IMPLEMENTASI ALLJOYN FRAMEWORK PADA LINGKUNGAN RUMAH CERDAS SECARA *PERVASIVE*

SKRIPSI

KEMINATAN TEKNIK KOMPUTER

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Nur Lailatul Choiriyah
NIM: 125150300111004

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2016

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

PENGESAHAN

IMPLEMENTASI ALLJOYN FRAMEWORK PADA LINGKUNGAN RUMAH CERDAS SECARA *PERVASIVE*

SKRIPSI

KEMINATAN TEKNIK KOMPUTER

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Nur Lailatul Choiriyah
NIM: 125150300111004

Skripsi ini telah diuji dan dinyatakan lulus pada
28 Juli 2016
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Sabriansyah Rizqika Akbar, S.T., M.Eng.	Adharul Muttaqin, S.T., M.T.
NIP: 19820809 201212 1 004	NIP: 19760121 200501 1 001

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T., M.T., Ph.D.
NIP: 19710518 200312 1 001

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata di dalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 15 Agustus 2016

Nur Lailatul Choiriyah

NIM: 125150300111004

KATA PENGANTAR

Puji dan syukur penulis panjatkan kepada Tuhan Yang Maha Esa, karena atas limpahan rahmat dan hidayahnya sehingga dapat terselesaikannya tugas akhir skripsi yang berjudul "Implementasi Alljoyn Framework Pada Lingkungan Rumah Cerdas Secara *Pervasive*". Atas bantuan moral dan materil yang diberikan kepada penulis, maka penulis juga mengucapkan banyak terima kasih kepada :

1. Bapak Nur Hasan dan Ibu Suyatmi yang penulis cintai serta seluruh keluarga Alfiatur Rohma, Mochammad Saifudin, Halimatus Sa'diyah, Zainul Arifin, Achmad Irfan Afandi, Wiwik Susanti, Supono dan Matarum yang selalu mendukung dan memberi doa agar penulis dapat dengan lancar menyelesaikan skripsi.
2. Bapak Sabriansyah Rizqika Akbar, S.T., M.Eng. selaku dosen pembimbing satu yang telah meminjamkan Raspberry Pi, membimbing serta memberikan ilmu, saran, dan motivasi kepada penulis.
3. Bapak Adharul Muttaqin, S.T., M.T. selaku dosen pembimbing dua yang telah membimbing serta memberikan ilmu, saran, dan motivasi.
4. Bapak Tri Astoto Kurniawan, S.T., M.T., Ph.D. selaku Ketua Jurusan Teknik Informatika Fakultas Ilmu Komputer.
5. Seluruh civitas akademika Fakultas Ilmu Komputer Universitas Brawijaya yang telah banyak memberi bantuan dan dukungan selama penulis menempuh studi Teknik Informatika di Universitas Brawijaya dan selama penyelesaian skripsi ini.
6. Elis Maulidiyah, Windatun Ni'mah, Melinda Rahman, Ainin Nur Asiyah, Siti Muntamimah, Alifatul Mufarohah, Warda Firdausi Karimah, Krisdiana Dyah Erawati, Melly Charlina, Shanti Siburian, teman-teman kos Kertopamuji 34, teman-teman Ten Sister, teman-teman Advokesma EMSK, teman-teman pengurus HIMASISKOM serta semua teman-teman seperjuangan skripsi Sistem Komputer yang selalu mendukung, memberi doa, semangat dan bantuan serta menemani penulis selama proses penyelesaian skripsi ini.

Penulis menyadari bahwa dalam penyusunan tugas akhir skripsi ini masih terdapat banyak kekurangan, sehingga kritik dan saran yang membangun sangat penulis harapkan. Akhir kata penulis berharap skripsi ini dapat bermanfaat bagi semua pihak yang menggunakannya.

Malang, 15 Agustus 2016

Penulis

Ilatul3@gmail.com

ABSTRAK

Saat ini komunikasi antar perangkat dengan konsep *pervasive* dalam sistem rumah cerdas sudah banyak diaplikasikan di dunia nyata. Konsep *pervasive* memungkinkan *user* dapat mengakses sistem dimana saja dan kapan saja saat dibutuhkan. Banyak *framework* yang mendukung konsep *pervasive*, diantaranya adalah DLNA, UPnP dan Bonjour. Namun *framework* tersebut masih belum mendukung komunikasi *multi-platform* dan integrasi perangkat masih rumit. Untuk itu dikembangkan teknologi baru bernama Alljoyn Framework. Alljoyn merupakan *framework* yang membuat aplikasi dapat mengenali perangkat terdekat meski menggunakan *platform* dan bahasa pemrograman yang berbeda.

Untuk membuktikan Alljoyn mendukung komunikasi *multi-platform*, pada penelitian ini dibuat sistem rumah cerdas yang terdiri dari Raspberry Pi sebagai *service* dan aplikasi pada *smartphone* Android sebagai klien. Sebagai *service*, dalam sistem ini Raspberry Pi menyediakan peralatan rumah cerdas yaitu LED dan sensor. Sensor yang digunakan adalah sensor yang dapat mewakili data sensor lain, yaitu sensor gerak PIR. Komponen LED dan sensor tersebut terhubung pada pin GPIO Raspberry Pi. Komponen-komponen tersebut memiliki fungsi-fungsi yang dapat digunakan oleh klien yaitu fungsi nyala, mati, kontrol PWM LED, serta deteksi sensor. Sebagai klien aplikasi merupakan *user interface* yang digunakan oleh *user* untuk menggunakan *service* pada Raspberry Pi. Dalam berkomunikasi dan mendeteksi satu sama lain *service* dan klien menggunakan salah satu fitur Alljoyn, yaitu *Advertisement* dan *Discovery*. Alljoyn Framework yang digunakan pada penelitian ini adalah versi 15.04.

Berdasarkan pengujian yang telah dilakukan dapat ditunjukkan bahwa *user* dapat melakukan kontrol terhadap perangkat-perangkat pada Raspberry Pi menggunakan aplikasi Android. Raspberry Pi juga dapat mengirimkan status keadaan perangkat kepada aplikasi. Dari hasil filter menggunakan Wireshark dapat ditunjukkan bahwa *advertisement* dan *discovery* perangkat menggunakan protokol Alljoyn Name Service.

Kata kunci: Alljoyn Framework , rumah cerdas, *pervasive*, Raspberry Pi, android, IoT

repository.ub.ac.id

ABSTRACT

Nowadays communication between devices with pervasive concept in smart home system has been widely applied in the real world. Pervasive concept allows user to access the system anywhere and anytime when needed. There are many frameworks that support the concept of pervasive, such as DLNA, UPnP and Bonjour. However, those frameworks are still not supporting multi-platform communication and integration of the device is still complicated. To overcome that problem, there is a new technology that was being developed that called Alljoyn Framework. Alljoyn is a framework that makes the app can identify nearby devices altough using different platforms and programming languages.

To prove Alljoyn can supports multi-platform communication, in this thesis, researcher builded smart home system that consists of a Raspberry Pi as a service and applications on Android smartphone as client. As a service, in this system Raspberry Pi provide smart appliances, namely LED and sensor. The sensor used is a sensor that can represent other sensor data, that is PIR motion sensor. LED components and sensors are connected to the Raspberry Pi GPIO pin. These components have functions that can be used by the client. That are On Off function, LED PWM control function and sensor detection function. As the client, application is a user interface that used by user to use the service on Raspberry Pi. In communicating and detecting each other, service and client using one of the Alljoyn feature, namely Advertisement and Discovery. Alljoyn Framework used in this thesis is Alljoyn 15:04 version.

Based on testing that has been done, it can be shown that the user can control the devices on the Raspberry Pi using the Android app. Raspberry Pi can also send state of the appliances to the application. From the results of the filter using Wireshark can be shown that the advertisement and device discovery protocol uses Alljoyn Name Service.

***Keyword**: Alljoyn Framework , smart home, pervasive, Raspberry Pi, android, IoT*

UNIVERSITAS BRAWIJAYA

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB 1 PENDAHULUAN

1.1 Latar Belakang

Dunia digital dan dunia *virtual* saat ini sudah sudah menciptakan suatu lingkungan dimana sistem kota, energi, transportasi dan sistem-sistem lain menjadi sebuah sistem cerdas. *Internet of Things* (IoT) merupakan paradigma yang menantang yang memungkinkan hal-hal seperti *embedded device* untuk selalu tersedia dimana saja dan kapan saja. IoT menawarkan solusi kebutuhan pengembangan sistem baru dengan kecerdasan pada *embedded device*. IoT memungkinkan pengembangan *user-driven applications* yang menghubungkan beberapa sensor dan obyek untuk berkomunikasi bersama melakukan suatu fungsi tertentu sesuai dengan keinginan *user*. Objek dan sensor terhubung melalui internet dengan menggunakan perangkat seperti Wi-Fi, Bluetooth, dan QR code. Saat ini, IoT sudah mencakup beberapa aplikasi dalam banyak bidang, salah satunya adalah adanya rumah cerdas (Massimo Villari, 2014).

Konsep rumah cerdas adalah konsep yang sedang ramai diperbincangkan. Dikutip dari jurnal perusahaan Lamudi tahun 2014, Allied Market Research memperkirakan bahwa pasar untuk bangunan rumah atau gedung dengan konsep cerdas telah mengalami lonjakan nilai mencapai US$ 7 Miliar. Angka tersebut diperkirakan akan terus tumbuh menjadi US$ 35.5 miliar pada tahun 2020. Salah satu yang menjadi daya tarik dari penggunaan rumah cerdas ini adalah keamanannya. Seperti diungkap pada tahun 2014 oleh *State of Smart Home*, 90% responden tertarik dengan gagasan rumah cerdas karena fitur keamanannya untuk personal dan keluarga di rumah. Rumah cerdas juga memberikan kenyamanan rumah berbasis teknologi (Lamudi, 2014).

Salah satu aplikasi trend masa kini adalah sistem rumah cerdas dengan konsep *pervasive*. Konsep *pervasive* mengacu pada konsep dimana suatu sistem komputer dapat mempengaruhi hidup, menyatu dan berinteraksi dengan lingkungannya, serta sistem dapat mengerti kebutuhan yang diperlukan dan memberikan solusi (Maryam, 2005). Menurut Webopedia.com, *pervasive computing* merupakan ide dimana semua perangkat mulai dari perangkat sederhana seperti pembuat kopi hingga perangkat dengan teknologi canggih dapat menjadi sebuah *embedded device* yang dapat saling terhubung pada jaringan yang tidak terbatas dan selalu tersedia (Ebling, 2016). Teknologi *pervasive* memungkinkan *user* dapat mengakses sistem dimana saja dan kapan saja saat dibutuhkan (*any where-any time/on demand*). Konsep *pervasive* yang diterapkan dalam rumah cerdas berarti semua peralatan rumah dapat diakses dari dan keluar rumah melalui *remote control*, jaringan telepon, SMS, web-cam dan jaringan internet (Alayderous, 2004). Rumah cerdas dengan konsep *pervasive* ini sangat sesuai dan mendukung penerapan *Internet of Things*.

Untuk membuat sebuah sistem rumah cerdas dimana komunikasi perangkatnya menggunakan konsep *pervasive*, telah banyak *framework* yang ada saat ini. Beberapa *framework* yang ada diantaranya adalah Digital Living Network

Alliance (DLNA), Universal Plug and Play (UPnP) dan Bonjour. Semua *framework* di atas menawarkan konsep komunikasi antar perangkat dengan sistem *pervasive* atau *zero configuration*. Namun *framework* di atas masih memiliki kelemahan. Kelemahan-kelemahan tersebut diantaranya adalah tidak mendukung *multi-wireless transports*, tidak mendukung *multi-platform*, manajemen jaringan yang masih cukup rumit, serta pengembangan teknologi yang lambat (Spencer, 2012).

Sebagai solusi dari permasalahan di atas, maka dikembangkanlah teknologi baru yaitu Alljoyn Framework. Alljoyn merupakan teknologi baru yang ditujukan untuk penerapan konsep IoT dan IoE. Dikutip dari jurnal berjudul AllJoyn Lambda: *an Architecture for the Management of Smart Environments in IoT*, Alljoyn merupakan *framework open source* yang dapat menjalankan aplikasi terdistribusi melalui bermacam-macam embedded *device*. Alljoyn memungkinkan *developer* dapat dengan mudah membuat aplikasi yang dapat mengenali perangkat lain yang berada di dekat perangkat tersebut. Aplikasi yang dibangun juga dapat mengkomunikasikan perangkat-perangkat secara langsung meskipun berbeda merk, protokol transport ataupun OS. Alljoyn menawarkan solusi untuk permasalahan jaringan, mobilitas, keamanan dan konfigurasi yang dinamis (Massimo Villari, 2014). Karena Alljoyn merupakan *open source* maka perkembangan teknologinya lebih cepat dari *framework* IoT yang lain.

Karena teknologi rumah cerdas semakin berkembang dan terdapat Alljoyn Framework yang menawarkan solusi untuk permasalahan *framework* IoT yang ada saat ini, peneliti menganggap perlu dilakukan penelitian untuk meneliti bagaimana Alljoyn Framework dapat menciptakan sebuah sistem rumah cerdas yang menggunakan komunikasi perangkat secara *pervasive*. Implementasi Alljoyn Framework pada sistem *pervasive* rumah cerdas dilakukan dengan memanfaatkan Raspberry Pi dan *smartphone* Android. Penelitian yang dilakukan diharapkan dapat memberi gambaran tentang penggunaan Alljoyn Framework dalam sistem *pervasive* rumah cerdas serta dapat mendorong berkembangnya teknologi rumah cerdas.

1.2 Rumusan Masalah

Berdasarkan latar belakang di atas dapat dirumuskan masalah penelitian sebagai berikut:

1. Bagaimana merancang sistem peralatan rumah cerdas yang dapat dikenali secara *pervasive*?
2. Bagaimana implementasi Alljoyn Framework pada Raspberry Pi sebagai peralatan rumah cerdas dan *smartphone* Android sebagai antarmuka *user*?
3. Bagaimana hasil pengujian fungsional terhadap keberhasilan kontrol peralatan rumah cerdas melalui aplikasi pada *smartphone* Android?

1.3 Tujuan

- Tujuan umum:

  Mengimplementasi Alljoyn Framework untuk merancang sistem peralatan rumah cerdas yang dapat dikenali secara *pervasive*.

- Tujuan khusus:

  1. Merancang sistem peralatan rumah cerdas yang dapat dikenali secara *pervasive*.
  2. Mengimplementasikan Alljoyn Framework pada Raspberry Pi sebagai peralatan rumah cerdas dan *smartphone* Android sebagai antarmuka *user*.
  3. Mengetahui hasil pengujian fungsional terhadap keberhasilan kontrol peralatan rumah cerdas melalui aplikasi pada *smartphone* Android.

1.4 Manfaat

- Bagi Masyarakat
  Dengan pengimplementasian Alljoyn Framework ini diharapkan dapat membantu pekerjaan dan mempermudah untuk pengoperasian perangkat-perangkat pada lingkungan rumah cerdas.

- Bagi Perguruan Tinggi
  Program ini merupakan perwujudan dari Tridharma Perguruan Tinggi. Yang diharapkan dapat mengembangkan ilmu pengetahuan dan teknologi yang bermanfaat bagi masyarakat.

- Bagi Mahasiswa
  Program ini diharapkan dapat meningkatkan kreatifitas mahasiswa sebagai agen perubahan dalam menerapkan ilmu pengetahuan untuk mengatasi masalah yang ada di sekitarnya.

1.5 Batasan Masalah

Batasan ruang lingkup penelitian yang dibahas sebagai berikut:

1. Peneitian berfokus pada implementasi Alljoyn Framework pada Raspberry Pi dan s*martphone* Android.
2. Penelitian berfokus pada kontrol GPIO Raspberry Pi.
3. Perangkat rumah cerdas disimulasikan menggunakan LED dan sensor.
4. Sensor yang digunakan adalah sensor PIR.
5. Penelitian ini mengabaikan aspek keamanan jaringan.
6. Penelitian ini mengabaikan aspek interaksi manusia dan komputer.
7. Penelitian ini terbatas pada implementasi Alljoyn Core Library.

1.6 Sistematika Pembahasan

Sistematika penulisan ini sebagai berikut:

BAB I : Pendahuluan
Bab ini membahas tentang latar belakang yang menjadi alasan untuk penulis menyusun topik penelitian, rumusan masalah, batasan masalah, tujuan penelitian, sistematika penulisan, dan jadwal pelaksanaan.

BAB II : Landasan Kepustakaan
Bab ini membahas mengenai teori-teori dan referensi yang digunakan dalam membantu penulisan dan menemukan jawaban atas rumusan masalah.

BAB III : Metodologi
Bab ini menguraikan tentang metode dan langkah kerja yang dilakukan dalam penulisan tugas akhir yang terdiri dari studi literatur, perancangan sistem, implementasi, pengujian serta pengambilan kesimpulan dan saran.

Bab IV : Persyaratan
Bab ini berisi detail pernyataan masalah, identifikasi sistem, serta daftar terstruktur kebutuhan perangkat secara fungsional, data, dan non-fungsional.

Bab V : Perancangan dan Implementasi
Bab ini berisi perancangan dan implementasi dari sistem rumah cerdas menggunakan Alljoyn Framework.

Bab VI : Pengujian dan Analisis
Bab ini berisi strategi dan data pengujian, ringkasan hasil pengujian serta evaluasi hasil kerja sistem secara keseluruhan.

Bab VII : Penutup
Bab ini berisi ringkasan kesimpulan dari pencapaian sistem dan saran untuk pengembangan lebih lanjut.

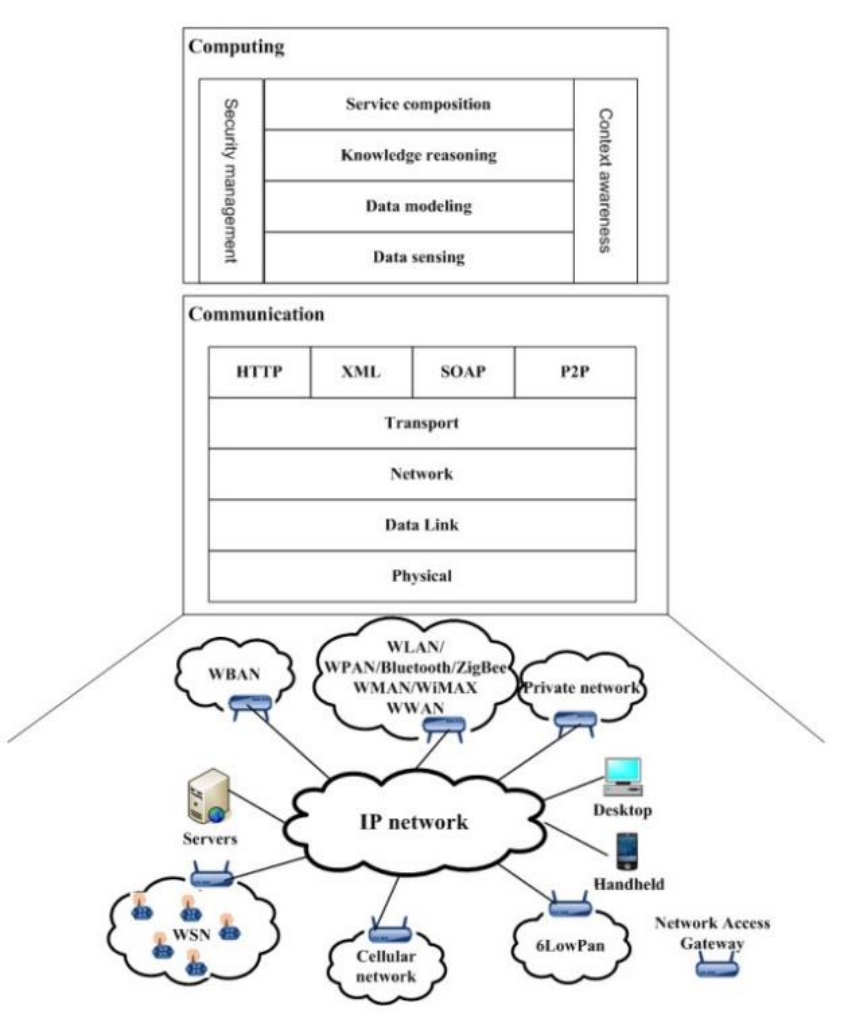
BAB 2 LANDASAN KEPUSTAKAAN

2.1 Kajian Pustaka

Dalam sebuah penelitian tugas akhir yang dilakukan oleh Fyanka Ginanjar Aditya tahun 2015 dengan judul "Analisis Dan Perancangan Prototype Rumah Cerdas Dengan Sistem *Client Server* Berbasis *Platform* Android Melalui Komunikasi *Wireless*" dirancang sebuah *prototype* dari rumah cerdas dengan sistem *client-server* berbasis Arduino Uno dengan *user interface* aplikasi Android yang akan melakukan komunikasi data melalui *wireless* (tanpa kabel). Sistem menggunakan bahasa pemrograman C dan C++ sebagai bahasa pemrograman pada sisi *server* sedangkan pada sisi *client* menggunakan bahasa pemrograman Java. *Server* menggunakan protokol Common Gateway Interface yang berfungi sebagai penghubung antara *platform* Android dengan modul Arduino Uno yang digunakan. Penelitian ini berhasil membangun keseluruhan sistem yang dapat berfungsi dan terintegrasi satu sama lain serta respon *hardware* dan *software* yang sesuai dengan *input* yang dimasukkan (Aditya, 2015).

