國立台灣大學電機資訊學院資訊工程學研究所

碩士論文

Department of Computer Science and Information Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Master Thesis

物聯網多元應用之拍賣式衝突解決方案

Auction-Based Actuator Conflict Resolution in IoT Multi-Applications

梁哲瑋

Che-Wei Liang

指導教授： 許永真 博士

Advisor: Jane Yung-jen Hsu, Ph.D.

中華民國 103 年 7 月

July, 2014

Acknowledgments

首先要感謝指導教授許永眞老師，這兩年來教導我做研究的態度，除此之外，從老師身上更是學到許多做人處世的道理，這些都是我這兩年碩士生活中十分重要的學習。

感謝 WuKong 的三位指導老師，林桂傑老師、施吉昇老師、王佑中博士，很感謝林桂傑老師指導我論文的不足之處並且帶領我修改論文，感謝施吉昇老師與王佑中博士，每次與兩位老師的討論總是能得到許多新的想法。

感謝 WuKong 的成員們，從 Niels 跟博倫兩位強者學到許多，兩位總是能夠回答並幫助我困擾許久的問題，跟政勳講講鬼話互相確認進度，還有士元，一起在 WuKong 奮鬥的戰友，從一開始的台南 TAAI 一直到遠征美國 CMU ，一起面臨做研究的低潮以及互相扶持，最後能夠一起畢業。

感謝 iAgent 的朋友們以及 Dynamic 的組員們，自均跟鈞奕兩位聽我說瘋話以及給我許多建議，季恩幫助我釐清研究問題，Janet 指出許多問題及建議，iAgent 就像個大家庭一樣，大家互相幫忙互相成長。

感謝彰友會的朋友們，偶爾與你們的小聚總是充滿歡樂。

最後感謝我的家人以及女朋友怡雯，總是在我身後默默支持我，讓我有勇氣繼續接受挑戰並走完碩士這條路。

Abstract

Internet of Things(IoT) technology enables billions of devices to connect to the Internet, including wearable devices, home appliances, ambient devices and so on. IoT application developers constantly create new services and applications to control the actuators to make our lives easier. Since many applications may exist simultaneously in a given environment, it is likely that some applications want to use the same actuator at the same time, which creates actuator conflicts. How to solve conflicts is important in IoT applications. The conflict resolution should be efficient and optimal among the users. We propose an auction-based mechanism to coordinate applications and resolve actuator conflicts. The simulation results show that our methods are efficient and can achieve good performances.

Keywords: Internet of Things, Conflict Resolution, Auction

摘要

物聯網(Internet of Things)將我們生活中數以萬計的裝置都連上網路。隨著物聯網科技的進步，物聯網的應用也持續的成長，並且適時地提供我們服務，讓我們的生活過得更輕鬆。然而，當這些不同應用同時存在在我們的身活環境時，這些不同的應用將可能同時使用同一個控制器(Actuator)導致衝突。如何有效地解決衝突是一項重要的問題，必須快速且無痕的將衝突化解，並且能夠符合使用者的喜好。我們提出拍賣式的衝突解決方案來化解衝突，模擬的實驗結果證實我們所提出的拍賣式解決方案是有效的且解決結果能夠達到滿足多數應用程式的喜好。

關鍵字：物聯網、拍賣、衝突解決

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

The Internet of Things (IoT) is an emerging technology in recent years [6]. It enables objects surrounding us with the ability to communicate through Internet. The idea is to make the interconnected objects harvest information from the environment and use Internet to provide services for information transfer, analytics, applications.

There are many application domains that IoT technology can provide very useful services, such as healthcare, transportation, smart city and smart home. In healthcare, sensors and wearable devices are used to detect the health condition of patients and older people. These information are sent to hospitals and family members so they can track the health condition and react immediately when an accident happens. In transportation applications, when people are driving, they become aware of the road condition and an automobile can give a shortest route that avoids traffic jams and car accidents. Moreover, the sensors around the car and the communication between cars help to warn the driver when there might be a reckless driver approaching. In smart home environment, sensors are used to detect user's context. These context information is further analyzed to predict user intentions and control actuators accordingly to provide a comfortable environment.

1.2 Motivation

In a smart home environment, family members may deploy different applications to control their home appliances for different purposes. For example, one wants to monitor the home security, while another adjusts the environment for entertainment. These different applications utilize shared sensors and actuators to provide their functionalities to users. An application keeps monitoring the environment by the sensor data as inputs to see how the application can help a user by controlling the actuators as output. There may be an actuator conflict when several applications in the environment want to control the same actuator with different actions for their own purposes.

For example, suppose a family of two parents and a 3-year-old child live in a smart home with sensors and actuators. The house has an application *House-elf* to provide them a comfortable entertainment. Another application *Guardian Angel* is to help them take care of the child. These two applications are developed separately by developers.

When these two applications are deployed in the same house, there might be some conflicts. When parents are watching TV in the living room, *House-elf* would turn on the volume to normal and turn on lights to make a comfortable environment to watch TV. At the same time, the child is too tired and falls asleep on the sofa. *Guardian Angel* wants to turn off the lights and TV to let the child sleep well. If the two applications compete, and one takes control of lights and the other on TV, both adults and child will be unhappy. A better solution is to let these two applications coordinate to find an acceptable setting of volume and luminance.

Resolving resource access conflict is an important issue [17, 4, 18]. We are interested in providing a good and user-friendly solution to the problem in the WuKong project. WuKong [15, 14, 8] builds a service oriented IoT framework. It eases the tedious work to build an IoT application. It provides a flow-based programming tool to create applications and encapsulates the devices setting via service discovery and mapping.

1.3 Objectives

In this thesis, we give an auction-based mechanism to coordinate applications to resolve actuator conflicts. Applications are treated as bidders, and IoT devices are the goods for bidding. Each application can set its preference on each actuator by submitting their bids to an auctioneer. The auctioneer should take each application's bid into account and make a decision on what action an actuator should perform. The outcome of the resolution should maximize the total utility received by all applications.

Auction is a useful technology for resolving resource access conflicts [1]. Many types of auction protocols have been designed. Two unique properties in a smart home application are that actuators can have different settings and several actuators must be used together to deliver a service. Moreover, an actuator may be set to a specific action that can be shared and used by multiple applications. For example, multiple users may want to control the room temperature from their own applications at the same time. Rather than just allowing one application to control the AC setting, it may be possible for several to share the AC if they can agree on the specific AC power level (weak, medium, or strong). Moreover, a user may need to have the right AC setting and also the room light in order to work on his chemistry project. We therefore should design our auction protocol for shareable resources to also meet the bundle requirement.

1.4 Thesis Organization

For the rest of the thesis, we first review related work in Chap. 2. We formally define the resource access requirement and the auction problem in our system in Chap. 3. Chapter 4 presents our auction protocols for smart home IoT. The implementation design in the WuKong framework is shown in Chap. 5. We then present the simulation results in Chap. 6 to compare our protocols with other auction protocols in terms of resource utilities and execution time. The paper is concluded in Chap. 7.

