博 士 論 文

A Programming Framework for Automatic Management of IoT-enabled Smart Buildings

（IoT-enabled スマートビルの自動管理のためのプログラミングフレームワーク）

彭　晓　晖

(Peng Xiaohui)

ABSTRACT

In recent times, with the development of advanced technologies, the connectivity of devices has significantly improved in smart buildings. Remote data access and device control are enabled through open RESTful interfaces and the application paradigm has changed as a result of interface unification. Further, the smart building has become a programmable architecture that can now be regarded as a distributed-hardware computer. Consequently, residents can now be regarded as users who will become a constant part of the smart environment in the near future.

Meanwhile, improving energy efficiency is a major issue in building automation. Energy efficiency involves several sub-factors, such as energy consumption, human comfort, and productivity. The factor(s) that needs to be addressed first depends on the specific real-time context. Building automation applications usually have to make compromises among these factors. Therefore, it is important to design a programming environment for such a "computer" to perform personalized, precise, and collaborative control as we did in classic computers for smart buildings. However, these users are usually ordinary people who are not good at programming or may not be familiar with the target building. Describing available services and providing an easier programming environment for these users is a significant challenge.

State-of-the-art protocol suites such as 6LoWPAN and CoAP connect constrained devices to the Internet. With RESTful open APIs, the heterogeneous networks in smart buildings have become homogeneous, resulting in such a building becoming a programmable architecture. Further, devices from different subsystems now cooperate very easily. However, a unified lightweight distributed programming framework for personalized, precise, flexible, and collaborative control of smart buildings is needed. Consequently, we propose a programming framework for automatic management of smart buildings that addresses these issues. This framework abstracts and hides lower-layer building structure and service details and provides descriptive automatic management languages for smart building users such as building administrators, IT managers, facility managers, and developers. A building resource description schema is also provided to enable these users to refer to services when writing management policies using proposed descriptive languages.

We evaluated the proposed framework via field experiments in the Daiwa Ubiquitous Computing Research Buildings (DUCRB). Several energy management policies were

written using the proposed framework. Prototypes of supporting tools were also implemented to provide an efficient programming environment for smart buildings. The results demonstrate that the proposed framework enables the users to develop efficient automatic management applications in IoT-enabled smart buildings.

The main contributions of this research include two descriptive languages for automatic management of IoT-enabled smart buildings, a smart building resource representation schema for describing the available resources in a building, and tools such as compiler, interpreter, and programming language editor to simplify the programming process for smart building users.

ACKNOWLEDGMENTS

First and foremost, I would like to express my special appreciation and thanks to my supervisor, Professor Ken Sakamura, Graduate School of Interdisciplinary Information Studies, The University of Tokyo, for the valuable guidance, motivation, and consistent encouragement I received throughout this research. I was constantly motivated by the joy and enthusiasm he has for his research, even during difficult times in my Ph.D. study. I will always remember the way he meticulously reviewed my papers. His advice on both research and my career has been priceless. Having him as my Ph.D. supervisor has been an honor, and he is the best supervisor I have ever met. It is impossible for me to achieve this dissertation without his supervision.

I would also like to thank Professor Noboru Koshizuka, Graduate School of Interdisciplinary Information Studies, The University of Tokyo, for his patience and careful guidance during my Ph.D. study. He worked as co-supervisor during my Ph.D. study and provided substantial support to my research. As a person with an amicable and positive disposition, he has always made himself available to clarify my doubts, although he has a very busy schedule every day. This thesis would likely not have been possible without the unconditional support he gave to me.

I also gratefully acknowledge the daily support and collaboration for my research received from Dr. Masahiro Bessho. We discussed research issues at the weekly meetings and I received a lot of advice from him at various phases in this research. I appreciate his enthusiasm, willingness, and patience very much during consultations on my research papers. Many thanks to present and past members of the Sakamura-Koshizuka laboratory—especially Dr. Takeshi Yashiro and Dr. Fahim Khan, who gave me many insightful suggestions and comments during this research.

I am especially grateful for the funding received from The Graduate Program for Social ICT Global Creative Leaders (GCL) at The University of Tokyo. I also obtained many novel ideas that proved helpful in my research from seminars and workshops held by GCL. Many thanks also go out to faculty members of our laboratory and department for the support I received during my Ph.D. study.

Finally, I would also like to thank the members of my Ph.D. committee for their invaluable comments and feedback on the preliminary version of this thesis.

Contents

LIST OF TABLES

LIST OF FIGURES

List of Appendices

1 INTRODUCTION

In this chapter, we look at the history of building automation in brief and then analyze building management system architecture. Some research issues are discussed in the second section of this chapter. Finally, we present the objective of our research and discuss the main challenges to realize that objective.

1.1 Background

1.1.1 Brief History of Building Automation

The advent of Building Automation Systems (BASs) can be traced back to the 1960s when the centralized computer systems were first used for building control [1]. They evolved from industrial process control systems with the deployment of "computer-controlled systems in the late 1960s" [1]. The energy crisis in the 1970s inspired researchers to study intelligent control algorithms for lighting and HVAC systems to address the energy efficiency issue. Minicomputers and programmable logic controllers (PLCs) were progressively adopted in BASs at that time [2] for automatically managing building services. The main focus was reduction of the energy consumption of buildings.

With the introduction of Direct Digital Control (DDC) [3] into building automation in

the 1980s, the term "Intelligent Building" was coined. Energy control technologies subsequently became progressively sophisticated, integrated energy management systems appeared, and researchers began to study the feedback approach for advanced energy reservation [4]. The feedback approach informs users in real-time of the energy consumption in buildings to affect positive behavior changes. Initial studies, primarily conducted by psychologists, showed that energy could potentially be reserved by introducing a feedback approach [4]. For example, Pallack et al. studied how consumption feedback information affected residents' behavior in reducing energy consumption [5] within residential buildings in 1980.

In the 1990s, environmental parameters were introduced to control lighting and Heating, Ventilating, and Air Conditioning (HVAC) [6] systems. Many control algorithms (e.g., Proportional-Integral-Derivative Control, Model-Predictive Control, Fuzzy Control, and Adaptive Control) [7]-[14] were also studied with a view to develop efficient facility controllers. However, those algorithms depend on environmental inputs, which are a set of real-time and detailed energy-related information. This was a major challenge for the researchers at that time. Many energy measurement approaches were studied; these approaches were divided into two categories: "single-point monitoring (NILM) and distributed monitoring (ILM)" [4], [15]-[16].

The adoption of Wireless Sensor Networks (WSNs) significantly improved information harvesting from smart buildings in the 2000s [4]. They enabled exhaustive inside information to be collected by wireless sensors deployed in the buildings [17]. The information collected primarily included environmental parameters (e.g., temperature, humidity, and $PM_{2.5}$ (i.e., particulate matter with a diameter of less than 2.5 micrometers)[18]), various energy consumption levels (e.g., building level, appliance level, and user level), and human-related information (e.g., activities and preferences). Thus, flexible and detailed control could be performed based on the harvested exhaustive context information. Stand-alone facility controllers and subsystems became sophisticated as a result of increased computer power and monitoring tools and technologies. Feedback systems were further studied to motivate users to save energy. Data communication protocols such as BACnet [19] and LonTalk [20] were

progressively developed, and significantly improved data sharing among subsystems. However, the subsystems were still isolated information islands and devices in different systems could not operate in a collaborative way to adapt to human activities. As a result, they could not optimize the overall energy efficiency of the building.

Technological advancements in the era of the Internet of Things (IoT) (e.g., device connectivity, big data analytics, and open API [21]-[23]) facilitate the collection and analysis of detailed information related to energy consumption in a building [4]. Further, improved connectivity enables web-based access to collected data [24]-[28] and control of appliances [29], [30] through open APIs. Hence, data acquisition services (e.g., energy consumption, environmental conditions, and human activities) and appliance control services are integrated into building management systems, and developing energy control applications across different subsystems to make devices operate in a collaborative manner is becoming possible.

Thus, the application programming paradigm has changed (see Figure 1-2), and smart buildings have become “Highly Functionally Distributed Systems” [31] as energy sensing and control have become ubiquitous through unified web-based services. Consequently, the overall building as a whole can be regarded as a programmable architecture when designing building automation applications, instead of focusing on designing the facility controller for an individual device or a sub-system. All energy-efficiency related factors (e.g., energy consumption, environmental conditions, and human activities) should be considered and energy control policies instituted.

In other words, the overall building is considered a distributed-hardware computer and embedded real-time operating systems inside the smart devices constitute a huge distributed operating system [32] for smart buildings. Users are living in this operating system and they need to interact with distributed components (smart devices). We will study how different kinds of users can be enabled to program with IoT-enabled smart buildings by providing simpler descriptive programming languages and resource representation schema to support the proposed programming architecture.

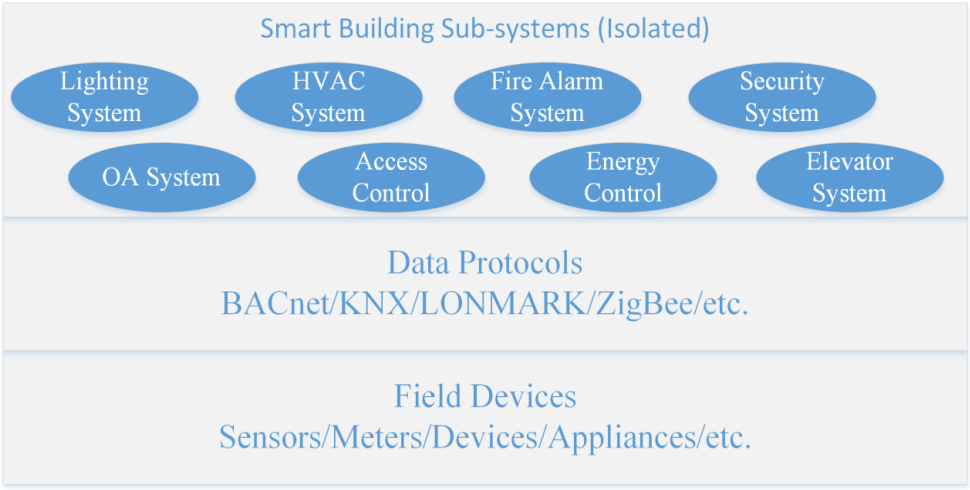


Figure 1-1. Isolated subsystems; devices in different systems cannot work in a collaborative way

1.1.2 Building Automation System

In this section, we introduce basic concepts associated with BAS, which is also sometimes referred to as Building Management System (BMS) or Building Energy Management System (BEMS). A BAS usually consists of many subsystems such as HVAC, lighting system, sensor network, metering system, appliances, and security system. A classic BAS conceptually follows a three-layer approach [4], [24]:

Field Layer (sensors and actuators)

This layer mainly consists of sensors, meters, appliances, etc. It is the physical layer that interacts with the environment and human beings inside buildings. These devices collect information called context from the building; these context serve as inputs for upper-layer algorithms and applications. For example,

- environmental conditions (e.g., temperature, humidity, and lighting);
- energy consumption at various levels; and
- human-related information (e.g., user preference and activities).

Such exhaustive detailed contexts provide a chance for fine-grained control in smart buildings. Actuation devices such as air conditioners, lights, and ventilators have autonomous controls [4], [33] based on contexts acquired from the environment. They accept commands and decision sent from upper-layer applications.

Integration Layer

This layer collects and stores raw data from the real world, and then performs analysis based on the gathered data to provide more abstract and meaningful data or interface for the upper layer. For example, by analyzing raw data from human sensors, we can ascertain the user distribution status along floors, and stop available elevators at the floors where there are more users in the building. Applying such rules to an elevator system will inevitably improve human comfort and productivity because it saves time for most users. We can regard this layer as the integration and abstraction layer for devices deployed in the building.

Application Layer

The application layer is the central part of the BAS [4], [33]. It acquires information from the environment and makes decisions based on the obtained information to adjust environmental parameters and device status; thereby, reducing energy consumption, improving human comfort and productivity, and enhancing the security of buildings. As shown in Figure 1-1, the functionality of a BAS is realized in this layer.

1.1.3 IoT-enabled Smart Buildings

The primary devices that comprised the Internet about 10 years ago were desktops and laptops. Things changed rapidly as other computing devices (e.g., consumer devices and industrial machines) were gradually connected to the Internet. In particular, after 2012, the number of connected machines and other devices increased exponentially and networks became homogeneous [34]. Facilitated by protocols such as 6LoWPAN [35] and CoAP [36] that were designed to enable resource constraint devices connected as the Internet does, data access and device control have become easy through open RESTful [37], [38] APIs. The area of smart buildings is a leading area adapting such evolution. This new paradigm is called IoT-enabled smart buildings (see Figure 1-2).

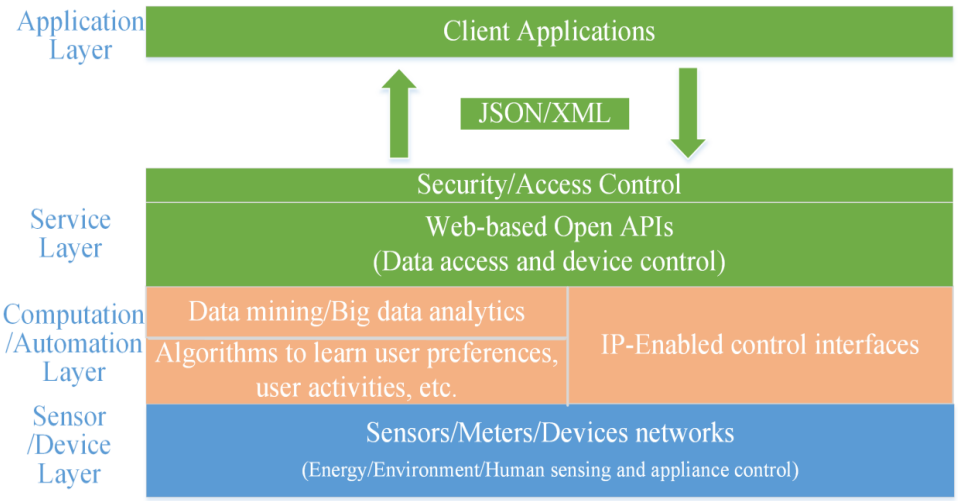


Figure 1-2. The new programming paradigm of IoT-enabled smart buildings

1.1.4 Descriptive Programming Languages

We define descriptive programming languages as the converse of procedural programming languages such as C, Java, and PHP. In this research, descriptive programming languages are usually high-level languages for non-expert users to describe control logics for the smart building systems. For example, we designed a device control rule definition language based on XML format to enable users to describe fine-grained rules for smart devices. Visual programming languages are also classified as descriptive programming languages in our research. Such languages enable users to create programs by "operating graphical program elements rather than by typing text" [39].

Owing to heterogeneous networks and the structures of smart buildings, it is quite difficult for non-expert users to program them with procedural programming languages. For example, a rule that turns off the air conditioners in room A when the temperature in this room exceeds 26 ℃ requires the programmer knowing the following:

- the number of air conditioners in room A;
- how to turn off each air conditioner; and
- how to ascertain the temperature of room A.

A domain expert can read professional documents of related techniques to get the above

necessary information to create automation applications. However, as various users will be a constant part of smart environments in the near future, a simpler descriptive programming language is needed for non-expert users to program smart buildings.

1.2 Problems

1.2.1 Energy Efficiency

Building automation issues include “reducing energy consumption, improving human comfort, productivity, safety, and security” [40]. Our definition of energy efficiency in this article includes all these issues except safety and security. Improving the energy efficiency of smart buildings is always complicated because of the complexity of smart buildings and the energy efficiency issue itself.

Many building automation applications in which intelligent algorithms [33], [41]-[44] are used to control actuator systems have been exploited by researchers in the literature. Because of the heterogeneous nature of networks in smart buildings, these applications usually can be developed only by domain experts associated with the specific vendor of an individual device or sub-system. The building system should function in a more collaborative manner for better energy efficiency [45].

Energy efficiency consists of several sub-factors, which we have already discussed at the beginning of this section. The factor that should be addressed first depends on the real-time context, such as environment parameters, human activities, and energy consumption. For example, there may be a need to reduce energy consumption first on a particular occasion, whereas improvement of human comfort may take priority on another occasion. The building automation system usually makes a compromise between these factors. To realize context-based control, users should be empowered to program smart buildings by enabling them to write control policies for context-related devices [17], [45].

1.2.2 Collaborative Control

In the beginning, building subsystems were built separately using independent

technologies, devices, and software. Consequently, subsystems were isolated. The integration of data platforms significantly improved information sharing by the subsystems over the past decades. Networked sensors and meters can provide consistent information for applications in different subsystems. However, subsystems are still isolated from the perspective of control. Collaboration control [46] among devices from different subsystems is therefore needed.

For instance, when the air conditioners in a room are on, the ventilating devices should be turned down and windows should be closed by the building automation system. In this scenario, air conditioning system, ventilating system, and the window control system are involved, but they act independently. A framework with which users can write a rule to control these systems to enable them to operate in a collaborative manner is needed. The rule may be, “When air conditioners in room A are on, then turn off ventilating devices and close the windows in this room.”

When smart buildings, homes, hospitals, and schools are connected to form a smart city, more sophisticated smart applications can be developed. For example, the security system of your home may be able to tell you when your child’s class is over; the kitchen system may check the refrigerator, compile a shopping list, and send it to the supermarket. Then, robots in the supermarket could prepare, pack, and send the ordered groceries to a pilotless automobile or an unmanned aerial vehicle. Your smart watch could also analyze your health status and suggest a menu for your dinner. If you agreed to the suggested menu then the smart kitchen system would prepare your dinner before you arrived home. These scenarios need collaborative control over the subsystems of different buildings in the smart city. Newly developed protocols such as 6LoWPAN and CoAP enable smart devices to expose their data and functionalities through web-based RESTful API, resulting in data access and device control becoming ubiquitous. However, a lightweight distributed framework that enables users to develop smart applications is needed for those scenarios.

1.2.3 Summary

Energy efficiency is the main concern of BASs. We have discussed this issue in

section 1.2.1. It involves a set of sub-factors such as reducing energy consumption, improving human comfort, and improving productivity. To optimize energy efficiency, personalized control should be possible [47], as it would enable users to control the building according to their preferences [48]. As more sensing devices are connected to building networks, detailed information about energy consumption (e.g., consumption by appliances, rooms, and building levels), human activities, and environmental parameters can be obtained from the building system. Precise and flexible control is necessary to perform optimization on energy efficiency based on sensed fine-grained information. Achieving such goals "remains a challenge mainly due to the lack of unified frameworks which can support the heterogeneous devices and services" of smart buildings [48].

Data integration has significantly improved information sharing among the subsystems of t buildings. However, from the perspective of control subsystems are still isolated. Working collaboratively, authentication, positioning, elevator, and HVAC systems can optimize the energy efficiency of smart buildings. State-of-the-art protocol suites such as 6LoWPAN and CoAP connect constrained devices to the Internet just like a computer. With RESTful open APIs, the heterogeneous networks of smart buildings have become homogeneous and cooperation by devices from different subsystems has become easy. A unified lightweight distributed programming framework for personalized, precise, flexible, and collaborative control of smart buildings is needed.

1.3 Objective

The ultimate objective of this research is to enable smart building users to program with IoT-enabled smart buildings. Thus, personalized, precise and collaborative context-based control can be performed by these users and energy efficiency will be finally optimized. We realize our goal by abstracting building resources and providing simpler programming languages [32], [49], [50] for smart building users to program the building easily.

The overall building can be regarded as a distributed-hardware computer [30], with the deployed smart devices as the distributed-hardware components of this computer.

Embedded operating systems in various devices constitute the building operating system. To realize our objective, we have to abstract distributed hardware resources and provide simple programming environment for common users to interact with smart devices in smart buildings. Dawson-Haggerty et al. proposed “a building operating system services for large commercial buildings” [32] called BOSS in 2013. They developed “a collection of services forming a distributed operating system, which solved several key issues that had prevented earlier systems from scaling across the building range” [32]. However, BOSS does not provide any mechanism for upper-layer users to develop automation applications easily and it is still a conceptual system.

1.4 Challenges

Two main challenges are dealt with in this research. The first challenge is resource representation and the second one is simpler programming languages. We explain them in the ensuing sections.

1.4.1 Resource Abstraction and Management

In the smart building system, sensors, actuators, appliances and other machines are connected into separate networks using different hardware and software protocols, which forms a heterogeneous building network structure [33]. Providing unified and easy access to these distributed resources is a significant challenge. For example, if an ordinary user wants to obtain the temperature value of a room through a smart building API, he/she has to know the API’s name and parameters. In this case, the relevant parameters may be the room number or ID specified by facility managers or system developers. Both API name and parameter information are difficult for an ordinary user to obtain and understand. Thus, a semantic self-describable representation for the heterogeneous distributed resource that the upper-layer users can use to easily refer to them is needed. The term semantic refers to the relationship representation among spaces, objects, etc. Self-describable refers to the means by which a service is represented (here, we refer to a RESTful API). Services should be self-describable so that they can be modeled and translated into executable code directly by a compiler or

an interpreter.

1.4.2 Simpler Programming Languages

We stated in section 1.2 that we need to enable users in IoT-enabled smart buildings to write rules for a specific context to get fine-grained control for energy efficiency. Such fine-grained control enables devices in different subsystems to operate in a collaborative manner to adapt to human activities. These controls share a common format: when some situation/context appears, then suitable actions should be applied to related devices [46]. We call this kind of control context-based control [45]. An automation application can be constituted by multiples of such kinds of rules to perform a particular goal. Considering that improving energy efficiency involves a trade-off among factors such as energy consumption, human comfort, and productivity, fine-grained personalized controls has significant potential for improving energy efficiency [33] in smart buildings.

However, the programming languages that are currently widely used, such as C/C++ and Java, are mainly for skilled programmers and are complicated for context-based controls. Users in building automation control systems may only need to write simple controls based on “IF-THEN” logic [50]. Thus, semantic and visual programming languages are more suitable for them. Therefore, we must provide concise and simpler programming languages for users in IoT-enabled smart buildings. This issue is discussed in-depth in section 2.5.

1.5 Summary

In this chapter, we presented the history of building automation in brief. A general architecture was presented and illustrated in section 1.1.2, and we defined the term IoT-enabled smart buildings in section 1.1.3. Two major features of IoT-enabled smart buildings are as follows:

- Inside devices are connected in a manner similar to connections on the Internet; that is, resource constrained devices are able to connect to the Internet using Internet-like protocols (such as 6LoWPAN and CoAP).

- Unified web-based RESTful data access and device control interface.

The adoption of Internet-like and Web-like protocols in smart buildings significantly eases the application development process and connects smart buildings to the Web of Things smoothly. We discussed the energy efficiency problem in section 1.2 and stated that energy efficiency consists of several sub-factors such as energy consumption, human comfort, and productivity. We also stated that the factor(s) that should be addressed first depends on the context. In most cases, a trade-off has to be made among these factors. We introduced the general and ultimate goal of this research in section 1.3. Finally, we discussed the challenges that have to be overcome in order to realize our objective.

1.6 Outline

This dissertation proceeds as follows:

Chapter 1 presents background information and defines some of the terms used in this dissertation. The general goal and challenges are also discussed.

Chapter 2 reviews related work. The paradigm shift from smart buildings and web technologies used in building systems is first discussed. Then, work related to resource representation and programming approaches is surveyed.

Chapter 3 introduces our proposed system along with general principles and terms.

Chapter 4 presents the design details of the proposed framework.

Chapter 5 outlines the implementation of each component.

Chapter 6 discusses the evaluation conducted of each component.

Chapter 7 discusses various issues we plan to address in the future.

Chapter 8 summarizes this dissertation and presents the results of our research.

2 RELATED WORK

2.1 Paradigm Shift in Smart Buildings

In the beginning, most researchers in the building automation field focused on designing facility controllers [7]-[14], [51] for individual devices and subsystems such as lighting and HVAC systems. These devices and systems were isolated from each other, and sensed and metered data could not be shared among them [4]. These isolated information islands constituted a vertical building automation system. Functional reduplicated sensing or metering devices were frequently deployed for different subsystems. However, this led to issues associated with redundancy and collaboration. For example, the temperature value of a place may be retrieved from two different devices, but it was not clear which value should be used for decision making.

Over the past decades, engineers have built BASs using disparate communication protocols [52] such as BACnet [19] and KNX [53]. This change improved data sharing and reduced information redundancy. As shown in Figure 1-1, sensing and metering became a uniform and integrated layer that supplied consistent environmental data. Further, collected raw data could be analyzed to provide a meaningful abstraction for the application layer. However, applications of the subsystems were developed independently. They optimized environmental parameters solely, which caused conflicting controls. From the perspective of control, these subsystems were still

isolated. A programming framework that regards the whole building as a programmable architecture is therefore needed, with applications running in such a framework enabling devices in different subsystems to operate in a collaborative manner. This is useful because a specific context may require that all related devices work together to adapt to human activities or environmental changes. This problem is solved by providing unified data access and device control interfaces for the overall system of smart buildings.

2.2 Unified Interface for Smart Buildings

Over the past decades, researchers and developers have become focused on the integration of sensing and metering networks [4]. This integration provides analyzed and semantic data for the application layer. The rise of IoT has resulted in significant changes in smart buildings and homes. Web technologies are now being applied to acquire information from building systems and many appliance manufacturers are starting to adopt HTTP-based controls for their products. For example, DAIKIN [54] provides a web-accessible controller (iTouch Controller) [55] by which users can control connected air conditioners through the HTTP protocol. Along with a campaign of open data, developers have started to issue RESTful Open API [21], which allows users to acquire data and control devices. This is the so-called paradigm of Web of Things (WoT) [56], which is regarded as the application layer of IoT. It will finally change the programming pattern of smart buildings to a new paradigm, as shown in Figure 1-2.

2.2.1 Web-based Sensing

Many proposals have been regarding integration of sensor networks with existing IT systems using web services [23]-[28]. Such integration has been used to acquire energy-related context (e.g., energy consumption, environmental parameters, and human activities) from the environment. In this section, we discuss proposals made for energy sensing in smart buildings.

Weiss et al. proposed an energy monitoring system using web-enabled power outlets [26] in 2010. Their objective was to study user awareness of energy consumption, so

their system did not include any control component. They measured device-level energy consumption using a smart power outlet called Plogg, which consisted of an electricity plug and a consumption logger. Bluetooth [57] and Zigbee [58] interfaces were also provided to retrieve logged consumption data. The devices were plugged into Plogg sensor nodes, which were connected to a gateway. Their system also included a micro-web server that "monitored the functionalities of the Ploggs as structured URLs, in a RESTful manner" [26]. Their work illustrated that constrained devices can be seamlessly integrated into the existing Internet and resources can be easily accessed via the RESTful approach. Agarwal et al. developed "an extensible and distributed data storage system called" BuildingDepot [59] for smart building management systems. They primarily focused on the design of the API and provided various RESTful APIs for access to the stored data.

2.2.2 Web-based Access and Control Interface

The technology advancements in the fields of IoT and WoT may significantly boost technology advancements in smart buildings and some researchers have already tried to integrate web-based data access and device control [29], [30], [60]-[66] in smart buildings or homes. Kamilaris et al. proposed the building of energy-aware smart homes using web technologies [29]. They developed a conceptual system for home automation that was built using web technologies to examine and evaluate their proposed framework. Their framework provided a "Presentation Layer" for the building systems, which "dynamically generated a representation of the deployed devices and their services to the web" [60] for public access. It enabled uniform interaction with deployed devices through RESTful interfaces. Their study is one of the initial studies that attempted to integrate devices in smart homes and buildings with the Web of Things, and expose the functionalities of these devices through RESTful interfaces.

A state-of-the-art smart building called the Daiwa Ubiquitous Computing Research Building (DUCRB) [30] was designed and built in 2014. Adopting newly emerged IoT technologies, DUCRB provides web-based information acquisition and device control interfaces called Smart Building APIs (In fact, they are a collection of RESTful APIs).

To acquire context information and to control devices, users only need to send HTTP GET, PUT, and POST requests to the API servers. Response data are returned in JSON format if necessary [67]. The inhabitants in DUCRB can use the smart building API to get fine-grained context and to control inside connected appliance from any devices that connected to the control network. Therefore, real-time context-based (e.g., environmental parameters, user activities and electricity consumptions) management of the building system is enabled and becomes easy [17], [45].

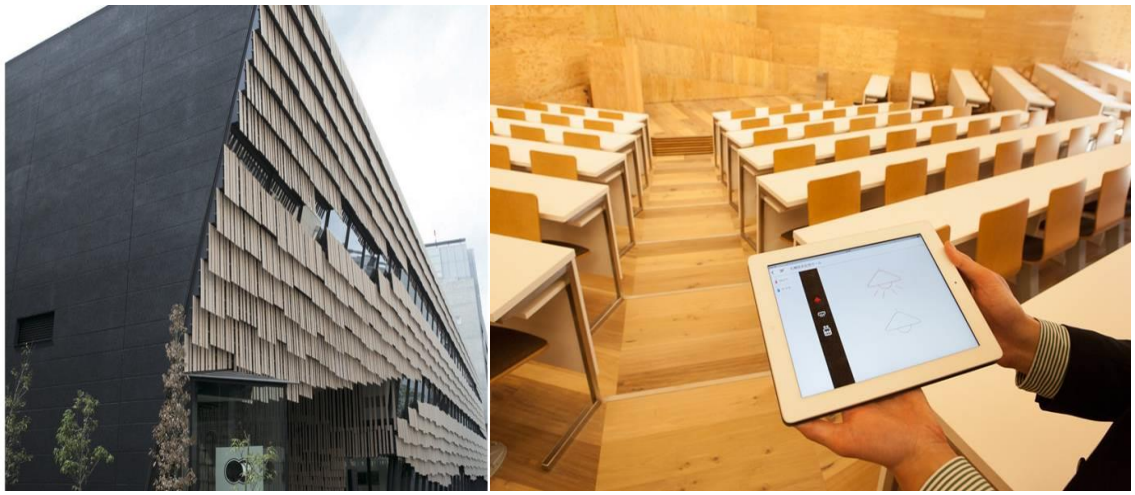


Figure 2-1. An IoT-enabled Smart Building-DUCRB [30]

DUCRB is an experimental environment for students studying in the applied computer science field of our department. By providing open web-based APIs, it enables rapid prototyping of smart building applications such as energy management, security, and safety, indoor navigation for the visually-impaired, etc. All the evaluations carried out in this research were performed in this building.

With the evolution of IoT and its adoption of WoT [56] as its application layer, sensors and appliances are able to access the web, which simplifies the development of user-side applications. Smart buildings and homes are following this paradigm, which shows the potential to enable ordinary users to program physical objects inside buildings and homes. However, to achieve this goal, a unified programming framework that hides lower-level complex building structures and services is needed so that

ordinary users can easily refer to services and control inside devices.

2.3 Rule-based Expert Systems for Building Management

2.3.1 Expert Systems

Expert systems are used extensively in artificial intelligence applications. “An expert system is a system that emulates the decision-making ability of a human expert” [68]. It usually comprises a database to store rules and facts, a reasoning engine, and user interfaces. A rule-based expert system is generally represented using logic rules (i.e., IF-THEN rule) rather than code written in procedural programming languages such as C and Java. The essential goal of an expert system is that “stores rules and facts in the knowledge base and uses the inference engine to find new facts by applying the rules to the known facts” [68].

2.3.2 Existing Studies

Many researchers have adopted rule-based frameworks [69]-[75] and intelligent algorithms to control smart devices for improving the energy efficiency in smart buildings. Tomic et al. designed “a semantic layer that integrates smart metering, building automation, and policy-based reasoning to offer energy optimization capability to energy consumers and providers” [71]. They stated that users can write system-level rules based on RDF triplets. However, they did not show how the rules for their framework are written. Kaliappan et al. proposed a “policy-based framework” for smart home management and they argued that their framework provided “intelligent and flexible energy management” [72] for smart home appliances. They “formalize the behavior of appliances using states and manage the energy consumption using policies” [72]. However, they also did not show how a policy is actually written and simply assumed that policies were written in well-formated English sentences. Kumar et al. proposed a technique for “intelligent semantic policy adaptation” [73] in smart buildings that “enables the agents of an application across different settings, context, environment, etc. to share and reuse semantic policies” among themselves. Kawakami

et al. proposed “a rule-based home energy management system that uses the Rete algorithm” [74], [76]. It is a widely used reasoning algorithm for IF-THEN expert systems. However, these solutions primarily focus on how to handle rules and do not describe the details of how suitable rules can be written for their systems.

2.3.3 Limitations

Using a rule-based expert system for smart building management enables flexibility in building automation systems. Domain experts can define rules and parameters for these systems; however, for modern smart buildings, which have programmable architecture, many limitations are apparent in existing studies.

Centralized Architecture

Conventional rule-based building management systems usually have a centralized architecture. In these systems, although a single rule is relatively simple, the interactions between a large number of rules may be complicated. Consequently, the system may crash when the number of rules increases beyond a certain value. Further, increasingly more smart devices are connected to building networks, and functionalities are exposed to the public by RESTful APIs. Users can control building systems from any device as long as it can access the building network. Therefore, users may require the ability to manage their personal workspace on their own devices instead of writing rules in a centralized system. A lightweight distributed framework is better than the traditional rule-based framework for IoT-enabled smart buildings.

No User-friendly Interface

Existing rule-based building management systems predominantly focus on reasoning and handling rules. None of them has studied how to define rules for their system. As discussed in previous sections, in order to enable collaborative control to optimize energy efficiency, it is important that users be able to write fine-grained control rules for a specific context. They may need personalized configuration for their own workspace; it is not possible for domain experts to predefine all the rules for every context.

In addition, smart buildings have heterogeneous networks and complex structures. Writing rules to control a building system requires knowledge of the following factors:

- building structures and semantic relations of inside objects and spaces;
- the method by which to refer to a service; and
- the services a space or device can provide.

This is not possible for users who are not good at programming or who are not familiar with the target building. Therefore, it is necessary to provide descriptive languages and lightweight distributed frameworks for them to write rules easily.

2.4 Resource Representation

From the viewpoint of a building as a distributed hardware computer, much like the classic computer, a resource abstraction layer that supports programming by limited programmers such as facility managers and residents is needed. Existing work on resource management is primarily concerned with resource discovery issues. Many researchers have studied resource representation in smart buildings [77]-[89] and have applied complicated information modeling methods to represent the relationship among objects and the static information of objects. To enable ordinary users to program buildings, the question of how to model the interfaces (in this case, RESTful APIs that are used to acquire context and control devices) of the devices and design translator or interpreter that can transfer an ordinary user's description of control logics into pieces of executable code that can be executed by machines is a critical issue [17], [45]. Therefore, the following requirements should be satisfied when developing a resource representation schema:

- relationship among objects, spaces, and services;
- property information of objects and spaces;
- detailed service representation and modeling; and
- an interpreter that can translate these representations into executable code.

