The Hague University of Applied Sciences

Project System Development

Version 1.0

February 17, 2023

Author:
Mike Rietveldt, 20183380

Supervisors:
Mr. J.G.M.W Vd. Helder
Mr. G.S Mijnarends

Socket communicatie (Verbinden)

Om te beginnen is het belangrijk te weten hoe de server precies werkt. In dit geval is het een server die altijd luistert, deze staat dus continu aan na het starten. Een client kan op elk gewenst moment verbinding maken zolang de server is gestart. Mocht de server crashen dan wordt deze opnieuw gestart via de terminal, in mijn geval gebeurd dit op de Pi. De client zal verbinding gaan maken wanneer deze gestart wordt en de juiste gegevens mee heeft gekregen om naartoe te verbinden.

Ervan uitgaand dat de server en client bestaan, zal hieronder toegelicht worden wat er gebeurd wanneer de server aan staat en de client wilt verbinden. Ook zal er toegelicht worden hoe de server en client handelen wanneer er een actie plaats vindt. Onderstaande uitleg is de deur samen met de server, het principe is hetzelfde voor de andere domotica. Het grote verschil is dat de actiefuncties zullen verschillen.

De volgende code snippets komen uit de Deur.ino file

```
#ifndef WIFISSID
#define WIFISSID "BeamersWifi"  // Wifi network name.
#define WIFIPASS "112233ja"     // Password of the wifi network.
#endif

const char* ssid = WIFISSID;
const char* password = WIFIPASS;
const char* host = "192.168.137.10";  //Ip of the Pi, can also be pi@mikePi.local.
const uint16_t port = 8080;
```

Dit is de client die zal verbinden naar de Pi (De server). Om dit te doen moet de client zich in dezelfde wifi omgeving begeven als de server. Daarom geven wij aan waar de client naar toe kan verbinden, zo geven wij een SSID mee, het wachtwoord , de host en de port.

```
if (WiFi.status() != WL_CONNECTED) {
  connectToWifi();
  return;
}
```

Zodra wij in de loop komen voeren wij een check uit. Deze check is er om te kijken of er al een verbinding met de wifi is. Als dit niet zo is wordt de functie “connectToWifi()” aangeroepen. Daarna returned het programma naar het begin van de loop. Dit gebeurd niet direct, er zit namelijk een timer in de connectToWifi functie zodat hij de tijd heeft om de verbinding te maken, deze functie zal zo toegelicht worden in het volgende plaatje.

```
void connectToWifi() {
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);

  Serial.print("\nEstablishing Connection to wifi network");
  for (int i = 0; i < 10; i++) {
    if (WiFi.status() != WL_CONNECTED) {
      Serial.print(".");
      delay(1000);
    } else {
      Serial.print("\nWiFi connection established on " + WiFi.localIP().toString());
      return;
    }
  }
  Serial.print("\nCouldn't establish a connection to wifi network");
}
```

Dit is de connectToWifi functie die zojuist genoemd werd. In de functie begin je met het aangeven van de wifi modus en vervolgens start je de wifi verbinding met het vooraf gegeven SSID en wachtwoord. Daarna zal een bericht in de client terminal weergeven worden wat er precies gebeurd, zolang de verbinding nog niet gemaakt is zal er om de seconde een “.” Geprint worden. Hierdoor kun je zien dat hij bezig is met verbinden. Wanneer de verbinding uiteindelijk gemaakt is wordt weergeven op welk IP dit is gebeurt. Uiteindelijk return je waardoor we terug gaan naar de main. Mocht de verbinding nou niet gelukt zijn dan wordt dit ook aangegeven.

```
if (!connectToServer())
  return;

// Set server configuration to match door usage.
Serial.print("\nSent: setDeur to server");
client.print("setDeur")
```

