A Re-configurable Interaction Model in Distributed IoT Environment

Fangze Qiu
School of Advanced Technology
Xi'an Jiaotong Liverpool University
Suzhou, China
Fangze.Qiu19@student.xjtlu.edu.cn

Huaxiao Huang
School of Advanced Technology
Xi'an Jiaotong Liverpool University
Suzhou, China
Huaxiao.Huang19@student.xjtlu.edu.cn

Yuji Dong*
School of Internet of Things
Xi'an Jiaotong Liverpool University
Suzhou, China
Yuji.Dong02@xjtlu.edu.cn

***Abstract*—Internet of Things (IoT) is a trending paradigm that more and more devices are connected nowadays. However, although the massive IoT devices have been the infrastructure everywhere, the utilization of these devices is still challenging, especially when interacting with multiple devices in a dynamic distributed IoT environment. In this paper, a novel interaction model named UnIEM is proposed to allow Distributed User Interfaces to be dynamically configured on several IoT devices based on user preferences. The proposed model is implemented on the operating system named HarmonyOS to utilize the native distributed processing support. A video migration demo application is developed to validate the proposed model. The eventual work can assist the developers to develop IoT applications with various different devices and interaction strategies with dynamic re-configurations in run-time.**

***Keywords*—Interaction Model, Distributed User Interface, Distributed User Interaction, Internet of Things, Human-Computer Interaction**

I. INTRODUCTION

With the rapid development of a various different Information and communications technologies (ICT), more and more devices are composed together to support some intelligent applications such as smart home [1], smart building [2] and smart city [3]. Since these intelligent applications contain many distributed IoT devices, the user usually needs to interact with several distributed IoT devices in any application. Sometimes the IoT devices may fail to connect the application from device issues or network issues, and new IoT devices can also join the application anytime. The interaction between users and the dynamic distributed IoT environment is complicated and challenging.

The new conditions also bring more possibilities in Distributed User Interfaces (DUI), where the configuration and deployment of user interfaces in IoT environment should be allowed to change in real-time. The interactions between users and IoT applications should be based on a feasible, general environment or paradigm facilitating the operations of the distributed processes and tasks in the interactive IoT systems.

Although some applications can already provide distributed user interfaces among devices such as smartphone, tablets and laptops, the design and development of such user interfaces is still complex and limited. An appropriate interaction model and related framework can provide a good abstraction to assist the developers to develop better interactive IoT systems. However, because the mainstream devices based on Android or MacOS do not have native distributed processing support and the entire technological ecosystems are extremely complicated [4], the previous work in DUI is difficult to be realized to support the developers.

Different from the previous research in DUI, the interaction model in this paper is designed based on the native distributed processing support to generate a flexible, re-configurable interactive environment. The developers can easily utilize the proposed model and framework to develop any high-level applications, where the UI elements can be dynamically configured and deployed on different IoT devices. In the scenarios of smart home, when the user enters in home, the agent in his/her smartphone can detect all the available devices and related interactive elements to support any interaction. Any IoT devices failure or buying a new IoT device will not terminate the user experience, while adapt the interactive environment.

The proposed interaction model is named Unified Interactive Environment Model (UnIEM), where an agent is responsible to generate an interactive environment that all the available devices, UI components, layouts and configuration policies are registered in the environment. The model is implemented based on HarmonyOS to have native distributed processing support. The implementation source-code[1] is open for any developers or researchers to develop further smart IoT applications. An video migration demo app has been developed to indicate how the model can be utilized.

The paper is structured as follows: Section II describe all the related work on the DUI domain. Section III introduces the Unified Interactive Environment Model. The implementation of the proposed model and related case study is explained in Section IV. Finally, Section V is the conclusion and future work.