To support upper-layer user programming of IoT smart buildings, the relationship of internal objects, services, and service description are the main concerns of resource representation in our research.

Ontology-based Resource Representation

Table 2-1. Studies that using ontologies to model the relationship of building resources

Objective	Feature	Authors
To ease the configuration process of heterogeneous BAS [78].	Ontologies for "seminal knowledge representation and auto-reasoning" on heterogeneous building networks.	Reinisch et al. 2008
To model every aspect of a service-oriented BAS [86].	Domain-dependent and specialized ontology for ambient intelligence.	Stavropoulos et al. 2012
To support effective management of home resources information [90].	Resource-aware with resource relation graph.	Son et al. 2011
To optimize energy consumption by designing ontology for both consuming devices and power resources [91].	Integration of two ontologies: devices and power generation/consumption description.	Grassi et al. 2011
To provide "a homogeneous layer for advanced smart building control" [92].	Interconnect disparate protocols used in BASs by developing an IPv6 service-oriented gateway for subsystems.	Jung et al. 2013

Reinisch et al. proposed using ontologies to represent knowledge as an abstraction of the heterogeneous network infrastructure [78]. The system could autonomously reason about the stored knowledge. Grassi et al. presented an ontology framework that provided necessary information modeling [91] for smart buildings. They focused on "two composing ontologies developed for describing devices and power generation and consumption for energy consumption optimization" [91]. Stavropoulos et al. designed a resource "ontology for ambient intelligence in smart buildings" [86]. They stated that the ontology includes "the classes needed to sufficiently model every aspect of the service-oriented smart building systems" [86]. However, these ontologies only describe the relationship among devices or model the devices themselves without presenting sufficient information about how to use the services.

Service-centered Resource Representation

Service-oriented architectures have also been used for building resources modeling [21], [94]. With the advancement of open web technologies such as Semantic Web increasing in the past decade, researchers have started to integrate services with building

ontologies using semantic web technologies. Han et al. presented a Service-Oriented Architecture-based building automation system [95] that models devices by "device profile for web services (DPWS)" [96]. They also designed a building resource ontology which included the descriptive information of concepts and relations that they use as the reference for the semantic schema data of buildings.

Table 2-2. Studies of service-centered resource representation

Objective	Feature	Authors
To ease the service mash-up for smart building users [21].	A semantic service mash-up building ontology.	Wan et al. 2013
To overcome the diversity of related technologies and protocols of BASs [94].	A "service-oriented reference architecture" with semantic descriptions for BAS.	Vicari et al. 2015

Combinatorial solutions were also proposed in the literature. Evchina et al. proposed an ontology for smart homes that focused on social services that they utilized for information monitoring [93]. Their ontology provides semantic connections for available data in smart homes, and information filtering can be performed by analyzing each user's profile with a query manager component. Asfand-e-yar proposed adopting semantic web technology to structure multifarious data for building management systems, with the objective of defining a middleware layer that simplifies the creation of sophisticated automation applications for the building management systems [84].

Industry Effort

The Open Mobile Alliance (OMA) is presently making an industry effort for resource constrained IoT device management. They have defined an application-layer communication protocol that "provides functionalities such as device management over sensor and cellular networks and transfer of service data from networks to devices" [97]. They primary focus is on device management and resource discovery, and they only provide a simple object-based resource model and resource operations. No detailed service representation schema is presented in their protocol. Consequently, an ordinary user cannot use it to make programs. We will discuss programming languages and approaches in section 2.5.

These studies modeled heterogeneous complex semantics among objects, spaces, services, and concepts in buildings. However, a common limitation is that detailed service representation is not presented in their proposals. Detailed service representation here refers to key information (e.g., service name, parameter, and return value) for modeling a service. Therefore, only domain experts can develop automation applications using frameworks based on these studies. Thus, a resource representation schema is needed such that once the resource is referred to, it can be translated into executable code by an interpreter.

2.5 Programming Languages for Smart Buildings

To date programming building systems has been an extremely difficult job that can only be accomplished by domain experts. The heterogeneity of all system levels prevents integration of smart building subsystems. For instance, field buses, communication protocols, embedded operating systems, programming languages, and software tools are quite different in each building. Even the same subsystem for different buildings may vary—from physical link protocols to programming languages. Thus, even a slight change may involve a series of dependent changes that may require complex tools and programming jobs that can only be achieved by domain experts [98].

In the age of IoT, smart building systems are becoming homogeneous structures via unified RESTful access and control interfaces. Programmers and users no longer have to know the details of subsystems, management of building systems can now be achieved through those RESTful APIs on any device that can be connected to building networks. However, using RESTful APIs to control smart building systems is still difficult for some users. In addition, users need to know the semantic relations of inside spaces, objects, and services. For example, when users wish to fetch the temperature of room A, they have to know whether there is a service for getting the temperature of room A and how to use this service. There should be mechanisms that can provide available services to which users can refer.

In 1988, Hartman published a proposed standard for an Operator's Control Language (OCL) [99] that standardizes the manner in which certain types of software is written

for DDC systems [3]. It followed the basic principle of control structure: IF, THEN, ELSE. However, it is designed for individual devices and subsystems. Although the advent of graphical programming languages eased the building system programming burden, they were still targeted at a single device or subsystem and lacked the flexibility possessed by text-based languages. In addition, protocol organizations and device vendors have their own control languages. For example, the BACnet [19] language argues that it provides interoperate ability for smart devices from various venders. However, they have to persuade all the venders to adopt their protocol and language. Devices that "use one of those languages can never interoperate directly with devices using the others" [100]. Procedural languages such as C and Java are also used to develop control software for a single device or subsystem. Currently, "all large building automation vendors only allow changes to their internal control languages using their proprietary software" [100]. The Sedona Framework is trying to define a universal programming language, which is "a general purpose component-oriented programming language for all kinds of embedded devices" [101], to address the interoperability issue between devices provided by different vendors. Sedona overcame the challenge of interoperating among vendors and protocols. However, it is still for a single device, which means that programmers have to install the software on the device first. Consequently, changes to the control logic may be costly and time-consuming.

In the age of IoT, both the device control and data acquisition interfaces can be implemented via the RESTful architecture. Therefore, the interfaces are unified and the whole building can be regarded as a programmable architecture or a distributed-hardware computer. This provides a chance for ordinary users who are not familiar with the target building or who have no programming skill to program the building system if we develop simpler programming languages for them. In addition, it can supply an overall perspective for controlling buildings. Users can write control sequences for different devices across various subsystems to make them operate in a collaborative manner to improve energy efficiency. Enabling end-users to program their surrounding environment has long been studied from the perspective of human-computer interaction [50]. This field also followed the "IF-THEN" paradigm.

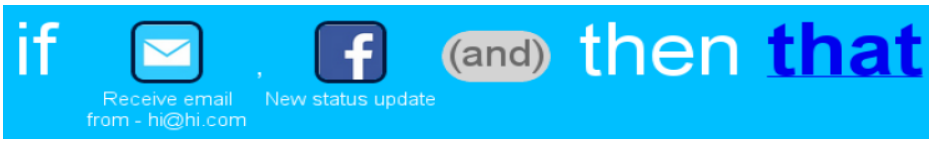
However, no mention is made of the lower-layer support for translating such "IF-THEN" representations into executable code that can be understood by machines.

Table 2-3. Studies for enabling end-users to program with the smart environment

Name	Description	Authors
CAMP	Studied the application scenarios conceptually and identified the user desired automation applications in smart home [102].	Truong et al. In *Proc. Ubicomp* 2004
iCAP	Visual programming interfaces for end-users to quickly create context-aware applications [103].	Dey et al. In *Proc. Pervasive* 2006
OSCAR	Supports "flexible and generic control of networked devices and services" [104] in smart home.	Newman et al. In *CHI 2006 Workshop*
	Studies the usability factors of "end-user composition interfaces for smart environments" [105].	Dahl et al. 2011

Blase et al. examined the practicality of enabling end-users to "customize the behaviors of smart home devices using a trigger-action programming" [50] method from the perspective of human-computer interaction. They concluded that "end-users can quickly learn to create automatic management applications containing multiple triggers-actions" [50]. They carried out user studies with various kinds of participants using visual programming interfaces (i.e., Figure 2-2) in the smart home.

a. Simple recipe of "If it is 6 pm, then turn the lights on."



b. Complex interface with multiple triggers in the recipe

Figure 2-2. Trigger-action programming interfaces used in [50]

Most of the participants did not have any programming experience with a few considered themselves sophisticated programmers. Overall, their results showed that the participants were "satisfied with the usability of both the simple and complex interfaces" [50] shown in Figure 2-2.

Several similar solutions were also proposed by researchers in the past decades. They prove that such a “drag and drop” interface is very efficient for common users to program smart environments. However, these studies supply no mechanism to translate such rules into executable code. Furthermore, much porting work has to be carried out in order to use such applications in different buildings because the structure and internal services vary. A resource management framework with standardized resource representation schema and operation interfaces is necessary to support programming of smart building systems by those users.

2.6 Contributions

This research provided a flexible programming framework for IoT-enabled smart building users to enable precise, personalized, and collaborative management of building systems.

We designed a lower-level descriptive rule definition language called DCRDL [17] for devices. DCRDL hides the building structure and service details so that users can focus on control logic description. It also adopts ConditionSet-ActionSet (CSAS), which can be used to express complex contexts and controls. Thus, a programmer can use DCRDL to write flexible and fine-grained control rules for devices in a particular context, which may be necessary for the collaborative control we discussed in previous sections.

It is still difficult for users to write control rules using DCRDL owing to the heterogeneous networks and complex structures constituting building systems. Moreover, DCRDL is designed using XML, which may be a bit difficult for users who are not good at programming. Therefore, we designed a simpler descriptive language called EPDL [45] to address these issues. EPDL has a very concise and semantic rule-based format that is very easy to understand.

We also proposed a smart building resource description schema for representing complex building resources. The proposed schema provides hint information for users when they need to refer to a certain service of a space or an object of the target building. It also assists the EPDL compiler to compile EPDL policies into DCRDL rules.

2.7 Summary

In this chapter, we discussed related work from all aspects of programming framework design for smart buildings. We explained the programming paradigm shift for smart buildings in the era of IoT and conducted surveys from three aspects: unification of data access and device control interface by web technologies, resource representation/modeling support for upper-layer applications, and programming languages for control of BAS. We regard the overall building as a distributed-hardware computer. Therefore, to enable ordinary users to program smart buildings, we stated that the distributed resources have to be abstracted (resource management and representation) and simpler programming languages (e.g., visual-based programming) provided, as done with classic computers.

3 PROPOSED SYSTEM

3.1 General Architecture

Our proposed system consists of four parts [45]: Smart Building Resource Management Framework (SBRMF), EPDL, DCRDL, and Policy Agent System (PAS).

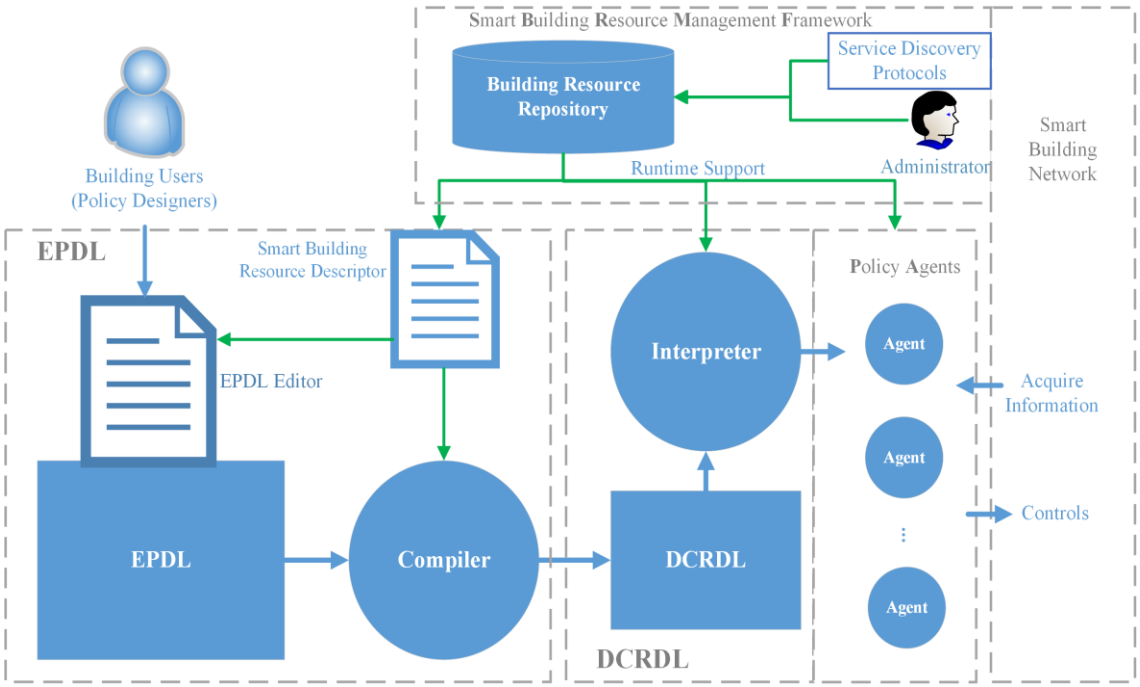


Figure 3-1. Architecture of the proposed framework

The output of SBRMF is a smart building resource descriptor. This descriptor is a

well-formed schema that includes the following: 1) Semantic relations between objects and spaces; 2) properties and available services information about each object and space; and 3) detailed information about each service (e.g., service name, parameters, and return value). EPDL consists of a smart building resource descriptor, a language editor, and a compiler. The energy policy designer writes policies in the EPDL editor, and the EPDL compiler translates source code to DCRDL with the help of the resource descriptor. The DCRDL interprets the compiled source to policy agents. Finally, these agents run as threads in the system with the support of smart building resource management framework. The thread checks rules defined in it at runtime to perform context-based controls on appliances in smart buildings.

3.2 Target Users

We classify smart building users into two categories: domain experts and non-expert users.

- **Domain experts** are smart building automation system developers. They are usually good at programming, especially in smart building systems.
- **Non-expert users** are those users that are not good at programming or who are not familiar with building systems.

We further classify non-expert users into two categories: building practitioners and end users.

- **Building practitioners:** These users include building administrators, facility managers, IT managers, and energy managers. They are limited programmers who can be trained to use simple tools to manage smart buildings.
- **End users:** These users include residents of the smart buildings. They usually have no knowledge of programming and are not familiar with smart buildings.

3.3 Application Scenarios

In this section, we discuss the potential application scenarios [106] for our framework. The automation applications developed using this framework can be deployed on any capable devices only if they are connected to the building network. People can write

policies that make the devices in different subsystems operate collaboratively to optimize the overall energy efficiency of smart buildings. Figure 3-2 shows the main potential applications of this framework. We discuss these applications in the ensuing paragraphs.

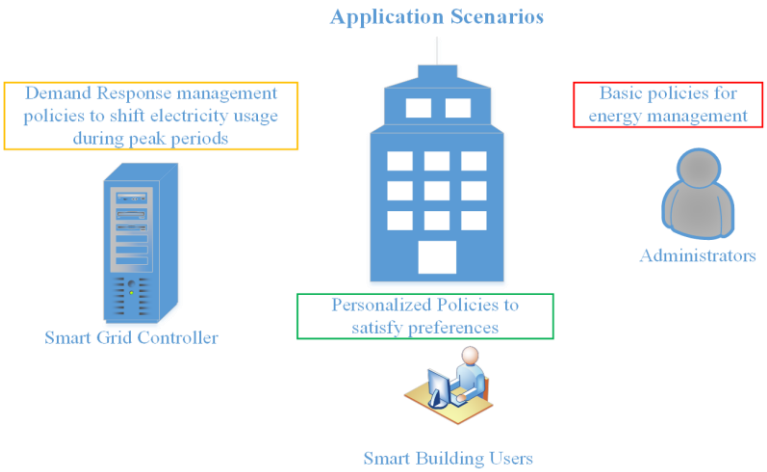


Figure 3-2. Example application scenarios

Basic Building Energy Policy

This kind of application is usually developed by building administrators (e.g., facility managers) to control the basic behaviors of internal appliances for energy efficiency. For example, the allowed set-point range of air conditioners and the basic movement rules of elevators. These policies should have the highest priority and should never be violated. The administrators can write and run such policies easily using EPDL instead of procedural languages such as C, Java and JavaScript.

Personalized Policy

The ultimate goal of energy efficiency is both maintaining comfort level and reducing energy consumption, but the detailed requirements may vary according to the different users and contexts. For instance, in a large room equipped with many sets of lights and several air conditioners, when sensors detect an occupant, the policy should turn on the lights and air conditioners related to this user, and set the parameters of the air

conditioners according to the preference of this user. Therefore, ordinary users in smart buildings can write personalized applications to configure their own workspace separately using EPDL.

Smart Grid Policy

Future smart buildings must be compliant with smart grids [107], in which demand response management is a critical issue and challenge. Smart grid controller can run some policies written in EPDL to control non-urgent tasks (e.g., dishwashing) in connected buildings during off-peak periods. We can also deploy some applications developed based on pricing policy to save on electricity spending.

3.4 Workflow

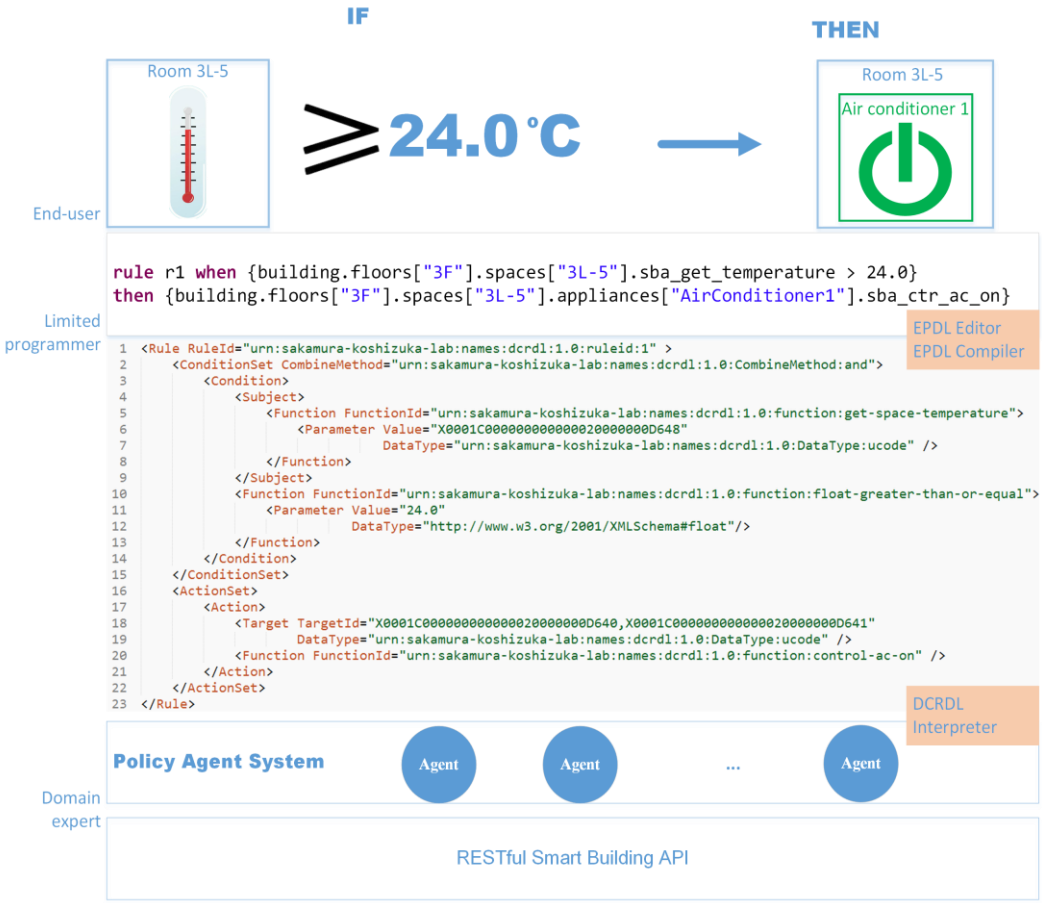


Figure 3-3. The workflow of an IF-THEN rule in this framework

In this section, we present a running example to explain how a rule defined by the end-user is translated into executable code in our framework.

- End users can drag and drop some icons representing resources, signs, etc. to compose rules. As shown in Figure 3-3, an end-user can drag icons that represent the temperature of room 3L-5, greater-than-or-equal sign, air conditioner 1 in room 3L-5, and value "24.0" to form the following rule:
  IF the temperature of room 3L-5 is greater than or equal to 24.0 °C, THEN turn on air conditioner 1 in room 3L-5.
- There can be more than one condition in the IF branch and the combination of these conditions constitute a complex context. Multiple actions also can be defined in the THEN branch. Actions should be defined to adopt the context for related devices.
- The composed "application" is then translated into EPDL (i.e., second part of Figure 3-3). A key function of EPDL is modeling of service reference in a simple manner. For example, fetching the temperature of room 3L-5 can be modeled as,
  building.floors["3F"].spaces["3L-5"].sba_get_temperature
- There is an ucode and a service called "sba_get_temperature" in space "3L-5" of the smart building resource descriptor. The service is compiled to a function ID defined in DCRDL and ucode, which represents the identifier of space "3L-5," is passed as the parameter of this function:

  <FunctionFunctionId="urn:sakamura-koshizuka-lab:names:dcrdl:1.0:function:get-space-temperature">

  <Parameter Value="X0001C000000000000020000000D648"

  DataType="urn:sakamura-koshizuka-lab:names:dcrdl:1.0:DataType:ucode" />

  </Function>
- Finally, the DCRDL code is interpreted into a policy agent. The key part is a smart building API modeling component that connects such description code to a real RESTful service. We present the detailed implementation of this component in section 5.2.2.

3.5 Novelty

3.5.1 Lightweight and Distributed Architecture

Building automation is the automatic management of building services such as heating, ventilating, air-conditioning, lighting, and elevator systems [108]. Although the devices and machines of subsystems are physically distributed, these systems are usually integrated with a centralized computer-based management platform. For example, the control room of a building may have several control systems installed to control subsystems respectively. Existing building automation applications are usually installed on powerful computers with high computing capability. With the number of installed applications increasing, the system might be overloaded.

With our framework, users can write and run automation policies on any device. The only requirement is that the device be able to run interpreters and connect to the network of the target building. It adopts the IF-THEN control logic, which is the basic principle for building automation. Many people may view this as similar to a rule-based expert system. However, they are quite different. The following table shows the differences between them. Our framework focuses on defining and running rules, whereas a rule-based expert system focuses on reasoning rules.

Table 3-1. Comparison of our framework with the rule-based expert system

	Our Framework	**Rule-based Expert System**
Architecture	Distributed	Usually centralized
Hardware/platform	From smartphones to powerful computers	Powerful computers
Objective	To achieve personalized, precise, and collaborative control	Find new facts from existing rules and data
Rule reasoning	Currently, NO	YES
Rule definition	Main focus	Secondary

3.5.2 Collaborative Control

A building automation system comprises a set of subsystems which are run solely to

control the environmental parameters with which they concerned. Currently, smart building subsystems are still isolated from the perspective of control. As we have discussed in section 1.2, a smarter building automation system requires the cooperation of devices from different systems.

For example, when a user walks from the front door of the building to his/her seat, subsystems such as authentication, positioning, elevator, and door control system act, respectively. The user may have to authenticate several times on the gate control system because there may be multiple doors on the route to his/her seat. If these subsystems operated collaboratively, human comfort and productivity may be improved and energy efficiency optimized. For instance, if the authentication, positioning, elevator, and gate control systems shared information about the user, the user would only need to authenticate once at the front door. Upon authenticating at the front door, the elevator system would arrange to have an available elevator at the ground floor. With the cooperation of positioning, authentication, and gate control system, internal doors would then open when the user approached those doors. In this scenario, the user's time would be saved; thus, productivity and user comfort level would be improved.

Our framework is designed above the level of the smart building APIs. It views the whole building as a distributed hardware computer with programmable architecture and provides a programming environment for users to write rules and policies. The rules and policies can get information from the building system and control individual devices from different subsystems working in a collaborative manner. The cooperation of devices from different subsystems increases the intelligence of the smart building, resulting in optimized building energy efficiency.

3.5.3 Personalized Control

Existing building automation systems are typically designed using a centralized architecture to minimize occupant interaction. With the rise of IoT, the connectivity of internal devices has significantly improved. Users are becoming a constant part of the smart environment of the building. They usually require personalized control for their workspaces according to their preferences. For example, users may have different

definitions of “hot” in a room, which may depend on the context, such as the climate, and their clothing. Therefore, they may require different set-points for their workspace in a large room where there are multiple air conditioners heating and cooling different areas.

Our framework is a lightweight distributed architecture. Any device that have the ability to run the compiler and interpreter of its two descriptive languages (DCRDL and EPDL) can be used to control a building system. Consequently, personalized control can be achieved by users writing rules and policies for building systems using their smartphones, pads, etc. This obviously optimizes the energy efficiency by improving user comfort, which is a sub-factor of energy efficiency.

3.5.4 Precise and Flexible Control

Personalized control mainly addresses the issue of personalized configuration of someone’s living and working spaces. Precise and flexible control also should be enabled to optimize energy efficiency. In general, building automation systems share a basic control principle called context-based control (or IF-THEN logic). It acquires context information (e.g., environment parameters and human activities) as input for automation algorithms and output controls as feedback to the building system to adjust its status. Early engineers adopted this principle to design facility controllers for smart buildings. For example, the lighting system controller detects the presence/absence of a human within a period to turn on/off lights. It is a very simple IF-THEN logic.

However, practical scenarios may vary according to time, season, user preferences, etc. For example, it is not necessary to turn on lights in the daytime even when a human presence is detected. More precisely, the lighting control system may need light intensity information to decide whether to turn lights on or not when a human presence is detected. Therefore, precise and flexible rules should be applied to control related devices to act properly according to a specific context.

In our framework, we designed two descriptive languages for writing flexible and precise control policies and rules. The rules in these two languages follow the ConditionSet-ActionSet (CSAS) format. ConditionSet represents a combination of

multiple conditions that form a specific context. ActionSet represents behaviors related devices should perform when the context represented by ConditionSet is detected. ConditionSet is designed recursively to model the flexible complex context in smart buildings. Therefore, a rule is able to represent flexible complex contexts and precise controls based on the context. A set of such rules forming a policy is able to achieve a certain energy efficiency optimization goal.

3.5.5 Building Resource Abstraction

A critical issue that confronts developers developing universal programming interfaces for smart buildings is the mix of heterogeneous networks and complex service details involved. Programmers need knowledge about building structures, network architectures, available services, etc. to develop proper automation applications for managing such buildings. For example, we adopted Ubiquitous ID technology [109] to identify objects and spaces in the Daiwa Ubiquitous Computing Research Building. If a developer wants to develop an automation application for smartphone users, he has to know the IDs for each device and the services (e.g., turn on/off) that a device can provide. In order to apply an application to a new building, it must be revised to adapt to the structure, internal networks, and services of that building. Thus, a resource representation schema that abstracts building resources is needed.

In addition, for a non-expert user who is not good at programming or who is not familiar with the target building, more standard resource representation schema is necessary. For example, in EPDL, we designed a building resource description schema to organize building resources. System developers or resource discovery protocols can issue resources for public reference using this schema. We also developed an EDPL editor so that users can use a semantic expression such as,

```
building.floors["3F"].spaces["3L-5"].appliances["AirConditioner1"].sba_ctr_ac_off
```

to refer to the service of turning the air conditioner off with the index of "*AirConditioner1*" in room "*3L-5*" of floor "*3F.*" In addition, such an expression can be translated to executable code that invokes the real service of turning an air conditioner

off. Therefore, non-expert users can write control policies easily using public building services. No existing study in building resource abstraction supports such a mechanism.

4 FRAMEWORK DESIGN

4.1 Design Principle

To enable ordinary users to program smart buildings, we designed our framework based on two principles:

- abstraction of complex building resources and structures; and
- separation of upper-layer user control logics and running details of the services.

We first designed an agent-based system to model the running environment of our future programming languages. Then, we created a Device Control Rule Definition Language (DCRDL) [17] that enables programmers to write control rules for devices easily. A more concise and simple Energy Policy Description Languages (EPDL) [45] with a set of tools was also designed to support users who are not good at programming or who are not familiar with the target building.

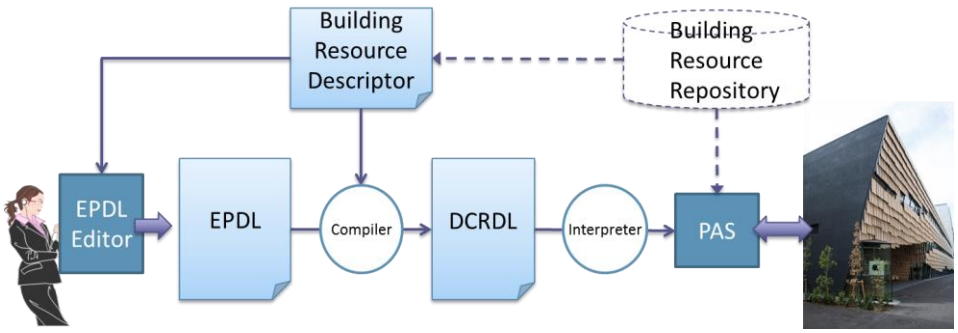


Figure 4-1. System overview

4.2 Policy Agent System (PAS)

The objective of PAS is to model the running environment of DCRDL and EPDL. Policy agents are interpreted from DCRDL rules. For example, when the building administrator wants to control peak energy consumption and shift the peak to an idle period, he/she can write rules to control devices or appliances running in a lower consuming state or turn them off if the energy consumption has reached a defined peak using DCRDL. Then, the DCRDL interpreter translates those rules into a policy agent. The agent checks all the inside rules within the given period or triggered by an event. The event can be any environmental changes that related to the inside rules. We studied the peak demand control problem based on PAS in [67] to evaluate the practicality of the PAS system.

4.2.1 Peak Demand Control

Smart building systems comprise a variety of subsystems, including a wide range of machines, appliances and constrained devices. When they are networked together by protocols, and some energy efficiency policies are applied, a significant reduction in energy usage and costs can be achieved. Peak electricity demand is the result of temporarily correlated energy demand surges caused by uncoordinated operation of subsystems such as air conditioning, heating, and ventilating [67]. It can cause serious problems such as service unavailable and high cost of energy. Future smart buildings must be able to balance their electricity usage and reduce their electricity consumption.

Energy control policies are usually applied to perform fine-grained controls on devices to save energy. They usually include a collection of device control rules that define a set of actions for target devices when specified conditions are met. There are also other static constraints in the system. For example, there should be a range for the set-point of air conditioners, which may differ in summer and winter; the air conditioners should never be turned off in an intensive care unit. The building system should have the ability to shift demand peak to idle periods. In other words, it should find a set of rules to execute to lower the real-time energy demand without breaking any

constraint in the building.

4.2.2 The Constraint Satisfaction Problem

Constraint satisfaction problem (CSP) is modeling method which has been widely used for resolving combinatorial issues in "operational research such as scheduling and timetabling" [110]. A CSP requires a value, selected from a given finite domain, to be assigned to each variable in the problem, such that all constraints related to the variables are satisfied [110]. The domain of a variable holds the information about what value a variable can take. The constraints of the problem limit the variable values that can be assigned simultaneously.

We model the peak demand control in the building energy management system as a CSP problem. The target devices are lights, air conditioners, and ventilators. The period of electricity consumption of light, air conditioner, and ventilator, *i*, are represented as $E_{li,}$ E_{aci}, E_{vi}, respectively. V represents variables (lights, air conditioners, and ventilators). D represents the variable domains (e.g., on/off status of light and set-point of the air conditioner). C represents the constraints. We define the constraints as follows:

- Feature constraint. The features of the devices. For example, the set-point ranges allowed by an air conditioner.
- Preference constraint. For a given situation, many sets of control actions can meet the constraints. That is, when the total consumption of electricity is approaching the peak value, more than one device can be turned off. Preference constraint indicates which one should be turned off first. For example, turn off the air conditioner in the room where the temperature is the lowest.
- Peak constraint. The total electricity consumption should not exceed the predefined peak. This is the main constraint of the peak demand control problem.

This CSP can be represented as follows:

$$(V, D, C) \tag{4-1}$$

Where:

$$V = \{E_{l1}, E_{l2,} \ldots, E_{ln}, E_{ac1}, E_{ac2,} \ldots, E_{acn}, E_{v1}, E_{v2,} \ldots, E_{vn}\} \tag{4-2}$$

D is given by,

$$D_{li} = \{0, 1\}, \quad D_{vi} = \{0, 1\},$$

$$D_{aci} = \{20, 21, 22, 23, 24, 25, 26\}, i \in N^{+} \tag{4-3}$$

C is the constraint applied to the peak electricity demand control:

$$\sum_{i=1}^{n} E_{l} + \sum_{i=1}^{n} E_{ac} + \sum_{i=1}^{n} E_{v} \leq E_{peak} \tag{4-4}$$

4.2.3 Design of the Peak Demand Control Framework

The framework consists of a set of constraints (i.e., Policy and rules), an interpreter, a context monitor, and an executor. All the feature and preference rules are defined in policy source file following the XML format. The interpreter translates policy source file and generates executors with the translated policies. It may generate more than one executor because the user may define peak control rules for the whole building, as well as for individual rooms.

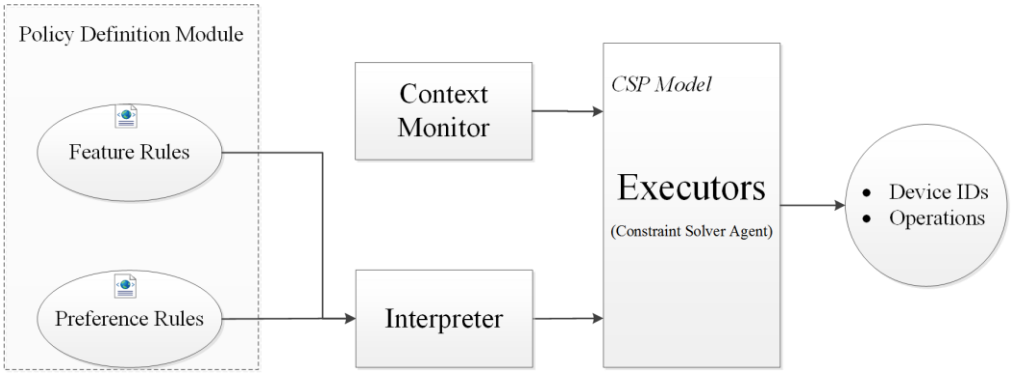


Figure 4-2. Architecture of the peak electricity demand control framework

Policy Definitions

The inputs are the operating rules for the smart devices in the building. These rules not only describe the features and functions of the devices, but also define the user preferences, such as “turn the air conditioners off in the room where the temperature is the lowest” and “never turn off the air conditioner if there is a server.”