Hier kijken wij of er al een connectie met de server is, als dit niet zo is gaan we terug naar het begin en proberen wij het opnieuw. Wanneer de verbinding gemaakt is wordt dit aangegeven in de functie connectToServer() die hieronder beschreven staat. Ook zal de client een bericht versturen met “setDeur” naar de server. Die weet nu met welk device hij te maken heeft.

```
bool connectToServer() {
  Serial.print("\nEstablishing connection to (Pi) server");

  if (!client.connect(host, port)) {
    Serial.print("\nCouldn't establish a connection to (Pi) server on ");
    Serial.print(host);
    Serial.print(":");
    Serial.print(port);
  }

  for (int i = 0; i < 10; i++) {
    if (client.connected()) {
      Serial.print("\n(Pi) Server connection established on ");
      Serial.print(host);
      Serial.print(":");
      Serial.print(port);
      //Serial.print("" + host + ":" + port.to);
      return 1;
    }
    Serial.print(".");
    delay(1000);
  }

  Serial.print("\nCouldn't establish a connection to (Pi) server on ");
  Serial.print(host);
  Serial.print(":");
  Serial.print(port);
  // Serial.print( + host + ":" + port);
  return 0;
}
```

Het eerste wat opvalt is dat deze functie een bool is. Dat komt doordat de waarde die deze functie teruggeeft eigenlijk laat zien of de connectie wel of niet is gelukt met de server.
In de connectToServer functie start een print waarin vermeld wordt in de client terminal dat er geprobeerd wordt een connectie met de pi server te maken. Vervolgens checken wij of de client nog geen verbinding heeft. Als die geen verbinding heeft wordt dit aangegeven en vervolgens komt hij in een 10 seconde loop waarin hij blijft verbinden. Als het na die 10 seconde niet gelukt is krijg je een melding dat er geen verbinding gemaakt kon maken. Mocht de verbinding wel gemaakt zijn dan wordt er vermeld dat de verbinding is gemaakt op host xxx : port xxx hierna returnen wij waardoor het programma verder gaat.

```
while (client.connected()) {
    int input = readInput();

    if (input != 0 && input == previousInput) {

      Serial.print("\nSame input detected, skipping...");
      delay(1000);
      continue;
    }
```

Wanneer de client connected is moeten wij zorgen dat de inputs gelezen worden. Deze zullen later aan de pi doorgegeven worden. Er zit een check in waardoor de pi en de client niet gespammed kan worden met requests. Dit wordt gedaan doormiddel van een check waarin de input vergelijken wordt met de previousInput, previousInput wordt aan het begin op 999 gezet. Wanneer er een input plaatst vindt zal de check gerunt worden en daarna wordt de input in de previousInput gezet. Vervolgens wordt de daadwerkelijke input weer op 0 gezet, er wordt ook gechecked of de input niet gelijk is aan 0, dit houdt namelijk in dat er niets in ingedrukt en er is dan dus ook geen check nodig.

Het lezen van de inputs (2 knoppen op de deur) en het versturen van de inputs naar de pi zal in volgende code snippets laten zien worden onder het kopje Socket communicatie (Acties).

Socket communicatie (Acties)

In het stukje acties gaan wij kijken naar het lezen van de waardes, het versturen van de waardes en het uitvoeren van verschillende acties. Er zullen code snippets uit meerdere files voorkomen. Boven elk plaatje zal schuingedrukt staan in welke file deze code staat.

Deur.ino

```
int readInput() {
  // Config PCA9554
  // Inside loop for debugging purpose (hot plugging wemos module into i/o board).
  // IO0-IO3 as input, IO4-IO7 as output.
  Wire.beginTransmission(0x38);       // slave adres meegeven
  Wire.write(byte(0x03));
  Wire.write(byte(0x0F));
  Wire.endTransmission();

  // Read PCA9554 outputs (IO40-IO3)
  Wire.beginTransmission(0x38);       // slave adres meegeven
  Wire.write(byte(0x00));
  Wire.endTransmission();
  Wire.requestFrom(0x38, 1);
  unsigned int inputs = Wire.read();  // Hier ken je de waarde toe aan input
  return inputs & 0x03;               // Input waarde terug geven aan de main
}
```

In dit stukje worden de knoppen uitgelezen. Om te beginnen geven we aan op welke slave de transmission begint. Hierna geven wij aan welke IO er op input en output moeten staan, vervolgens zetten wij de juiste bitjes naar gewenste waarde en eindigen wij de transmission. Wanneer de transmission beeindigt wordt zullen de bits die ervoor gezet zijn ook daadwerkelijk verstuurd worden naar de slave. beginTransmission zorgt er dus eigenlijk voor dat deze bitjes in een soort wachtrij komen te staan. In het tweede gedeelte gebeurd ongeveer het zelfde maar deze keer zetten wij geen inputs. In de requestFrom geven wij aan van welke slave wij iets willen lezen en het aantal bytes dat hieruit gelezen moet worden. In dit geval lezen wij 1 byte uit van slave 0x38. Vervolgens wordt er een int inputs gemaakt en daar kennen wij de waarde van de knop aan toe. Dit returnen wij vervolgens aan de main zodat hier gebruik van kan worden gemaakt.

