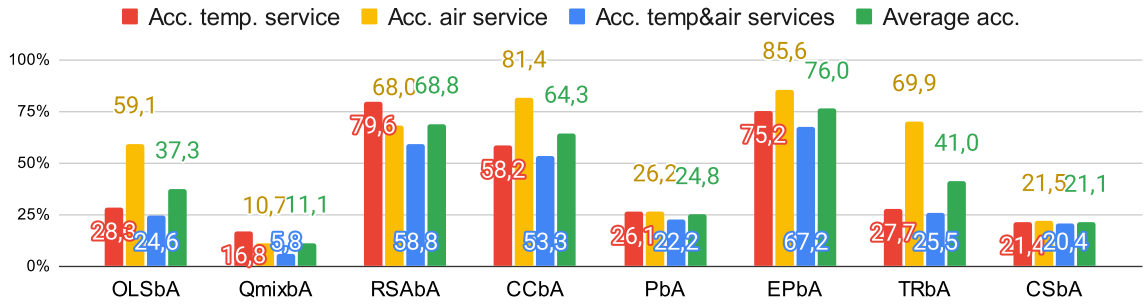

FIGURE 6.10 – Architecture evaluations with constraint and two services

that the performance of PbA drops sharply. With the exception of OLSbA and QmixbA, the other architectures also show slight performance degradation, but not as severe as PbA. To remove the doubt that adding the constraint in the reward function is not a good way to realize energy saving, we select architectures : RSAbA, CCbA, PbA, EPbA, and TRbA, with the general performance greater than or close to 50% based on the result in Fig.6.9, for three service-based experiments.

Experiment results with three services

Fig.6.11 shows the metric results : accuracy of the light intensity service (deep red legend), the temperature service (light red legend), the air quality service (yellow legend), the three services that correctly suggest the state of the actuators simultaneously (blue legend), and the average accuracy (green legend), without the constraint. Compared to Fig.6.9, the performance of RSAbA and PbA improves, which means that introducing the light intensity service helps to improve the learning performance of the temperature and air quality services. For RSAbA, this can be because introducing the light intensity service reduces the number of actuators controlled by the temperature and air quality services. For PbA, introducing the light intensity service may provide more information to decide which service should have higher priority. For EPbA, the performance in the two figures is almost the same, so introducing the light intensity service has no significant impact. The performance of CCbA and TRbA is still not as good as the other three architectures : The performance of CCbA has slightly decreased. The performance of TRbA has dropped significantly, so introducing the light intensity service makes services with TRbA architecture harder to learn the system patterns.

Fig.6.12 shows the results when there are three services and with constraining the

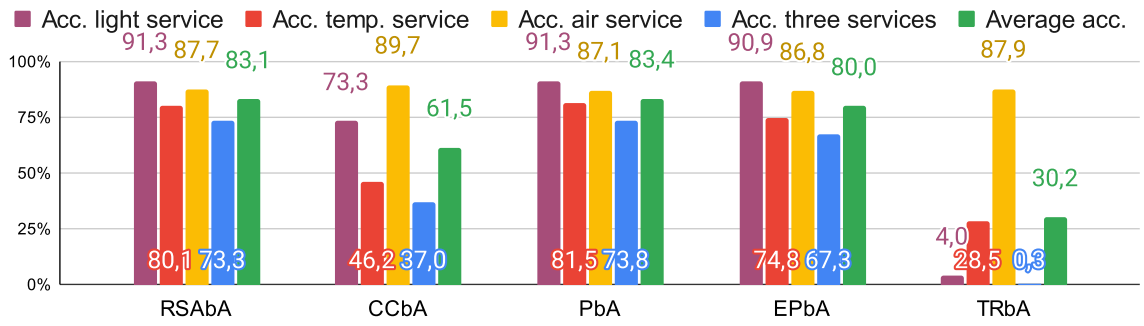

FIGURE 6.11 – Architecture evaluations without constraint and with three services

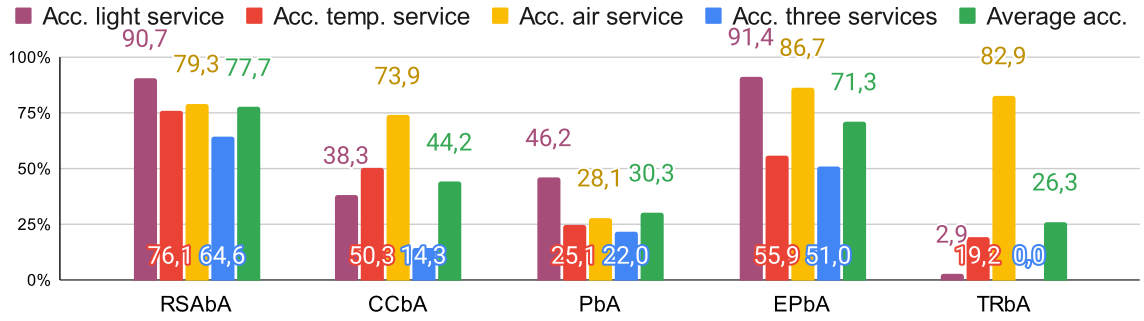

FIGURE 6.12 – Architecture evaluations with constraint and three services

use of the actuator with lower priority. From this figure, we can observe that, EPbA and RSAbA always have the best performance that is hardly affected. This shows that constraining the use of the actuator with lower priority and expressing it as the inhabitant profile work well for them. However, the performance of PbA decreased significantly compared with Fig.6.11, which is the same phenomenon as that in Fig.6.10. Therefore, constraining the use of the actuator with lower priority in the inhabitant profile is not a suitable method for PbA. Perhaps other methods should be used to describe this constraint. The performance of CCbA and TRbA has decreased, which may be because introducing more services along with the constraint makes the environments more complex. As a result, learning the features of the environment using the two architectures becomes more complicated.

The above four experiments show that OLSbA and QmixbA perform worse than most composite service-based architectures, which proves the advantage of composite service-based architectures. Moreover, the comparison of the composite service-based architectures shows that EPbA and RSAbA have better performance than the others, and thus are selected for our future research on the smart home system. Even though only experiments with two and three services are discussed in this document, these

experiments considered all possible problems that may occur when more than three services work together. Therefore, our results are most likely applicable for systems with more than three services, even though these systems may need more time to be well trained.

6.4 Architecture deployment in the real world

The process of deploying a SHOMA architecture-based system into the real world is similar to deploying the system to create a single dynamic smart home service presented at the end of Section 5.2. We can already deploy SHOMA architecture-based system in the real world. In this case, since the real world already provides the environment, developers only need to design different service modules to construct the SHOMA architecture.

However, at the beginning of the training process, proposing the actuators' states by the SHOMA architecture-based system can be regarded as a random process. As a result, the inhabitant may need to frequently express his/her dissatisfaction to the updated monitored states, which does not contribute to building a user-friendly smart home.

Therefore, we propose to pretrain the SHOMA architecture-based system in the simulated environment. When the system is well-trained, it continues its training process in the real world. This pretraining allows the SHOMA architecture-based services to know what actuators' states need to be proposed to achieve the approximate target monitored states. Therefore, the pretrained system will be less disruptive to the inhabitant and contribute to a better user experience when deployed in the real world. In addition, pretraining may increase the system's training rate when deployed in the real world. When pretraining the system in the simulated environment, developers need to design the service modules of various services and simulate the environment including input scenarios, transfer functions, and inhabitant profiles. Since we do not have access to a real smart home, we only focus on the logic of creating multiple dynamic smart home services in the simulation.

6.5 Conclusion

Having proposed an RL-based structure for the creation of one single dynamic service where each service is regarded as an agent containing an interpreter, an RL algorithm, and a policy, in this chapter, we reuse this structure to propose architectures for creating multiple dynamic smart home services. These architectures share a common name : SHOMA. The SHOMA architectures ensure no conflicts among the actuators' states proposed by related services.

The SHOMA architectures are divided into merged service-based and composite service-based architectures, depending on whether all services are merged as one single service. For each category, there are several subcategories. We conduct several simulated experiments, where each contains two or three smart home services, and the inhabitant's profile does or does not constrain the use of the actuator with lower priority, to prove the advantage of defining composite service-based architectures. We then select the architectures with better performance. Although we only conducted experiments for two and three services, these experiments considered all possible problems when there are more than three services. Therefore, the results of our experiments are most likely applicable to systems with more than three services.

In addition, the deployment of SHOMA architectures in the real world involves two phases, just like a single smart home service. In the first phase, the smart home services modeled by the SHOMA architecture are pretrained in advance in the simulated environment which contains three components. First component is the input scenarios that generate values for the physical environment states and the inhabitant state at each time step. Second is the transfer functions that compute the value for the monitored states when the values of the environment states and the states of the actuators are known at each time step. Third is the inhabitant profiles that describe the target values for the monitored states given the inhabitant state. In the second phase, the services continue their training in the real environment to adapt to the preferred monitored states of the real inhabitant. Since we do not have access to a real smart home, our studies only focus on the logic of creating smart home services in simulation.

7

Rule extraction from data-driven smart home services

Contents

7.1 Introduction

In the previous chapter, we proposed SHOMA architectures to create multiple smart home services that can dynamically propose actuators' states considering the inhabitant's (dis)satisfaction. However, the created services are based on data-driven RL algorithms that are black boxes. Thus, users have no idea in which situations the services have proposed certain actuators' states. Nevertheless, this knowledge can be essential for users concerning system maintenance and security issues.

To solve this problem, we propose to extract knowledge-based services from a learning method-based data-driven smart home services by extracting rules from these services. The extracted rules show the inhabitant in which situations the data-driven services have suggested certain actuators' states, and enrich the knowledge base so that it can cover more situations.

However, most existing rule extraction methods focus on learning systems with discrete inputs or learning systems designed for binary classification problems. However, in a smart home, there are both discrete (window state : open or closed) and continuous states (light intensity or temperature). Moreover, the control of smart home services can be regarded as not only a binary classification problem, but also a multi-class classification problem.

In this chapter, we propose a method called PBRE (Pedagogic Based Rule Extractor) that can extract rules from a trained learning system that accepts both discrete and continuous states as inputs. We conduct several experiments to evaluate PBRE and compare its performance with that of existing rule extraction methods, and the results show the good and better performance of PBRE. In addition, under the hypothesis that the input states of smart home services are all important, which means that the service considers all these states, we propose a PBRE variant to dynamically extract rules from untrained learning systems. Moreover, when there are multiple smart home services, we propose to combine all inputs from these services into a single input, and all outputs from these services into a single output. The single input and output are then used as input to the PBRE variant to realize dynamic rule extraction from multiple services.

In the rest of the chapter [1], we first explain the motivation for rule extraction. Then, we introduce the context of rule extraction from learning systems, including its defini-

1. Main part of this chapter was published in 2022 International Conference on Database and Expert Systems Applications (DEXA'22) and can be found in [103]

tion and classification. Next, we describe existing work on rule extraction. After that, we explain the principle of PBRE. We then present how the rule extraction method can be integrated into a smart home system. We also conduct several experiments to evaluate PBRE by first comparing it with an existing method and then using it to extract rules from multiple simulated single light intensity services with different inhabitant profiles. Moreover, we propose a PBRE variant to dynamically extract rules from an untrained smart home service. Further, we propose a strategy to dynamically extract rules from multiple smart home services using the PBRE variant. Finally, we summarize the main contributions of this chapter.

7.2 Motivation of rule extractions

Several existing work has mentioned various motivations for rule extractions. For example, [66] states that the main reason for extracting rules is to obtain a comprehensible description of the hypothesis underlying the model, which is also provided by [90, 12, 91]. In addition, knowledge verification, which can be acquired through rule extractions, is crucial in many domains. For example, in safety-critical applications [12], such as airlines, autonomous vehicles, and power plants, where users can validate the decisions made by the underlying models. In addition, knowledge verification is also essential for domains required by law, such as the U.S. federal Equal Credit Opportunity Act, which prohibits lenders from certain forms of discrimination by consequently providing specific reasons if an application is denied.

Another motivation mentioned by [66] is the capability of automatic knowledge acquisition. Such a capability makes it possible to first build the knowledge base with new knowledge discovered by learning method-based systems, and then debug a knowledge base that can be complex to maintain. In other words, rule extraction enables the integration of symbolic and connectionist approaches [12]. Finally, [90] and [66] have also justified that rule extraction contributes to the induction of scientific theories or the study of the generalization behavior of the underlying model.

In addition to comprehensibility, automatic knowledge acquisition, and contribution to induction and generalization research [66], [12] has introduced other new motivations. For example, there is an increasing need for software verification. Suppose artificial learning systems are integrated into these software systems. In this case, rule extraction techniques are needed for these learning systems to realize soft-

ware verification and debug these learning systems. [91] explains that rule extraction improves the performance of rule induction techniques by removing idiosyncrasies in the data.

In a smart home system, the introduction of rule extraction allows the inhabitant to understand in which situations certain actuators' states are dynamically proposed, which is especially important when the smart home system involves safety-critical decisions. In addition, since the data-driven services can dynamically propose actuators' states, the knowledge-based services can be automatically created by extracting rules from data-driven services, avoiding the manual creation of knowledge-based services. Finally, the extracted rules can be used to guide the learning direction of the data-driven services.

7.3 Context of rule extractions

Several definitions have been proposed to explain rule extractions. For example, according to [56, 39], rule extraction is the process of producing a description of the neural network hypothesis that is comprehensible and closely approximates the predictive behavior of the neural network, given a trained neural network and the data on which the neural network was trained. The definition in [126] is similar to that in [56, 39], but specifies the name of the algorithm. It states that in the context of high-dimensional feature spaces, giving a trained SVM (Support Vector Machine) and the data on which it was trained, rule extraction should provide an understandable description of the SVM hypothesis. This description can closely approximate the predictive behavior of the SVM.

According to [66], the above definitions of rule extractions did not insist that the returned description consists of rules, but of a description format that is comprehensible to humans. For example, "if-then" rules, "M-of-N" rules, and decision trees. Thus, there is an inherent duality between comprehensibility and accuracy. The extracted rules should make a trade-off between comprehensibility, so that humans can easily understand the descriptions, and accuracy, to approximate the model under study as closely as possible. The trade-off usually depends on the users' preferences and the problem's requirements for which the application was developed.

Several taxonomies have been defined to classify rule extraction algorithms. In this work, we mainly present the taxonomy proposed by [13], which is used by multiple

existing work [56, 66, 90]. [13] classifies rule extraction algorithms into decompositional, pedagogical, and eclectic methods. Decompositional and pedagogical methods are also called linked rule extraction techniques and black-box rule extraction techniques [118]. The decompositional approach focuses on extracting rules at the level of individual neurons (input, hidden, and output neurons) within the trained neural network [13]. The outputs of the input and hidden neurons must be a binary (yes/no) result corresponding to the rule consequent notion. This approach seems to be able to generate a complete set of rules for the trained neural network, but could be time-consuming and computationally expensive [120]. In pedagogical approaches, the trained neural network is viewed as a black box that maps inputs and corresponding outputs. Such a mapping is regarded as an instance rule describing only a particular situation. These instance rules are then refined to obtain more general rules. The pedagogical approaches can be faster than the decompositional ones, but they are less likely to cover all the rules contained in the trained neural network [120]. The eclectic approaches are the approaches that include decompositional and pedagogical approaches.

7.4 Existing work

There is a lot of work on extracting rules from trained neural networks or learning method-based systems. For example, [25] proposes a pedagogical algorithm called RxNCM. This algorithm first removes insignificant input attributes from a trained neural network, where the "insignificant attributes" means that the deletion of these input attributes has no significant influence on the precision of the neural network. Then, it determines the ranges for each attribute by selecting its minimum and maximum values from the training samples. The rules are created by combining attributes with ranges of values and the corresponding outputs. These rules are then pruned by removing conditions from a rule if the accuracy of the rules can be increased in the unseen dataset, dataset that are never used to extract rules. Finally, the pruned rules are updated by removing overlapping ranges of attribute values between rules if the accuracy of the new rules in the unseen dataset is increased.

In a tree-based machine learning approach [73], [29] collects all formulas from the root to a leaf node with a decision value and conjugates all these formulas to obtain a rule.

[110] proposes a decompositional algorithm. After pruning the neural network by reducing the number of hidden layers, the algorithm first discretizes the activation values of each hidden unit into multiple domains using the algorithm mentioned in [83]. Then, it extracts rules matching the activation domain with the corresponding output of each hidden unit. Next, it captures rules describing each activation domain regarding the inputs. Finally, it merges the two sets of rules to obtain the final rules.

[71] proposes a decompositional algorithm to extract rules from a three-layer based neural network : The algorithm first determines the neural network architecture, then removes redundant connections within the neural network, subsequently discretizes the output values of the hidden nodes to cluster the input attribute values, next generates rules by associating the determined input attribute value with the corresponding output values, and finally repeatedly prunes the rules if the accuracy of the extracted rules is not satisfactory.

[50] uses the decomposition-based KT algorithm [49] to generate rules by iteratively describing a given class as positive or negative. The weighted sum of the positive attribute and the rest of the negative attributes should be greater than the hidden layer threshold.

[123] proposes decomposition-based MOFN to extract rules. Unlike [50], MOFN introduces the concept of similar unit groups. To extract rules, first, a neural network is created using KBANN [122] and then trained. Next, units with similar weighted connections are grouped. The average values replace the weight values in each group, and the groups with low link weights are deleted. The updated neural network is subsequently trained again by optimizing the bias values. Then rules with weights and biases are extracted by combining inputs and outputs. The final rules are obtained by removing the weights and biases.

[118] proposes three rule extraction algorithms. The first is a type of black box rule extraction technique, which requires the inputs of neural networks to be binary or banalized. The second and the third are linked rule extraction techniques with the difference that the former limits the number of premises of the final extracted rules, while the latter has no such limitations.

However, except for RxNCM, the above work only focuses on neural networks with specific structures, e.g., [29, 110, 71] are only suitable for tree-based machine learning methods. Alternatively, it only accepts categorical and limited integer inputs, e.g., [123, 50] are only suitable for neural networks with limited integer inputs. Al-

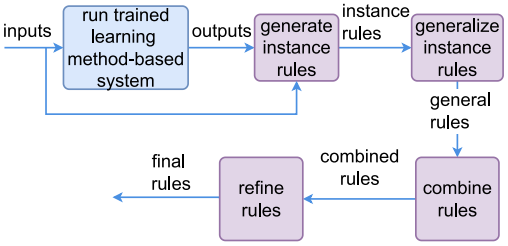

FIGURE 7.1 – PBRE rule extraction process

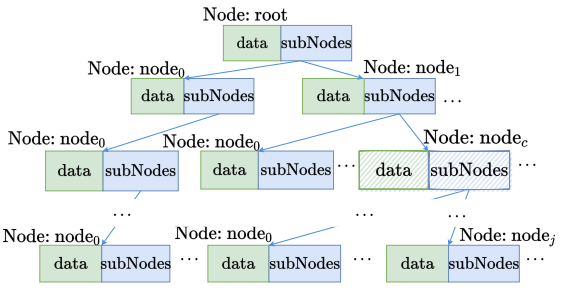

FIGURE 7.2 – Tree data structure

ternatively, it only concentrates on classification problems with categorical or limited integer outputs, e.g., the proposed algorithms in [118] only interest in classification problems.

In this chapter, we propose a method called PBRE (Pedagogic Based Rule Extractor) to extract rules from trained neural networks or other trained learning method-based systems by ignoring their input and output data types and their structures. We then compare PBRE with RxNCM to prove the better performance of PBRE.

