Kadecot: HTML5-based Visual Novels Development System for Smart Homes

Shigeru Owada
Sony Computer Science Laboratories, Inc. Tokyo, Japan

Fumiaki Tokuhisa
Tokyo Institute of Technology

Abstract—**We introduce the Kadecot system, which allows visual novel (*anime*-styled story games) developers to develop easily HTML5-based contents that simultaneously control home network appliances to enhance the atmosphere of a story. Our system consists of 1) a script system to simplify event handling of devices and effects in games, 2) a device manager that dynamically downloads driver modules from the Web, and 3) nickname-based matching between content requirement and real devices. We developed several game contents that communicate with real smart homes.**

Keywords: HTML5; Smart House; Gaming

I. INTRODUCTION

The market for Smart Homes is growing fast because of the increasing interest in home energy management systems (HEMS) [1,2]. However, most application vendors focus mainly on visualization of energy, so they have not been successful in gaining greater interest from consumers. One drawback is that most HEMS suppliers do not provide open access to the data and functionality to end users. We believe the applications of smart homes will become much richer if we have a good application development environment for a home network system.

We are interested in applications of smart homes, which are not limited to energy visualization. Specifically, we believe a smart home is an excellent platform to play next-generation computer games because smart home appliances can change the whole environment of game space. From this point of view, we are currently developing a HTML5-based game development system that can simultaneously control home appliances.

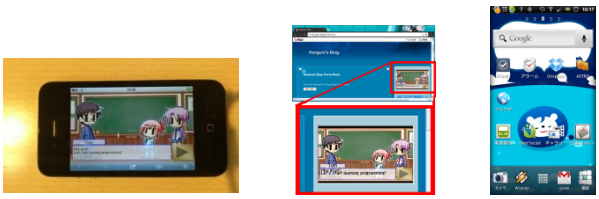
II. RELATED WORK

There are several common visual novel development systems. NScriptor and Kirikiri are the most widely used such systems in Japan [3,4]. Both platforms have a version of Android whose source files are compatible with the original PC version. However, their focus is only on developing visual novels, and device controllability is not considered at all. Processing is a very successful Java-based platform of 'software sketchbook' that is widely used by researchers, media artists, and hardware engineers [5]. Combined with a hardware prototyping kit such as Arduino, Processing is an extremely useful platform for rapid prototyping of applications that combines visuals and hardware controls. However, the language is basically Java itself, which is too difficult for visual novel developers. Flash is a complete set of ECMA-based portable software development system that has huge number of users [7]. However, our targets, visual novel developers, still prefer a more tailored scripting system that matches their purposes.

III. PROPOSED SYSTEM

A. The whole design

Our system, Kadecot, is developed in HTML5 and supports various platforms and execution modes. For example, it runs on an iPhone browser (Fig.1a), within another web page in the 'Single content mode' (Fig.1b), and a live wallpaper mode on Android (Fig.1c). The primary running environment is Android, where the system can communicate with other devices through raw TCP/IP connections.



(a) (b) (c)

Figure 1. Our system supports various execution modes.

Game contents are written in tailored programming language that is transcribed to JavaScript automatically (the language extends JavaScript with custom grammars.) Alternatively, the developers can directly use JavaScript to keep gaining support for game development and online device drivers.

B. User Interface

Our system boots with a menu that contains two tabs: 'Applications' and 'Device network'.

Figure 2. The boot menus of Kadecot

The 'Applications' tab shows bookmarks for the game contents on the Web. The 'Device network' tab contains three lists: device list, protocol list, and bookmarks for protocol definition files on the Web. Entries in device/protocol lists are the instances of available devices and protocols in the current environment. A protocol definition file contains all information of a protocol (such as DLNA ECHONET Lite) as well as all device drivers that use the protocol (TV, air conditioner, and so on.) Note that the system only stores the bookmarks of the

protocol definition files, so the bodies of drivers are dynamically downloaded from the Web.

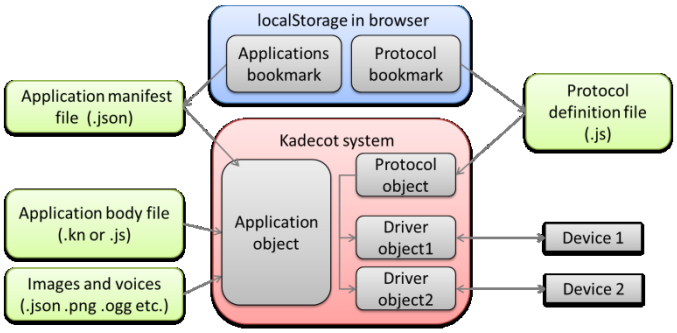


Figure 3. Diagram of Kadecot system.

The protocol/devices are instanciated by 'installing' a protocol or a device defined in the referenced protocol definition file. The user has to give a 'nickname' to each device to associate it with contents and to disambiguate similar devices.

If the user clicks on an item in the device list, a control panel of the device appears. If the device is simple enough to be controlled by a single button, such as ceiling lights (requiring only on/off control), the device is directly controlled by the clicking without showing the control panel.

A game content runs when the user clicks on an item in the applications list. If the content requires specific devices, the system prompts the user to associate installed devices by their nickname. This information of device requirements is described in an application manifest file in JSON format.

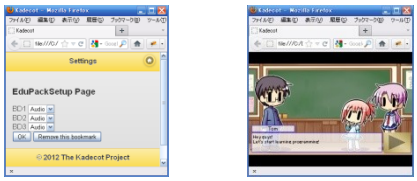


Figure 4. Device association (left) and running a game (right).