Dalam penelitian lain oleh Tri Fajar Yurmama S dan Novi Azman tahun 2009 dengan judul "Perancangan *Software* Aplikasi *Pervasive* Rumah Cerdas" dibangun suatu sistem rumah cerdas dengan konsep cerdas dan *pervasive* yang terdiri dari perangkat kontrol, monitoring dan otomatisasi beberapa perangkat atau peralatan rumah yang saling berinteraksi dan dapat diakses melalui sebuah komputer. Monitoring dilakukan menggunakan beberapa sensor dan kamera yang dihubungkan pada komputer. Pada sistem kontrol, komputer dapat memberikan perintah langsung untuk mengaktifkan peralatan. Apabila terjadi bahaya atau kerusakan pada peralatan tersebut, maka secara otomatis komputer akan memberikan laporan kepada pemilik. Seluruh laporan tersebut akan disimpan ke dalam *database*, sehingga pemilik rumah dapat mengetahui setiap saat kejadian yang terjadi didalam rumah dan diharapkan dapat menghasilkan suatu rumah yang nyaman dan aman (Tri Fajar Yurmama S, 2009).

Dalam sebuah jurnal oleh Jiehan Zhou, Ekaterina Gilman, Mika Ylianttila, Jukka Riekki yang berjudul "*Pervasive Service Computing: Visions and Challenges*" dijelaskan bahwa layanan *pervasive* mengatur susunan dari perangkat pada layanan rumah cerdas. Karakteristik dari *pervasive service computing* adalah: pendekatan berorientasi layanan, deskripsi eksplisit aktivitas *user*, alam luas, semantik, P2P, *trust-awareness*, dan jaringan sensor nirkabel. Gambar 2.1 menjelaskan tentang arsitektur *pervasive computing* yang dibangun oleh blok-blok sistem *pervasive service computing*. Arsitekturnya terdiri dari minimal 6 komponen yaitu : 1) data *sensing* yang memberikan layanan *sensing*, 2) Data *modeling* yang mengaplikasikan model dekripsi untuk formal data 3) *knowledge reasoning* yang memproduksi data, 4) *service composition* yang mengakomodasi kolaborasi antara layanan dan koordinasi untuk memfasilitasi *user*, 5) *context-awareness* yang mengacu pada kemampuan sistem mendeteksi *physical environment* dan 6) *security management* (Jiehan Zhou, 2010).



Gambar 2.1 Arsitektur Komunikasi *Pervasive Service Computing*

Sumber : (Jiehan Zhou, 2010)

2.2 Dasar Teori

2.2.1 Rumah Cerdas

Rumah cerdas didefinisikan sebagai tempat tinggal yang dilengkapi dengan komputasi data dan teknologi informasi yang dapat merespon kebutuhan penghuni rumah, bekerja dengan mengandalkan efisiensi, otomatisasi perangkat, kenyamanan, keamanan dan penghematan. Sistem rumah cerdas adalah sistem aplikasi yang merupakan gabungan antara teknologi dan pelayanan yang dikhususkan pada lingkungan rumah dengan fungsi tertentu yang bertujuan meningkatkan efesiensi, kenyamanan dan keamanan penghuninya. Sistem rumah cerdas biasanya terdiri dari perangkat kontrol, *monitoring* dan otomatisasi beberapa perangkat atau peralatan rumah yang dapat diakses melalui sebuah komputer. Sistem rumah cerdas adalah sistem yang terdiri dari beberapa komponen pendukung yang saling berinteraksi satu sama lain. Sebuah rumah dapat dikatakan sebagai rumah cerdas apabila memiliki komponen

personal internal networking, intelligent control dan *home auotomation* (Tri Fajar Yurmama S, 2009).

2.2.2 Sistem *Pervasive*

Sistem *pervasive* dibangun dengan beberapa komponen pendukung yang harus memenuhi karakteristik dan sifat dari konsep *pervasive computing*. Karakteristik dan sifat suatu sistem dikatakan memenuhi katagori *Pervasive Computing* adalah (Tri Fajar Yurmama S, 2009):

- *Adaptability and Dynamic*
  Sifat dari sistem yang dapat beradaptasi dan dinamis terhadap lingkungan sekelilingnya.
- *Resources awareness*
  Sifat pelayanan *multi fidelity* sistem disesuaikan dengan sumber daya lingkungan sekitarnya.
- *Secure and Privacy*
  Kemampuan pengamanan dan perlakuan khusus terhadap subyek dan obyek dengan sistem pengamanan dalam pengaksesan maupun kerjanya.

Dalam sistem *pervasive computing*, "*Context awareness*" sangat memegang peranan penting bagaimana suatu sistem dapat dikatakan sebagai sistem *pervasive*. *Context awareness* ini berhubungan erat dengan model dari *user interface* yang digunakan yang meliputi *Task model*, *dialog model*, *domain model,* dan *user model*. *Task model* menerangkan bagaimana *user* memberi perintah terhadap sistem. *Dialog model* menerangkan hubungan sisi *technical* sistem, *Domain model* menerangkan aturan beberapa obyek dalam sistem dan *user model* menerangkan bagaimana aturan *user* dalam sistem (Tri Fajar Yurmama S, 2009).

2.2.3 Alljoyn Framework

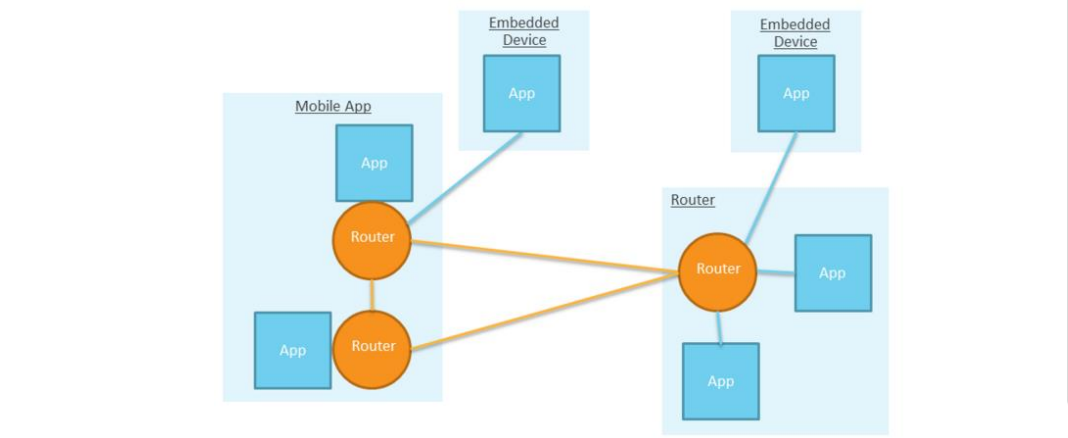
Alljoyn (AJ) merupakan *software framework* yang *open source* yang memungkinkan *developer* dapat dengan mudah membuat aplikasi yang dapat mengenali perangkat lain yang berada di dekat perangkat tersebut. Aplikasi yang dibangun juga dapat mengkomunikasikan perangkat-perangkat secara langsung meskipun berbeda merk, protokol transport ataupun OS. Alljoyn tidak membutuhkan *cloud*. Framework ini fleksibel dengan banyak fitur untuk mewujudkan konsep *Internet of Things* (Alliance, 2015).

Dalam mengenali perangkat, Alljoyn Framework membuat *session* untuk komunikasi antar perangkat. *Session* dapat berupa *multiple connection session* termasuk *point-to-point* (P2P) dan *session group*. *Framework* ini mendukung banyak mekanisme untuk keamanannya. AJ juga mendukung banyak *platform* mulai dari *platform embedded* RTOS sampai *full-featured* OS. AJ juga mendukung banyak bahasa pemrograman dan *media transport data*. Alljoyn dapat berjalan pada media Wi-Fi, ethernet, serial, dan *Power Line* (PLC).

Alljoyn Framework dapat berjalan pada jaringan lokal dan tidak membutuhkan *cloud*. *App* dan *device* dapat berkomunikasi secara langsung, cepat, dan aman. Namun pada kasus tertentu *cloud* dibutuhkan, *framework* mendukung penggunaan internet melalui Agen *Gateway*. Hanya agen *gateway* yang bisa terkoneksi dengan internet sehingga dapat mengurangi jumlah *device* yang terkoneksi dengan internet dan mengurangi kemungkinan terjadi serangan.

Alljoyn Framework terdiri dari Alljoyn Apps dan Alljoyn Routers. App berkomunikasi dengan router begitu pula sebaliknya. Namun app hanya dapat berkomunikasi dengan app lain melalui router. App dan router dapat berada pada satu *physical device* ataupun *device* yang berbeda. Dalam arsitekturnya terdapat 3 topologi dalam Alljoyn Framework yang digambarkan pada Gambar 2.2 , yaitu (Alliance, 2015) :

1. Apps menggunakan routernya sendiri. Routernya disebut *Bundled Router*
2. Beberapa app pada *device* yang sama menggunakan router yang sama. Routernya disebut dengan *Standalone Router*.
3. App menggunakan router pada *device* yang berbeda. Biasanya digunakan pada *embedded devices* yang CPU nya kecil dan memorinya berada pada router.



Gambar 2.2 Komunikasi Pada Alljoyn Framework

AllJoyn Framework memiliki dua jenis *framework* yaitu *Standard*, untuk *non-embedded devices* (seperti Android, iOS, Linux) dan *Thin*, untuk *embedded device* dengan sumber daya terbatas seperti Arduino, ThreadX, Linux dengan memori yang terbatas.

Jaringan Alljoyn terdiri dari Aplikasi dan Router. Router Alljoyn dapat berjalan sendiri sebagai *standalone router* atau terkadang terikat dengan AllJoyn Core Library. Aplikasi Alljoyn terdiri dari Alljoyn App Code, AllJoyn Service Frameworks Libraries dan AllJoyn Core Library (Alliance, 2015).

1. Alljoyn Core Library menyediakan set API level paling rendah untuk berinteraksi dengan jaringan Alljoyn. Menyediakan akses langsung untuk

advertisements, *discovery*, pembuatan *session*, konfigurasi *interface*, serta *object creation and handling*. Digunakan untuk mengimplementasikan AllJoyn Service Frameworks atau untuk mengimplementasikan *private interfaces*.

2. AllJoyn *Service* Framework Libraries mengimplementasikan layanan seperti *onboarding, notification*, atau *control panel*. Dengan ini *device* dapat saling beroperasi dengan yang lain untuk menjalankan fungsi tertentu
3. AllJoyn App Code merupakan *logic* aplikasi dari Alljoyn application. App Code dapat diprogram ke AllJoyn *Service* Frameworks Libraries, yang menyediakan fungsionalitas dengan level yang lebih tinggi, atau AllJoyn Core Library, yang menyediakan akses langsung ke AllJoyn Core APIs.

2.2.4 Software Development Kit

Sebuah *Software Development Kit* (SDK atau devkit) merupakan satu set perkakas pengembangan *software* yang digunakan untuk mengembangkan atau membuat aplikasi untuk paket *software* tertentu, *software framework*, *hardware platform*, sistem komputer, konsol *video game*, sistem operasi atau *platform* sejenis lainnya. SDK mencakup mulai dari pemrograman sederhana seperti sebuah *Application Programming Interface* (API), sampai dengan pemrograman yang lebih rumit dengan *hardware* yang canggih atau pada sistem *embedded* termasuk perangkat mobile (Shop, 2013).

Perkakas yang lazim disertakan bersama SDK, termasuk *debugging aids* yang memudahkan penelusuran kekeliruan dan utiliti lainnya yang biasa dijumpai pada lingkungan pengembang terpadu (*Integrated Development Environment*) IDE. SDK acapkali menyertakan contoh-contoh kode pemrograman dan dokumentasi teknis untuk memudahkan pembelajaran dari materi referensi utama (Shop, 2013).

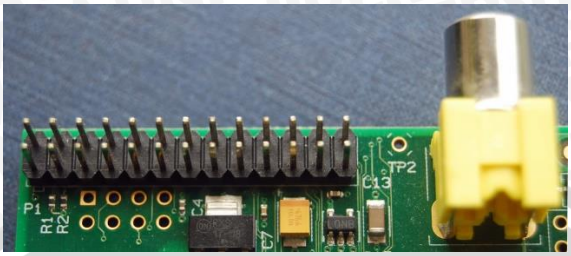
2.2.5 Android SDK

Android SDK (*Software Development Kit*) adalah tools API(*Application Programming Interface*) yang diperlukan untuk memulai pengembangan suatu aplikasi pada *platform* android menggunakan bahasa pemrograman Java. Android merupakan subset perangkat lunak untuk ponsel yang meliputi sistem operasi, *middleware* dan aplikasi kunci yang dirilis oleh Google. Saat ini tersedia Android SDK sebagai alat bantu dan API untuk mulai mengembangkan aplikasi pada *platform* android menggunakan bahasa pemrograman Java. Sebagai *platform* aplikasi netral, android memberi kesempatan untuk membuat aplikasi yang dibutuhkan yang bukan aplikasi bawaan *smartphone* (Onserda, 2013).

2.2.6 Raspberry Pi

Raspberry Pi adalah sebuah mini kit yang bisa dijadikan komputer mini seukuran kartu kredit dengan beratnya hanya 45 gram. Komputer yang diberi nama Raspberry Pi ini, berjalan dengan sistem operasi Linux. Komputer ini dikembangkan selama 6 tahun oleh lembaga non profit Raspberry Pi Foundation,

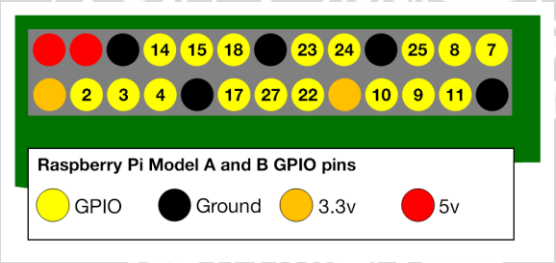
yang terdiri dari relawan dan akademisi teknologi Inggris. Untuk mengoperasikan Raspberry Pi, *user* bisa menghubungkan komputer ke monitor ataupun ke televisi, lalu mengkoneksikan keyboard dan mouse dengan Bluetooth (Ilham Megantara, 2014) .



Gambar 2.3 Letak Pin Pada Papan Raspberry Pi

Sumber: (Pi, 2015)

Salah satu fitur yang terdapat pada Raspberry Pi adalah pin GPIO (*General Purpose Input/Output*). GPIO terletak di tepi papan Raspberry Pi di samping *socket video output* seperti pada Gambar 2.3. Pin GPIO merupakan *interface* fisik antara Pi dan dunia luar. GPIO dapat diumpamakan sebagai *switches* dimana *developer* dapat menyalakan atau mematikan *input* maupun *output*. Pada Raspberry Pi terdapat 26 pin dimana 17 pin merupakan pin GPIO. Sedangkan yang lain merupakan pin daya dan *ground* (Pi, 2015). Konfigurasi pin pada Raspberry Pi terdapat pada Gambar 2.4.



Gambar 2.4 Pin GPIO Pada Raspberry Pi

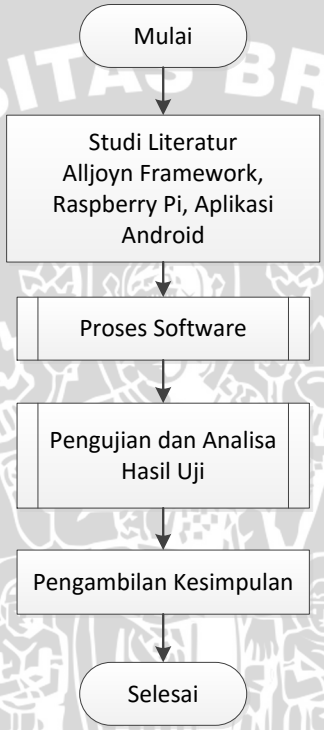
Sumber: (Pi, 2015)

Dengan GPIO, *developer* dapat memprogram pin untuk dapat berinteraksi dengan dunia luar. Input tidak harus *switch* fisik, namun dapat berupa *input* dari sensor atau sinyal dari *device* lain. *Output* juga dapat berupa apapun, mulai dari menyalakan LED hingga mengirim sinyal atau data ke *device* lain. Jika Raspberry Pi terhubung dengan jaringan, maka dapat dilakukan control terhadap perangkat yang tersambung ke Raspberry Pi. *Device* tersebut juga dapat mengirim kembali data ke Raspberry Pi. Konektivitas dan kontrol dari perangkat fisik melalui internet lebih baik dan Raspberry Pi sangat ideal untuk ini (Pi, 2015).

BAB 3 METODOLOGI

3.1 Diagram Alir Metodologi

Dalam membangun suatu sistem terdapat proses-proses yang saling berkaitan untuk menciptakan suatu perancangan yang terstruktur dengan baik. Pada bab ini akan dijelaskan tentang metodologi penelitian yang akan digunakan dalam melakukan penelitian maupun dalam penulisan skripsi. Adapun gambaran umum tahapan-tahapan metodologi penelitian dapat dilihat dari diagram alir digambarkan pada Gambar 3.1 sebagai berikut:



Gambar 3.1 Diagram Alir Langkah-Langkah Penelitian

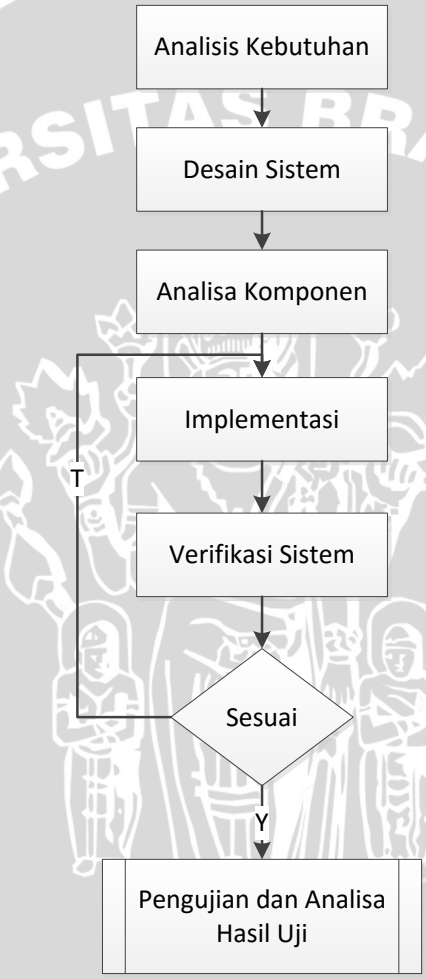
3.2 Studi Literatur

Studi literatur menjelaskan seluruh dasar teori yang mendukung dalam pengimplementasian Alljoyn Framework pada lingkungan rumah cerdas secara *pervasive*. Adapun yang dijadikan bahan dalam studi literatur adalah dasar teori pendukung untuk dapat merancang sistem tersebut meliputi :

1. Arsitektur Alljoyn Framework
2. Pemrograman Java
3. Pemrograman Bahasa C
4. Andoid SDK
5. Raspberry Pi
6. Raspbian
7. Wireshark

3.3 Proses *Software*

Pada tahap proses *software* dilakukan dengan menggunakan model Pengembangan *Reuse-Oriented*. Model Pengembangan *Reuse-Oriented* merupakan model dimana sistem diintegrasikan dari dari komponen yang ada atau sistem COTS (*Commercial-Off-The-Shelf*) (Sommerville, 2007). Model ini digunakan karena dalam membangun sistem digunakan *library-library* yang telah ada pada Alljoyn Framework. Dari *library-library* tersebut dilakukan pengembangan agar sesuai dengan tujuan pembuatan sistem. Model proses *software* adalah seperti pada Gambar 3.2.