Deur.ino

```
if (input != previousInput) {
      previousInput = input;
    }

    if (client.connected()) {
      if (input == 1) {
        client.print("deurOpenBinnen");
        delay(1000); // Depends on wifi speed
      }
      if (input == 2) {
        client.print("deurOpenBuiten");
        delay(1000); // Depends on wifi speed
      }
    }

    if (client.available()) {  // Lezen wat de server verstuurt en printen.
      char message = static_cast<const char>(client.read());

      // Print callback from server
      Serial.print("\nPi zegt: ");
      Serial.print(message);
```

Hier werken wij naar het versturen van de data naar de Pi. We beginnen met het checken of de input niet gelijk is aan de previousInput, als dit inderdaad niet gelijk is zetten wij de input in previousInput. Hierna checken wij of de client nog geconnect is en kijken wij of de input 1 of 2 is. Als dit het geval is wordt het bijbehorende bericht doorgestuurd aan de Pi. Daarna wordt gecheked of de client nog avalaible is. Als dit het geval is lezen wij de reactie die de Pi stuurt uit, en stoppen wij dit in een char genaamd message. De client print nu in de terminal “Pi zegt: x” waarin x het gestuurde getal van de pi is.

Server.cpp

```
void Server::recvInputFromConnection(int fd)
{
    // Receive data
    int checkrecv = recv(fd, buffer, BUFFER_SIZE, 0);
```

Hier wordt de data die de client stuurt, ontvangen op de Pi. Dit wordt in de buffer gezet.

Actions.cpp

```
void Apartment::processDeur(const std::string& action) const {
    // Call the function corresponding to the action code
    if (strcmp(action.c_str(), "deurOpenBinnen") == 0) {
        deur->deurOpenBinnen();
    } else if (strcmp(action.c_str(), "deurOpenBuiten") == 0) {
        deur->deurOpenBuiten();
    } else if(strcmp(action.c_str(), "brand") == 0) {
        deur->brand();
        meldBrand();
    }
}
```

Zoals in een eerder code snippet zichbaar was wordt er een “setDeur” gestuurd naar de Pi. Na een paar tussenstapjes komt het nieuw gestuurde bericht hier terecht. Er wordt dan een string compare uitgevoerd. Wanneer de actie overeenkomt met de string, in dit geval dus “deurOpenBinnen” of “deurOpenBuiten” wordt de daarbij behorende functie aangeroepen. In deze functie sturen we het juiste karakter. Bij deurOpenBinnen is dit bijvoorbeeld “1”. Hierdoor weet de client dus wat hij moet doen.

Controller.cpp

```
void handleServerInput(Apartment *apartment, uint16_t fd, char *buffer)
{
    // Display the received input and the sender
    std::cout << "Got input '" << buffer << "' from " << fd << ".\n";

    // Parse the input
    Parser prs;
    prs.parse(buffer);

    // Retrieve the action code
    std::string key;
    try
    {
        key = prs.values.at(0);
    }
    catch(const std::out_of_range& e)
    {
        std::cerr << "no action" << std::endl;
    }
```

Hier wordt de ontvangen data netjes weergeven in de terminal, van welk device het bericht komt. En wat de message is. De message wordt geparsed waardoor je alleen het keyword over houdt. Dit keyword wordt gebruikt in het vorige stukje code snippet, om de juiste actie uit te voeren.

Deur.h

```
    void deurOpenBinnen() const { sendMessage(this->fd, "1"); }

    /**
     * @brief Confirm that the door is allowed to open/close
    */
    void deurOpenBuiten() const { sendMessage(this->fd, "2"); }
```

Hier zien we de functies die aangeroepen worden in de vorige code snippet van Actions.cpp.
Wanneer de functie wordt aangeroepen zul je dus een karakter sturen naar de Pi die bij de specifieke actie behoort.