7.5 The proposed PBRE method

The principle of PBRE is illustrated in Fig.7.1 : First, PBRE extracts an instance rule from a trained learning method-based system, where an instance rule is a mapping between the inputs and outputs of a learning method-based system at a given time step. Then, it generalizes the instance rule. Next, it combines the generalized rules by merging those whose conclusions are the same and the range values of the states in the conditions overlap. Finally, it refines the combined rules by removing insignificant states in the conditions based on the accuracy of the rules in the unseen dataset. The "insignificant states" means that the deletion of these input states has no significant influence on the precision of the learning system. The unseen dataset is a dataset that contains samples from which no rules are ever extracted, while the seen dataset has been used to extract rules. The use of the unseen dataset allows the evaluation of the generalization capability [71, 118] of the extracted rules.

7.5.1 Generate instance rules

A learning method-based system can directly output the desired action that an actuator will perform. In our study, a deep Q-Network (DQN) [60] is used. It can dyna-

mically propose actuators' states by considering the (dis)satisfaction of the inhabitant and the high dimensional environment states within a smart home system. Suppose that there is a trained learning method-based system with an unknown structure, its input $\{i_0, i_1, \cdots, i_n\}$ has value $\{s_{i_0,t}, \cdots, s_{i_n,t}\}$ where the first subscript indicates the type of the input state and the second subscript denotes the time step. The proposed states for related actuators are $\{s_{d_0,t}, \cdots, s_{d_m t}\}$ where the first subscript represents the type of the actuator and the second subscript denotes the time step. Therefore, we can obtain and represent the instance rule ir_t at time step t as

$$\begin{aligned} &\textit{if state } i_0 \textit{ has value } s_{i_0,t}, \cdots, \textit{ and state } i_n \textit{ has value } s_{i_n,t}, \\ &\textit{then actuator } d_0 \textit{ will have state } s_{d_0,t}, \cdots, \textit{and actuator } d_m \textit{ will have state } s_{d_m,t}. \end{aligned} \tag{7.1}$$

However, instance rules are not like rules for general situations. We should generalize them into rules by performing the following procedure.

7.5.2 Generalize instance rules

We generalize instance rules by first expressing instance rules in a tree structure with linked lists. This tree structure gives a concise and clear idea of an instance rule's compositions and generalization process. Then, we generalize an instance rule by merging it with another instance rule or rule whose conclusions are the same as those of the instance rule and whose certain conditions have close values or contain the values of those of the instance rule.

Fig.7.2 shows the structure of the linked list tree. Each branch is a linked list and stores an instance rule or a rule. Each branch node consists of a state value belonging to either conclusions or conditions. Except for the root node whose value is constant, the other nodes at the same level describe the values of the same type of states in different instance rules or rules. The data structure of each node belonging to the $Node$ class is shown in Fig.7.2. It consists of two parts : *data* being the state value of the current node and *subNodes* storing the child nodes of the current node. The state value of the node is represented as

$$node.data = \{Float\colon mean, Float\colon min, Float\colon max\} \tag{7.2}$$

where *mean, min,* and *max* are the average, minimum, and maximum of the state va-

lue of all combined samples. The introduction of a range value between the minimum and the maximum makes it possible to combine instance rules or an instance rule and a rule when their conclusions are the same, while the values of certain states in the conditions are the same or close or overlap. The data structure of $subNodes$ is shown in Eq.7.3. It is a vector consisting of all subnodes of the current node. Each element in this vector contains a subnode of class $Node$ and an integer $count$ indicating how many times the state value of the current subnode appears simultaneously with the state values of the nodes in the same branch and at higher levels.

$$node.subNodes = \{\{Node\colon node_0, Integer\colon count_0\}, \cdots\}. \tag{7.3}$$

To access a node and its subnodes, we use the dot notation like the property accessor in JavaScript, i.e., the node n_i in slashes in Fig.7.2 can be described as $root.node_1.node_c$, which belongs to the class $Node$, its state value is $n_i.data$, and the subnodes are $n_i.subNodes$. To access its average, minimum and maximum values, we use $n_i.data.mean, n_i.data.min$ and $n_i.data.max$. To query its j^{th} subnode, we use $n_i.subNodes[j]$.

However, after generalizing an instance rule, some particular state value ranges overlap between rules with the same conclusions. Therefore, we need to combine the obtained rules.

7.5.3 Combine rules

We use the algorithm COR (Combine Overlapping Rules) in Algo.9 to combine rules whose conclusions are the same while the states in the condition have overlapping values. To combine rules within a rule set GR_a where their conclusions are the same, and a particular state in the condition, say state c, has overlapping values, we first sort GR_a by the ascending order of minimum values with respect to state c (line 5). Next, we name the first rule of GR_a as gr_m (line 6). Then we iterate GR_a. For each rule gr_a of GR_a, its minimum value of state c is compared with the maximum value of the same state in gr_m. If the former is not larger than the latter, we change $gr_m.c.max$ to the larger value between $gr_a.c.max$ and $gr_m.c.max$ (lines 7∼9). If there is no such gr_a, we store gr_m in GR_b and define the current gr_a as gr_m for the next iteration(lines 10∼17). Once the iteration of GR_a is complete, we store gr_m in GR_b one more time to ensure that the last gr_m is concatenated with GR_b (lines 19∼23). Finally, we need

Algorithm 9: Combine Overlapping Rules (COR)

Result: combined rules
1 GR_a : rule set whose conclusions are the same while states in the condition may have overlapping values
2 $gr_a.c$: state value of the state c in the condition in a rule gr_a
3 GR_b : final returned rule set
4 **Function COR(GR_a) :**
5 sort GR_a in ascending order with respect to $gr_a.c.min$ for gr_a in GR_a
6 $gr_m = GR_a[0]$
7 **for** gr_a *in* GR_a **do**
8 **if** $gr_a.c.min \leq gr_m.c.max$ **then**
9 $gr_m.c.max = max(gr_a.c.max, gr_m.c.max)$
10 **else**
11 **if** GR_b *equals None* **then**
12 $GR_b = gr_m$
13 **else**
14 GR_b =concatenate GR_b and gr_m
15 **end**
16 $gr_m = gr_a$
17 **end**
18 **end**
19 **if** GR_b *equals None* **then**
20 $GR_b = gr_m$
21 **else**
22 GR_b=concatenate GR_b and gr_m
23 **end**
24 Delete duplicate rule from GR_b
25 return GR_b

to delete duplicate rules from GR_b (line 24).

7.5.4 Refine rules

In this section, we focus on removing insignificant states in the conditions using Algo.10. First, we use Pearson product-moment correlation coefficients (PPMCC) [20] to calculates the states-targets correlation vector. This vector stores the correlation between available state types and the targets. Then, we sort this correlation vector in ascending order to ensure the least correlated state types come first (lines 8∼10). For each $stateType$ in available state types $stateTypes$, and for each unseen sample d_{unseen} in unseen dataset D_{unseen}, first, we define a rule set GR_1 acquired from GR by removing states with types in $insignificantStates$ and $stateType$. Then, we create a rule set GR_2 containing rules from GR_1, whose states' range values in the conditions contain the states' values of the unseen sample under study d_{unseen}. We also define a rule set GR_0 equal to GR_2 with the deleted states being added (line 11∼16). If the size of GR_2 is 1, we select the conclusions of this rule as the targets of

Algorithm 10: Remove Insignificant States (RIS)

```
   Result: final rules
1  D : dataset storing samples with each containing state values and target values
2  D_unseen : unseen dataset with each containing state values and target values
3  GR = {gr_1, ..., gr_k} : extracted rules
4  maxAcc : the maximum accuracy
5  stateTypes : the available state types
6  insignificantStates : vector storing insignificant state types
7  Function RIS(D, D_unseen, GR) :
8      corrState = D.corr().iloc[:-1, -1]
9      sort corrState in ascending order
10     sort stateTypes with the same order of corrState
11     for stateType in stateTypes do
12         numCorrect = 0
13         for d_unseen in D_unseen do
14             create GR_1 ∈ GR without states with types in insignificantStates and stateType
15             select GR_2 ∈ GR_1, whose rules' states ranges contain those of d_unseen
16             create GR_0 by adding deleted states to GR_2
17             if size(GR_2) equals 1 then inference ← conclusions of GR_2
18             else
19                 arr=concatenate d_unseen.states and GR_0.states.mean
20                 arr, corrState2=remove states with types in insignificantStates and
                     stateType from arr, corrState
21                 arr = arr * corrState2
22                 inference = Inference(arr)
23             end
24             if inference equals d_unseen.targets then numCorrect+ = 1
25         end
26         acc = numCorrect/length(D_unseen)
27         if acc ≥ maxAcc then add stateType to insignificantStates ; maxAcc = acc
28     end
29     for stateType in insignificantStates do
30         for d_unseen in D_unseen do
31             arr=execute lines 14∼19
32             arr, corrState2=arr, corrState remove states with types in insignificantStates
                 and add state with type stateType
33             execute lines 22∼26
34         end
35         if acc ≥ maxAcc then remove stateType from insignificantStates ; maxAcc = acc
36     end
37     remove states with types in insignificantStates from R ; return GR
```

d_{unseen} (line 17). Otherwise, we remove states with types in $insignificantStates$ and $stateType$ from the concatenated states matrix which combines the states of d_{unseen} and the averages of the states in the conditions of GR_0. The states with types in $insignificantStates$ and $stateType$ are also removed from the states-targets correlation vector (lines 18∼20). Then, we calculate the weighted matrix by multiplying the states-targets correlation vector and the concatenated matrix (line 21). The resulted weighted matrix allows to put more importance on the states that are more correlated

Algorithm 11: Infer the unseen sample (Inference)

Result: Inference result
1 arr :matrix concatenating states of the unseen sample and averages of the states in the conditions of the rules
2 ϵ : predefined small number
3 **Function** `Inference`(arr) **:**
4 $corr = correlation(arr)$
5 **if** $\forall i, j \in corr, (i - j) < \epsilon$ **then** select $conclusions$ of the rule which has the maximum sum of frequencies of occurrence of the conclusions
6 **else**
7 select $conclusions$ of the rule with the maximum correlation
8 **end**
9 return $conclusions$

(or significant) with the targets. Next, we use Algo.11 to derive the unseen sample d_{unseen} with the inputs being the concatenated matrix (line 22).

As shown in Algo.11, to obtain inference, we use PPMCC to calculate the correlation between d_{unseen} and rules in R_2 based on the input concatenated state matrix (line 4). If all rules have close correlations with d_{unseen}, we choose the conclusions of the rule which has the maximum sum of frequencies of occurrence ($count$ in Eq.7.3) of the conclusions ; otherwise, we select the conclusions of the rule whose states are most strongly correlated with those of d_{unseen} (lines 5∼7).

After all unseen samples are derived, we compute the accuracy of GR without state with types in $insignificantStates$ and $stateType$ (line 26). If the accuracy is not less than the maximum accuracy, the state with type $stateType$ is not important for the rules to correctly make inference. It will be added to the $insignificantStates$ vector, and the current accuracy will be the maximum accuracy (line 27). Next, after having run through all $stateTypes$ and obtained the final $insignificantStates$, we decide which $stateType$ in $insignificantStates$ can be re-added to rules to maintain or improve the accuracy on the unseen dataset. If such a $stateType$ exist, it will be removed from $insignificantStates$ vector, and the updated accuracy will be the new maximum accuracy as shown in lines 29∼35. When the updated final $insignificantStates$ is acquired, we remove states belonging to types in $insignificantStates$ from GR and return the updated GR as the final extracted rules (line 37). An extracted rule after having been converted to "if-then" rule can be written as :

if state i_0 *is between* $s_{i_0,min}$ *and* $s_{i_0,max}$ *and has average* $s_{i_0,m}, \cdots,$ *then actuator* d_0 *will have state* s_{d_0} *with frequency of occurrence* $count_{d_0}, \cdots$.

(7.4)

Reasoning process

After having obtained the extracted rules without insignificant states in the conditions, we can use these rules to make inference for any dataset including seen and unseen datasets. The process is shown in Algo.12.

Suppose that we have the extracted rules GR; a vector of insignificant states $insignificantStates$ acquired from Algo.10; and a dataset D_2 storing samples with each only containing the states.

For each sample d in D_2 (line 5), we remove the insignificant states from d (line 6). Then, we select the extracted rules GR_2 whose range values of the states in the conditions contain the values of the states of the current sample d (line 7). If there is only one extracted rule being activated (line 8), we select the conclusion of the activated rule as the proposition for the current sample d (line 9). Otherwise, if there are several extracted rules being activated (line 10), we perform the following process : first, we concatenate the states of d and the average values of the states in the conditions of the activated extracted rules GR_2 (line 11). Next, we calculate the PPMCC correlation between d and GR_2 based on the similarity between the states of d and averages of the states in the conditions of GR_2 (line 12). If all the extracted rules have close correlations with d (line 13), we select the conclusions of the

Algorithm 12: Infer samples (Reasoner)

```
   Result: Inference result
1  GR : extracted rules
2  insignificantStates : insignificant states
3  D_2 : dataset to be inferred by the extracted rules
4  Function Reasoner(GR, insignificantStates, D_2) :
5      for d in D_2 do
6          d ← remove states of insignificantStates from d
7          GR_2 ← select the extracted rules from GR whose ranges of states in the conditions
             contain the values of the states of d
8          if size(GR_2) equals 1 then
9          select conclusions of GR_2
10         else
11             arr=concatenate d.states and GR_2.states.mean
12             corr = correlation(arr)
13             if ∀i, j ∈ corr, (i − j) < ϵ then select conclusions of the rule which has the
                 maximum sum of frequencies of occurrence of the conclusions
14             else
15                 select conclusion of the rule with the maximum correlation for d
16             end
17         end
18     end
19     return conclusions for D_2
```

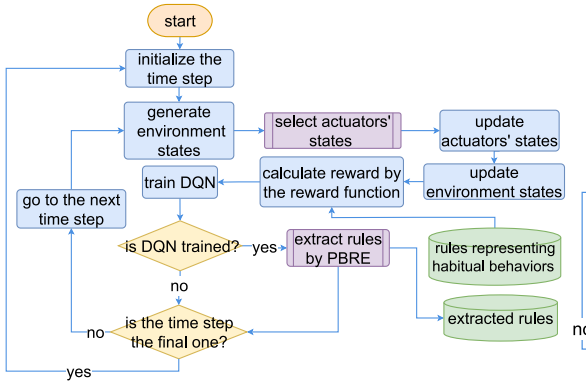

FIGURE 7.3 – Rule extraction in a smart home system with one service in simulation

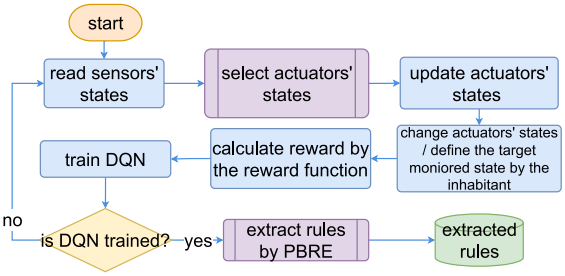

FIGURE 7.4 – Rule extraction in a smart home system with one service in the real world

rules with the maximum sum of the frequencies of occurrences in terms of the states in the conclusions (line 13). Otherwise, we select the conclusions of the rule which is the most correlated with d (line 15). At the end of the algorithm, we return the conclusions for all samples within D_2 (line 19)

7.6 Integration of rule extraction in smart home service creation

Applying the learning method-based system and rule extraction methods in a smart home helps to build a smart home system that automatically creates explicable services. In this section, we present the working process of this system in simulation and the real world in subsections 7.6.2 and 7.6.1. The two processes correspond to the fact that the deployment of a smart home system in the real world involves two phases : The first phase is the pretraining of the system in simulation, and the second phase is the continuous training of the smart home system in the real world.

7.6.1 Integration of rule extraction in simulation

Fig.7.3 shows how the smart home system looks like in simulation : First, the time step t is initialized. Next, the predefined functions, including the input scenario, the transfer function and the inhabitant profile, generate the environment states, such as the inhabitant state and the indoor and outdoor light intensities at time step t. Then, the service simulated by the learning method selects the states that the actuators should take at time step t, and the actuators update their states to the selected ones. The monitored state is subsequently updated regarding the actuators' executions. De-

pending on the predefined reward function and the simulated inhabitant profile, a reward is calculated to indicate whether the inhabitant is satisfied with the updated monitored state. After that, the system uses a specific optimization algorithm to train the learning method-based system based on the transitions. Once the learning method-based system is well-trained with high and stable accuracy, the rules are extracted using PBRE and stored in a database. The system then determines whether the current time step is the last. However, if the learning method-based system is not well-trained, the system directly checks if the current time step is the last. If it is not the last time step, the system proceeds to the next time step ; otherwise, it returns to the first time step and repeats the entire process. The DQN mentions in the following two subsections correspond to the RL algorithm within a service as shown in Chapter 5. As a result, the states in the conditions of a rule or an instance rule are from the input states to the RL algorithm within a service model, where the input states are extracted by the interpreter within a service model.

7.6.2 Integration of rule extraction in the real world

Fig.7.4 shows how rule extractions are integrated into the learning method-based smart home system in the real world : When the system starts, it reads the sensor variables capturing the values of the observable environment states associated with the service. Then the service selects the involved actuators' states. The actuators update their states to the selected ones. The executions of the actuators lead to changes in the monitored state, e.g., indoor light intensity. Optionally, if the inhabitant is satisfied with the updated monitored state, he/she takes no action ; otherwise, he/she can change some or all associated actuators' states to match his/her habitual behavior. Alternatively, the inhabitant can define the target value for the monitored state (these target values can also be predefined). Taking into account the changes the inhabitant makes to the actuators' states, the system can obtain the reward calculated by the predefined reward function. Or, the system can calculate the reward by comparing the updated monitored state with its corresponding target value if the inhabitant specifies the target value for the monitored state. It then trains the learning method-based system using the transitions as input. Each transition contains the values of the environment states sensed by the sensors, the states of the actuators proposed by the service, the reward, and the updated environment states. If the learning method-based system is well-trained with high and stable accuracy, it uses PBRE to extract

rules and store them in a database ; otherwise, it repeats the process described above.

7.7 Evaluation experiment

To evaluate the performance of PBRE, we first compare it with an existing method based on benchmark datasets and then evaluate it based on three simulated smart home services.

7.7.1 Metrics

We use the following metrics to do the evaluation : (1) The number of extracted rules. [40, 71] (2) Accuracy describes the number of samples where the updated monitored states conform to the inhabitant's preferences as a percentage of the total number of samples. [16, 123, 135] (3) Similarity, or fidelity [16, 123, 135], is the number of samples where conclusions derived from the rules are the same with propositions proposed by the learning method-based system as a percentage of the total number of samples. (4) Inference is the number of samples that the extracted rules can derive as a percentage of the total number of samples.

Metrics 7.7.2∼(4) are evaluated for both seen and unseen samples. The procedure for determining the metrics is shown in Fig.7.5. First, we simulate input sample 1 and input sample 2. The learning method-based system makes predictions and stores them in two databases for the two samples. Next, input sample 1 is used with the learning method-based system to extract rules and obtain metric (1). These rules are

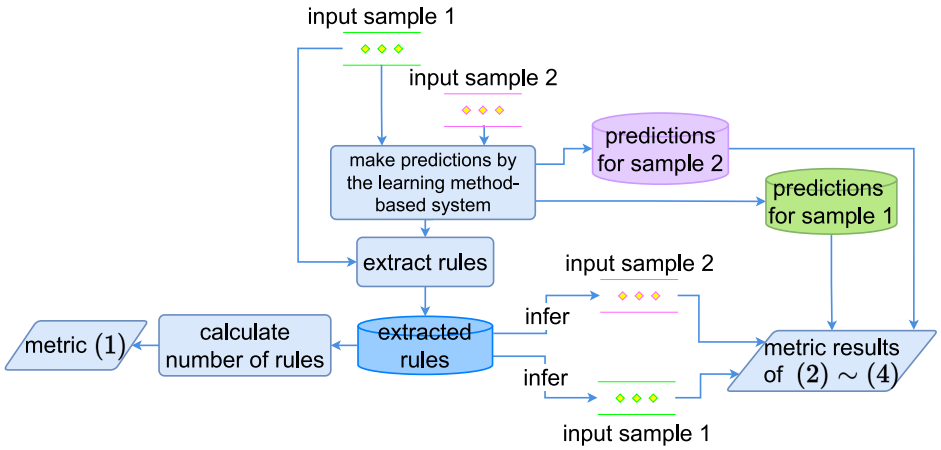

FIGURE 7.5 – Metrics acquiring procedure

TABLE 7.1 – Datasets used for evaluating the performance of the extracted rules

Dataset	Num. of samples	Num. of attributes	Attribute characteristics	Num. of class
Iris dataset	150	4	Real	3
Wisconsin breast cancer	569	30	Real	2
Sonar dataset	208	60	Real	2
German credit dataset	1000	9	Categorical, Integer	2
Ionosphere dataset	350	34	Integer, Real	2
Heart disease dataset	303	13	Categorical, Integer, Real	5

used to derive the two samples. Finally, the derived conclusions are compared with the predictions to evaluate metrics 7.7.2∼(4).