II. RELATED WORK

There are massive amount of work proposed to solve various problems related to DUI systems. A number of different

supported by XJTLU RDF-21-02-087

[1]https://github.com/CrashDream1219/CD-firstLab

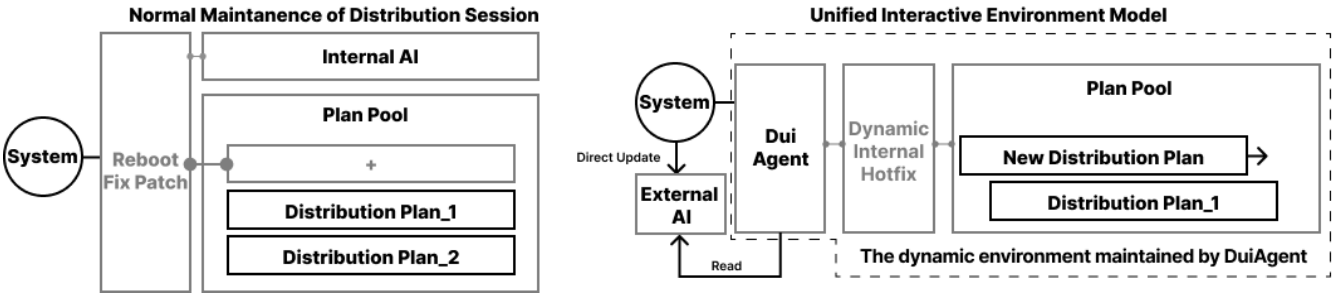

Fig. 1. Comparison of the static and dynamic DUI configurations of the distribution IoT systems

approaches has been developed in order to regulate, unify and optimize the process of distribution.

One of the most featured studies in DUI area is the proposition of traversal model of DUI system. [5] has introduced a conceptual framework trying to emerge the topic of distribute user interface by explicitly proposing the roles in a DUI system and their quantitative correspondence. Despite the lack of operating principle and detailed implementation of such model, the specification of possible important nodes in distribution system served as a general instruction when developing a DUI system. Hence, this kind of general description can be regarded as a guidance when verifying the usability of a DUI system.

Other authors have also contributed notion of other models or independent systems. For instance, in [6], the author illustrates a layered model: the 4-C model who has a close relationship with our model by proposing the notion of habitat of an interactive system. Also, the work potentially provides a hint to the possible software architecture when building up a DUI-based application. However the 4-C model suffers from the inflexibility problems caused by centralized CPU structure and the isolation of the functions within a single device due to the contemporary approaches for human-computer interaction and the missing concept of smart phone (PDA instead).

Alternatively, in the introduction of the Composition of User Interfaces [7], the distribution of UI elements is also defined as an feasible approach for dealing with composition/decomposition of UI. In a dynamic interactive environment, the component-based approach described by [7] can ensure the dynamicity of function distribution in run-time, but it will increase the complexity for data management thus blur the mapping relation of components and functions. Hence, we enhance the mapping relation by defining the concept of ACS. Through clear indication of Ability and UI components, their connection is further stabilized for normal distribution operation.

Several other studies also offer different implementation of DUI system with various levels of abstraction and provided inspiration and guidance to the completeness of the UnIEM model. For example, the discussion of classic distribution concerns raised in [8], which contributes to the design principle of distributing users, tasks, platforms and environments. As well as the optimization formulation brought up in [9], a model-based approaches for DUI in [10] and the concept of generating a natural user interfaces in multi-user collaborative scenarios [11] in IoT area. Additionally, independent DUI systems such as DistriXML [12] and i-Land [13] also show good properties and rationale when dealing with closed system at distribution basis.

With the increasing number of studies on DUI systems, there exists sufficient investigations on the holistic conceptual models or independent interactive system of DUI. Such devotion to a comprehensive context can be of a constructive assistance to the approaches of DUI research, which stresses the investigation of the system's general functionality and distribution architecture. However, with the prevalence of isolation of different operating systems, the common practice of designing app fitting merely one OS and high stickiness of users to a specific brand of product caused by data binding and habits, the modern context of intelligent device usage calls on the re-scaling of DUI researches. Therefore, we aim to produce a modular-type of distribution pattern that sustain a dynamic interactive environment, and promote more discussions on DUI with increasing specialization and flexibility.