Interpreter

The interpreter translates these policies into constraints, which constitute the inputs for executors that apply constraint programming technology. The executors determine a set of operations that turn off some devices based on the input constraints from the interpreter.

Context Monitor

To simplify the search algorithm for the CSP, we designed a combination priority mechanism. The framework adjusts the situation of the proper devices (i.e., devices that have a higher priority to be adjusted) according to the defined policies. The combination priority consists of high priority and low priority. High priority is defined by the user and cannot be changed at runtime. Low priority is initialized by the user, but can be changed if the environmental situation changes.

For example, the low priority of air conditioners in two rooms can initially be set at the same level. Then, when the temperature in one room is lower than that of the other, the low priority of air conditioners in this room will change to a higher level than that of the air conditioners in the other room. In other words, when electricity demand reaches its peak, the air conditioners in this room will turn off or increase the set-point to reduce electricity consumption.

Executor (Constraint Solver Agent)

We only implement one agent, called constraint solver agent for the peak demand control issue, to run in the PAS. It adopts a Backtracking Search algorithm to find a set

of predefined rules that comply with the system constraints in the policy file and then run the selected rules to cut the demand peak. During the backtracking search, it also refers to the priority values of the devices that are monitored by the Context Monitor.

4.3 DCRDL: Device Control Rule Definition Language

4.3.1 Motivation for DCRDL

In the previous section, we have introduced the modeling of peak demand control issue of smart buildings using constraint satisfaction method. The constraint solver agent finds a set of devices to be operated to shift the peak electricity demand without breaking user-defined constraints. The ensuing research directions can be,

- Design framework and interfaces for smart building users to define constraints (i.e., Write control rules/policies) for the building automation system; and
- Design efficient rule reasoning algorithms to resolve the rule conflict issue.

The second direction is related to an old topic: rule-based reasoning which is also the key issue of expert systems. Substantial work has been down to deal with this issue. We do not discuss the rule conflict problem in this thesis and it will be addressed in future publications . We select the first directions as the next step of this research.

Therefore, we created a rule definition language, called DCRDL, for context-based device control in IoT-enabled smart buildings. DCRDL allows users to write complex control rules that control device behavior based on the sensed context to achieve the goal of energy efficiency in a programming approach. ConditionSet-ActionSet (CSAS) or multiple Conditions-Actions rule is applied as the basic design principle of our rule definition language.

The ultimate objective of our work is to achieve context-based device control of energy efficiency in smart buildings. Although the effect of improving energy efficiency depends on the designed rules, a proper rule exhibits the potential to improve energy efficiency in smart buildings. Complex and flexible rules should be expressible and adaptable to the various situations in the building. A rule must be capable of modeling complex contexts and defining multiple control actions. When executing actions, the

rule should be capable of selecting targets using services or combination of services. Therefore, the following requirements should be satisfied:

- Multiple conditions: multiple actions format rules;
- Logic algebra for complex condition combinations; and
- Function description.

A DCRDL rule primarily consists of a ConditionSet and an ActionSet. To realize the complex expressivity of the logical relations of conditions, ConditionSet adopts a recursive design, as illustrated in Figure 4-6. Another recursive design is the function and parameter elements. As shown in Figure 4-9, the parameter element can call another function and the return value will be treated as the value of the parameter. The function is a key element of DCRDL and is used to obtain situations, select targets, control targets, and evaluate the conditions of a rule. The details of these functions are defined in the language specification [111].

4.3.2 Context-based Device Control

With the aid of wireless sensor networks, building energy management systems (BEMSs) have begun to acquire detailed information about energy consumption at different levels [4], such as appliance level, room level, and building level. Consequently, sophisticated approaches for energy efficiency are now possible with the detailed information supplied. To optimize the overall energy efficiency of smart buildings, devices need to behave properly and collaboratively according to certain rules related to context (e.g., energy consumption, environmental conditions, and human activities). We call this kind of control context-based device control which is regarded as the basic control principle of smart buildings. The DCRDL is designed to deal with this kind of control.

4.3.3 DCRDL Class Diagram

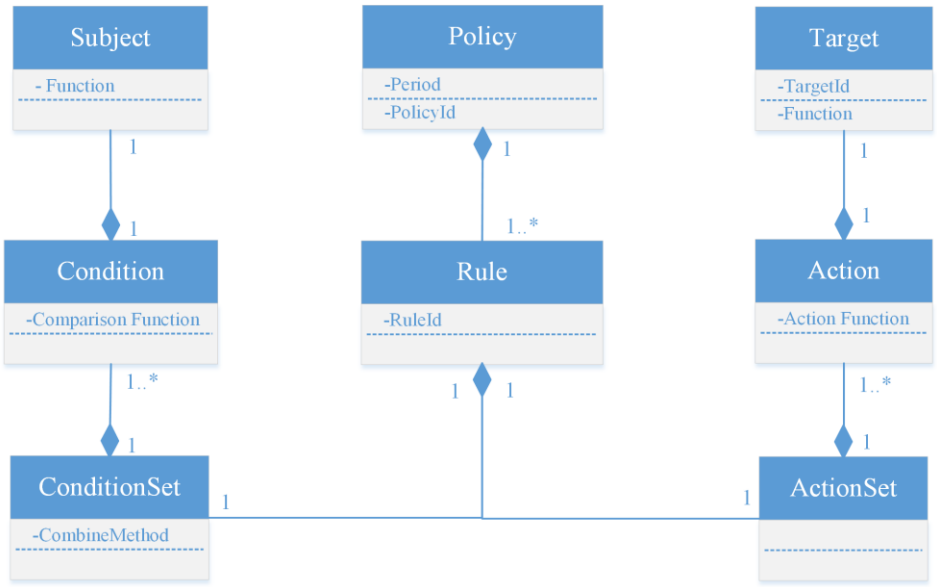


Figure 4-3. DCRDL class diagram

4.3.4 Elements of DCRDL

Policy

The policy is the topmost element of DCRDL. A policy includes several rules defined in the policy element to achieve a general control goal or algorithm. Figure 4-4 shows example code for a policy in DCRDL.

```
<Policy PolicyId="org:sakamura-lab:names:dcrdl:1.0:policyid:1"
        PolicyName="Discomfort Index Control Policy"
        Period="900">
    <Rule RuleId="org:sakamura-lab:names:dcrdl:1.0:ruleid:1" >
        ...
    </Rule>
    <Rule RuleId="org:sakamura-lab:names:dcrdl:1.0:ruleid:2" >
        ...
    </Rule>
    <Rule RuleId="org:sakamura-lab:names:dcrdl:1.0:ruleid:3" >
        ...
    </Rule>
</Policy>
```

Figure 4-4. Code example of a Policy in DCRDL

The PolicyId attribute in the policy element identifies this policy in the system. This attribute is not used in the current version; however, we plan to use it in a future version as a functional extension. The DCRDL interpreter executes a policy as a polling thread, and the Period attribute indicates the polling cycle of this policy. The event-driven mode (see Appendix 4) should be implemented if the data platform of the building provides eventing mechanism.

Rule

A rule is a key element of DCRDL. It is embedded inside the policy element to define specific behaviors for devices. It consists of a ConditionSet section and an ActionSet section. When the ConditionSet section of the rule is evaluated to be true, all the actions defined in the ActionSet section are executed.

```
<Rule RuleId="org:sakamura-lab:names:dcrdl:1.0:ruleid:1" >
<ConditionSet CombineMethod="org:sakamura-lab:names:dcrdl:1.
0:CombineMethod:and">
    <Condition>...</Condition>
    <Condition>...</Condition>
</ConditionSet>
<ActionSet>
    <Action>...</Action>
    <Action>...</Action>
</ActionSet>
</Rule>
```

Figure 4-5. Code example of a Rule in DCRDL

ConditionSet

ConditionSet is a recursive element that consists of conditions and sub-ConditionSets. It is designed to represent the complex logical algebra of conditions.

```
<ConditionSet CombineMethod="org:sakamura-lab:names:dcrdl:1.
0:CombineMethod:and">
    <Condition>A</Condition>
    <ConditionSet CombineMethod="org:sakamura-lab:names:dcrdl:1.
    0:CombineMethod:or">
        <Condition>B</Condition>
        <Condition>C</Condition>
    </ConditionSet>
</ConditionSet>
```

Figure 4-6. Code example of a ConditionSet in DCRDL

The CombineMethod attribute describes the logic relation in this ConditionSet. In Figure 4-6, the first layer condition is A and the conditions in the inner ConditionSet are B and C. The logic relations of these conditions can be expressed by the following logical algebra formula:

$$A \wedge (B \vee C) \quad (4\text{-}5)$$

ActionSet

An ActionSet is composed of several actions that control behaviors of context related devices. All the defined actions in the ActionSet are executed when the corresponding ConditionSet in the rule is evaluated to be true.

Condition

A condition compares the building information with the user-defined value. It consists of a subject and a comparison function.

```
<Condition>
    <Subject>
        <Function FunctionId="org:sakamura-lab:names:dcrdl:1.
        0:function:get-space-discomfort-index">
            <Parameter Value="00001C000000000000020000000D648"
                       DataType="org:sakamura-lab:names:dcrdl:1.
                       0:DataType:ucode" />
        </Function>
    </Subject>
    <Function FunctionId="org:sakamura-lab:names:dcrdl:1.
    0:function:float-larger-than">
        <Parameter Value="21.0"
                   DataType="http://www.w3.org/2001/
                   XMLSchema#float"/>
    </Function>
</Condition>
```

Figure 4-7. Code example of a Condition in DCRDL

The above condition gets the discomfort index of a room using the get-space-discomfort-index function with a parameter that stands for the space ID. Then, the float-larger-than function is invoked to compare the obtained discomfort index value with the specified value of 21.0. The return value of float-larger-than represents the evaluation result of the condition.

Action

The action includes a target and an action function. The target element uses a TargetID, the ucode [109], to indicate the control target. A target can also invoke functions to get the control targets. An action function takes a ucode and parameters as input to carry out control of the target indicated by a ucode.

```
<Action>
    <Target TargetID="org:sakamura-lab:names:dcrdl:1.
    0:ucode:00001C0000000000000020000000D448A"
            DataType="org:sakamura-lab:names:dcrdl:1.0:DataType:ucode"/>
    <Function FunctionId="org:sakamura-lab:names:dcrdl:1.
    0:function:control-ac-setpoint-increase">
        <Parameter Value="1"
                   DataType="http://www.w3.org/2001/XMLSchema#integer"/>
    </Function>
</Action>
```

Figure 4-8. Code example of an Action in DCRDL

Function

A function is the most important element of DCRDL. It performs building information acquisition, comparison, and device control. The parameter element inside a function can also invoke other functions to get building information as the value of this parameter. Defining new functions can extend the functionality of this language. We only need to define the human detection and email sending function if we wish to send a notification email to the security department when a stranger is detected in the building. In addition, service mash-up is allowed in DCRDL. Users can compose new functions by using standard built-in functions defined in [111] to perform complex context acquisition, as shown in Figure 4-9.

```
<Function FunctionId="org:sakamura-lab:names:dcrdl:1.
0:function:get-acs-by-spaces" >
    <Parameter DataType="org:sakamura-lab:names:dcrdl:1.
    0:DataType:ucode-list">
        <Function FunctionId="org:sakamura-lab:names:dcrdl:1.
        0:function:get-space-by-highest-temperature" />
    </Parameter>
</Function>
```

Figure 4-9. Code example of a Function in DCRDL

Functions in DCRDL can be classified into three categories:

- Comparison Function

  These functions are used in a condition. The comparison function compares the obtained environment parameter with the given value to determine whether a condition is true or false. For example, the float-larger-than function compares the temperature of the target space with the given value to evaluate if the temperature has exceeded the given value or not.

- Information Obtaining Function

  Information obtaining functions can be used in both conditions and actions. These functions fetch the information from the environment and also perform statistics such as calculating the total electricity consumption of specified rooms or selecting target spaces and objects (e.g., selecting the room in which the temperature is lowest in the building).

- Action Function

  Action functions perform device controls in the smart building, such as turning on/off lights and changing the set-point of an air conditioner.

4.4 EPDL: Energy Policy Description Language

DCRDL is a low-level language created to enable expert users to write fine-grained rules for appliances in order to improve energy efficiency. Users have to manually describe the raw structure of the rules in a well-formed XML file when using DCRDL. In addition, users must be familiar with the details of the target building when using DCRDL. In this section, we introduce EPDL, a high-level language that supports human readable energy control policy writing without exhaustive knowledge of the building. EPDL policies are compiled to DCRDL format, and the execution environment is designed and implemented in DCRDL and PAS.

In this section, we present the design of the following components of EPDL:

- EPDL: A high-level user-friendly energy control policy description language for improving energy efficiency in smart buildings.
- Smart building resource description schema that eases the energy control policy

design.

- A set of tools that support writing and compiling energy control policies using EPDL.

4.4.1 Motivation for EPDL

The objective for EPDL is to create a user-friendly energy policy description language to ease the policy writing for smart building users. Therefore, we designed a simpler rule format based on DCRDL and added more flexible control logics such as LOOP and IF. We also designed a smart building resource descriptor, language editor, and a compiler to support writing and running energy control policies with EPDL. Figure 4-10 shows the detailed architecture of EPDL.

The smart building collects the raw data relating to occupants on each floor from the sensor network and obtains real-time occupant distribution state using big data analytics technologies. This analyzed semantic information can be opened to users via RESTful APIs. Then, we can apply a rule, for example, to the elevator system that stops idle elevators on the floor where the most occupants in the building are. As a result, both the waiting time for most people and the movement of elevators are reduced in the long run. In other words, the energy efficiency of the smart building is improved.

However, programming skills and knowledge about smart buildings are required for writing such a rule in DCRDL. It is difficult for users to design energy control policies easily without providing them with information of available services and the semantic structure of the smart building. For instance, writing a rule for the above case, the required basic information includes 1) how to get occupant distribution status on each floor, 2) how many floors and elevators there are, and 3) how to control the elevators. Users need to study how to use the available resources and services before writing an energy control policy for smart buildings. Expressing building resources in a simple well-formed data schema and providing them to users is a challenge. Therefore, we propose a more concise Energy Policy Description Language to address these issues.

4.4.2 Architecture

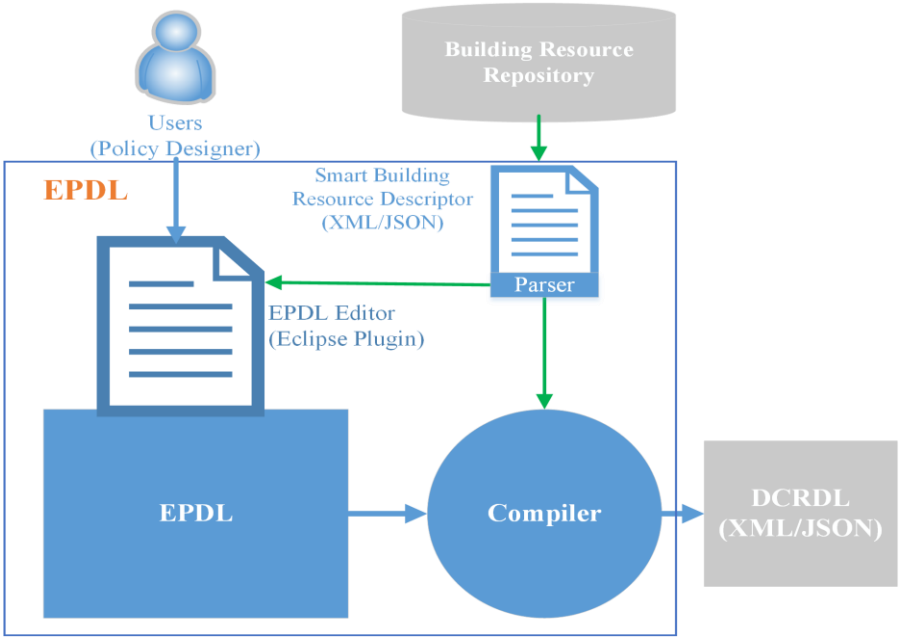


Figure 4-10. The architecture of EPDL

EPDL consists of a smart building resource descriptor, a language editor, and a compiler. The energy policy designer writes policies in the EPDL editor, and the EPDL compiler translates source code to DCRDL with the help of the resource descriptor. The DCRDL interprets the compiled source to policy agents. Finally, these agents run as threads in the system with the support of the smart building resource management framework. The thread checks the rules defined in it periodically at runtime to perform context-based controls on appliances in smart buildings.

4.4.3 Smart Building Resource Description Schema

Overview

```
<?xml version="1.0" encoding="UTF-8"?>
<descriptor>
<building ucode="00001CB00000000000002000000100000" name="Daiwa Ubiquitous Computing Research
Building">
    <functions system-time="time()" season="system-get-season"
                sba_get_space_of_lowest_Temperature="org:skl:names:dcrdl:1.0:function:get-space
                -of-lowest-temperature"/>
    <floor  ucode="00001CB00000000000002000000110000" name="Third Floor" index="3F">
        <space  ucode="00001CB00000000000002000000111000" name="Sakamura-Koshizuka Lab" index="
        3L-5" type="student_room">
            <functions sba_get_DiscomfortIndex="org:skl:names:dcrdl:1.0:function:get-space-
            discomfort-index"
                sba_get_Temperature="org:skl:names:dcrdl:1.0:function:get-space-temperature"
                ..... />
            <appliance ucode="00001CB00000000000002000000111001" name="AirConditioner 1" index="
            AirConditioner1" type="air_conditioner">
                <functions sba_ctr_ac_on="org:skl:names:dcrdl:1.0:function:control-ac-on"
                ..... />
            </appliance>
            <appliance ucode="00001CB00000000000002000000111004" name="Light Set 1" index="
            light1" type="light" >
                <functions sba_ctr_light_on="org:skl:names:dcrdl:1.0:function:control-light-on"
                sba_ctr_light_off="org:skl:names:dcrdl:1.0:function:control-light-off" />
            </appliance>
            .....
        </space>
        .....
    </floor>
    .....
</building>
<sbapi>
    <function return-type="org:skl:names:dcrdl:1.0:data-type:void" id="org:skl:names:dcrdl:1.
    0:function:control-ac-on" description="Turn on an air conditioner with the specified ucode.
    ">
        <parameter name ="ucode" data-type="http://www.w3.org/2001/XMLSchema#String"
        description="The ucode of the target air conditioner."/>
    </function>
    .....
</sbapi>
</descriptor>
```

Figure 4-11. An example smart building resource descriptor in the XML format

To assist users who have little knowledge of smart buildings to write an energy control policy, we designed a smart building resource descriptor that contains the following types of information organized in the XML/YAML format.

- Semantic relations between spaces and objects. This kind of information describes semantic relations such as a sensor or an appliance and the room to which it belongs, and the kinds of rooms a floor consists of. Four main elements

are defined in the descriptor: building, floor, space, and appliance.

- Properties of spaces and objects. Properties are name-value pairs that represent the attributes of a space or an object. An identifier and index property are required for every object and space. The identifier is the unique identification of an object or a space in the system and the index property is used to help users refer to an object or a space. The ucode [109] is used to identify property in the descriptor.
- Services for acquiring context information and controlling appliances. All the services of an object or a space are represented as attributes of the function element in the form of name-value pairs. The function element is a sub-element of the building, floor, space, and appliance elements.
- Description of functions. The description of a function includes return type, function ID, and parameters. The attributes of the parameter element consist of name, data type, and index. The index indicates the position of the parameter in the parameter list. Users are encouraged to add description attribute, which describes the details of this element, to the function and parameter elements.

In brief, the descriptor describes the semantic information of spaces and objects inside smart buildings via a well-formed XML/YAML file. It also includes features and available services of each space and appliance. We are planning to extend this descriptor to a building resource management framework for supporting flexible resource management in future works.

Structure

In this section, we introduce in detail elements of the smart building resource descriptor. The smart building resource description schema is defined in Appendix 1. The schema of the descriptor is designed in both XML and YAML format. Fundamentally, there are four main elements in this schema: <building>, <floor>, <space>, and <appliance>. Properties and function-list are required for each element. Properties and function-list are described using key-value pairs. Keys are for upper-layer users to understand the resource better. Values are inner properties or

service ids for lower-layer systems. Two properties are required for each element:

- Index: for upper-users to refer to an element.
- Ucode: the unique identification for objects, locations, concepts, etc. using Ubiquitous ID technology [109]

<building> is the top element of the descriptor. It describes all the information directly related and inside the building. The inner <functions> element lists all the available services of the building layer. Attributes in the <building> element describe the properties of the building layer.

<floor> is a sub-element of <building> and gives information about the floor layer. The inner <functions> element lists all the available services of the floor. Attributes in the <floor> element describe the properties of the floor layer.

<space> element describes information of all the physical space. The inner <functions> element lists all the available services of a space. Attributes in the <floor> element describe the properties of space layer.

<appliance> describes information about a device, appliance, etc. The inner <functions> element lists all the available services of the appliance. Attributes in the <appliance> element describe the properties of an appliance.

<sbapi> contains a detailed description of smart building APIs. It includes return-type, parameter details, and function ID defined in the DCRDL specification [111].

4.4.4 Elements of EPDL

There are four main elements in EPDL: Policy, Rule, IF and Foreach statement. A policy contains the Rule, IF, and Foreach statements.

Policy: the policy element is the wrapper of Rule, IF, and Foreach statements. The properties of a policy include id, period, and name. The period string indicates the polling cycle of the interpreted policy agent. For instance, when the following policy is transferred to a policy agent, the agent checks all the rules in this policy every 900 seconds as illustrated by “policy.period” in Figure 4-12. The id property can be used to control the running and stopping of the agent.

```
policy discomfort_index_control {
    policy.id = "org.skl.pdl.names.policy-id:1"
    policy.period = "900"
    policy.name = "Discomfort-Index-Control Policy"
    ...
}
```

Figure 4-12. Example code of a policy in EPDL

Rule

Rules are the fundamental elements of EPDL. All the elements defined in the policy are compiled into a list of rules. All rules share the same format:

rule name when{ ConditionSet }then{ ActionSet } (4-6)

In the above sample rule statement, the name property is optional. All the conditions and their relations are represented in ConditionSet. ActionSet defines the actions that should be executed when ConditionSet is evaluated to true. To express complex contexts consisting of many conditions, ConditionSet is described recursively and may contain other ConditionSets or conditions.

```
rule r1 when { building.floors["3F"].spaces["3L-5"].sba_get_discomfortIndex > 21.0
        && building.floors["3F"].spaces["3L-5"].sba_get_temperature < 22.0
        }then {
            building.floors["3F"].spaces["3L-5"].appliances["AirConditioner1"].sba_ctr_ac_off
            building.floors["3F"].spaces["3L-5"].appliances["AirConditioner2"].sba_ctr_ac_off
        }
```

Figure 4-13. Example code of a rule in EPDL

Figure 4-13 shows an example code of a rule: when the discomfort index is larger than 21.0 and the temperature is less than 22.0 in room “3L-5” on the third floor, turn off “AirConditioner1” and “AirConditioner2” in this room. The ConditionSet of this rule contains two conditions, and the logic “And” indicated by “&&” shows the relation between the two conditions. The left part of a condition and the action statement should be input with the help of smart building resource descriptor because most users have no knowledge of what information can be acquired and what controls can be performed in a smart building. In Figure 4-13, the user should input “building” and select the third floor, the space 3L-5, finally the “sba_get_temperature,” which is described in the

resource descriptor, for referring to the service that obtains the real-time temperature of room 3L-5.

IF Statement

The IF block decides whether the internal code should be compiled or not. Only the static properties are allowed to be referred to in the condition part of an if statement, because the condition part of an if statement is checked only at the compiling stage. For example, in the condition part of the if statements in Figure 4-14, the service "sba_get_temperature," which acquires the real-time temperature value of space "3L-5," is not allowed, but "type" attribute is allowed to be referred here. In addition, it is meaningless for the compiler to acquire the real-time temperature at the compiling stage.

```
if(building.floors["3F"].spaces["3L-5"].type == "student_room"){
    rule r3 when {building.floors["3F"].spaces["3L-5"].sba_get_PeriodElectricityConsumption("00:15:00") > 0.30
                }then {
                    building.floors["3F"].spaces["3L-5"].appliances["AirConditioner1"].sba_ctr_ac_off
                    building.floors["3F"].spaces["3L-5"].appliances["AirConditioner2"].sba_ctr_ac_off
                }
}
```

Figure 4-14. Example code of if statement in EPDL

Foreach

When users want to apply the same rule to multiple spaces or appliances, the foreach statement can be useful and convenient. The example code in Figure 4-15 shows the usage of the foreach statement: turn off all the air conditioners in room "3L-5" when the electricity consumption in the last 15 minutes in this room is greater than 0.30 kWh, and turn off all the lights when the electricity consumption is greater than 0.50 kWh. Rule "r4" is applied to all the air conditioners and "r5" is applied to all the lights in the room "3L-5".

```
foreach app within building.floors["3F"].spaces["3L-5"].appliances {
    if(app.type == "air_conditioner"){
        rule r4 when {building.floors["3F"].spaces["3L-5"].sba_get_PeriodElectricityConsumption("00:15:00") > 0.30
                }then {
                    app.sba_ctr_ac_off
                }
    }
    if(app.type == "light"){
        rule r5 when {building.floors["3F"].spaces["3L-5"].sba_get_PeriodElectricityConsumption("00:15:00") > 0.50
                } then {
                    app.sba_ctr_light_off
                }
    }
}
```

Figure 4-15. Example code of a foreach statement in EPDL

4.4.5 EPDL Compiler

Compiler

A compiler [112] is usually a set of programs that translate source code that can be easily read by human to another language that can be easily understood by a computer. The former is usually a high-level programming language (e.g., C/C++, and Java) for programmers, while the latter is usually a lower level language (e.g., assembly language) for machines. High-level programming languages provide extra functions and simplify the programming procedure. In this context, EPDL is a high-level language and DCRDL is a low-level language. The major tasks of a compiler include the following:

- Lexical analysis;
- syntactic analysis; and
- code generation.

Lexical analysis processes the program source code roughly and divides it into tokens such as keywords, literals, and punctuations. Syntactic analysis extracts meaningful elements from source code and ensures that no grammar rule is violated.

JavaCC and JJTree

"Java Compiler Compiler (JavaCC) is a popular parser generator" [113] for LL(k) grammars. JJTree is "a preprocessor for JavaCC, which inserts parse tree building

actions" [114] into the JavaCC grammar file by translating a JJTree grammar file (*.jjt) to a JavaCC grammar file (*.jj). For example, the following two figures show JJTree and JavaCC grammar for the condition element of EPDL, respectively.

```
void AndCondition() :
{Token t;}
{
  Condition()(t=< AND >{jjtThis.value = String.valueOf(t.image);} Condition())*
}

void Condition() :
{}
{
  (< LPAR >OrCondition() < RPAR >|UnaryExpression())(RelationSign() Literal())*
}

void UnaryExpression() :
{}
{
  LOOKAHEAD(SbaApi())SbaApi()
  |Identifier()
}
```

Figure 4-16. JJTree grammar for condition element of EDPL (EpdlParser.jjt)

```
void Condition() :
{/*@bgen(jjtree) Condition */
  PdlNode jjtn000 = new PdlNode(JJTCONDITION);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/}
{/*@bgen(jjtree) Condition */
  try {
/*@egen*/
  (< LPAR >OrCondition() < RPAR >|UnaryExpression())(RelationSign() Literal())*/*@bgen(jjtree)*/
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      throw (RuntimeException)jjte000;
    }
    if (jjte000 instanceof ParseException) {
      throw (ParseException)jjte000;
    }
    throw (Error)jjte000;
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}
```

Figure 4-17. JavaCC grammar for condition element of EDPL (EpdlParser.jj)

Grammar File

There are many parser generators for creating a source code parser. In this research, we adopt JavaCC which generates code parser for EPDL. JJTree allows users to define grammars in a manner similar to Extended Backus-Naur Form (EBNF) [115] for JavaCC. The entire JJTree grammars for EPDL written in EBNF-like fashion are available in [116] and Appendix 2. We present the EBNF notations of the main terms of EPDL as follows [106]:

<*Policy*> := ' policy ' <*Identifier*> ' { ' <*PolicyAttributeDefine*>*|<*Rule*>*
|<*RuleForeach*>*|<*IfStatement*>* ' } '

<*PolicyAttributeDefine*> := ' policy. ' <*Identifier*> ' = ' <*Literal*>

<*Literal*>:= <*INTEGER_LITERA*>|<*FLOATING_POINT_LITERAL*>
|<*CHARACTER_LITERA*>|<*STRING_LITERAL*>

<*Identifier*> := <*LETTER*>|(<*LETTER*>|<*DIGIT*>)*

<*Rule*> := ' rule ' <*Identifier*> 'when{'<*ConditionSet*> '}then {'<*ActionSet*> ' }'

<*RuleForeach*> := ' foreach' <*Identifier* >' within' <*SbaApi*> '{' <*IfStatement*>*|<*Rule*>* '}'

<*IfStatement*> := 'if (' <*ConditionSet*>'){' <*Rule*>*|<*RuleForeach*>* '}'

<*ConditionSet*> := <*OrCondition*>

<*OrCondition*> := <*AndCondition*>('||'<*AndCondition*>)*

<*AndCondition*> := <*Condition*>('&&'<*Condition*>)*

<*Condition*> := ('('<*OrCondition*>')'|<*UnaryExpression*>) (<*RelationSign*><*Literal*>)*

<*UnaryExpression*> := <*SbaApi*>|<*Identifier*>

<*RelationSign*> := '>'|'>='|'<'|'<='|'=='|'!='

<SbaApi> := 'building'|<*MemberIdentifier*> ('. '<*MemberIdentifier*>)*[<*Arguments*>]

<*MemberIdentifier*> := <*Identifier*>['[' <*Literal*> ']']

<*Arguments*> := '(' [<*Literal*> (','<*Literal*>*)] ')'

4.5 Summary

In this chapter, we introduced in detail the design of our framework. We designed a PAS in section 4.2. The peak demand control issue was also studied and designed as a

policy agent. In section 4.3, we presented DCRDL: a device control rule definition language for context-based device control in smart buildings. Policies written in DCRDL can be interpreted into policy agents and run in PAS. We explained the design details of EDPL and supporting components in section 4.4. A resource representation schema called the smart building resource descriptor was also presented in section 4.4.3. The descriptor is a key supporting component for supporting non-expert users in writing energy control policies with EPDL.

5 System Implementation

We will introduce the implementation details of the proposed programming framework in this chapter. The framework consists of three main components: PAS, DCRDL, and EPDL. These components are primarily developed using the Java programming language, and the total Java code is about 13,500 lines. The DCRDL and Smart Building Resource Descriptor can be written in the XML or YAML format. We implemented parser prototypes for the XML format.

5.1 Policy Agent System

5.1.1 Smart Building API

The Smart Building API is a set of RESTful interfaces deployed in the Daiwa Ubiquitous Computing Research Building (hereinafter referred to as “DUCRB”) for data access and device control. There are API servers that host the services for public access in this building. These APIs follow the wildly successful REST model. Data can be accessed and devices can be controlled through the Web using HTTP methods such as GET, PUT, POST, and DELETE. Any programming language that supports HTTP protocols can be used to develop automation applications at the client side for IoT-enabled smart buildings. These APIs are introduced as follows:

- Sensors: obtain building context information through the Smart Building API including temperature, humidity, $PM_{2.5}$ (i.e., particulate matter with a diameter of less than 2.5 micrometers [18]), occupants, human location, etc.
- Lighting System: control lights (i.e., turn lights on/off), and query the on/off status of lights through the Smart Building API.
- Air conditioner: fetch status information and perform fine-grained controls on target air conditioner through the Smart Building API.
- Elevator: move elevator to target floor through the Smart Building API.
- Smart Meter: get real-time or historical electricity usage of a measured device or unit through the Smart Building API.