Gambar 3.2 Model Proses *Software* Pengembangan *Reuse-Oriented*

3.3.1 Analisis Kebutuhan

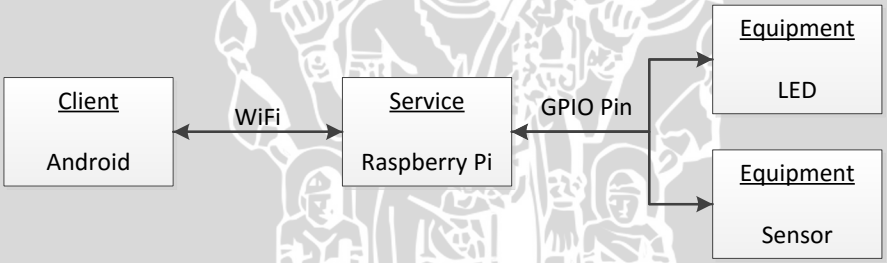
Analisis kebutuhan bertujuan untuk mendapatkan semua kebutuhan yang diperlukan oleh sistem yang akan dibangun. Pada analisis kebutuhan yang dilakukan adalah mengidentifikasi *hardware* dan *software* yang digunakan dalam perancangan, implementasi dan pengujian sistem. Dengan demikian diharapkan dapat mempermudah dalam mendesain sistem.

Analisa kebutuhan ini disesuaikan dengan kebutuhan yang digunakan, kebutuhan tersebut meliputi :

- Kebutuhan *user*
  Pengguna dapat mengontrol perangkat-perangkat pada rumah cerdas menggunakan Alljoyn Framework.
- Kebutuhan fungsional
  a. Sistem dapat menemukan dan mengenali perangkat-perangkat secara *pervasive*.
  b. Sistem dapat membuat perangkat-perangkat dapat saling berkomunikasi untuk melakukan fungsi-fungsi tertentu.
  c. Sistem dapat melakukan kontrol *input output* pada Raspberry Pi sebagai *controller* pada rumah cerdas.
  d. Sistem dapat mengirimkan informasi berdasarkan keadaan sensor.
  e. Sistem dapat berjalan pada jaringan lokal dan internet.

3.3.2 Desain Sistem

Desain sistem pada penelitian ini digambarkan dalam bentuk diagram blok dimana diagram blok tersebut memiliki bagian-bagian dari beberapa sistem yang dibuat. Diagram blok dari sistem yang akan dibuat digambarkan dalam Gambar 3.3 berikut :



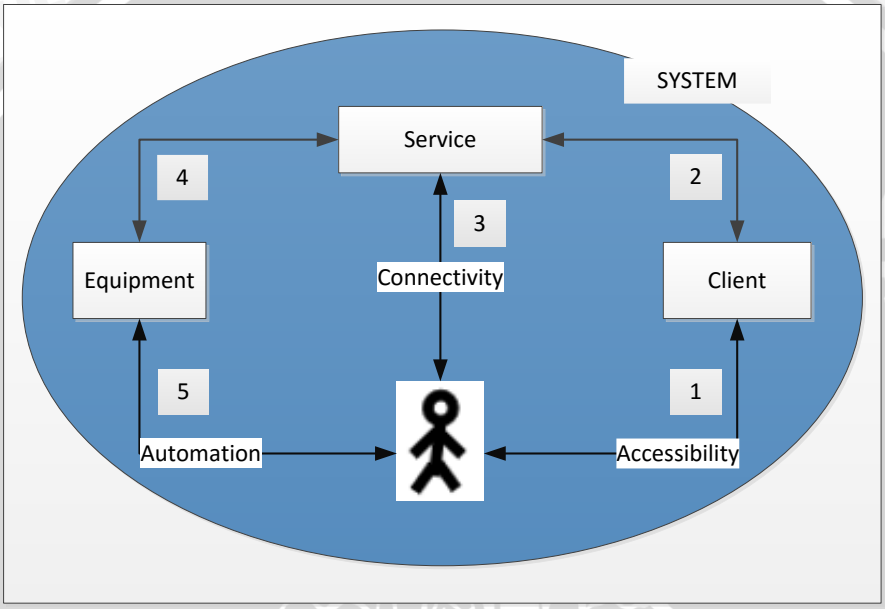
Gambar 3.3 Diagram Blok Sistem

Sistem terdiri dari :

1. *Client* berupa aplikasi Android yang berfungsi sebagai *user-interface* untuk melakukan kontrol terhadap equipment melalui Raspberry Pi. Aplikasi android ini juga bisa disebut sebagai *consumer device*. Aplikasi android dirancang menggunakan bahasa Java yang berisi notifikasi dari keadaan peralatan dan juga layanan-layanan lain untuk mengontrol peralatan.
2. *Service* berupa perangkat Raspberry Pi yang berfungsi sebagai penerima *input* dari *client* untuk selanjutnya melakukan perubahan *state* dari *equipment* yang dikontrol. Raspberry Pi berisi program yang menggunakan bahasa C++ yang di-*compile* pada sistem operasi linux. Raspberry Pi akan mengontrol *device-device* berdasarkan port-port GPIO yang ada pada Raspberry Pi.
3. *Equipment* atau peralatan berupa LED dan sensor yang diibaratkan sebagai peralatan-peralatan rumah tangga yang ada pada rumah cerdas. *Equipment*

ini nanti akan menerima perintah dari *service* Raspberry Pi untuk kemudian dijalankan. Sebagai contoh perangkat televisi yang dicontohkan menggunakan lampu LED. Jika lampu LED menyala maka diasumsikan televisi juga menyala begitu juga sebaliknya.

Sistem *pervasive* mengasumsikan *user*, perangkat dan layanan terhubung melalui satu jaringan kapanpun dan dimanapun. Hal ini memungkinkan layanan dengan struktur dan bahasa yang berbeda-beda menjadi satu kesatuan layanan komputasi *pervasive*. Layanan ini membuat jaringan dengan akses yang berbeda dapat dikonfigurasi sendiri dan dapat menyesuaikan dengan infrastruktur akses *wireless* yang berbeda-beda melalui penyedia jaringan yang dapat memenuhi kebutuhan kapasitas dan kualitas dari layanan secara efisien.



Gambar 3.4 Langkah-Langkah Komunikasi *Pervasive*

Pada Gambar 3.4 di atas digambarkan langkah-langkah komunikasi dalam suatu sistem *pervasive* dimana *user, client, service* dan *equipment* merupakan satu kesatuan sistem yang saling terhubung. Langkah-langkah komunikasi pada sistem *pervasive* berdasarkan gambar di atas adalah :

1. *User* melakukan akses ke *client application* yaitu aplikasi pada *smartphone* Android untuk mengontrol peralatan.
2. *Client application* mengirimkan pesan ke *service* yaitu Raspberry Pi.
3. *User* dapat terhubung dengan Raspberry Pi melalui *client application*.
4. *Service* memproses pesan dari *user* untuk melakukan kontrol pada *equipment*.
5. *User* dapat melakukan otomatisasi pada *equipment*.

3.3.3 Analisa Komponen

Pada tahap ini dilakukan analisa komponen–komponen apa saja yang harus dipakai untuk membangun sistem. Untuk mengimplementasikan Alljoyn Framework pada sistem rumah cerdas secara *pervasive* dibutuhkan beberapa *library* Alljoyn yang diimplementasikan pada sisi *client* dan *service*, yaitu:

1. BusAttachment Library
2. Session Library
3. Signal Library
4. Status Library

Untuk sisi *client application* dibutuhkan *library-library* untuk membuat *user interface*, diantaranya adalah :

1. Activity Library
2. Dialog Library
3. Notification Library
4. *Array*Adapter Library
5. Textview Library
6. Listview Library
7. Button Library

Sedangkan pada sisi Raspberry Pi dibutuhkan *library* untuk mengontrol GPIO. *Library* yang dapat digunakan adalah WiringPi Library.

3.3.4 Implementasi

Implementasi sistem ini akan dilakukan sesuai dengan perancangan sistem yang telah dibuat sebelumnya. Pada bagian ini terdapat beberapa proses implementasi yaitu,

1. Implementasi *hardware*
   Pemasangan LED dan sensor pada pin GPIO yang sesuai dengan perancangan.
2. Implementasi *software* pada Raspberry Pi
   Implementasi menggunakan bahasa pemrograman C++ dan menggunakan Alljoyn Framework versi 15.04. Implementasi juga menggunakan *library* wiringPi untuk GPIO Raspberry Pi.
3. Implementasi *software* pada *smartphone*
   Implementasi meliputi implementasi desain aplikasi, implementasi Alljoyn Framework, serta implementasi pengolahan data. Implementasi menggunakan IDE Android Studio dengan bahasa pemrograman Java dan XML.

3.3.5 Verifikasi Sistem

Pada tahap ini dilakukan verifikasi apakah sistem yang dibangun sudah dapat berjalan dengan baik. Semua fungsi yang ada pada kode program harus dipastikan dapat berjalan sesuai dengan tujuan fungsi tersebut. Fungsi-fungsi yang harus dilakukan verifikasi diantaranya adalah :

1. Fungsi kontrol GPIO Raspberry Pi
2. Fungsi menyimpan daftar *equipment* dan perilaku
3. Fungsi koneksi Raspberry Pi dan aplikasi
4. Fungsi kirim data antara Raspberry Pi dan aplikasi
5. Fungsi menampilkan data dari Raspberry Pi ke aplikasi

Jika fungsi-fungsi di atas sudah dapat berjalan sesuai dengan tujuan fungsi, maka dapat dilakukan tahap pengujian dan analisis.

3.4 Pengujian dan Analisis

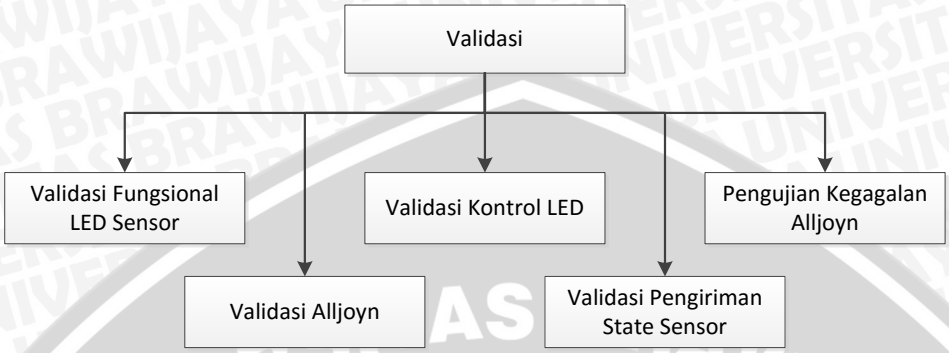
Tahap pengujian dan analisis hasil dilakukan untuk memastikan sistem dapat bekerja sesuai dengan spesifikasi kebutuhan. Tahapan proses pengujian dan analisis dijelaskan dengan Gambar 3.5. Pada tahap perencanaan dibuat skenario pengujian agar menghasilkan data hasil uji yang akurat. Pada tahap persiapan dilakukan persiapan terhadap parameter-parameter pengujian dan persiapan alat uji. Pada tahap pengujian dilakukan pengujian sesuai dengan skenario yang sudah dibuat. Pada tahap *follow up* dan analisis dilakukan perbaikan sistem jika data hasil uji tidak sesuai dengan spesifikasi kebutuhan. Namun jika sudah sesuai maka dilakukan analisis apakah sistem sudah memenuhi spesifikasi kebutuhan.

Gambar 3.5 Tahapan Proses Pengujian dan Analisis

Pengujian yang akan dilakukan disebutkan seperti pada Gambar 3.6. Pengujian-pengujian yang akan dilakukan adalah:

1. Validasi fungsional LED dan sensor
   Pengujian ini bertujuan untuk memastikan sensor Pir dan LED dapat berfungsi dengan baik.
2. Validasi Alljoyn
   Pengujian ini bertujuan untuk memastikan Alljoyn Framework berhasil diterapkan pada sistem dan dapat bekerja sesuai dengan tujuan penerapan.
3. Validasi kontrol LED
   Pengujian ini bertujuan untuk memastikan kontrol yang diberikan *user* melalui aplikasi sesuai dengan kontrol yang dilakukan oleh Raspberry Pi terhadap *equipment* yang terhubung pada GPIO
4. Validasi pengiriman *state* sensor
   Pengujian ini bertujuan untuk memastikan Raspberry Pi dapat memberikan notifikasi pada aplikasi ketika *input* sensor mengalami perubahan status.

5. Pengujian kegagalan Alljoyn
   Pengujian ini dilakukan untuk memastikan apa saja yang dapat menyebabkan Alljoyn tidak dapat berjalan dengan baik.



Gambar 3.6 Validasi Penelitian

Dari pengujian yang dilakukan, data dari tiap-tiap pengujian akan diolah sebagai perbandingan untuk memperolah hasil penentu agar bisa diterapkan dengan sebaik-baiknya. Setelah semua sistem sudah diuji maka akan dilakukan analisa untuk mengetahui hasil yang sudah dilakukan dan menarik kesimpulan dari penelitian yang sudah dilakukan. Hasil pengujian akan dianalisis untuk menentukan apakah sistem sudah berjalan sesuai yang diharapkan dan berjalan dengan baik.

3.5 Kesimpulan

Tahapan ini merupakan tahapan terakhir dari penelitian dimana dari hasil dari pengujian dan analisa data yang sudah dilakukan ditarik kesimpulan tentang bagaimana kinerja sistem yang telah dibangun atau diimplementasikan. Kesimpulan diambil berdasarkan permasalahan yang telah disebutkan. Kesimpulan menentukan apakah sistem telah mengatasi permasalahan yang telah disebutkan atau tidak.

BAB 4 PERSYARATAN

4.1 Deskripsi Umum

Bab ini bertujuan untuk menjelaskan secara rinci tentang rekayasa kebutuhan yang harus dipenuhi untuk perancangan hingga implementasi sistem. Sehingga diharapkan implementasi Alljoyn Framework pada lingkungan rumah cerdas secara *pervasive* ini dapat berjalan dengan baik.

4.1.1 Perspektif Sistem

Sistem ini dikatakan berhasil sesuai dengan tujuan penelitian apabila sistem dapat mengkomunikasikan Raspberry Pi dengan aplikasi pada *smartphone* Android menggunakan Alljoyn Framework. Sistem juga harus dapat melakukan kontrol terhadap *output* LED melalui aplikasi yang tertanam pada *smartphone* Android. Sistem juga harus dapat mengolah data sensor PIR sebagai *input* untuk kemudian dikirimkan ke *user* melalui aplikasi. Yang terakhir sistem harus dapat mengirimkan *state* atau keadaan dari peralatan-peralatan rumah cerdas yang terpasang pada Raspberry Pi kepada *user* melalui aplikasi. Program pada Raspberry Pi dan aplikasi android harus menggunakan Alljoyn Framework.

4.1.2 Ruang Lingkup

Ruang lingkup sistem menggunakan aplikasi Android sebagai sistem kontrol perangkat rumah cerdas yang ada pada Raspberry Pi. Sistem ini menggunakan LED sebagai *output* digital dan sensor PIR sebagai *input* digital. Lingkup jaringan yang digunakan dalam sistem ini adalah satu jaringan. Antara Raspberry Pi dan *smartphone* terhubung pada jaringan yang sama. Dalam implementasiannya jaringan yang digunakan adalah jaringan lokal. Sistem ini dikhususkan untuk melakukan kontrol pada GPIO Raspberry Pi. Sistem ini terbatas pada pengolahan data *output* analog dan digital serta *input* analog dan digital.

4.1.3 Karakteristik Pengguna

Sistem rumah cerdas menggunakan Alljoyn Framework ini diharapkan dapat menjadi pilihan bagi masyarakat untuk mempermudah kontrol peralatan pada rumah cerdas. *User* cukup melakukan kontrol melalui aplikasi pada *smartphone* sehingga kontrol peralatan bisa menjadi lebih efektif dan efisien. Karena Alljoyn Framework ini mendukung pengembangan sistem dengan *multi-platform* dan *multi-language* diharapkan pengimplementasian Alljoyn Framework ini dapat menjadi referensi bagi pelaku dunia IT dalam mengembangkan teknologi *Internet of Things*.

4.1.4 Lingkungan Operasi Sistem

Pada lingkungan operasi sistem persyaratan kebutuhan lingkungan yang mendukung kebutuhan sistem sebagai berikut:

1. Lingkungan yang menyediakan suatu jaringan yang digunakan sebagai media komunikasi antara Raspberry Pi dan aplikasi pada *smartphone* Android.
2. *Smartphone* yang digunakan harus *smartphone* yang menggunakan API 16 atau lebih. Aplikasi tidak dapat berjalan pada *smartphone* dengan API di bawah 16.
3. Lingkungan yang memungkinkan *user* untuk melakukan suatu gerakan sebagai *input* sensor PIR. Lingkungan yang dibutuhkan sesuai dengan jangkauan maksimal dari sensor PIR.

4.1.5 Batasan Perancangan dan Implementasi

Beberapa batasan perancangan dan implementasi yang ditetapkan sebagai berikut:

1. Sistem hanya dirancang menggunakan satu Raspberry Pi dan satu *smartphone* Android karena perancangan sistem berfokus pada komunikasi antara perangkat yang memiliki *platform* dan bahasa pemrograman yang berbeda.
2. Program pada Raspberry Pi tidak memperhatikan aspek alokasi memori.
3. Aplikasi pada *smartphone* Android tidak memperhatikan aspek interaksi manusia dan komputer.
4. Perangkat rumah cerdas diimplementasikan pada *project board*, tidak menggunakan papan PCB karena penelitian berfokus pada program dan aplikasi.

4.1.6 Asumsi dan Ketergantungan

Beberapa asumsi dan ketergantungan persyaratan sistem sebagai berikut.

1. Komponen LED dan sensor pada sistem diamsusikan sebagai peralatan-peralatan yang ada pada rumah cerdas seperti lampu, TV, AC, indikator pintu.
2. Kecepatan pengiriman data antara Raspberry Pi dan *smartphone* tergantung pada kecepatan akses pada jaringan yang digunakan. Kecepatan akses yang lambat dapat menyebabkan lambatnya pengiriman data.
3. Komponen-komponen yang terhubung dengan GPIO Raspberry Pi dapat dikontrol oleh *smartphone* jika penempatan pin GPIO sesuai dengan data setting pin GPIO pada program.

4.2 Rekayasa Kebutuhan

Pada sub bab ini menjelaskan seluruh kebutuhan agar sistem dapat bekerja sesuai dengan tujuan penulis. Pada sub bab ini menjelaskan kebutuhan antarmuka sistem, kebutuhan komunikasi, kebutuhan fungsional sistem, dan kebutuhan lainnya. Berikut adalah penjelasannya.

4.2.1 Kebutuhan Antarmuka Sistem

Antarmuka sistem ditanamkan sebagai aplikasi pada *smartphone* Android. Antarmuka ini digunakan oleh *user* untuk melakukan kontrol terhadap komponen pada Raspberry Pi. Antarmuka sistem dirancang menggunakan IDE Android Studio. Perancangannya menggunakan bahasa pemrograman Java dan XML. Antarmuka sistem menampilkan data peralatan-peralatan yang ada pada Raspberry Pi, data perilaku apa saja yang dapat dilakukan oleh peralatan-peralatan tersebut, serta *state* atau keadaan dari peralatan-peralatan tersebut. Antarmuka ini juga perlu menyediakan fitur bagi *user* untuk menghubungkan aplikasi dengan Raspberry Pi yang ada.