III. Unified Interactive Environment Model

Compared with the traditional approaches, the proposed UnIEM provides an abstraction to allow different devices, components and layouts to join the interaction processes without worrying too much technical details. From user perspective, all the available devices and interaction methods are expressed in a device with the support from an agent. The available devices and interaction methods may change depending on the environment change. However, the user experience will not be interrupted.

A sophisticated DUI system usually stresses the interaction method of interactive object within the system, the overall efficiency and the practical implementation of distribution. On the basis of those three core requirements, our framework promotes the flexibility with a dynamic user-centric approach, emphasising the low cost for maintaining and extending DUI system in the dynamic distributed IoT environment.

The Fig. 1 indicate the differences between the traditional interaction model and UnIEM interaction model. The Dui Agent is responsible to generate and maintain a dynamic environment for user interactions. Since the Dui agent is acted as a proxy, all the technical details such as device registration

and UI configuration can be abstracted from the lower layer. The Dui Agent can also load new DUI configuration policies from external sources, which leads to a high flexibility for re-configuration.

In normal distribution session, the insertion of new distribution plans or modification to internal AI deciding the selection of corresponding plans requires deep fix of code structure. However, by implementing a session-bound maintenance agent and adding necessary support, the distribution setting can be configured at run-time thus securing the continuity of the newly generated interactive environment.

Instead of embedding the configuration of distribution pages and tasks into the main operation logic, the manager for distribution is proposed as an external plugin, providing a hub for data transferring and UI components dynamic re-configuration in a way that allows users to interact with the distribution system and get instant feedback rapidly. Fully execution of the environment can act as a lightweight plugin to fulfill users' basic need for distribution for all kind of interactions.

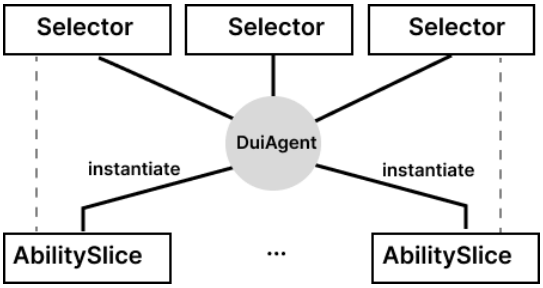

Fig. 2. Relations between DuiAgent, AbilitySlice and Selector

Such a configurable Dui model bypass the process of compiling every DUI elements when initiating program and offers a customizable re-partition experience. Using DuiAgent as a hub, one can implement real-time AI algorithm to dynamically deciding optimized solution for the layout of system elements [9] or use the Selector in DuiAgent to implement manual modification, thus avoid reboot of the whole system normally needed when a completely new distribution plan is ought to be installed. Additionally, by importing the package into the AbilitySlice that requires distribution, only minor adjustments of the view initiation logic are needed to make the page distribution customizable. Fig. 2 indicate the relations between DuiAgent, AbilitySlice and Selector, where the agent can select different configuration policies for different interaction requirements.

To be more explicit, a successful distribution session using this model will go through the following steps in order:

1. Initialization: Including the starting of Ability Slice and the initialization of DuiAgent. In this steps, detailed data of distributable UI elements will be registered into the DuiAgent for later use.

2. Configuration: Once the distribution session is called, the system will make preliminary preparation: The discovery of usable devices, initialization of the UI elements on the pages to be distributed.

3. Alternative Selection: Among all the feasible distribution plans, the system will select the desired one according to the Selector implemented, users' preference or external factors. For instance, an algorithm may be used to promote the optimized distribution plan using the real-time data of DUI elements in DuiAgent.