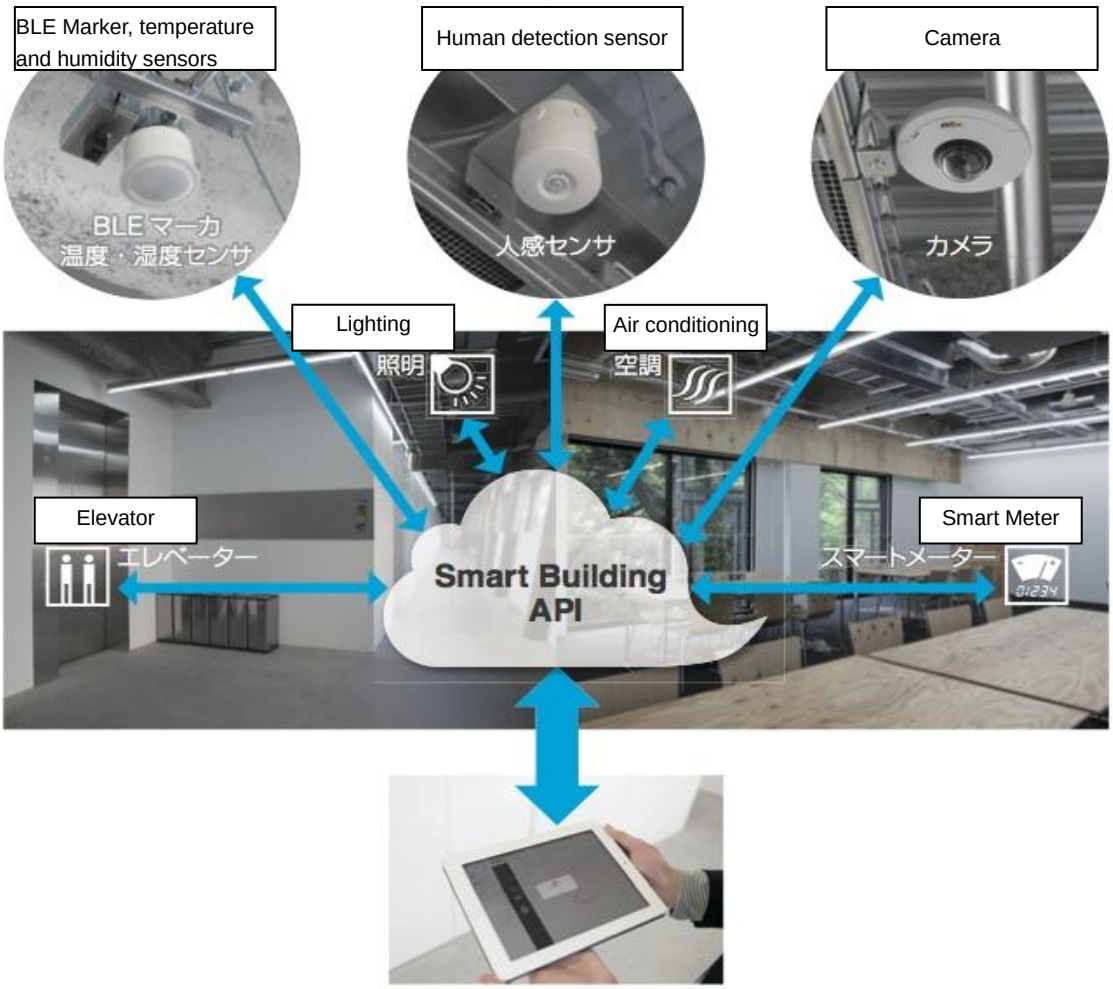


Figure 5-1. Smart Building API [30]

We use an air conditioner as an example to illustrate how the Smart Building API works. Methods used for air conditioner operations are:

Obtain Status	**GET**
Control	**PUT**

The parameters for controlling an air conditioner are shown in the following table:

Table 5-1. Structure of JSON data of the PUT Request for controlling air conditioners [55]

Item	Value	Description
url	/api/v1/air_conditioners/{id}.json	Path of the Smart Building API for air conditioners
id	Integer	The identifier (ID) of target air conditioner in the building system
setting_bit	Integer (4 bytes) Enable/disable (1:enable, 0:disable)	**Operations:** Bit 0: On/Off Bit 1: Operation mode Bit 2: Ventilation mode Bit 3: Ventilation amount Bit 4: Set-point Bit 5: Fan Speed Bit 6: Fan Direction
set_point	Float	Set-point of air conditioner
on_off	Integer	0:Off, 1:On
operation_mode	Integer	1:Fan, 2:Heat, 4:Cool, 16:Dependent, 64:Dry, 128: Auto
ventilation_mode	Integer	1:Automatic, 2:Heat Exchange, 4: Bypass
ventilation_amount	Integer	1: Normal, 2: Low, 4: High, 8: Automatic, 16: Low, 32: High
fan_speed	Integer	0: Low, 1: Middle, 2: High
fan_direction	Integer	0 - 4: Directions, 7: Swing
filter_sign_reset	Integer	1: Reset

For example, to turn on an air conditioner with an ID of 49 and with a set-point of 23.0, the following JSON data is sent to http://IP/api/v1/air_conditioners/49.json (where IP is the IP address of the API server):

```
{"air_conditioner":
    {"id":49, "setting_bit":0x11, "on_off":1, "set_point":23.0, "operation_mode":0,
"ventilation_mode":0, "ventilation_amount":0, "fan_speed":0, "fan_direction":0, "filter_sign_reset":0}
}
```


If we access "/api/v1/air_conditioners/49.json" using the HTTP GET method, JSON response data is returned as described in the following Table 5-2:

Table 5-2. JSON response data after querying the status of an air conditioner [55]

Item	Value	Description
address	Integer	The identifier (id) of target air conditioner in the building system
status	Integer	1: Normal, 0:Error, -1:Unconnected
malfunction_code	Integer	
on_off	Integer	1:On, 0:Off
operation_mode	Integer	Refer to above table, 0:Unknown
ventilation_mode	Integer	Refer to above table, 0:Unknown
ventilation_amount	Integer	Refer to above table, 0:Unknown
enable_disable_temp	Integer (4 bytes)	Bit 0: if 1 Set Temp. enable Bit 1: if 1 Room Temp. enable
room_temp	Float	Real temperature of the room
set_temp	Float	Set-point of air conditioner
fan_speed	Integer	Refer to above table, -1: Unknown
fan_direction	Integer	Refer to above table, -1: Unknown
filter_sign	Integer	0:On, 1:Off
name	String	Name of the air conditioner

5.1.2 Local Java Library of the Smart Building API

We implemented a local Java library of RESTful APIs for the evaluation of PAS. The *SmartBuildingAPI* class is a wrapper for *SensorController, LightController, SmartMeter,* and *ACController* classes. Users invoke the Smart Building API with a given ucode, which is a unique identifier for a target object/space. The Smart Building API changes ucodes to the IDs used by different vendor sub-systems. *HTTPRequestHelper* is a static

class with three static methods for sending HTTP GET, PUT, and POST requests. The class dependency of the local Java library is shown in the following figure:

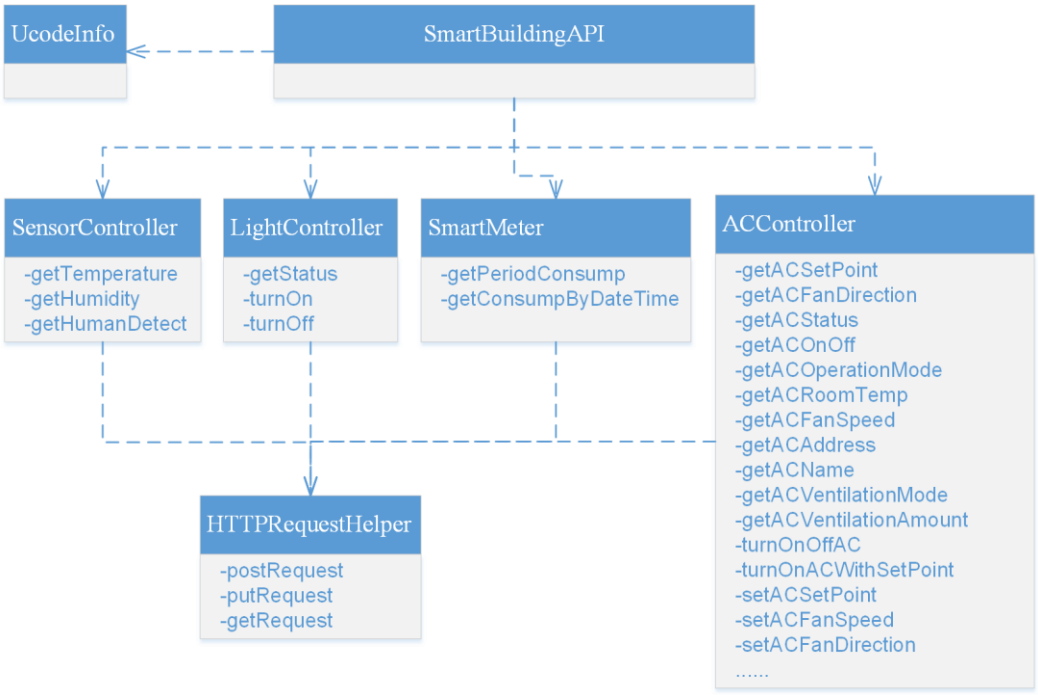


Figure 5-2. Class dependency of the local Java library for the Smart Building API

We introduce APIs for each type of devices as follows:

- Smart Meter API: The electricity consumptions at different levels (e.g., building, floor, space, device level, etc.) are being recorded by the Smart E-Power Meter named "Data Logger Light," which is a product of Panasonic Company. Some query APIs are deployed on the API server. Users can obtain real-time historical electricity consumption for each measured device or unit through these APIs.
- Air Conditioner API: The air conditioner system is controlled by "Intelligent Touch Controller," which is produced by Daikin Industries, Ltd. Some RESTful APIs have been deployed on the API server to provide querying and controlling services for remote users. Users can get the property and the status of the connected air conditioners by sending HTTP GET requests. Users can control the connected air conditioners by sending HTTP PUT requests.
- Light and Sensor API: Status information is collected from sensors and lights and

stored in a data server. The API server may query the data server when users try to get the status information of a device.

5.1.3 Architecture of PAS

We implemented the peak control policy as a policy agent using the local Java library. The architecture of this system is shown in the following Figure 5-3:

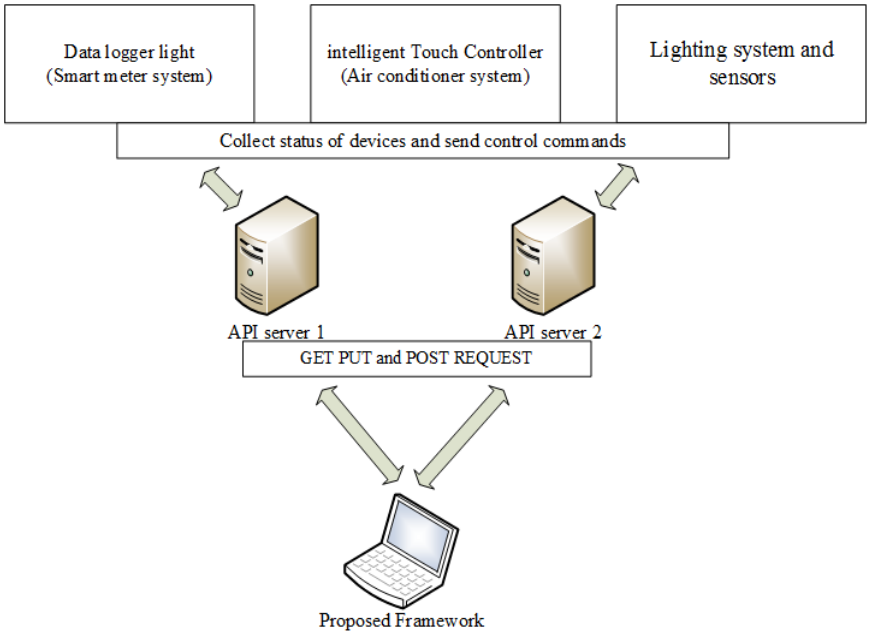


Figure 5-3. Architecture of the peak demand control framework

The system was developed using Eclipse 4.4.0 [117], and Figure 5-4 presents the source tree of this system. The interpreter parses predefined rules (i.e., the constraints defined in a policy file in XML format) and stores the rules in the system.

The context monitor dynamically changes the priority of devices according to real-time context. The context monitor is also responsible for finding a set of suitable rules from the system and sending them to the executor. The rules are selected with the help of JaCoP [118], which is a constraint solver implemented in Java. We adopted a simple backtracking search algorithm for selecting rules.

The executor is responsible for evaluating rules that are selected by the context monitor. The executor will turn off the target devices or let them run at a lower

electricity consumption level without breaking the constraints defined in the policy.

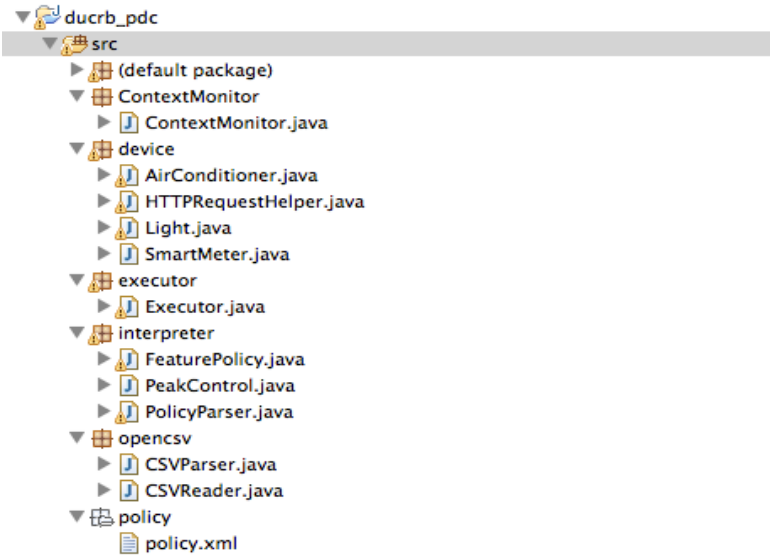


Figure 5-4. Source tree of the peak demand control agent

5.2 DCRDL

In order to enable the flexible, precise, and collaborative control of the smart building system, we extended the peak demand control framework [67] and implemented a rule-based energy management framework for smart buildings. A descriptive language was also designed to enable users to write control rules for the framework. This framework allows rules that are written in DCRDL to be executed to control smart devices for optimizing energy efficiency in smart buildings. It hides the lower-layer building structure and service semantics so the upper-layer can write simple rules to control the smart building system. We focus on enabling users to write rules to describe control logic in DCRDL.

5.2.1 System Overview

The architecture of DCRDL is shown in Figure 5-5. The DCRDL was implemented using the Java programming language with approximately 4,400 lines of code. The architecture consists of rule files, an interpreter, and a policy agent system.

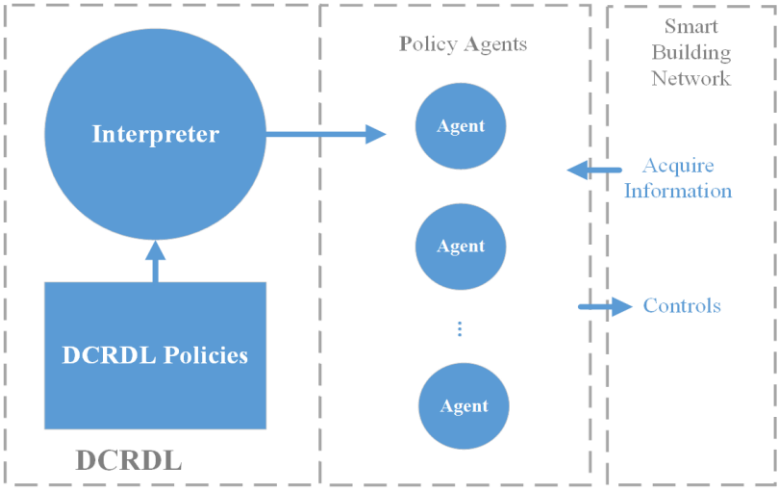


Figure 5-5. Architecture of DCRDL

Rules Source File (DCRDL rules)

Users can define fine-grained rules using the DCRDL for each device in the source file, and a set of rules can be embedded in a policy element to achieve a specific energy control objective. We will design a high-level policy description language named EPDL in the future to ease the policy writing for IoT-enabled smart buildings.

The Policy Parser

The policy parser is implemented in a package called interpreter. The policy parser reads policy files written in DCRDL and parses policies to elements defined in the *org.sakamura-lab.dcrdl.model* package. The classes in this package include Policy, Rule, ConditionSet, ActionSet, Condition, Action, Subject, Target, Function, and Parameter. The details of these elements are introduced in section 4.3.4.

Interpreter

The interpreter translates policies into policy agents (Java Threads). The period attribute in the policy element indicates the polling cycle of this agent. According to the design of DCRDL, the agents also can be run in event-driven mode if the data platform provides event mechanisms for the programming environment. Currently, the data

platform of DUCRB does not support event mechanisms, so we have not implemented an event-driven mode for the DCRDL interpreter.

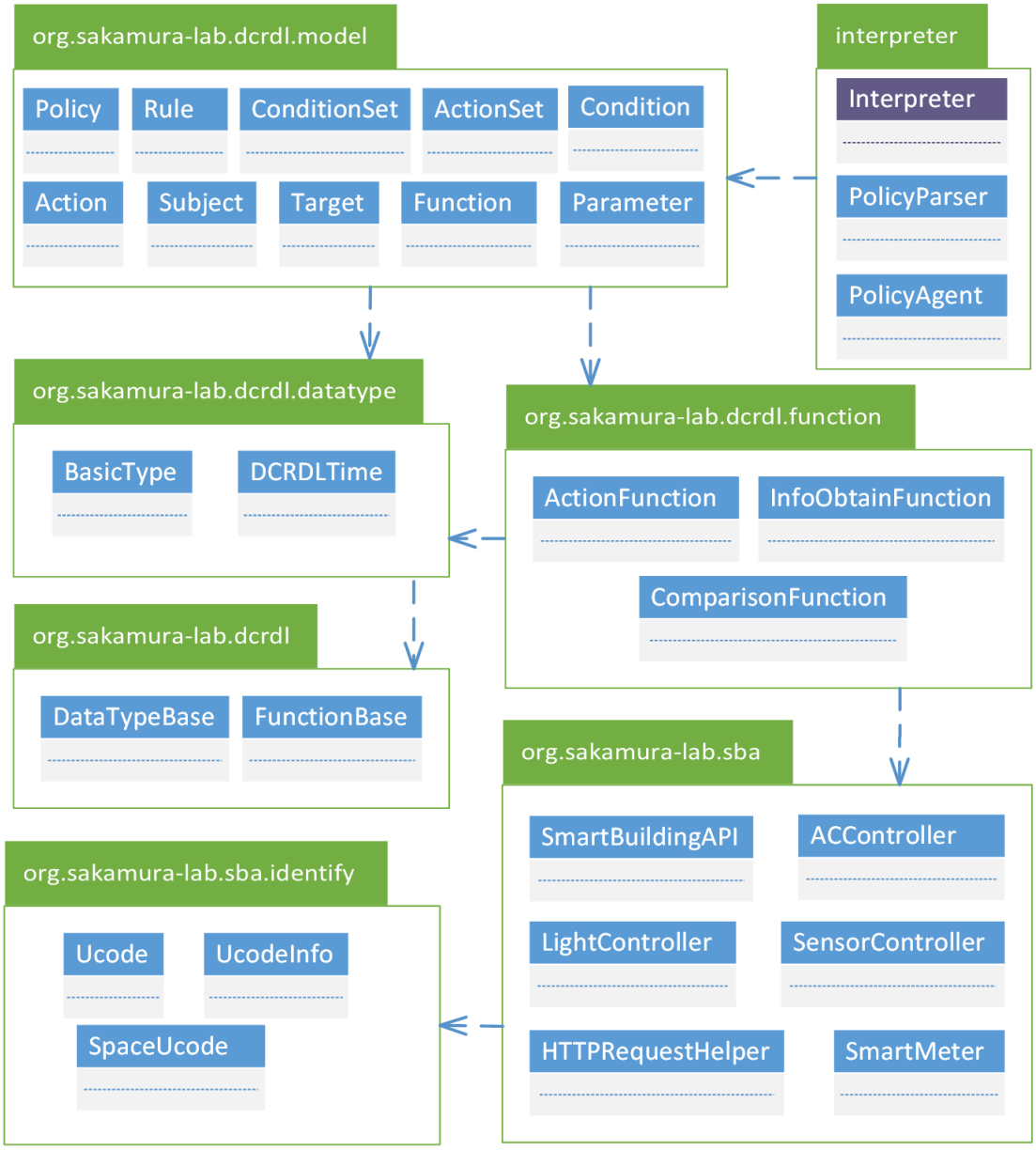


Figure 5-6. Package dependency of DCRDL

Policy Agent

The policy agent checks all the rules defined within the policy and performs control actions. When the ConditionSet section of a rule is evaluated to be true, the actions defined in the ActionSet section will be executed by the parent policy agent.

Figure 5-6 presents the classes, packages, and package dependencies of the prototype system we implemented using Java for DCRDL. Language-related packages are defined in *org.sakamura-lab.dcrdl*. The interpreter parses the policies and rules defined in a source file using the models defined in the *model* package. These models depend on the *datatype* and *function* packages. The *function* package implements the Smart Building API component, which is a key part of the DCRDL interpreter. It invokes Smart Building APIs that are defined in the package *org.sakamura-lab.sba*. The *InfoObtainFunction* gets context information from the building system, while the *ComparisonFunction* evaluates conditions by comparing the acquired information with user-defined values. The *ActionFunction* defines the semantics of the controlling APIs. The *org.sakamura-lab.sba* package implements the function semantics of data acquisition and device control, which is the local java library we implemented, as discussed in section 5.1.2. For example, a DCRDL function

org:sakamura-lab:names:dcrdl:1.0:function:get-space-temperature

will invoke the method that implements *SensorController* to obtain the temperature of the target space (see Figure 5-6).

5.2.2 Smart Building API Modeling

Smart Building API modeling is a key design of DCRDL. It is implemented in the *org.sakamura-lab.dcrdl.function* package. This component translates the user description of a function into executable code. In other words, this component is the connection between user descriptions and real services provided by the building system. Functions in DCRDL are classified into three categories: comparison functions, information acquisition functions, and action functions.

Comparison Functions

Comparison functions are used to evaluate conditions. For example, in the condition shown in Figure 5-7, the *float-greater-than-or-equal* function represents “>=” logic for the float data. The “subject” and parameter value “24.0” are two components of this comparison function. The following code represents the condition that evaluates

whether the "subject" (i.e., the temperature of the target room: "X0001C00000000000020000000D648") is greater than or equal to 24^{o} C, or not. If it is greater than or equal to 24^{o} C, the condition is evaluated as true by the *float-greater-than-or-equal* function.

```
<Condition>
    <Subject>
        <Function FunctionId="urn:sakamura-koshizuka-lab:names:dcrdl:1.0:function:get-space-temperature">
            <Parameter Value="X0001C00000000000020000000D648"
                       DataType="urn:sakamura-koshizuka-lab:names:dcrdl:1.0:DataType:ucode" />
        </Function>
    </Subject>
    <Function FunctionId="urn:sakamura-koshizuka-lab:names:dcrdl:1.0:function:float-greater-than-or-equal">
        <Parameter Value="24.0"
                   DataType="http://www.w3.org/2001/XMLSchema#float"/>
    </Function>
</Condition>
```

Figure 5-7. Sample DCRDL code of a condition

The following Figure 5-8 shows code from the *ComparisonFunction* class. The constructor of this class takes three parameters:

- String functionName: function ID that defined in Device Control Rule Definition Language Specification [111] and Appendix 4;
- Object arg1: the left item of the conditional expression;
- Object arg2: the right item of the conditional expression.

A method *compare* is implemented to perform the evaluation of the condition. If the *compare* method returns true, the conditional expression is evaluated as true; otherwise it is evaluated as false.

```
38    public ComparisonFunction(String functionName, Object arg1, Object arg2){
39        this.functionName = functionName;
40        this.para1 = arg1;
41        this.para2 = arg2;
42    }
43
44    public boolean compare(){
45        boolean result = false;
46        switch(FunctionBase.getFunctionID(this.functionName)){
47
48        /* Integer comparison*/
49        case ID_INTEGER_EQUAL: {
50
51            int arg1 = (int)this.para1;
52            int arg2 = (int)this.para2;
53            result = (arg1 == arg2);
54            break;
55        }
56
57        case ID_INTEGER_GREATER: {
58
59            int arg1 = (int)this.para1;
60            int arg2 = (int)this.para2;
61            result = (arg1 > arg2);
62            break;
63        }
64
65        case ID_INTEGER_GREATER_OR_EQUAL: {
66
67            int arg1 = (int)this.para1;
68            int arg2 = (int)this.para2;
69            result = (arg1 >= arg2);
70            break;
71        }
```

Figure 5-8. Part of Java code of the ComparisonFunction class

Information Acquisition Functions

Information acquisition functions are modeled as the *InfoObtainFunction* class. They are used to retrieve context information from the building system. The context information is usually analyzed and abstracted from raw data obtained from sensors and devices. The following Figure 5-9 shows Java code of the *InfoObtainFunction* class. The constructor of this class takes two parameters:

- String functionName: function ID that defined in Device Control Rule Definition Language Specification [111] and Appendix 4;
- List <Parameter> paraList: contains all the arguments passed from the policy parser.

```
16    public InfoObtainFunction(String functionName, List<Parameter> paraList) {
17        super();
18        this.functionName = functionName;
19        this.paraList = paraList;
20    }
21
22    public Parameter getInformation(){
23
24        Parameter returnVal = new Parameter(null,null,null);
25        for(Parameter p : this.paraList){
26            Function f = p.getFunction();
27            if(f != null){
28                InfoObtainFunction iof = new InfoObtainFunction(f.getFunctionName(), f.getParaList());
29                p.setValue(iof.getInformation().getValue());
30            }
31        }
32        SmartBuildingAPI sba = new SmartBuildingAPI();
33        switch(FunctionBase.getFunctionID(this.functionName)){
34
35        case ID_GET_SPACE_TEMPERATURE: {
36            Parameter p = (Parameter)paraList.get(0);
37            returnVal.setDataType(BasicType.DCRDL_DATA_TYPE_FLOAT);
38            returnVal.setValue(Float.toString(sba.getSpaceTemperature(p.getValue())));
39            break;
40        }
41
42        case ID_GET_SPACE_OCCUPANCY: {
43            Parameter p = (Parameter)paraList.get(0);
44            returnVal.setDataType(BasicType.DCRDL_DATA_TYPE_INTEGER);
45            returnVal.setValue(Integer.toString(sba.getSpaceOccupancy(p.getValue())));
46            break;
47        }
48
```

Figure 5-9. Part of Java code of the InfoObtainFunction class

The *getinformation* method implements the actual semantics of the information acquisition function. For instance, if the function name that is passed from the policy parser is *org:sakamura-lab:names:dcrdl:1.0:function:get-space-temperature*, the method will execute the "ID_GET_SPACE_TEMPERATURE" branch. Parameters released from *paraList* and *sba.getSpaceTemperature* will be invoked to retrieve information from the Smart Building API server. The *sba* is an instance of the local Java library of the Smart Building API.

Action Functions

Action functions are responsible for performing control actions on target devices. The semantics of action functions are implemented in the *ActionFunction* class. The constructor is similar to that of an information acquisition function. Actual controls are performed in the *control* method.

```
26    public ActionFunction(String functionName, List<Parameter> paraList) {
27        super();
28        this.functionName = functionName;
29        this.paraList = paraList;
30    }
31
32    public void control(){
33        SmartBuildingAPI sba = new SmartBuildingAPI();
34        switch (FunctionBase.getFunctionID(this.functionName)){
35        case ID_CTR_AC_ON: {
36
37            Parameter p = (Parameter)paraList.get(0);
38            sba.controlACON(p.getValue());
39            break;
40        }
41
42        case ID_CTR_AC_OFF: {
43
44            Parameter p = (Parameter)paraList.get(0);
45            sba.controlACOff(p.getValue());
46            break;
47        }
48
49        case ID_CTR_AC_ON_WITH_SETPOINT: {
50
51            Parameter p1 = (Parameter)paraList.get(0);
52            Parameter p2 = (Parameter)paraList.get(1);
53            sba.controlACOnWithSetpoint(p1.getValue(), p2.getValue());
54            break;
55        }
```

Figure 5-10. Part of Java code of the ActionFunction class

5.3 EPDL

To help energy policy designers write energy control policies with EPDL and compile them in DCRDL, we implemented a set of supporting tools using Java. The supporting tool set consists of a resource descriptor parser, an Eclipse [117] editor plugin as the EDPL editor, and a compiler. The implementation details of the policy-processing environment are provided in the PAS and DCRDL sections. The total Java code of EPDL is approximately 9,000 lines.

5.3.1 Smart Building Resource Descriptor Parser

Seven classes are defined in the smart building resource descriptor (hereinafter referred to as “resource descriptor”) parser: *BuildingBase*, *Building*, *Floor*, *Space*, *Appliance*, *SBApi*, and *Parameter*. The *Building*, *Floor*, and *Space* classes model the physical structure of the building and the available services within them. All these classes extend the *BuildingBase* class in which containers of services and properties are defined. The *Appliance* class models all the electrical appliances that are deployed in the

building. The *SBApi* class models the services (i.e., the smart building APIs). The *parameter* class models the input data of a service. The output of this parser is an instance of the *Building* class, which contains the information about the spaces and objects in the target building. The parser parses the information that is defined in the resource descriptor, and the parsed information is used to support smart building users in writing and compiling energy control policies written with the EPDL. The class diagram of the resource descriptor parser is presented in Figure 5-11.

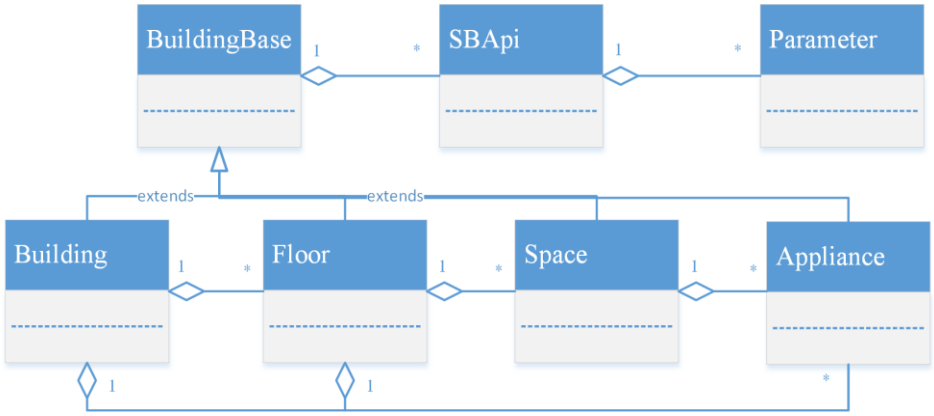


Figure 5-11. Class diagram of the smart building resource descriptor parser

5.3.2 The EPDL Editor Plugin

To assist users in writing EDPL energy control policies, we implemented an EDPL editor using Eclipse Plugin, which provides information and suggestions to users during the writing process. The editor plugin parses the resource descriptor with the parser, computes suitable code completion proposals, and provides completion proposals to the user. When the user inputs a string that represents a space or an appliance, the editor provides all the related service names and object names in a selection menu for users (see figure 5-12).

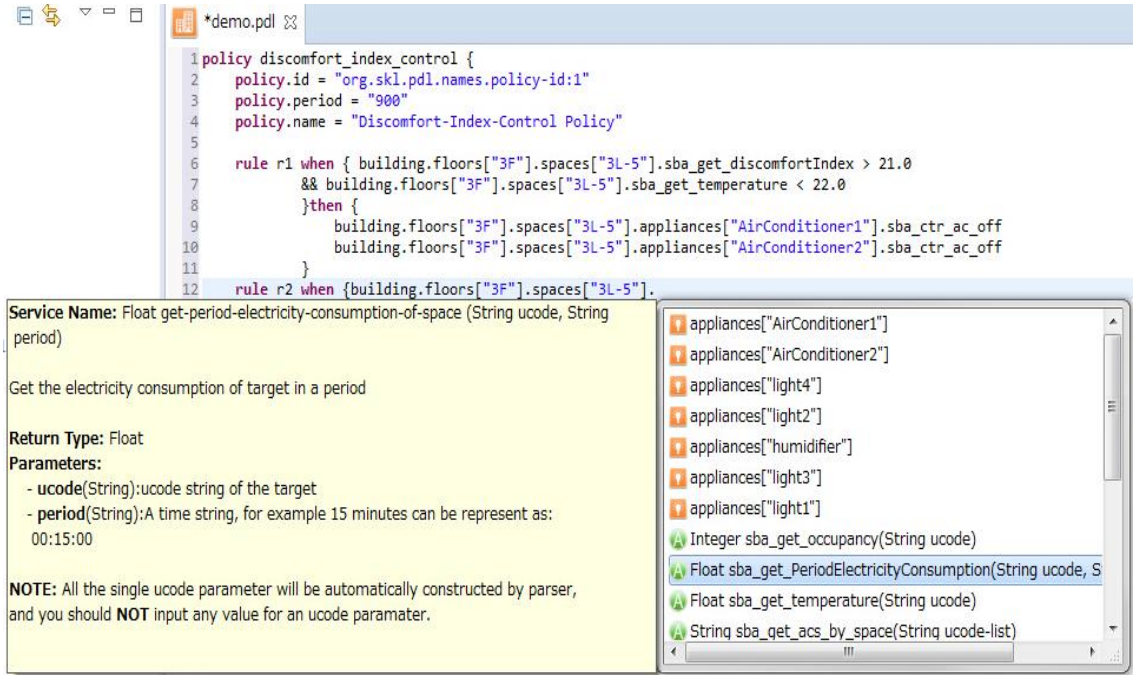


Figure 5-12. Example of a pop-up selection menu. When users type a string (e.g., "building.floors['3F'].spaces['3L-5'].."), the editor displays the available sub-level objects, spaces, and services. The yellow tip shows the detailed information about the selected item.

JFace

The EPDL editor plugin was implemented using JFace [119]. JFace is a sophisticated UI toolkit with SWT-based [120] views for "handling common Java UI programming tasks" [119] for Eclipse users. It abstracts and simplifies visual components (e.g., table viewer, tree viewer, text viewer), and provides an assistant layer for managing system resource efficiently (e.g., image, color, font). SWT is a fundamental part of JFace and developers can invoke SWT APIs directly when developing applications with JFace. The JFace resource management and text components are used in the EDPL editor plugin. The source code tree is shown in Figure 5-13 .

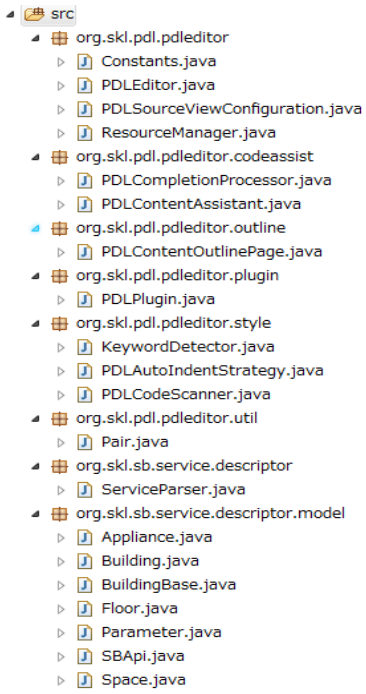


Figure 5-13. Source code tree of the EDPL editor plugin

Code Completion

Code completion is the key function of the EPDL editor and is the fundamental reason for developing this editor. As shown in Figure 5-12, when users input a string *building.floors["3F"].spaces["3L-5"]* ending with a dot ".", the editor plugin provides a pop-up selection menu of all the referable properties, services, and sub-level spaces or appliances. After selecting an item in the menu, a proper completion string will be determined by the editor instead of requiring users to type the full name of objects or services manually.

Code Coloring

In addition to the code completion function, we also implemented code coloring, which shows different colors for string, keyword, and comments. Additional features, including error recovery, code hyperlinks, and others, will be supported in future versions. The keywords of EPDL include *policy*, *rule*, *when*, *then*, *foreach*, *within*, and *if*.

5.3.3 EDPL Compiler

As we introduced in chapter 4, a compiler [112] is a set of programs that translate source code which can be easily read by human to another language which can be easily understood by a computer. Therefore, the main job of the EPDL compiler is to parse the EDPL rule entities and translate them into DCRDL rules. A key step involves translating user-readable resource expressions into resource formats defined in DCRDL. For example, expression 5-1 will be translated into expression 5-2 as shown in the following, which is a function ID defined in DCRDL format. These expressions illustrate the service for obtaining the real-time temperature of a space (here, the space is *spaces["3L-5"]*). The compiler translates the corresponding EPDL elements into DCRDL elements.

building.floors["3F"].spaces["3L-5"].sba_get_temperature *(5-1)*

uri:sakamura-koshizuka-lab:names:dcrdl:1.0:function:get-space-temperature *(5-2)*

The EPDL compiler consists of a source code parser, a semantic analyzer, and a rule generator. As shown in Figure 5-14, the code parser is implemented using JavaCC and JJTree. We have introduced JJTree and JavaCC in chapter 4, and will not introduce them here any more. We developed a grammar file named "PdlParser.jjt" [121] as input for JJTree to produce a JavaCC grammar file "PdlParser.jj" [122]. Finally, JavaCC generates Java classes for the code parser using "PdlParser.jj."