7.7.2 Evaluation and Comparison with Existing Work

This section presents the first evaluation experiment comparing PBRE with an existing method called RxNCM. The comparison is based on six datasets from the University of California Irvine machine learning repository : the Iris dataset, the Wisconsin Breast Cancer (WBC) dataset, the Sonar dataset, the German Credit dataset, the Ionosphere dataset, and the Heart Disease dataset. The iris dataset classifies flowers, the Wisconsin breast cancer dataset diagnoses whether the cancer is benign or malignant, the sonar dataset uses sonar to identify whether an object is a metal or a stone, the German credit dataset identifies each person who takes out a loan from a bank as a good or bad borrower, the Ionosphere Dataset analyses whether a radar is good or bad, and the heart disease dataset diagnoses heart disease. The detailed descriptions of the datasets can be found in Table 7.1.

Experiment results

Table 7.2, Fig.7.6 and Fig.7.7 show the metric results for PBRE and RxNCM. Table 7.2 shows us the result for metric 7.7.2 defined in Section 7.7.1 of the current chapter. We see that the number of rules extracted from each dataset by PBRE or RxNCM is not large, which ensures that storing the extracted rules does not require large memory, which is an important metric for a high-dimensional smart home system.

Fig.7.6 and Fig.7.7 show us the results of metrics 7.7.2 to defined in Section 7.7.1 of the current chapter, where the blue legend "NN Acc.", the dark corn flower blue legend "PBRE Acc." and the light corn flower blue legend "RxNCM Acc." respectively

TABLE 7.2 – Number of rules extracted with PBRE and RxNCM

	Iris	WBC	Sonar	German credit	Ionosphere	Heart disease
PBRE num. of rules	3	2	2	2	2	5
RxNCM num. of rules	3	2	2	2	2	5

describes the accuracy of the learning method-based system, the extracted rules by PBRE, and the extracted rules by RxNCM ; the dark orange legend "PBRE Sim." and light orange legend "RxNCM Sim." respectively represents the similarity of the PBRE and the RxNCM ; the light magenta legend "PBRE Infe." and the light purple legend "RxNCM Infe." respectively denotes the inference of the PBRE and the RxNCM ; the light green legend "PBRE Ave." and the dark green legend "RxNCM Ave." respectively illustrates the average of the three previous metric results (accuracy, similarity, and inference) and represents the general performance of the corresponding algorithm.

From the two figures, we can observe that although RxNCM, like PBRE, can infer all seen and almost all unseen samples (see "PBRE Infe." and "RxNCM Infe."), the rules extracted with PBRE generally have higher accuracy and similarity than those extracted with RxNCM (see "PBRE Acc.", "RxNCM Acc.", "PBRE Sim." and "RxNCM Sim."). To illustrate the general performance, we calculate the average of metrics 7.7.2 to (4) and denote it as "PBRE Ave." and "RxNCM Ave.". The results show that PBRE has higher general performance than RxNCM for both datasets, which is consistent with the observations made above for metrics 7.7.2 to (4). Moreover, including general performance, RxNCM has higher metric results in the unseen datasets than in the seen datasets, and PBRE does the opposite and has higher metric results than RxNCM in both datasets. This is because to refine rules, PBRE not only deletes the states in the conditions but also adds them back, which ensures the number of states in the conditions and, thus, guarantees that PBRE achieves good performance in the unseen datasets and maintains the performance in the seen datasets.

7.7.3 Experiment in the smart home context

In this section, we present the second evaluation experiment where we use PBRE to extract rules from three individual light intensity services (DQN_v1, DQN_v2, and DQN_v3). Each service is simulated based on a DQN, following the principle shown in Fig.5.7. The detailed working process of the corresponding smart home system based on a single light intensity service can be seen in Fig.7.3. To perform the simulated

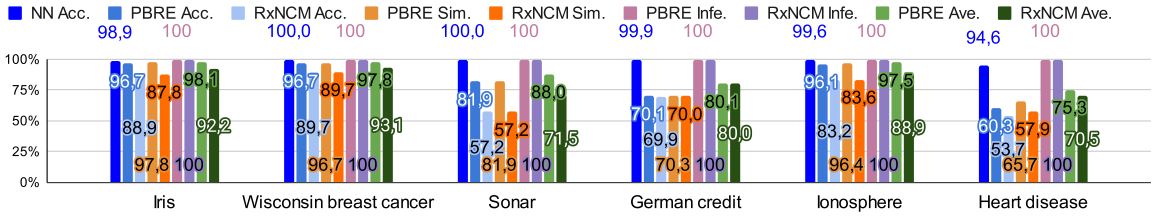

FIGURE 7.6 – PBRE and RxNCM experiment results by working on seen datasets

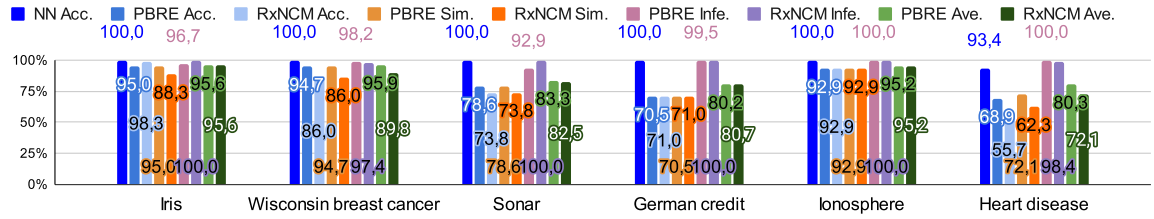

FIGURE 7.7 – PBRE and RxNCM experiment results by working on unseen datasets

TABLE 7.3 – Number of rules extracted by PBRE from different DQNs

	DQN_v1	DQN_v2	DQN_v3
Num. of rules	4	4	19

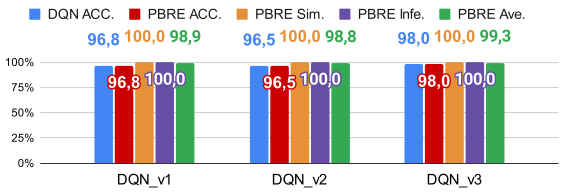

FIGURE 7.8 – PBRE with the seen datasets

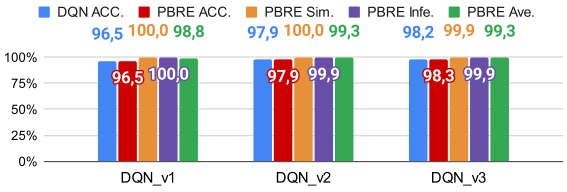

FIGURE 7.9 – PBRE with the unseen datasets

experiment for each light intensity service, we use the simulated environment shown in Section 5.5.1 in Chapter 5. In addition, the inhabitant profile for each service can be found in Table 7.4.

Experiment Results

The metric results of PBRE in extracting rules from different DQN-based services for the seen and unseen datasets are shown in Table 7.3, Fig.7.8 and Fig.7.9. We can see that the number of extracted rules in Table 7.3 for each simulated service is not large, which ensures that storing these rules does not require large memory. Moreover, we can see that the number of rules in DQN_v3 has a larger value because the corresponding inhabitant profile is more complex, as shown in Table 7.4. Furthermore, from Fig.7.8 and Fig.7.9, we see that PBRE can extract rules from DQN-based

services with satisfactory general performance (see "PBRE Ave.'), which can be further explained as follows : The extracted rules achieve the same and sometimes even higher accuracy than the DQN-based services ; they have the same similarity to the DQN-based services in the seen datasets and almost the same similarity in the unseen datasets ; moreover, they can infer all seen and almost all unseen samples. One of the extracted rules after having hidden averages and frequencies of occurrence from Eq.7.4 in DQN_v3 is :

if the inhabitant is working, and the outdoor light intensity is between 0.35 and 243.28, then the lamp is at level 3, and the curtain is closed. (7.5)

More rules can be found in Table 7.4 where we approximate the lowest and highest values of each state's range to integers. This table shows that DQN_v1 and DQN_v2 have the same extracted rules suggesting always closing the curtain. However, the rules in DQN_v3 have a curtain setting more varying as constraining the use of the actuator with lower priority is required in the inhabitant profile. Moreover, when expressing the rules, we only show the states' ranges instead of the states' averages to make it easier to compare with the inhabitant profile.

7.8 PBRE variant for dynamic rule extraction from an untrained service

However, during the dynamic proposition of the actuators' states by a service, it is impractical to stop the working process of the service to select important input states for the rules to be extracted. Therefore, we assume that the input states are already refined and need not be removed. Under this hypothesis, we use PBRE without the process of "refining rules" to dynamically extract rules when the received reward is positive. This rule extraction can take place even during the training process of PBRE. The dynamically extracted rules allow the inhabitant to know the situations under which certain actuators' states are proposed. Furthermore, because the rules are extracted only when the reward for the service is positive, the knowledge-based services corresponding to these extracted rules will have higher accuracy than the related learning method-based systems.

In this case, we define a customized inference process for the PBRE variant. Com-

TABLE 7.4 – Extracted Rules for the light service simulated by different DQNs

DQN	Habitual behaviors	Extracted rules
v1	(1) If the inhabitant is absent, then the indoor light intensity is 0 lux; (2) If the inhabitant is working, then the indoor light intensity is between 250 lux and 350 lux; (3) If the inhabitant is seeing a movie, then the indoor light intensity is between 350 lux and 450 lux; (4) If the inhabitant is sleeping, then the indoor light intensity is 0 lux.	(1) If the inhabitant is absent, then the lamp is off, and the curtain is closed; (2) If the inhabitant is working, then the lamp is at level 3, and the curtain is closed; (3) If the inhabitant is seeing a movie, then the lamp is at level 4, and the curtain is closed; (4) If the inhabitant is sleeping, then the lamp is off, and the curtain is closed.
v2	(1) If the inhabitant is absent, then the indoor light intensity is 0 lux; (2) If the inhabitant is working, then the indoor light intensity is between 250 lux and 350 lux; (3) If the inhabitant is seeing a movie, then the indoor light intensity is between 350 lux and 450 lux; (4) If the inhabitant is sleeping, then the indoor light intensity is 0 lux.	(1) If the inhabitant is absent, and the outdoor light intensity is between 0 and 605 lux, then the the lamp is off, and the curtain is closed; (2) If the inhabitant is working, and the outdoor light intensity is between 0 and 605 lux, then the lamp is at level 3, and the curtain is closed; (3) If the inhabitant is seeing a movie, and the outdoor light intensity is between 0 and 605 lux, then the lamp is at level 4, and the curtain is closed; (4) If the inhabitant is sleeping, and the outdoor light intensity is between 0 and 605 lux, then the lamp is off, and the curtain is closed.
v3	With the preference of decreasing the use of the related energy consuming actuator : (1) If the inhabitant is absent, then the indoor light intensity is 0 lux; (2) If the inhabitant is working, then the indoor light intensity is between 250 lux and 350 lux; (3) If the inhabitant is seeing a movie, then the indoor light intensity is between 350 lux and 450 lux; (4) If the inhabitant is sleeping, then the indoor light intensity is 0 lux.	(1) If the inhabitant is absent, and the outdoor light intensity is between 0 and 605 lux, then the lamp is off, and the curtain is closed; (2) If the inhabitant is working, and the outdoor light intensity is between 246 and 357 lux, then the lamp is off, and the curtain is fully open; (3) If the inhabitant is working, and the outdoor light intensity is between 512 and 605 lux, then the lamp is off, and the curtain is half-open; (4) If the inhabitant is seeing a movie, and the outdoor light intensity is between 356 and 452 lux, then the lamp off, and the curtain is fully open; (5) If the inhabitant is sleeping, and the outdoor light intensity is between 0 and 605 lux, then the lamp is off, and the curtain is closed; $\cdots$

pared to the one in Algo.12 for PBRE, the difference is that for each sample to be inferred, we do not need to remove the insignificant states as shown in line 6 of Algo.12. The other operations are the same. The details of the inference process of the PBRE variant is shown in Algo.13.

In addition, Fig.7.3 and Fig.7.4 are changed to Fig.7.10 and Fig.7.11, respectively. In the situation of deploying the smart home system in simulation and the real world, the condition that determines whether to "extract rules by PBRE" is changed from "is DQN trained" to "is the reward positive".

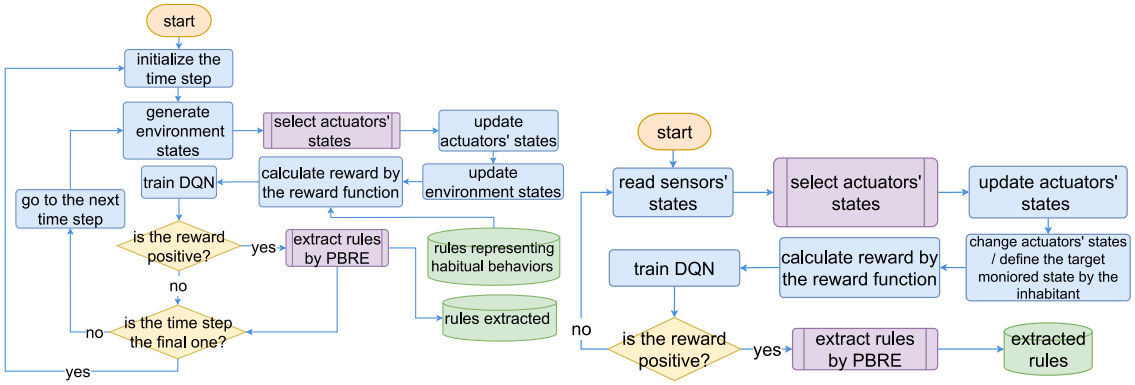

FIGURE 7.10 – Rule extraction without "refining rules" in a smart home system with one service in simulation

FIGURE 7.11 – Rule extraction without "refining rules" in a smart home system with one service in the real world

7.9 Dynamic rule extraction from multiple smart home services

Since a smart home usually contains multiple services, and we cannot stop the training process of the learning method-based smart home services to do the rule extraction, we use the PBRE variant to realize the dynamic rule extraction from multiple services.

To do this, the PBRE variant will regard all services as one single service. Therefore, at a given time step, the inputs and outputs of all services will be concatenated as

Algorithm 13: Infer samples (Reasoner)

Result: Inference result
GR : available extracted rules
D_2 : dataset to be inferred by the extracted rules
Function `Inference`(GR, D_2) :
    for d *in* D_2 **do**
        $GR_2 \leftarrow$ select the extracted rules from GR whose ranges of states in the conditions contain the values of the states of d
        if $size(GR_Z)$ $equals$ 1 **then**
            select $conclusions$ of GR_2
        else
            arr=concatenate $d.states$ and $GR_2.states.mean$
            $corr = correlation(arr)$
            if $\forall i, j \in corr, (i - j) < \epsilon$ **then**
                select $conclusions$ of the rule which has the maximum sum of frequencies of occurrence of the conclusions
            else
                select $conclusions$ of the rule with the maximum correlation
            end
        end
    end
    return $conclusions$ for D_2

one single input and one single output. The single input and output are then used as input of the PBRE variant to obtain the final extracted rule. During the rule extraction process, the rules can be extracted only when the rewards for all services are positive. The inference process is the same with the one shown in Algo.13

The reason we consider the input and output of all services as a single input and output in rule extraction is to ensure that the propositions of the actuators' states by the extracted knowledge-based services are the same with the propositions by the corresponding data-driven services when the environment is in certain states.

7.10 Conclusion

We have proposed data-driven architectures to create dynamic smart home services. However, the created data-driven services are like black boxes, and the inhabitant has no idea in which situations certain actuators' states are proposed.

To solve this problem, we propose to extract knowledge-based services from data-driven services by extracting rules from these data-driven services. Different from existing work [64] aiming at making learning method-based systems understandable by the inhabitant (e.g., using natural language easily understood by the inhabitant), we solely concentrate on extracting rules to showing the behavior of the corresponding learning method-based system.

To extract rules, we propose a new rule extraction method called PBRE that can extract rules from a data-driven service without considering the structures of the associated learning method and the data types of the input states, and can be used for problems that can be regarded as multi-class classification problems. The working principle of PBRE includes four parts : generate instance rules, generalize instance rules, combine rules, and refine rules.

The deployment of the smart home service in the real environment involves two phases. The first phase is the pretraining of the smart home system in simulation, and the second phase is the continuous training of the smart home service in the real world. Therefore, the integration of PBRE into the smart home system includes two phases : The integration of PBRE into the smart home system in simulation and in the real world. Several experiments are conducted to evaluate the performance of PBRE in extracting rules. The first is the extraction of rules from learning systems used for six benchmark datasets, and the second is the extraction of rules from three single

smart home light intensity services with each being simulated by a DQN. The results show the satisfactory performance of PBRE.

Moreover, to refine rules in PBRE, states are removed from conditions and added back if the accuracy of the extracted rules on the unseen dataset does not decrease. However, it is impractical to stop the working process of a service and then select important states in the condition for the extracted rules. Therefore, assuming that the input states for a data-driven smart home service are already refined, we propose a PBRE variant to realize the dynamic rule extraction. In this variant, the process of"refining rules" is not necessary, and the condition for extracting rules depends on whether the reward of the smart home service is positive. In this way, the inhabitant can dynamically know the situations in which certain actuators' states are proposed. In addition, the knowledge-based service based on these extracted rules will have higher accuracy than the corresponding data-driven service. This is because these rules will only be extracted when the reward of the data-driven service is positive.

Nevertheless, in a smart home, there are usually multiple smart home services. Therefore, we propose a technique based on the PBRE variant to dynamically extract rules from multiple smart home services : at a given time step, all inputs and outputs of all services are viewed as one single input and output. The single input and output are then used as input of the PBRE variant. The rules can be extracted only when the rewards of all services are positive.

Considering the fact that a smart home usually contains multiple services and that it is impractical to stop the learning method-based services to do the rule extraction, in the following chapters, the technique of using the PBRE variant to do the dynamic rule extraction from multiple learning method-based services will be used, which contributes the acquisition of explicable and dynamic smart home services.

8

Hybrid system for smart home services creation

Contents

8.1 Introduction

In this chapter, we propose to combine knowledge-based and data-driven approaches in order to deliver hybrid solution for service creation. We will start from a set of manually defined knowledge-based services (a set of rules and associated reasoner). Considered individually these services are simple enough to be defined in this setting. We will be using these services to accelerate the learning phase that we will be using to combine these services. We will be using some of the SHOMA architectures defined in Chapter 6. During this learning phase, we will also put in practice the rules extraction mechanism for PBRE variant defined in Chapter 7.

In our hybrid architecture, we will have three ways to decide what actions will be taken at each time step, namely (sorted by higher priority first) :

— the reasoner for manually defined services

— the reasoner for extracted rules

— the SHOMA architecture-based system

The actions that will be decided by considering the highest priority first, then turning to the lowest priority if the higher priorities are not satisfactory.