4. Distribution: Distribution of data, stream, UI and other objects.

IV. Implementation and Case Study

The implementation and deploy of the multi-devices DUI environment fully exploit the inherent structure of HarmonyOS to create multiple modules maintaining the overall operation of the environment. In the following section, we will first introduce the operating logic of HarmonyOS system briefly, describe the design of core character: DuiAgent that is used to sustain and regulate distribution behaviors of the environment along with the unique data structure for managing DUI information in that agent. Then we will explain the general logic of the model and further verify its usability by analyzing a scenario using a demo video migration application implemented within this unified environment.

A. Implementation of the UnIEM Model

*1) **Data structure of UI Components: ACS**:* One of the key issue with distribution is the fundamental consistency of data structure of distributable items. Theoretically speaking, the management of interactive objects requires the existence of a universal description regulation on those objects. According to the concept of Concrete UI (CUI), Abstract UI (AUI) and Concrete Interaction Object (CIOs) [14]. The UI elements in a distribution environment are classified into different types based on its attributes, distribution time and splittability [6]. In the notion of CUI model, we further developed a new definition: **Ability-Component Set(ACS)** to describe the mapping relation of a group of distributable UI items with their specific functions. And define a flat data structure called **DuiComponent** to record the basic information of such item, so as to facilitate the communication and distribution between devices. Similiar to the notion of component-based approach for UI composition and distribution [7], ACS states a clear mapping relation of function and UI element, but instead of the direct connection of elements to function, ACS mechanism will use external list to obtain to mapping relation, so as to reduce the complexity of transferring ACS information.

*2) **Core Distribution manager: DuiAgent**:* With the data structure for storing item information specified, the manager Class: DuiAgent is designed to gather, store, exploit and output the ACS information while initiating and maintaining the dynamic distribution session. The strategy is to keep ACS data within the agent, and allow external access and update. When user demands a distribution of certain function/pages, the corresponding changes made to those UI components will be implemented through DuiAgent, who offers customization

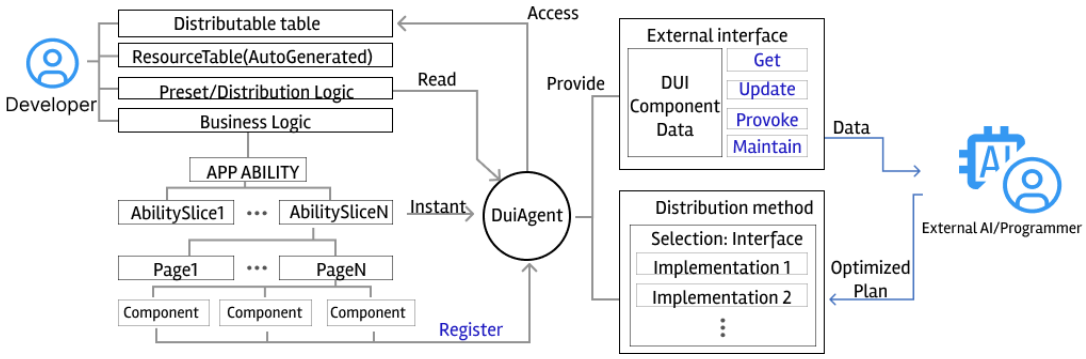

Fig. 3. UnIEM model architecture

to detailed distribution plan through preset **Selector**. As a associated class of DuiAgent, the Selector part of the model mainly focus on the distribution logic and the extensibility. It is designed to allow users to implement different distribution arrangement in order to take advantage of the unique capabilities [10] of devices through customization on software engineering level. When a DuiAgent is activated and instructed to conduct the distribution, it will invoking the customized distribution method in the corresponding selector. At the mean time, the particularity of a DuiAgents' connection with its Ability Slice can also be realized by the selector that DuiAgent called on. In other words, the uniqueness of a Ability Slice with a distribution plan is maintained by the Selector, which directly manage the layout of the UI components in the page, and the DuiAgent is merely just a center for requests and data transmission.