4.2.2 Kebutuhan Perangkat Keras

Kebutuhan perangkat keras yang dibutuhkan untuk membangun sistem ini adalah sebagai berikut :

1. PC atau Laptop untuk menjalankan program Android Studio dan melakukan perancangan pada Raspberry Pi dengan spesifikasi minimal yaitu:
   - Sistem operasi windows 10/8/7 (32 atau 64 bit)
   - RAM 2GB
   - Disk space 2 GB
   - Screen Resolution 12800 x 800
   - Memiliki Ethernet Adapter
   - Memiliki Wireless LAN Adapter
2. Raspberry Pi minimal Versi 1
3. *Smartphone* dengan sistem operasi minimal Android Jelly Bean, *support* API 16
4. Kabel ethernet atau wifi dongle.

4.2.3 Kebutuhan Perangkat Lunak

Kebutuhan perangkat lunak yang dibutuhkan untuk membangun sistem ini adalah sebagai berikut :

1. Android Studio
2. Java Developement Kit (JDK) 8
3. Alljoyn Framework untuk Android dan Linux
4. Moba Xtream atau Putty sebagai perangkat untuk melakukan *remote* pada Raspberry Pi
5. *Package* Linux untuk perancangan pada Raspberry Pi. *Package* yang dibutuhkan adalah build-essential, git, curl, libgtk2.0-dev, libssl-dev, libcap-

dev, xsltproc, libxml2-dev, lib32z1, lib32ncurses5, mono-complete, monodevelop, python, scons

6. Wireshark untuk menganalisis *traffic* jaringan antara *smartphone* dan Raspberry Pi
7. TFTPD32 atau TFTPD64 untuk *monitoring* IP jaringan virtual pada laptop.

4.2.4 Kebutuhan Komunikasi

Komunikasi antara Raspberry Pi dan *smartphone* dilakukan melalui satu jaringan yang terhubung secara *wireless*. Perintah kontrol peralatan dari aplikasi diberikan kepada Raspberry Pi untuk kemudian dilakukan perubahan *state* pada peralatan tersebut oleh Raspberry Pi.

4.2.5 Kebutuhan Fungsional

Kebutuhan fungsional adalah kebutuhan yang harus dipenuhi agar suatu sistem dapat bekerja sesuai dengan tujuan. Jika salah satu dari kebutuhan fungsional ini tidak terpenuhi maka sistem tidak berjalan dengan baik atau dapat dikatakan sistem dapat mengalami kegagalan. Berikut adalah beberapa sistem yang harus ada pada sistem:

4.2.5.1 Fungsi Kontrol GPIO Raspberry Pi

Fungsi ini mengharuskan Raspberry Pi dapat melakukan kontrol pada pin GPIO. Kontrol berupa mengatur mode pin GPIO (*output*, *input* atau PWM), memberi nilai (*write*) pada pin GPIO (1 atau 0 atau nilai PWM) serta membaca nilai (*read*) pada pin GPIO.

4.2.5.2 Fungsi Menyimpan Daftar Peralatan dan Perilaku

Fungsi ini mengharuskan Raspberry Pi dan aplikasi dapat menyimpan daftar peralatan yang terhubung pada pin GPIO serta perilaku apa saja yang dapat dilakukan atau diberikan kepada peralatan-peralatan tersebut. Daftar disimpan dalam bentuk variabel *array*.

4.2.5.3 Fungsi Koneksi Raspberry Pi dan Aplikasi

Fungsi ini mengharuskan aplikasi dapat terkoneksi dengan Raspberry Pi melalui satu jaringan. Jika aplikasi sudah dapat terkoneksi dengan Raspberry Pi maka akan membentuk sebuah *session* yang akan digunakan untuk pertukaran data antara Raspberry Pi dan aplikasi.

4.2.5.4 Fungsi Kirim Data Raspberry Pi dan Aplikasi

Fungsi ini mengharuskan Raspberry Pi dapat mengirim data ke aplikasi begitu juga sebaliknya. Fungsi ini juga mengharuskan Raspberry Pi dapat memberikan respon atas data atau pesan yang diterima dari aplikasi. Respon ini berupa data *state* dari komponen yang dikontrol oleh *user*.

4.2.5.5 Fungsi Menampilkan Data Pada Aplikasi

Fungsi ini mengharuskan aplikasi dapat menampilkan data yang diterima dari Raspberry Pi. Tampilan data disesuaikan dengan jenis data yang diterima. Data tentang daftar peralatan ditampilkan dalam bentuk *dialog list*, data tentang perilaku yang dapat dilakukan juga ditampilkan dalam bentuk *dialog list*, sedangkan data tentang *state* peralatan ditampilkan dalam bentuk dialog.

4.2.6 Kebutuhan Performansi Sistem

Sistem ini mampu bekerja dengan performa maksimal jika faktor pendukung yang ada telah terpenuhi. Faktor pendukung performansi sistem adalah kecepatan pengirim data pada jaringan yang cepat. Jika kecepatan pengirimannya lambat maka akan terjadi *delay* saat pengiriman kontrol dari aplikasi maupun respon dari Raspberry Pi. Pengiriman data *state* sensor juga akan mengalami *delay* jika faktor ini tidak terpenuhi.

BAB 5 PERANCANGAN DAN IMPLEMENTASI

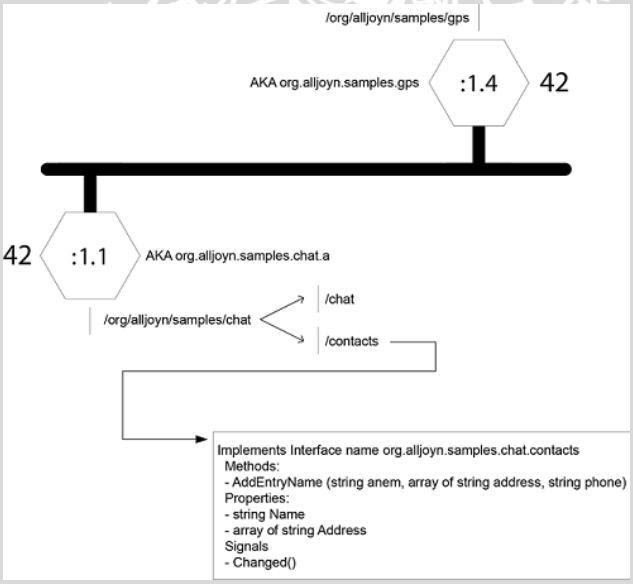
Bab ini akan membahas mengenai perancangan dari sistem yang akan dibuat dan berdasarkan hasil perancangan tersebut akan dilakukan implementasi pada perangkat keras dan perangkat lunak.

5.1 Perancangan Sistem

Tahap perancangan sistem dibagi menjadi 3 (tiga) bagian yaitu perancangan komunikasi perangkat, perancangan perangkat keras dan perancangan perangkat lunak. Perancangan sistem digunakan untuk mempermudah melakukan pengimplentasian dan juga tahap pengujian sistem.

5.1.1 Alur Kerja Sistem Alljoyn

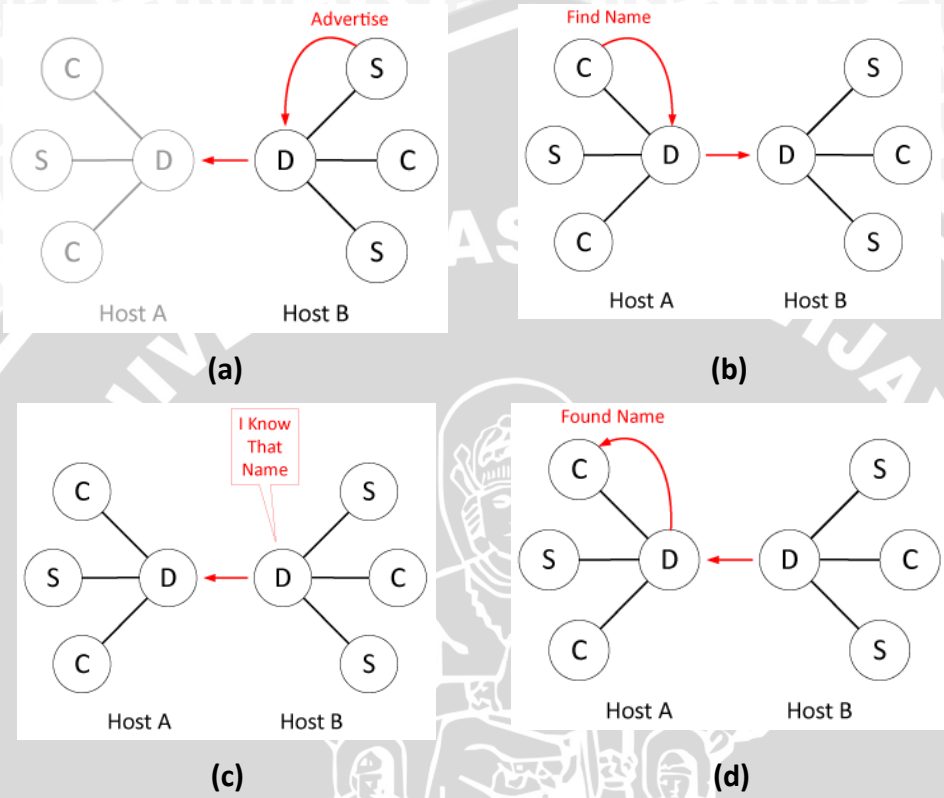
Komponen utama dalam Alljoyn Framework adalah *software bus*. *Virtual Bus* diimplementasikan dengan Alljoyn Routing Node yang merupakan *background* program yang berjalan pada tiap perangkat. *Client* dan *Service* atau peers terhubung melalui *bus attachment*. *Bus attachment* menyediakan komunikasi *interprocess* yang dibutuhkan untuk komunikasi dengan Alljoyn router lokal. Tiap *bus attachment* memiliki nama unik yang diberikan oleh sistem ketika terhubung (Alliance, 2015). Penjelasan dari penamaan bus dijelaskan pada Gambar 5.1.



Gambar 5.1 Perumpamaan Penamaan Alljoyn Bus
Sumber: (Alliance, 2015)

Garis gelap di tengah merupakan Alljoyn bus. Gambar segi enam merupakan *bus attachments* yang diberi nama unik :1.1 dan :1.4. Pada gambar *bus attachments* :1.1 telah diberi *well-known name* yaitu org.alljoyn.samples.chat.a. 'a' ditambahkan untuk memastikan nama *bus*-nya unik. Nama ini diberikan karena struktur *path object* yang ada adalah /org/alljoyn/samples/chat dimana *path* ini memiliki sub *chat* dan *contacts*. /org/alljoyn/samples/chat/contacts

mengimplementasikan *interface* dengan nama org.alljoyn.samples.chat.contacts begitu juga sebaliknya dengan *chat*. Karena *bus* ini memiliki interface maka *bus* ini harus mengimplementasikan *bus method*, *bus signal* dan *bus properties*. Angka 42 merupakan *port session* yang digunakan untuk menginisialisasi komunikasi dengan *service* (Alliance, 2015).



Gambar 5.2 *Advertisement and Discovery Alljoyn Service*
Sumber: (Alliance, 2015)

Setelah *bus* memiliki *well-known name*, *service* meng-*advertise* agar *client* dapat men-*discover* layanannya. Gambar 5.2 menunjukkan proses *advertisement and discovery* pada Alljoyn. Gambar 5.2 (a) menunjukkan bagaimana *service* melakukan *request advertise* ke router lokal. Router yang akan menentukan media apa yang digunakan untuk terhubung dengan jaringan. Ketika *client* ingin menemukan *service*, maka ia melakukan *request find name* pada router lokal seperti pada Gambar 5.2 (b). Ketika kedua perangkat berdekatan, maka perangkat-perangkat ini saling melakukan *advertisement* dan *discovery* seperti pada Gambar 5.2 (c). Melalui proses ini maka *client* akan menerima layanan dan dapat menggunakannya seperti pada Gambar 5.2 (d).

Client dan *service* pada Alljoyn menggunakan *methods* dan *callback*. *Service* menggunakan *bus* untuk menyediakan layanannya, *client* menggunakan *proxy* untuk menyediakan *interface* yang mudah digunakan untuk berkomunikasi dengan *service*. *Proxy* yang digunakan adalah AllJoyn ProxyBusObject yang menyedakan fungsi *marshaling* dan *unmarshaling* (Alliance, 2015).

Sebelum fungsi dipanggil, *session* harus terbentuk untuk memisahkan *bus segment*. *Advertisement and discovery* berbeda dengan pembuatan *session*. Satu dapat menerima *advertisement* dan tidak melakukan apapun, Hanya ketika *advertisement* diterima, *client* memutuskan untuk membuat *session* atau tidak. Untuk mencapai hal ini, *service* harus membuat *session endpoint* dan meng-*advertise* keberadaannya, dan *client* harus menerima dan bergabung dengan *session* tersebut. Abstrak yang akan dibentuk adalah seperti berikut :

```
{reliable IP messages, org.alljoyn.samples.chat.a, 42}
```

Abstrak di atas mengindikasikan bahwa pesan *transport* yang digunakan adalah *reliable* dengan *well known name* seperti di atas dan *session port* 42. Diasumsikan bahwa terdapat *bus attachment* dengan nama unik 2.1 ingin terhubung dari *routing node*, maka akan membentuk *session baru* dengan abstrak seperti di bawah ini :

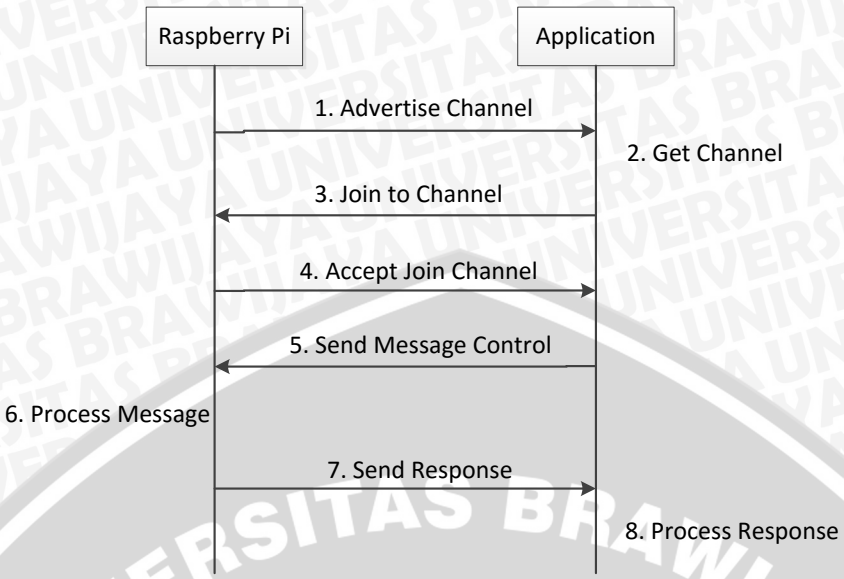
```
{reliable IP messages, org.alljoyn.samples.chat.a, :2.1,
1025}
```

Session baru menggunakan protokol pesan *reliable* dengan *bus attachment* di atas dan *bus attachment client* adalah :2.1. *Session ID* yang digunakan adalah 1025. Hasilnya, terjadi komunikasi *end-to-end* antar *client* dan *service* menggunakan Alljoyn (Alliance, 2015).

5.1.2 Perancangan Komunikasi Perangkat

Dalam sistem rumah cerdas secara *pervasive* yang akan dibangun terdapat komunikasi yang terjadi antara Raspberry Pi dan aplikasi pada *smartphone* Android. *User* harus dapat melakukan kontrol terhadap peralatan-peralatan yang terhubung dengan GPIO Raspberry Pi melalui aplikasi. Raspberry Pi juga perlu memberikan informasi kepada *user* tentang keadaan dari peralatan-peralatan yang terhubung. Agar persyaratan di atas dapat terpenuhi maka dibutuhkan komunikasi atau pengiriman dan penerimaan data antara Raspberry Pi dan *smartphone* Android.

Model komunikasi data pada sistem ini dijelaskan pada Gambar 5.3. Raspberry Pi yang bertindak sebagai *server* menjalankan *service* Alljoyn terlebih dahulu. Salah satu fitur yang ada pada Alljoyn adalah adanya *channel*. *Channel* ini yang akan digunakan sebagai 'ruang' untuk komunikasi antara Raspberry Pi dan aplikasi. Raspberry Pi sebagai *server* meng-*advertise channel* yang dibentuk kepada jaringan. *Smartphone* harus berada pada jaringan yang sama dengan Raspberry Pi. Kemudian aplikasi akan mendapatkan *list channel* Alljoyn yang ada pada jaringan. Selanjutnya aplikasi memilih *channel* yang telah di-*advertise* oleh Raspberry Pi dengan mengirim pesan *request* untuk *join channel*. Jika aplikasi dan program pada Raspberry Pi berada pada *session port* yang sama, maka permintaan *join channel* akan diterima dan program memberi id dan *name* kepada aplikasi sebagai alamat. Setelah aplikasi mendapatkan id dan *name*, aplikasi mengirimkan pesan ACK "*Joined*" yang menyatakan bahwa dia telah terhubung.



Gambar 5.3 Model Komunikasi Sistem

Pesan ACK "*Joined*" ini yang akan menginisialisasi program pada Raspberry Pi untuk mengirimkan *list device*, *list action* dan *list state* kepada aplikasi. Data ini dikirimkan dalam bentuk pesan-pesan yang memiliki *prefix* tertentu. Data ini juga dikirimkan secara berurutan sesuai dengan apa yang dituliskan dalam kode program Raspberry Pi. Setelah data *list* diterima oleh aplikasi, selanjutnya *user* dapat melakukan kontrol terhadap peralatan-peralatan yang ada. Kontrol dilakukan dengan mengirimkan pesan kontrol. Kemudian Raspberry Pi akan memproses pesan tersebut dengan melakukan kontrol terhadap peralatan. Setelah pesan selesai diproses maka Raspberry Pi memberikan respon berupa informasi perubahan *state* pada aplikasi. Selanjutnya aplikasi memproses pesan respon dari Raspberry Pi, apakah ditampilkan dalam bentuk *notification dialog* atau hanya ditampilkan pada *action list dialog*.

Tabel 5.1 Daftar Jenis *Prefix* Pesan

Prefix Pesan	Nama Prefix	Kegunaan
D-	*Device*	Pesan tentang daftar peralatan yang dapat dikontrol
A-	*Action*	Pesan tentang daftar kontrol yang dapat dilakukan pada peralatan
S-	*State*	Pesan tentang keadaan peralatan
E-	*Event*	Pesan ketika terjadi perubahan *state* peralatan karena perubahan sensor *input*
C-	*Control*	Pesan tentang kontrol yang dilakukan oleh *user*

Pesan yang dikirim dan diterima oleh Raspberry Pi dan aplikasi Android memiliki tujuan-tujuan yang berbeda. Seperti pesan untuk melakukan kontrol peralatan, pesan untuk memberikan informasi daftar peralatan, pesan untuk memberikan informasi tentang *state* peralatan dan lain-lain. Untuk itu perlu diberikan *prefix* untuk menentukan jenis pesan. Daftar *prefix* pesan yang digunakan dalam sistem dijelaskan dalam Tabel 5.1.

Raspberry Pi yang bertindak sebagai *server* atau pelayan mengirim semua jenis pesan dengan *prefix-prefix* di atas kecuali pesan *control*. Sedangkan aplikasi yang bertindak sebagai *client* atau *channel joiner* hanya mengirim pesan jenis *control* kepada Raspberry Pi. Hal ini dilakukan karena aplikasi hanya dapat melakukan kontrol, tidak dapat menambah peralatan atau menambah daftar *action* yang dapat dilakukan terhadap peralatan. Ini menyebabkan Raspberry Pi tidak akan menerima pesan jenis *device, action, state,* dan *event*.

Tiap-tiap jenis pesan diproses dengan cara yang berbeda oleh aplikasi. Pesan *device* diproses dengan menampilkan pesan-pesan tersebut pada *device list* pada program. Pesan *action* ditampilkan dalam bentuk *text list* pada kotak dialog *action*. Kotak dialog *action* sendiri muncul ketika *user* melakukan klik terhadap salah satu peralatan pada *device list*. Pesan *state* ditampilkan dalam bentuk *text* pada kotak dialog *action*. Sehingga sebelum melakukan kontrol *user* sudah mengetahui *state* dari peralatan yang akan dikontrol. Pesan *event* ditampilkan dalam bentuk dialog notifikasi. Pesan *control* hanya dikirimkan oleh aplikasi dan hanya diterima oleh Raspberry Pi. Sehingga tidak diperlukan pengaturan tampilan terhadap pesan ini pada aplikasi. Pada Raspberry Pi, pesan *control* ini diproses dengan melakukan kontrol terhadap peralatan sesuai dengan pesan yang diterima.