Chapter 2

Related Work

In this chapter, we first show existing work that resolve conflicts in home automation systems. Then we show some multi-agent systems that utilize auction to coordinate agents in different domains.

2.1 Conflict Resolution in Home Automation

Some researchers have studied in resolving conflict in home automation. A common way to resolve the conflict is based on users' preferences. The resolution should take the users' preferences into account and decide what to do. The action to resolve the conflicts are taken according to the weighted users' preferences. Different approaches have been studied on how to calculate the weights of the users. Some work chooses the action of the first priority while some considers all weighted preferences of users.

Shin et al.[16] propose a history-based approach to resolve conflicts. The system keeps the historical data and utilizes Bayesian theory to decide the priority of users. User can use a tangible remote controller to send the feedback so the change of user preference is captured.

Park et al. [9] use a semantic ontology to represent the relationship between the environment context and the applications. The links between the context and the applications represent how the action will affect the context attribute, e.g., turn on the light will increase the lightness. The system calculates the weight by predefined users preference and decide which application get executed accordingly.

Thyagaraju et al.[19] resolves the conflicts when multiple users want to watch different TV channels. Users are required to provide their preferences on each channel and their roles in the family to construct the user profile. The system uses these user profiles to calculate a weighted score for each channel when conflict arises. The channel with the highest score is selected as the output.

Petrushevski et al.[10] resolves conflicts in a lighting control system among multiple users. Several lighting sources are located in different locations of a room sharing by users. When users have different preferences on the light sources, the system needs to decide how should the light sources perform. The system calculates the weighted average of preferred outputs of a light source. The distance between the user and the light source is considered as the weight of the user to control the light sources. The closer a user is to the light, the higher weight the user has.

2.2 Coordination in Multi-Agent Systems

Auction has been used to coordinate multi-agent systems in different domains, including task allocation, exploration task.

Koenig et al. [7] use the auction mechanism on the robot exploration problem. There are many targets separately in the environment that need to be visit at least once by robot agents. The robot agents are also separately around the environment. The problem is to efficiently allocate the targets for each robot that minimize the total visited distances. They use sequential auction mechanism to coordinate agents.

Capra et al. [2] focus on the preference conflict on mobile application between several users. The users can use the same mobile application to communicate with each other. However, the application provides different communicate protocol. Each protocol has different quality of services and requirement. Each user has different preferences on each protocol. They use the auction mechanism to coordinate mobile agents to decide the protocol.

Chapter 3

Actuator Conflicts in IoT Applications

In this chapter, we first give a formal definition of our problem definition. Then we give our proposed methods in the next section. Finally, we give a example to see how our methods solve the problem.

3.1 IoT Application

In the future IoT environment, there will be lots of sensors monitoring the environment and actuators controlling the environment. An IoT application takes sensor data as input and produces operations to control the actuators. Sensors provide environment data and the application evaluates the environment state by its utility function. The utility function reports a score on the desirable degree of the current environment. An application should always try to maximize its utility to satisfy users. When the environment is not what the application wants, the utility value will be less. Thus, the application will send out an operation for actuators to change the environment to maximize its utility.

When there exist multiple applications in the environment, these applications may send different operations to control the actuator for their different goals. Therefore, an actuator may receive more than one commands to perform. To serve acceptable services to users, the actuator conflict resolution component is employed to resolve actuator conflicts. It takes all applications into account and make a final decision on what the operation should be. The whole process of IoT applications is shown in Fig. 3.1.

3.2 Resolution Timing

The conflict happens when the two applications want to perform two different actions in an actuator. An action is to keep a state in an environment. For example, the action turn on, is to keep the environment light. On the other hand, turn off, is to keep the environment dark. The application sends the action only when it wants to change the state. Therefore, we can detect the conflict when two applications want two different states.

When the environment changes, the application evaluates the utility of the environment. The application might send a new action to the actuator. The conflict might happen again and the resolution process starts again to resolve the process.

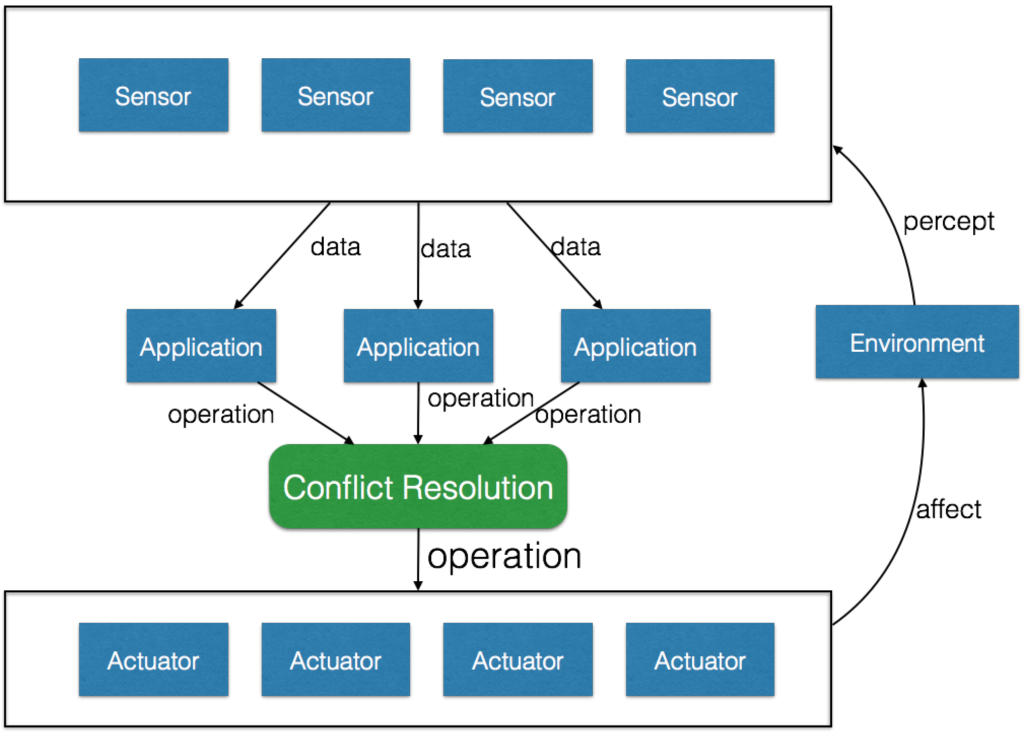


Figure 3.1: IoT applications flow diagram with conflict resolution

3.3 Problem Formulation

An actuator conflict resolution problem is defined as a tuple $(s, \mathcal{M}, (\mathcal{A}_i)_{i\in\mathcal{M}}, \delta, \mathcal{N}, (u_i)_{i\in\mathcal{N}}, (x_i)_{i\in\mathcal{N}})$