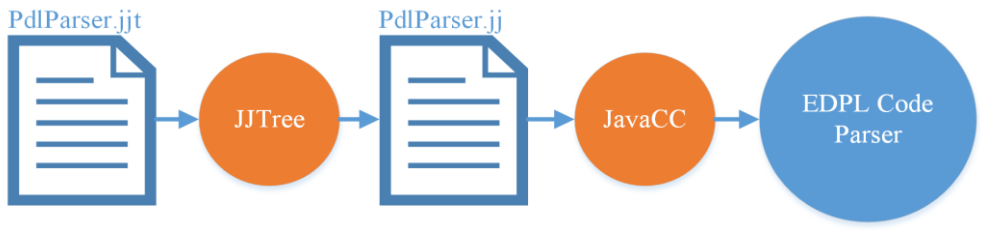
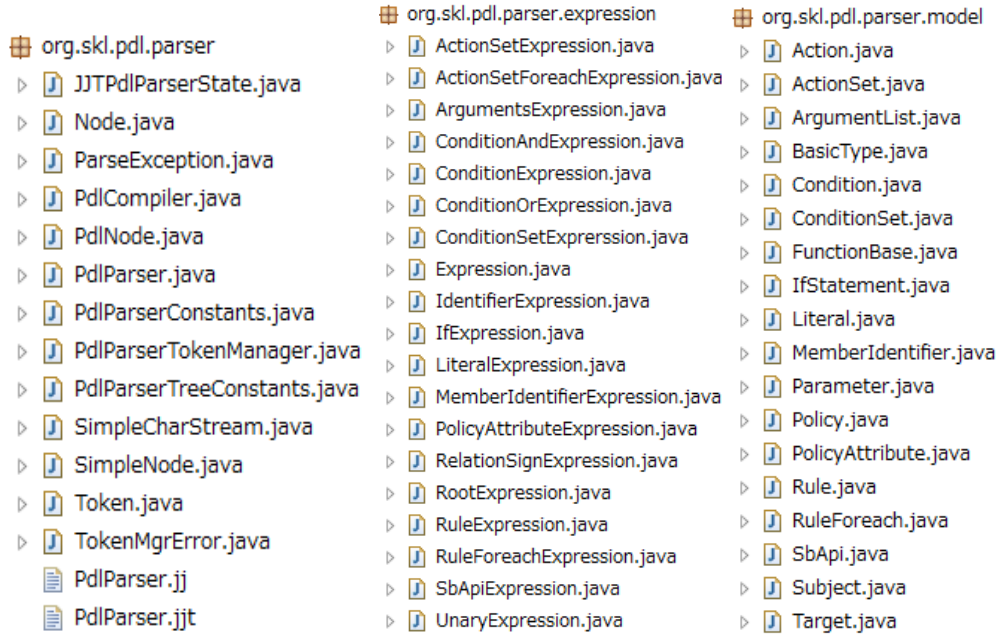


Figure 5-14. Implementation of EDPL code parser

The output of the source code parser is an Abstract Syntax Tree (AST). Figure 5-15b shows the classes of node visitors for the AST. The semantic analyzer parses the AST

using the node visitors and data models defined in Figure 5-15 to get a list of EPDL rules. Finally, the compiler translates all the rules into the DCRDL format.



a. The parser package b. Node visitors c. Data model

Figure 5-15. Source code tree of the compiler

5.4 Summary

In this chapter, we introduced the implementation details of three components of our proposed framework. Section 5.1 discussed how a peak demand control policy was implemented as an agent to model the peak demand issue of smart buildings. It presented the fundamental concepts of the Policy Agent System. Section 5.2 described the implementation of DCRDL. We developed the DCRDL interpreter that parses DCRDL rules into a Policy Agent, which is a Java thread. The thread checks each rule that is defined in a policy, within a specified period, and executes the actions defined in the rule if the corresponding ConditionSet is evaluated to be true. We implemented a modeling component for the Smart Building API in the DCRDL interpreter that translates user descriptive expressions into executable code. The implementation of EDPL was presented in section 5.3. We also developed a smart building resource descriptor parser, an EPDL editor, and an EPDL compiler, as discussed in this section.

6 EVALUATION

This chapter outlines experiments conducted and the results obtained for three components of the proposed framework. To enable non-expert users to program smart buildings, the following two issues have to be addressed:

- Separate upper-layer user application control logic from lower-layer building structure and service details, and hide lower-layer details. Further, building resources representation schema should be designed for both upper-layer and lower-layer users. As a result, upper-layer users can focus on describing control logics and lower-layers can focus on developing services.
- Provide simpler programming languages and tools. The services developed by lower-layer users should be represented in a human readable format for upper-users to read and refer to, and descriptive control logics should be translated back to executable codes which is usually wrote using procedural programming languages.

The objective of the evaluation was to verify whether the above issues have been addressed or not.

6.1 Evaluation of the Policy Agent System

6.1.1 Objective

We implemented only one agent for peak demand control to run as the Policy Agent System (PAS). It was run as a basic energy policy in the building management system. The objective in evaluation of the peak demand control framework was to examine the practicability of the PAS and control demand peak with writing a set of rules for the building automation system. We observed and compared electricity consumption patterns (i.e., The pattern before applying the policy and that after applying the policy) to evaluate the efficacy of the peak demand control framework.

6.1.2 Experimental Setup

Experimental Environment

Owing to security issues, we selected two rooms on the third floor in DUCRB to simulate the overall building: room 3L-5 and room 3L-6. The rooms consisted of four air conditioners and eight sets of lights. An application developed using a local Java library of Smart Building API was executed on a MacBook Pro to control them. The setup parameters used are given in Table 6-1.

Table 6-1. Experimental environment of peak demand control framework

Target Rooms	3L-5, 3L-6	Simulated as the whole building
PC	MacBook Pro Mid 2012	
OS	OS X 10.8	
Network	DUCRB-UBI	WIFI SSID
Java	1.7.0_60	
Eclipse	Luna Release 4.4.0	

Constraints (Rules definition)

Table 6-2. Pre-defined constraints of the peak demand control framework

Items	**Constraint**	**Remark**
Air Conditioners	[20-25] (°C)	Allowed set-point range
Lights	[0, 1]	0: off, 1: on
Human Detection	If a person is present, the lights in this space should not be turned off.	
Server Room	If there is a server, air conditioners in this room should not be turned off.	
Electricity Consumption Recording Interval	15 minutes	The electricity consumption is logged in every 15 minutes.
Peak Value	140 kWh	

Context Monitor

A combination priority was initialized and monitored by the Context Monitor. This priority helped the framework to adjust the state of the appropriate devices according to the defined policies. There were both high and low priority levels. As stated above, high priority is defined by the user and cannot be changed at runtime. In this experiment, high priority was assigned to room priority, which is defined according to the property of the room. For example, we assigned a higher priority to the doctoral students' room than the master students' room. Low priority can be changed when the environmental situation changes. For example, when a person is not detected in an area for a while, the appliances deployed in that area should have a lower low priority. This causes the appliances in other areas of the same room to turn off or be adjusted to operate at a lower consumption level.

6.1.3 Experiment

The experiment was carried out in the second half of August. We recorded the electricity consumption pattern without applying the framework in the first week. The

electricity consumption pattern of the first week is shown in Figure 6-1 as "Daily Electricity Consumption 1 (kWh)"—the solid red line. Subsequently, we applied the peak demand control framework and recorded the electricity consumption pattern in the second week. It is shown as "Daily Electricity Consumption 2 (kWh)"—the blue line. The dashed red line illustrates the defined peak value (140 kWh) in Table 6-2. It signifies the instances where the total electricity consumption of the two rooms exceed 140 kWh, which results in some appliances being turned off or adjusted to run at a lower energy consumption level without breaking rules defined in Table 6-2. The result of the experiment is shown in Figure 6-1.

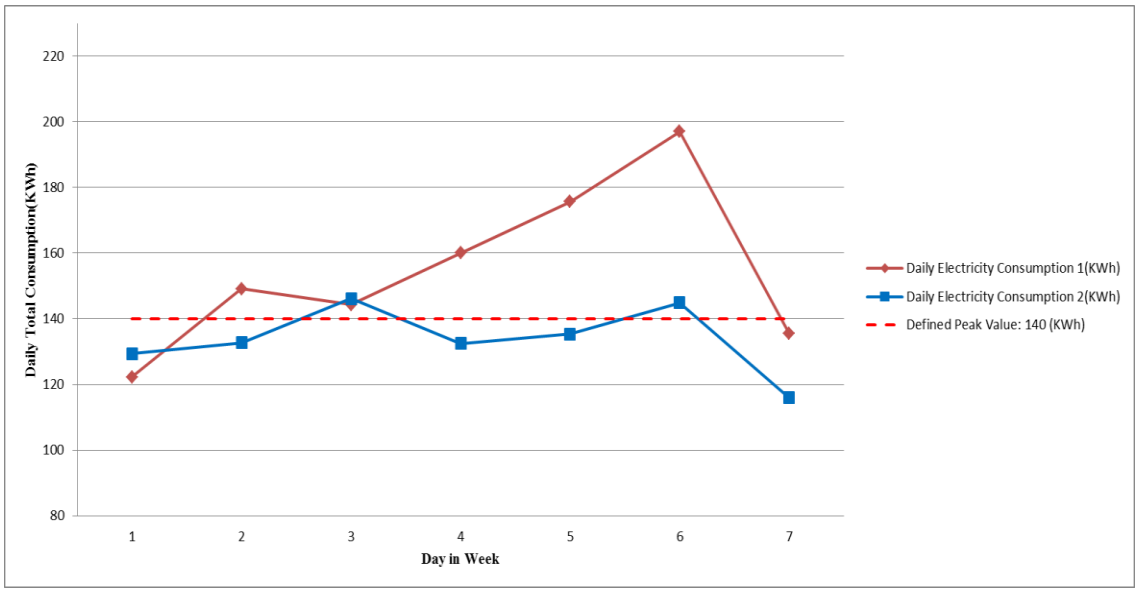


Figure 6-1. Evaluation result of the Peak Demand Control Framework

Owing to control system delay, the consumption may slightly exceed the defined peak. We observed this phenomenon in the above figure on days 3 and 6.

6.1.4 Result Analysis

Peak demand control: The experimental results show that this framework is able to balance the peak electricity demand in smart buildings. When the electricity demand reached a defined peak value, some devices in the target rooms turned off or were set at a lower electricity consumption level without breaking the constraints. This framework

applied the constraint programming model to find one or more devices and set them in an appropriate state according to the defined rules. It successfully verified that this framework can efficiently balance the electricity demand and verified that PAS is feasible.

Reduced electricity consumption: We conducted another experiment over a period of approximately one week with the objective of measuring how much electricity can be saved. Unfortunately, we witnessed no obvious electricity saving. The reasons for this result are as follows:

- Season and weather: Cooler days may result in less electricity consumption.
- Peak value: The definition of peak value also affects experimental results. A lower peak value may bring significant electricity energy saving. However, some devices may not normally function under a low peak value.
- The number of controllable targets.

6.2 Evaluation of DCRDL

6.2.1 Objective

To evaluate the expressivity of DCRDL, we wrote three policies in DCRDL to control the electricity consumption of devices for various purposes. The three policies were “Peak Consumption Control Policy,” “Tiered Electricity Pricing Policy,” and “Discomfort Index Control Policy.” We evaluated these policies within the real smart building environment—DUCRB in the winter and spring of 2015. We observed parameters such as temperature and electricity usage before the experiments and designed rules based on the observed values. The number of lines for each policy is shown in Table 6-3:

Table 6-3. Lines of code for each policy in DCRDL

Peak Consumption Control Policy	**155** lines
Tiered Electricity Pricing Policy	**124** lines
Discomfort Index Control Policy	**120** lines

Three case studies were conducted based on the above policies. The details of these case studies are given in the following sections. The objective of studying these cases is to show how device control can be personalized, precise and flexible by using the DCRDL.

6.2.2 Peak Consumption Control Policy

In this policy, we designed several rules to control the peak electricity consumption in a certain interval by lowering and even turning off the equipment in the DUCRB when the consumption exceeded a defined peak. We selected rooms 3L-5 and 3L-6, along with the central air conditioning system for the third floor as the control targets. The rules were defined as follows:

- **Rule 1**: WHEN the electricity consumption of the building in the last 15 minutes exceeds 0.6 kWh, THEN decrease the set-point of the air conditioners by 2 in the room where the temperature is the highest.
- **Rule 2**: WHEN the electricity consumption of room 3L-6 in the last 15 minutes exceeds 0.2 kWh, THEN turn off the lights in this room.
- **Rule 3-6**: For each air conditioner in these two rooms, WHEN the set-point is less than 22.0, THEN turn off this air conditioner.

The parameter values defined in the above rules are based on the observation before the experiment. The limit of the peak for the building in the last 15 minutes for our policy was 0.6 kWh. This policy was executed as a policy agent (Java Thread). The information obtaining function, “get-acs-by-spaces,” uses the return value of the function “get-space-by-highest-temperature” as the parameter to ascertain the running air conditioners in the room where the temperature is highest. Rules 1-6 are checked every 15 minutes. When the ConditionSet section of the rule is evaluated to true, the actions in the ActionSet section are executed. Owing to the space limitations of this chapter, we are unable to describe the rules written in the DCRDL format in detail. Figure 6-2 shows the electricity consumption pattern for every 15 minutes during the experiment.

The red dotted line in Figure 6-2 shows the user-defined peak value, which should

not be exceeded. The blue polyline shows the total consumption every 15 minutes after application of the Peak Consumption Control Policy, which was written in DCRDL. From the system log, we observed that the air conditioners in room 3L-5 were turned off by the policy in cycles 1 and 2, and the air conditioners in room 3L-6 were turned off in cycle 6. As result of these actions, the polyline is very steep in cycles 1, 2, and 6. To prove the efficiency of these rules, we turned on two air conditioners in the seventh cycle manually; they were promptly turned off by the policy agent in the eighth cycle. Thus, the results of this experiment verify that DCRDL is efficient and expressive.

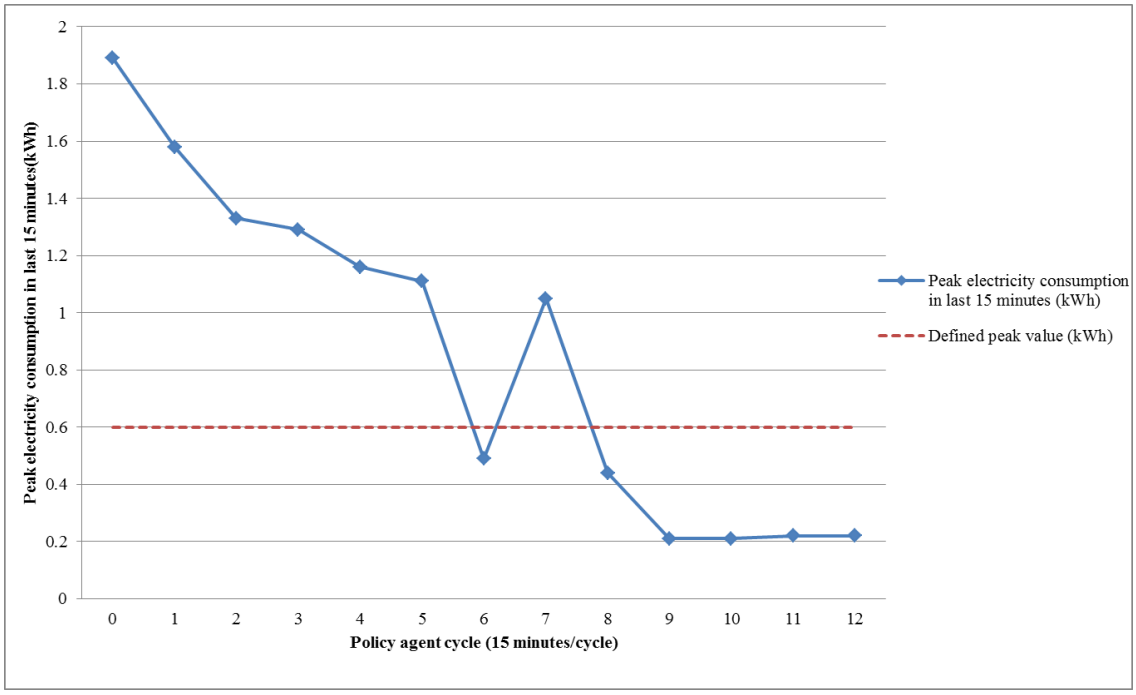


Figure 6-2. Electricity consumption pattern of the peak consumption control policy

6.2.3 Tiered Electricity Pricing Policy

This policy was designed to restrain the growth rate of the real-time electricity price of the building. The simulated tiered price for electricity used in this experiment is given in Table 6-4. We defined several rules in this policy as follows:

– **Rule 1:** WHEN the real-time electricity price is between 500 Yen and 1000 Yen, THEN decrease the set-point of the air conditioners by 2 in the room where the temperature is the highest.

- **Rule 2:** WHEN the real-time electricity price exceeds 1000 Yen, THEN turn off the air conditioners in the room where the temperature is the highest.
- **Rule 3:** WHEN the real-time electricity price exceeds 1000 Yen and the humidity of room 3L-5 is higher than 30%, THEN turn off the humidifier in this room.
- **Rule 4:** WHEN the real-time electricity price exceeds 1500 Yen, THEN turn off the lights in room 3L-6.

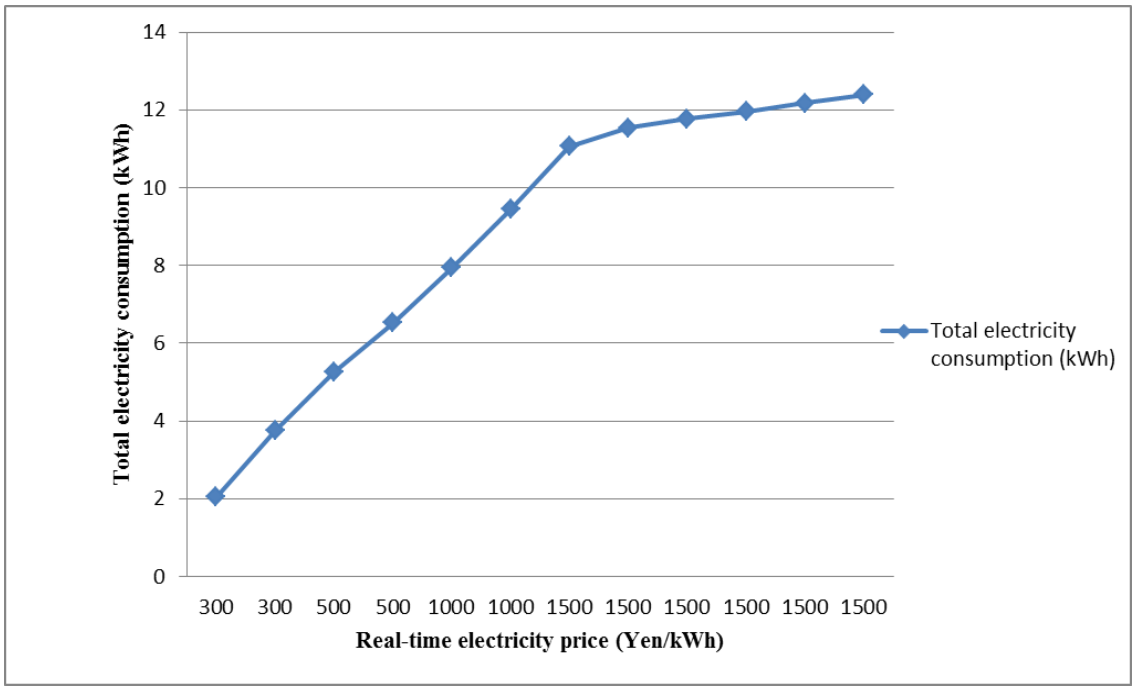


Figure 6-3. Total electricity consumption at different prices

Table 6-4. Simulated tiered pricing for electricity

Electricity Consumption (kWh)	1.1–4.0	4.1–7.0	7.1–10.0	10.1~
Electricity Price (Yen/kWh)	300	500	1000	1500

When the real-time electricity price exceeded 1000 Yen/kWh, rule 2 turns off the air conditioner in the room where the temperature is the highest; rule 3 turns off the humidifier in room 3L-5 if the humidity of this room is higher than 30%. Rule 4 turns off all the lights in room 3L-6 when the real-time electricity price exceeds 1500 Yen/kWh. We can infer from the above settings that when the real-time price exceeds

1000 Yen/kWh, the growth rate of total consumption should decrease substantially. As a result of the delay in the control system, we observed this phenomenon only at the end of the second cycle when the real-time price was 1000 Yen/kWh.

6.2.4 Discomfort Index (DI) Control Policy

Discomfort Index (DI) [123] was proposed by Thom (1959). They designed a simple linear equation to calculate DI based on temperature and relative humidity:

$$DI = T - 0.55(1 - 0.01\,H)(T - 14.5) \qquad (6\text{-}1)$$

where T is the temperature in ℃ and H is the relative humidity in %. When DI exceeds 21.0, people feel uncomfortable. Based on Formula 6-1, we designed a Discomfort Index Control Policy to control the comfort level in room 3L-5. We observed the value ranges of the environmental parameters and designed the rules for this policy based on the observed temperature and humidity values in room 3L-5:

Table 6-5. Parameter definitions of discomfort index control policy

	Temperature ℃	**Humidity %**	**DI**
Observed values	21–22	35–40	19.1–22.2
Values for rules	23–24	42–45	Less than 21.0

- **Rule 1**: WHEN the DI of room 3L-5 is greater than 21.0, AND temperature is greater than 24 ℃, THEN turn off the air conditioners in this room.
- **Rule 2**: WHEN the temperature in room 3L-5 is less than 23 ℃, THEN turn on the air conditioners in this room.
- **Rule 3**: WHEN the DI of room 3L-5 is greater than 21.0, AND humidity is greater than 45%, THEN turn off the humidifiers in this room.
- **Rule 4**: WHEN the humidity in room 3L-5 is less than 42%; THEN turn on the humidifiers in this room.

We adopted this policy in room 3L-5 for several hours. The patterns of discomfort index, humidity, and temperature are shown in Figure 6-4:

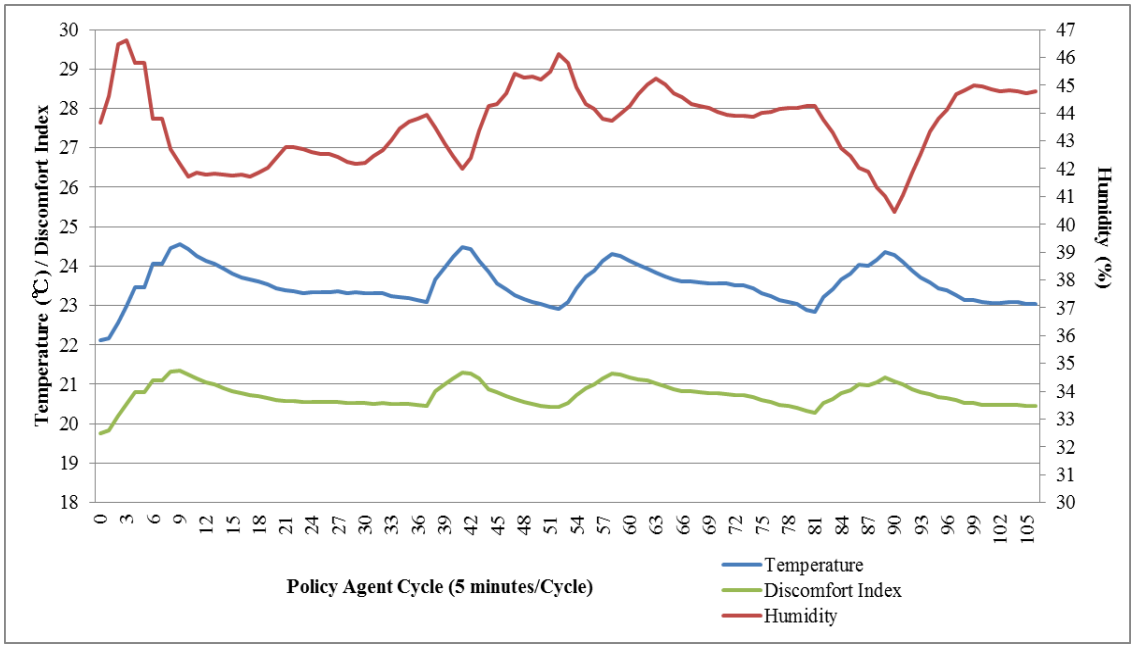


Figure 6-4. Temperature, humidity, and DI patterns during the experiment

Owing to the delay in the actuator system, the values for control targets (humidity, temperature, DI) slightly exceeded the boundaries defined in the rules. The values for the controlled parameters approximately fall in the ranges defined by the user. In general, Figure 6-4 shows that users can write detailed rules using DCRDL to control the environmental conditions and can obtain the expected results in smart buildings.

6.2.5 Conclusion

We evaluated DCRDL, which addresses the problem of context-based device control for energy efficiency. DCRDL enables the writing of fine-grained control rules for devices in smart buildings to balance the energy usage and human comfort. Using DCRDL, users can define various policies with detailed rules for networked electrical devices for different energy saving scenarios. Three policies were written and experiments were conducted to demonstrate the functionality and expressivity of DCRDL in a real smart environment: the Daiwa Ubiquitous Computing Research Building.

Some issues such as rule conflicts and duplicate rule detection were not discussed here. However, implementing a rule-optimizer using the existing algorithms can solve these problems. Access control to collected data and devices is another critical problem associated with running the PAS. However, those issues are beyond the scope of this research; we may consider those problems in future work.

6.3 Evaluation of EDPL

6.3.1 Objective

We will discuss the readability and writability [124], [125] of EPDL in this section. The objective of discussing readability and writability of EPDL is to show the user-friendliness of it. The rule format for EPDL and DCRDL are also compared. In addition, we compare average code lines for rules in both EPDL and DCRDL. Finally, we show how a user can refer to available resources in the smart building easily with the EPDL editor plugin in the Eclipse IDE [117].

6.3.2 Readability

Readability simply refers to “the ease with which programs can be read and understood” [125]. For the descriptive languages we proposed in this research, readability specifically refers to the support for expressing control sequences in natural ways. In this section, we compare rules, which are the key elements in both EPDL and DCRDL, to show the improvement of readability and expressivity of EPDL.

```
<Rule RuleId="uri:skl:names:dcrdl:1.0:ruleid:1" >
    <ConditionSet>
        <Condition>
            <Subject>
                <Function FunctionId="uri:skl:names:dcrdl:1.0:function:get-space-electricity-consumption">
                    <Parameter Value="X0001C0000000000000020000000D648"
                               DataType="uri:skl:names:dcrdl:1.0:DataType:ucode" />
                    <Parameter Value="00:15:00"
                               DataType="http://www.w3.org/2001/XMLSchema#string" />
                </Function>
            </Subject>
            <Function FunctionId="uri:skl:names:dcrdl:1.0:function:float-greater-than">
                <Parameter Value="0.35"
                           DataType="http://www.w3.org/2001/XMLSchema#float"/>
            </Function>
        </Condition>
    </ConditionSet>
    <ActionSet>
        <Action>
            <Target TargetId="X0001C0000000000000020000000D640"
                    DataType="uri:skl:names:dcrdl:1.0:DataType:ucode" />
            <Function FunctionId="uri:skl:names:dcrdl:1.0:function:control-ac-setpoint-decrease">
                <Parameter Value="1"
                           DataType="http://www.w3.org/2001/XMLSchema#integer" />
            </Function>
        </Action>
        <Action>
            <Target TargetId="X0001C0000000000000020000000D641"
                    DataType="uri:skl:names:dcrdl:1.0:DataType:ucode" />
            <Function FunctionId="uri:skl:names:dcrdl:1.0:function:control-ac-setpoint-decrease">
                <Parameter Value="1"
                           DataType="http://www.w3.org/2001/XMLSchema#integer" />
            </Function>
        </Action>
    </ActionSet>
</Rule>
```

a. The rule in DCRDL

```
rule "uri:skl:names:dcrdl:1.0:ruleid:1" when {
        building.floors["3F"].spaces["3L-5"].sba_get_PeriodElectricityConsumption("00:15:00") > 0.35
    }then{
        building.floors["3F"].spaces["3L-5"].appliances["AirConditioner1"].sba_ctr_ac_off
        building.floors["3F"].spaces["3L-5"].appliances["AirConditioner2"].sba_ctr_ac_off
    }
```

b. The rule in EPDL

Figure 6-5. The same rule in DCRDL and EPDL

Figure 6-5 describes a rule that states that when the electricity consumption of room “3L-5” exceeds 0.35 kWh in the last 15 minutes, then the two air conditioners in the room should be turned off. The code shown in Figure 6-5a is written in DCRDL, whereas that in Figure 6-5b is written in EPDL. It is clear that the more concise EPDL rule description is easier to read. Furthermore, the IF and Foreach statements are introduced in EPDL, which make this language more flexible. For example, when we want to apply a rule to all appliances that are air conditioners in a room, we can compose a fragment of code using the IF and Foreach statements to achieve this. In

addition, the number of code lines is significantly reduced for a rule with the same goal. Figure 6-5 shows that the number of code lines has decreased from 36 in DCRDL to six in EPDL for the same rule. We rewrote the two policies used in the evaluation of DCRDL in the winter of 2014–2015 with EPDL. These two policies are described in English in this section for better understanding.

- Peak Consumption Control Policy (see section 6.2.2);

```
policy Peak_Consumption_Control {
    policy.id = "uri:skl:names:epdl:1.0:policy:PCC"
    policy.period = "00:15:00"
    foreach app1 within building.sba_get_space_of_highest_Temperature{
        rule "air_conditioner_de" when {
            app1.type=="air_conditioner" && building.sba_get_PeriodElectricityConsumption("00:15:00") > 0.6
        }then{  app1.sba_ctr_ac_decrease_setpoint(2.0)  }
    }

    foreach app2 within building.floors["3F"].spaces["3L-6"].appliances{
        rule "light_off" when {
            app2.type=="light" && building.floors["3F"].spaces["3L-6"].sba_get_PeriodElectricityConsumption("00:15:00") > 0.2
        }then{  app2.sba_ctr_light_off  }

        rule "air_conditioner_setpoint_3l6" when {
            app2.type=="air_conditioner" && app2.sba_get_ac_setpoint < 22.0
        }then{  app2.sba_ctr_ac_off  }
    }

    foreach app3 within building.floors["3F"].spaces["3L-5"].appliances{
        if(app3.type=="air_conditioner"){
            rule "air_conditioner_setpoint_3l5" when {
                app3.sba_get_ac_setpoint < 22.0
            }then{  app3.sba_ctr_ac_off  }
        }
    }
}
```

Figure 6-6. The EDPL code for the peak consumption control policy

- Discomfort Index (DI) Control Policy (see section 6.2.4).

```
policy discomfort_index_control{
    policy.id = "uri:skl:names:dcrdl:1.0:policy:DIC"
    policy.period="00:15:00"
    rule "r_ac_1" when{
        building.floors["3F"].spaces["3L-5"].sba_get_discomfortIndex > 21.0
        && building.floors["3F"].spaces["3L-5"].sba_get_temperature > 24.0
    } then {
        building.floors["3F"].spaces["3L-5"].appliances["AirConditioner1"].sba_ctr_ac_off
        building.floors["3F"].spaces["3L-5"].appliances["AirConditioner2"].sba_ctr_ac_off
    }
    rule "r_ac_2" when{
        building.floors["3F"].spaces["3L-5"].sba_get_temperature < 23.0
    }then{
        building.floors["3F"].spaces["3L-5"].appliances["AirConditioner1"].sba_ctr_ac_on
        building.floors["3F"].spaces["3L-5"].appliances["AirConditioner2"].sba_ctr_ac_on
    }
    rule "r_humidifier_1" when{
        building.floors["3F"].spaces["3L-5"].sba_get_discomfortIndex > 21.0
        && building.floors["3F"].spaces["3L-5"].sba_get_humidity > 0.45
    }then{
        building.floors["3F"].spaces["3L-5"].appliances["humidifier"].sba_ctr_humidifier_off
    }
    rule "r_humidifier_2" when{
        building.floors["3F"].spaces["3L-5"].sba_get_humidity < 0.42
    }then{
        building.floors["3F"].spaces["3L-5"].appliances["humidifier"].sba_ctr_humidifier_on
    }
}
```

Figure 6-7. The EPDL code for the discomfort index control policy

Table 6-6. The number of code lines of EPDL and DCRDL Rules

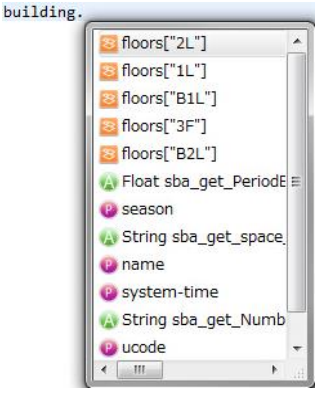
	Lines of each policy		**Average lines in a rule**
	PCC[a] **(6 rules)**	DIC[b] **(4 rules)**	
DCRDL	155	214	37
EPDL	37	28	7

a: peak consumption control; b: discomfort index control;

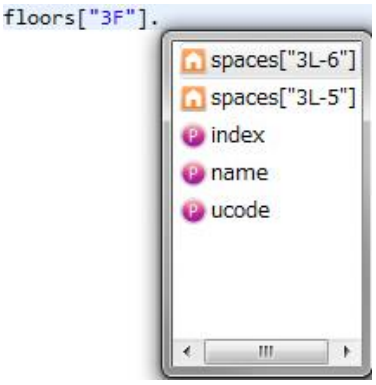
6.3.3 Writability

The writability simply refers to "the ease with a language can be used to create programs" [125]. For this research, we proposed two descriptive languages which are used for describing control sequences for smart building systems. Moreover, the writability of EPDL also includes the ease of using available services. Referring to a service is very difficult for most users because of the heterogeneous networks and complex structure of smart buildings. Figure 6-5a shows a DCRDL rule. In order to write a DCRDL rule, the user has to study XML and have knowledge of the target smart building. For example, the number of rooms on each floor, the appliances in a room, and the operations that can be performed on each appliance. Figure 6-5b shows the improved rule format in EPDL, fewer lines than the DCRDL rule is required, which means that it is easier to write.