Terdapat beberapa peralatan yang terhubung pada GPIO Raspberry Pi. Sehingga banyak juga peralatan yang dapat dikontrol oleh *user* melalui aplikasi. *Device* yang satu dengan *device* yang lain memiliki a*ction list* yang berbeda. Hal ini mengharuskan sistem untuk membedakan pesan untuk tiap peralatan agar Raspberry Pi mengetahui peralatan mana yang harus diaktifkan atau dinonaktifkan serta aplikasi dapat membedakan *action-action* dari masing-masing peralatan. Misalkan pesan *action list* untuk peralatan A berbeda dengan pesan *action list* untuk peralatan B. Pada perancangan sistem ini yang digunakan untuk membedakan pesan masing-masing peralatan adalah id *device*. Peralatan memiliki id masing-masing dimana id ditentukan oleh Raspberry Pi pada program. Id *device* pada Raspberry Pi sama dengan id *device* pada aplikasi.

Gambar 5.4 Format Pesan yang Dikirim dan Diterima

Dengan adanya *prefix* dan id *device* maka pesan yang dikirim dan diterima oleh Raspberry Pi dan aplikasi memiliki format seperti Gambar 5.4 . Pesan yang dikirim berbentuk string. Dua karakter pertama adalah *prefix* pesan, karakter ketiga adalah id *device*, sedangkan karakter keempat dan seterusnya adalah isi

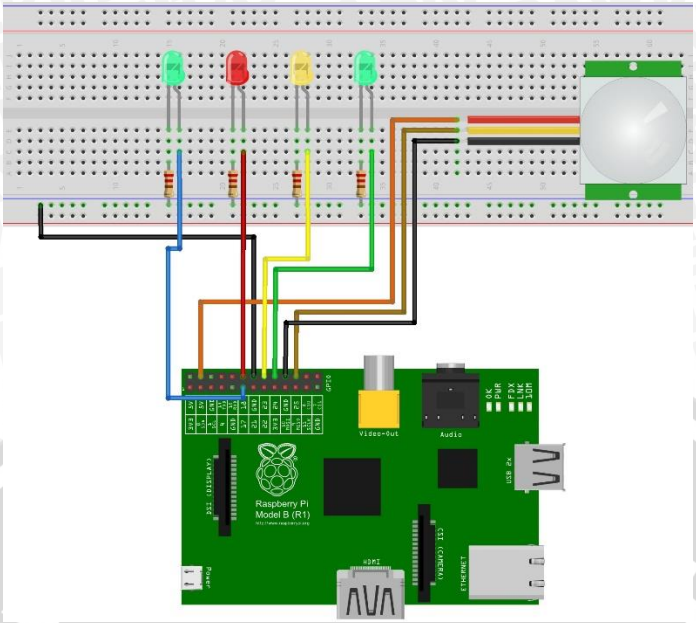
dari pesan yang ingin dikirimkan. Misalkan ingin memberikan informasi mengenai peralatan maka pesan yang dikirimkan adalah "D-1LAMPU KAMAR TIDUR". Misalkan Lampu Kamar Tidur dapat dinyalakan dan dimatikan maka pesan *action*-nya berupa "A-1Matikan" dan "A-1Nyalakan". Sedangkan contoh untuk pesan *state*-nya adalah "S-1MATI". Misal *user* ingin melakukan kontrol untuk mematikan lampu maka pesan *control*-nya adalah C-1-1. Detail daftar *device, action, state* dan *control* dalam sistem dijelaskan pada Tabel 5.2.

Tabel 5.2 Daftar *Device, Action, State* dan *Control* Sistem

Id *Device*	Nama *Device*	*Action*	State	Control
1	TV	- Matikan - Nyalakan	- Mati - Nyala	- 1 - 2
2	Lampu Kamar Tidur	- Matikan - Redupkan - Remangkan - Terangkan	- Mati - Nyala Redup - Nyala Remang - Nyala Terang	- 1 - 2 - 3 - 4
3	AC	- Matikan - Nyalakan	- Mati - Nyala	- 1 - 2
4	Pintu	- Tutup - Buka	- Terbuka - Tertutup	- 1 - 2

5.1.3 Perancangan Perangkat Keras

Perangkat keras yang digunakan pada sistem ini adalah Raspberry Pi sebagai *server*, sensor PIR sebagai *input* serta LED sebagai *output*. Sensor PIR dan LED dihubungkan dengan Raspberry Pi melalui GPIO (*General Pin Input Output*). Terdapat dua jenis penomoran pin GPIO, yaitu penomoran pin berdasarkan *chip broadcom* dan penomoran pin berdasarkan papan fisik. Untuk penulisan program penomoran yang dimasukkan adalah penomoran berdasarkan *chip broadcom*. Empat LED *output* yang memiliki dua kaki terhubung dengan pin data GPIO pada kaki positif dan pin *ground* pada kaki negatif. Antara pin *ground* dengan kaki negatif LED terdapat resistor sebesar 220 Ohm sebagai penghambat. Sensor dan LED terhubung dengan pin *ground* GPIO. Sensor PIR mendapatkan daya 5v melalui pin *power* DC GPIO. Skematik diagram *hardware* digambarkan dengan Gambar 5.5.



Gambar 5.5 Skematik Perangkat Keras Sistem

LED terdiri dari LED 1 *output* berwarna hijau sebagai representasi perangkat televisi, LED PWM *output* berwarna merah sebagai representasi Lampu Kamar Tidur, LED 2 *output* berwarna kuning sebagai representasi perangkat Air Conditioner, serta LED sensor PIR berwarna hijau sebagai indikator untuk sensor PIR dan representasi pintu rumah. Masing-masing LED dan sensor terhubung pada pin yang berbeda. Detail peletakkan komponen pada pin GPIO Raspberry Pi dijelaskan pada Tabel 5.3.

Tabel 5.3 Konfigurasi Pin GPIO Raspberry Pi

No Pin GPIO	Nama Pin	Kegunaan
2	5v DC *Power*	*Power* Sensor
6	*Ground*	*Ground* LED
11	GPIO17	LED 1 *Output* (hijau)
12	GPIO18	LED PWM *Output* (merah)
16	GPIO23	LED 2 *Output* (kuning)
18	GPIO24	LED Sensor Pir (hijau)
20	*Ground*	*Ground* Sensor Pir
22	GPIO25	Sensor PIR

Pin *ground* untuk LED berbeda dengan pin *ground* untuk sensor PIR. Hal ini dikarenakan LED terpasang pada *project board*, sedangkan sensor PIR tidak terpasang pada *project board*. Pada GPIO Raspberry Pi terdapat dua jenis penomoran pin yaitu penomoran pin secara fisik dan penomoran pin

berdasarkan *chip broadcom*. Pada Tabel 5.3 kolom pertama adalah penomoran pin secara fisik, yaitu penomoran pin pada *hardware* Raspberry Pi. Sedangkan kolom kedua adalah kolom nama pin dan penomoran pin berdasarkan *chip broadcom*. Nomor pin ini yang dimasukkan ke dalam kode program. Misalkan untuk konfigurasi LED 1 *output*, pada *hardware* LED dihubungkan dengan pin nomor 11, namun pada program dikonfigurasi dengan nomor pin 17. Untuk pin *output* PWM hanya dapat menggunakan pin nomor 12 untuk pin fisik atau pin 18 untuk pin program.

5.1.4 Perancangan Perangkat Lunak

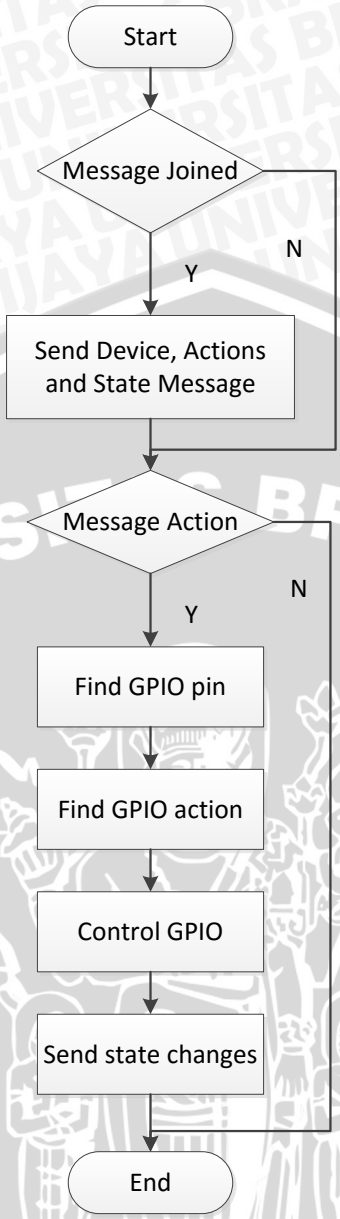
Perangkat lunak terdiri dari program pada Raspberry Pi dan program pada aplikasi pada *smartphone* Android. Program pada Raspberry Pi terdiri dari program untuk *handling message* dan program pembacaan sensor. Sedangkan program pada aplikasi terdiri dari program *handling message.*

5.1.4.1 Perancangan Program *Handle Message* Raspberry Pi

Inti dari sistem yang akan dibangun adalah komunikasi atau pertukaran pesan antara Raspberry Pi dan aplikasi pada *smartphone* Android. Terdapat beberapa jenis pesan yang dipertukarkan dengan masing-masing pesan memiliki *prefix* yang berbeda. Tiap jenis pesan diproses dengan cara yang berbeda. Untuk itu program harus dirancang untuk menangani masing-masing jenis pesan dengan perlakuannya yang berbeda. Alur perancangan penanganan pesan pada Raspberry Pi dijelaskan dengan Gambar 5.6.

Pesan "*Joined*" merupakan pesan yang menginformasikan bahwa aplikasi sudah bergabung dengan *channel* Raspberry Pi. Untuk itu Raspberry Pi perlu menginformasikan tentang daftar peralatan yang dapat dikontrol oleh *user* atau peralatan yang terhubung dengan Raspberry Pi beserta dengan aktivitas atau *action* yang dapat dilakukan terhadap peralatan tersebut. *State* peralatan juga harus dikirimkan agar *user* dapat mengetahui keadaan peralatan. Jadi ketika Raspberry Pi menerima pesan berisi "*Joined*" maka Raspberry Pi mengirimkan pesan *device, actions,* dan *state* kepada aplikasi.

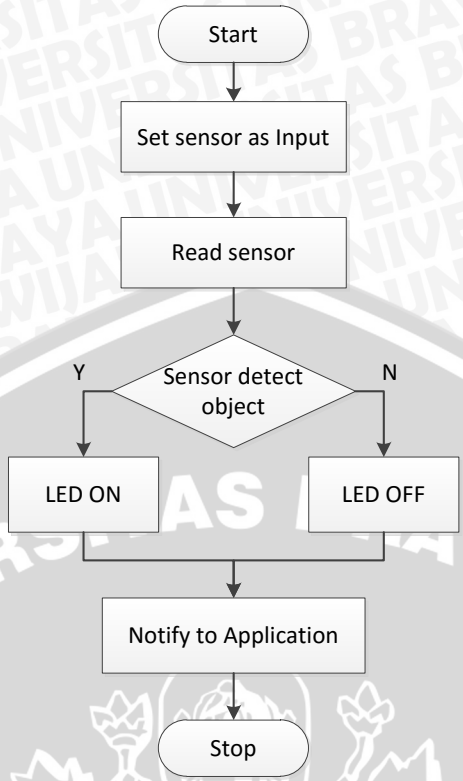
Pesan *action* merupakan pesan *request* dari *user* untuk melakukan kontrol terhadap peralatan. Ketika pesan ini diterima, maka pesan menentukan pin GPIO yang akan dikontrol dengan melihat id *device* pada pesan. Selanjutnya sistem menentukan jenis *action* yang akan dilakukan (menyalakan, mematikan atau mengubah nilai PWM) dengan melihat isi pesan *action*. Setelah ditentukan pin GPIO dan jenis *action*-nya, sistem melakukan proses kontrol terhadap peralatan. Kontrol ini menyebabkan perubahan keadaan dari peralatan (0 menjadi 1 atau mati menjadi nyala). Untuk itu sistem juga perlu menginformasikan kepada aplikasi akan terjadinya perubahan *state* peralatan dengan mengirimkan pesan *state*.



Gambar 5.6 Alur Penanganan Pesan Pada Raspberry Pi

5.1.4.2 Perancangan Program Sensor Pir Pada Raspberry Pi

Sensor PIR terhubung dengan pin GPIO Raspberry Pi. Untuk itu Raspberry Pi perlu mengkonfigurasi sensor ini agar dapat berfungsi sebagai *input*. RPI juga perlu mengkonfigurasi apa saja yang dapat dilakukan dan dihasilkan oleh sensor PIR. Alur kerja konfigurasi sensor PIR pada Raspberry Pi dijelaskan dengan Gambar 5.7.



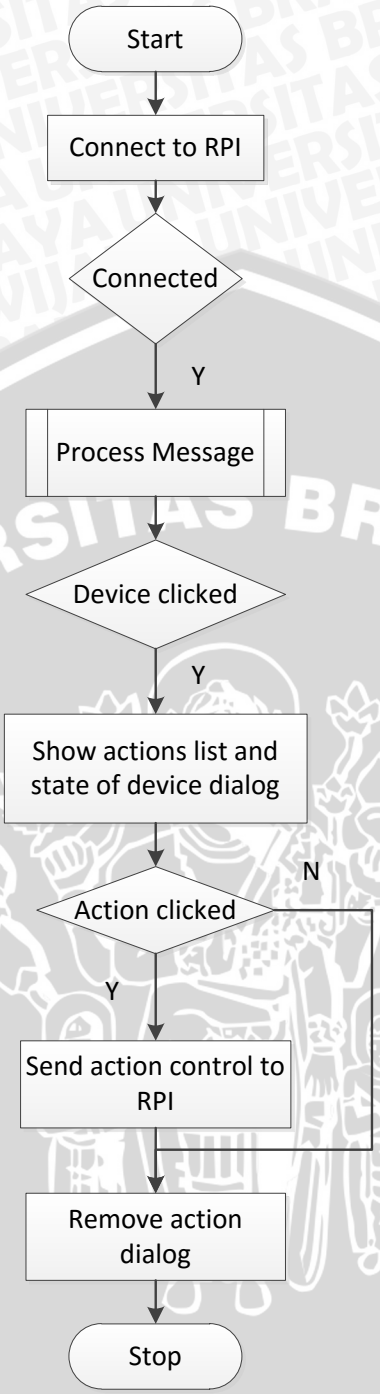
Gambar 5.7 Alur Konfigurasi Sensor PIR Pada Raspberry Pi

Program untuk sensor PIR dimulai dengan menkonfigurasi pin sensor sebagai pin *input* GPIO. Selanjutnya program membaca data hasil *sensing* sensor PIR. Jika sensor mendeteksi obyek atau sensor bernilai 1 atau *HIGH* maka program mengecek *state* dari LED indikator sensor. Jika LED telah menyala maka program tidak melakukan apa-apa. Namun jika LED belum menyala maka program mengaktifkan LED indikator sensor kemudian mengirimkan informasi perubahan *state* LED sensor kepada aplikasi. Jika sensor tidak mendeteksi adanya obyek maka program menonaktifkan LED sensor dan memberi informasi kepada aplikasi tentang *state* LED sensor yang nonaktif. Proses pengecekkan hasil *sensing* data dilakukan secara terus menerus dengan *delay* yang sesuai.

5.1.4.3 Perancangan Program pada Aplikasi Android

Aplikasi pada *smartphone* Android merupakan *interface* yang digunakan oleh *user* untuk melakukan kontrol terhadap peralatan yang terhubung dengan Raspberry Pi. Aplikasi ini juga menerapkan Alljoyn Framework untuk berkomunikasi dengan RPI. Alur kerja aplikasi pada *smartphone* dijelaskan dengan Gambar 5.8.

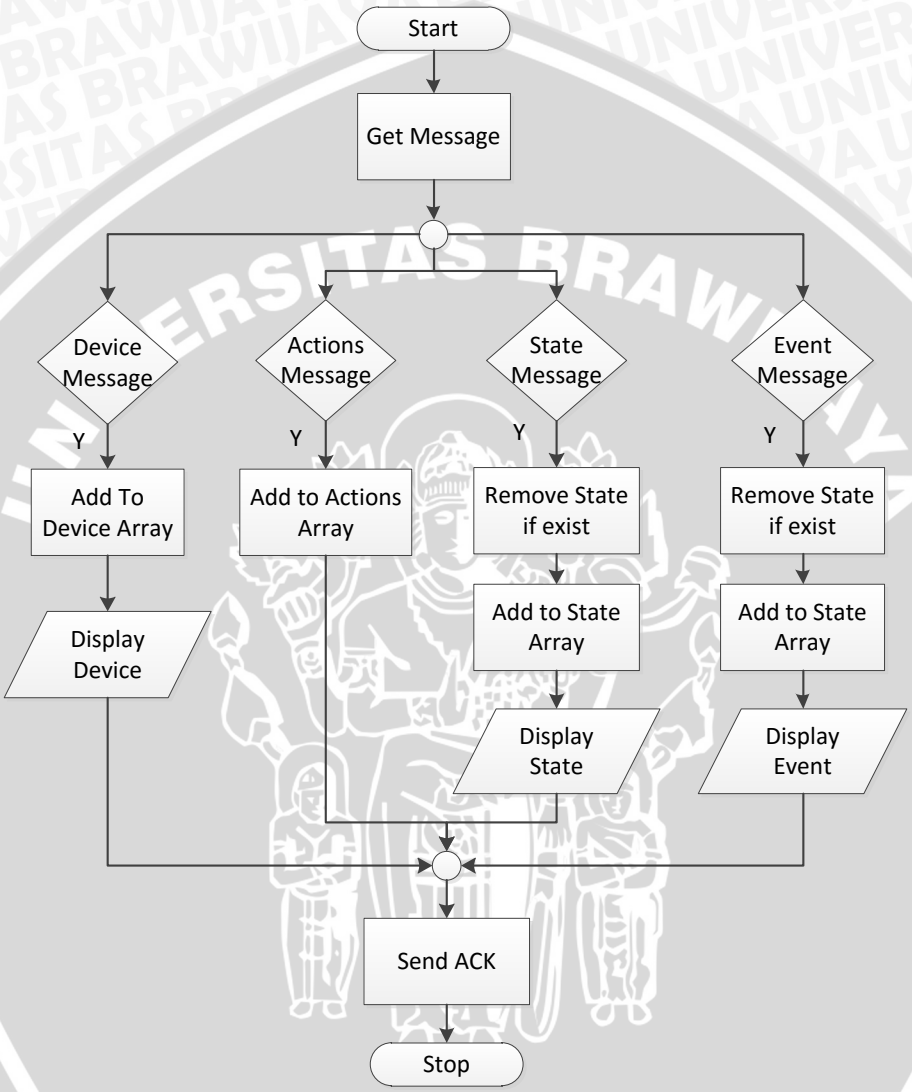
Proses komunikasi antara aplikasi dan RPI dimulai dengan menghubungkan kedua perangkat ini melalui jaringan yang sama. Aplikasi mengirimkan *request name* untuk *join* pada *channel* RPI. Jika proses *join* berhasil maka akan terbentuk *session* dan aplikasi akan memiliki *name* tersendiri.