In this chapter, we first present the structure of our hybrid system that we call HKD-SHO (Hybrid Knowledge-based and Data-driven services-based Smart HOme system). Then, we explain the working process of the different components of this hybrid system. Next, we describe the working process of the system as a whole. Afterward, we run several simulated comparative experiments to evaluate this hybrid system, and prove its performance is better than the system with only the learning method and the system with the learning method and the preexisting rules. Finally, we summarize the chapter.

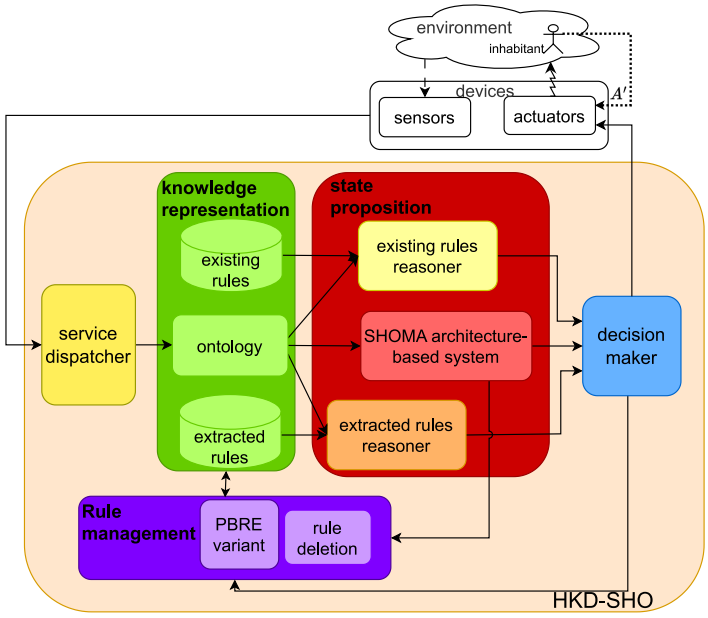

FIGURE 8.1 – Structure of HKD-SHO

8.2 Proposed HKD-SHO system

Fig.8.1 shows the structure of HKD-SHO, which consists of five modules : the "service dispatcher", the "knowledge representation", the "state proposition", the "decision maker", and the "rule management". In addition, the "existing rules", the "ontology" and the "extracted rules" modules are contained in the "knowledge representation" module. The "existing rules reasoner", the "SHOMA architecture-based system", and the "extracted rules reasoner" are included in the "state proposition" module. The "PBRE variant" and the "rule deletion" are contained in the "rule management" module. Before presenting the working process of HKD-SHO, we define the following hypotheses that HKD-SHO should fulfill. First. the existing rules and ontology are predefined either by the inhabitant or by the developers. Second, the SHOMA architecture-based system is already predefined based on the selected SHOMA architectures in Chapter 6 : EPbA and RSAbA.

The general working process of HKD-SHO is as follows : First, the values of the actuators and the sensors are sent to the "service dispatcher" which allows the selection of the associated sensors and actuators for each service. The "service dispatcher" then sends these dispatched states to the "ontology" in the "knowledge representation" module to update the state values of the corresponding instances (including instances of actuators and sensors, which associate with some instances of services). Sometimes, the "service dispatcher" tells the ontology whether to create new instances for the actuators, the sensors, and even the services. The updated "ontology" comprising the latest observable states (including the states of the actuators and the sensors) are sent to the "SHOMA architecture-based system" within the "state proposition" module.

Before proposing new states for actuators, the "SHOMA architecture-based system" calculates the rewards for the iteration of the previous time step. In addition, if the replay memory contains at least a minibatch number of transitions, the "SHOMA architecture-based system" also involves the training process to update its parameters. These rewards are passed to the "PBRE variant" and "rule deletion" in the "rule management" module to decide whether to extract rules or delete the extracted rules based on the activities of the previous time step. In addition, the input states to the RL algorithms within the "SHOMA architecture-based system" of the previous time step, and the information coming from the "decision maker" module specifying the final actuators' states are made by which services (existing rules-based services, data-

driven services and extracted rules-based services) of the previous time step, are also sent to the rule management module for deleting or extracting rules in the next time step.

The updated "extracted rules" along with the state values in the "ontology" are sent to the "extracted rules reasoner" in the "state proposition" module. In addition, the "existing rules", which are predefined and can only be deleted by the users, are sent along with the state values in the "ontology" to the "existing rules reasoner" in the "state proposition" module. The "extracted rules reasoner", the "SHOMA architecture-based system", and the "existing rules reasoner" are three structures used in the "state proposition" module to create services that propose new states for the related actuators.

The "existing rules reasoner" used in our work is Pellet [111]. It infers new states for actuators based on the input states and the existing rules. The existing rules are expressed in SWRL (Semantic Web Rule Language) [63] and stored in the "existing rules" database. The extracted rules are expressed in "if-then" rules, where each state in the condition contains its range value and the average of all training samples concerning that state, as shown in Eq. 7.4 in Chapter 7. These rules are stored in the "extracted rules" database. The "extracted rules reasoner" is customized. It considers not only the ranges of the states in the conditions, but also the PPMCC similarity. The similarity is between the average values of the states in the conditions and the corresponding states of the sample under consideration. The "SHOMA architecture-based system" contains a selected SHOMA architecture to propose states for the corresponding actuators.

The propositions of the states of the actuators coming from the "extracted rules reasoner", the "existing rules reasoner", and the "SHOMA architecture-based system" are sent to the "decision maker" to determine the final states of the actuators. In addition, the "decision maker" also sends information to the "PBRE variant" and the "rule deletion" in the "rule management". The information contains the determined final states of the actuators and the fact that the propositions of which services were selected as the determined final states of the actuators in the current time step.

The selected states of the actuators are sent to the actuators, and the state changes of the actuators can lead to changes in the monitored states. Then, the inhabitant may express his/her reaction to the updated monitored states by changing the states of the actuators to new values $A' = \{a'_1, a'_2, \cdots\}$ to meet the target monitored states.

In this case, the services are target-undefined services without explicitly specifying the target values for the monitored states. Alternatively, the inhabitant can explicitly specify the target values $a'_0 \in A'$ for monitored states. He/She can also predefine these target values in terms of his/her different states, and the services select the target values that correspond to the current environment states. In this case, the services are the target-defined services.

Then, the new states of the sensors and actuators are sent to the "service dispatcher". The HKD-SHO comes to the next time step and repeats the process described above. In the new time step, the "SHOMA architecture-based system" calculates the rewards for the previous time step that precedes the new time step. Together with the states used as inputs to the RL algorithms within the services in the previous time step and the information provided by the "decision maker" in the previous time step, the "PBRE variant" decides whether to extract rules or the "rule deletion" decides whether to delete the extracted rules. If rules are extracted, they are stored in the "extracted rules" database in the "knowledge representation" module. Otherwise, the extracted rules can be deleted from the "extracted rules" database.

As the "SHOMA architecture-based system" is presented in Chapter 6, the "existing rules reasoner" is based on the existing Pellet reasoner, the "extracted rules reasoner" and the "PBRE variant" are presented in Chapter 7, and the "rule deletion" is just about removing certain extracted rules from the "extracted rules" database. In the following sections, we will briefly introduce the working process of "state proposition", "PBRE variant" and "rule deletion", and mainly focus on the presentation of the "service dispatcher" and the "decision maker".

8.2.1 Service dispatcher

After having defined that each device is a service, [35] proposes an IoT service model for service z and represents the model as $< z_{id}, z_{name}, F, Q >$ where (1) z_{id} is a unique identifier. (2) z_{name} is the name of the service. (3) $F = \{f_1, f_2, \cdots, f_n\}$ denotes the functions offered by the service. (4) $Q = \{q_0, q_1, \cdots, q_m\}$ denotes non-functional attributes or quality of service (QoS) preferences. This model clarifies the composition of a service and provides a better understanding of a service.

Motivated by this service model and considering our proposed service definition, we introduce a new data-based model called "service dispatcher" for smart home services by selecting the observable states related to that service. The service dispatcher

is used for two purposes. First, it can be used to visualize better the associated services' components (actuators and sensors). Second, it can tell the ontology whether to create new instances for the sensors, the actuators, and particular types of service when new actuators and sensors that are components of certain types of services are added to the smart home, and the sensors have non-null values. The new data-based model $z_{type,id}$ for a service is expressed as a tuple $< id, type, location, p, D >$. The definition of each component is as follows :

(1) id and $type$ are the unique identifier and type of the service z ;

(2) $location$ is the place where the service is located, e.g. $location = living\ room$ means that this service is in the living room ;

(3) p registers the variables related with the monitored state that z is trying to adjust, i.e. $p = \{air\ quality\}$ indicates that z is an air quality service and is trying to manage the air quality at location $location$;

(4) D stores variables representing devices that are associated with z : $D = \{d_0, \cdots, d_j, \cdots, d_k\}$ where d represents either an actuator or a sensor.

The devices contain both actuators and sensors. The data-based model of an actuator can be expressed as $d_{type,id} = \{id, type, location, W, Z, S_{z,t}, s_t\}$, and the data-based model of a sensor can be expressed as $d_{type,id} = \{id, type, location, W, Z, s_t\}$, where :

(1) id and $type$ are the unique identifier and category of the current device, respectively ;

(2) $location$ is the place where the device is located, e.g. $location = living\ room$ means that the device is located in the living room ;

(3) W stores all possible values of the current device, i.e. a window has possible states $W = \{0, 1\}$ indicating closed or open.

(4) Z contains all services with which the current device is associated. For example, a window may be associated with an air quality service with a unique identifier 1 and a temperature service with a unique identifier 2, such that $Z = \{z_{air,1}, z_{air\ temperature,2}\}$;

(5) $S_{z,t}$ is the state proposed by the service z for the device at time step t. For example, $S_{air\_temperature,t} = \{0\}$ for a window means that the window is proposed to be off by an air temperature service at time step t ;

(6) s_t is the state of the device at time step t. For example, $s_t = 0$ for a window means that the window is off at time step t.

For the service and device models proposed above, if there are no values for a component, we use $None$ to represent it. An example of the data-based model for a light intensity service instance $z_{light\_intensity,1}$ with type light intensity can be expressed as $< id = 1, type = light\ intensity, location = living\ room, p = light\_intensity, D = \{d_{lamp,1}, d_{curtain,1}, d_{inhabitant\_state,1}, d_{light\_intensity,1}, d_{light\_intensity,2}\} >$, where $d_{light\_intensity,1}$ and $d_{light\_intensity,2}$ are the sensor variables respectively capturing the indoor light intensity and the outdoor light intensity.

8.2.2 Rule extraction

To do the rule extraction in a dynamic way, we leverage the PBRE variant for multiple smart home services as shown in Section 7.9 of Chapter 7. The working process is briefly shown in Algo.14.

Suppose that the current time step is t. Thus, the rule extraction to be performed is based on the activities at time step $t-1$. Before extracting rules, we know that, first, the input states for the RL algorithms of the SHOMA architecture-based system are X_{t-1} obtained from the "SHOMA architecture-based system". Second, the rewards R_t computed at time step t represent the reactions of the inhabitant to the proposed actuators' states at time step $t-1$, which are also obtained from the "SHOMA architecture-based system". Third, the determined final actuators' states at time step $t-1$ are A_{t-1}. They are acquired from the "decision maker". Fourth, the message

Algorithm 14: Rule extraction

Result: final extracted rules
X_{t-1} : input states to the RL algorithms within SHOMA architecture-based system
R_t : rewards calculated at time step t
A_{t-1} : final determined actuators' states for time step $t-1$
$Info_{t-1}$: information specifying the propositions of which service are taken at time step $t-1$
GR_t : the updated extracted rules at time step t
Function `RuleExtraction`$(X_{t-1}, A_{t-1}, R_t, Info_{t-1})$:
  if *all rewards in* R_t *are positive* **then**
    if $Info_{t-1}$ *specifies that the propositions of the SHOMA architecture based system are parts of the new states of the actuators at time step* $t-1$ **then**
      $GR_t \leftarrow$ using the PBRE variant (Section 7.9) with inputs X_{t-1} and A_{t-1} to extract rules
    end
  end
  return GR_t

$Info_{t-1}$ received from the "decision maker" indicates which services proposed A_{t-1} at time step $t-1$. Fifth, the updated extracted rules are GR_t.

If the rewards of all services R_t are positive (line 7), and the "decision maker" used $info_{t-1}$ to indicate that the propositions of the learning method-based system are part of A_{t-1} (line 8). In this case, with the input being A_{t-1} and X_{t-1}, the PBRE variant presented in Section 7.9 is used to extract rules and return the updated extracted rules GR_t (line 9). Finally, this algorithm returns GR_t to the database of the extracted rules (line 12)

8.2.3 Rule deletion

Algo.15 shows the process of deleting rules based on the activities at time step $t-1$. Suppose the current time step is t. The total extracted rules is GR_t, the extracted rules that proposed actuators' states at time step $t-1$ is gr_{t-1}, the reward set calculated at time step t representing the reactions of the inhabitant to the actuators' states proposed at time step $t-1$ is R_t, and the message coming out from the "decision maker" is $info_{t-1}$. We have the following working process of the "rule deletion" :

If there is at least one service whose reward calculated at time step t is negative (line 6), and the message $info_{t-1}$ sent by the "decision maker" at time step $t-1$ tells that the propositions of the extracted rules are parts of the final new actuators' states at time step $t-1$ (line 7), in this case, we delete the extracted rules gr_{t-1} that are used to propose actuators' states at time step $t-1$ from GR_t (line 8). Finally, we return the updated extracted rules GR_t to the database (line 11).

Algorithm 15: Rule deletion

Result: final extracted rules
1 GR_t : total extracted rules at time step t
2 gr_{t-1} : the extracted rules that proposed the actuators' states at time step $t-1$
3 R_t : rewards calculated at time step t
4 $Info_{t-1}$: information specifying the propositions of which service are taken at time step $t-1$
5 **Function** `RuleDeletion`(GR_t) :
6 **if** *not all rewards in* R_t *are positive* **then**
7 **if** $Info_{t-1}$ *specifies that the propositions of the extracted rules are parts of the new states of the actuators at time step* $t-1$ **then**
8 $GR_t \leftarrow$ Delete gr_{t-1} from GR_t
9 **end**
10 **end**
11 return GR_t

8.2.4 State proposition

Three structures are available to create services to propose actuators' states : existing rules reasoner, machine learning method-based system, and extracted rules reasoner. In this section, we first explain why we defined three service creation structures in the HKD-SHO, and then provide a brief description of proposing actuators' states using the three structures.

Available service creation structures

Assume that the target monitored states are easy to realize. For example, turning off the lamp and closing the curtain is evident if the inhabitant wants the light intensity to be zero lux when entering the room. Then, there is no need to create a data-driven service that takes time to be well trained. Suppose that it is the preexisting rules based services that handle the decision-making. In this case, the user can always track the preexisting rules to find out in which situation certain preexisting rules based services suggested certain actuators' states. As a result, we define an "existing rules reasoner" to create services.

In addition, rules are extracted from data-driven services only if they propose states of actuators that result in monitored states that satisfy the inhabitant. Therefore, the services created by extracted rules can achieve higher inhabitant satisfaction than the data-driven services. Moreover, during the learning process, the data-driven services sometimes randomly propose states for actuators instead of selecting states more likely to cause monitored states to have target values. This exploration process [37] can discover the inhabitant's less frequent behavior pattern and satisfy the inhabitant. In this case, the extracted rules corresponding to this decision-making can represent the inhabitant's less frequent preferences. Nevertheless, data-driven approaches may ignore these less frequent preferences and retain the more frequent ones. Moreover, when the extracted services are used for decision-making, users can always track the corresponding extracted rules to find out in which historical environment states certain services suggested certain actuators' states. For these reasons, we also define the "extracted rules reasoner".

However, preexisting and extracted rules based services cannot always ensure to cover all possible situations. In this case, we need data-driven services that can propose actuators' states for any environment states. Moreover, the data-driven services can adapt to the changing inhabitant's target monitored states and environment

states. As a result, we also need the "SHOMA architecture-based system" service creation structure.

State proposition

In the "state proposition" module, three ways are provided : existing rules reasoner, extracted rules reasoner and SHOMA architecture-based system. Some attention should be paid for them :

— Existing rules are manually created by the inhabitant, thus they can be incomplete or conflicting.

— Extracted rules are progressively generated during the learning phase, thus they can be incomplete.

— SHOMA architecture-based system by nature is able to propose actuators' states for any environment states at any time step.

The brief process of proposing actuators' states by the three available services is shown in Algo.16. Specifically, knowing that the observable states at time step t is O_t. The updated extracted rules after having potentially performed rule extraction or rule deletion are GR_t. The preexisting rules are ER_t. The selected SHOMA architectures are either EPbA or RSAbA. To propose states for the actuators, the Pellet reasoner infers the actuators' states at time step t when given the observable states O_t and the preexisting rules ER_t (line 6).

Algorithm 16: Propositions of actuators' states

Result: Propositions of the states of the actuators
1 O_t : observable states at time step t
2 GR_t : updated extracted rules after potentially performing rule extraction or rule deletion at time step t
3 ER_t : existing rules at time step t
4 SHOMA : selected SHOMA architectures : EPbA or RSAbA
5 **Function** **`StateProposition`**(O_t) **:**
    `// preexisting rules`
6 actuators' states proposed by preexisting rules $\leftarrow$ Pellet(ER_t, O_t)
    `// extracted rules`
7 $d_t \leftarrow$ maintaining the states in O_t which have the same state types as the states of the conditions of the extracted rules GR_t
8 actuators' states proposed by extracted rules$\leftarrow Inference(GR_t, d_t)$ as shown in Algo.13
    `// learning method-based system`
9 actuators' states proposed by SHOMA architecture-based system $\leftarrow$ SHOMA(O_t)
10 return actuators states respectively proposed by the existing rules, extracted rules and SHOMA architecture-based system

The extracted rules propose actuators states based on a customized reasoner with the inference process as shown in Algo.13 in Chapter 7 by considering not only the ranges of the states in the conditions, but also the PPMCC similarity between the states in the conditions and the corresponding states of the sample under study. Specifically, the observable states O_t of the current sample first maintain those states which have the same types as the states in the conditions of GR_t, and the new states of the sample is denoted as d_t, as shown in line 7 of the current Algo.16. Then, we use Algo.13 with the input being d_t and GR_t to obtain the actuators' states proposed by the extracted rules, as shown in line 8 of the current Algo.16.

To propose actuators' states by the SHOMA architecture-based system, the selected SHOMA architectures, either EPbA or RSAbA, are used to propose actuators' states when given O_t as its input (line 9).

Finally, the actuators' states proposed by the three available services are returned as shown in line 10, so that the "decision maker" presented in the next section can choose the final actuators states.

8.2.5 Decision maker

The "decision maker" is applied to select the services whose proposed actuators' states are chosen to be the new actuators' states in a given time step. The principle of the "decision maker" is shown in Algo.17. To select the final services among preexisting rules based services, extracted rules based services, and data-driven services, we need to determine their priorities. Since it is the inhabitant or the developers who

Algorithm 17: Decision-making for final actuator states

```
   Result: Final states for actuators
1  Az : proposed final states for actuators
2  Function DecisionMaker() :
3      Az ← select actuators' states proposed by preexisting rules
4      for a in Az do
5          if a equals ∅ or has conflicts then
6              a ← select those proposed by extracted rules
7          end
8      end
9      for a in Az do
10         if a equals ∅ then
11             a ← select those proposed by the data-driven services
12         end
13     end
14     return Az
```

define the preexisting rules, the preexisting rules based services can ensure the satisfaction of the inhabitant's preferences or something to which the developers want the home system to pay attention. As a result, the "decision maker" in HKD-SHO prefers the actuators' states proposed by the preexisting rules based services, as shown in lines 3.

