Kadecot: Android Web API Server for Home Appliances and Sensors based on WAMP Protocol

Shigeru Owada
Sony Computer Science Labs, Inc.
sowd@csl.sony.co.jp

Kazuhito Nakamura
Sony corporation

Masahiro Karaki
Crestec, inc.

1. OUR INTEREST

We are developing an Android home server that communicates with home electronics and sensor devices. This server provides a WAMP-based web API interface for easy access to devices through common Web based technology. We are interested in developing services and user interfaces to utilize our APIs. We are ready to demonstrate our agent-based application at the workshop.

2. IMPLEMENTATION

Our Android home server application, named 'Kadecot', automatically recognizes ECHONET Lite devices through UDP. At the same time, it works as a WebAPI server using WebSocket to enable connections from Web applications. Our API system is based on WAMP (The Web Application Messaging Protocol).

The system architecture is as follows.

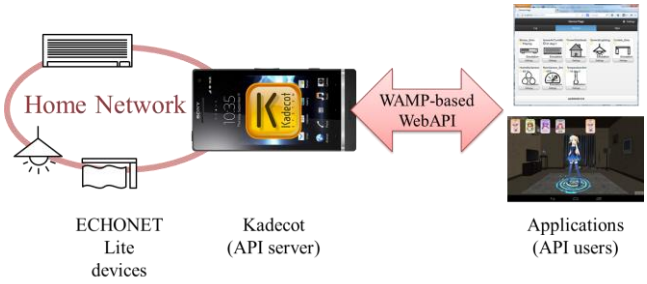


Figure 1. System Architecture

WAMP consists of two entities: A router and a client. The client generates and consumes events while the router transfers messages. On the other hand, our system has three entities: API users (Web applications), an API server (Kadecot), and home devices. We map API users and home devices as clients and the API server mainly as a router (some functions of the API server work as clients.)

WAMP defines two messaging types: RPC and PubSub. Each type consists of three subjects. The RPC subjects are caller, callee, and dealer, while the PubSub subjects are subscriber, publisher, and broker. These subjects are called roles. In a WAMP framework, each client can be a caller, a callee, a publisher, and a subscriber. However, in order to simplify the explanation for this document, we have assigned each API user's role as either a caller or a subscriber, while a device object's role is either a callee or a publisher. The API server's role is mainly a dealer and a broker. The flow of messages is shown below.

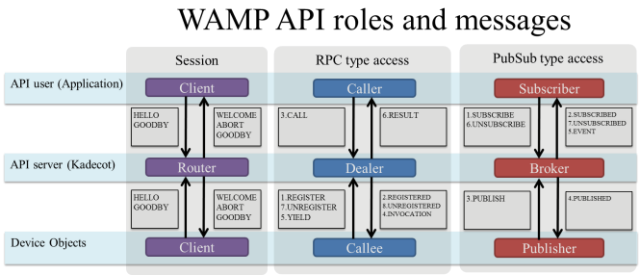


Figure 2. WAMP messages in our system

3. ISSUES

All WAMP frameworks work well for RPC-type messaging. On the other hand, applying PubSub types to device message passing introduces difficulties in performance. In a PubSub framework, an information provider (publisher) autonomously publishes updated information to the broker without receiving concrete requests from the subscriber. It requires all possibly necessary information to be published by devices, regardless of the API user's request. This requirement increases unnecessary access to devices. For example, ECHONET Lite Air-conditioner has up to 26 (super class) + 45 (air-conditioner specific) node properties. It is not realistic to maintain all these values all the time. Therefore, it is necessary to adaptively control publish strategies for application requests.

4. EXAMPLE APPLICATION

We are trying to implement some sample applications using our own API. One is a 3D-character-based remote control application implemented using Unity, popular 3D multi-platform middleware. The socket library we used within Unity was WebSocketSharp.

Figure 3. Application main screen

Once connected, a concierge character appears and connected devices are listed as cards on top of the screen. We describe RPC and PubSub patterns within this application below.

4.1 Sending remote control signals (RPC)

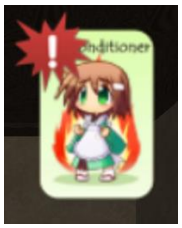
When the user touches a card, a magic effect is applied to the character and an RPC message is sent to the dealer.

There is also a traditional button-based user interface.

4.2 Receiving error notifications (PubSub)

This application dynamically receives error notices from devices. This is implemented as a PubSub pattern. Error notification is defined as a mandatory field in ECHONET Lite. All devices publish this property on change.

In our application, an error notification is shown as a red balloon at the top-left corner of a card.



When the user touches a card with an error notification, the related HTML manual is displayed.

When necessary, the user is forwarded to an online shop to buy consumables.

The user is notified on completion.

5. FUTURE WORK

Our system does not perform any authentication of clients (especially API users.) We are planning to implement authentication using the advanced profiles defined in WAMP. In addition, we are interested in adding UPnP device discovery to the API server.

REFERENCES

WAMP homepage http://wamp.ws

Kadecot Project http://kadecot.net

WebSocketSharp https://github.com/sta/websocket-sharp