- s denotes the environmental state. The environmental state is represented by a vector of context attributes values. $s =< \beta_1, \beta_2, \ldots, \beta_t >$. Each element in s represents the value of a context attribute. $\mathcal{S}$ denotes the set of all states.
- $\mathcal{M} = \{1, 2, \ldots, m\}$ denotes a finite set of actuators.
- $\mathcal{A}_i$ denotes the set of actions that actuator i can perform. $\mathcal{A}$ denotes the set join action space or set of all operation, i.e., $\mathcal{A} = \prod_{i\in R} \mathcal{A}_i$. We call $\alpha \in \mathcal{A}$ an *operation*, i.e., $\alpha =< \alpha_1, \alpha_2, \ldots, \alpha_m >$ where $\forall i \in \mathcal{M}, \alpha_i \in \mathcal{A}_i$
- $\delta : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the state transition function that defines the transition between states by performing an operation. For a current state s and the operation α, it outputs the next state s' that results from doing α in state s.
- $\mathcal{N} = \{1, 2, \ldots, n\}$ denotes a finite set of applications.
- $u_i : \mathcal{S} \to \mathbb{R}$ is the utility function for application i. The application uses the utility function to evaluate the utility for each state $s \in \mathcal{S}$. The utility value stands how much the application likes the state. An application should always try to maximize its utility.
- $x_i : \mathcal{S} \to \mathcal{A}$ is the decision function for application i. For any current environmental state s, an application would try to maximize its utility by finding an operation to control the environment, i.e., $x_i(s) = \text{argmax}_{\alpha\in\mathcal{A}} u_i(\delta(s, \alpha))$

- Actuator conflict: the actuator conflict happens when two applications have different operations trying to control the same actuator with different actions in the same state, i.e., $\exists i, j \in \mathcal{N}, x_i(s) \neq x_j(s)$. In particular, $\exists t \in \mathcal{M}, \alpha_{t,i} \neq \alpha_{t,j}$, the actuator t is in conflict by application i and j.

3.4 Problem Definition

To resolve the conflict, we coordinate the applications to decide on an operation α^* that maximizes the utilities of all applications, which we call the total utility function. The total utility function considers all utilities u_i of applications into account and gives one utility value for all applications.

$$\alpha^* = \operatorname*{argmax}_{\alpha \in \mathcal{A}} U(u_1, u_2, \ldots, u_n, s)$$

where U is the total utility function. How to derive the best total utility function is out of scope for this paper. In multi-objective optimization, a well-known method is called linear scalarization, i.e., the total utility is the sum of utilities of all applications. Therefore, the problem is to find an operation α^* to resolve the actuator conflict that maximizes the sum of utilities of all applications:

$$\alpha^* = \operatorname*{argmax}_{\alpha \in \mathcal{A}} \sum_{i \in \mathcal{N}} u_i(s)$$

To summarize, the problem we aim to solve is an actuator conflict resolution problem. The output is the operation α^*. We make two assumptions. First, the number of actions are discrete and finite. Second, the environmental state transitions are deterministic

Chapter 4

Auction-Based Actuator Conflict Resolution

In this chapter, we first briefly review three resources allocation problems, including multi-agent resource allocation with shareable resources, multi-round single-item auction, and combinatorial auction. We compare the differences between their problems and ours. Then, we adopt some of their ideas and propose our auction algorithms.

4.1 Existing Resource Allocation Problems

4.1.1 Multi-Agent Resource Allocation with Shareable Resources

[1] has studied the resource allocation with shareable resources. They model the multi-agent resource allocation with shareable resources with congestion games. The more agents share a resource, the higher cost these agents will receive. Therefore, the agent should be reluctant to share the resource with others by nature.

There are a few different issues between their work and ours. In their case, agents are less willing to share resources with others. Because if it shares the resource, the cost will be higher. An agent has no incentive to share resources. In our case, although an actuator is a shareable resource, an actuator has different actions which are mutually exclusive. Some actions, however, may unite the applications with the same preference. The applications with the same preference can cooperate together to maximize their utilities. Applications are more willing to share an actuator with other applications that have the same preferences on the actuator.

4.1.2 Combinatorial Auction

Combinatorial auction is a powerful auction mechanism [3, 11]. The application bids on bundles. A bundle is a subset of the actuators. The application bids on the maximum utility that the bundle of actuators can bring to it. The auctioneer receives bids from all applications and determines the operation which maximizes the sum of utilities of all applications. The total utility is optimal because a combinatorial auction considers the synergizes between actuators.

However, there are two issues which make it difficult to apply combinatorial auction to resolve actuator conflicts. First, for applications, it is hard to calculate the bids for all possible bundles because the number of the bundles is extremely large (exponential in the number of actuators and actions). Second, for auctioneer, it is hard to determine the result of the auction in a short time since the winner determine problem is NP-hard. Although some research focuses on finding an approximated result in an efficient way, we focus on the simple auction mechanism that prevents the system from being overly complex.

4.1.3 Multi-Round Single-Item Auction

In multi-round single-item auction, each application bids on each actuator and submit bids to the auctioneer. Every application bids the actuator based on the how many utility the actuator can bring to the application. Only one actuator is sold in a round. The application with the highest bid wins the actuator and the actuator performs the action from the winner application. The auction continues the next round for the remaining actuators until all actuators are sold. Applications can change their bids between rounds. A winner application can control the actuator to affect the environment. As the environment changes, it affects applications and applications might change their minds to ask the remaining actuators to perform different action. When applications change their minds, they can also change their bids in the next round.

Multi-round auction has been applied in other domain and has good results [7]. However, the mechanism can not be directly applied. The reason is that the auctioneer can't correctly optimize the total utility due to the lack of information. The bids only provide the information about the preference to the actuator, but not what action the application would like to perform. Each actuator has several actions, and each action has different effects on the environment. Therefore, the applications has different preferences on each action. We should investigate the preferences on actions to maximize the utilities of all applications.

4.2 Proposed Auctions with Actions and Rebidding

Combinatorial auction synergizes between goods but suffering exponential time complexity while multi-round auction has better time complexity and also synergizes between actuators. Multi-Agent Resource Allocation with Shareable Resources synergizes between bidders. An action, which is shareable among applications, but exclusive in an actuator, makes the applications with similar preferences to work together to maximize their utilities.

We propose two auction algorithms considering the above issues. The first one is *multi-round auction considering actions*. The other is *multi-round auction considering actions and rebidding*. The latter one is to consider rebid on actuators. The rebidding makes it possible for applications to change previous results. As the auction process goes, not all applications can always get what they want. They may win in the first few rounds. However, they may lose some actuators and the environment may change to what they don't like. In this situation, application might regret their previous decisions and want to change the actions. Rebidding provides the opportunity for the applications to change their previous bids.

In our study, we assume that each application doesn't know other's utility function. Each application bids truthfully based on their utility function.

4.2.1 Multi-Round Single-Item Auction considering Actions

Multi-round auction synergizes between actuators by letting applications bid differently in the next round. When an action is determined in one round, applications take the fact and recalculate the operation to control the remaining actuators to maximize their utility. Therefore, the new operation will reflect on the new bids.