If the "exiting rules reasoner" cannot create services to suggest states for specific actuators, or there are conflicts among the propositions of the preexisting rules based services, the "decision maker" selects the actuators' states proposed by the extracted rules, as shown in lines 4∼8. The reason is that the extracted rules are obtained from the SHOMA architecture-based system when the inhabitant is satisfied with the updated monitored states. Therefore, they can almost guarantee (with some uncertainty due to the mechanism used to extract rules) the satisfaction of the inhabitant's preferences. In addition, the extracted rules have higher accuracy than the learning method-based SHOMA architecture-based system [103]. Moreover, the inhabitant or the developers can track the extracted rules to find out in which situations certain services suggested certain states of the actuators. Finally, the propositions by the extracted rules based services will be without conflicts because of the inference mechanism concerning the similarity and frequency of occurrence presented in Section 7.8 of Chapter 7.

However, the rules cannot ensure to cover all possible situations for no matter which environment states. Nevertheless, the "SHOMA architecture-based system" can always create services that propose actuators' states for no matter which environment states. Thus, if there are some situations where the rules based services cannot be created, as shown in lines 9∼13, the "decision maker" selects the actuators' states proposed by data-driven services created by the "SHOMA architecture-based system".

8.3 Working process of HKD-SHO

In this section, we introduce the working process of HKD-SHO, which can be divided into two situations : The first is that the services are target-undefined services, where the inhabitant expresses dissatisfaction with the updated monitored states by changing the actuators' states. The second is that the services are target-defined services, where the inhabitant specifies the target values for the monitored states. The related variables within the working duration are explained as follows :

(1) $O_t = \{O_t^{z_0}, O_t^{z_1}, \cdots\}$ are the observable states at time step t, with $z_0, z_1, \cdots$ being the available services.

(2) $X_t = \{X_t^{z_0}, X_t^{z_1}, \cdots\}$ are the input states at time step t.

(3) R_t are a set of rewards for each service, where $R_t = \{r_t^{z_0}, r_t^{z_1}, \cdots\}$.

(4) $Az_t = \{Az_t^{z_0}, Az_t^{z_1}, \cdots\}$ are the actuators' states proposed by available services.

(5) $S_{z0,t} = \{sz_{0,t}^{z_0}, sz_{0,t}^{z_1}, \cdots\}$ are a set of updated monitored states after the actuators change their states to Az_t, .

(6) $A'_t = \{a'_1, a'_2, \cdots\}$ are a set of actuators' states defined by the inhabitant when the services are target-undefined services.

(7) $A'_t = \{a_0'^{,z_0}, a_0'^{,z_1}, a'_1, a'_2, \cdots\}$ are a set of states including the target monitored states and actuators' states defined by the inhabitant when the services are target-defined services.

(8) $A'_{0,t} = a_{0,t}'^{,z_0} \cup a_{0,t}'^{,z_1}, \cdots$ are a set of target monitored states defined by the inhabitant at time step t for different services.

(9) $S'_{0,t} = \{s_{0,t}'^{,z_0}, s_{0,t}'^{,z_1}, \cdots\}$ are a set of updated monitored states after the actuators' states change their values to $\{a'_1, a'_2, \cdots\}$ at time step t.

Working process of HKD-SHO

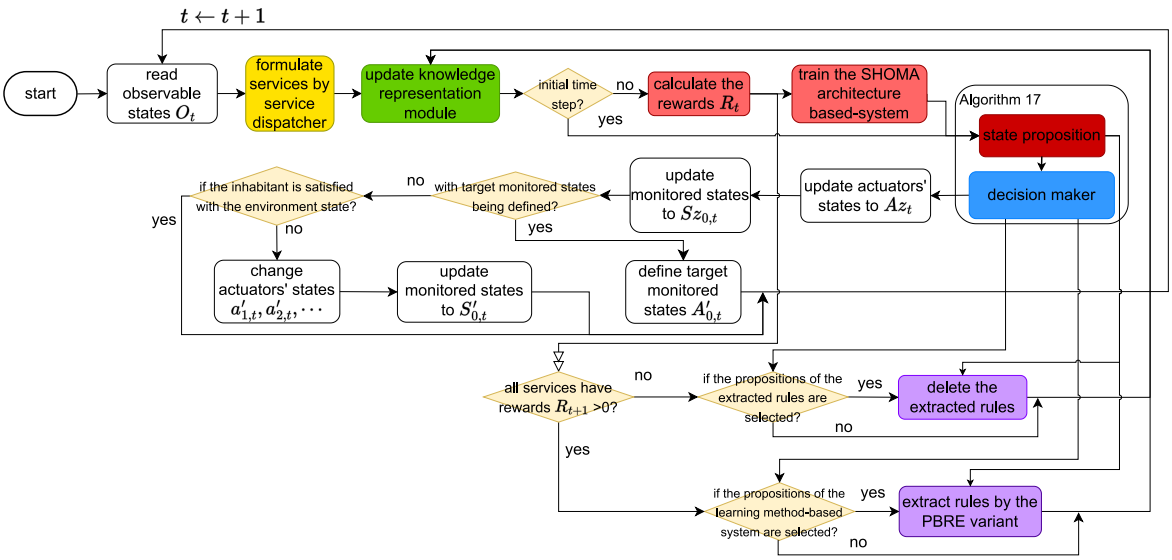

FIGURE 8.2 – Detailed working process of HKD-SHO

The working process of HKD-SHO can be shown in Fig.8.2. When the system starts, we denote the current time step as t. The observable states O_t including sensor variables and actuator variables are sent to the "service dispatcher".

The "service dispatcher" selects the observable states for each service from O_t. Then, the observable states are used to update the knowledge representation module, such as the values in the range of the data properties of the sensor instances and the actuator instances associated with each service instance in the ontology.

In this situation, the system asks whether the current time step is the initial time step. If yes, the updated knowledge is used as input to the Algo.17 so that the algorithm can suggest the final states of the actuators. Otherwise, if the current time step is not the initial one, the SHOMA architecture-based system computes the rewards R_t of the previous time step $t-1$. The rewards R_t correspond to the activities at the previous iteration rather than the rewards R_{t+1} for the current iteration.

Thus, when R_t are sent to the "rule management" module, it is used to potentially perform the rule extraction or deletion considering the activities of the previous time step $t-1$. The two white triangle arrows mean that the diamond condition is waiting for the rewards calculated in the next time step to match the activities of the current time step. The transition of the previous time step for each service z is $d^z_{t-1} = \{O^z_{t-1}, Az^z_{t-1}, sz^z_{0,t-1}, O^z_t, r^z_t\}$.

Then, the SHOMA architecture-based system is trained using the transitions by following Algo.4 in Chapter 5. The updated SHOMA architecture-based system, the updated extracted rules, and the preexisting rules are used to propose actuators' states. The proposed actuators' states are used by the "decision maker" that contains Algo.17 to determine the final states for the actuators. In addition to proposing the final actuators' states Az_t, the "decision maker" also sends some information to the "rule management" module. The information contains the knowledge indicating whether Az_t is proposed by the extracted rules reasoner or the SHOMA architecture-based system. It also contains the values of Az_t. Furthermore, the "state proposition" also sends the input states extracted by the "interpreter" within the "SHOMA architecture-based system" to the "rule management" module.

The information from the "decision maker" is used together with the rewards R_{t+1} computed in the next time step $t+1$ to decide whether to extract rules for the current time step t, delete the extracted rules used to propose the states of the actuators at the current time step t, or do nothing. The actuators change their states to the final

selected states Az_t. The changes in the actuators' states cause the monitored states to switch to $Sz_{0,t}$. Subsequently, HKD-SHO asks whether the inhabitant is satisfied with the updated monitored states so as to calculate R_{t+1} in the next time step.

Suppose the inhabitant is dissatisfied with the updated monitored states. When the services are target-undefined services, in this case, he/she can change the actuators' states to A'_t, and the monitored states will be updated to $S'_{0,t}$. Otherwise, he/she takes no action. Alternatively, when the services are target-defined services, the inhabitant can set the target values for the monitored states. The setting of the target monitored states can also be predefined. The changes of the actuators' states in target-defined services are not mandatory.

The system comes to the next time step $t + 1$, and repeats the above process. During the next time step $t + 1$, the rewards R_{t+1} are calculated. They are sent to the "rule management" module. Suppose that R_{t+1} are not positive for all services, and the message from the "decision maker" states that the propositions of the extracted rules are parts of A_{t+1}. In this case, the extracted rules that are used to make the propositions at time step t are deleted from the "knowledge representation" module. However, suppose that R_{t+1} are positive for all services, and the message of the "decision maker" states that the propositions of the SHOMA architecture-based system are parts of A_{t+1}. In this case, a rule is extracted from the SHOMA architecture-based system using the PBRE variant by considering the inputs X_t to the RL algorithms within the "SHOMA architecture-based system" and the determined final actuators' states Az_t. The extracted rule is stored in the "knowledge representation" module.

Changing the actuators' states to express the dissatisfaction of the inhabitant can almost only happen when HKD-SHO is deployed in the real world. This is because in the simulation, we need to simulate the inhabitant profile specifying how to set the states of the actuators. However, it is difficult to calculate the inverse equations to obtain the states of the actuators to be proposed knowing the input environment states. The calculation becomes incredibly complex when the preferences of the inhabitant are complex, or the number of the associated actuators is large.

8.4 Comparative experiment

To evaluate the performance of HKD-SHO, we use the simulated environments presented in Section 5.5 of Chapter 5 and predefined metrics presented in Section

6.3.1 of Chapter 6 to conduct several experiments. All services are target-defined services. These experiments are divided into experiments with two or three services and without or with constraining the use of the actuators with lower priority in the inhabitant profiles. The performance of HKD-SHO is compared with the system with only a learning method-based system and the system with the learning method-based system and preexisting rules (Its working process can be described as the working process of HKD-SHO without preexisting rules).

8.4.1 Experiment results and analysis

Four comparisons are performed. In each comparison, the simulated environment consists of two or three services with or without the constraint being integrated into the inhabitant profile. Also, in each comparison, two structures, RSAbA and EPbA, selected in Chapter 6, are respectively used to model the learning system. Moreover, the involved preexisting rules in SWRL (Semantic Web Rule Language) are :

(1) smartHome :Curtain(?cur) $\wedge$ *smartHome :Lamp(?lp)* $\wedge$ *smartHome :Person(?p)*$\wedge$ *smartHome : hasState(?p, ?ps)* $\wedge$ *sameAs(?ps, smartHome :sleeping)*
$\rightarrow$ *smartHome :hasState(?lp, smartHome :off)* $\wedge$ *smartHome :hasState(?cur, smartHome :close)*
(2) smartHome :Window(?win) $\wedge$ *smartHome :AirPurifier(?ap)* $\wedge$ *smartHome :AirCondition(?ac)* $\wedge$ *smartHome :Lamp(?lp)* $\wedge$ *smartHome :Person(?p)* $\wedge$ *smartHome :hasState(?p, ?ps)* $\wedge$ *sameAs (?ps, smartHome :absent)*
$\rightarrow$ *smartHome :hasState(?ac, smartHome :off)* $\wedge$ *smartHome :hasState(?lp, smartHome :off)*$\wedge$ *smartHome :hasState(?ap, smartHome :off)* $\wedge$ *smartHome :hasState(?win, smartHome :close)* (8.1)

The first SWRL specifies that if the inhabitant is sleeping, then the lamp is off and the curtain is closed. The second SWRL describes that if the inhabitant is absent, then the air conditioner, the lamp and the air purifier are off, and the window is closed.

In each figure of the evaluation results in the following sections, the yellow bar chart corresponds to the result of the learning method-based system with the learning method indicated in the heading (RSAbA or EPbA) ; the red bar chart shows the result of the learning method-based system with preexisting rules. The green bar chart show the results of HKD-SHO based on the learning method-based system, preexisting rules, and extracted rules. In addition, the horizontal axis indicates the type of metric. For example, the label "temp" explains the accuracy of the temperature service to correctly propose the states of the actuators. The label "air" shows the accuracy of the air quality service to correctly proposes the states of the actuators. The

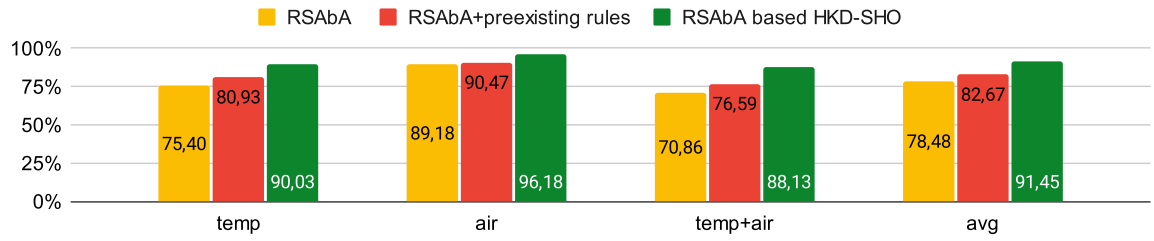

FIGURE 8.3 – Evaluation results with RSAbA being the machine learning-based system and without constraint in the inhabitant's profile

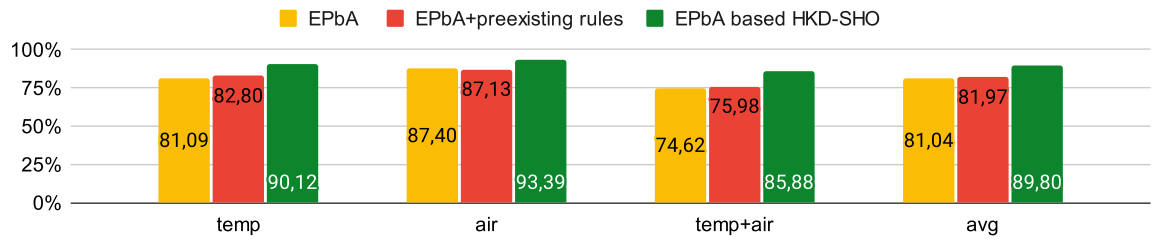

FIGURE 8.4 – Evaluation results with EPbA being the machine learning-based system and without constraint in the inhabitant's profile

label "temp+air" indicates the accuracy of the temperature service and the air quality service to simultaneously and correctly propose the states of the actuators. The label "light+temp+air" specifies the accuracy of the light service, the temperature service and the air quality service to correctly and simultaneously propose actuators' states, and "avg" is the average of all the previous accuracy, which shows the general performance of the corresponding system.

Comparison result under the environment with two services

Figures Fig.8.3 and Fig.8.4 show the evaluation results of the machine learning-based system, the machine learning-based system with existing rules, and the HKD-SHO in the experiment where there are a temperature service and an air quality service. The RSAbA and EPbA are respectively used to model the machine learning-based system in the three systems, and no constraint of minimizing the use of the actuator with lower priority is simulated in the inhabitant's preferences. From the two figures, it can be seen that, first, by observing the metric "avg" and the metric "temp+air", in most of the situations, the systems where the machine learning method is RSAbA generally give better metric results than the systems where the machine learning method is EPbA. Second, when RSAbA and EPbA are respectively the machine learning method, by observing all metrics, the system with only the machine learning

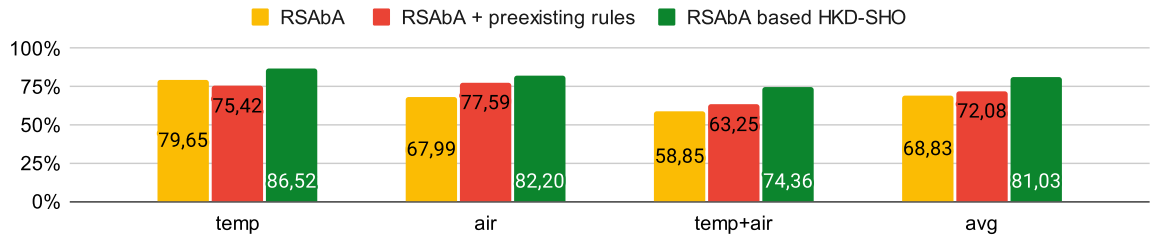

FIGURE 8.5 – Evaluation results with RSAbA being the machine learning-based system and with constraint in the inhabitant's profile

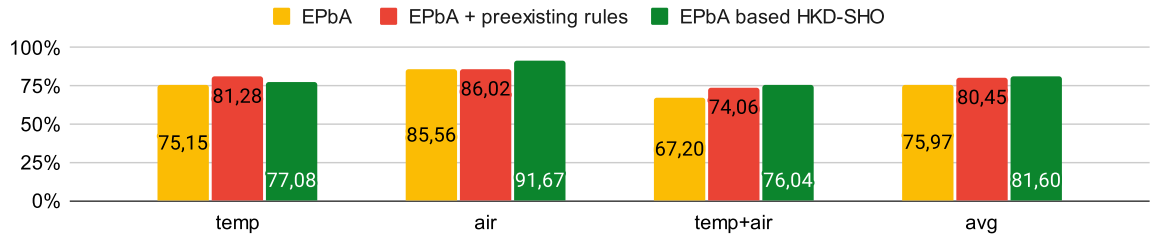

FIGURE 8.6 – Evaluation results with EPbA being the machine learning-based system and with constraint in the inhabitant's profile

method has worse performance compared to the other two systems : the machine learning-based system with existing rules and the HKD-SHO system. The machine learning-based system with existing rules performs better than the systems with only the machine learning method, while the HKD-SHO system performs the best.

Fig.8.5 and Fig.8.6 show the metric results of the systems in an environment with two services, where RSAbA and EPbA are respectively the method for the machine learning system and the constraint of minimizing the use of the actuator with lower priority is integrated into the preferences of the inhabitant. Based on the results, we can conclude that, first, by observing the metrics of "avg" and "temp+air", the systems where the machine learning method is EPbA generally give better metric results than the systems where the machine learning method is RSAbA. Second, by observing all metrics, the system with only the machine learning method generally has the worst performance, while the system HKD-SHO has the best performance.

Comparison result under the environment with three services

Fig.8.7 and Fig.8.8 show the evaluation results of the system based on the machine learning method, the system based on the machine learning method and the existing rules, and the system HKD-SHO. These experiments contain three services and do not specify the constraint of minimizing the use of the actuator with lower priority

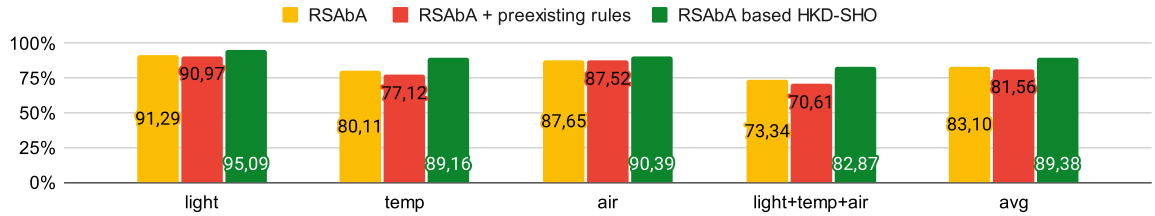

FIGURE 8.7 – Evaluation results with RSAbA being the machine learning-based system and without constraint in the inhabitant's profile

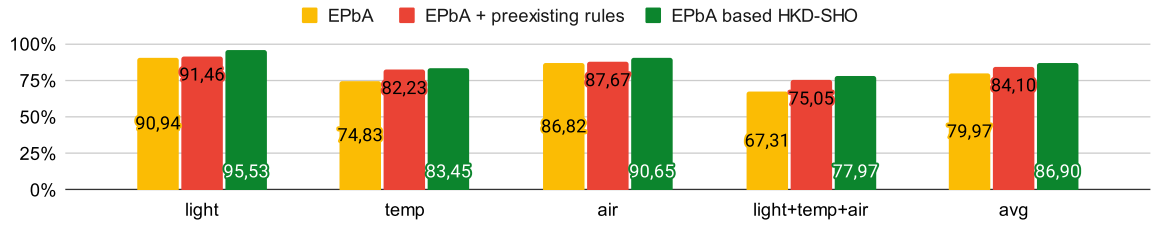

FIGURE 8.8 – Evaluation results with EPbA being the machine learning-based system and without constraint in the inhabitant's profile

in the inhabitant's preferences. From the two figures, it can be seen that, firstly, by observing metrics of "avg" and "light+temp+air", the systems using RSAbA as the method of the machine learning system can achieve better performance than the systems using EPbA as the method of the machine learning system. Second, in both situations where RSAbA and EPbA are the methods of the machine learning system, by observing all metrics, the system containing only the machine learning method in most of the situations performs worse than the system comprising the machine learning method and the existing rules. Meanwhile, the system HKD-SHO has the best performance.