Socket communicatie (Overige info)

In dit stukje zal ik nog wat extra dingen toelichten. Hoe de Pi verschillende berichten weergeeft met verschillende acties.

```
void handleServerConnection(uint16_t fd)
{
    // Display the connected device
    std::cout << "Controller: got connection from " << fd << std::endl;
}
```

Hier zien we hoe de Pi een connection ontvangt van de client. Fd is het nummer dat een bepaald device ontvangt. Het ziet er dan uit als “Controller: got connection from x” waarin x dan het nummer is dat de device(client) krijgt toegewezen.

Testrapport

In dit stukje vindt u het testplan van Mike Rietveldt. Dit document is onderdeel van de herkansing project software system development. Als opdracht voor de herkansing moest het volgende opgeleverd worden: twee devices kunnen aansluiten, documentatie, socket communicatie en een klassediagram met toelichting. Om de socket communicatie en de twee devices te kunnen testen, volgt hieronder een testplan.

Testplan

De testen moeten voldoen aan de volgende eisen.

1. Testnummer: hierin staat de eis genoteerd waaraan het systeem moet voldoen.
2. Test handleiding: hierin wordt stap na stap beschreven hoe het getest moet worden.
3. Het verwachte resultaat van de test: de voorspelling van de uitkomst van de test
4. Het werkelijke resultaat: het daadwerkelijke resultaat van de test
5. Geslaagd of niet geslaagd.

Testen deur

TestID: 1D	*Deur gaat open na druk op knop*
Test handleiding	- *Druk eenmalig op een van de knoppen.* - *Laat knop direct weer los*
Het verwachte resultaat van de test	*Deur opent naar de tegenovergestelde richting*
Het werkelijke resultaat	*Deur opent naar de tegenovergestelde richting*
Geslaagd of niet geslaagd.	*Geslaagd*

TestID: 2D	*Deur gaat dicht na druk op knop*
Test handleiding	- *Druk eenmalig op een van de knoppen.* - *Laat knop direct weer los* - *Druk knop nog eens in wanneer deur open is* - *Laat direct weer los*
Het verwachte resultaat van de test	*De deur sluit niet*
Het werkelijke resultaat	*Deur sluit niet, de timer wordt gewoon afgemaakt*
Geslaagd of niet geslaagd.	*Geslaagd*