The auction works as follows: Each actuator is unallocated at first. Every application bids on each action of each unallocated actuator. The application bids the action based on evaluation of next state resulting from the action. The bids are summed up for each action. The actuator whose action is the highest bid is allocated. Then, each application bids the remaining unallocated actuators and repeat the process until all actuators are allocated. These actions are the final decision as the result. The algorithm is shown in Algorithm 1.

Algorithm 1: Multi-round auction considering actions

```
Input: A set of application $\mathcal{N}$
Input: A set of actuators $\mathcal{M}$
Input: Current environmental state $s$
Output: An operation $\alpha^*$
$\alpha^* \leftarrow \mathrm{Array}()$
foreach $i \in \mathcal{M}$ do
    foreach $j \in \mathcal{A}_i$ do
        $\mathrm{bids}(i, j) \leftarrow 0$
    end
end
while $\mathcal{M} \neq \emptyset$ do
    foreach $i \in \mathcal{M}$ do
        foreach $j \in \mathcal{A}_i$ do
            $s' \leftarrow \delta(s, j)$
            foreach $k \in \mathcal{N}$ do
                $\mathrm{bids}(i, j) \leftarrow \mathrm{bids}(i, j) + u_k(s')$
            end
        end
    end
    $i^*, j^* \leftarrow \mathrm{argmax}_{i \in \mathcal{M}, j \in \mathcal{A}_i} bids(i, j)$
    $\alpha^*[i^*] \leftarrow j^*$
    $\mathcal{M} \leftarrow \mathcal{M} \setminus \{i^*\}$
    $s \leftarrow \delta(s, j^*)$
end
return $\alpha^*$
```

4.2.2 Multi-Round Single-Item Auction considering Actions and Rebidding

This auction is like the previous one. The main difference is that actuators will keep in the list after sold in the previous rounds. When applications regret their previous bids based on the latest bidding result, they can bid on sold actuators again to indicate their preference changes. For some applications that require atomic operation, it is important for them to withdraw the operation instead of executing partial operation. Rebidding provides them a mechanism to change their bids when their atomic operations can not be executed. The auction continues until the total utility can not be further maximized. The result is the final decision of the operation. The algorithm is shown in Algorithm 2.

Algorithm 2: Multi-round auction considering actions and rebidding

```
Input: A set of applications $\mathcal{N}$
Input: A set of actuators $\mathcal{M}$
Input: Current environmental state $s$
Output: An operation $\alpha^*$
$\alpha^* \leftarrow$ Array()
foreach $i \in \mathcal{M}$ do
    foreach $j \in \mathcal{A}_i$ do
        bids$(i, j) \leftarrow 0$
    end
end
while True do
    foreach $i \in \mathcal{M}$ do
        foreach $j \in \mathcal{A}_i$ do
            $s' \leftarrow \delta(s, j)$
            foreach $k \in \mathcal{N}$ do
                bids$(i, j) \leftarrow$ bids$(i, j) + u_k(s')$
            end
        end
    end
    $i^*, j^* \leftarrow \mathrm{argmax}_{i \in \mathcal{M}, j \in \mathcal{A}_i} bids(i, j)$
    if $\alpha^*[i^*] = j^*$ then
        break
    end
    $\alpha^*[i^*] \leftarrow j^*$
    $s \leftarrow \delta(s, j^*)$
end
return $\alpha^*$
```

Chapter 5

Implementation Design

In this chapter, we show the implementation design of conflict resolution in the WuKong framework. We will first introduce the WuKong framework and original development flow. Then we will introduce how to support multiple applications and conflict resolution in WuKong framework. Then the detail of the conflict resolution component is presented.

5.1 WuKong Framework

5.1.1 Goal

WuKong aims to provide a simple application development and deployment flow for developers and users to easily create IoT applications. The framework should intelligently perform development details, including sensor identification, device configuration, service deployment, user personalization, network management, and system

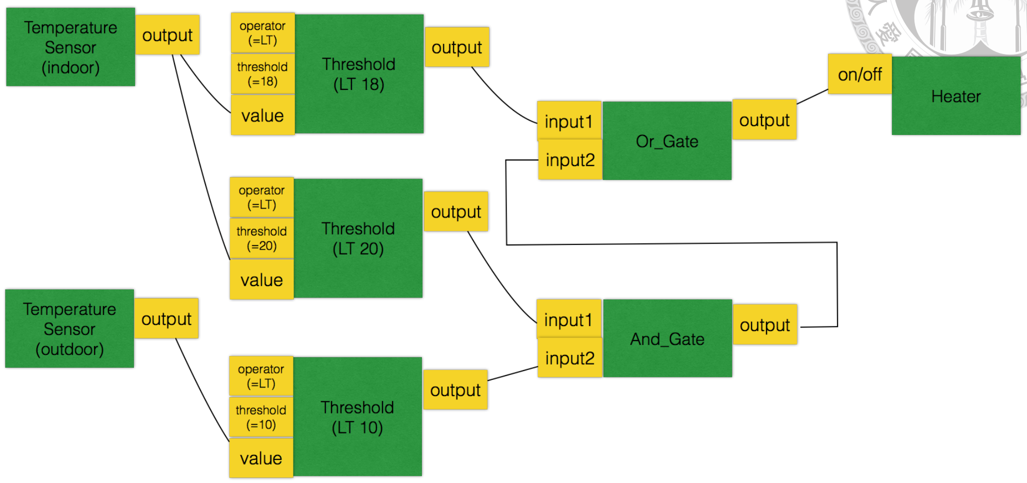


Figure 5.1: A simple application by Flow-Based Programming (FBP)

sustainment. Also, the system can adapt to environment changes. The framework reduces the cost and human efforts to deploy and manage applications and automatically perform the optimization and proactive support for users.

5.1.2 Flow-Based Programming

IoT applications are distributed by nature. An application consists of several components. IoT applications are usually event-driven. The events are detected and the data are sent to the next component. Flow-based programming (FBP) focuses on the information flow between components instead of focusing on data processing in typical applications. Therefore, FBP is good for developing IoT applications.

Figure 5.1 shows an example using FBP to program an application. The application is to turn on the heater if the temperature is too cold. There are two conditions that the heater will be turned on. Either condition is met will turn on the heater. The first condition is when the indoor temperature is below 18 degrees Celsius. The second condition is when the indoor temperature is below 20 degrees Celsius and the outdoor temperature is below 10 degrees Celsius. The application consists of 8 components. The sensors parts consists of two temperatures components. The actuator parts consists of one heater component. And five computing logic components in the processing parts.

WuKong provides a FBP developing tool for application developers to use. The interface provides components available to compose an application. Application developers only need to focus the abstract data flow. The communication and implementation detail are encapsulated by the component block. If predefined components in the WuKong framework can not fulfill the requirement of developers, developers can create custom components to satisfy specific task. WuKong takes care of the physical implementation details and eases the effort of managing devices.

5.1.3 Compilation Flow

Figure 5.2 shows the flow of WuKong compilation process. The left part is the process of generating components libraries and device installation. The top-right part is the FBP IDE that introduced in the previous section to develop applications. Then, the bottom-right part is the build process of generating application Java bytecode from application drawn by the FBP.

