COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

COMS 6181

Advanced Internet Services

Fall 2011

Professor Henning Schulzrinne

Course Project Report

SECE Explorer

Kum2104
Kunal Mudgal
(Server Implementation)

md3073
Manu Dhundi
(Client Implementation)

Goal:

To develop an application (SECE Explorer) that discovers IP enabled services (may or may not be devices) in a local (home) network and reports the same to a centrally located server (SECE Server). Also send the service details like IP, Port, service type, service addition or service removal to the central server. The server to display the list of devices discovered on a Web Page and to send some commands like print using a discovered printer.

Communication between Client and Server has to be using standard protocols like HTTP and should use structured languages like JSON or XML. Code to be kept modular enough to accommodate any additions or improvements.

SECE: Sense Everything, Control Everything

SECE Explorer Client:

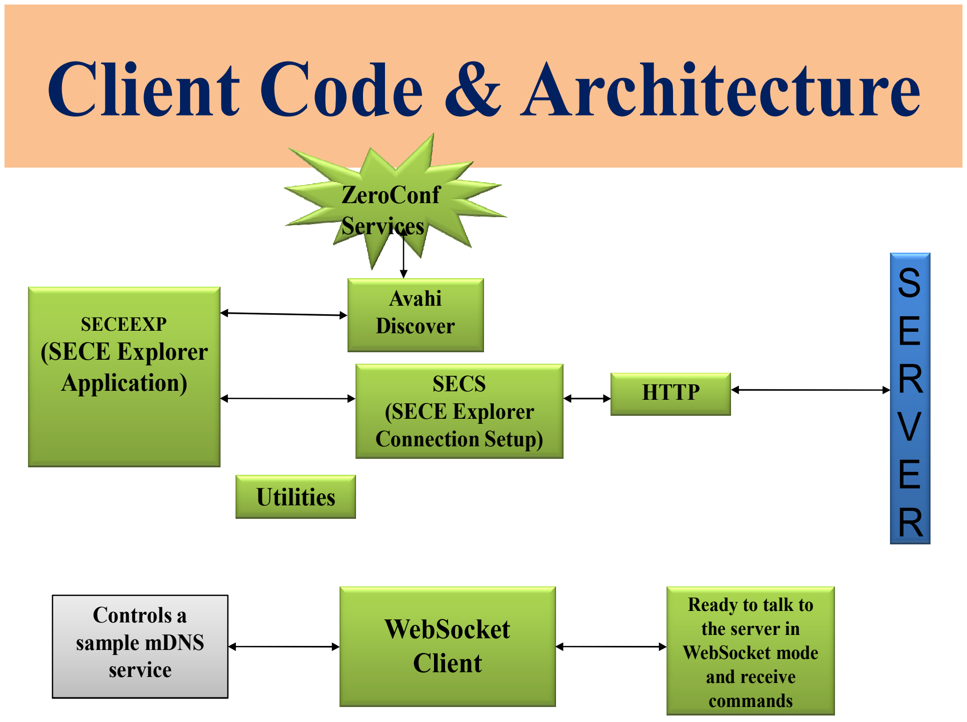

Client Code & Architecture
ZeroConf Services
Avahi Discover
SECEEXP (SECE Explorer Application)
SECS (SECE Explorer Connection Setup)
HTTP
SERVER
Utilities
Controls a sample mDNS service
WebSocket Client
Ready to talk to the server in WebSocket mode and receive commands

Figure 1: SECE Explorer Client Architecture

The SECE Explorer Client Architecture is captured in figure 1. It has 2 distinct applications.

1. SECE Explorer Application which discovers mDNS Services:

The activity of this application is to discover mDNS services in the local (home) network and send them to the central SECE Server. There are 3 modules in this application.

(a) Avahi Discover:

This module discovers all the mDNS services in the local (home) network. It runs on a separate thread and polls continuously for service additions or removals. It also gets the details like IP Address, Port, Service Name, Host Name, whether the service is being added or removed and sends those to the application module, SECEEXP, in a structured format as agreed between the 2 modules. It is important to abstract out the details of service discovery to the SECEXP because, there may be more than one mechanism to discover services may be employed in future. Further, the SECEXP is concerned only with the services discovered and not the way in which they are discovered.

This module uses avahi-core.a and avahi-common.a libraries. The code for this module has been adapted from the sample codes for avahi discover.

(b) SECEEXP:

This is the application module. It creates necessary resources like threads for all other modules. It maintains the configuration parameters. It receives the discovered devices from the Avahi Discover module and sends them to the SECS (SECE Explorer Connection Setup) module.

SECS module and SECEEXP talk to each other in a structured format as agreed upon before. SECEEXP is not concerned with the mechanism of server communication. It only wants to make sure that the data it passes on to the SECS module reaches the server.