Fig.8.9 and Fig.8.10 show the evaluation results of the machine learning systems, the machine learning system with existing rules, and the HKD-SHO under the environment with three services and with the constraint of minimizing the use of the actuator with lower priority being integrated into the inhabitant's preference. From the two figures, we can see that, first, by observing the metrics "avg" and "light+temp+air", the systems using RSAbA as the machine learning method can generally achieve better performance than the systems using EPbA as the method of the learning system. Second, by observing all metrics, the system using only the machine learning method generally has the worst performance among the three systems, while the system HKD-SHO has the best performance.

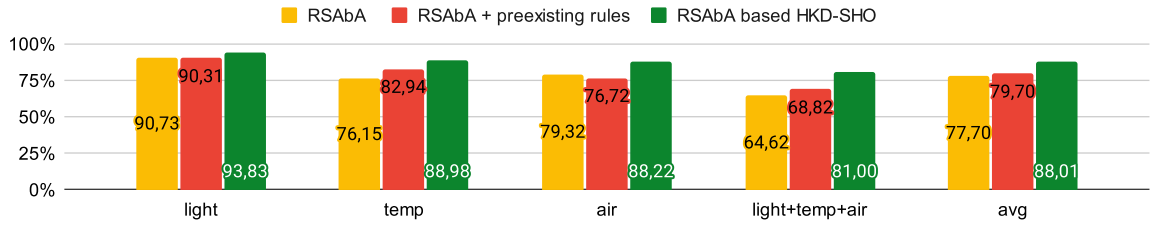

FIGURE 8.9 – Evaluation results with RSAbA being the machine learning-based system and with constraint in the inhabitant's profile

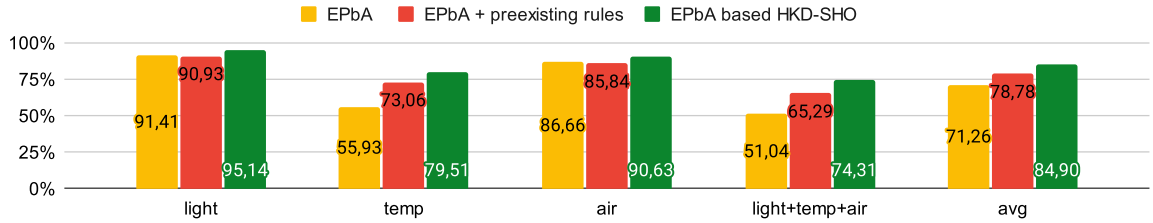

FIGURE 8.10 – Evaluation results with EPbA being the machine learning-based system and with constraint in the inhabitant's profile

Experiment Analysis

From the above experiment results, we can conclude that, first, in most of the situations, by observing the general performance (metric "avg") and the satisfaction of the inhabitant (metric temp+air" for two services and metric "light+temp+air" for three services), the systems using RSAbA as the method of the machine learning system have better performance than systems using EPbA as the method of the machine learning system. Second, in most situations, the system using only the machine learning method has the worst performance, while HKD-SHO has the best performance. The better performance of HKD-SHO can be explained by the fact that the introduction of growing extracted rules provides the machine learning method-based system with the learning direction. Moreover, these rules are extracted when the machine learning-based system can contribute to obtaining satisfying monitored state values. As a result, these rules can better ensure the satisfaction of the inhabitant than the machine learning method-based system or the machine learning method-based system with a fixed number of existing rules.

8.5 Conclusion

Data-driven approaches can create data-driven services to dynamically propose actuators' states to adapt to the changing environment states and the inhabitant's preferences. The propositions of actuators' states can be made by data-driven services for any environment states. However, the data-driven services cannot explain to the inhabitant in which situations certain services have suggested certain actuators' states.

Knowledge-based approaches allow the inhabitant to manually create simple smart home services if the inhabitant knows how to set the actuators' states to meet the target monitored states. The created knowledge-based services can guide the learning direction of the data-driven services. Moreover, the inhabitant understands in which situations certain actuators' states are suggested. However, knowledge-based services are usually static and cannot evolve as the inhabitant's preferences and the environment states change. In addition, they cannot always ensure that the knowledge-based services have covered all possible environment states. Moreover, there can be potential conflicts among knowledge-based services.

To overcome the disadvantage of the two approaches while maintaining their advantages, we propose HKD-SHO, which combines knowledge-based and data-driven services. There are three types of services in HKD-SHO : The first type is created by preexisting rules. These preexisting rules based services are usually created by the inhabitant who knows how to set the corresponding actuators to meet the target monitored states. Manual service creation saves the time of training the corresponding data-driven services.

The second type is the extracted rules based services, which have higher accuracy than the data-driven services and are used to enrich the preexisting rules based services. Moreover, their propositions are without conflicts and take over the propositions of the preexisting rules based services when there are conflicts among those propositions by preexisting rules based services, which guarantees that the final determined services are conflict-free. Furthermore, the two types of services can be tracked to show the inhabitant in which situations the services suggested certain actuators' states.

The third type is the data-driven services that can propose actuators' states for any environment states. This type of services can adapt to the changing environment

states. They can also ensure the adaptation of the extracted rules based services.

HKD-SHO consists of five modules : "service dispatcher", "knowledge representation", "state proposition", "rule management", and "decision maker". The "service dispatcher" is used to describe the components (sensors and actuators) of each associated service. The "knowledge representation" stores the ontology and the rules, whose values are updated using the "service dispatcher" and the "PBRE variant" rule extraction module.

In "state proposition", three service creation structures are defined for three types of services : the "existing rules reasoner", the "extracted rules reasoner", and the "SHOMA architecture-based system". They respectively create preexisting rules based service, extracted rules based service and data-driven service by proposing actuators' states that match the current environment states.

Then, the "decision maker" is used to select the final services that will be used to propose states for actuators. During the selection of the final services, preference is given to the preexisting rules based service created by the "existing rules reasoner", then the extracted rules based service created by the "extracted rules reasoner", and finally the data-driven service created by the "SHOMA architecture-based system".

Next, the "rule management" module is used to update the extracted rules. Suppose that the selected services can cause the monitored states to have target values, and the propositions of the data-driven service are parts of the final determined actuators' states, in that case, the rules can be extracted by the "PBRE variant" in the "rule extraction" module. However, suppose that the selected actuators' states cannot cause some monitored state to have its target value, and the propositions of the extracted rules based service are parts of the final determined actuators' states. In this case, the corresponding triggered extracted rule is deleted.

To evaluate HKD-SHO, we compare it with two other systems : the system that uses only the learning method and the system that uses both the learning method and the preexisting rules. These comparisons are realized through several simulated experiments. Each experiment is with two or three target-defined services and with or without constraining the use of the actuators with lower priority. The selected SHOMA architectures are used in each environment to model the learning system. The results show that HKD-SHO has better performance concerning our predefined metrics.

Part III

Conclusion and Annexes

9

Conclusion

Contents

In this chapter, we first present the conclusion of our entire work, which includes context, problems, and contributions. We then provide several interesting perspectives to explore.

9.1 Main results

With the introduction of the IoT, the smart home is becoming more and more popular. Its intelligence is realized by creating various services. The establishment of smart home services usually involves logical and physical specification. The logical specification concerns on designing the logic of decision making by proposing states to actuators after having considered the environment state values sensed by sensors. The physical specification involves the hardware part including how to collect data from the environment, how to transmit data to services using different protocols and how to deploy the smart home system in the real world. In our study, we only focus on logical decision-making to create smart home services. Knowledge-based and data-driven approaches are two primary approaches to creating services. Nevertheless, neither of them can realize the service creation in a satisfactory manner

In our setting, a smart home service is a mechanism that communicates with devices (sensors and actuators) in order to produce the appropriate changes of the cor-

responding monitored state so as to comply with the inhabitant's needs.

Our first proposal (Chapter 5) is a RL method to design a single service for a smart home. We distinguished two kinds of services, namely, (a) target-defined where the user sets a value for a special variable, the target variable, and where the service monitors another special variable, the monitored variable which is brought the closest possible to the target. (b) target-undefined where the service relies only on the observations of the inhabitant's actions on actuators which produce either a positive or negative reward. We then presented a simulation solution that allows to pretrain the service in order to accelerate the learning phase.

Then, in Chapter 6, we extended the RL-based single service model and proposed several architectures with a collective name SHOMA, in order to create multiple smart home services. The SHOMA architectures ensure that there are no conflicts among the created services. They are classified into merged service-based architectures and composite service-based architectures depending on whether all services are merged as one service or not. The two architecture categories are further divided into several subcategories. We evaluated these architectures through several simulated experiments. In each experiment, two or three simulated target-defined smart home services are modeled. In addition, the simulated inhabitant profile includes the targets of the monitored states and, optionally, constrains the use of the actuators with lower priority. The experiments demonstrate the better performance of the composite service-based architectures and select the architectures : RSAbA and EPbA.

In Chapter 7, in order to associate an equivalent reasoning based decision-making mechanism to the data-driven services, we proposed a rule extraction method called PBRE. PBRE has been designed to extract rules that make conflict-free state propositions from already trained data-driven approaches while not relying on the number of the output, the input data types and the structure of the data-driven algorithms. We conducted several experiments to evaluate the performance of PBRE and compare it with an existing rule extraction method. These experiments involve extracting rules from learning systems used to model six benchmark datasets and from learning systems used to simulate three light intensity services. The results showed better performance of PBRE. We also showed how rule extraction can be integrated into service creation for a smart home system. Furthermore, we modified PBRE to ensure that it can dynamically extract rules that can make conflict-free propositions during the training processes of the data-driven approaches (before the training is completed).

Moreover, we proposed a mechanism to realize the rule extraction from multiple services based on the PBRE variant so that the rule extraction can be applied to a general smart home system.

Finally, we proposed a hybrid system called HKD-SHO, which combines the selected SHOMA architectures (EPbA and RSAbA) and the PBRE variant (for dynamic rule extraction from multiple services). HKD-SHO can create dynamic services based on three types of smart home services : preexisting rules based service, extracted rules based service, and data-driven service. These created services are explicable, conflict-free and with an accelerated learning phase. The reasons for the existence of the three types of services are explained as follows.

For preexisting rules based services, if the inhabitant's target monitored state is easy to realize, then there is no need to create a data-driven service that takes time to be well trained. Moreover, if the decision-making from these rules is not satisfactory, the user can track and modify the unsatisfying rule.

For extracted rules based services, rules are extracted from data-driven services only when they propose actuators' states that can meet the targets of the monitored states. Therefore, the services created by extracted rules can achieve higher inhabitant satisfaction than the corresponding data-driven services. In addition, during the learning process, the data-driven mechanism sometimes randomly suggest states for actuators instead of selecting actuators' states more likely to lead to satisfying monitored states. This exploration process can discover the inhabitant's less frequent behavior patterns. In this case, the extracted rules corresponding to these patterns can represent less frequent services. Such less frequent services can be ignored by data-driven services. Moreover, when the extracted rules based services are used for decision-making, they can help to justify the decision-making. Furthermore, the states proposed for the actuators are conflict-free.

For data-driven services, since both preexisting rules based services and extracted rules based services cannot ensure to cover all possible situations, we still need data-driven services to perform decision-making by interacting with the inhabitant. Since the data-driven services adapt to the inhabitant's preferences and the changing environment states, and rules are extracted only for the cases where the data-driven services succeeded in providing satisfying actuators' states, the data-driven services can guarantee the correctness and adaptation of the extracted rules based services.

HKD-SHO is evaluated through several simulated experiments. In each expe-

riment, HKD-SHO simulates a smart home system with two and three simulated target-defined services with and without constraining the use of the actuator wit lower priority. Furthermore, in each experiment, HKD-SHO-based smart home system is compared with smart homes, respectively modeled by two other systems. The first is the learning method-based system without preexisting rules, and the second is the learning method-based system with preexisting rules. The evaluation results show the better performance of HKD-SHO.

9.2 Discussion

Our proposals have some limitations :

1. In our approach to define a single service, we used a time step and made the hypothesis that the inhabitant reacts within the time step. This hypothesis may be too strong and is not easy to be satisfied by some services.
2. We did not provide a method to ensure that the interval of the time step used by a given service is appropriate.
3. During the proposition of SHOMA architectures, we supposed that the time steps of all services have the same time intervals. Finding this common interval is not guaranteed in all situations.
4. In our PBRE algorithm to generate rules, we did not consider the properties (e.g., conciseness) of the generated rules and whether they are easily used for the purpose of explainability.
5. In the simulation phase, it is difficult to consider target-undefined services because of the complexity of modeling environment behavior.
6. In our SHOMA architectures to create multiple services, we did not work on mixing target-defined and target-undefined services.

9.3 Perspectives

Some perspectives are interesting to be explored in our future studies. We could improve the simulation from the following perspectives :

1. When simulating a smart home system with multiple target-defined services, the inhabitant profile can contain the target monitored states and the constraint

on the use of the actuators with lower priority. This constraint is a first step to realize energy saving. It can be extended by considering the working duration to the states of energy consuming actuators.

2. To deploy a smart home system with RL-based services in the real world, we propose to pretrain these services in simulation. After being well trained, these services are deployed in the real world and continue their training process. To ensure that the pretrained services have a better knowledge about the preferences of the real inhabitant, we should ensure that the simulated inhabitant profile and the simulated physical phenomena of the environment states are as similar as possible to those of the real world.
3. When trying to model the dynamic characteristics of the environment states in the simulation phase, it can be interesting to consider the co-influence between different physical phenomena, for example, the indoor temperature can influence the indoor humidity through co-simulation.

Considering the energy saving, we can have the following perspectives :

1. To realize the energy saving for a specific service, it would be interesting in a multiple services architecture to define an energy saving service and combine this energy saving service with that specific service to try to realize the energy saving for that specific service.
2. It would also be interesting to define an energy saving service to try to achieve energy savings for all smart home services and not just one specific service.

Finally, we can work on improving the inhabitant's experience from the following perspectives :

1. When evaluating the rule extraction capability of PBRE and the HKD-SHO, it is essential to assess the explicability of the extracted rules and the performance of HKD-SHO using qualitative results obtained through focus groups.
2. Our study only considers a smart home system with only one inhabitant. It would be promising to study how to improve the contributions for a smart home system with multiple inhabitants.

A

Synopsis en Français

Chapitre 1

Avec l'introduction de l'IdO (Internet des Objets), la maison intelligente devient de plus en plus populaire. Son intelligence se concrétise par la création de services de maison intelligente implique généralement une spécification logique et physique. La spécification logique concerne la conception de la logique de prise de décision en proposant des états aux actionneurs après avoir pris en compte les valeurs d'état de l'environnement détectées par les capteurs. La spécification physique concerne la partie matérielle, compris la manière de collecter les données de l'environnement, de transmettre les données aux services à l'aide de différents protocoles et de déployer le système de maison intelligente dans le monde réel. Dans notre étude, nous nous concentrons uniquement sur la prise de décision logique pour créer des services de maison intelligente.

Pour créer des services, nous cherchons à résoudre quatre questions de recherche : (1). Comment créer un service de maison intelligente qui peut s'adapter à l'évolution dynamique des préférences de l'habitant ? (2). Comment créer des services multiples qui fonctionnent sans conflits ? (3). Comment rendre les services explicables ? (4). Comment garantir les caractéristiques ci-dessus en même temps ? Pour répondre à ces questions, dans notre étude, nous avons fait un état de l'art des approches existantes pour créer des services. Ces approches peuvent être classées en trois catégories : Les approches basées sur la connaissance, les approches dirigées par les données, et les approches hybrides qui contiennent les deux catégories. Nous avons analysé les avantages et les inconvénients de ces approches. Pour résoudre ces inconvénients, hé-

riter de ses avantages et résoudre les questions de recherche, nous avons fait quatre contributions, chacune pouvant répondre à l'une des questions de recherche.

Chapitre 2

Les approches basées sur la connaissance sont l'une des plus importantes pour le développement de services de maison intelligente. Un système basé sur les connaissances se compose généralement d'une représentation des connaissances et d'un moteur d'inférence. La représentation des connaissances contient divers faits et règles. La création de services de maison intelligente à l'aide d'approches basées sur la connaissance consiste à établir un ensemble de règles. Il existe plusieurs types de représentations des connaissances, dont l'ontologie est l'une des plus connues, et OWL (Web Ontology Language) est l'une des ontologies les plus connues conçues pour le web sémantique. Comme notre étude se concentre sur la maison intelligente, malgré les nombreuses ontologies, nous avons choisi d'utiliser l'ontologie SAREF (Smart Appliances REFerence) dans notre travail, qui facilite la mise en correspondance des modèles sémantiques existants dans le domaine des appareils intelligents et réduit l'effort de conversion d'un modèle à l'autre.

Chapitre 3

Les approches dirigées par les données constituent une autre catégorie principale d'approches utilisées pour le développement de services de maison intelligente. Dans ce type d'approches, les services sont créés en proposant des états aux actionneurs correspondants après avoir pris en compte les états actuels de l'environnement. Notre étude se concentre principalement sur les approches dirigées par les données et basées sur l'apprentissage automatique. Cette approche peut créer dynamiquement des services en apprenant à partir des ensembles de données disponibles. Selon que les suggestions affectent ou non l'environnement, les algorithmes actuels d'apprentissage automatique sont divisés en RL (Reinforcement Learning) lorsque la réponse est positive et en d'autres algorithmes lorsque la réponse est négative. Selon qu'il existe une cible ou une étiquette pour chaque exemple au cours du processus d'apprentissage, ces autres algorithmes peuvent être divisés en apprentissage supervisé lorsqu'il existe des cibles ou des étiquettes et en apprentissage non supervisé lorsqu'il n'y a pas de

cibles ou d'étiquettes. Comme le RL peut créer dynamiquement des services en interagissant avec l'environnement, nous avons tendance à l'utiliser pour créer des services de maison intelligente qui peuvent suggérer dynamiquement l'état des actionneurs en fonction de la satisfaction ou de l'insatisfaction de l'habitant. En outre, bien qu'il existe de nombreux algorithmes de RL, compte tenu de la difficulté de modéliser l'environnement sur la base d'un ensemble de probabilités de transition et de la nécessité de prendre en compte la réaction de l'habitant à chaque étape du temps, dans notre étude, nous utilisons Q learning comme algorithme de RL sélectionné. Il est également proposé d'intégrer le RL à d'autres algorithmes d'apprentissage automatique tels que les MLP (MultiLayer Perceptron) et les cellules LSTM (Long Short Term Memory) afin que le RL puisse apprendre les modèles de systèmes plus complexes.

Chapitre 4

Ni les approches basées sur les connaissances ni celles dirigées par les données ne peuvent créer des services de manière satisfaisante, car les approches basées sur les connaissances sont limitées en termes d'adaptabilité dynamique (les règles de comportement sont prédéfinies au moment de la conception) et les approches dirigées par les données sont limitées en termes d'explicabilité (la logique de la prise de décision n'est pas explicite). Pour résoudre ces problèmes, certains travaux ont proposé des systèmes hybrides qui combinent les deux approches : ils utilisent les services basés sur la connaissance pour contrôler le système jusqu'à ce que les services dirigés par les données soient bien entraînés. Ensuite, les services dirigés par les données remplaceront les services basés sur les connaissances pour contrôler le système. Cependant, les services basés sur les connaissances ne peuvent pas garantir qu'ils peuvent faire des propositions pour n'importe quel environnement, tandis que les services dirigés par les données sont comme des boîtes noires, et l'habitant ne sait pas dans quelles situations certains services ont proposé certains états d'actionneurs. Par conséquent, nous envisageons un autre système hybride pour surmonter les problèmes du système hybride existant.