Gambar 5.8 Alur Kerja Aplikasi Pada *Smartphone* Andorid

Selanjutnya aplikasi akan mengirimkan pesan *"joined"* untuk memberitahukan kepada RPI bahwa aplikasi telah terhubung agar selanjutnya RPI memberikan informasi-informasi tentang peralatan yang ada pada Raspberry Pi. Pesan-pesan yang diterima dari RPI diolah sesuai dengan jenis pesan yang diterima. Salah satunya adalah pesan *device* yang ditampilkan dalam bentuk *list device*. *User* dapat memilih peralatan mana yang akan dikontrol dengan melakukan klik terhadap salah satu peralatan. Jika salah satu peralatan telah di klik maka aplikasi akan menampilkan daftar *action* apa saja yang dapat dilakukan

oleh *user* serta *state* peralatan tersebut. Daftar *action* dan *state* peralatan ditampilkan dalam bentuk kotak dialog. Kemudian *user* dapat memilih *action* dengan cara mengklik salah satu *action* dalam kotak dialog tersebut. Jika *action* telah dipilih maka aplikasi mengirimkan pesan *control* kepada RPI. Namun jika *user* tidak ingin melakukan kontrol terhadap peralatan maka *user* dapat me-*remove* kotak dialog tersebut.



Gambar 5.9 Alur Penanganan Pesan Pada Aplikasi *Smartphone* Android

Sama seperti Raspberry Pi, aplikasi juga memperlakukan tiap jenis pesan dengan perlakuan yang berbeda. Alur aplikasi penangan pesan pada aplikasi dijelaskan dengan Gambar 5.9. Semua pesan kecuali pesan *event* dimasukkan ke dalam *array*. Hal ini karena informasi dalam pesan tersebut dibutuhkan oleh *user* berulangkali. Untuk itu perlu disimpan ke dalam *array* agar aplikasi tidak perlu meminta informasi kepada Raspberry tiap kali informasi tersebut dibutuhkan. Pesan *device* ditampilkan pada bagian depan dalam bentuk *device list*. *Device list* ini nanti dapat dipilih oleh *user*. Pesan *action* hanya dimasukkan ke dalam *array*. Pesan ini ditampilkan ketika *user* mengklik salah satu peralatan. Pesan ini

ditampilkan dalam kotak *dialog actions*. Ketika aplikasi menerima pesan *state*, maka aplikasi mengecek peralatan dari *state* tersebut melalui id_*device* pada pesan. Jika peralatan telah memiliki *state* yang lain maka *state* sebelumnya dihapus dan diganti dengan *state* baru yang dikirimkan oleh Raspberry Pi. Proses pergantian *state* dilakukan dengan menghapus *state* lama dari *array* dan menambah *state* baru. Namun jika belum ada data *state* dengan id *device* yang sama maka aplikasi menyimpan pesan *state* tersebut dalam bentuk *array*. *State* peralatan ditampilkan dalam bentuk *Toast* Adroid. Proses penanganan pesan *event* sama dengan pesan *state* yaitu dicek terlebih dahulu apakah peralatan pada pesan *event* sudah memiliki *state* yang tersimpan dalam *array*. Jika iya maka *state* tersebut dihapus dan diganti dengan yang baru. Jika tidak maka *state* disimpan dalam bentuk *array*. Selanjutnya *event* ditampilkan dalam bentuk kotak dialog *event notification*. Setelah aplikasi menerima pesan dan memproses pesan, selanjutnya aplikasi mengirim pesan ACK kepada Raspberry Pi yang menyatakan bahwa pesan telah diterima oleh aplikasi.

5.2 Implementasi Sistem

Implementasi sistem merupakan tahapan lanjutan setelah perancangan dilakukan. Implementasi sistem berarti menerapkan sistem sesuai dengan perancangan yang dibuat. Implementasi pada sistem ini diterapkan pada sisi *hardware* dan *software*. Implementasi *software* ada pada sisi *software* Raspberry Pi dan *software smartphone* Android.

5.2.1 Implementasi *Hardware*

Hardware yang digunakan pada sistem ini meliputi Raspberry Pi 1b, LED Merah Kuning Hijau, Resistor, Sensor Pir dan *smartphone*. Komponen LED dan resistor diletakkan pada sebuah *project board*. Gambar 5.10 merupakan gambar Raspberry Pi yang digunakan dalam sistem ini.

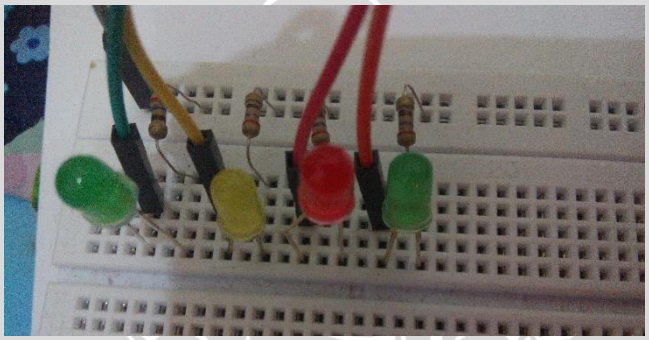
Gambar 5.10 Raspberry Pi 1B

Tabel 5.4 menjelaskan tentang spesifikasi dari Raspberry Pi 1B yang berperan sebagai *controller* sekaligus Alljoyn *service*.

Tabel 5.4 Spesifikasi Raspberry Pi 1B

1.	512 MB of RAM
2.	26 GPIO Pin
3.	100mb Ethernet *port*
4.	SD Card 12GB

Pin GPIO berjumlah 26 pin digunakan sebagai penghubung dengan komponen *hardware* yang lain (LED dan sensor PIR). *Port* ethernet digunakan sebagai media komunikasi dengan laptop agar dapat dilakukan *remote* Raspberry Pi pada laptop. Spesifikasi RAM tidak terlalu besar karena untuk menjalankan program tidak memerlukan memory yang terlalu banyak. Untuk dapat menggunakan Raspberry Pi diperlukan *SD Card*. *SD Card* yang digunakan pada sistem ini memiliki kapasitas sebesar 12 GB. Kapasitas ini cukup untuk penyimpanan sistem operasi dan program r*umah cerdas* Alljoyn.



Gambar 5.11 LED dan Resistor

LED yang digunakan pada sistem ini berjumlah empat dengan dua LED berwarna hijau, satu berwarna kuning, dan satu berwarna merah. Masing-masing LED terhubung dengan resistor sebagai hambatan tegangan. Gambar 5.11 merupakan gambar LED dan resistor yang sudah disusun sesuai perancangan sistem.

Tabel 5.5 Spesifikasi LED dan Resistor

1.	2V *power*
2.	20 mA *current*
3.	530 nm LED diameter
4.	220 Ohm resistor

Spesifikasi LED dan resistor ada pada Tabel 5.5. LED yang digunakan adalah LED dengan diameter 530 nm yang membutuhkan *power* sebesar 2V dan mengeluarkan arus sebesar 20 mA. LED ini dipilih karena untuk simulasi perangkat tidak dibutuhkan LED yang terlalu besar spesifikasinya. *Output* sistem

yang dibutuhkan hanyalah *output HIGH, LOW* dan *output* PWM. Sehingga LED jenis ini sudah cukup memenuhi persyaratan. Resistor ditentukan 220 Ohm. Untuk menentukan besar resistor maka harus dihitung terlebih dahulu resistansi minimal dengan menggunakan hukum kirchoff yaitu sebesar

$$V\ resistor = V\ gpio - V\ led$$

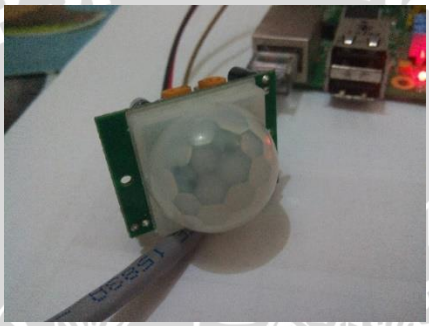
$$I.R = 5\ V - 2\ V$$

$$0{,}02\ A\ .\ R = 3\ V$$

$$R = 3\ V / 0.02\ A$$

$$R = 150\ Ohms.$$

Agar tegangan yang masuk tidak terlalu besar maka resistansi ditentukan lebih dari resistansi minimal, yaitu 220 Ohms.



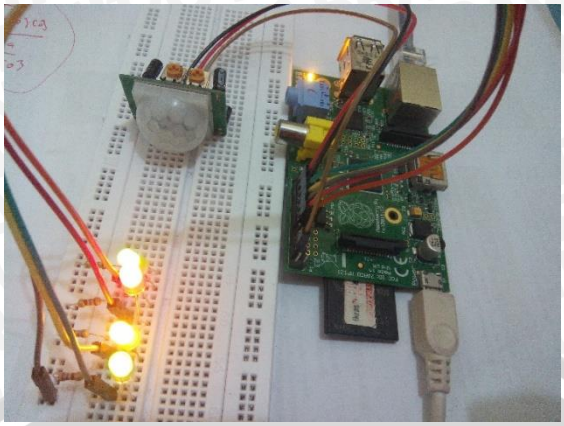
Gambar 5.12 Sensor PIR

Sensor yang digunakan pada penelitian ini adalah sensor PIR. Gambar 5.12 menunjukkan gambar implementasi sensor PIR.

Tabel 5.6 Spesifikasi Sensor PIR

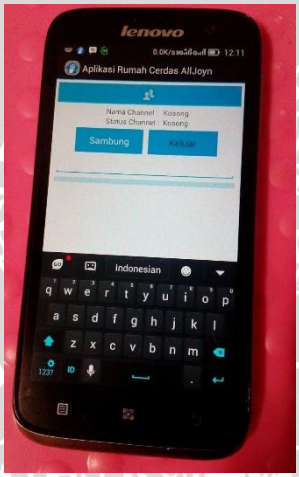
1.	*5V-16V power*
2.	*Digital signal output is 3.3V*
3.	*Sensing range is about 7 meters*
4.	*Height 24.66mm/0.97in, Width 32.34mm/1.27in*

Tabel 5.6 menjelaskan spesifikasi dari sensor PIR yang digunakan. Sensor PIR dihubungkan pada pin daya GPIO sebesar 5V. Sensor PIR akan menghasilkan sensor digital sebesar 3.3 V. Penelitian ini tidak berfokus pada jangkauan *sensing* sensor. Oleh karena itu dipilih sensor dengan jarak *sensing* yang tidak terlalu jauh, yaitu sejauh 7 meter. Gambar 5.12 menunjukkan gambar sensor PIR yang digunakan pada penelitian ini. Gambar 5.13 menunjukkan implementasi dari keseluruhan *hardware* dari sisi Raspberry Pi yang telah disusun sesuai dengan perancangan.



Gambar 5.13 Implementasi *Hardware* Sistem

Penelitian ini menggunakan *smartphone* sebagai perangkat untuk aplikasi. *Smartphone* yang digunakan adalah *smartphone* Lenovo A859. Gambar 5.14 menunjukkan implementasi perangkat *smartphone*.



Gambar 5.14 Smartphone Lenovo A859

Tabel 5.7 menunjukkan spesifikasi dari *smartphone* Android yang digunakan pada penelitian ini. *Smartphone* harus dapat terkoneksi dengan jaringan melalui WLAN atau Wifi. RAM ukuran 512 MB cukup untuk menjalankan aplikasi, namun pada saat impementasi peneliti menggunakan RAM ukuran 1 GB. Aplikasi dirancang untuk dijalankan pada sistem operasi Android. Untuk itu sistem operasi yang digunakan pada penelitian ini adalah Android Jelly Bean v4.2. Ukuran dari aplikasi cukup kecil sehingga tidak membutuhkan *memory* yang besar. Pada penelitian ini, peneliti menggunakan *smartphone* dengan *memory* 8 GB. Gambar 5.14 merupakan gambar *smartphone* yang digunakan pada penelitian ini.

Tabel 5.7 Spesifikasi *Smartphone* Android
Sumber : (Arena, 2014).

1.	*1.3 GHz MTK6582 Quad Core Processor*
2.	*1 GB RAM*
3.	*5 Inch HD IPS Touch Screen Display*
4.	*Connectivy Bluetooth, WiFi, USB, GPS*
5.	*Data GPRS/EDGE/3G*
6.	*Operating System Android Jelly Bean v4.2*
7.	*Memory 8GB microSD*

5.2.2 Implementasi *Software*

Implementasi *software* pada sistem ini meliputi implementasi pada sisi Raspberry Pi dan sisi aplikasi pada *smartphone* Android. Implementasi pada sisi Raspberry Pi adalah satu *source code* program dengan ekstensi .cc. Sedangkan untuk implementasi pada sisi aplikasi terdiri dari beberapa *source code* program dengan ekstensi .java dan .xml. Beberapa program ini nantinya akan menghasilkan aplikasi dengan ekstensi .apk yang ditanamkan pada *smartphone* Android.

5.2.2.1 Implementasi *Software* Pada Raspberry Pi

Program yang dijalankan pada sisi Raspberry Pi adalah satu program. Program sudah mencakup *source-code* untuk implementasi Alljoyn, *source-code* untuk komunikasi RPI dan aplikasi, *source-code* untuk kontrol GPIO, serta *source-code* untuk pembacaan sensor. Program menggunakan bahasa C sehingga Alljoyn Framework yang digunakan adalah *framework* untuk bahasa C yang dijalankan pada sistem operasi Linux.

```
16
17  #include <alljoyn/AllJoynStd.h>
18  #include <alljoyn/BusAttachment.h>
19  #include <alljoyn/BusObject.h>
20  #include <alljoyn/DBusStd.h>
21  #include <alljoyn/Init.h>
22  #include <alljoyn/InterfaceDescription.h>
23  #include <alljoyn/ProxyBusObject.h>
24  #include <qcc/Log.h>
25  #include <qcc/String.h>
26  #include <cassert>
27  #include <cstdio>
28  #include <cstdlib>
29  #include <signal.h>
30
31  #include <string>
32  #include <wiringPi.h> // Include WiringPi library!
33  #include <pthread.h>
```

Gambar 5.15 Include *Library* Program Raspberry Pi

Alljoyn Framework diimplementasikan pada *software* dengan memanggil terlebih dahulu *library-library* Alljoyn. Gambar 5.15 menunjukkan *library-library* yang digunakan pada *software* di Raspberry Pi. Komponen utama dalam Alljoyn Framework adalah *BusObject*, sehingga dibutuhkan *library BusAttachment*, *BusObject, DBusStd,* dan *ProxyBusObject*. Selain *library* untuk *Bus*, implementasi Alljoyn Framework juga membutuhkan *library* Alljoyn lain seperti AlljoynStd, Init,

dan *InterfaceDescription*. Komunikasi atau pertukaran pesan antara Raspberry Pi dan aplikasi menggunakan *signal* sehingga program menyertakan *library signal*. Untuk kontrol GPIO digunakan *library* WiringPi. Sedangkan untuk membuat thread sensor agar proses pembacaan sensor dapat berjalan bersamaan dengan proses lain digunakan *library* pthread.

Gambar 5.16 Inisialisasi Variabel Program Raspberry Pi

Variabel-variabel yang digunakan perlu diinisialisasi terlebih dahulu. Gambar 5.16 menunjukkan inisialisasi variabel pada *software* di Raspberry Pi. Penjelasan dari program pada gambar adalah :

- Baris 38-41 inisialisasi variabel untuk *service* Alljoyn. Nilai dari variabel-variabel ini harus sama dengan variabel yang ada pada aplikasi. Variabel ini merupakan "jembatan" atau tempat komunikasi bagi kedua program.
- Baris 44 – 50 inisialisasi untuk *BusObject* Alljoyn dan variabel-variabel yang digunakan untuk komunikasi menggunakan Alljoyn seperti *advertisement Name* dan *Session*.
- Baris 52 – 76 inisialisasi variabel untuk peralatan-peralatan pada GPIO. Variabel-variabel yang digunakan meliputi variabel *array* untuk daftar peralatan, daftar *action*, daftar *state*. Juga terdapat variabel *array* untuk daftar pin GPIO, set GPIO (*Input*, *Output* atau PWM), dan juga daftar nilai *device* (0, 1 atau nilai PWM). Tiap variabel *array* memiliki ukuran. Ukuran ini

yang dibutuhkan untuk proses selanjutnya. Untuk itu perlu diinisialisaikan variabel untuk ukuran dari variabel-variabel *array* seperti pada baris 62-76.

- Baris 80-81 inisialisasi variabel untuk menyimpan nilai sensor PIR dan LED indikator sensor PIR. Variabel ini digunakan pada proses pembacaan sensor.
- Baris 82-83 inisialisasi variabel untuk indeks pada perulangan.
- Baris 84 inisialisasi variabel thread untuk pembacaan sensor.
- Baris 84-89 inisialisasi variabel untuk proses pengiriman pesan. Karena pesan yang dikirimkan harus dalam bentuk pointer char maka perlu variabel untuk mengubah pesan dalam bentuk pointer char. Juga dibutuhkan variabel untuk menyimpan *prefix* pesan dan id *device*.

```
151
152     /** Send a Chat signal */
153     QStatus SendChatSignal(const char* msg) {
154
155         MsgArg chatArg("s", msg);
156         uint8_t flags = 0;
157         if (0 == s_sessionId) {
158             printf("Mengirim pesan tanpa session id\n");
159         }
160         printf("Kirim pesan : %s \n", msg);
161         return Signal(NULL, s_sessionId, *chatSignalMember, &chatArg, 1, 0, flags);
162     }
163
```

Gambar 5.17 Fungsi SendChatSignal Program Raspberry Pi

Gambar 5.17 menunjukkan fungsi untuk mengirim pesan ke aplikasi. Pesan dikirim dalam bentuk *signal*. *Signal* berisi isi *session id,* alamat yang dituju atau *signal member*, *flags*, dan informasi yang ingin dikirimkan. Informasi disimpan pada pesan dalam bentuk *pointer* char. Jika aplikasi belum terhubung dengan Raspberry Pi maka RPI mengirimkan pesan tanpa *session id*. Karena *session* terbentuk ketika terdapat 2 atau lebih perangkat yang terhubung.

```
180         /* Jika pesan yang masuk adalah pesan Join */
181         if (strcasecmp("Joined", responn) == 0) {
182             prefix = "D-";
183             for(i = 0; i < size_devices; i++){
184                 strcpy(ptr_message, devices[i].c_str());
185                 strcpy(ptr_prefix, prefix.c_str());
186                 snprintf(id_device, 3,"%d",i);
187                 SendChatSignal(strcat(strncat(ptr_prefix, id_device,10), ptr_message));
188             }
189             prefix = "A-";
190
191             for(i = 0; i < row_acts_device; i++){
192                 for(int j = 0; j < column_acts_device; j++){
193                         strcpy(ptr_message, acts_device[i][j].c_str());
194                     if (strlen(ptr_message) != 0){
195                         strcpy(ptr_prefix, prefix.c_str());
196                         snprintf(id_device, 3,"%d",i);
197
198                         SendChatSignal(strcat(strncat(ptr_prefix, id_device,10), ptr_message));
199                     }
200                 }
201             }
202
203             prefix = "S-";
204             int a = 0;
205             for(i = 0; i < size_devices; i++){
206                 a = digitalRead(pins_gpio[i]);
207                 strcpy(ptr_message, state_device[i][a].c_str());
208                 strcpy(ptr_prefix, prefix.c_str());
209                 snprintf(id_device, 3,"%d",i);
210                 SendChatSignal(strcat(strncat(ptr_prefix, id_device,10), ptr_message));
211             }
212         }
```

Gambar 5.18 Kode *Handling Message* "*Joined*"