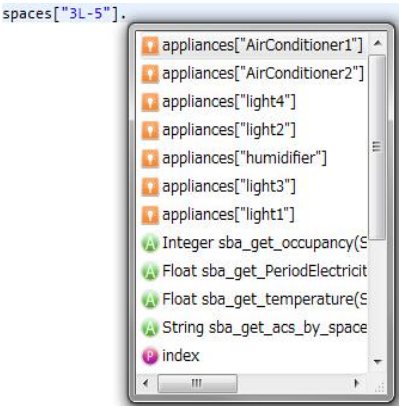
Furthermore, we designed a smart building resource descriptor to assist the user to refer to the available resources when writing a policy. Hence, the user does not need to study the details of the target building in advance. The descriptor is parsed and the information is used to compute the code completion proposals. For instance, when the user types "building" in the language editor, a pop-up menu with all the available objects and services under the "building" level appears. Figure 6-8 shows code completion proposed examples.



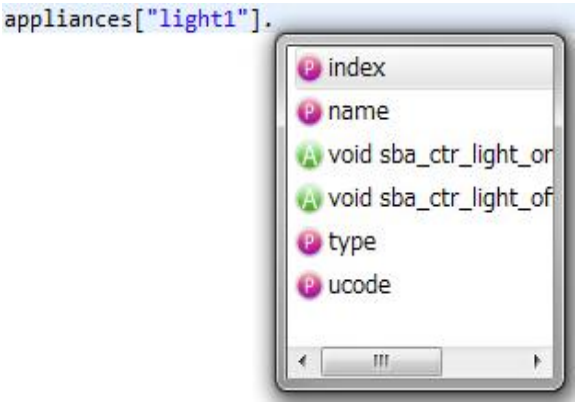
a. Code completion proposals of the building



b. Code completion proposal of a floor



c. Code completion proposals of a space



d. Code completion proposals of an appliance

Figure 6-8. Code completion proposal examples

6.4 Summary

In this section, we discussed the readability and writability of EPDL and compared it with DCRDL. DCRDL is a kind of programming enabler for the physical objects in IoT-enabled smart buildings. However, it is a low-level rule description language. Detailed information about the target building and programming skills are required when using DCRDL. From the discussion, we can conclude that EPDL is a user-friendly energy policy description language for IoT-enabled smart buildings. The proposal of smart building resource descriptor enables non-expert users to write an energy control policy for smart buildings easily and quickly.

7 FUTURE WORK

7.1 Conflict Resolution

We have discussed the peak demand issue of smart buildings in previous chapters. A policy agent, which consisted of predefined constraints (i.e., rules), and a constraint solver were designed to address this problem. Two research problems could be subsequently studied: (1) the design of interfaces to ease the writing of control rules/policies for the smart building system; and (2) the design of efficient rule reasoning algorithms for the policy agent system used in this framework. We chose to focus on the first problem for this thesis.

However, the need for algorithms for conflicting rules resolution is a critical issue for this framework. For example, there are two rules in a policy agent shown in Table 7-1:

Table 7-1. Example of conflict rules

Rule 1	Set-point of Air conditioners should be in [22-26]
Rule 2	If the discomfort index of this room is less than 21.0 and the temperature of this room is greater than 24, decrease the set-points of air conditioners in this room by 2.
Context	Discomfort index: 20.0; Air conditioners' set-point: 22; Season: summer

Rule 1 and 2 will conflict with each other in the context shown in Table 7-1. Rule 1 tries to keep the set-point of conditioners between 22 and 26, while Rule 2 tries to set

the set-point to 20 in this context. When the number of rules grows, conflicts may occur frequently which will cause system crashes.

In fact, this is a classical problem of rule-based expert systems. Many algorithms (e.g., Rete Algorithm [76]) have already been proposed to address this issue. We will study conflicting rules resolution algorithms and implement a rule optimizer for this framework in future work.

7.2 Resource Management Framework

With the adoption of IoT protocols (e.g., 6LoWPAN [35] and Constrained Application Protocol (CoAP) [36]), the Internet of Things (IoT) has evolved to encompass the Web of Things (WoT) as its application layer. Constraint devices have been integrated into IP-based IoT via those protocols, and they are seamlessly integrated into the legacy Web. Consequently, existing web technologies can be directly applied to the newly evolved WoT. Using web-based APIs, data produced by a device can be easily accessed. Meanwhile, manufacturers have begun to produce web controllable devices (such as wiring boards, lights, and air conditioners). In particular, in smart buildings equipped with such devices, data access and device control have become easy and ubiquitous as long as the user has a device connected to the web.

The data access and device control interface can be unified via RESTful APIs in the age of WoT. A critical issue of bringing ordinary people to participate in programming with physical objects is the means by which the upper-layer user translates the described application logic into executable code that can be understood by machines. That is, ordinary people can only use simple tools such as declarative programming and drag and drop programming [126]; however, applications developed using descriptive languages cannot be easily understood by machines.

Therefore, resource discovery and management have become key issues that should be addressed in the flexible programming framework in IoT-enabled buildings. There are already many studies associated with resource discovery. Most of them provide excessive ontology modeling of building structures that are not necessary in most cases. However, none of them address the resource representation problem, especially service

modeling of smart buildings. Providing an efficient service representation can help language interpreters to translate referred service into executable code that can be executed by machines. Further, standardized resource representation operation interfaces must be defined to provide a unified access path to various applications.

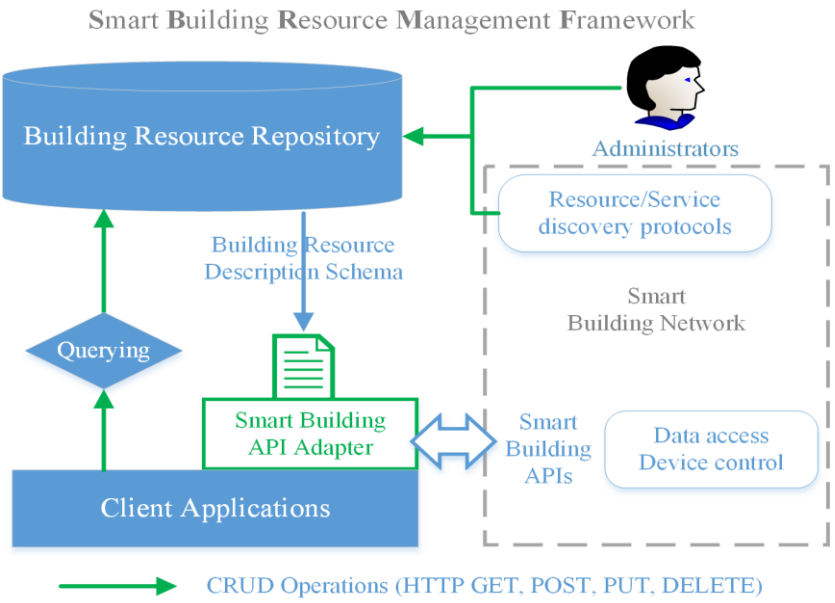


Figure 7-1. Architecture of smart building resource management framework

We are currently designing a smart building resources management framework with the following features:

- Building Resource Description Schema (BRDS)
  - ✧ Description of building structure and semantic relations of internal objects.
  - ✧ Description of RESTful Smart Building APIs using RAML [127].
- Building Resource Operation Interfaces (Create, Retrieve, Update, Delete operations)
  - ✧ Define standard RESTful APIs for operating building resources. Resource discovery protocols or building administrators can store/delete the BRDS data in the building resource repository using standardized Create, Update, and Delete interfaces; users can obtain resources via the Retrieve interface. For example, when users want to refer to the service to turn on air conditioner 1 in

room 3L-5 on the third floor, they can simply retrieve all service and object information of room 3L-5 using the following RESTful interface:

GET http://baseurl/DUCRB/3F/3L-5/

GET	HTTP GET;
baseurl	The IP of the resource repository
DUCRB	ID of the building
3F	floor ID
3L-5	Space ID

✧ Smart Building API adapter: This adapter models RESTful APIs. It contains an interpreter that translates user referred services into executable code with the help of service description information stored in the resource repository.

7.3 Visual Programming

Many researchers have proposed ideas for giving end-users efficient tools to program their surrounding environment. They often follow the "IF-THEN" paradigm, which is also the basic principle of control languages in smart buildings. The design of DCRDL and EPDL also follow this paradigm. The objective of DCRDL is to separate upper-layer user control logic description from lower-layer service and structure details of smart buildings so that upper-layer users can focus on describing the control sequence for their application without being concerned about how to describe sequences in building systems. We plan to design a universal visual programming application for smart buildings. It should be able to run on any capable device connected to the network of a target smart building. It should require no porting work when used in a different building. In other words, it should have one universal programming interface for all buildings with the promise embodied by the unified resource management and representation protocols discussed in section 7.2.

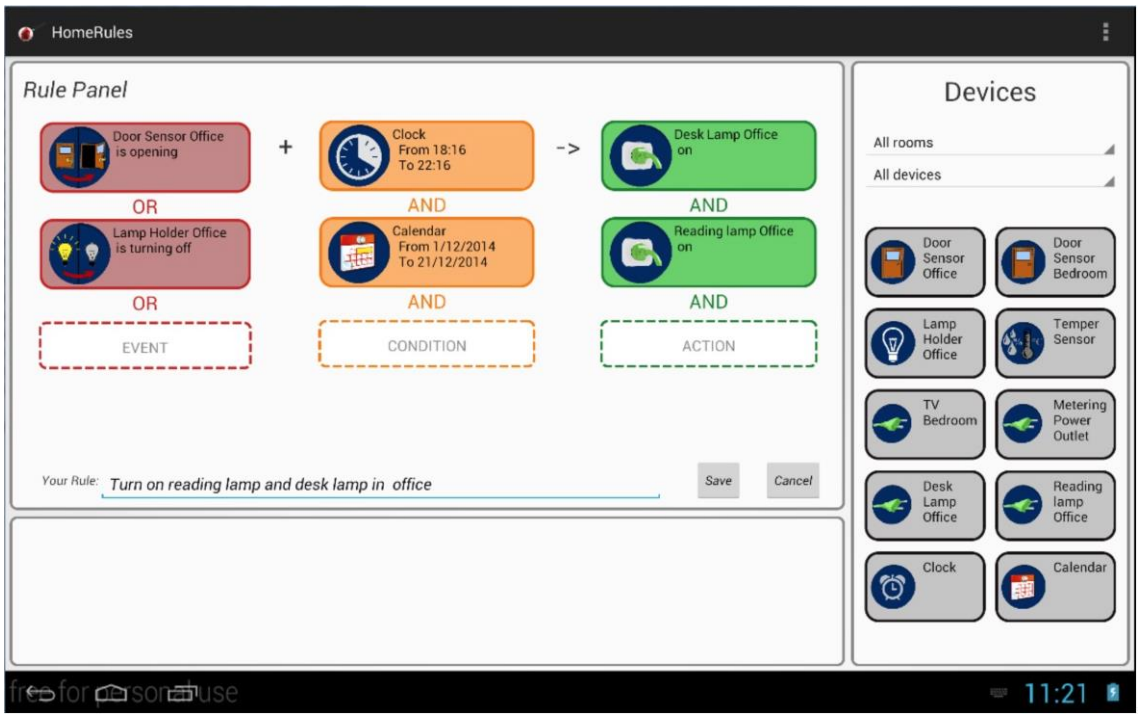


Figure 7-2. Demo of the HomeRules [126]

Russis et al. developed a prototype of "a tangible end-user programming interface" [126] called HomeRules for smart homes. As shown in Figure 7-2, HomeRules also follows the "IF-THEN" paradigm. It enables the end-user to write descriptive rules for their home appliances via drag and drop resource icons. However, there is no explanation of how their application would work with smart building systems. There should be an interpreter or translator that translates the descriptive rules into pieces of executable code. In addition, there is no demonstration of how they can get available resources data. As discussed in section 7.2, standardized resource retrieval and management protocols are important in order to adopt these applications to different buildings without extra porting work.

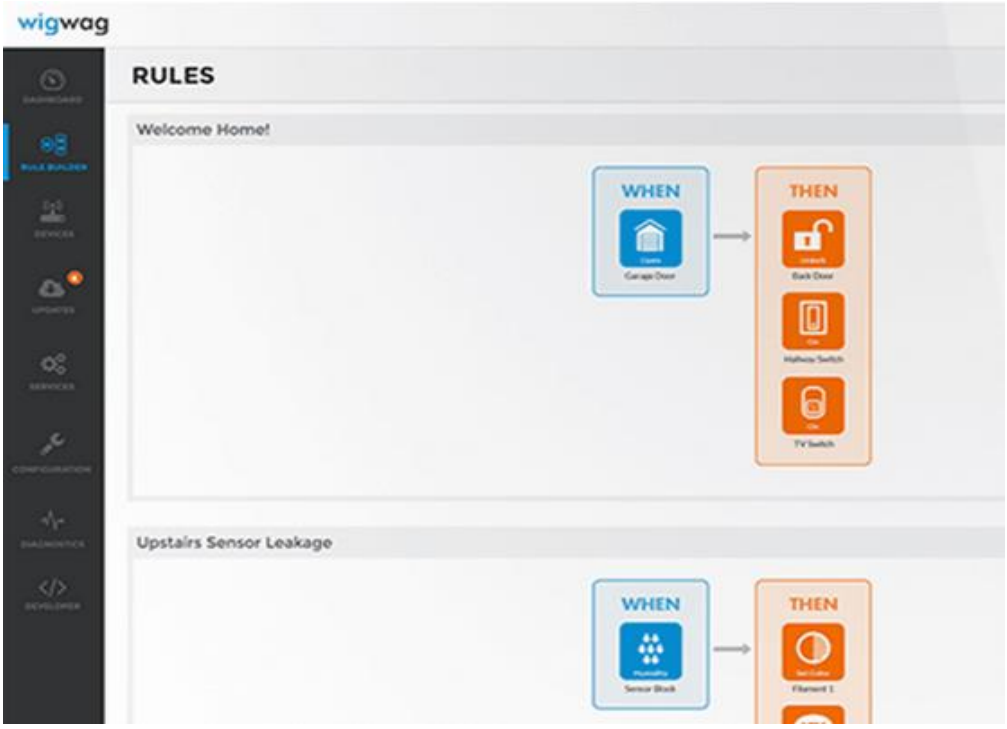


Figure 7-3. WigWag rules editor for smart home [128]

In industry, WigWag [128] also designed a demo application that they claim can enable end-users to write rules for home appliances by dragging and dropping resource icons. However, some of the issues discussed above are apparent in this application. The universal visual programming application we plan to develop with the standardized resource representation protocol and operation interfaces in the future will overcome these shortcomings.

7.4 Access Control to Data and Services

Access control to data and services is an extensive research direction in IoT and is actually out of the scope of this research. Nevertheless, we give a short discussion here because it is a significant and critical issue for this programming framework. The Access Control to Data and Services issue we define here should clearly answer the question: To ***whom*** and ***when*** can access be granted or ***what*** can be used from ***where***? We call it the 4W issue.

The rapid advancement of smart devices and improved connectivity among them introduce a new programming paradigm for IoT-enabled smart buildings. That data access and device control achieved through web-based open APIs eases the programming procedure for smart buildings, but also introduces myriad new risks. Unauthorized access and control have led to critical privacy and security concerns. We are considering adding access control mechanisms (e.g., access control ontology for buildings) into our BRDS to address this problem in the future. The ontology may be a combination of sub-ontologies that describe various aspects of security and privacy factors.

8 CONCLUSION

In this thesis, we introduced a proposed programming framework for building automation in IoT-enabled smart buildings. The connectivity of smart devices has significantly improved as a result of protocols such as 6LoWPAN. Resource constrained devices can be directly connected to the Internet in like manner as a computer. Further, the Constrained Application Protocol (CoAP) makes resources available under a URL using the wildly successful REST model. It significantly accelerated the seamless integration of constrained devices to existing Internet as the application layer. Data access and device control interfaces are becoming unified through open RESTful APIs deployed in IoT smart buildings. This has substantially simplified application development in smart buildings. In addition, collaboration of devices among different sub-systems has become possible as a result of unified interfaces. Users can develop and execute building automation applications on devices if they are able to connect to the building's network. We call such an IoT-enabled building a smart building.

We also regard the overall IoT-enabled smart building as a distributed hardware computer and the users as living in the computer. Thus, it is very important to enable those users to program this "computer." However, the hardware components (i.e., devices) are distributed across building scale, and those users are usually not good at programming. Therefore, an easier programming environment is needed for them.

We outlined the background to this research in chapter 1. A brief introduction of building automation history was presented in section 1.1.1. Then, the general structure of Building Automation System, which is also referred to as BMS and BEMS was presented in section 1.1.2. Finally, we defined the concept of IoT-enable smart buildings. The energy efficiency problem was introduced in section 1.2. We also explained the objective and challenges of this research in general at the end of chapter 1.

In chapter 2, related work was reviewed in several aspects. We first introduced the programming paradigm changes associated with smart buildings. Then, the web technologies used in smart buildings were presented. Finally, we analyzed existing work that related to resource representation and programming languages. A general description of our proposal was also presented in chapter 3.

We described the design of each part of our framework in chapter 4. The Policy Agent System (PAS) is the basic running environment of the whole framework. Policy Agents are translated from DCRDL rules. They can run as the threads of procedural languages such as Java, C, JavaScript, and PHP. We also studied the Peak Demand Control issue of smart buildings and modeled it as a Constraint Satisfaction Problem. DCRDL was proposed to separate the upper-layer control logic description and lower-layer service and structural details of buildings. Finally, we presented the design of EPDL to support the writing of energy management policies.

In chapter 5, we presented the implementation details of our proposed framework. It was developed using Java. We developed a local Java library as a wrapper for the RESTful Smart Building APIs. Policy Agents are executed in the form of Java Threads. The interpreter of DCRDL was also developed in Java. Finally, we provided various efficient tools to make the writing energy policies easier for non-expert users in EPDL.

We outlined the experiments conducted in this research in chapter 6. All the experiments were carried out in DUCRB—a state-of-the-art IoT-enabled smart building constructed in 2014. The Peak Demand Control framework was implemented and run as a policy agent. Three policies written in DCRDL were evaluated in the policy agent system. We analyzed the experimental data and got the expected results. Finally, we discussed the readability and writability of EPDL. The results showed that it is an

efficient tool for non-expert users to read and write energy management policies.

Finally, resource management and representation protocols, visual programming, and access control to data and services were introduced in chapter 7 as future work associated with this research.

To summarize, we designed a programming framework work for IoT-enabled smart buildings. Two descriptive languages were proposed as the input interfaces for smart building users to define rules or policies to manage the building system. A smart building resource description schema was designed for system developers to publish available resources of smart buildings in a format that both machine and human can understand. It is also can be used by resource discovery protocols to build resource repository automatically. A set of tools was implemented to assist users in writing policies and translated them into executable code. Some important issues (e.g., Rule conflict resolution, building resource management protocols) weren't addressed in this thesis. We will study them in future publications.

From the perspective of a building as a distributed hardware computer, we can also regard the IoT as a huge distributed hardware computer. The embedded operating systems inside the constituent devices constitute a huge distributed operating system for IoT. People are living inside this "computer"; therefore, we have to enable them to program their surrounding environment. Thus, as with classic computers, resource abstraction and simpler programming languages have to be provided to help those ordinary users to interact with surrounding devices. We intend to extend this framework the entire IoT in the future.

9 REFERENCES

[1] M.R. Brambley, D. Hansen, P. Haves, D.R. Holmberg, S.C. McDonald, K.W. Roth, and P. Torcellini, "Advanced Sensors and Controls for Building Applications: Market Assessment and Potential R&D Pathways," *Pacific Northwest National Lab.*, Richland, Washington, Rep. PNNL-15149, Apr. 2005.

[2] T. Samad and Albert T.P. So, "Building control and automation systems," In *Perspectives in Control Engineering Technologies, Applications, and New Directions, 1st ed*. Wiley-IEEE Press, 2001, ch. 16, pp. 393-416.

[3] "Direct Digital Control." Internet: https://en.wikipedia.org/wiki/Direct_digital_control, Jan. 7, 2016 [Feb. 4, 2016].

[4] A. H. Kazmi, M. J. O'Grady, D. T. Delaney, A. G. Ruzzelli, and G. M. P. O'Hare, "A review of wireless-sensor-network-enabled building energy management systems," In *ACM Trans. Sensor Netw.*, vol. 10, no. 4, Article 66, 43 pages, Jun. 2014.

[5] M. Pallack, D. Cook, and J. Sullivan, "Commitment and energy conservation," In *Appl. Social Psychol*. vol. 1, no. 1, pp. 235-253, 1980.

[6] "HVAC." Internet: https://en.wikipedia.org/wiki/HVAC, Jan. 30, 2016 [Feb. 4, 2016].

[7] Karl Johan Åström, "PID Control," In *Control System Design*, 2002, ch. 6, pp. 216-251.

[8] Arthur Richards and Jonathan How, "Robust Stable Model Predictive Control with Constraint Tightening," In *Proc. 2006 American Control Conference*, Minneapolis, USA, Jun. 14-16, 2006.

[9] Peng Wang and Daniel P. Kwok, "Analysis and synthesis of an intelligent control system based on fuzzy logic and the PID principle," In *Intelligent Systems Engineering*, vol. 1, no. 2, pp. 157-171, Winter 1992.

[10] Luigi Martirano, Matteo Manganelli, Luigi Parise, and Danilo A. Sbordone, "Design of a fuzzy-based control system for energy saving and users comfort," In Proc. *Environment and Electrical Engineering*, Krakow, 2014, pp. 142-147.

[11] A. Cziker, M. Chindris, and Anca Miron, "Fuzzy controller for a shaded daylighting system," In Proc. *11th Int. Conf. on Optimization of Elect. and Electron. Equipment*, Brasov, May 22-24 2008, pp. 203-208.

[12] J. Haase, G. Zucker, F. AlJuheshi, M. k. Allah, and M. Alahmad, "A Survey of Adaptive Systems Supporting Green Energy in the Built Environment," In Proc. *IECON 2015, Yokohama*, Nov. 9-12, 2015.

[13] J. M. Sousa, R. Babuska, P. Bruijn, and H. B. Verbruggen. "Comparison of conventional and fuzzy predictive control," In Proc. *5th IEEE International Conference on Fuzzy Systems*, Vol. 3, pp. 1782-1787, 1996.

[14] A. M. Vainio, M. Valtonen, and J. Vanhala, "Proactive fuzzy control and adaptation methods for smart homes," In *IEEE Intell. Syst.*, vol. 23, pp. 42-49, 2008.

[15] G. W. Hart, "Nonintrusive appliance load monitoring," In *Proc. IEEE*, vol. 80, no. 12, pp. 1870-1891, 1992.

[16] Y. Kim, T. Schmid, Z. M. Charbiwala, and M. B. Srivastava, "Viridiscope: Design and implementation of a fine grained power monitoring system for homes," In Proc. *11th International Conference on Ubiquitous Computing (Ubicomp'09), ACM Press*, New York, 2009, pp. 245-254.

[17] X. H. Peng, M. Bessho, N. Koshizuka, and K. Sakamura, "DCRDL: An Energy Management Rule Definition Language for Context-based Device Control in Smart Buildings," In Proc. *41st Annu. Conf. of the IEEE Ind. Electronics. Soc. (IECON 2015)*, Nov. 2015, pp. 279-285.

[18] "Particulates." Internet: https://en.wikipedia.org/wiki/Particulates, Mar. 12, 2016 [Mar. 19, 2016].

[19] *BACnet standard*, ANSI/ASHRAE Standard 135-2008.

[20] Enron Corporation, "*LonTalk Protocol Specification*," Enerlon, 1994.

[21] W. Jung, S. I. Kim, and H. S. Kim, "Ontology Modeling for REST Open APIs and Web Service Mash-up Method," In Proc. *2013 Int. Conf. Inform. Netw.*, Jan. 2013, pp. 523-528.

[22] R. P. V. Chander, S. Elias, S. Shivashankar, and Manoj P, "A REST based design for Web of Things in smart environments," In Proc. *2nd IEEE International Conference on Parallel, Distributed and Grid Computing*, 2012, pp. 337-342.

[23] I. Claros, R. Cobos, E. Guerra, J. de Lara, A. Pescador, and J. Sánchez-Cuadrado, "Integrating Open Services for Building Educational Environments," In Proc. *2013 IEEE Global Engineering Education Conference (EDUCON)*, Berlin, March 13-15, 2013, pp. 1147-1156.

[24] W. Kastner, M. Kofler, M. Jung, G. Gridling, and J. Weidinger, "Building Automation Systems Integration into the Internet of Things The IoT6 approach, its realization and validation," In Proc. *2014 Emerging Technology and Factory Automation (ETFA)*, Barcelona, Sept. 16-19, 2014.

[25] D. Yazar and A. Dunkels, "Efficient application integration in ip-based sensor networks," In Proc. of *1st ACM Workshop on Embedded Sensing Syst. for Energy-Efficiency in Buildings*, California, 2009, pp. 43-48.

[26] M. Weiss and D. Guinard, "Increasing energy awareness through web-enabled power outlets," In Proc. *9th MUM*, Limassol, Dec. 2010.

[27] A. D. Paola, S. Gaglio, G. L. Re, and M. Ortolani, "Sensor9k: A testbed for designing and experimenting with WSN-based ambient intelligence applications," In *Pervasive and Mobile Computing*, vol. 8, no. 3, pp. 448-466, Jun. 2012.

[28] G. Ghidini and S. K. Das, "Improving home energy efficiency with E2Home: A Web-based application for integrated electricity consumption and contextual information visualization," In Proc. *IEEE SmartGridComm2012*, Nov. 2012, pp. 471-475.

[29] A. Kamilaris, A. Pitsillides, and M. Yiallouros, "Building energy-aware smart homes using web technologies," In *J. Ambient Intell. Smart Environ.*, vol. 5, no. 3, pp. 161-186, Mar. 2013.

[30] K. Sakamura, "Programable Architecture: A smart control system in Daiwa Ubiquitous Computing Research Building," In *TRONWARE*, vol. 148, Aug. 2014.

[31] K. Sakamura, "The objectives of the TRON project," *TRON Project 1987 Open-Architecture Computer Systems: Proceedings of the Third TRON Project Symposium*, Tokyo: Springer-Verlag, 1987, pp. 3-16.

[32] S. Dawson-Haggerty, A. Krioukov, J. Taneja, S. Karandikar, G. Fierro, N. Kitaev, and D. Culler, "BOSS: Building Operating System Services," In Proc. *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI '13)*, Apr. 2-5, 2013, pp. 443-458.

[33] A. D. Paola, M. Ortolani, G. L. Re, G. Anastasi, and S. K. Das, "Intelligent management systems for energy efficiency in buildings: A survey," In *ACM Comput. Surv.*, vol. 47, no. 1, Article 13, 38 pages, May 2014.

[34] Mirko Presser (Jan. 12, 2016), "The Rise of IoT – Why Today?" Newsletter of IEEE Internet of Things, Available: http://iot.ieee.org/newsletter/january-2016/the-rise-of-iot-why-today.html

[35] Transmission of IPv6 Packets over IEEE 802.15.4 Networks (6LoWPAN), IETF rfc4944.

[36] The Constrained Application Protocol (CoAP), IETF rfc7252.

[37] Roy T. Fielding and Richard N. Taylor, "Principled Design of the Modern Web Architecture," In Proc. *ICSE 2000*, Limerick, Ireland, 2000, pp. 407-416.

[38] Roy T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Ph.D. dissertation, Univ. OF CALIFORNIA, IRVINE, California, U.S, 2000.

[39] "Visual programming language." Internet: https://en.wikipedia.org/wiki/Visual_programming_language, Mar. 18, 2016 [Mar. 19, 2016].

[40] Ronald Greaves, Scott McCauley, Max McLeod, and Tom Rule, "Total Building Solutions for Hospitals – The next generation of intelligence," *Siemens Industry*, Inc, 2014.

[41] Felix Iglesias Vazquez and Wolfgang Kastner, "Clustering methods for occupancy prediction in smart home control," In Proc. *2011 IEEE International Symposium on Industrial Electronics (ISIE)*, 2011, pp. 1321-1328.

[42] M. C. Mozer, "The neural network house: An environment hat adapts to its inhabitants," In Proc. *AAAI Spring Symp. Intelligent Environments*, 1998, pp. 110-114.

[43] R. H. Dodier, G. P. Henze, D. K. Tiller, and X. Guo, "Building occupancy detection through sensor belief networks," In *Energy and Buildings*, vol. 38, no. 9, pp. 1033-1043, 2006.

[44] J. Lu, T. Sookoor, V. Srinivasan, G. Gao, B. Holben, J. Stankovic, E. Field, and K. Whitehouse, "The smart thermostat: Using occupancy sensors to save energy in homes," In Proc. *8th ACM Conf. on Embedded Networked Sensor Syst. (SenSys'10)*, 2010, pp. 211-224.

[45] X. H. Peng, M. Bessho, N. Koshizuka, and K. Sakamura, "EPDL: Supporting Context-based Energy Control Policy Design in IoT-enabled Smart Buildings," In Proc. *2015 IEEE International Conference on Data Science and Data Intensive Systems (DSDIS)*, Sydney, Dec. 11-13, 2015, pp. 297-303.

[46] Naohiko Kohtake, Kenta Matsumiya, Kazunori Takashio, and Hideyuki Tokuda, "Smart Device Collaboration for Ubiquitous Computing Environment," In Proc. *the Workshop on Multi-Device Interface for Ubiquitous Peripheral Interaction at the 5th International Conference on Ubiquitous Computing*, 2003.

[47] Huafen Hu, Yonghong Huang, Milan Milenkovic, Chad Miller, and Ulf Hanebutte, "Personalized sensing towards building energy efficiency and thermal comfort," In Proc. *2014 International Joint Conference on Neural Networks (IJCNN)*, Beijing, Jul. 2014, pp. 1963-1969.

[48] Orestis Evangelatos, Kasun Samarasinghe, and Jose Rolim, "Syndesi: A Framework for Creating Personalized Smart Environments Using Wireless Sensor Networks," In Proc. *2013 IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS)*, Cambridge, May 2013, pp. 325-330.

[49] Andrew Krioukov, Gabe Fierro, Nikita Kitaev, and David Culler, "Building Application Stack (BAS)," In Proc. *Buildsys'12*, Toronto, Canada, Nov. 6, 2012. pp. 72-79.

[50] Blase Ury, Elyse McManus, Melwyn Pak Yong Ho, and Michael L. Littman, "Practical Trigger-Action Programming in the Smart Home," In Proc. *CHI 2014*, Toronto, ON, Canada.

[51] J. Sousa, R. Babuska, P. Bruijn, and H. Verbruggen, "Comparison of conventional and fuzzy predictive control," In Proc. *5th IEEE Int. Conf. on Fuzzy Syst.*, vol. 3, Sept. 1996, pp. 1782-1787.

[52] F. Praus, W. Granzer, and W. Kastner, "Enhanced Control Application Development in Building Automation," In Proc. *7th IEEE Int. Conf. on Industrial Informatics*, Jun. 2009, pp. 390-395.

[53] KNX standard, ISO/IEC 14543-3, 2006.

[54] DAIKIN [Online]. http://www.daikin.com

[55] DAIKIN Industries, Ltd., "intelligent Touch Controller HTTP INTERFACE OPTION (For Home Automation) COMMISSIONING MANUAL," *Electrical Device Engineering Division, Air Conditioner Manufacturing Department*, Mar. 06, 2009.

[56] "Web of Things." Internet: https://en.wikipedia.org/wiki/Web_of_Things, Jan. 24, 2016 [Feb. 6, 2016].

[57] Bluetooth SIG. https://www.bluetooth.com, 2016.

[58] ZigBee Alliance. http://www.zigbee.org, 2016.

[59] Y. Agarwal, B. Balaji, R. Gupta, J. Lyles, M. Wei, and T. Weng, "Occupancy-driven energy management for smart building automation," In Proc. *the 2nd ACM Work. on Embedded Sensing Syst. for Energy-Efficiency in Building*, Zurich, Nov. 2010, pp. 1-6.

[60] A. Kamilaris, V. Trifa, and A. Pitsillides, "HomeWeb: An application framework for Web-based smart homes," In Proc. *ICT'11*, May 2011.

[61] Yunfei Qu, Hongjie Wang, Shau-Ming Lun, Hsiao-Dong Chiang, and Tao Wang, "Design and implementation of a Web-based Energy Management Application for smart buildings," In Proc. *2013 IEEE Electrical Power & Energy Conference (EPEC)*, Halifax, NS, Aug. 2013.

[62] C. M. Angelopoulos, G. Filios, S. Nikoletseas, D. Patroumpa, T. P. Raptis, and K. Veroutis, "A Holistic IPv6 Test-Bed for Smart, Green Buildings," In Proc. *2013 IEEE International Conference on Communications (ICC)*, Budapest, Jun. 2013.

[63] Matthias Kovatsch, Markus Weiss, and Dominique Guinard, "Embedding internet technology for home automation," In Proc. *2010 IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, Bilbao, Sept. 2010.

[64] Nissanka B. Priyantha, Aman Kansal, Michel Goraczko, and Feng Zhao, "Tiny web services: design and implementation of interoperable and evolvable sensor networks," In Proc. *SenSys'08*, Raleigh, Nov. 5-7, 2008.

[65] Daniel Schachinger and Wolfgang Kastner, "Model-driven integration of building automation systems into Web service gateways," In Proc. *2015 IEEE World Conference on Factory Communication Systems (WFCS)*, Palma de Mallorca, May 2015.

[66] M. Jung, J. Weidinger, C. Crettaz, A. Olivieri, and Y. Bocchi, "A Transparent IPv6 Multi-protocol Gateway to Integrate Building Automation Systems in the Internet of Things," In Proc. *2012 IEEE International Conference on Green Computing and Communications (GreenCom)*, Besancon, Nov. 2012.

[67] Xiaohui Peng, Masahiro Bessho, Noboru Koshizuka, and Ken Sakamura, "A Framework for Peak Electricity Demand Control Utilizing Constraint Programing Method in Smart Building," In Proc. *IEEE 3rd Global Conf. on Consumer Electronics*, Tokyo Japan, Oct. 2014, pp. 744-748.

[68] Peter Jackson, "What Are Expert Systems," in *Introduction to Expert Systems*, 3rd ed. Boston, Addison Wesley, 1998, ch. 1.

[69] T. Gu, H. K. Pung, and D. Q. Zhang, "Toward an OSGi-based infrastructure for context-aware applications," In *IEEE Pervasive Computing*, vol. 3, pp. 66-74, 2004.

[70] H. Doukas, K. D. Patlitzianas, K. Iatropoulos, and J. Psarras, "Intelligent building energy management system using rule sets," In *Building Environ.*, vol. 42, no. 10, pp. 3562-3569, Oct. 2007.

[71] S. Tomic, A. Fensel, and T. Pellegrini, "SESAME Demonstrator: Ontologies, Services and Policies for Energy Efficiency," In Proc. *6th Int. Conf. on Semantic Systems*, Graz, Austria, Sept. 2010, Article No. 24.