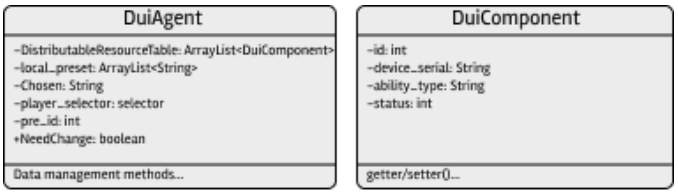

Fig. 4. Data structure for DuiAgent

Through DuiAgent, a distributable UI data set with unified management methods and defining DUI primitives is created, which provides a prerequisite for the repartition of UI components.

*3) **Full operation logic**:* Aiming to keep the system dynamic to implement changes consistently, we first adapt ability slices into a distributable class by instantiating DuiAgent within each AbilitySlice. By registering initial ACS information of the Slice into the agent, the AbilitySlice is ready for later distribution operations. Once the system received the command to start UI migration, the related Selector in the agent bounded with the corresponding Slice will dynamically re-configure the attributes of the ACS based on either user's explicit requirements or forward the given information of the ACS and the user's demand to external decision-making algorithm, and then further restart the page's layout to accomplish the re-shape of UI elements. The entire distribution workflow is indicated in Fig. 5.

With assistance to the basic operation logic, the detailed implementation of the environment in Fig. 3 defines the following terminology:

ResourceTable: An auto generated table of HarmonyOS application(Similar to R. in normal Android environment). Containing the assigning relation of every resource item(page, elements, layout...), device information and a serial number.

Distributable Table: A local table which is a subset of ResourceTable, records the information of distributable UI components of related AbilitySlice in the form of DuiComponent.

Preset/Distribution Logic: Regulation of how the distribution will take place when requested.

External Interfaces: Set of data management methods for external usage.

Register: A basic command of the DuiAgent: record/upload the selected component's information.

GET: A basic command of the DuiAgent: return the information of a target element.

Update: A basic command of the DuiAgent: Update the information of a target element.

Provoke: A basic command of the DuiAgent: Triggered when a touchable UI component is used by users. It will record and upload a series of data to the cloud server to help analyze the user's pattern.

Maintain: Upload certain temporary information to the cloud server for later usage

The given process demonstrates the flexibility of the model from two aspects: First, a task can load the AbilitySlice with distributed functions and offload the AbilitySlice who does not contain. Second, the method of deciding the ultimate layout and the usage of corresponding ACS data is possible to be extended. By proposing flexibility in those two dimensions, this model is expected to generate a interactive distribution environment which offers simple approaches of direct interaction, distribution plan implementation and high extendibility supported by continuous access to ACS information. We will validate this implementation further in a case study.

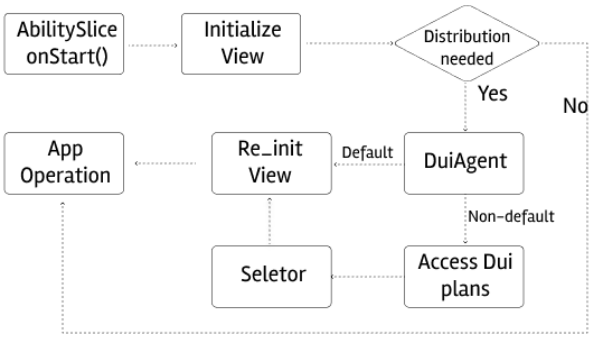

Fig. 5. Distribution workflow in the implementation

B. Case Study: Video Migration App

A multi-device video migration app is implemented based on the previously implemented UnIEM model in this section. The project structure is shown in Fig. 6, where the duiagent package is responsible to generate and maintain the interactive environment.