C. *Metaprogramming language system*

One difficulty in JavaScript-based software development is single threading. Because of its limited ability, the application developer always has to set up event callbacks to wait for completion of time-consuming tasks while avoiding blocking of the event and redraw loop. Callback is convenient for asynchronously handling event triggers, but it complicates the execution flow shown in the source file (the execution pointer goes back and force.) To compensate for this problem, we developed a metaprogramming language that is easily converted into JavaScript by simple rules. By using this language, the application developer can easily perform time-consuming tasks without setting callbacks. The following is an example of non-blocking call:

```
\dev.Aircond.setpower 'on'
```

This line begins with a backslash ('Aircond' is a nickname of the device). The line with a backslash heading is a non-blocking call. In this case, the system sends the 'power on' command to the device and yields the thread to the browser until a result is obtained from the device. This non-blocking synchronization style is also useful for controlling visual effects. For example, the system can wait until completion of fade-in effect without having to bother about the screen refreshing.

```
\Ann.eye 'sad',1000
```

The above command changes the appearance of Ann's eye to 'sad' with a fading effect in 1000ms in a non-blocking manner ('Ann' is the name of a character). The association of a state name (such as 'sad') with images is written in a configuration file as again, JSON format. This configuration file can be written manually or exported from Photoshop.

We also have the similar structure for text messaging:

```
Tom::Hi!
Ann::Hello!
```

A line that contains '::' is a text message command. This shows a dialog message with the speaker's name, and pauses until the user clicks the prompt button.

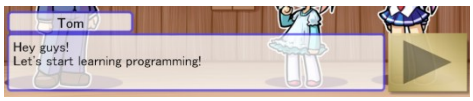


Figure 5. An example of non-blocking message display.

Another feature of the language is an event triggering by monitoring variable change in a device driver. In most cases, an application should be notified by important changes in device states. Although this is usually performed by setting callbacks in JavaScript, our system allows hooking a handler to variables in a device driver:

```
\dev.Aircond.setWatch 'temperature', temp_change
#temp_change
console.log('Temp : ' + vm.dev.AirCond.temperature ) ;
```

The first line sets a hook to the 'temperature' variable in the 'Aircond' device. '#' at the beginning of a line is a label that creates a new code block. Whenever the temperature changes, the 'temp_change' block is executed.

IV. APPLICATIONS

We developed some game applications that play visual novels while controlling real devices.

Figure 6. An episode of fan and air-controller

Figure 7. Dating remote controler of a fan.

Figure 8. Software updating with character goods

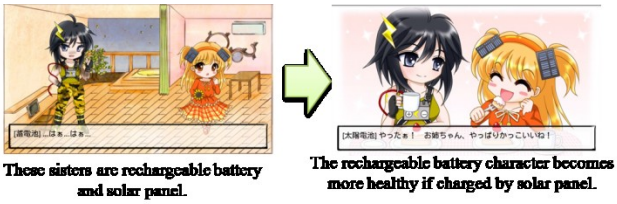


Figure 9. An episode to promote green energy

Figure 10. An air-conditioner game to urge the game player to set the temperature to the character's preference

The story of a fan and an air-conditioner shown in Fig. 6 tries to teach the game player how to save electricity by simultaneously using a fan and an air-conditioner. The fan controller dating and software updating episodes (Figs. 7 and 8) try to make ordinary operations of the devices more fun. The battery sisters episode (Fig. 9) tries to promote power accumulation by green energy (solar panel.) Another air-conditioner game (Fig. 10) assumes there is a predefined strategy for room temperature control (programmed by the game player or someone else) and the application tries to incentivize the player to continue the strategy with the cute appearance and entertaining behavior of the characters. The underlying idea is that the automated temperature control is, in most cases, not flexible enough for real use because of the daily fluctuations or unusual events. This game tries to give gentle reminders to the player.

V. PUBLIC EXPOSURES

We performed a joint demonstration event with Daiwa House Industry co. ltd. in July 2011, to announce and test our system. In this event, we controlled devices by Housing API protocol on DHEMS server, both developed by Daiwa House Industry. Housing API uses the ECHONET protocol as the back-end system and connected to air-conditioners and fans, lights, a water heater, and so on. The event was reported by various media, including two TV stations, seven newspapers, and over 16 news websites. They generally accepted the concept of programmable homes and combining home services with game contents. The feedback from readers of the news stories and event audience members contained strong criticism about the quality of the contents themselves.

We also released a corresponding Android application on the Google Play market. This version contains driver modules for Sony's Blu-ray recorders and power loggers for Housing API. There had been 6405 downloads and 2586 active installs by July 31, 2012.

Figure 11. Kadecot distribution page at Google Play

VI. CONCLUSION AND FUTURE WORK

We introduced a visual novels development system implemented in HTML5 that can simultaneously control home appliances. We developed a scripting system on top of JavaScript to handle asynchronous control, as well as visual effects necessary for visual novels. Our system dynamically includes device/protocol drivers from the Web and allows management in a unified user interface.

One drawback of our current system is the lack of a method for searching for good contents or protocol definition files on the Web. Our future work is to develop a portal function into the application. Another future work is to add network game support.

REFERENCES

[1] U.S. Congress, "American Recovery and Reinvestment Act of 2009", .
[2] ECHONET CONSORTIUM (http:// www.echonet.gr.jp/).
[3] N.Takahashi. NScriptor (http://www.nscripter.com/)
[4] W.Dee. KiriKiri (http://kikyou.info/tvp/)
[5] B.Fry and C.Reas. Processing (http://processing.org/)
[6] Arduino (http://www.arduino.cc/)
[7] Adobe Flash Professional (http://www.adobe.com/jp/products/flash.html)