Gambar 5.18 menunjukkan fungsi untuk memproses pesan yang masuk. Argumen dari pesan ini yang nantinya akan diolah berdasarkan isinya. Ketika pesan yang diterima berisi string *Joined* maka program mengirim daftar peralatan, daftar *action*s dan daftar *state* seperti pada Gambar 5.18. Pesan *device* dikirim dengan menggunakan *prefix* D- yang jumlahnya mengikuti ukuran dari *array device*. Pesan *action*s dikirim menggunakan *prefix* A- dan jumlahnya sesuai dengan ukuran *array* acts_device. Pesan *state* dikirim menggunakan *prefix* S- dan jumlahnya mengikuti ukuran dari *array device* Karena *state* merupakan keadaan untuk tiap peralatan, maka pesan *state* yang dikirimkan berisi nilai hasil dari pembacaan peralatan. Untuk itu tiap peralatan dibaca dahulu keadaannya dengan digitalRead(). Kemudian hasilnya (0 atau 1) merupakan index dari *state* peralatan (baris 206-207). Semua pesan dikirimkan dalam bentuk *pointer* char. Karena isi dari variabel berbentuk string, maka perlu diubah menjadi *pointer* char menggunakan fungsi strcpy(). Kemudian *prefix*, id *device*, dan isi dari masing-masing pesan digabungkan menjadi satu menggunakan fungsi strcat(). Hasil dari penggabungan ini yang dikirimkan menggunakan fungsi *SendChatSignal()*.

```
213         else
214
215         /* Jika pesan yang masuk pesan Action */
216         if (responn[0] == 'C') {
217             /* Cari dulu yang mau dicontrol pin berapa */
218             pin = (int)responn[2] - 49;
219             set_pin = (int)responn[4] - 48;
220
221             /* Kemudian lakukan control */
222             if (sets_gpio[pin] == 1){
223                 digitalWrite(pins_gpio[pin],acts_gpio[pin][set_pin]);
224                 printf("digitalWrite %d Acts %d \n",pins_gpio[pin],acts_gpio[pin][set_pin]);
225             }
226             else if (sets_gpio[pin] == 3){
227                 pwmWrite(pins_gpio[pin],acts_gpio[pin][set_pin]);
228                 printf("pwmWrite %d Acts %d \n",pins_gpio[pin],acts_gpio[pin][set_pin]);
229             }
230             prefix = "S-";
231             strcpy(ptr_message, state_device[pin][set_pin].c_str());
232             strcpy(ptr_prefix, prefix.c_str());
233             snprintf(id_device, 3,"%d",pin);
234             SendChatSignal(strcat(strncat(ptr_prefix, id_device,10), ptr_message));
235         }
```

Gambar 5.19 Kode *Handling Message "Control"*

Ketika pesan yang diterima adalah pesan *control* maka dilakukan proses seperti pada kode program Gambar 5.19. Pesan *control* menggunakan *prefix* C- sehingga karakter pertama atau index ke 0 dari pesan adalah C. Jika pesan yang diterima adalah pesan *control* maka program akan mengidentifikasi peralatan mana yang akan dikontrol dan *action* apa yang akan dilakukan. Identifikasi dilakukan dengan mengubah pesan karakter tertentu menjadi integer. Hasil dari *convert* ke integer ini adalah kode ascii. Sehingga agar nilai yang dihasilkan sesuai dengan isi pesan sebenarnya maka hasil *convert* perlu dikurangkan dengan nilai tertentu. Misalkan pesan yang diterima adalah C-1-2 maka karakter ke 3 yaitu 1 akan di-*convert* menjadi integer. Integer dari 1 adalah 49, maka agar nilai yang dihasilkan adalah 1 maka harus dikurangi dengan 48. Begitu juga dengan 2 yang nilai ascii nya adalah 50 juga harus dikurangi 48 agar menghasilkan 2. Selanjutnya program melakukan proses kontrol dengan menggunakan digitalWrite()

atau `pwmWrite()` tergantung dari jenis peralatan yang dikontrol (*Output* atau `PWM Output`). Kemudian program mengirim perubahan *state* peralatan dengan pesan *state* (baris 230 – 234).

```
238     static void* readSensor(void *arg)
239     {
240         static ChatObject* s_chatObj2 = NULL;
241         ChatObject *chatObj2 = reinterpret_cast<ChatObject *>(arg);
242         s_chatObj2 = chatObj2;
243         unsigned long i = 0;
244         int pinSensor=0;
245         int pinLedSensor=0;
246 
247         for(i = 0; i < size_gpio; i++){
248             if (sets_gpio[i] == 2){
249                 pinSensor = i;
250             } else
251             if (sets_gpio[i] == 4){
252                 pinLedSensor = i;
253             }
254         }
255         while(1){
256 
257             nilaiPir = digitalRead(pins_gpio[pinSensor]);
258             nilaiLedPir = digitalRead(pins_gpio[pinLedSensor]);
259             if (nilaiPir == HIGH){
260                 if (nilaiLedPir == LOW){
261                     printf("Sensor mendeteksi objek \n");
262                     digitalWrite(pins_gpio[pinLedSensor],1);
263                     prefix = "E-";
264                     strcpy(ptr_message, "TERBUKA");
265                     strcpy(ptr_prefix, prefix.c_str());
266                     snprintf(id_device, 3,"%d",3);
267                     s_chatObj2->SendChatSignal(strcat(strncat(ptr_prefix, id_device,10), ptr_message));
268 
269                     prefix = "S-";
270                     strcpy(ptr_message, state_device[3][1].c_str());
271                     strcpy(ptr_prefix, prefix.c_str());
272                     snprintf(id_device, 3,"%d",3);
273                     s_chatObj2->SendChatSignal(strcat(strncat(ptr_prefix, id_device,10), ptr_message));
274 
275                     delay(5000);
276                 }
277             }
278             else
279             {
280                 prefix = "S-";
281                 strcpy(ptr_message, state_device[3][0].c_str());
282                 strcpy(ptr_prefix, prefix.c_str());
283                 snprintf(id_device, 3,"%d",3);
284                 s_chatObj2->SendChatSignal(strcat(strncat(ptr_prefix, id_device,10), ptr_message));
285 
286                 digitalWrite(pins_gpio[pinLedSensor],0);
287             }
288             delay(3000);
289         }
290         for(i=0; i<(0xFFFFFFFF); i++);
291         return NULL;
292     }
293 
```

Gambar 5.20 Kode Pembacaan Sensor

Fungsi untuk membaca sensor dan mengirimkan hasilnya ke aplikasi terdapat pada Gambar 5.20. Fungsi ini merupakan fungsi yang dijalankan pada thread yang berbeda. Sehingga fungsi ini akan terus berjalan atau dijalankan bersamaan dengan fungsi lain yang ada pada program. Pada baris 244-254 program mengidentifikasi terlebih dahulu pin sensor dan pin LED untuk indikator sensor. Pin sensor menggunakan `set_gpio` 2 sedangkan pin LED untuk sensor menggunakan `set_gpio` 4. Kemudian program membaca nilai dari kedua pin tersebut. Jika pin sensor bernilai 1 atau HIGH maka program mengecek nilai dari LED sensor. Jika LED sensor belum menyala atau bernilai 0 maka program

memberikan nilai 1 atau menyalakan LED dan mengirimkan pesan perubahan *state* kepada aplikasi (baris 263-273). Pesan yang dikirimkan adalah pesan *state* dan pesan *event*. Jika LED sudah menyala maka program melanjutkan ke proses *looping* selanjutnya. Jika sensor bernilai *LOW* atau tidak mendeteksi adanya pergerakan obyek maka program mengirimkan informasi kepada aplikasi bahwa *state* dari LED sensor adalah 0 dalam sistem ini pintu tertutup.

```
297
298  class MyBusListener : public BusListener, public SessionPortListener, public SessionListener {
299      void FoundAdvertisedName(const char* name, TransportMask transport, const char* namePrefix)
300      {
328      void LostAdvertisedName(const char* name, TransportMask transport, const char* namePrefix)
329      {
333      void NameOwnerChanged(const char* busName, const char* previousOwner, const char* newOwner)
334      {
338      bool AcceptSessionJoiner(SessionPort sessionPort, const char* joiner, const SessionOpts& opts)
339      {
349
350      void SessionJoined(SessionPort sessionPort, SessionId id, const char* joiner)
351      {
366  };
367
```

Gambar 5.21 Kode BusListener Alljoyn

Fungsi-fungsi BusListener Alljoyn ditunjukkan pada Gambar 5.21. Fungsi yang dibutuhkan diantaranya adalah `FoundAdvertisedName` yaitu fungsi ketika program mendeteksi adalah *channel* Alljoyn lain, fungsi `LostAdvertisedName` yaitu fungsi ketika program kehilangan sambungan dengan *channel* Alljoyn lain, fungsi `NameOwnerChanged` yaitu fungsi untuk memberi identitas pada perangkat-perangkat yang terhubung dengan *channel* Raspberry Pi, fungsi `AcceptSessionJoiner` yaitu fungsi untuk menerima permintaan join session dari perangkat lain, serta fungsi `SessionJoined` yaitu fungsi ketika terdapat perangkat baru yang terhubung dengan *channel* Raspberry Pi. Fungsi-fungsi ini sudah disediakan oleh Alljoyn Framework sehingga peneliti hanya perlu menambahkan fungsi-fungsi ini ke dalam kode program.

```
428
429  QStatus CreateInterface(void)
430  {
444
445  QStatus StartMessageBus(void)
446  {
457
458  QStatus RegisterBusObject(void)
459  {
470
471  QStatus ConnectBusAttachment(void)
472  {
483
484  QStatus RequestName(void)
485  {
496
497  QStatus CreateSession(TransportMask mask)
498  {
511
512  QStatus AdvertiseName(TransportMask mask)
513  {
524
525  QStatus FindAdvertisedName(void)
526  {
538
539  QStatus WaitForJoinSessionCompletion(void)
540  {
557
```

Gambar 5.22 Fungsi - Fungsi Alljoyn

Gambar 5.22 juga menunjukkan fungsi-fungsi Alljoyn yang diimplementasikan pada program Raspberry Pi. Fungsi-fungsi ini diantaranya adalah fungsi untuk membuat interface Alljoyn, fungsi untuk membuat Bus, fungsi untuk meminta

nama ketika program join ke *channel* lain, fungsi untuk membuat session, fungsi untuk mengadvertise *service* dan fungsi untuk menunggu session terbentuk.

```
586     // Inisialisasi GPIO
587     wiringPiSetupGpio();
588
589     // set pin GPIO
590     for(i = 0; i < size_gpio; i++)
591     {
592         if(sets_gpio[i]==1 || sets_gpio[i]==4) {
593             pinMode(pins_gpio[i], OUTPUT);
594         } else if(sets_gpio[i]==2) {
595             pinMode(pins_gpio[i], INPUT);
596         } else if(sets_gpio[i]==3) {
597             pinMode(pins_gpio[i], PWM_OUTPUT);
598         }
599         printf ("Set Pinmode %d set %d \n",pins_gpio[i],sets_gpio[i]);
600     }
601
602     // Tes pin GPIO
603     for(i = 0; i < row_acts_gpio; i++){
604         for(j = 0; j < column_acts_gpio; j++){
605             if(sets_gpio[i]==1 || sets_gpio[i]==4)
606             {
607                 digitalWrite(pins_gpio[i], acts_gpio[i][j]); // Turn LED ON
608                 printf("Set digitalWrite pin %d acts %d \n",pins_gpio[i], acts_gpio[i][j]);
609                 delay(500);
610             }
611             else
612             if(sets_gpio[i]==3)
613             {
614                 pwmWrite(pins_gpio[i], acts_gpio[i][j]); // Turn LED ON
615                 printf("Set pwmWrite pin %d acts %d \n",pins_gpio[i], acts_gpio[i][j]);
616                 delay(500);
617             }
618             else
619             {
620                 printf("Sensor %d digitalRead() \n",pins_gpio[i]);
621                 if (digitalRead(pins_gpio[i]) == HIGH)
622                     printf("Sensor mendeteksi objek \n");
623                 else
624                     printf("Sensor tidak mendeteksi objek \n");
625             }
626         }
627     }
628     printf("GPIO Sudah Diatur \n");
629
```

Gambar 5.23 Kode Inisialisasi Perangkat Pada GPIO

Gambar 5.23 menunjukkan kode program untuk menginisialisasi peralatan pada GPIO. Penjelasan dari program tersebut adalah :

- Baris 587 : Inisisalisasi wiringPi
- Baris 590-600 : Set `pinMode` GPIO. Data `pinMode` diambil dari *array* `pins_gpio`
- Baris 603-627 : Memastikan perangkat dapat bekerja dengan baik dengan melakukan semua *action* yang dapat dilakukan peralatan. Jika `pinMode` nya `OUTPUT` maka dilakukan *action* `digitalWrite`. Jika `pinMode` nya *`PWM`*`_OUTPUT` maka dilakukan *action* `pwmWrite`. Jika `pinMode` nya *`INPUT`* maka dilakukan *action* `digitalRead`. Nilai *action* `digitalWrite` dan `pwmWrite` diambil dari *array* `acts_gpio`.

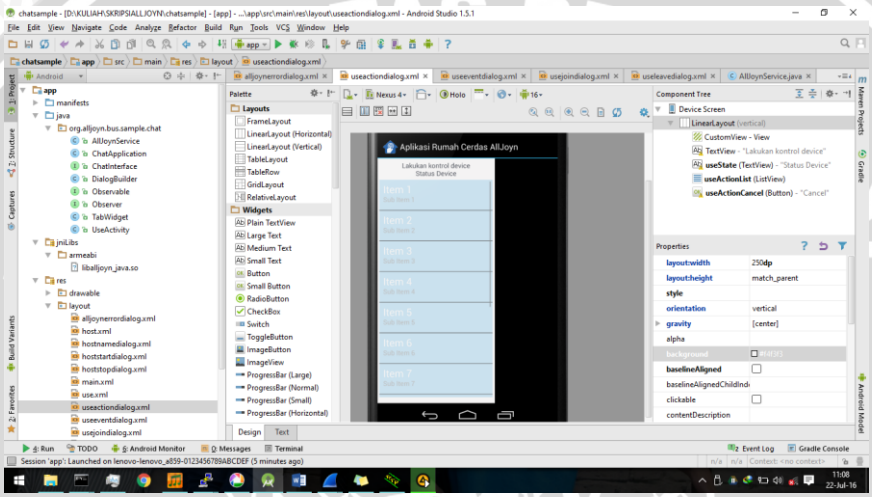
```
712         int err=0;
713         err = pthread_create(&(tid[0]), NULL, &ChatObject::readSensor, &chatObj);
714         if (err != 0)
715             printf("Gagal membuat thread \n");
716         else
717             printf("Berhasil membuat Thread\n");
718
```

Gambar 5.24 Kode Inisialisasi Thread Untuk Sensor

Gambar 5.24 menunjukkan kode program untuk membuat thread untuk pembacaan sensor. Program harus selalu membaca sensor secara terus menerus. Namun program juga harus dapat menerima pesan dari aplikasi kapan saja. Untuk itu dibutuhkan thread agar dua proses dapat dijalankan secara bersamaan. Baris 713 menunjukkan kode untuk membuat thread dimana thread yang dibuat menjalankan fungsi `readSensor`. Thread ini juga memberikan parameter berupa `chatObj` agar di dalam thread tersebut dapat dijalankan fungsi untuk mengirim pesan kepada aplikasi.

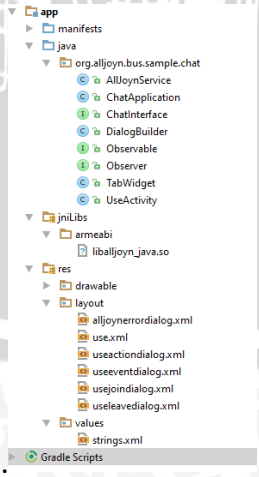
5.2.2.2 Implementasi Desain Aplikasi Android

Dalam membangun aplikasi untuk *smartphone* peneliti menggunakan Android Studio sebagai IDE seperti pada Gambar 5.25.



Gambar 5.25 Android Studio IDE

Dalam implementasi yang telah dilakukan aplikasi yang dibangun terdiri dari beberapa *file* java dan beberapa *file* xml. Daftar *file* tersebut ada pada Gambar 5.26



Gambar 5.26 Daftar *File* Implementasi Pada Aplikasi

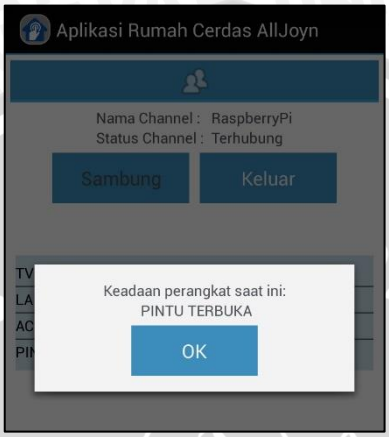
File AlljoynService merupakan *file* yang berisi layanan-layanan Alljoyn yang digunakan dalam sistem seperti layanan *joinChannel*, *leaveChannel*, *discovery*, dan lain-lain. *File* ChatApplication digunakan untuk mengatur proses pengiriman dan penerimaan pesan. File ini juga digunakan untuk menyimpan variabel-variabel hasil pesan yang diterima. *File* DialogBuilder digunakan untuk mengatur kotak dialog join, *action*, *event*, dan dialog *error*. *File* UseActivity digunakan untuk mengatur aktifitas dari tampilan *file* use.xml. *File* ini yang mengatur apa yang ditampilkan dalam *list view*, apa yang dilakukan ketika *button* ditekan, dan aktivitas lain. *File* liballjoyn_java.so merupakan *library* alljoyn yang harus disertakan pada aplikasi yang menggunakan java. *File-file* yang ada pada direktori *layout* merupakan *file-file* yang berisi desain tampilan dari aplikasi.

Gambar 5.27 Desain Halaman Depan Aplikasi

Langkah awal yang dilakukan adalah mendesain tampilan aplikasi. Gambar 5.27 merupakan desain untuk halaman depan aplikasi. Layout yang digunakan adalah `LinierLayout` yang artinya komponen-komponen diatur dalam satu kolom atau baris. Ukuran dari layout menggunakan ukuran `match_parent` yang artinya ukurannya sebesar ukuran *parent*-nya dalam hal ini adalah ukuran layar yang digunakan. Nama *channel* dan status *channel* ditampilkan menggunakan `EditText`. Terdapat dua *button* untuk menyambungkan aplikasi ke Raspberry Pi dan untuk memutuskan sambungan. *Device list* ditampilkan menggunakan `ListView`.

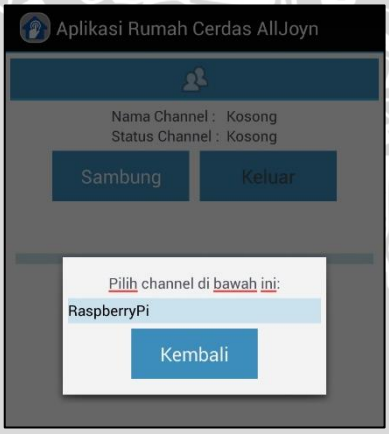
Gambar 5.28 Desain Dialog *Action Device*

Gambar 5.28 merupakan desain tampilan untuk kotak dialog *action*. Kotak dialog ini muncul ketika *user* menekan salah satu peralatan yang ada pada *device list*. Pada kotak dialog ini terdapat *text* untuk menampilkan *state* dari peralatan yang dipilih. Daftar *action* yang dapat dilakukan terhadap peralatan tersebut ditampilkan dalam bentuk `ListView`. Isi dari `ListView` *action* berubah-ubah sesuai dengan peralatan yang dipilih.



Gambar 5.29 Desain Dialog Pesan *Event*

Gambar 5.29 merupakan gambar desain tampilan kotak dialog *event*. Kotak dialog ini muncul ketika terdapat pesan *event* yang masuk ke aplikasi. Kotak dialog ini menampilkan *state* dari peralatan yang dikontrol berdasarkan *input* sensor. *State* peralatan ditampilkan menggunakan `TextView`.



Gambar 5.30 Desain Dialog *Join Channel*

Gambar 5.30 merupakan gambar desain tampilan untuk kotak dialog *join channel*. Dialog ini muncul ketika *user* menekan *button* "Sambung" pada halaman depan. Dialog ini berisi daftar *channel* apa saja yang dapat disambungkan dengan aplikasi. Daftar *channel* juga merupakan *channel* yang berada pada jaringan yang sama dengan jaringan *smartphone*. Daftar *channel* ditampilkan dengan menggunakan `ListView`.