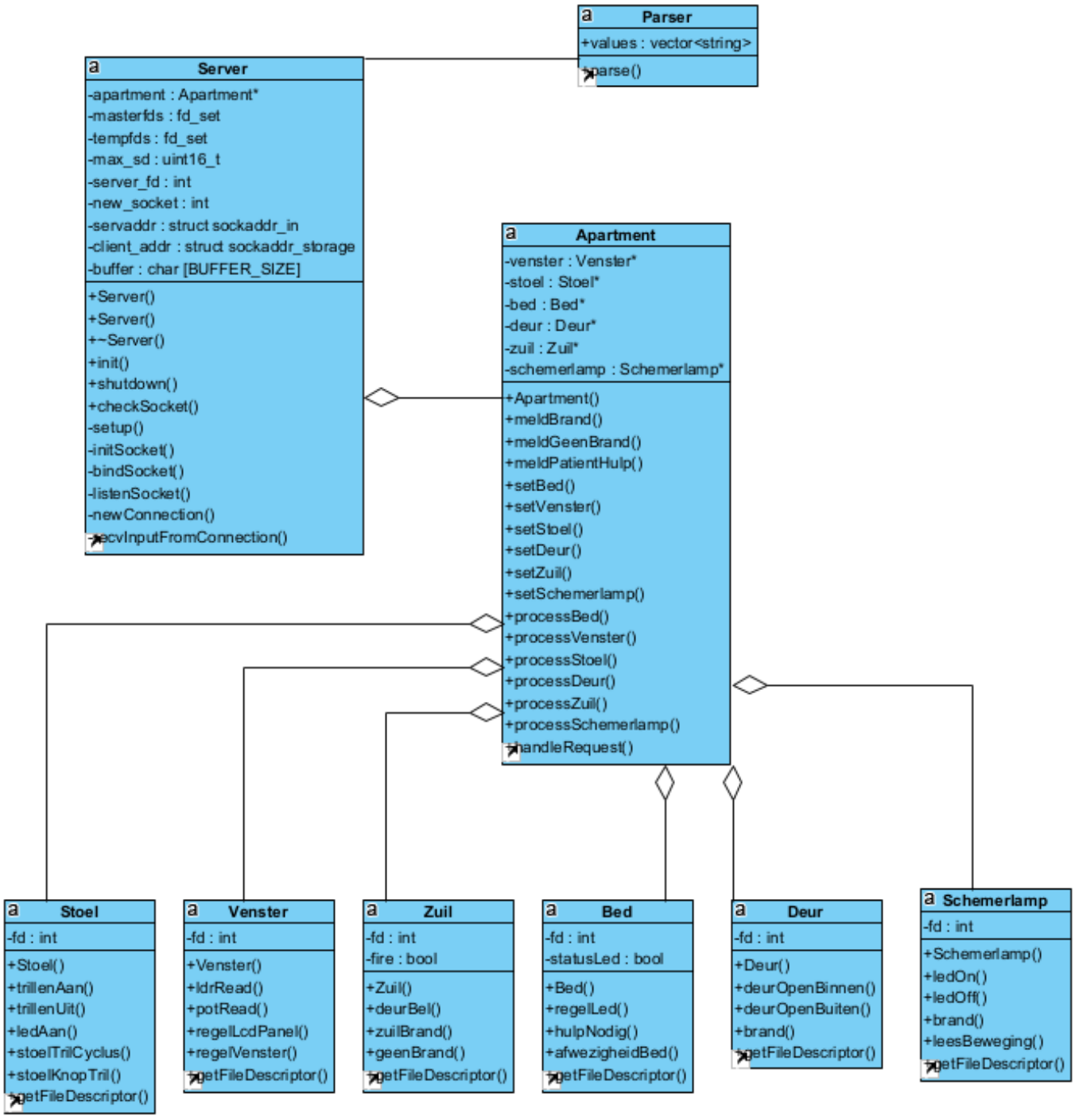
TestID: 3D	*Deur sluit wanneer hij open staat en de stekker eruit getrokken wordt*
Test handleiding	- *Druk eenmalig op een van de knoppen.* - *Laat knop direct weer los* - *Trek stekker uit de deur wanneer deze opent*
Het verwachte resultaat van de test	*De deur sluit na de timer*
Het werkelijke resultaat	*Deur sluit na de timer*
Geslaagd of niet geslaagd.	*Geslaagd*

Testen schemerlamp

TestID: 1S	*Schemerlamp gaat aan na detecteren van beweging*
Test handleiding	- *Zorg dat schemerlamp beweging detecteert* - *Ga na de beweging uit de sensor zijn bereik*
Het verwachte resultaat van de test	*Groen licht zal aan gaan.*
Het werkelijke resultaat	*Groen licht gaat aan*
Geslaagd of niet geslaagd.	*Geslaagd*

TestID: 2S	*Schemerlamp gaat uit na ongeveer 10 seconde*
Test handleiding	- *Zorg dat de lamp aan gaat zoals in test 1S* - *Wacht ongeveer 10 seconde*
Het verwachte resultaat van de test	*Het licht gaat uit*
Het werkelijke resultaat	*Licht gaat uit*
Geslaagd of niet geslaagd.	*Geslaagd*

Klassendiagram



Toelichting Klassediagram

Associatie tussen parser en server: *de server maakt wel gebruik van de parser maar de parser is geen onderdeel van de server. Vandaar dat het een associatie is.*

Aggregatie tussen server en apartment: *De server werkt met een pointer naar apartment. Bij een compositie wordt klasse B verwijderd wanneer de destructor van klasse A wordt aangeroepen, met pointers is dit niet altijd het geval. Vandaar is dit een aggregatie.*

Aggregatie tussen server en meubels: *Hetzelfde gebeurd tussen Apartment en meubels. De meubels worden niet verwijderd wanneer het apartment verwijderd wordt. Hierdoor hebben wij dus te maken met een aggregatie.*

De reden dat ik voor deze structuur heb gekozen is het feit dat dit qua toekomstmogelijkheden goed uitbreidbaar is. Het zou omgebouwd kunnen worden tot een server die meerdere appartementen kan hendelen. Ook heb ik hier alle meubels erin gelaten zodat deze makkelijk toegevoegd kunnen worden. Dit zou met wat herschrijven van de functies redelijk makkelijk mogelijk zijn.