The application.xml file is passed to the mapping compiler (mapper) in the Master. The Master will perform the discovery process to discover the information from physical devices in the environment. The mapper will match the requirement

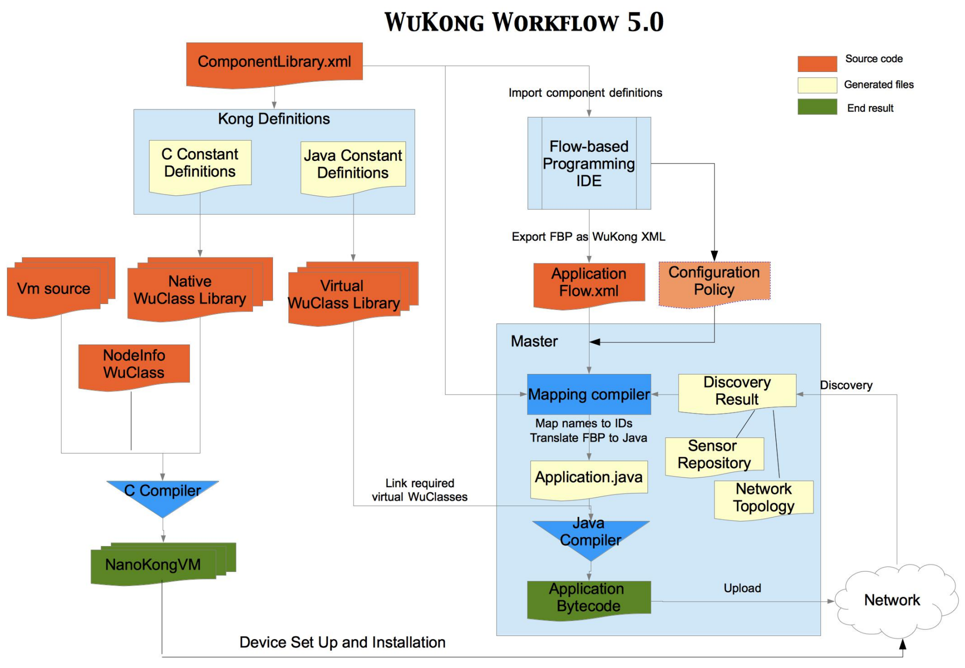


Figure 5.2: WuKong Compilation Process

from the application.xml file with the physical devices. The mapper will generate the Application.java to store the mapping result. It contains the links between components and the physical device of the component. Finally, the application.java will be sent to the Java compiler to generate the Java bytecodes which contains is the application code. Then the Master uploads the application to the devices wirelessly.

5.2 Support Multiple Applications

The original WuKong framework doesn't support running multiple applications. In fig. 5.2, the application bytecode is uploaded to devices. Each device runs only one application. Whenever a new application is uploaded to a device, the old application will be erased.

An naive way to enable running multiple applications is to let devices store multiple application bytecodes and each devices will switch from one application to another. It's like the context switch in common operation system. However, this will increase the overhead of devices. The devices may have very low computing capability. Managing context switch consumes the computing support.

Another way is to keep the device only run one application bytecode and let the Master merge multiple applications into one. This keeps the complexity of devices low and let the Master do the most heavy work which is reasonable since the Master should be a powerful machine.

Figure. 5.3 shows the original mapping process. The mapper takes application.xml which is created by developer using FBP and discovery result as input to map the abstract components to physical devices. The output is the mapping result stored by another xml file named WKDeploy.xml. Then the mapping result is sent to the code generator to generate application bytecode. Finally, the bytecode is uploaded to the physical devices.

The Master has two options to merge applications during the mapping process.

- Merge the application FBP xml files before mapping.
- Merge the WKDeploy.xml files after mapping.

Master choose the first way. Master merges the application FBP files. Once Master merges the FBP files, Master can put the merged FBP file into the mapper, then the mapper will map. The flow is shown in Fig. 5.4

However, we need to further add the conflict resolution component before shared actuators. When merging the application FBPs, Master should try to avoid merging actuators if possible. But the mapping might fail because there might be insufficient physical devices to map the new components. Therefore, if the mapping fails at the first round, Master can merge the unmappable components with other components. And we add an arbitrator component before the actuator to resolve conflict resolution. Then, try the second round mapping with merged components. The flow is shown in Fig. 5.5.

- Stage I: concatenate the xml without any merge and run the mapper.
- Stage II: merge unmappable components and run the mapper again.

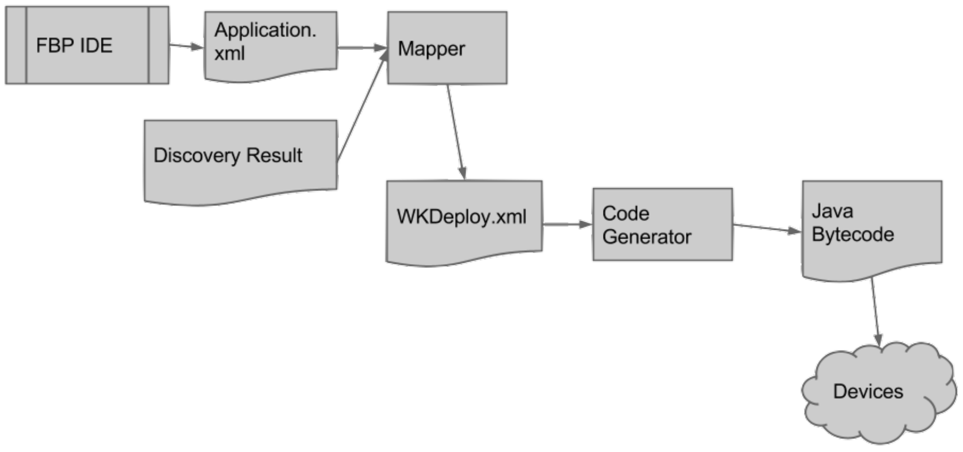


Figure 5.3: Original WuKong Mapping Flow

An example is shown in Fig. 5.6. There is an application running in the environment. The application is to use a touch pad to control the light. The touch pad is mapped to the first physical device and the light is mapped to the second physical device. Now we want to add a new application that uses the light sensor to turn on the light if the room is too dark. Master first merges the application into one FBP file. In first stage, the new application is added at the end of the existing application. These are two isolated connected components and no component has been merged yet. Next Master sends the merge result to the mapper to see if Master can map the merge FBP to the physical devices. Assume there is only one light in the environment now so the mapping is failed because Master requires two light in the FBP. Then, Master goes to the second stage to merge the light component and Master adds an arbitrator before the light component to coordinate applications.