[72] A. T. Kaliappan, S. Sathiakumar, and N. Parameswaran, "Flexible Power Consumption Management in Smart Homes," In Proc. *Int. Conf. on Advances in Computing, Commun. and Informatics*, Aug. 2012, Chennai India, pp. 161-167.

[73] V. Kumar, A. Fensel, and P. Fröhlich, "Context Based Adaptation of Semantic Rules in Smart Buildings," In Proc. *iiWAS2013*, Vienna Austria, Dec. 2013.

[74] T. Kawakami, N. Fujita, T. Yoshihisa, and M. Tsukamoto, "An Evaluation and Implementation of Rule-Based Home Energy Management System Using the Rete Algorithm," In *The Scientific World Journal*, Vol. 2014, Article ID 591478, 8 pages.

[75] H. Takatsuka, S. Saiki, S. Matsumoto, and M. Nakamura, "Design and Implementation of Rule-Based Framework for Context-Aware Services with Web Services," In Proc. *iiWAS2014*, Hanoi, Vietnam, Dec. 2014, pp. 233-242.

[76] "Rete algorithm." Available: https://en.wikipedia.org/wiki/Rete_algorithm.

[77] Pablo A. Valiente-Rocha and Adolfo Lozano-Tello, "Ontology-based expert system for home automation controlling," In Proc. *IEA/AIE'10*, Springer-Verlag Berlin, 2010, pp. 661-670.

[78] Christian Reinisch, Wolfgang Granzer, Fritz Praus, and Wolfgang Kastner, "Integration of heterogeneous building automation systems using ontologies," In Proc. *34th Annual Conference of IEEE Industrial Electronics(IECON 2008)*, Orlando, FL, Nov. 2008. pp. 2736-2741.

[79] A. Daouadji, K.-K. Nguyen, M. Lemay, and M. Cheriet, "Ontology-Based Resource Description and Discovery Framework for Low Carbon Grid Networks," In Proc. *2010 First IEEE International Conference on Smart Grid Communications (SmartGridComm)*, Gaithersburg, MD, Oct. 2010. pp. 477-482.

[80] Simon Mayer and Dominique Guinard, "An extensible discovery service for smart things," In Proc. *Second International Workshop on Web of Things (WoT '11)*, Article No. 7, 2011.

[81] Wolfgang Granzer and Wolfgang Kastner, "Information modeling in heterogeneous Building Automation Systems," In Proc. *9th IEEE International Workshop on Factory Communication Systems (WFCS)*, Lemgo, May 2012, pp. 291-300.

[82] Seppo Torma, "Semantic Linking of Building Information Models," In Proc. *IEEE Seventh International Conference on Semantic Computing (ICSC)*, Irvine, Sept. 2013, pp. 412-419.

[83] D. Guinard, V. Trifa and E. Wilde, "A Resource Oriented Architecture for the Web of Things," In Proc. *Internet of Things (IOT)*, Nov. 29-Dec. 1 2010.

[84] M. Asfand-e-yar, A. Kucera, and T. Pitner, "Smart Buildings: Semantic Web Technology for Building Information Model and Building Management System," In Proc. *ICODSE*, Bandung, Nov. 2014.

[85] Shen Bin, Zhang Guiqing, Wang Shaolin, and Wei Dong, "The Development of Management System for Building Equipment Internet of Things," In Proc. *IEEE ICCSN*, May, 2011.

[86] T. G. Stavropoulos, D. Vrakas, D. Vlachava, and N. Bassiliades, "BOnSAI: a Smart Building Ontology for Ambient Intelligence," In Proc. *WIMS'12*, Jun. 2012, Craiova, Romania.

[87] F. Paganelli, S. Turchi, and D. Giuli, "A Web of Things Framework for RESTful Applications and Its Experimentation in a Smart City," In *IEEE Syst. J.*, Issue: 99, Sept. 2014.

[88] Jasvinder Singh, Navid Hassanzadeh, Susan Rea, and Dirk Pesch, "Semantics-empowered middleware implementation for home ecosystem gateway," In Proc. *PERCOM Workshops*, Irvine, Mar. 2014, pp. 449-454.

[89] Gerome Bovet and Jean Hennebert, "Distributed Semantic Discovery for Web-of-Things Enabled Smart Buildings," In Proc. *6th International Conference on New Technologies, Mobility and Security (NTMS)*, Dubai, 2014.

[90] Ji-Yeon Son, Jun-Hee Park, Kyeong-Deok Moon, and Young-Hee Lee, "Resource-aware smart home management system by constructing resource relation graph," In *IEEE Transactions on Consumer Electronics*, vol. 57, no. 3, pp. 1112-1119, Aug. 2011.

[91] Marco Grassi, Michele Nucci, and Francesco Piazza, "Towards an ontology framework for intelligent smart home management and energy saving," In Proc. *2011 IEEE International Symposium on Industrial Electronics (ISIE)*, Gdansk, pp. 1753-1758, Jun. 2011.

[92] MMarkus Jung, Jurgen Weidinger, Wolfgang Kastner, and Alex Olivieri, "Building Automation and Smart Cities: An Integration Approach Based on a Service-Oriented Architecture," In Proc. *27th International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, Barcelona, pp. 1361-1367, Mar. 2013.

[93] Yulia Evchina, Aleksandra Dvoryanchikova, and José L. Martinez Lastra, "Semantic information management for user and context aware smart home with social services," In Proc. *2013 IEEE International Multi-Disciplinary Conference on Cognitive Methods in Situation Awareness and Decision Support (CogSIMA)*, San Diego, pp. 262-268, Feb. 2013.

[94] N. Vicari, E. Wuchner, A. Bröring, and C. Niedermeier, "Engineering and operation made easy - a semantics and service oriented approach to building automation," In Proc. *IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA)*, Luxembourg, Sept. 2015.

[95] Son N. Han, Gyu Myoung Lee, and Noel Crespi, "Semantic Context-Aware Service Composition for Building Automation System," In *IEEE Transactions on Industrial Informatics*, vol. 10, no. 1, pp. 752-761, Mar. 2013.

[96] *OASIS*. (2009, Jul. 1). Devices Profile for Web Services (Ver. 1.1) [Online]. http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01.

[97] *LightWeight Machine to Machine Technical Specification*, Candidate Version 1.0, Open Mobile Alliance, Dec. 2015.

[98] Sebastian Zug, Michael Schulze, Andre Dietrich, and Jorg Kaiser, "Programming abstractions and middleware for building control systems as networks of smart sensors and actuators," in Proc. *2010 IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, Bilbao, Sept. 13-16 2010.

[99] THE HARTMAN COMPANY, OPERATORS' CONTROL LANGUAGE. Rev. 2.2. THE HARTMAN COMPANY. Sept. 1998.

[100] Ken Sinclair (Aug. 2011), The Past and Future of Control Languages [Online]. Available: http://www.automatedbuildings.com/news/aug11/articles/ksin/110720111001ksin.html

[101] Sedona Framework. [Online]. Available: www.sedonadev.org.

[102] Khai N. Truong, Elaine M. Huang, and Gregory D. Abowd, "CAMP: A magnetic poetry interface for end-user programming of capture applications for the home," In Proc. *Ubicomp (2004)*.

[103] Anind K. Dey, Timothy Sohn, Sara Streng, and Justin Kodama, "iCAP:Interactive prototyping of context-aware applications," In Proc. *Pervasive (2006)*.

[104] M. W. Newman, "Now we're cooking: Recipes for end-user service composition in the digital home," Position Paper– *CHI 2006 Workshop IT@Home*, 2006.

[105] Y. Dahl and R.-M. Svendsen, "End-user composition interfaces for smart environments: A preliminary study of usability factors," In *Design, User Experience, and Usability. Theory, Methods, Tools and Practice*, 2011, 118-127.

[106] X. H. Peng, M. Bessho, N. Koshizuka, and K. Sakamura, "The Design and Implementation of a Context-based Energy Management Policy Description Language," In *TRON Symposium 2015*, Tokyo, 9-12 Dec. 2015.

[107] Ishaan Khanna, "Smart Grid Application: Peak Demand Management Trial - The Western Australian Experience," In Proc. *Innovative Smart Grid Technologies Asia (ISGT), IEEE PES*, Nov. 2011, Perth, Australia.

[108] "Building Automation." Internet: https://en.wikipedia.org/wiki/Building_automation, Feb. 14, 2016 [Feb. 22, 2016].

[109] Sakamura, K. Ubiquitous ID Technologeis. (2011). http://www.t-engine.org/ja/wp-content/themes/wp.vicuna/pdf/en_US/UID910-W001-110324.pdf

[110] Sally C. Brailsford, Chris N. Potts, and Barbara M. Smith, "Constraint satisfaction problems: Algorithms and applications," In *European Journal of Operational Research 119*, pp. 557-581, 1999.

[111] Xiaohui Peng (Feb. 2015). Device control rule description language specification (1st ed.) [Online]. Available: http://www.sakamura-lab.org/~peng/dcrdl-spec-v1.pdf

[112] "Compiler." Internet: https://en.wikipedia.org/wiki/Compiler, Jan. 26, 2016 [Feb. 6, 2016].

[113] "JavaCC: The Java Parser Generator." [Online]. Available: https://javacc.java.net [Jul. 28, 2015].

[114] "JJTree Reference Documentation." [Online]. Available: https://javacc.java.net/doc/JJTree.html [Jul. 2015].

[115] "Extended Backus–Naur Form" [Online]. Available: https://en.wikipedia.org/wiki/Yacc [Sept. 2015].

[116] Xiaohui Peng (Aug. 2015). PdlParser.jjt (1st ed.) [Online]. Available: http://www.sakamura-lab.org/~peng/PdlParser.jjt

[117] "Eclipse." [Online]. Available: eclipse.org [Jun. 2015].

[118] Krzysztof Kuchcinski and Radosław Szymanek, JaCoP Library User's Guide, Ver. 4.4, Jan. 5, 2016.

[119] "JFace." [Online]. Available: https://wiki.eclipse.org/JFace [Feb. 2016].

[120] "SWT." [Online]. Available: https://wiki.eclipse.org/SWT [Feb. 2016].

[121] Xiaohui Peng. (Aug. 2015). PdlParser.jjt (1st ed.) [Online]. Available: http://www.sakamura-lab.org/~peng/PdlParser.jjt

[122] Xiaohui Peng. (Aug. 2015). PdlParser.jj (1st ed.) [Online]. Available: http://www.sakamura-lab.org/~peng/PdlParser.jj

[123] E.C. Thom, "The discomfort index, " in *Weatherwise*, vol. 12, no. 2, pp. 57–60, 1959.

[124] D. Orchard, "The Four Rs of Programming Language Design," In Proc. of *Onward! 2011*, Portland, Oregon, USA, Oct. 2011.

[125] Amirhossein Chinaei. Class Lecture, Topic: "Programming Languages - Language Evaluation Criteria." Dept. of Electrical & Computer Engineering, University of Puerto Rico, Mayagüez, Puerto Rico, 2010.

[126] Luigi De Russis and Fulvio Corno, "HomeRules: A Tangible End-User Programming Interface for Smart Homes," In *CHI'15 Extended Abstracts*, Seoul, Korea, Apr. 18-23, 2015.

[127] raml.org, "RAML 1.0 (RC)," [Online]. Available: http://docs.raml.org/specs/1.0/. [Feb. 8, 2016].

[128] wigwag.com, "wigwag RULES," [Online]. Available: http://www.wigwag.com/wigwag_relay.html. [Feb. 8, 2016].

10 APPENDICES

APPENDIX 1: SMART BUILDING RESOURCE DESCRIPTION SCHEMA

Smart **B**uilding **R**esource **D**escription **S**chema

(SBRDS)

Authors	Xiaohui Peng
Laboratory	Sakamura-Koshizuka Laboratory, The University of Tokyo
Version	0.2
Date	February 2016

1. Introduction

Owing to the heterogeneous networks of smart buildings, developing automation applications for the building system is a very difficult job. To develop an automation application, the programmers have to know building physical structures, subsystems, device functionalities and the semantic relationship of internal objects. The Internet of Things has changed the application-programming paradigm of smart buildings, and information acquisition and appliance control become ubiquitous through the web-based open APIs. The application development process based on such integrated web-based service platform becomes easier than before. Therefore, IoT-enabled smart buildings become a homogeneous platform with a programmable architecture. As a result, the heterogeneous networks are hidden by web-based remote APIs, and only data access and device control interfaces are exposed to automation applications.

To enable users to program with smart buildings, we design a resource description schema called SBRDS for smart buildings which abstracts building resources to provide programming supports for upper-layer users. Currently, the SBRDS is mainly designed for EDPL [45] which is a user-friendly energy control policy description language for

IoT-enabled smart buildings.

1.1 Objective

The main target users of SBRDS are system developers who develop web services, especially RESTFul APIs for public access and control of smart buildings. The parsed information from SBRDS can provide suggestions to users for using available resources when they write control policies for the smart building systems. The following information should be included in the SBRDS:

- human readable resource representation;
- describes details of web services, especially RESTful APIs. User described expressions should be able to be translated into executable code that can invoke web services which are issued by system developers.
- describes semantic relations of spaces, objects and services; and
- models the inside objects and spaces.

1.2 Terminology

This specification defines the main terminologies as the following table:

Space	Rooms, corridors, halls, etc.
Appliance	All inside devices.
Service	Web services, access and control interfaces of the building system.
Attribute	String values that represent features of a space or an object.
Resource	Spaces, objects, services, etc.
REST	Representational state transfer [38].

2. Design

2.1 Overview

Basically, there are four main elements in the SBRDS: <building>, <floor>, <space> and <appliance>. Attributes and <functions> are required for each element. They are described in Key-Value pairs. Keys are the semantic description of an attribute or service of a space or object for upper-layer users to refer. The “index” and “ucode”

[109] attributes are required for every space or object.

index: for upper-users to refer to an element; **ucode:** the unique identifier for an object, location, concept in a building.

The SBRDS describes resources in the XML or YAML format. We only give examples in the XML format in this specification. There are mainly four types of information in SBRDS.

Semantic relations of spaces and objects: it describes relations such as the semantic position of a sensor or an appliance, building structures and so forth.

Attributes of spaces and objects: attributes are Key-Value pairs which represent the properties of a space or an object. The "identifier" (we use ucode [109] to identify objects and spaces in this specification) is a unique identification for a space or an object in the building systems. The "index" attribute is a semantic reference for a space or an object. These two attributes are required for each element in the SBRDS.

Service list: services of a space or object are described as the attributes of the <functions> element in the form of Key-Value pairs. The <functions> element is a sub-element of the <building>, <floor>, <space>, and <appliance>.

Detailed description of services: the description of a service includes a return type, a function ID and parameters. The attributes of <parameter> element consist of a name, a data-type, and an index property. The "index" property indicates the position of the parameter in the parameter list. It is encouraged to add a "description" attribute, which describes the explanation of a <function> or <parameter>.

2.2 Elements

2.2.1 Building

<building> is the top element of SBRDS. The attributes of <building> element describe properties of it. For example, the "name" attribute of the <building> element in the following figure shows that the name of the building is "Daiwa Ubiquitous Computing Research Building". The inner element <functions> lists all the available service of the building layer. The value of an attribute of <functions> is the function ID that we defined in the DCRDL specification [111].

```
<?xml version="1.0" encoding="UTF-8"?>
<descriptor>
<building ucode="00001CB0000000000002000000100000" index="DUCRB" name="Daiwa Ubiquitous Computing Research Building">
    <functions system-time="org:skl:names:dcrdl:1.0:function:system-get-time"
               season="org:skl:names:dcrdl:1.0:function:system-get-season"
               sba_get_space_of_lowest_Temperature="org:skl:names:dcrdl:1.0:function:get-space-of-lowest-temperature"
               sba_get_NumberOfOccupancy="org:skl:names:dcrdl:1.0:function:get-occupancy-number-of-space"
               sba_get_PeriodElectricityConsumption="org:skl:names:dcrdl:1.0:function:get-period-electricity-consumption-of-space" />
    <floor  ucode="00001CB0000000000002000000110000" name="Third Floor" index="3F">
    </floor>
</building>
</descriptor>
```

2.2.2 Floor

<floor> is the sub-element of <building>. It describes all the related information of the floor layer.

```
<?xml version="1.0" encoding="UTF-8"?>
<descriptor>
<building>
    <functions/>
    <floor  ucode="00001CB0000000000002000000110000" name="Third Floor" index="3F">
        <functions sba_get_Total_Electricity_Consumption="org:skl:names:dcrdl:1.0:function:get-total-consumption-of-floor"/>
    </floor>
    <floor ucode="00001CB0000000000002000000120000" name="Second Floor" index="2F">
    </floor>
    <floor ucode="00001CB0000000000002000000130000" name="First Floor" index="1F">
</building>
</descriptor>
```

2.2.3 Space

<space> is the sub-element of <floor>. It describes all the related information of the space layer.

```
<?xml version="1.0" encoding="UTF-8"?>
<descriptor>
<building>
    <functions/>
    <floor>
        <space  ucode="00001CB0000000000002000000111000" name="Sakamura-Koshizuka Lab" index="3L-5" type="student_room" >
            <functions sba_get_NumberOfOccupancy="org:skl:names:dcrdl:1.0:function:get-occupancy-number-of-space"
                sba_get_PeriodElectricityConsumption="org:skl:names:dcrdl:1.0:function:get-period-electricity-consumption-of-space"
                sba_get_temperature="org:skl:names:dcrdl:1.0:function:get-space-temperature"
                sba_get_humidity="org:skl:names:dcrdl:1.0:function:get-space-humidity"
                sba_get_occupancy="org:skl:names:dcrdl:1.0:function:get-space-occupancy"
                sba_get_discomfortIndex="org:skl:names:dcrdl:1.0:function:get-space-discomfort-index"
                sba_get_acs_by_space="org:skl:names:dcrdl:1.0:function:get-acs-by-spaces"
                sba_get_lights_by_space="org:skl:names:dcrdl:1.0:function:get-acs-by-spaces" />
        </space>

    </floor>
</building>
</descriptor>
```

2.2.4 Appliance

<appliance> is a sub-element of <building>, <floor> or <space> elements. It describes all the related information of a device.

```
<?xml version="1.0" encoding="UTF-8"?>
<descriptor>
<building>
    <functions/>
    <floor>
        <space>
            <functions/>
            <appliance ucode="00001CB000000000000020000000111001" name="AirConditioner 1" index="
            AirConditioner1" type="air_conditioner" >
                <functions sba_ctr_ac_on="org:skl:names:dcrdl:1.0:function:control-ac-on"
                sba_ctr_ac_off="org:skl:names:dcrdl:1.0:function:control-ac-off"
                sba_ctr_ac_on_with_setpoint="org:skl:names:dcrdl:1.0:function:control-ac-on-with-setpoint"
                sba_ctr_ac_increase_setpoint="org:skl:names:dcrdl:1.0:function:control-ac-setpoint-increase"
                sba_ctr_ac_decrease_setpoint="org:skl:names:dcrdl:1.0:function:control-ac-setpoint-decrease"
                sba_get_ac_setpoint="org:skl:names:dcrdl:1.0:function:get-ac-setpoin" />
            </appliance>
        </space>
    </floor>
</building>
```

2.2.5 sbapi

The <sbapi> element describes details of all the available services that the target building can provide. It consists of multiple <function> elements which describe the structure of services such as function id, return-type, description, and the <parameter> sub-element.

```
<?xml version="1.0" encoding="UTF-8"?>
<descriptor>
<building ucode="00001CB000000000000020000000100000" name="Daiwa Ubiquitous Computing Research Building">
</building>
<sbapi>
    <function return-type="org:skl:names:dcrdl:1.0:data-type:void" id="org:skl:names:dcrdl:1.
    0:function:control-ac-on" description="Turn on an air conditioner with the specified ucode.">
        <parameter name ="ucode" data-type="http://www.w3.org/2001/XMLSchema#String" description="The ucode
        of the target air conditioner."/>
    </function>
    <function return-type="org:skl:names:dcrdl:1.0:data-type:void" id="org:skl:names:dcrdl:1.
    0:function:control-ac-off" description="Turn on an air conditioner with the specified ucode.">
        <parameter name ="ucode" data-type="http://www.w3.org/2001/XMLSchema#String" description="The ucode
        of the target air conditioner."/>
    </function>
    <function return-type="http://www.w3.org/2001/XMLSchema#Float" id="org:skl:names:dcrdl:1.0:function:get-
    period-electricity-consumption-of-space" description="Get the electricity consumption of target in a
    period">
        <parameter name ="ucode" data-type="http://www.w3.org/2001/XMLSchema#String" description="ucode
        string of the target" />
        <parameter name ="period" data-type="http://www.w3.org/2001/XMLSchema#String" description="A time
        string, for example 15 minutes can be represent as: 00:15:00" />
    </function>
</sbapi>
</descriptor>
```

APPENDIX 2: EPDLPARSER.JJT

```
/**
 * JJTree template file created by SF JavaCC plugin 1.5.28+ wizard for JavaCC 1.5.0+
 */
options
{
  STATIC=false;
  NODE_CLASS="PdlNode";
}
PARSER_BEGIN(PdlParser)
package org.skl.pdl.parser;
public class PdlParser
{
  public static void main(String args [])
  {
    System.out.println("Reading from standard input...");
    System.out.print("Enter an expression like ¥"1+(2+3)*var;¥" :");
    PdlParser parser = new PdlParser(System.in);
    try
    {
      SimpleNode n = parser.Start();
      n.dump("");
      System.out.println("Thank you.");
    }
    catch (Exception e)
    {
      System.out.println("Oops.");
      System.out.println(e.getMessage());
    }
  }
}
PARSER_END(PdlParser)
SKIP :
{
  " "
| "¥t"
| "¥n"
| "¥r"
| < "//" (~[ "¥n", "¥r" ])*
    (
      "¥n"
    | "¥r"
    | "¥r¥n"
    ) >
| < "/*" (~[ "*" ])* "*"
    (
      ~[ "/" ] (~[ "*" ])* "*"
    )*
    "/" >
}
```

```
TOKEN : /* Keywords*/
{
  < POLICY:"policy">
  | < RULE:"rule">
  | < WHEN:"when">
  | < THEN:"then">
  | < INT:"int">
  | < FLOAT:"float">
  | < FOREACH:"foreach">
  | < WITHIN:"within">
  | < IF:"if">
  | < CHAR: "char" >
  | < BYTE: "byte" >
  | < EXTENDS: "extends" >
  | < SUPER: "super" >
  | < DOUBLE: "double" >
  | < BOOLEAN: "boolean" >
  | < BREAK: "break" >
  | < FALSE: "false" >
  | < FOR: "for" >
  | < LONG: "long" >
  | < NULL: "null" >
  | < SHORT: "short" >
  | < THIS: "this" >
  | < TRUE: "true" >
  | < BUILDING: "Building">
}
TOKEN : /* LITERALS */
{
  < INTEGER_LITERAL :
    < DECIMAL_LITERAL > ([ "l", "L" ])?
  | < HEX_LITERAL > ([ "l", "L" ])?
  | < OCTAL_LITERAL > ([ "l", "L" ])?
    >
| < #DECIMAL_LITERAL : [ "1"-"9" ] ([ "0"-"9" ])* >
| < #HEX_LITERAL : "0" [ "x", "X" ] ([ "0"-"9", "a"-"f", "A"-"F" ])+ >
| < #OCTAL_LITERAL : "0" ([ "0"-"7" ])* >
|
  < FLOATING_POINT_LITERAL:
        (["0"-"9"])+ "." (["0"-"9"])* (<EXPONENT>)? (["f","F","d","D"])?
      | "." (["0"-"9"])+ (<EXPONENT>)? (["f","F","d","D"])?
      | (["0"-"9"])+ <EXPONENT> (["f","F","d","D"])?
      | (["0"-"9"])+ (<EXPONENT>)? ["f","F","d","D"]
  >
|
  < #EXPONENT: ["e","E"] (["+","-"])? (["0"-"9"])+ >
|
  < CHARACTER_LITERAL:
      "'"
      (   (~["'","¥¥","¥n","¥r"])
        | ("¥¥"
           ( ["n","t","b","r","f","¥¥","'","¥""]
           | ["0"-"7"] ( ["0"-"7"] )?
           | ["0"-"3"] ["0"-"7"] ["0"-"7"]
```

```
          )
        )
      )
      "'"
  >
|
  < STRING_LITERAL:
      "¥""
      (   (~["¥"","¥¥","¥n","¥r"])
        | ("¥¥"
           ( ["n","t","b","r","f","¥¥","'","¥""]
           | ["0"-"7"] ( ["0"-"7"] )?
           | ["0"-"3"] ["0"-"7"] ["0"-"7"]
           )
        )
      )*
      "¥""
  >
}

TOKEN : /* sign */
{
  < COMMA:"," >
  |< SEMICOLON:";">
  |< LPAR:"(">
  |< RPAR:")">
  |< ASSIGN:"=">
  |< LSBRA:"[">
  |< RSBRA:"]">
  |< LBRA:"{">
  |< RBRA:"}">
  |< EQU:"==">
  |< NEQ:"!=">
  |< GTR:">">
  |< GEQ:">=">
  |< LSS:"<">
  |< LEQ:"<=">
  |< NOT:"!">
  |< AND:"&&">
  |< OR:"||">
}
TOKEN : /* IDENTIFIERS */
{
  < IDENTIFIER :
    < LETTER >
    (
      < LETTER >
    | < DIGIT >
    )* >
| < #LETTER : [ "_", "a"-"z", "A"-"Z" ] >
| < #DIGIT : [ "0"-"9" ] >
}
SimpleNode Start() :
{}
```

```
{
  < POLICY >< IDENTIFIER >< LBRA >
  (
    PolicyAttributeDefine()
  | Rule()
  | RuleForeach()
  )*
  < RBRA >
  < EOF >
  {
    return jjtThis;
  }
}

void VariableDeclaration() :
{}
{
  ((< INT >
          | < FLOAT>)["[]"]) Identifier()(< COMMA >Identifier())*
}
void VariableAssignment() :
{}
{
  < IDENTIFIER >("."< IDENTIFIER >)* < ASSIGN > (< INT >|< FLOAT >|< STRING_LITERAL >)
}

void PolicyAttributeDefine() :
{Token t;}
{
  < POLICY >"."t=< IDENTIFIER > {jjtThis.value = String.valueOf(t.image); }
   < ASSIGN > (t=< INT >{jjtThis.value += "="+Integer.valueOf(t.image); }
   |t=< FLOAT >{jjtThis.value += "="+Float.valueOf(t.image); }
   |t=< STRING_LITERAL >{String temp = String.valueOf(t.image);jjtThis.value += "="+temp.substring(1,
temp.length()-1);})
}

void Rule() :
{}
{
  < RULE > Identifier() < WHEN > < LBRA > ConditionSet() < RBRA > < THEN > < LBRA > ActionSet() <
RBRA >
}
/*  ConditionSet  */
void ConditionSet() :
{}
{
  OrCondition()
}

String RelationSign() :
{Token t;}
{
  ( t=< EQU >{jjtThis.value = String.valueOf(t.image); return t.image;}
  |t=< NEQ >{jjtThis.value = String.valueOf(t.image); return t.image;}
```

```
  |t=< GTR >{jjtThis.value = String.valueOf(t.image); return t.image;}
  |t=< GEQ >{jjtThis.value = String.valueOf(t.image); return t.image;}
  |t=< LSS >{jjtThis.value = String.valueOf(t.image); return t.image;}
  |t=< LEQ >{jjtThis.value = String.valueOf(t.image); return t.image;}
  |t=< NOT >{jjtThis.value = String.valueOf(t.image); return t.image;} )
}

/*  ActionSet  */
void ActionSet() :
{}
{
  (SbaApi())+
| (ActionSetForeach())*
//  (Expression())+
}
void RuleForeach() :
{}
{
  < FOREACH > Identifier() < WITHIN > SbaApi() < LBRA > (IfStatement()|Rule())+ < RBRA >
}
void IfStatement() :
{}
{
  < IF >< LPAR >ConditionSet()< RPAR >< LBRA > (Rule()|RuleForeach())+< RBRA >
}
void ActionSetForeach() :
{}
{
  < FOREACH > Identifier() < WITHIN > SbaApi() < LBRA > (SbaApi())+ < RBRA >
}
void OrCondition() :
{Token t;}
{
  AndCondition()(t=< OR >{jjtThis.value = String.valueOf(t.image);} AndCondition())*
}
void AndCondition() :
{Token t;}
{
  Condition()(t=< AND >{jjtThis.value = String.valueOf(t.image);} Condition())*
}
void Condition() :
{}
{
  (< LPAR >OrCondition() < RPAR >|UnaryExpression())(RelationSign() Literal())*
}
void UnaryExpression() :
{}
{
  LOOKAHEAD(SbaApi())SbaApi()
  |Identifier()
}
void SbaApi() :
{}
{
```

```
  (< BUILDING >|MemberIdentifier())("."MemberIdentifier())* [Arguments()]

}
void MemberIdentifier() :
{}
{
  Identifier()[< LSBRA >Literal()< RSBRA >]
}
void Literal():
{Token t;}
{
  t=< INTEGER_LITERAL > {jjtThis.value = Long.valueOf(t.image);}
  |t=< FLOATING_POINT_LITERAL > {jjtThis.value = Double.valueOf(t.image);}
  |t=<CHARACTER_LITERAL> {jjtThis.value = String.valueOf(t.image);}
  |t=<STRING_LITERAL> {String temp =
String.valueOf(t.image).replaceFirst("¥"","");jjtThis.value=temp.substring(0, temp.length()-1);}
}
void Arguments():
{}
{
  < LPAR > [ Literal() ( "," Literal() )* ] < RPAR >
}
void Identifier() :
{Token t;}
{
  t=< IDENTIFIER >{jjtThis.value = String.valueOf(t.image);}
}
void Number() :
{Token t;}
{
  t=< INTEGER_LITERAL > {jjtThis.value = Long.valueOf(t.image);}
  |t=< FLOATING_POINT_LITERAL > {jjtThis.value = Double.valueOf(t.image);}
}
void Integer() :
{Token t;}
{
  t=< INTEGER_LITERAL > {jjtThis.value = Double.valueOf(t.image);}
}
void Float() :
{Token t;}
{
  t=< FLOATING_POINT_LITERAL > {jjtThis.value = Double.valueOf(t.image);}
}
```

APPENDIX 3: THREE CONTROL POLICIES WRITTEN IN DCRDL

1. Peak Consumption Control Policy

```
<Policy PolicyId="org:sakamura-lab:names:dcrdl:1.0:policyid:1"
        PolicyName="Peak Electricity Consumption Control"
        Period="900" Priority="98">
 <Rule RuleId="org:sakamura-lab:names:dcrdl:1.0:ruleid:1" >
  <ConditionSet>
   <Condition>
    <Subject>
     <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:get-building-electricity-consumption">
      <Parameter Value="00:15:00"
            DataType="http://www.w3.org/2001/XMLSchema#string" />
     </Function>
    </Subject>
    <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:float-greater-than">
     <Parameter Value="0.60"
           DataType="http://www.w3.org/2001/XMLSchema#float"/>
    </Function>
   </Condition>
  </ConditionSet>
  <ActionSet>
   <Action>
    <Target>
     <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:get-acs-by-spaces" >
      <Parameter>
       <Function FunctionId=
                 "org:sakamura-lab:names:dcrdl:1.0:function:get-space-by-highest-temperature-with-ac-on" />
      </Parameter>
      <Parameter Value="on"
            DataType="org:sakamura-lab:names:dcrdl:1.0:constant:status">
      </Parameter>
     </Function>
    </Target>
    <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:control-ac-setpoint-decrease">
     <Parameter Value="2"
              DataType="http://www.w3.org/2001/XMLSchema#integer" />
    </Function>
   </Action>
  </ActionSet>
 </Rule>
  <Rule RuleId="org:sakamura-lab:names:dcrdl:1.0:ruleid:2" >
  <ConditionSet>
   <Condition>
    <Subject>
```

```
      <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:get-space-electricity-consumption">
       <Parameter Value="X0001C000000000000020000000D649"
            DataType="org:sakamura-lab:names:dcrdl:1.0:DataType:ucode" />
       <Parameter Value="00:15:00"
            DataType="http://www.w3.org/2001/XMLSchema#string" />
      </Function>
     </Subject>
     <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:float-greater-than">
      <Parameter Value="0.20"
          DataType="http://www.w3.org/2001/XMLSchema#float"/>
     </Function>
    </Condition>
  </ConditionSet>
  <ActionSet>
   <Action>
    <Target TargetId="00001C000000000000020000000D448A,00001C000000000000020000000D448B,
                      00001C000000000000020000000D448C,00001C000000000000020000000D448D"
          DataType="org:sakamura-lab:names:dcrdl:1.0:DataType:ucode-list" />
    <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:control-light-off" />
   </Action>
  </ActionSet>
</Rule>
  <Rule RuleId="org:sakamura-lab:names:dcrdl:1.0:ruleid:3" >
  <ConditionSet>
   <Condition>
    <Subject>
     <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:get-ac-setpoint">
      <Parameter Value="X0001C000000000000020000000D640"
            DataType="org:sakamura-lab:names:dcrdl:1.0:DataType:ucode" />
     </Function>
    </Subject>
    <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:float-less-than-or-equal">
     <Parameter Value="22.00"
          DataType="http://www.w3.org/2001/XMLSchema#float"/>
    </Function>
   </Condition>
  </ConditionSet>
  <ActionSet>
   <Action>
    <Target TargetId="X0001C000000000000020000000D640"
          DataType="org:sakamura-lab:names:dcrdl:1.0:DataType:ucode" />
    <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:control-ac-off" />
   </Action>
  </ActionSet>
</Rule>
<Rule RuleId="org:sakamura-lab:names:dcrdl:1.0:ruleid:4" >
  <ConditionSet>
   <Condition>
    <Subject>
     <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:get-ac-setpoint">
      <Parameter Value="X0001C000000000000020000000D641"
            DataType="org:sakamura-lab:names:dcrdl:1.0:DataType:ucode" />
     </Function>
    </Subject>
```

```
    <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:float-less-than-or-equal">
     <Parameter Value="22.00"
          DataType="http://www.w3.org/2001/XMLSchema#float"/>
    </Function>
   </Condition>
  </ConditionSet>
  <ActionSet>
   <Action>
    <Target TargetId="X0001C00000000000020000000D641"
          DataType="org:sakamura-lab:names:dcrdl:1.0:DataType:ucode" />
    <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:control-ac-off" />
   </Action>
  </ActionSet>
</Rule>
<Rule RuleId="org:sakamura-lab:names:dcrdl:1.0:ruleid:5" >
  <ConditionSet>
   <Condition>
    <Subject>
     <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:get-ac-setpoint">
      <Parameter Value="X0001C00000000000020000000D642"
          DataType="org:sakamura-lab:names:dcrdl:1.0:DataType:ucode" />
     </Function>
    </Subject>
    <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:float-less-than-or-equal">
     <Parameter Value="22.00"
          DataType="http://www.w3.org/2001/XMLSchema#float"/>
    </Function>
   </Condition>
  </ConditionSet>
  <ActionSet>
   <Action>
    <Target TargetId="X0001C00000000000020000000D642"
          DataType="org:sakamura-lab:names:dcrdl:1.0:DataType:ucode" />
    <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:control-ac-off" />
   </Action>
  </ActionSet>
</Rule>
<Rule RuleId="org:sakamura-lab:names:dcrdl:1.0:ruleid:6" >
  <ConditionSet>
   <Condition>
    <Subject>
     <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:get-ac-setpoint">
      <Parameter Value="X0001C00000000000020000000D643"
          DataType="org:sakamura-lab:names:dcrdl:1.0:DataType:ucode" />
     </Function>
    </Subject>
    <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:float-less-than-or-equal">
     <Parameter Value="22.00"
          DataType="http://www.w3.org/2001/XMLSchema#float"/>
    </Function>
   </Condition>
  </ConditionSet>
  <ActionSet>
   <Action>
```

```
      <Target TargetId="X0001C000000000000020000000D643"
                Datatype="org:sakamura-lab:names:dcrdl:1.0:DataType:ucode" />
      <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:control-ac-off" />
     </Action>
   </ActionSet>
  </Rule>
</Policy>
```