Chapitre 5

Dans notre étude, nous définissons qu'un service de maison intelligente est un mé-

canisme qui communique avec des dispositifs (capteurs et actionneurs) afin de produire les changements appropriés de l'état monitoré correspondant afin de se conformer aux besoins de l'habitant. Dans ce contexte, notre première proposition est une méthode RL pour concevoir un seul service pour une maison intelligente. Nous avons distingué deux types de services, à savoir : (a) cible définie où l'utilisateur fixe une valeur pour un état monitoré que le service surveille. (b) cible non définie où le service se base uniquement sur les observations des actions de l'habitant sur les actionneurs, et cette action produit soit une récompense positive, soit une récompense négative. Nous avons ensuite présenté une solution de simulation qui permet de pré-entraîner le service afin d'accélérer la phase d'apprentissage.

Chapitre 6

nous avons étendu le modèle de service unique basé sur le RL et proposé plusieurs architectures sous le nom collectif de SHOMA (Smart HOme-based Multi-services Architectures), afin de créer de multiples services de maison intelligente. Les architectures SHOMA garantissent qu'il n'y a pas de conflits entre les services créés. Elles sont classées en architectures basées sur des services fusionnés et en architectures basées sur des services composites, selon que tous les services sont fusionnés en un seul service ou non. Les deux catégories d'architecture sont ensuite divisées en plusieurs sous-catégories. Nous avons évalué ces architectures au moyen de plusieurs expériences simulées. Dans chaque expérience, deux ou trois services simulés de maison intelligente de type cible définies sont modélisés. En outre, le profil de l'habitant simulé inclut les cibles des états monitorés et, optionnellement, contraint l'utilisation des actionneurs de moindre priorité. Les expériences démontrent la meilleure performance des architectures composites basées sur les services et sélectionnent les architectures : RSAbA (Remove Shared Actuators-based Architecture) et EPbA (Equal Priority-based Architecture).

Chapitre 7

Afin d'associer un mécanisme décisionnel équivalent basé sur le raisonnement aux services dirigés par les données, nous avons proposé une méthode d'extraction de règles appelée PBRE (Pedagogic Based Rule Extractor). PBRE a été conçue pour ex-

traire des règles qui font des propositions d'état sans conflit à partir d'approches dirigées par les données déjà entraînées, sans dépendre du nombre de sorties, des types de données d'entrée et de la structure des algorithmes dirigés par les données. Nous avons mené plusieurs expériences pour évaluer les performances de PBRE et les comparer à une méthode d'extraction de règles existante. Ces expériences impliquent l'extraction de règles à partir de systèmes d'apprentissage utilisés pour modéliser six ensembles de données de référence et à partir de systèmes d'apprentissage utilisés pour simuler trois services d'intensité lumineuse. Les résultats ont montré une meilleure performance de PBRE. Nous avons également montré comment l'extraction de règles peut être intégrée dans la création de services pour un système de maison intelligente. En outre, nous avons modifié PBRE pour garantir qu'il peut extraire dynamiquement des règles qui peuvent faire des propositions sans conflit pendant les processus de l'apprentissage des approches dirigées par les données. En outre, nous avons proposé un mécanisme pour réaliser l'extraction de règles à partir de services multiples basés sur la variante PBRE afin que l'extraction de règles puisse être appliquée à un système de maison intelligente général.

Chapitre 8

Nous avons proposé un système hybride appelé HKD-SHO (Hybrid Knowledge-based and Data-driven services-based Smart HOme system), qui combine les architectures SHOMA sélectionnées (EPbA et RSAbA) et la variante PBRE (pour l'extraction dynamique de règles à partir de services multiples). HKD-SHO peut créer des services dynamiques basés sur trois types de services : les services basés sur des règles préexistantes, les services basés sur des règles extraites et les services basés sur des données. Ces services créés sont explicables, exempts de conflits et dotés d'une phase d'apprentissage accélérée. Les raisons de l'existence de ces trois types de services sont expliquées ci-dessous.

Pour les services basés sur des règles préexistantes, si la valeur cible de l'état monitoré définie par l'habitant est facile à réaliser, il n'est pas nécessaire de créer un service basé sur des données qui nécessite du temps pour être bien formé. De plus, si la prise de décision à partir de ces règles n'est pas satisfaisante, l'utilisateur peut suivre et modifier la règle qui n'est pas satisfaisante.

Pour les services basés sur des règles extraites, les règles sont extraites des services

dirigés par les données uniquement lorsqu'elles proposent des états d'actionneurs qui peuvent atteindre les cibles des états monitorés. Par conséquent, les services créés par les règles extraites peuvent atteindre une plus grande satisfaction de l'habitant que les services correspondants dirigés par les données. De plus, pendant le processus d'apprentissage, le mécanisme dirigés par les données suggère parfois au hasard des états pour les actionneurs au lieu de sélectionner les états des actionneurs les plus susceptibles de conduire à des valeurs cibles des états monitorés. Ce processus d'exploration peut découvrir les comportements les moins fréquents de l'habitant. Dans ce cas, les règles extraites correspondant à ces motifs peuvent représenter des services moins fréquents. Ces services moins fréquents peuvent être ignorés par les services basés sur les données. De plus, lorsque les services basés sur des règles extraits sont utilisés pour la prise de décision, ils peuvent aider à justifier la prise de décision. De plus, les états proposés pour les actionneurs sont sans conflit.

Pour les services dirigés par les données, étant donné que les services préexistants basés sur des règles et les services extraits basés sur des règles ne peuvent pas couvrir toutes les situations possibles, nous avons toujours besoin de services dirigés par les données pour prendre des décisions en interagissant avec l'habitant. Étant donné que les services dirigés par les données s'adaptent aux préférences de l'habitant et à l'évolution de l'environnement, et que les règles ne sont extraites que dans les cas où les services dirigés par les données ont réussi à fournir des états d'actionneurs satisfaisants, les services dirigés par les données peuvent garantir l'exactitude et l'adaptation des services basés sur des règles extraites.

HKD-SHO est évalué au moyen de plusieurs expériences simulées. Dans chaque expérience, HKD-SHO simule un système de maison intelligente avec deux et trois services simulés de type cible définis avec et sans contrainte d'utilisation de l'actionneur avec une priorité plus faible. En outre, dans chaque expérience, le système de maison intelligente basé sur HKD-SHO est comparé à des maisons intelligentes modélisées par deux autres systèmes. Le premier est le système basé sur la méthode d'apprentissage sans règles préexistantes, et le second est le système basé sur la méthode d'apprentissage avec règles préexistantes. Les résultats de l'évaluation montrent les meilleures performances de HKD-SHO.

Chapitre 9

La conclusion sur les principaux résultats de notre travail est fournie. En outre, nous avons discuté de certaines limites de notre travail. Par exemple, l'habitant doit réagir dans le pas de temps, ce qui peut être une hypothèse forte lorsque le système est déployé dans le monde réel. En outre, des perspectives intéressantes sur différents aspects, tels que l'économie d'énergie et l'expérience de l'habitant, sont fournies.

B

Publications

- Qiu, M., Najm, E., Sharrock, R., & Traverson, B. (2022, July). Pbre : A rule extraction method from trained neural networks designed for smart home services. In Database and Expert Systems Applications : 33rd International Conference, DEXA 2022, Vienna, Austria, August 22–24, 2022, Proceedings, Part II (pp. 158-173). Cham : Springer International Publishing. [103]

- M. Qiu, E. Najm, R. Sharrock and B. Traverson, "Reinforcement Learning Based Architectures for Dynamic Generation of Smart Home Services," 2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA), Nassau, Bahamas, 2022, pp. 7-14, doi : 10.1109/ICMLA55696.2022.00010. [104]

- M. Qiu, E. Najm, R. Sharrock and B. Traverson, "Reinforcement Learning Based Architectures for the Creation of Dynamic Smart Home Services" submitted to the journal Neurocomputing.

- M. Qiu, E. Najm, R. Sharrock and B. Traverson, "HKD-SHO : A hybrid system based on knowledge-based and data-driven services for a smart home" submitted the 35th IEEE International Conference on Tools with Artificial Intelligence (ICTAI).

C

Acronyms

(1). **SAREF** : Smart Appliances REFerence

(2). **OWL** : Web Ontology Language

(3). **IoT** : Internet of Things

(4). **FOL** : First Order Logic

(5). **DL** : Description Logic

(6). **SWRL** : Semantic Web Rule Language

(7). **RDF** : esource Description Framework

(8). **RL** : Reinforcement learning

(9). **MBRL** : Model-Based Reinforcement Learning

(10). **MBFL** : Model-Free Reinforcement Learning

(11). **MC** : Monte Carlo Learning

(12). **TD** : Temporal Difference

(13). **DRL** : Deep Learning-based Reinforcement Learning

(14). **MLP** : MultiLayer Perceptions

(15). **LSTM** : Long Short Term Memory

(16). **RNN** : Recurrent Neural Network

(17). **SHOMA** : Smart HOme-based Multi-services Architectures

(18). **OLSbA** : One Learning System-based Architecture

(19). **QmixbA** : Qmix-based Architecture

(20). **RSAbA** : Remove shared Actuators based Architecture

(21). **CCbA** : Common Controller-based Architecture

(22). **PbA** : Priority-based Architecture

(23). **EPbA** : Equal Priority based Architecture

(24). **TRbA** : Total Reward-based Architecture

(25). **CSbA** : Context Sharing-based Architecture

(26). **PBRE** : Pedagogic Based Rule Extractor

(27). **HKD-SHO** : Hybrid Knowledge and Data-driven services based Smart HOme system

(28). **PPMCC** : Pearson Product Moment Correlation Coefficients

(29). **DQN** : Deep Q-Network

D

Glossary

(1). **environment states** : The environment stands for the surroundings inside and outside the house. The environment states are various physical variables used to describe the environment, e.g., temperature, air quality, light intensity, etc.

(2). **monitored state** : A monitored state is a state that is of interest to the inhabitant, usually inside the smart home, such as the temperature, light intensity, or air quality in the room. Its value is influenced either by external phenomena (natural or artificial) or controlled by the execution of different actions by different actuators.

(3). **observable environment states** : The environment states whose values can be observed by users are called observable environment states, such as the states of the actuators and the states whose values can be captured by sensors

(4). **target monitored state** : The value that the monitored state is required to achieve, and can only be determined by the inhabitant.

(5). **smart home** : A smart home is a house with numerous services that can dynamically evolve through appropriate devices.

(6). **smart home service** : Each smart home service adjusts one monitored state by commanding a set of actuators to perform corresponding actions according to the environment state values detected by the associated sensors.

(7). **action quality value** : An action quality value represents the long-term reward that the RL agent could receive if it selects the corresponding state as the new state for the associated actuator.

(8). **training dataset** : The dataset used to train the data-driven based systems.

(9). **test dataset** : This dataset usually comes from the same distribution with the training dataset. It is used to evaluate the data-driven based systems and is never used to train the data-driven system.

(10). **Simple smart home system** : A simple smart home system is a smart home system that contains one smart home service.

(11). **knowledge-based service** : The knowledge-based service is composed by a set of rules. These rules can be triggered to propose actuators' states so as to change the values of monitored states.

(12). **well-trained learning system** : a learning system is well-trained if the parameter values are stable and do not show large changes. Or the difference between the prediction of the learning method-system and the corresponding target value is small and does not show large changes.

(13). **insignificant states** : Insignificant states, or more specifically, insignificant input states, means that the deletion of these input states has no significant influence on the precision of the learning method-based system.

E

Variables

(1). r : reward

(2). t : time step t.

(3). T : total number of time steps.

(4). ep : one episode.

(5). $Epoch$: total number of episodes.

(6). O : observable environment states.

(7). X : input states of (an) RL algorithm(s)

(8). S : a set of sensor variables storing values of different states. $S = \{s_0, s_1, \cdots\}$, were s_0 is the monitored variable holding the value of the monitored state.

(9). A : actuator variables storing actuators' current states. $A = \{a_0, a_1, a_2, \cdots\}$, where a_0 is the target variable holding target values to be met by the monitored state, and can only be defined by the inhabitant through a'_0.

(10). A' : a set of variables used to assign values to actuator variables by the inhabitant. $A' = \{a'_0, a'_1, a'_2, \cdots, \}$, where a'_0 is the target assignment variable, and the inhabitant uses it to define target value for a_0.

(11). Az : variables used to assign values to actuator variables by the service, where $Az = \{a_{z1}, a_{z2}, \cdots, \}$.

(12). Q : action quality value

(13). M : replay memory composed by multiple transitions.

(14). e : one transition

(15). L : minibatch representing the minimum number of transitions used to train the RL algorithm

(16). sz_0 : updated monitored state, resulting from the actuators changing their states to the states proposed by the service

(17). s'_0 : updated monitored state, resulting from the actuators changing their states to the states proposed by the inhabitant

(18). us : sensor variable capturing the inhabitant state.

(19). le : sensor variable capturing the outdoor light intensity.

(20). te : sensor variable capturing the outdoor temperature.

(21). ae : sensor variable capturing the outdoor air quality.

(22). lr : sensor variable capturing the indoor light intensity.

(23). tr : sensor variable capturing the indoor temperature.

(24). ar : sensor variable capturing the indoor air quality.

(25). lp : actuator variable representing the lamp state.

(26). cur : actuator variable representing the curtain state.

(27). ac : actuator variable representing the air conditioner state.

(28). win : actuator variable representing the window state.

(29). ap : actuator variable representing the air purifier state.

(30). wct : actuator variable representing the working duration for windows and curtain.

(31). act : actuator variable representing the air conditioner working duration.

(32). apt : actuator variable representing the air purifier working duration.

(33). GR : extracted rules

(34). ER : preexisting rules

.

Bibliography

[1] Smart appliances reference (saref) ontology. https://sites.google.com/site/smartappliancesproject/ontologies/reference-ontology.

[2] Facts, rules, goals and queries. http://www.ablmcc.edu.hk/~scy/prolog/pro02.htm.

[3] Air flow volume and velocity. https://www.engineeringtoolbox.com/natural-draught-ventilation-d_122.html.

[4] Rule builder's guide. https://publib.boulder.ibm.com/tividd/td/tec/GC32-0669-01/en_US/HTML/RBGmst324.htm.

[5] Ifttt. https://ifttt.com/, .

[6] What is ifttt ? the ultimate guide. https://www.safewise.com/au/what-is-ifttt/#:~:text=IFTTT%20stands%20for%20%E2%80%9CIf%20This,camera%20and%20some%20smart%20lights., .

[7] What is ifttt ? how to use if this, then that services. https://www.computerworld.com/article/3239304/what-is-ifttt-how-to-use-if-this-then-that-services.html, .

[8] A brief history of smart home automation. https://zeusintegrated.com/blog/item/a-brief-history-of-smart-home-automation, . Accessed : 2022-12-26.

[9] The number of smart homes in europe and north america reached 105 million in 2021. https://www.berginsight.com/the-number-of-smart-homes-in-europe-and-north-america-reached-105-million-in-2021, . Accessed : 2022-12-26.

[10] F. K. Aldrich. Smart homes : past, present and future. In *Inside the smart home*, pages 17–39. Springer, 2003.

[11] E. Alpaydin. *Machine learning*. MIT Press, 2021.

[12] R. Andrews, J. Diederich, and A. B. Tickle. Survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-Based*

Systems, 8(6) :373–389, 1995. ISSN 0950-7051. doi : https://doi.org/10.1016/0950-7051(96)81920-4. Knowledge-based neural networks.

[13] R. Andrews, J. Diederich, and A. B. Tickle. Survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-based systems*, 8(6) :373–389, 1995.

[14] G. Ansanay-Alex. Estimating occupancy using indoor carbon dioxide concentrations only in an office building : a method and qualitative assessment. In *REHVA World Congress on Energy efficient, smart and healthy buildings (CLIMA)*, pages 1–8, 2013.

[15] S. Appliances. Smartm2m ; smart appliances ; reference ontology and onem2m mapping, 2017.

[16] A. D. Arbatli and H. L. Akin. Rule extraction from trained neural networks using genetic algorithms. *Nonlinear Analysis : Theory, Methods & Applications*, 30(3) :1639–1648, 1997.

[17] R. B. Baghli. *Approche sémantique de la conception de services connectés : cadre d'architecture, algorithmique de composition, application à la maison connectée*. PhD thesis, Télécom ParisTech, 2017.

[18] Z. Baida, J. Gordijn, and H. Akkermans. Service ontology. *Vrije Universität Amsterdam*, 2003.

[19] N. Balta-Ozkan, B. Boteler, and O. Amerighi. European smart home market development : Public views on technical and economic aspects across the united kingdom, germany and italy. *Energy Research & Social Science*, 3 :65–77, 2014.

[20] J. Benesty, J. Chen, Y. Huang, and I. Cohen. Pearson correlation coefficient. In *Noise reduction in speech processing*, pages 1–4. Springer, 2009.

[21] J. Š. Benth and F. E. Benth. A critical view on temperature modelling for application in weather derivatives markets. *Energy Economics*, 34(2) :592–602, 2012.

[22] H. Berlink and A. H. Costa. Batch reinforcement learning for smart home energy management. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[23] T. Berners-Lee and D. Connolly. Notation3 (n3) : A readable rdf syntax. https://www.w3.org/TeamSubmission/n3/.

[24] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific american*, 284(5) :34–43, 2001.

[25] S. K. Biswas, M. Chakraborty, B. Purkayastha, P. Roy, and D. M. Thounaojam. Rule extraction from training data using neural network. *International Journal on Artificial Intelligence Tools*, 26(03) :1750006, 2017.

[26] H. Boley, M. Dean, B. Grosof, M. Sintek, B. Spencer, S. Tabet, and G. Wagner. Fol ruleml : The first-order logic web language. https://www.w3.org/Submission/FOL-RuleML/.

[27] H. Boley, A. Paschke, and O. Shafiq. Ruleml 1.0 : the overarching specification of web rules. In *International Workshop on Rules and Rule Markup Languages for the Semantic Web*, pages 162–178. Springer, 2010.

[28] O. Brdiczka, J. L. Crowley, and P. Reignier. Learning situation models in a smart home. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 39(1) :56–63, 2008.

[29] H. Bride, J. Dong, J. S. Dong, and Z. Hóu. Towards dependable and explainable machine learning using automated reasoning. In *International Conference on Formal Engineering Methods*, pages 412–416. Springer, 2018.

[30] S. L. Brunton and J. N. Kutz. *Data-driven science and engineering : Machine learning, dynamical systems, and control*. Cambridge University Press, 2022.

[31] F. Burzlaff, M. Ackel, and C. Bartelt. A mapping language for iot device descriptions. 2019.

[32] R. Buyya, R. N. Calheiros, and A. V. Dastjerdi. *Big data : principles and paradigms*. Morgan Kaufmann, 2016.

[33] S. Çalışır and M. K. Pehlivanoğlu. Model-free reinforcement learning algorithms : A survey. In *2019 27th Signal Processing and Communications Applications Conference (SIU)*, pages 1–4. IEEE, 2019.

[34] D. Chaki and A. Bouguettaya. Fine-grained conflict detection of iot services. In *2020 IEEE International Conference on Services Computing (SCC)*, pages 321–328. IEEE, 2020.

[35] D. Chaki, A. Bouguettaya, and S. Mistry. A conflict detection framework for iot services in multi-resident smart homes. In *2020 IEEE International Conference on Web Services (ICWS)*, pages 224–231. IEEE, 2020.