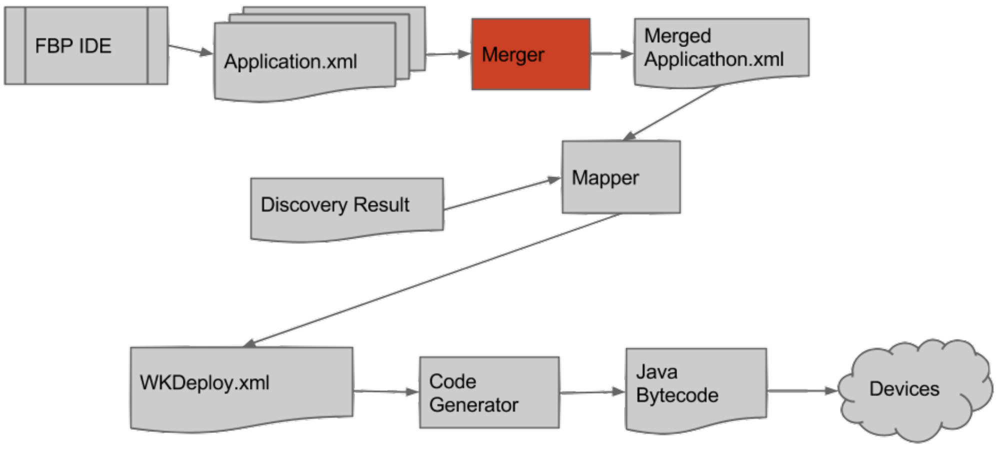


Figure 5.4: WuKong Mapping Flow with Supporting Multi-Application

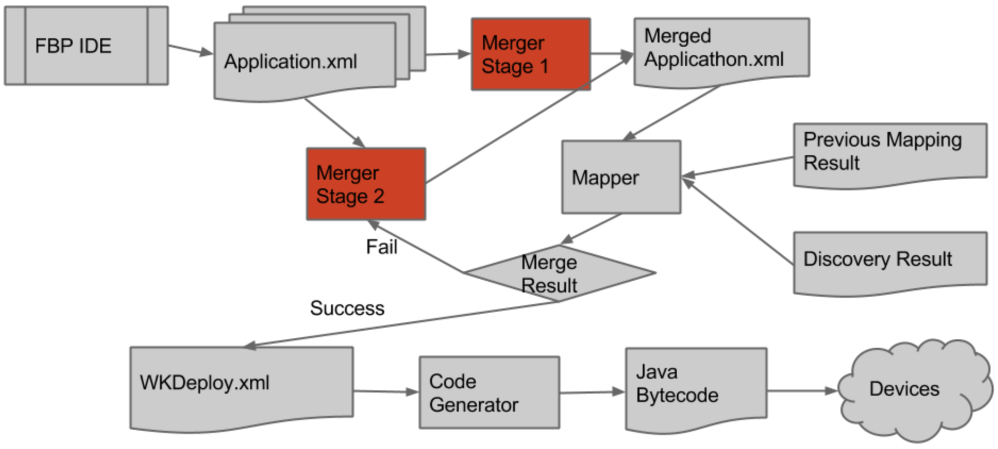
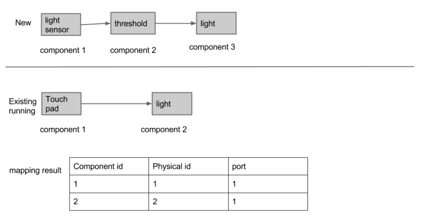
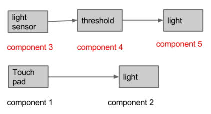


Figure 5.5: WuKong Mapping Flow with Supporting Multi-Application and Conflict Resolution

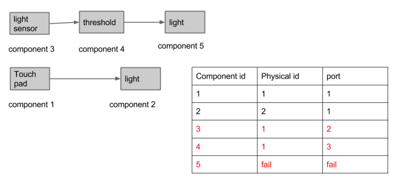


Component id	Physical id	port
1	1	1
2	2	1

(a) Add an new application in a existing application

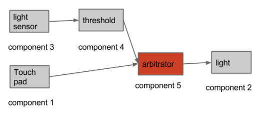


(b) Stage I: Concatenate the existing application and the new application

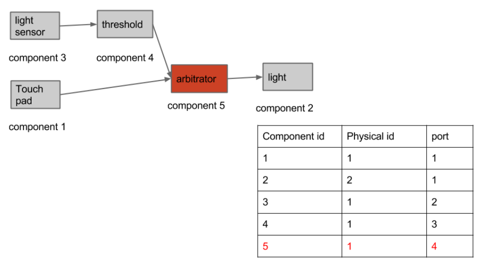


Component id	Physical id	port
1	1	1
2	2	1
3	1	2
4	1	3
5	fail	fail

(c) Map the merged FBP



(d) Merge the failed actuator and add an arbitrator component



Component id	Physical id	port
1	1	1
2	2	1
3	1	2
4	1	3
5	1	4

(e) Map the merged FBP again

Figure 5.6: Example of Merging FBP and Mapping

5.3 Arbitrator Component Implementation

The arbitrator component is used to coordinate applications to control an actuator. The arbitrator takes the actions of applications as input and check if they are the same. If two applications ask the same action, there's no conflict. However, if two applications want to perform different actions, the arbitrator component needs to resolve the conflict. The arbitrator evaluates the utility of each application according to the action of the actuator and chooses the action that can maximize the sum of utilities.

The arbitrator component resolves the conflict when applications want the actuator to perform different actuators. The conflict resolution algorithm is introduced in Chap. 4. The arbitrator component needs several information to resolve the conflict. It needs parameters of the utility function of each applications. The utility function will be introduced in more detail Chap. 6. It needs to know the effect on the environment of the each action of the actuator which we call the actuator profile. The actuator profile is generated and updated by the progression framework in WuKong. It also needs the current environmental state from the sensors. With these information, the arbitrator component can run the auction-based conflict resolution algorithm when applications sends different actions to control the same actuator.

The arbitrator component is a logical computing component. It doesn't evolves physical sensors or actuators. It just computes and implemented in Java in WuKong framework. Using Java, the master can dynamically upload the new implementation easily when the actuator profile or utility function is updated.

Chapter 6

Experiment & Results

We use the simulation to examine our auction algorithms. We first describe how we simulate the environment.

6.1 Simulation Settings

6.1.1 Environmental State

In indoor human comfort index domain, four basic indexes are considered. They are thermal comfort, visual comfort, indoor air quality (IAQ), and acoustic comfort [5, 13]. We use four context attributes to represent an environment state to capture these four comfort indexes. They are temperature, lightness, CO_2 concentration, sound level. $\mathcal{C}$ denotes the set of all considered context attributes. In current case, $\mathcal{C} = \{\text{temperature}, \text{lightness}, \text{CO}_2, \text{sound}\}$, To simplify the experiment, we normalize the values of each context attributes to 0 and 1.

$$s = < \beta_{\text{temperature}}, \beta_{\text{lightness}}, \beta_{\text{CO}_2}, \beta_{\text{sound}} >$$

$$\mathcal{C} = \{\text{temperature}, \text{lightness}, \text{CO}_2, \text{sound}\}$$

$$\forall c \in \mathcal{C}, \quad \beta_c \in [0, 1]$$

6.1.2 Actuator Simulation

An actuator has several actions to perform. Each action has different effects on each context attribute. We use an uniform random generator to generate the effect of each action. These effects are the transition function of the environmental state, i.e., the δ function. Applications can evaluate the utility value of each action based on the transition function, .