2. Step Tariffs Policy

```
<Policy PolicyId="org:sakamura-lab:names:dcrdl:1.0:policyid:2"
        PolicyName="step tariffs policy"
        Period="900" Priority="97">
 <Rule RuleId="org:sakamura-lab:names:dcrdl:1.0:ruleid:1" >
  <ConditionSet CombineMethod="org:sakamura-lab:names:dcrdl:1.0:CombineMethod:and">
   <Condition>
    <Subject>
     <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:get-realtime-electricity-price" />
    </Subject>
    <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:float-greater-than-or-equal">
     <Parameter Value="500"
          DataType="http://www.w3.org/2001/XMLSchema#float"/>
    </Function>
   </Condition>
   <Condition>
    <Subject>
     <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:get-realtime-electricity-price" />
    </Subject>
    <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:float-less-than">
     <Parameter Value="1000"
          DataType="http://www.w3.org/2001/XMLSchema#float"/>
    </Function>
   </Condition>
  </ConditionSet>
  <ActionSet>
   <Action>
    <Target DataType="org:sakamura-lab:names:dcrdl:1.0:DataType:ucode-list">
     <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:get-acs-by-spaces" >
      <Parameter>
       <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:get-space-by-highest-temperature" />
      </Parameter>
      <Parameter Value="on"
           DataType="org:sakamura-lab:names:dcrdl:1.0:constant:status">
      </Parameter>
     </Function>
    </Target>
    <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:control-ac-setpoint-decrease" >
     <Parameter Value="2"
          DataType="http://www.w3.org/2001/XMLSchema#float" />
    </Function>
```

```
    </Action>
  </ActionSet>
</Rule>
<Rule RuleId="org:sakamura-lab:names:dcrdl:1.0:ruleid:2" >
  <ConditionSet>
    <Condition>
      <Subject>
        <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:get-realtime-electricity-price" />
      </Subject>
      <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:float-greater-than-or-equal">
        <Parameter Value="1000"
              DataType="http://www.w3.org/2001/XMLSchema#float"/>
      </Function>
    </Condition>
  </ConditionSet>
  <ActionSet>
    <Action>
      <Target DataType="org:sakamura-lab:names:dcrdl:1.0:DataType:ucode-list">
        <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:get-acs-by-spaces" >
          <Parameter>
            <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:get-space-by-highest-temperature" />
          </Parameter>
          <Parameter Value="on"
               DataType="org:sakamura-lab:names:dcrdl:1.0:constant:status">
          </Parameter>
        </Function>
      </Target>
      <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:control-ac-off" />
    </Action>
  </ActionSet>
</Rule>
<Rule RuleId="org:sakamura-lab:names:dcrdl:1.0:ruleid:3" >
  <ConditionSet CombineMethod="org:sakamura-lab:names:dcrdl:1.0:CombineMethod:and">
    <Condition>
      <Subject>
        <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:get-realtime-electricity-price" />
      </Subject>
      <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:float-greater-than-or-equal">
        <Parameter Value="1000"
              DataType="http://www.w3.org/2001/XMLSchema#float"/>
      </Function>
    </Condition>
    <Condition>
      <Subject>
        <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:get-space-humidity">
          <Parameter Value="X0001C000000000000020000000D648"
              DataType="org:sakamura-lab:names:dcrdl:1.0:DataType:ucode"/>
        </Function>
      </Subject>
      <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:float-greater-than">
        <Parameter Value="30.0"
              DataType="http://www.w3.org/2001/XMLSchema#float"/>
      </Function>
    </Condition>
```

```
  </ConditionSet>
  <ActionSet>
   <Action>
    <Target TargetId="X0001C00000000000020000000D540"
      DataType="org:sakamura-lab:names:dcrdl:1.0:DataType:ucode-list" />
    <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:control-humidifier-off" />
   </Action>
  </ActionSet>
 </Rule>
  <Rule RuleId="org:sakamura-lab:names:dcrdl:1.0:ruleid:2" >
  <ConditionSet>
   <Condition>
    <Subject>
     <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:get-realtime-electricity-price" />
    </Subject>
    <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:float-greater-than-or-equal">
     <Parameter Value="1500"
          DataType="http://www.w3.org/2001/XMLSchema#float"/>
    </Function>
   </Condition>
  </ConditionSet>
  <ActionSet>
   <Action>
    <Target TargetId="00001C00000000000020000000D448A,00001C00000000000020000000D448B,
                00001C00000000000020000000D448C,00001C00000000000020000000D448D"
          DataType="org:sakamura-lab:names:dcrdl:1.0:DataType:ucode-list" />
    <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:control-light-off" />
   </Action>
  </ActionSet>
 </Rule>
</Policy>
```

3. Discomfort Index Control Policy

```
<Policy PolicyId="org:sakamura-lab:names:dcrdl:1.0:policyid:3"
    PolicyName="Discomfort Index Control Policy"
    Period="300" Priority="99">
 <Rule RuleId="org:sakamura-lab:names:dcrdl:1.0:ruleid:1" >
  <ConditionSet CombineMethod="org:sakamura-lab:names:dcrdl:1.0:CombineMethod:and">
   <Condition>
    <Subject>
     <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:get-space-discomfort-index">
      <Parameter Value="X0001C00000000000020000000D648"
          DataType="org:sakamura-lab:names:dcrdl:1.0:DataType:ucode" />
     </Function>
    </Subject>
    <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:float-greater-than-or-equal">
     <Parameter Value="21.0"
          DataType="http://www.w3.org/2001/XMLSchema#float"/>
    </Function>
   </Condition>
   <Condition>
    <Subject>
```

```
      <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:get-space-temperature">
       <Parameter Value="X0001C000000000000020000000D648"
            DataType="org:sakamura-lab:names:dcrdl:1.0:DataType:ucode" />
      </Function>
     </Subject>
     <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:float-greater-than-or-equal">
      <Parameter Value="24.0"
           DataType="http://www.w3.org/2001/XMLSchema#float"/>
     </Function>
    </Condition>
  </ConditionSet>
  <ActionSet>
    <Action>
     <Target TargetId="X0001C000000000000020000000D640,X0001C000000000000020000000D641"
          DataType="org:sakamura-lab:names:dcrdl:1.0:DataType:ucode" />
     <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:control-ac-off" />
    </Action>
  </ActionSet>
</Rule>
<Rule RuleId="org:sakamura-lab:names:dcrdl:1.0:ruleid:2" >
  <ConditionSet>
    <Condition>
     <Subject>
      <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:get-space-temperature">
       <Parameter Value="X0001C000000000000020000000D648"
            DataType="org:sakamura-lab:names:dcrdl:1.0:DataType:ucode" />
      </Function>
     </Subject>
     <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:float-less-than-or-equal">
      <Parameter Value="23.0"
           DataType="http://www.w3.org/2001/XMLSchema#float"/>
     </Function>
    </Condition>
  </ConditionSet>
  <ActionSet>
    <Action>
     <Target TargetId="X0001C000000000000020000000D640,X0001C000000000000020000000D641"
          DataType="org:sakamura-lab:names:dcrdl:1.0:DataType:ucode-list" />
     <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:control-ac-on" />
    </Action>
  </ActionSet>
</Rule>
<Rule RuleId="org:sakamura-lab:names:dcrdl:1.0:ruleid:3" >
  <ConditionSet CombineMethod="org:sakamura-lab:names:dcrdl:1.0:CombineMethod:and">
    <Condition>
     <Subject>
      <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:get-space-humidity">
       <Parameter Value="X0001C000000000000020000000D648"
            DataType="org:sakamura-lab:names:dcrdl:1.0:DataType:ucode" />
      </Function>
     </Subject>
     <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:float-less-than">
      <Parameter Value="42.0"
           DataType="http://www.w3.org/2001/XMLSchema#float"/>
```

```
      </Function>
    </Condition>
  </ConditionSet>
  <ActionSet>
    <Action>
      <Target TargetId="X0001C000000000000020000000D540"
            DataType="org:sakamura-lab:names:dcrdl:1.0:DataType:ucode" />
      <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:control-humidifier-on" />
    </Action>
  </ActionSet>
 </Rule>
 <Rule RuleId="org:sakamura-lab:names:dcrdl:1.0:ruleid:4" >
  <ConditionSet CombineMethod="org:sakamura-lab:names:dcrdl:1.0:CombineMethod:and">
    <Condition>
      <Subject>
        <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:get-space-discomfort-index">
          <Parameter Value="X0001C000000000000020000000D648"
                DataType="org:sakamura-lab:names:dcrdl:1.0:DataType:ucode" />
        </Function>
      </Subject>
      <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:float-greater-than">
        <Parameter Value="21.0"
              DataType="http://www.w3.org/2001/XMLSchema#float"/>
      </Function>
    </Condition>
    <Condition>
      <Subject>
        <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:get-space-humidity">
          <Parameter Value="X0001C000000000000020000000D648"
                DataType="org:sakamura-lab:names:dcrdl:1.0:DataType:ucode" />
        </Function>
      </Subject>
      <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:float-greater-than">
        <Parameter Value="45.0"
              DataType="http://www.w3.org/2001/XMLSchema#float"/>
      </Function>
    </Condition>
  </ConditionSet>
  <ActionSet>
    <Action>
      <Target TargetId="X0001C000000000000020000000D540"
            DataType="org:sakamura-lab:names:dcrdl:1.0:DataType:ucode" />
      <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:control-humidifier-off" />
    </Action>
  </ActionSet>
 </Rule>
</Policy>
```

APPENDIX 4: DCRDL SPECIFICATION

Device **C**ontrol **R**ule **D**efinition **L**anguage Specification

Authors	Xiaohui Peng
Laboratory	Sakamura-Koshizuka Laboratory, The University of Tokyo
Version	v1.1
Date	February 2016

1. Introduction

Ubiquitous computing describes a vision that all kinds of devices are connected in "a loosely-coupled manner" [31] to provide a smart environment that can automatically adapt to users' behaviors. A smart building is a typical implementation of the ubiquitous computing. "The ideal application scenario for smart buildings considers the user as the focus of a pervasive environment augmented with sensors and actuators where an intelligent system monitors environmental conditions and takes the proper actions to satisfy users' requirements" [33]. Devices are becoming smarter in the age of the Internet of Things and fine-grained environment information can be collected and shared by the connected devices that deployed in smart buildings.

The complexity of the management system of smart buildings keeps growing dues to the advancements of technology. In order to efficiently manage smart devices to optimize energy efficiency, smart devices should be operated diversely according to the real-time context. Users should be able to perform the personalized, precise, flexible and collaborative controls based on various contexts in the building systems. Therefore, we designed a rule definition language called DCRDL for context-based device control in smart buildings. Using DCRDL, users can write fine-grained device control rules to

control device behaviors to adapt to the context. The energy efficiency will be potentially improved by writing personalized, precise control rules using DCRDL to control building systems.

2. Language Model

The elements of DCRDL are:

- **Policy;**
- **Rule;**
- **ConditionSet;**
- **Condition**
- **ActionSet**
- **Action**
- **Subject**
- **Target**
- **Function**
- **Parameter**

2.1 Basic Rules

- A rule consists of a ConditionSet and an ActionSet;
- ConditionSet contains one or more conditions;
- Condition consists of a subject, comparison function;
- The action section contains one or more actions, and all the actions should be executed when the corresponding ConditionSet is evaluated to be true;
- An action consists of targets and action functions;
- Conditions should be wrapped up by the <ConditionSet> tag, and a “CombineMethod” attribute in ConditionSet indicates the relationship of the conditions in this ConditionSet.

- Function elements describe the details of functions that defined in the section six of this specification. Functions will increase gradually with the increasing available public services of smart buildings. We believe that more and more building functions will be exposed through web-based APIs in the near future. More functions will be defined in future versions of this specification.

2.2 Elements

Policy

Multiple rules constitute a policy which generally used to achieve a certain control goal. The policy is interpreted as an agent that running in the policy agent system of the proposed programming framework. Several attributes are defined in the <policy> element:

PolicyId	MUST	The ID of a policy.
PolicyName	OPTIONAL	The name of the policy.
Period	OPTIONAL	An integer value that indicates the polling cycle of the policy in seconds. If Mode is set to *"event"*, this attribute should be ignored by the interpreter.
Priority	MUST	An integer value that indicates the priority of the policy agent.
Mode	MUST	The running mode of the policy. There are two types of models defined in DCRDL: *"polling"* and *"event"* and the default model is polling. The event mode means that the policy will be triggered by an event from the building system. For example, if a rule defined in a policy concerns the temperature of a room, the policy should be triggered when the temperature of that room changed.

```
<Policy PolicyId="org:sakamura-lab:names:dcrdl:1.0:policyid:1"
        PolicyName="Discomfort Index Control Policy"
        Period="900"
        Mode="polling">
  <Rule RuleId="org:sakamura-lab:names:dcrdl:1.0:ruleid:1" priority="">
    ...
  </Rule>
  <Rule RuleId="org:sakamura-lab:names:dcrdl:1.0:ruleid:2" priority="">
    ...
  </Rule>
  <Rule RuleId="org:sakamura-lab:names:dcrdl:1.0:ruleid:3" priority="">
    ...
  </Rule>
</Policy>
```

Rule

A rule is the basic element of DCRDL. It follows the CSAS (i.e., ConditionSet-ActionSet) format. In other words, a DCRDL rule consists of a ConditionSet and an ActionSet. When the ConditionSet is evaluated to be true, the actions defined in ActionSet should be executed. The priority is designed for the rule conflicting resolution issue in the future.

```
<Rule RuleId="org:sakamura-lab:names:dcrdl:1.0:ruleid:1" priority="">
  <ConditionSet CombineMethod="org:sakamura-lab:names:dcrdl:1.0:CombineMethod:and">
    <Condition>...</Condition>
    <ConditionSet>...</ConditionSet>
  </ConditionSet>
  <ActionSet>
    <Action>...</Action>
    <Action>...</Action>
  </ActionSet>
</Rule>
```

ConditionSet

The ConditionSet consists of multiple conditions and a "*CombineMethod*" attribute indicates the relation between the inner conditions. The ConditionSet is designed recursively to represent the complex context in smart buildings.

CombineMethod	"and", "or". The default value is "or".

```
<ConditionSet CombineMethod="org:sakamura-lab:names:dcrdl:1.0:CombineMethod:and">
   <Condition>A</Condition>
      <ConditionSet CombineMethod="org:sakamura-lab:names:dcrdl:1.0:CombineMethod:or">
         <Condition>B</Condition>
         <Condition>C</Condition>
      </ConditionSet>
</ConditionSet>
```

The above code of ConditionSet represents the logical combination:

$$A \wedge (B \vee C)$$

Condition

A condition consists of a <Subject> element and a <Function> element. <Subject> contains a <Function> element which gets information from the building system. For example, *get-space-temperature* function in a <Subject> element refers to the service of obtaining the temperature of a space. The comparison <Function> in a condition with a parameter is responsible for evaluating the condition.

```
<Condition>
   <Subject>
      <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:get-space-temperature">
         <Parameter Value="00001C00000000000020000000D648"
                    DataType="org:sakamura-lab:names:dcrdl:1.0:DataType:ucode" />
      </Function>
   </Subject>
   <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:float-larger-than">
      <Parameter Value="26" DataType="http://www.w3.org/2001/XMLSchema#float"/>
   </Function>
</Condition>
```

The above code represents the logical expression that the temperature of the room identified by the ucode “00001C00000000000020000000D648” is larger than 26 ℃. When the temperature of this room exceeds 26 ℃, this condition is evaluated to be true.

ActionSet

ActionSet in a rule defines the actions of related devices which should be executed when the corresponding ConditionSet is evaluated to be true. It contains multiple actions.

Action

The <Action> element defines an action for one or multiple devices. The inner TargetID can be plural. For example, we can define an action that turns off all the air conditioners whose set-point is less than 20 ℃. The following code represents that increases the set-point of air conditioner: 00001C00000000000020000000D448A by 1 ℃.

```
<Action>
   <Target TargetID="org:sakamura-lab:names:dcrdl:1.0:ucode:00001C00000000000020000000D448A"
           DataType="org:sakamura-lab:names:dcrdl:1.0:DataType:ucode"/>
      <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:control-ac-setpoint-increase">
         <Parameter Value="1" DataType="http://www.w3.org/2001/XMLSchema#integer"/>
      </Function>
</Action>
```

Function

The <Function> element describes the details of a service. The function is also designed recursively. The following code represents that *get-acs-by-spaces* service uses the return value of *get-space-by-highest-temperature* service as the parameter to get ucodes of air conditioners in the room where the temperature is highest of the building.

```
<Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:get-acs-by-spaces" >
   <Parameter DataType="org:sakamura-lab:names:dcrdl:1.0:DataType:ucode-list">
      <Function FunctionId="org:sakamura-lab:names:dcrdl:1.0:function:get-space-by-highest-temperature" />
   </Parameter>
</Function>
```

A function contains a FunctionId attribute and several <Parameter> elements. The <Parameter> element consists of a “Value” and a “DataType” attribute. Functions are the main functional elements of DCRDL. They are classified into three categories:

- Comparison Function

These functions are used in the condition section. The specified environment parameter (e.g., temperature) and the given value in <Parameter> element are compared by comparison functions to evaluate a condition is true or not. For example, the “*float-larger-than*” function can compare the temperature of target space with the given value.

- Information Obtaining Function

Information obtaining functions are used in both conditions and actions. These functions fetch the information from the environment.

- Action Function

Action functions describe the device control services such as turn on/off lights and

change the set-point of an air conditioner.

3. Code Examples

We have written three policies named “Peak Consumption Control”, “Step tariffs” and “Discomfort Index Control” using DCRDL and showed them in appendix 3.

4. Implementation Conventions

4.1 Running Mode

We define two running modes for an agent in the Policy Agent System:

Polling: the policy agent will run periodically, the polling period is specified by period attribute of the policy element. Policy agent will check inside rules one by one. If context part of the rule is evaluated to be true, corresponding actions should be performed.

Event-driven: When a condition of the ConditionSet of a rule is changed, the rule should be re-evaluated by the policy agent. The event-driven mode needs the eventing-support from the lower building data platform.

4.2 Interpreter

The interpreter translates policies into policy agents (i.e., threads in procedural languages). The *period* attribute in the policy element indicates the polling cycle of this agent if the *mode* attribute is set to “polling”. The policy agent checks all the rules defined within it in every cycle or upon receiving an event. When the ConditionSet of a rule is evaluated to be true, the actions defined in the ActionSet are executed. A key component of the interpreter is the Smart Building API Modeling component. It is responsible for translating the descriptive expressions into a piece of executable code which invokes smart building APIs to get information and control devices in the building.

4.3 Rule Optimization

A critical issue called rules conflicting resolution is not discussed in this specification. Many algorithms have proposed to address it. A rule optimizer should be implemented

for detecting conflicts and duplications. We will deal with this problem in future versions of this specification.

5. Data types

- http://www.w3.org/2001/XMLSchema#string
- http://www.w3.org/2001/XMLSchema#boolean
- http://www.w3.org/2001/XMLSchema#integer
- http://www.w3.org/2001/XMLSchema#float
- http://www.w3.org/2001/XMLSchema#time
- http://www.w3.org/2001/XMLSchema#date
- http://www.w3.org/2001/XMLSchema#dateTime
- org:sakamura-lab:names:dcrdl:1.0:data-type:ucode
- org:sakamura-lab:names:dcrdl:1.0:data-type:ucode-list

6. Functions

All the functions share a prefix: “org:sakamura-lab:names:dcrdl:1.0:function”. The functions currently defined in DCRDL are listed as follows:

6.1 Comparison Function

Comparison functions only have two parameters. It returns a boolean value which represents the result of logical comparison of the two parameters.

- org:sakamura-lab:names:dcrdl:1.0:function:integer-equals
- org:sakamura-lab:names:dcrdl:1.0:function:integer-greater-than
- org:sakamura-lab:names:dcrdl:1.0:function:integer-greater-than-or-equal
- org:sakamura-lab:names:dcrdl:1.0:function:integer-less-than
- org:sakamura-lab:names:dcrdl:1.0:function:integer-less-than-or-equal
- org:sakamura-lab:names:dcrdl:1.0:function:float-equals

- org:sakamura-lab:names:dcrdl:1.0:function:float-greater-than
- org:sakamura-lab:names:dcrdl:1.0:function:float-greater-than-or-equal
- org:sakamura-lab:names:dcrdl:1.0:function:float-less-than
- org:sakamura-lab:names:dcrdl:1.0:function:float-less-than-or-equal
- org:sakamura-lab:names:dcrdl:1.0:function:string-equals
- org:sakamura-lab:names:dcrdl:1.0:function:string-greater-than
- org:sakamura-lab:names:dcrdl:1.0:function:string-greater-than-or-equal
- org:sakamura-lab:names:dcrdl:1.0:function:string-less-than
- org:sakamura-lab:names:dcrdl:1.0:function:string-less-than-or-equal
- org:sakamura-lab:names:dcrdl:1.0:function:time-equals
- org:sakamura-lab:names:dcrdl:1.0:function:time-greater-than
- org:sakamura-lab:names:dcrdl:1.0:function:time-greater-than-or-equal
- org:sakamura-lab:names:dcrdl:1.0:function:time-less-than
- org:sakamura-lab:names:dcrdl:1.0:function:time-less-than-or-equal

6.2 Information Obtaining Function

- org:sakamura-lab:names:dcrdl:1.0:function:get-space-discomfort-index

  -Parameter 1

  Description: ucode of the target space.

  Data type: org:sakamura-lab:names:dcrdl:1.0:data-type:ucode

  -Return

  Description: the discomfort index value of the target space.

  Data type: http://www.w3.org/2001/XMLSchema#float

- org:sakamura-lab:names:dcrdl:1.0:function:get-space-temperature

  -Parameter 1

  Description: ucode of the target space.

  Data type: org:sakamura-lab:names:dcrdl:1.0:data-type:ucode

-Return

Description: the temperature of the target space

Data type: http://www.w3.org/2001/XMLSchema#float

- org:sakamura-lab:names:dcrdl:1.0:function:get-space-occupancy

-Parameter 1

Description: ucode of the target space.

Data type: org:sakamura-lab:names:dcrdl:1.0:data-type:ucode

-Return

Description: Boolean value indicates if there is a person in the target space or not.

Data type: http://www.w3.org/2001/XMLSchema#boolean

- org:sakamura-lab:names:dcrdl:1.0:function:get-space-humidity

-Parameter 1

Description: ucode of the target space.

Data type: org:sakamura-lab:names:dcrdl:1.0:data-type:ucode

-Return

Description: the humidity value of the target space.

Data type: http://www.w3.org/2001/XMLSchema#float

- org:sakamura-lab:names:dcrdl:1.0:function:get-electricity-consumption

-Parameter 1

Description: list of ucode of the target spaces.

Data type: org:sakamura-lab::names:dcrdl:1.0:data-type:ucode-list

-Parameter 2

Description: time interval. Ex.“00:30:00” means the last 30 minutes.

Data type: http://www.w3.org/2001/XMLSchema#string

-Return

Description: the electricity consumption of the rooms in a time interval.

The unit of the return value is “kWh”.

Data type: http://www.w3.org/2001/XMLSchema#float

- org:sakamura-lab:names:dcrdl:1.0:function:get-space-by-lowest-temperature

-Parameter 1:

Description: list of ucodes of spaces. If not given, search the building.

Data type: org:sakamura-lab::names:dcrdl:1.0:data-type:ucode-list

-Return

Description: ucode of the space where the temperature is lowest.

Data type: org:sakamura-lab::names:dcrdl:1.0:data-type:ucode

- org:sakamura-lab:names:dcrdl:1.0:function:get-space-by-highest-temperature

-Parameter 1:

Description: list of ucodes of spaces. If not given, search the building.

Data type: org:sakamura-lab:names:dcrdl:1.0:data-type:ucode-list

-Return

Description: ucode of the space where the temperature is highest.

Data type: org:sakamura-lab:names:dcrdl:1.0:data-type:ucode

- org:sakamura-lab:names:dcrdl:1.0:function:get-space-by-lowest-humidity

-Parameter 1:

Description: list of ucodes of spaces. If not given, search the building.

Data type: org:sakamura-lab:names:dcrdl:1.0:data-type:ucode-list

-Return

Description: ucode of the space where the humidity is lowest.

Data type: org:sakamura-lab:names:dcrdl:1.0:data-type:ucode

- org:sakamura-lab:names:dcrdl:1.0:function:get-space-by-highest-humidity

-Parameter 1:

Description: list of ucodes of spaces. If not given, search the building.

Data type: org:sakamura-lab:names:dcrdl:1.0:data-type:ucode-list

-Return

Description: ucode of the space where the humidity is highest.

Data type: org:sakamura-lab:names:dcrdl:1.0:data-type:ucode

- org:sakamura-lab:names:dcrdl:1.0:function:get-humidifiers-by-lowest-humidity

-Parameter 1:

Description: list of ucodes of spaces. If not given, search the building.

Data type: org:sakamura-lab:names:dcrdl:1.0:data-type:ucode-list

-Return

Description: the string of ucodes for the target devices that split by commas.

Data type: org:sakamura-lab:names:dcrdl:1.0:data-type:ucode-list

- org:sakamura-lab:names:dcrdl:1.0:function:get-humidifiers-by-highest-humidity

-Parameter 1:

Description: list of ucodes of spaces. If not given, search the building.

Data type: org:sakamura-lab:names:dcrdl:1.0:data-type:ucode-list

-Return

Description: the string of ucodes for the target devices that split by commas.

Data type: org:sakamura-lab:names:dcrdl:1.0:data-type:ucode-list

- org:sakamura-lab:names:dcrdl:1.0:function:get-acs-by-lowest-temperature

-Parameter 1:

Description: list of ucodes of spaces. If not given, search the building.

Data type: org:sakamura-lab:names:dcrdl:1.0:data-type:ucode-list

-Return

Description: the string of ucodes for the target devices that split by commas.

Data type: org:sakamura-lab:names:dcrdl:1.0:data-type:ucode-list

- org:sakamura-lab:names:dcrdl:1.0:function:get-acs-by-highest-temperature

-Parameter 1:

Description: list of ucodes of spaces. If not given, search the building.

Data type: org:sakamura-lab:names:dcrdl:1.0:data-type:ucode-list

-Return

Description: the string of ucodes for the target devices that split by commas.

Data type: org:sakamura-lab:names:dcrdl:1.0:data-type:ucode-list

- org:sakamura-lab:names:dcrdl:1.0:function:get-humidifiers-by-space

-Parameter 1

Description: ucode of the target space.

Data type: org:sakamura-lab:names:dcrdl:1.0:data-type:ucode

-Return

Description: ucode list of the target devices in the string format that split by commas.

Data type: org:sakamura-lab:names:dcrdl:1.0:data-type:ucode-list

- org:sakamura-lab:names:dcrdl:1.0:function:get-acs-by-space

  -Parameter 1

  Description: ucode of the target space.

  Data type: org:sakamura-lab:names:dcrdl:1.0:data-type:ucode

  -Return

  Description: ucode list of the target devices in the string format that split by commas.

  Data type: org:sakamura-lab:names:dcrdl:1.0:data-type:ucode-list

- org:sakamura-lab:names:dcrdl:1.0:function:get-humidifiers-by-space

  -Parameter 1

  Description: ucode of the target space.

  Data type: org:sakamura-lab:names:dcrdl:1.0:data-type:ucode

  -Return

  Description: ucode list of the target devices in the string format that split by commas.

  Data type: org:sakamura-lab:names:dcrdl:1.0:data-type:ucode-list

- org:sakamura-lab:names:dcrdl:1.0:function:get-lights-by-space

  -Parameter 1

  Description: ucode of the target space.

  Data type: org:sakamura-lab:names:dcrdl:1.0:data-type:ucode

  -Return

  Description: ucode list of the target devices in the string format that split by commas.

  Data type: org:sakamura-lab:names:dcrdl:1.0:data-type:ucode-list

- org:sakamura-lab:names:dcrdl:1.0:function:get-realtime-electricity-price

  -Parameter: None

  -Return

  Description: the real time electricity price of the building.

  Data type: http://www.w3.org/2001/XMLSchema#float.

6.3 Action Function

- org:sakamura-lab:names:dcrdl:1.0:function:control-humidifier-on

-Parameter 1

Description: ucode of the target device.

Data type: org:sakamura-lab:names:dcrdl:1.0:data-type:ucode

-Return

Description: error code of the control action.

Data type: http://www.w3.org/2001/XMLSchema#integer 0: success, else failed.

- org:sakamura-lab:names:dcrdl:1.0:function:control-humidifier-on

-Parameter 1

Description: ucode of the target device.

Data type: org:sakamura-lab:names:dcrdl:1.0:data-type:ucode

-Return

Description: error code of the control action.

Data type: http://www.w3.org/2001/XMLSchema#integer 0: success, else failed.

- org:sakamura-lab:names:dcrdl:1.0:function:control-ac-on

-Parameter 1

Description: ucode of the target device.

Data type: org:sakamura-lab:names:dcrdl:1.0:data-type:ucode

-Return

Description: error code of the control action.

Data type: http://www.w3.org/2001/XMLSchema#integer 0: success, else failed.

- org:sakamura-lab:names:dcrdl:1.0:function:control-ac-on-with-setpoint

-Parameter 1

Description: ucode of the target device.

Data type: org:sakamura-lab:names:dcrdl:1.0:data-type:ucode

-Parameter 2

Description: the set-point.

Data type: http://www.w3.org/2001/XMLSchema#float

-Return

Description: error code of the control action.

Data type: http://www.w3.org/2001/XMLSchema#integer 0: success, else failed.

- org:sakamura-lab:names:dcrdl:1.0:function:control-ac-off

  -Parameter 1

  Description: ucode of the target device.

  Data type: org:sakamura-lab:names:dcrdl:1.0:data-type:ucode

  -Return

  Description: error code of the control action.

  Data type: http://www.w3.org/2001/XMLSchema#integer 0: success, else failed.

- org:sakamura-lab:names:dcrdl:1.0:function:control-light-on

  -Parameter 1

  Description: ucode of the target device.

  Data type: org:sakamura-lab:names:dcrdl:1.0:data-type:ucode

  -Return

  Description: error code of the control action.

  Data type: http://www.w3.org/2001/XMLSchema#integer 0: success, else failed.

- org:sakamura-lab:names:dcrdl:1.0:function:control-light-off

  -Parameter 1

  Description: ucode of the target device.

  Data type: org:sakamura-lab:names:dcrdl:1.0:data-type:ucode

  -Return

  Description: error code of the control action.

  Data type: http://www.w3.org/2001/XMLSchema#integer 0: success, else failed.

- org:sakamura-lab:names:dcrdl:1.0:function:control-ac-setpoint-decrease

  -Parameter 1

  Description: ucode of the target device.

  Data type: org:sakamura-lab:names:dcrdl:1.0:data-type:ucode

  -Parameter 2

  Description: decrement.

  Data type: http://www.w3.org/2001/XMLSchema#float

  -Return

  Description: error code of the control action.

Data type: http://www.w3.org/2001/XMLSchema#integer 0: success, else failed.

- org:sakamura-lab:names:dcrdl:1.0:function:control-ac-setpoint-increase

  -Parameter 1

  Description: ucode of the target device.

  Data type: org:sakamura-lab:names:dcrdl:1.0:data-type:ucode

  -Parameter 2

  Description: increment.

  Data type: http://www.w3.org/2001/XMLSchema#float

  -Return

  Description: error code of the control action.

  Data type: http://www.w3.org/2001/XMLSchema#integer 0: success, else failed.

- org:sakamura-lab:names:dcrdl:1.0:function:find-devices-with-lower-operation-priority

  -Parameter 1:

  Description: list of ucodes of target spaces. If empty string is given, search the building.

  Data type: org:sakamura-lab::names:dcrdl:1.0:data-type:ucode-list

  -Parameter 2:

  Description: integer value indicates how many devices should be returned.

  Data type: http://www.w3.org/2001/XMLSchema#integer

  -Return

  Description: ucode list of devices whose operation priorities are highest.

  Data type: org:sakamura-lab::names:dcrdl:1.0:data-type:ucode-list

- org:sakamura-lab:names:dcrdl:1.0:function:turn-on-off-device

  -Parameter 1:

  Description: ucode of the target device.

  Data type: org:sakamura-lab::names:dcrdl:1.0:data-type:ucode

  -Parameter 2:

  Description: on/off.

  Data type: http://www.w3.org/2001/XMLSchema#string

  -Return

  Description: error code of the control action.

Data type: http://www.w3.org/2001/XMLSchema#integer 0: success, else failed.

- org:sakamura-lab:names:dcrdl:1.0:function:turn-down-device

  -Parameter 1:

  Description: ucode of the target device.

  Data type: org:sakamura-lab::names:dcrdl:1.0:data-type:ucode

  -Parameter 2:

  Description: the decrement of control target (i.e., set-point of air conditioner).

  Data type: http://www.w3.org/2001/XMLSchema#integer

  -Return

  Description: error code of the control action.

  Data type: http://www.w3.org/2001/XMLSchema#integer 0: success, else failed.

- org:sakamura-lab:names:dcrdl:1.0:function:turn-up-device

  -Parameter 1:

  Description: ucode of the target device.

  Data type: org:sakamura-lab::names:dcrdl:1.0:data-type:ucode

  -Parameter 2:

  Description: the increment of control target (i.e., set-point of air conditioner).

  Data type: http://www.w3.org/2001/XMLSchema#integer

  -Return

  Description: error code of the control action.

  Data type: http://www.w3.org/2001/XMLSchema#integer 0: success, else failed.