[36] M. Chan, D. Estève, C. Escriba, and E. Campo. A review of smart homes—present state and future challenges. *Computer methods and programs in biomedicine*, 91(1) :55–81, 2008.

[37] M. Coggan. Exploration and exploitation in reinforcement learning. *Research supervised by Prof. Doina Precup, CRA-W DMP Project at McGill University*, 2004.

[38] T. Contributor. first-order logic. https://www.techtarget.com/whatis/definition/first-order-logic.

[39] M. W. Craven. *Extracting comprehensible models from trained neural networks*. The University of Wisconsin-Madison, 1996.

[40] M. W. Craven and J. W. Shavlik. Learning symbolic rules using artificial neural networks. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 73–80, 2014.

[41] A. Dasgupta and A. Nath. Classification of machine learning algorithms. *International Journal of Innovative Research in Advanced Engineering (IJIRAE)*, 3 (3) :6–11, 2016.

[42] H. Daumé. *A course in machine learning*. Hal Daumé III, 2017.

[43] P. Dayan, M. Sahani, and G. Deback. Unsupervised learning. *The MIT encyclopedia of the cognitive sciences*, pages 857–859, 1999.

[44] D. Delande, P. Stolf, R. Feraud, J.-M. Pierson, and A. Bottaro. Horizontal scaling in cloud using contextual bandits. In *Euro-Par 2021 : Parallel Processing : 27th International Conference on Parallel and Distributed Computing, Lisbon, Portugal, September 1–3, 2021, Proceedings*, pages 285–300. Springer, 2021.

[45] K. R. Dittrich, S. Gatziu, and A. Geppert. The active database management system manifesto : A rulebase of adbms features. In *Rules in Database Systems : Second International Workshop, RIDS'95 Glyfada, Athens, Greece, September 25–27, 1995 Proceedings 2*, pages 1–17. Springer, 1995.

[46] A. dos Santos Mignon and R. L. d. A. da Rocha. An adaptive implementation of ε-greedy in reinforcement learning. *Procedia Computer Science*, 109 :1146–1151, 2017.

[47] A. Fantechi, G. Gori, J. Parri, and S. Sampietro. Stingray project : Smart stations as hubs of infomobility services for smart cities. 02 2022.

[48] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau. An introduction to deep reinforcement learning. *arXiv preprint arXiv :1811.12560*, 2018.

[49] L. Fu. Rule learning by searching on adapted nets. In *AAAI*, volume 91, pages 590–595, 1991.

[50] L.-M. Fu. Knowledge-based connectionism for revising domain theories. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(1) :173–182, 1993.

[51] C. G. García, B. C. P. G-Bustelo, J. P. Espada, and G. Cueva-Fernandez. Midgar : Generation of heterogeneous objects interconnecting applications. a domain specific language proposal for internet of things scenarios. *Computer Networks*, 64 :143–158, 2014.

[52] C. Gershenson. Artificial neural networks for beginners. *arXiv preprint cs/0308031*, 2003.

[53] Z. Ghahramani. Unsupervised learning. In *Summer school on machine learning*, pages 72–112. Springer, 2003.

[54] S. Gu, E. Holly, T. P. Lillicrap, and S. Levine. Deep reinforcement learning for robotic manipulation. *arXiv preprint arXiv :1610.00633*, 1, 2016.

[55] D. Ha, A. Dai, and Q. V. Le. Hypernetworks. *arXiv preprint arXiv :1609.09106*, 2016.

[56] T. Hailesilassie. Rule extraction algorithm for deep neural networks : A review. *arXiv preprint arXiv :1610.05267*, 2016.

[57] D. A. Handelman, S. H. Lane, and J. J. Gelfand. Integrating neural networks and knowledge-based systems for intelligent robotic control. *IEEE Control Systems Magazine*, 10(3) :77–87, 1990.

[58] E. Hansen, A. Barto, and S. Zilberstein. Reinforcement learning for mixed open-loop and closed-loop control. *Advances in Neural Information Processing Systems*, 9, 1996.

[59] R. Harper. *Inside the smart home*. Springer Science & Business Media, 2006.

[60] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, I. Osband, et al. Deep q-learning from demonstrations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[61] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8) :1735–1780, 1997.

[62] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. Swrl : A semantic web rule language combining owl and ruleml. https://www.w3.org/Submission/SWRL/.

[63] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, M. Dean, et al. Swrl : A semantic web rule language combining owl and ruleml. *W3C Member submission*, 21(79) :1–31, 2004.

[64] E. Houzé. *A generic and adaptive approach to explainable AI in autonomic systems : the case of the smart home*. PhD thesis, Institut Polytechnique de Paris, 2022.

[65] Hugh Perkins. How to distinguish episodic task and continuous tasks ? https://stats.stackexchange.com/a/271358, 2017.

[66] J. Huysmans, B. Baesens, and J. Vanthienen. Using rule extraction to improve the comprehensibility of predictive models. 2006.

[67] S. Ilyas et al. The impact of revegetation on microclimate in coal mining areas in east kalimantan. *Journal of Environment and Earth Science*, 2(11) :90–97, 2012.

[68] A. Jena. Apache jena : A free and open source java framework for building semantic web and linked data applications. https://jena.apache.org/.

[69] A. S. Juraimi, M. Saiful, M. Begum, A. Anuar, and M. Azmi. Influence of flooding intensity and duration on rice growth and yield. *Pertanika J. Trop. Agric. Sci*, 32(2) :195–208, 2009.

[70] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning : A survey. *Journal of artificial intelligence research*, 4 :237–285, 1996.

[71] S. Kamruzzaman, M. Islam, et al. Extraction of symbolic rules from artificial neural networks. *arXiv preprint arXiv :1009.4570*, 2010.

[72] L. H. Karlsen. Description logic 1 : Syntax and semantics. https://www.uio.no/studier/emner/matnat/ifi/INF3170/h15/undervisningsmateriale/dl1.pdf.

[73] C. Kern, T. Klausch, and F. Kreuter. Tree-based machine learning methods for survey research. In *Survey research methods*, volume 13, page 73. NIH Public Access, 2019.

[74] J. M. K.W. Thoning, A.M. Crotwell. Atmospheric carbon dioxide dry air mole fractions from continuous measurements at mauna loa, hawaii, barrow, alaska, american samoa and south pole. https://doi.org/10.15138/yaf1-bk21, 2021.

[75] P. Langley et al. The changing science of machine learning. *Machine learning*, 82(3) :275–279, 2011.

[76] O. Lassila, F. van Harmelen, I. Horrocks, J. Hendler, and D. L. McGuinness. The semantic web and its languages. *IEEE Intelligent Systems and their Applications*, 15(6) :67–73, 2000.

[77] S. Lee and D.-H. Choi. Reinforcement learning-based energy management of smart home with rooftop solar photovoltaic system, energy storage system, and home appliances. *Sensors*, 19(18) :3937, 2019.

[78] S. Lek and Y. Park. Multilayer perceptron. In S. E. Jørgensen and B. D. Fath, editors, *Encyclopedia of Ecology*, pages 2455–2462. Academic Press, Oxford, 2008. ISBN 978-0-08-045405-4. doi : https://doi.org/10.1016/B978-008045405-4.00162-2.

[79] S. Li, L. D. Xu, and S. Zhao. The internet of things : a survey. *Information systems frontiers*, 17(2) :243–259, 2015.

[80] P. Lin, Q. Song, and Y. Wu. Fact checking in knowledge graphs with ontological subgraph patterns. *Data Science and Engineering*, 3(4) :341–358, 2018.

[81] C. P. Ling, N. M. M. Noor, F. Mohd, et al. Knowledge representation model for crime analysis. *Procedia computer science*, 116 :484–491, 2017.

[82] B. Liu. Supervised learning. In *Web data mining*, pages 63–132. Springer, 2011.

[83] H. Liu and R. Setiono. Chi2 : Feature selection and discretization of numeric attributes. In *Proceedings of 7th IEEE International Conference on Tools with Artificial Intelligence*, pages 388–391. IEEE, 1995.

[84] R. Liu and J. Zou. The effects of memory replay in reinforcement learning. In *2018 56th annual allerton conference on communication, control, and computing (Allerton)*, pages 478–485. IEEE, 2018.

[85] R. Lutolf. Smart home concept and the integration of energy meters into a home based system. In *Seventh international conference on metering apparatus and tariffs for electricity supply 1992*, pages 277–278. IET, 1992.

[86] M. Ma, S. M. Preum, W. Tarneberg, M. Ahmed, M. Ruiters, and J. Stankovic. Detection of runtime conflicts among services in smart cities. In *2016 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 1–10. IEEE, 2016.

[87] L. Mainetti, V. Mighali, L. Patrono, and P. Rametta. A novel rule-based semantic architecture for iot building automation systems. In *2015 23rd International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, pages 124–131. IEEE, 2015.

[88] S. N. Makhadmeh, A. T. Khader, M. A. Al-Betar, S. Naim, A. K. Abasi, and Z. A. A. Alyasseri. Optimization methods for power scheduling problems in smart home : Survey. *Renewable and Sustainable Energy Reviews*, 115 :109362, 2019.

[89] D. Marikyan, S. Papagiannidis, and E. Alamanos. A systematic review of the smart home literature : A user perspective. *Technological Forecasting and Social Change*, 138 :139–154, 2019.

[90] D. Martens, B. Baesens, and T. Van Gestel. Decompositional rule extraction from support vector machines by active learning. *IEEE Transactions on Knowledge and Data Engineering*, 21(2) :178–191, 2008.

[91] D. Martens, J. Huysmans, R. Setiono, J. Vanthienen, and B. Baesens. Rule extraction from support vector machines : an overview of issues and application

in credit scoring. *Rule extraction from support vector machines*, pages 33–63, 2008.

[92] D. L. McGuinness and F. van Harmelen. Owl web ontology language overview. https://www.w3.org/TR/owl-features/.

[93] D. L. McGuinness, F. Van Harmelen, et al. Owl web ontology language overview. *W3C recommendation*, 10(10) :2004, 2004.

[94] T. M. Moerland, J. Broekens, and C. M. Jonker. Model-based reinforcement learning : A survey. *arXiv preprint arXiv :2006.16712*, 2020.

[95] M. Mozer. *Lessons from an Adaptive Home*, pages 271 – 294. 01 2005. ISBN 9780471686590. doi : 10.1002/047168659X.ch12.

[96] M. C. Mozer. The neural network house : An environment hat adapts to its inhabitants. In *Proc. AAAI Spring Symp. Intelligent Environments*, volume 58, 1998.

[97] K. P. Murphy. *Machine learning : a probabilistic perspective*. MIT press, 2012.

[98] C. Nandi and M. D. Ernst. Automatic trigger generation for rule-based smart homes. In *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security*, pages 97–102, 2016.

[99] T. Nolle. service-oriented architecture (soa). https://www.techtarget.com/searchapparchitecture/definition/service-oriented-architecture-SOA#:~:text=Service%2Doriented%20architecture%20(SOA)%20is%20a%20software%20development%20model,to%20complete%20a%20specific%20task., 2020.

[100] ontology2.com. Forward chaining with the jena rules language. https://ontology2.com/the-book/jena-rule-language.html.

[101] N. W. Paton and O. Diaz. Active database systems. *ACM Computing Surveys (CSUR)*, 31(1) :63–103, 1999.

[102] A. Persily and L. de Jonge. Carbon dioxide generation rates for building occupants. *Indoor air*, 27(5) :868–879, 2017.

[103] M. Qiu, E. Najm, R. Sharrock, and B. Traverson. Pbre : A rule extraction method from trained neural networks designed for smart home services. In *International Conference on Database and Expert Systems Applications*, pages 158–173. Springer, 2022.

[104] M. Qiu, E. Najm, R. Sharrock, and B. Traverson. Reinforcement learning based architectures for dynamic generation of smart home services. In *2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 7–14, 2022. doi : 10.1109/ICMLA55696.2022.00010.

[105] T. Rashid, M. Samvelyan, C. Schroeder, G. Farquhar, J. Foerster, and S. Whiteson. Qmix : Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 4295–4304. PMLR, 2018.

[106] M. Ribeiro, K. Grolinger, and M. A. Capretz. Mlaas : Machine learning as a service. In *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, pages 896–902. IEEE, 2015.

[107] K. Rose, S. Eldridge, and L. Chapin. The internet of things : An overview. *The internet society (ISOC)*, 80 :1–50, 2015.

[108] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[109] A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani. Deep reinforcement learning framework for autonomous driving. *arXiv preprint arXiv :1704.02532*, 2017.

[110] R. Setiono and H. Liu. Neurolinear : From neural networks to oblique decision rules. *Neurocomputing*, 17(1) :1–24, 1997.

[111] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. Pellet : A practical owl-dl reasoner. *Journal of Web Semantics*, 5(2) :51–53, 2007.

[112] R. M. Smullyan. *First-order logic*. Courier Corporation, 1995.

[113] B. K. Sovacool and D. D. F. Del Rio. Smart home technologies in europe : A critical review of concepts, benefits, risks and policies. *Renewable and sustainable energy reviews*, 120 :109663, 2020.

[114] H. Strese, U. Seidel, T. Knape, and A. Botthof. Smart home in deutschland. *Institut für Innovation und Technik (iit)*, 46 :13, 2010.

[115] Y. Sun, X. Wang, H. Luo, and X. Li. Conflict detection scheme based on formal rule model for smart building systems. *IEEE Transactions on Human-Machine Systems*, 45(2) :215–227, 2014.

[116] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv :1706.05296*, 2017.

[117] R. S. Sutton and A. G. Barto. Reinforcement learning : An introduction. chapter 5-6, pages 91–240. Cambridge, MA : MIT Press, 2018.

[118] I. A. Taha and J. Ghosh. Symbolic interpretation of artificial neural networks. *IEEE Transactions on knowledge and data engineering*, 11(3) :448–463, 1999.

[119] M. M. Taye. Understanding semantic web and ontologies : Theory and applications. *arXiv preprint arXiv :1006.4567*, 2010.

[120] B. J. Taylor and M. A. Darrah. Rule extraction as a formal method for the verification and validation of neural networks. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 5, pages 2915–2920. IEEE, 2005.

[121] M. Tokic and G. Palm. Value-difference based exploration : adaptive control between epsilon-greedy and softmax. In *Annual conference on artificial intelligence*, pages 335–346. Springer, 2011.

[122] G. G. Towell. Symbolic knowledge and neural networks : Insertion, refinement and extraction. 1993.

[123] G. G. Towell and J. W. Shavlik. Extracting refined rules from knowledge-based neural networks. *Machine learning*, 13(1) :71–101, 1993.

[124] H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

[125] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv :1511.06581*, 2015.

[126] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3) :279–292, May 1992. ISSN 1573-0565. doi : 10.1007/BF00992698.

[127] B. Winkler. *An implementation of an ultrasonic indoor tracking system supporting the OSGI architecture of the ICTA lab*. PhD thesis, University of Florida, 2002.

[128] M. Woźniak and D. Połap. Intelligent home systems for ubiquitous user support by using neural networks and rule-based approach. *IEEE Transactions on Industrial Informatics*, 16(4) :2651–2658, 2019.

[129] X. Xu, Y. Jia, Y. Xu, Z. Xu, S. Chai, and C. S. Lai. A multi-agent reinforcement learning-based data-driven method for home energy management. *IEEE Transactions on Smart Grid*, 11(4) :3201–3211, 2020.

[130] R. Yang and M. W. Newman. Learning from a learning thermostat : lessons for intelligent systems for the home. In *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing*, pages 93–102, 2013.

[131] L. Yu, W. Xie, D. Xie, Y. Zou, D. Zhang, Z. Sun, L. Zhang, Y. Zhang, and T. Jiang. Deep reinforcement learning for smart home energy management. *IEEE Internet of Things Journal*, 7(4) :2751–2762, 2019.

[132] J. Zhang, W. Zhao, G. Xie, and H. Chen. Ontology-based knowledge management system and application. *Procedia Engineering*, 15 :1021–1029, 2011.

[133] K. Zhang, Z. Yang, and T. Başar. Multi-agent reinforcement learning : A selective overview of theories and algorithms. *Handbook of Reinforcement Learning and Control*, pages 321–384, 2021.

[134] Y. Zhang, G. Tian, S. Zhang, and C. Li. A knowledge-based approach for multiagent collaboration in smart home : From activity recognition to guidance service. *IEEE Transactions on Instrumentation and Measurement*, 69(2) :317–329, 2019.

[135] Z.-H. Zhou. Rule extraction : Using neural networks or for neural networks ? *Journal of Computer Science and Technology*, 19(2) :249–253, 2004.

ECOLE
DOCTORALE

Titre : Conception de services pour maison intelligente à l'aide d'approches basées sur l'apprentissage automatique et la représentation de connaissances

Mots clés : maison intelligente, conception de services, apprentissage par renforcement, représentation de connaissance et raisonnement

Résumé : L'intelligence de la maison intelligente est réalisée en créant divers services. Chaque service tente d'ajuster un état monitoré en contrôlant les actionneurs associés après avoir pris en compte les états de l'environnement détectés par les capteurs. Cependant, la conception de la logique des services déployés dans une maison intelligente se heurte à des limitations soit d'adaptabilité dynamique (règles prédéfinies) soit d'explicabilité (techniques d'apprentissage). Quatre propositions s'inscrivant dans une approche hybride combinant des règles prédéfinies et des techniques d'apprentissage visent à lever ces limitations.
La première proposition consiste à utiliser l'apprentissage renforcé pour créer un service dynamique. Le déploiement de ce service unique comprend deux phases : le pré-entraînement dans la simulation et l'entraînement continu dans le monde réel. Notre étude se concentre uniquement sur la partie simulation. En étendant la première proposition, la deuxième proposition propose plusieurs architectures pour créer plusieurs services dynamiques et sans conflit. Cependant, les services dirigés par les données ne sont pas explicables. Par conséquent, la troisième proposition vise à extraire des services explicables basés sur la connaissance à partir de services dynamiques dirigés par les données. La quatrième proposition tente de combiner les deuxième et troisième propositions pour créer des services dynamiques et explicables. Ces propositions sont évaluées dans un environnement simulé sur des services de contrôle de la température, de l'intensité lumineuse et de la qualité de l'air adaptés aux activités de l'habitant. Elles peuvent être étendues selon plusieurs perspectives, telles que la co-simulation de phénomènes physiques, l'adaptation dynamique à différents profils d'habitant, et l'efficacité énergétique des services déployés.

Title : Designing smart home services using machine learning and knowledge-based approaches

Keywords : smart home, service design, reinforcement learning, knowledge representation and reasoning

Abstract : The intelligence of a smart home is realized by creating various services. Each service tries to adjust one monitored state by controlling related actuators after considering environment states detected by sensors. However, the design of the logic of the services deployed in a smart home faces limitations of either dynamic adaptability (predefined rules) or explicability (learning techniques). Four proposals that are parts of a hybrid approach combining predefined rules and learning techniques aim at mitigating these limitations.
The first proposal is to use reinforcement learning to create a dynamic service. The deployment of this single service includes two phases : pretraining in the simulation and continuous training in the real world. Our study only focuses on the simulation part. Extending the first proposal, the second proposal proposes several architectures to create multiple dynamic and conflict-free services. However, the created data-driven services are not explicable. Therefore, the third proposal aims to extract explicable knowledge-based services from dynamic data-driven services. The fourth proposal attempts to combine the second and third proposals to create dynamic and explicable services. These proposals are evaluated in a simulated environment on temperature control, light intensity, and air quality services adapted to the activities of the inhabitant. They can be extended according to several perspectives, such as the co-simulation of physical phenomena, the dynamic adaptation to various inhabitant profiles, and the energy efficiency of the deployed services.

Institut Polytechnique de Paris
91120 Palaiseau, France