(c) SECS (SECE Explorer Connection Setup):

This module is responsible for communicating with the server with appropriate protocols. It prepares the JSON payload for the HTTP module. The JSON payload contains the service info which can be interpreted by the server. JSON operations are done using json.a library.

If there are any changes in mode of server communication in future, like addition of security or xml payload, then those changes are to be handled in this module.

(d) HTTP:

This module establishes HTTP connection with the server to send the service details to the server. It uses curl.a library. HTTP POST and HTTP PUT methods are made possible. As of now, the server supports HTTP POST and hence HTTP POST is being used.

(e) Utilities:

This module aims to contain utility functions which are needed by different modules. It is important to have a separate utility module to avoid code duplication and unnecessary dependencies across the modules.

2. WebSocket Client to receive control commands from the server

This application has websocket client ready. It can receive data from the servers that support websockets. It send the HTTP GET request with "connection upgrade" field parameters. It can parse the HTTP 101 server response which indicates websocket support and keep the socket connection open to receive the data from the server.

Until the websocket is ready at the SECE server, it reads a dummy command in the JSON format and sends them to a sample mDNS service.

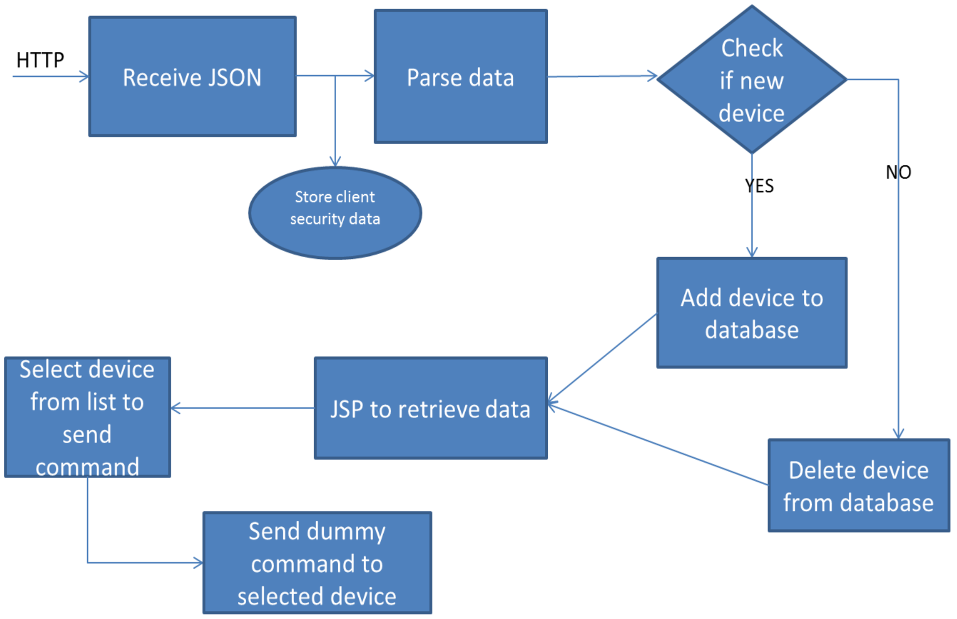
3. A sample mDNS Service:

This is a sample mDNS service which gets discovered by the SECE Explorer and can read and display the commands received from the websocket client application.

Server

Technologies: Java Servlets, JSP, MySQL Database, Glassfish Server 3.1

Architecture:


HTTP
Receive JSON
Parse data
Store client security data
Check if new device
YES
NO
Add device to database
Delete device from database
JSP to retrieve data
Select device from list to send command
Send dummy command to selected device

Database table
Devices

Column	Attribute
ServiceName	varchar(200)
ServiceType	varchar(50)
ServiceIP	varchar(100)
HostName	varchar(100)
Port	int(10)
ServiceState	varchar(10)

Design:

The server has been deployed on Glassfish 3.1 server. The JSP servlet is ready to ready communication once deployed via HTTP POST method. The JSON data received in payload is stored in a temporary file at the server side.

The server parses this JSON file and stored in variables. If the device is a new device, servicestate =1 and does not exist in the table, we add it to the table. If the device comes with servicestate = 0 i.e. marked for deletion, the we delete it from the table

This data can be displayed on a webpage via JSP hosted on the server.

The list of devices is shown from which the user can select one. Once the user selects a particular device, all the details of that device are displayed. We intend to send the command to the device via JSP.

Future Scope:

1. Make SECE Explorer control an actual service as per commands from SECE server.
2. Integrate our SECE server with the main SECE server.
3. Add a security feature (probably light weight) in the SECE Explorer so that the explorer at the home talks securely with the SECE server and intruders are prevented from controlling the home devices.
4. Develop UPnP discovery mechanism to discover more devices in the home network.