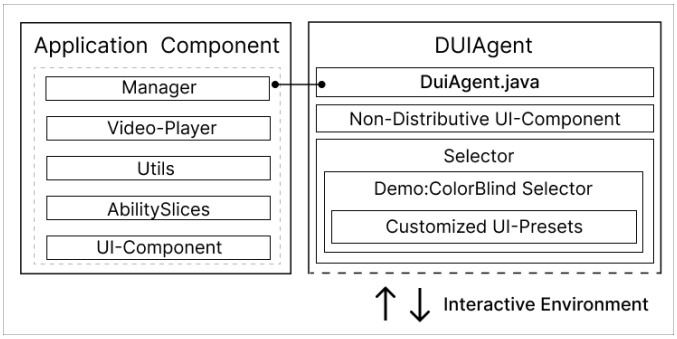

Fig. 6. UnIEM project structure

To provide clearer explanation of the interaction scheme with the support of UnIEM model, we put the entire distribution session into a scenario: In a class of dozens of students, there are two students with color blindness: Zhang and Ruby, suffering from protanopia and tritanopia respectively. At the end of the semester, the instructor, Mr. Palmer decided to give a video demonstration course work, and installed the video migration app with distribution function to facilitate the presentation. The distribution session allows students to forward the video assignments in their devices to the projector, and turn their portable device into a remote controller to manipulate the playing of the video. Knowing that Zhang has trouble distinguishing certain colors, Mr. Palmer downloaded the the app pre-installed with protanopia distribution plan ahead of time. When it was Zhang's turn to do the presentation, he surprisingly found that he can initiate a unique preset of UI that help people with red blindness to perform tasks. With the help of this designed preset, Zhang encountered no trouble when demonstrating his homework. However, when ruby tried to use the system for the first time, she discovered that both the normal plan and the protanopia plan were not optimized for her vision. Fearing that this might effect the fluency thus the score of her presentation, she informed the instructor of that issue. After knowing Ruby's trouble, instead of pausing the presentation session to update the app or postpone her round, Mr. Palmer directly installed the corresponding settings from online repository in a couple of seconds. Ruby then finished her presentation according to her plan.

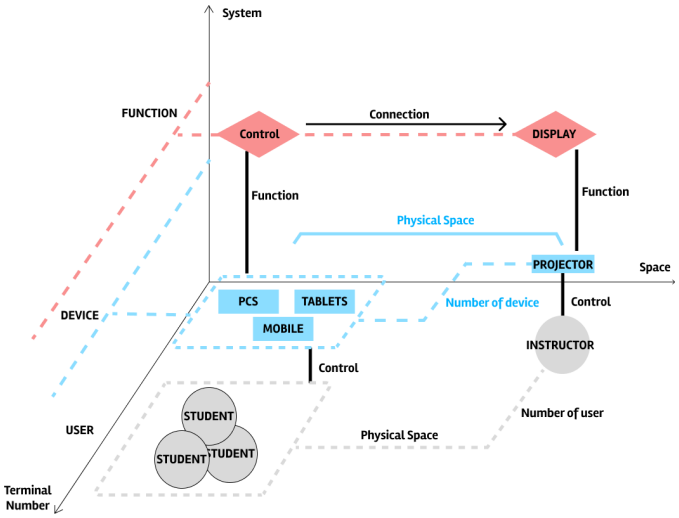

Fig. 7. Interaction model of the scheme in the case study scenario

Figure 7 indicates the structure of this whole session which includes mainly two types of distribution.

- *User Distribution*: represents the binding relation of users and their devices
- *Function Distribution*: represents the distribution of task to the appropriate devices based on the feature of the terminals and the UI components controlling the necessary functions to complete the task

When students successfully migrate the video onto the projector and try to use their own portable device as a remote Controller. The DUI system will pull up usable devices and built-in table for distributable UI elements, examine the devices information and initiate distribution based on aforementioned data. In this case, the system could rearrange the UIs on student devices to modify those devices as a controller, and adaptively change the colors of the UI element. Meanwhile, if a student registered as a colorblind student which has no corresponding color preset stored in the system, the instructor can download the required preset from online repository, and install the new preset to fit the needs by hot fix. The overall working situation is indicated in Fig. 8, where the control panel over distributed video migration can be switched from default mode to color blind mode.