6.1.3 Application Simulation

The application uses a utility function to evaluate the preference of the environmental state. It is concerned about some kinds of context attributes, and ignore some context attributes depending on its goal. The application has a target value of context attribute value. The application would want to keep the environmental state to stay in the target value. When the environmental state is closer to the target range, the utility value would be higher. For example, for the application which wants to keep the environment warm, the main context attributes it will consider are thermal and other kinds of context attributes can be ignored. When the environment is warm, the utility will be the high since it achieves the main goal. When the environment is cold or hot, the utility should be low.

Next, we describe how we simulate the utility function. The application calculates a score for each context attribute separately and the utility of the environmental state is the weighted sum of context attributes scores. For context attributes that the application cares, we use a linear scoring function to calculate the score between the environmental state and the target value.

The utility function of application i is denoted by u_i. $u_{i,c}$ denotes the scoring function of context attribute c, and w_c is the weight for context attribute c. The number $a \in [0, 1]$ denotes the parameter that represents the target value, e.g., $c = \text{lightness}, a = 0$ indicates that the application wants to minimize the lightness in the environment. Fig. 6.1 shows different shapes with different target values a.

$$u_i(s) = \sum_{c \in \mathcal{C}} w_c u_{i,c}(\beta_c)$$

$$u_{i,c}(\beta_c) = \begin{cases} 0 & \text{if don't care} \\ -|\beta_c - a| + 1 & \text{else} \end{cases}$$

$$\sum_{c \in \mathcal{C}} w_c = 1, \quad \forall c \in \mathcal{C}, w_c \in [0, 1]$$

In our simulation, for any context attribute, the probability for an application to ignore the particular context attribute is 50%, and the target value, a, is generated by an uniform random generator. The weights for each context attributes, $w_{\text{temperature}}$, $w_{\text{lightness}}$, w_{CO_2}, w_{sound}, are 0.5, 0.3, 0.1, and 0.1 respectively. The thermal comfort gets the highest weight because it influences the human body the most [12].

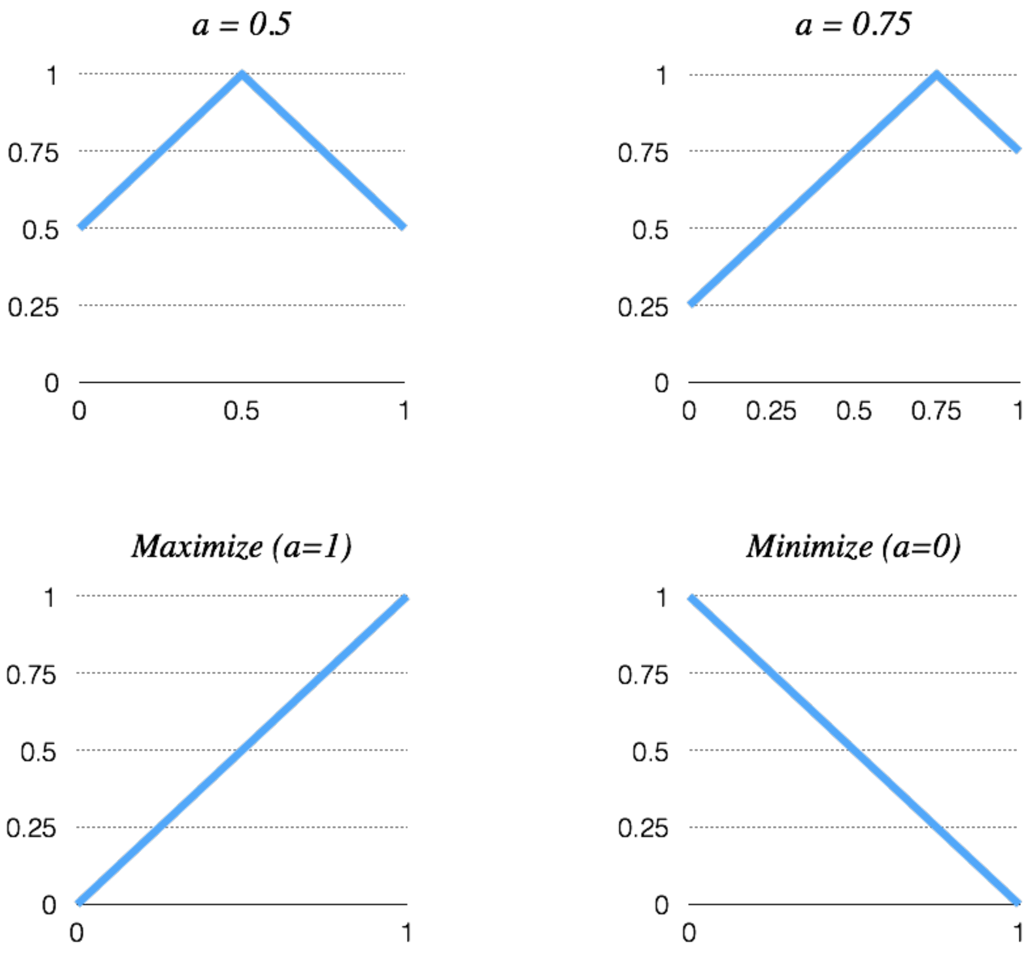


Figure 6.1: Utility functions with different input a

6.2 Evaluation

6.2.1 Compared Methods

We choose two methods to compare with our auction methods. The first one is the random method. It randomly selects an action from each actuator to form the output operation. The second one is the brute force method. Random method is to randomly select an action for each actuator as the output of the resolution. Brute force method iterates all possible operations to find the operation o^* which maximizes the total utilities. It is a global optimal solution.

6.2.2 Evaluation Metrics

For the conflict resolution, it is important to resolve the conflict in a short time while satisfy applications preferences. If the resolution guarantees to find the global optimal to maximize total utility but takes too long, users will be annoyed by the late response. On the other hand, if the resolution resolves conflict very quickly but does not meet applications goals, users will be confused about the resolution result. Either factor is important.

Therefore, we examine our algorithm with these two factors by two metrics. The first one is the time. We calculate time of conflict resolution process. The second one is the utility ratio. The utility ratio is to compare the result from our auction algorithms and the result from the brute force method which gives optimal results.

The formulation is bellowed:

$$u_{\text{ratio}} = \frac{u_{\text{a}}}{u_{\text{b}}}$$

where u_{a} is the utility that auction method achieves, u_{b} is the utility that brute force method achieves.

6.2.3 Evaluation Results

First, we investigate the time complexity of these different algorithms. n denotes the number of applications. m denotes the number of actuators. k denotes the number of actions of each actuator. The time complexity of brute force is exponential. Auction algorithms are polynomial time. The time complexity of multi-round auction is $O(nm^2)$. If we consider action into account, the time complexity will be increased by k. The time complexity table is summarized in 6.1.