During the distribution process, the change of attribute of UI elements can be accomplished in many ways. Here we explore two basic approaches: dynamic attribute assigning and preset switching. The former(Top on Fig. 9) is achieved by dynamically altering the visual attributes of a UI elements like color, size and coordinate (only color in this case study) and re-assigning the new setting to the components before the re-

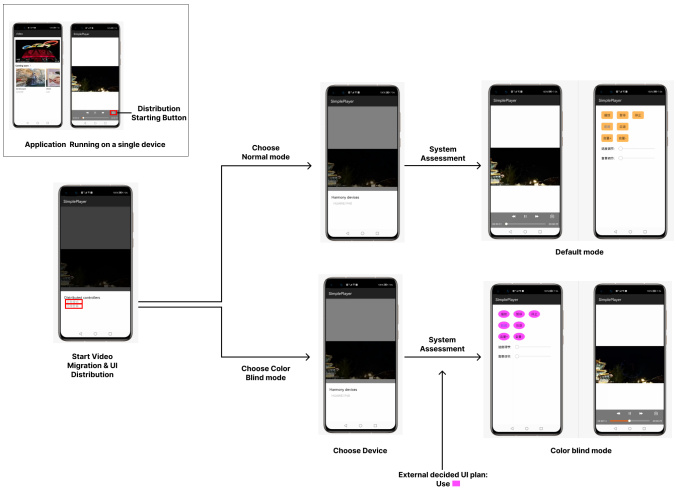

Fig. 8. The demo application working in the DUI context

initiation of the view. The other approach requires multiple sets of XML files to be edited for selection. These could be pre-designed XML layout or generated XML files based on some features.

```
public void change_col(){  // motive method of changing xml layout
    ShapeElement buttonPara = new ShapeElement();
    buttonPara.setRgbColor(new RgbColor( red: 255,  green: 0,  blue: 255));
    buttonPara.setShape(ShapeElement.OVAL);
    this.shape_button = buttonPara;

}

public ShapeElement get_shape() { return this.shape_button; }

public void change_pre(DuiAgent x){
    if(x.Select().equals("正常模式")){
        x.setter_id(ResourceTable.Layout_remote_controller);
    }else{
        x.setter_id(ResourceTable.Layout_remote_controller_two);
    }
}
```

Fig. 9. Two example methods to change the UI presentation

As a case study and brief experiment for the usability of UnIEM, the demo app is rather simple compared to a fully automated application with DUI system installed. However, although there are not too many devices or DUI configuration strategies, the current application can still offer a straightforward demonstration of how the UnIEM model can assist the IoT applications for dynamic DUI distribution at run-time and how the distribution can be determined by rendering control to the user.

In this demo app, we use the UnIEM model to realize the dynamic distribution of color blind related UI elements to provide better user experience. In the software sturcture shown in Fig. 6, *Non-distributive UI component* are the necessary classes for creating a selecting Slider for user to choose the desired distribution plan and has no functional purpose considering the operation of the system. ColorBlindSelector is a Selector class implementing Selector interface specifying the detailed logic of the execution of distribution plan. The example DuiAgent is bounded with AbilitySlice of video migration, which will migrate the video playing on one device to another device, and transmute the original one into a remote controller which is able to send commands to the new device assigned to display the video.

V. Conclusion and Future Work

In order to provide the re-configurable user interactions in a dynamic distributed IoT environment, an UnIEM interaction model is proposed in this paper. The UnIEM model is implemented based on HarmonyOS with the native distributed processing support.

The UnIEM model can generate and maintain an unified interactive environment by an agent called DuiAgent. Through the hub function of DuiAgent, each AbilitySlice will be able to gather and output UI component information and dynamically assigning the DUI configuration based on the received external command. The implemented model can provide strong flexibility and extendibility for efficient DUI distribution for any applications in a dynamic distributed IoT environment. In the case study, we have shown how the DUI run-time reconfiguration can benefit the users with different requirements without interrupt the user experience. The different users can choose their own interaction preferences on a various IoT devices.

The future work for this study could be more DUI distribution algorithms based on different technologies or scenario specific requirements. For example, the distribution can be AI-driven by deploy more machine learning algorithms. Or sometimes the DUI distribution needs to be location-aware in some scenarios. And despite that partial performance of the UnIeEM can be displayed through the initial test that involve two device and an example distributive task, the security, time efficiency and payload capacity of a UnIEM implemented application remains to be evaluated further.

References

[1] B. L. R. Stojkoska and K. V. Trivodaliev, "A review of internet of things for smart home: Challenges and solutions," *Journal of cleaner production*, vol. 140, pp. 1454–1464, 2017.

[2] A. Verma, S. Prakash, V. Srivastava, A. Kumar, and S. C. Mukhopadhyay, "Sensing, controlling, and iot infrastructure in smart building: A review," *IEEE Sensors Journal*, vol. 19, no. 20, pp. 9036–9046, 2019.

[3] A. Kirimtat, O. Krejcar, A. Kertesz, and M. F. Tasgetiren, "Future trends and current state of smart city concepts: A survey," *IEEE access*, vol. 8, pp. 86 448–86 467, 2020.

[4] F. J. García-Peñalvo, "Technological ecosystems," *IEEE Revista Iberoamericana de Tecnologias del Aprendizaje*, vol. 11, no. 1, pp. 31–32, 2016.

[5] J. A. Gallud, R. Tesoriero, J. Vanderdonckt, M. Lozano, V. Penichet, and F. Botella, "Distributed user interfaces," in *CHI '11 Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 2429–2432. [Online]. Available: https://doi.org/10.1145/1979742.1979576

[6] A. Demeure, J.-S. Sottet, G. Calvary, J. Coutaz, V. Ganneau, and J. Vanderdonckt, "The 4c reference model for distributed user interfaces," in *Fourth International Conference on Autonomic and Autonomous Systems (ICAS'08)*. IEEE, 2008, pp. 61–69.

[7] S. Lepreux and P. Renevier-Gonin, "Composition of user interfaces," in *Handbook of Human Computer Interaction*. Springer, 2022, pp. 1–38.

[8] J. Vanderdonckt *et al.*, "Distributed user interfaces: how to distribute user interface elements across users, platforms, and environments," *Proc. of XI Interacción*, vol. 20, 2010.

[9] S. Park, C. Gebhardt, R. Rädle, A. M. Feit, H. Vrzakova, N. R. Dayama, H.-S. Yeo, C. N. Klokmose, A. Quigley, A. Oulasvirta *et al.*, "Adam: Adapting multi-user interfaces for collaborative environments in real-time," in *Proceedings of the 2018 CHI conference on human factors in computing systems*, 2018, pp. 1–14.

[10] J. Melchior, J. Vanderdonckt, and P. Van Roy, "A model-based approach for distributed user interfaces," in *Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems*, 2011, pp. 11–20.

[11] N. Rizzo, G. Corti, and R. Guidi, "Bust: Distributed user interface for an iot framework," in *Proceedings of the 2018 International Conference on Advanced Visual Interfaces*, ser. AVI '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3206505.3206568

[12] D. Grolaux, P. Van Roy, and J. Vanderdonckt, "Migratable user interfaces: beyond migratory interfaces," in *The First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004.*, 2004, pp. 422–430.

[13] N. A. Streitz, J. Geißler, T. Holmer, S. Konomi, C. Müller-Tomfelde, W. Reischl, P. Rexroth, P. Seitz, and R. Steinmetz, "I-land: An interactive landscape for creativity and innovation," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '99. New York, NY, USA: Association for Computing Machinery, 1999, p. 120–127. [Online]. Available: https://doi.org/10.1145/302979.303010

[14] L. Bouillon and J. Vanderdonckt, "Retargeting web pages to other computing platforms with vaquita," in *Ninth Working Conference on Reverse Engineering, 2002. Proceedings.*, 2002, pp. 339–348.