We use the tuple (n, m, k) to denote the setting of the simulation. We run 500 times to get the average result for each setting. The first setting (10, 6, 6) contains 10 applications and 6 actuators. Each actuator has 6 actions. The performance of random is about 44%. The performance of multi-round is about 84%. By using auction, the performance increases to 96.7% which is two times higher than random. We can see that considering action indeed synergizes applications and gets a more satisfying result. If we enable rebidding, the performance further increases to almost 97%. Rebidding let applications change their mind and modify the previous result which leads to a better outcome. The performance is shown in Table 6.2.

The second setting (10, 6, 13) contains the same number of applications and

Table 6.1: Time complexity

Algorithm	Time Complexity
Brute Force	$O(nk^m)$
Multi-Round	$O(nm^2)$
Multi-Round with Action	$O(nkm^2)$

Table 6.2: Experiment result (10, 6, 6)

(10, 6, 6)	avg u_{ratio}	std u_{ratio}
Random	43.96%	0.228
Multi-Round	84.20%	0.104
Multi-Round with actions	96.72%	0.028
Multi-Round with actions & rebidding	96.97%	0.027

actuators as the previous setting, 10 and 6 respectively. We increase the number of action to 13. The result is almost the same as the previous setting. Multi-round still achieves high performance. The result in shown in Table 6.3.

Table 6.3: Experiment result (10, 6, 13)

(10, 6, 13)	avg u_{ratio}	std u_{ratio}
Random	44.05%	0.221
Multi-Round	84.80%	0.090
Multi-Round with actions	97.60%	0.021
Multi-Round with actions & rebidding	97.77%	0.019

Chapter 7

Conclusion and Future Work

IoT is a fast growing area. It enables objects around us to communicate with each other, as well as to connect to servers and human users. Many developers are creating new applications to access these objects to provide next generation services to make our lives better. However, multiple applications may need to control smart things at the same time for different purposes. The resource conflict must be resolved to provide adequate services to all users.

In this work, we propose to resolve conflicts by leveraging auction mechanisms. We use the auction mechanism to coordinate applications to share actuators. We examine different auction algorithms and compare the performance with the brute force method and the randomly generated method in both time efficiency and utility ratio aspect. From the result we can see the auction protocols are quite effective to achieve a good system utility. We also give the design on how to implement the resource conflict resolution protocols in the actual IoT middleware that we are developing.

For future work, we can consider the environmental state transition to become non-deterministic. When an actuator performs an action, it might lead to different states with a probability. The problem will become more complicated. Applications need to consider the probability and evaluate the expected utility of performing an action. However, it will also be interesting to model the probability into the problem. Another direction is to implement the design in the real IoT middleware to evaluate the performance of the conflict resolution in real world.

Bibliography

[1] S. Airiau and U. Endriss. Multiagent Resource Allocation with Sharable Items: Simple Protocols and Nash Equilibria. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1 - Volume 1*, AAMAS '10, pages 167–174, Richland, SC, 2010. International Foundation for Autonomous Agents and Multiagent Systems.

[2] L. Capra, W. Emmerich, and I. C. Society. CARISMA : Context-Aware Reflective mIddleware System for Mobile Applications. 29(10):929–944, 2003.

[3] P. Cramton, Y. Shoham, and R. Steinberg. *Combinatorial Auctions.* MIT press, 2006.

[4] F. C. Delicato, P. F. Pires, T. Batista, E. Cavalcante, B. Costa, and T. Barros. Towards an IoT ecosystem. *Proceedings of the First International Workshop on Software Engineering for Systems-of-Systems - SESoS '13*, pages 25–28, 2013.

[5] A. Dounis and C. Caraiscos. Advanced control systems engineering for energy and comfort management in a building environment—A review. *Renewable and Sustainable Energy Reviews*, 13(6-7):1246–1261, Aug. 2009.

[6] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645–1660, Sept. 2013.

[7] S. Koenig, C. Tovey, M. Lagoudakis, V. Markakis, D. Kempe, P. Keskinocak, A. Kleywegt, A. Meyerson, and S. Jain. The Power of Sequential Single-item Auctions for Agent Coordination. In *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 2*, AAAI'06, pages 1625–1629. AAAI Press, 2006.

[8] K.-J. Lin, N. Reijers, Y.-C. Wang, C.-S. Shih, and J. Y. Hsu. Building Smart M2M Applications Using the WuKong Profile Framework. *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*, pages 1175–1180, Aug. 2013.

[9] I. Park, D. Lee, and S. J. Hyun. A Dynamic Context-Conflict Management Scheme for Group-aware Ubiquitous Computing Environments. In *29th Annual International Computer Software and Applications Conference (COMPSAC'05)*, volume 1, pages 359–364. IEEE, 2005.

[10] F. Petrushevski, M. Sipetic, and G. Suter. Conflict management in a personalized, space model based lighting control system. 2013.

[11] S. J. Rassenti, V. L. Smith, and R. L. Bulfin. A Combinatorial Auction Mechanism for Airport Time Slot Allocation. *Bell Journal of Economics*, 13(2):402–417, 1982.

[12] M. I. M. Rawi and A. Al-Anbuky. Wireless sensor networks and human comfort index. *Personal and Ubiquitous Computing*, 17(5):999–1011, May 2012.

[13] R. Reffat and E. Harkness. Environmental comfort criteria: weighting and integration. *Journal of Performance of Constructed Facilities*, (August):104–108, 2001.

[14] N. Reijers, K.-J. Lin, Y.-C. Wang, C.-S. Shih, and J. Y. Hsu. Design of an Intelligent Middleware for Flexible Sensor Configuration in M2M Systems. In *SENSORNETS'13*, pages 41–46, 2013.

[15] N. Reijers, Y.-C. Wang, C.-S. Shih, J. Y. Hsu, and K.-J. Lin. Building intelligent

middleware for large scale CPS systems. In *Service-Oriented Computing and Applications (SOCA), 2011 IEEE International Conference on*, pages 1–4, 2011.

[16] C. Shin, Y. Oh, and W. Woo. History-based Conflict Management for Multi-users and Multi-services. In *Context2005 Workshop (Proc. of the Workshop on Context Modeling and Decision Support)*, 2005.

[17] J. Song, A. Kunz, M. Schmidt, and P. Szczytowski. Connecting and Managing M2M Devices in the Future Internet. *Mobile Networks and Applications*, 19(1):4–17, Nov. 2013.

[18] T. Teixeira, S. Hachem, and N. Georgantas. Service Oriented Middleware for the Internet of Things : (Invited Paper). 257178(257178):220–229, 2013.

[19] G. S. Thyagaraju, S. M. Joshi, U. P. Kulkarni, and S. K. N. a. R. Yardi. Conflict Resolution in Multiuser Context-Aware Environments. *2008 International Conference on Computational Intelligence for Modelling Control & Automation*, pages 332–338, 2008.