5.2.2.3 Implementasi Alljoyn Service

Alljoyn Framework juga diimplementasikan pada aplikasi untuk *smartphone* Android. Pada penelitian ini layanan-layanan Alljoyn yang akan digunakan diletakkan pada satu *file* java. Fungsi-fungsi yang ada pada *file* ini nanti akan dipanggil oleh fungsi-fungsi lain. Untuk dapat mengimplementasikan, maka *library-library* Alljoyn harus di-*import* terlebih dahulu. Gambar 5.31 menunjukkan *library-library* yang di-*import* ke dalam program Alljoyn *Service*.

```
1
2   package org.alljoyn.bus.sample.chat;
3
4   import android.app.Notification;
5   import android.app.PendingIntent;
6   import android.app.Service;
7   import android.content.Intent;
8   import android.os.Handler;
9   import android.os.HandlerThread;
10  import android.os.IBinder;
11  import android.os.Looper;
12  import android.os.Message;
13  import android.util.Log;
14
15  import org.alljoyn.bus.BusAttachment;
16  import org.alljoyn.bus.BusException;
17  import org.alljoyn.bus.BusListener;
18  import org.alljoyn.bus.BusObject;
19  import org.alljoyn.bus.MessageContext;
20  import org.alljoyn.bus.Mutable;
21  import org.alljoyn.bus.SessionListener;
22  import org.alljoyn.bus.SessionOpts;
23  import org.alljoyn.bus.SessionPortListener;
24  import org.alljoyn.bus.SignalEmitter;
25  import org.alljoyn.bus.Status;
26  import org.alljoyn.bus.annotation.BusSignalHandler;
27
28  public class AllJoynService extends Service implements Observer {
29      private static final String TAG = "chat.AllJoynService";
30
31      public IBinder onBind(Intent intent) {
32          Log.i(TAG, "onBind()");
33          return null;
35
36      public void onCreate() {
37          Log.i(TAG, "onCreate()");
38          startBusThread();
39          mChatApplication = (ChatApplication)getApplication();
40          mChatApplication.addObserver(this);
41
42          CharSequence title = "AllJoyn";
43          CharSequence message = "Aplikasi Rumah Cerdas";
44          Intent intent = new Intent(this, TabWidget.class);
45          PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, intent, 0);
46          Notification notification = new Notification(R.drawable.icon, null, System.currentTimeMillis());
47          notification.setLatestEventInfo(this, title, message, pendingIntent);
48          notification.flags |= Notification.DEFAULT_SOUND | Notification.FLAG_ONGOING_EVENT | Notification.FLAG_NO_CLEAR;
49
50          Log.i(TAG, "onCreate(): startForeground()");
51          startForeground(NOTIFICATION_ID, notification);
52
53          mBackgroundHandler.connect();
54          mBackgroundHandler.startDiscovery();
55      }
56
```

Gambar 5.31 Kode Inisialisasi Alljoyn *Service*

Terdapat *library* android dan *library* Alljoyn. *Library* android yang digunakan antara lain adalah `PendingIntent` untuk menggunakan Intent dalam aplikasi. Intent adalah pesan asynchronous yang memungkinkan aplikasi untuk memberikan *request* secara fungsionalitas dari komponen yang berbeda di sitem android. Terdapat juga *library* log agar aktifitas aplikasi tersimpan dalam log. *Library* `notification` digunakan melakukan pemberitahuan pada sistem. *Library* Alljoyn yang digunakan antara lain adalah *library* untuk mengatur *Alljoyn Bus* yaitu `BusAttachment`, `BusException`, `BusListerner` dan `BusObject` serta *library* untuk mengatur *Alljoyn Session* yaitu

SessionListener, *SessionOpts*, *SessionPortListener*. Terdapat juga *library* *SignalEmitter* untuk menerima *signal* dan juga *library* Status untuk menggunakan status-status pada Alljoyn.

Ketika program Alljoyn *Service* dijalankan pertama kali, program ini menjalankan fungsi onCreate(). Program ini menjalankan *Bus* dengan fungsi startBusThread(). Kemudian program mendapatkan aplikasi dari *file* ChatApplication. Kemudian program membuat Intent agar dapat berhubungan dengan sistem. Selanjutnya notifikasi dibuat dengan isi variabel *title* dan *message*. Notifikasi ini menampilkan indikator bahwa aplikasi sedang aktif. Kemudian program melakukan koneksi ke backgroundHandler dan melakukan *discovery* perangkat.

5.2.2.4 Implementasi Pengolahan Data Array

Pesan-pesan yang masuk ke aplikasi akan diolah menjadi *array-array*. *Array-array* ini nantinya yang akan ditampilkan pada tampilan aplikasi. Pada sistem ini aplikasi menerima 4 jenis pesan, yaitu pesan *device*, *actions*, *state*, dan *event*. Untuk itu sistem ini juga menggunakan 4 variabel *array* untuk menyimpan pesan-pesan tersebut.

```
321     final int HISTORY_MAX = 20;
322
323     private List<String> mHistory = new ArrayList<~>();
324
325     private void addHistoryItem(String message) {
326         if (mHistory.size() == HISTORY_MAX) {
327             mHistory.remove(0);
328         }
329
330         mHistory.add(message);
331         notifyObservers(HISTORY_CHANGED_EVENT);
332     }
333
334     private void clearHistory() {
335         mHistory.clear();
336         notifyObservers(HISTORY_CHANGED_EVENT);
337     }
338
339     public synchronized List<String> getHistory() {
340         List<String> clone = new ArrayList<~>(mHistory.size());
341         for (String string : mHistory) {
342             clone.add(new String(string));
343         }
344         return clone;
345     }
```

Gambar 5.32 Kode Pengolahan *Array Device*

Gambar 5.32 menunjukkan kode program untuk mengolah *array device*. *Array device* diinisialisasi dengan nama mHistory (baris 232). Sebelum isi pesan dimasukkan ke dalam *array* mHistory, *array* tersebut dicek terlebih dahulu, apakah ukurannya melebihi ukuran maksimal *array* yaitu 20 atau tidak. Jika iya maka isi dari *array* mHistory indeks ke 0 dihapus (baris 326-328). Selanjutnya program menambahkan isi pesan ke dalam *array* dengan fungsi add(). Selanjutnya program memanggil fungsi notifyObservers dengan parameter HISTORY_CHANGED_EVENT untuk menambahkan peralatan baru tadi ke dalam *device list* (baris 330-331). Jika aplikasi berganti *channel*, maka aplikasi akan memanggil fungsi clearHistory. Fungsi ini akan mengosongkan *array* mHistory dan mengosongkan *device list* yang ada pada tampilan depan (baris 334-337). Untuk menampilkan isi dari *arraylist* mHistory ke dalam *history* *listView* pada tampilan depan maka program akan memanggil fungsi

`getHistory`. Fungsi ini memasukkan kembali isi dari `mHistory` ke dalam *array* baru. *Array* baru ini yang nanti akan dikembalikan ke kode yang memanggil fungsi.

```
115
116     public synchronized String getDevice(int device_id){
117         String clone = "";
118         int a=0;
119         for (String string : mHistory) {
120             if(a == device_id) {
121                 Log.i(TAG, "getDevice(): added " + string);
122                 clone = string;
123             }
124             a++;
125         }
126         return clone;
127     }
128
```

Gambar 5.33 Kode Get *Device* Name

Gambar 5.33 merupakan kode program untuk mendapatkan nama peralatan. Pesan *State* dan *Event* berisi id *device*. Untuk itu agar dapat ditampilkan pada kotak dialog maka dibutuhkan nama peralatan. Fungsi ini melakukan pengecekan pada semua isi dari *array* `mHistory` dengan indeks. Jika indeks nya sama dengan id *device* maka nilai dari isi *array* dengan indeks tersebut yang dikembalikan.

```
84      public synchronized void addState(Module m, String state) {
85          mModule = m;
86          removeState(state);
87          if(state.length() != 3) {
88              mStates.add(state);
89              Log.i(TAG, "addState(): added " + state);
90          }
91      }
92
93      public synchronized void removeState(String state) {
94          for (Iterator<String> i = mStates.iterator(); i.hasNext();) {
95              String string = i.next();
96              if (string.substring(2,3).equals(state.substring(2, 3))) {
97                  Log.i(TAG, "removeState(): removed " + state);
98                  i.remove();
99              }
100         }
101     }
102
103     public synchronized String getState(int device_id) {
104         String clone = "";
105         for (String string : mStates) {
106             if(Integer.parseInt(string.substring(2,3)) == device_id) {
107                 Log.i(TAG, "getState(): added " + string);
108                 clone = " " + getDevice(device_id) + " " + string.substring(3);
109             }
110         }
111         return clone;
112     }
113
114     private List<String> mStates = new ArrayList<>();
```

Gambar 5.34 Kode Pengolahan *Array State*

Gambar 5.34 merupakan kode program untuk mengolah *arraylist state*. Proses penambahan nilai baru ke dalam *array state* dituliskan pada program baris ke 84-91. Jika panjang pesan lebih dari 3 atau pesan tidak hanya berisi *prefix* maka pesan tersebut dimasukkan ke *array*. Sebelum ditambahkan, *state* tersebut juga dihapuskan atau di-*remove* jika *state* baru tersebut sudah ada dalam *array state*. Jika *state* peralatan sama, maka *state* yang lama dihapus (baris 93-101). Untuk menampilkan *state* pada kotak dialog *action* maka program memanggil fungsi `getState`. Fungsi ini mengambil *state* dari *array* `mState`. Jika *device* id nya sama maka *state* tersebut yang dikembalikan ke kode program yang memanggil (baris 103-112).

```
50      public synchronized void addAction(Module m, String action) {
51          mModule = m;
52          removeAction(action);
53          if(action.length() != 3) {
54              mActions.add(action);
55              Log.i(TAG, "addAction(): added " + action);
56          }
57      }
58
59      public synchronized void removeAction(String action) {
60          for (Iterator<String> i = mActions.iterator(); i.hasNext();) {
61              String string = i.next();
62              if (string.equals(action)) {
63                  Log.i(TAG, "removeAction(): removed " + action);
64                  i.remove();
65              }
66          }
67      }
68
69      public synchronized List<String> getActionList(int device_id) {
70          List<String> clone = new ArrayList<>(mActions.size());
71          for (String string : mActions) {
72              if(Integer.parseInt(string.substring(2,3)) == device_id) {
73                  Log.i(TAG, "getActionList(): added " + string);
74                  clone.add(new String(string));
75              }
76          }
77          return clone;
78      }
79
80      private List<String> mActions = new ArrayList<>();
```

Gambar 5.35 Kode Pengolahan *Array Actions*

Gambar 5.35 merupakan kode program untuk mengolah *arraylist action*. Proses penambahan nilai baru ke dalam *array Action* dituliskan pada program baris ke 50-57. Jika panjang pesan lebih dari 3 atau pesan tidak hanya berisi *prefix* maka pesan tersebut dimasukkan ke *array*. Sebelum ditambahkan, *action* tersebut juga dihapuskan atau di-*remove* jika *action* baru tersebut sudah ada dalam *array action*. Jika *action* peralatan sama, maka *action* yang lama dihapus (baris 59-67). Untuk menampilkan *action* pada kotak dialog *action* maka program memanggil fungsi `getAction`. Fungsi ini mengambil *action* dari *array* `mAction`. Jika *device* id nya sama maka *action* tersebut yang dikembalikan ke kode program yang memanggil (baris 69-78).

```
128
129     public synchronized void addEvent(Module m, String s) {
130         int id = Integer.parseInt(s.substring(2,3));
131         mModule = m;
132         mEventString = " " + getDevice(id) + " " + s.substring(3);
133     }
134
135     public String getEvent() { return mEventString; }
138
```

Gambar 5.36 Kode Penambahan *Event*

Gambar 5.36 merupakan kode program untuk menambahkan pesan *event* ke dalam variabel `mEventString`. Fungsi ini mengganti isi dari variabel `mEventString` dengan pesan *event* yang diterima. Isi `mEventString` secara keseluruhan adalah nama peralatan yang didapat dari pemanggilakn fungsi `getDevice` dan isi pesan *event* mulai dari string ke 3. Kotak dialog akan memanggil fungsi `getEvent` yang mengembalikan nilai `mEventString` untuk ditampilkan.

5.2.2.5 Implementasi *Handle Message*

Sistem ini menggunakan 5 jenis pesan. Tiap pesan memiliki perlakuan yang berbeda-beda. Untuk itu diperlukan program untuk menangani masing-masing pesan. Juga perlu diatur bagaimana mengirim pesan ke Raspberry Pi. Sub bab ini menjelaskan tentang cara menangani pesan yang dikirim dan diterima.

```
261    public synchronized void newLocalUserMessage(String message) {
262
263        if (useGetChannelState() == AllJoynService.UseChannelState.JOINED) {
264            addOutboundItem(message);
265        }
266    }
```

```
248    public void sendMessages() {
249        Log.i(TAG, "mBackgroundHandler.sendMessages()");
250        Message msg = mBackgroundHandler.obtainMessage(SEND_MESSAGES);
251        mBackgroundHandler.sendMessage(msg);
252    }
253
```

Gambar 5.37 Kode Mengirim Pesan

Gambar 5.37 merupakan kode untuk mengirim pesan ke Raspberry Pi. Pesan dikirim jika aplikasi telah bergabung dengan *channel*. Potongan kode bagian awal (baris 261-266) merupakan kode program yang ada pada *file* ChatApplication. Fungsi ini yang akan dipanggil ketika *user* melakukan kontrol atau dikirim untuk mengirim pesan *control*. Fungsi ini akan memanggil fungsi `sendMessages` yang ada pada *file* AlljoynService (baris 248-253). Pengiriman pesan sendiri sudah ada dalam layanan Alljoyn, sehingga peneliti hanya perlu menyisipkan kode program tersebut.

```
267
268    public synchronized void newRemoteUserMessage(String message) {
269
270        if(message.startsWith("A-")) {
271            addAction(Module.USE, message);
272        } else
273
274        if(message.startsWith("D-")) {
275            addHistoryItem(message.substring(3));
276        } else
277
278        if(message.startsWith("S-")) {
279            addState(Module.USE, message);
280            int id = Integer.parseInt(message.substring(2,3));
281            Toast.makeText(getBaseContext(), " " + getDevice(id) + " " + message.substring(3), Toast.LENGTH_LONG).show();
282        } else
283
284        if(message.startsWith("E-")) {
285            addEvent(Module.USE, message);
286            notifyObservers(DEVICE_EVENT);
287        }
288
289        addOutboundItem("Diterima");
290    }
291
```

Activate Windows

Gambar 5.38 Kode *Handling Message*

Gambar 5.38 merupakan kode program untuk mengolah pesan yang masuk pada aplikasi. Pesan yang masuk memiliki tipe data String. Jika pesan yang masuk adalah pesan *action* atau pesan diawali dengan *prefix* "`A-`" maka program memanggil fungsi `addAction` untuk menambahkan isi pesan tersebut ke dalam *array Action*s (baris 270-272). Jika pesan yang masuk adalah pesan *device* atau pesan diawali dengan *prefix* "`D-`" maka program memanggil fungsi `addHistoryItem` untuk menambahkan isi pesan ke *array Device* (baris 274-276). Jika pesan yang masuk adalah pesan *state* atau pesan diawali dengan *prefix* "`S-`" maka program memanggil fungsi `addState` untuk menambahkan isi pesan ke dalam *array State*. Kemudian program menampilkan *state* peralatan dalam bentuk `Toast` (baris 278-282). Jika pesan yang masuk adalah pesan *event* atau pesan yang diawali dengan *prefix* "`E-`" maka program memanggil fungsi `addEvent` untuk menambahkan pesan *event* ke variabel `mEventString`. Kemudian program memanggil fungsi `notifyObserver` dengan parameter `DEVICE_EVENT` untuk menampilkan kotak dialog *event* (baris 284-287).

5.2.2.6 Implementasi Kotak Dialog

Kotak dialog muncul sebagai kotak *pop up* ketika terjadi keadaan yang telah ditentukan. Pada penelitian ini terdapat empat macam kotak dialog yang muncul. Yaitu kotak dialog *join channel* ketika *user* menekan *button* "Sambung", kotak dialog *action* ketika *user* menekan salah satu peralatan pada *device list*, kotak dialog *event* ketika terdapat pesan *event* yang masuk dan kotak dialog *error* ketika terjadi *error* pada aplikasi. Kotak dialog menggunakan *library* `dialog` yang ada pada `android.app`. Untuk membuat kotak dialog harus dideklarasikan terlebih dahulu variabel dengan tipe data `Dialog`.

```
16  public class DialogBuilder {
17      private static final String TAG = "chat.Dialogs";
18
19      public Dialog createUseJoinDialog(final Activity activity, final ChatApplication application) {
20          Log.i(TAG, "createUseJoinDialog()");
21          final Dialog dialog = new Dialog(activity);
22          dialog.requestWindowFeature(dialog.getWindow().FEATURE_NO_TITLE);
23          dialog.setContentView(R.layout.usejoindialog);
24
25          ArrayAdapter<String> channelListAdapter = new ArrayAdapter<~>(activity, android.R.layout.test_list_item);
26          final ListView channelList = (ListView)dialog.findViewById(R.id.useJoinChannelList);
27          channelList.setAdapter(channelListAdapter);
28
29          List<String> channels = application.getFoundChannels();
30          for (String channel : channels) {
31              int lastDot = channel.lastIndexOf('.');
32              if (lastDot < 0) {
33                  continue;
34              }
35              channelListAdapter.add(channel.substring(lastDot + 1));
36          }
37          channelListAdapter.notifyDataSetChanged();
38
39          channelList.setOnItemClickListener((parent, view, position, id) -> {
41                  String name = channelList.getItemAtPosition(position).toString();
42                  application.useSetChannelName(name);
43                  application.useJoinChannel();
44                  activity.removeDialog(UseActivity.DIALOG_JOIN_ID);
45          });
47
48          Button cancel = (Button)dialog.findViewById(R.id.useJoinCancel);
49          cancel.setOnClickListener((view) -> {
51                  activity.removeDialog(UseActivity.DIALOG_JOIN_ID);
52          });
54
55          return dialog;
56      }
```

Activate Window
Go to Settings to activ

Gambar 5.39 Kode Dialog *Join Channel*

Gambar 5.39 merupakan kode program fungsi untuk membuat kotak dialog *Join Channel*. Fungsi ini mengambil dua parameter yaitu `activity` dan `chatApplication`. `Activity` merupakan *container* untuk *User Interface* (UI). `ChatApplication` digunakan karena dialog ini akan mengambil beberapa nilai dari variabel yang ada pada *file* ChatApplication. Fungsi ini membuat kotak dialog dengan tampilan yang ada pada *file* usejoindialog.xml (baris 23). Kotak dialog join *channel* menampilkan daftar *channel* yang pada jaringan. Daftar *channel* tersebut didapat dari pemanggilan fungsi `getFoundChannel` yang ada pada *file* ChatApplication (baris 29). Hasil kembalian dari pemanggilan fungsi berbentuk *array* kemudian di-*convert* menjadi bentuk `arrayAdapter` agar dapat ditampilkan pada `ListView` (baris 30-37). Jika *user* melakukan klik pada salah satu *channel* yang ada pada `ListView` maka program menjalankan kode baris ke 39-45 yaitu memanggil fungsi `setChannelName` untuk mengubah *Channel* pada tampilan depan dan fungsi `useJoinChannel` untuk melakukan *request* untuk *join channel* kepada Raspberry Pi. Jika *user* melakukan klik pada *button*

Cancel maka program akan menjalankan kode program baris ke 48-52 yaitu menghilangkan kotak dialog *join channel*.

```
58  public Dialog createActionDialog(final Activity activity, final ChatApplication application, final int device_id) {
59      Log.i(TAG, "createActionDialog()");
60      final Dialog dialog = new Dialog(activity);
61      dialog.requestWindowFeature(dialog.getWindow().FEATURE_NO_TITLE);
62      dialog.setContentView(R.layout.useactiondialog);
63
64      TextView stateText = (TextView)dialog.findViewById(R.id.useState);
65      stateText.setText(application.getState(device_id));
66
67      ArrayAdapter<String> actionListAdapter = new ArrayAdapter<>(activity, android.R.layout.test_list_item);
68      final ListView actionList = (ListView)dialog.findViewById(R.id.useActionList);
69      actionList.setAdapter(actionListAdapter);
70
71      List<String> actions = application.getActionList(device_id);
72      String a;
73      for (String action : actions) {
74          a = action.substring(3);
75          Log.i(TAG, "createActionDialog() add : "+action);
76          actionListAdapter.add(a);
77      }
78      actionListAdapter.notifyDataSetChanged();
79
80      actionList.setOnItemClickListener((parent, view, position, id) -> {
82              String name = actionList.getItemAtPosition(position).toString();
83              int f = device_id + 1;
84              application.newLocalUserMessage("C-"+f+"-"+(int)id);
85              dialog.cancel();
86              activity.removeDialog(UseActivity.DIALOG_ACTION_ID);
87      });
89
90      Button cancel = (Button)dialog.findViewById(R.id.useActionCancel);
91      cancel.setOnClickListener((view) -> {
93              dialog.cancel();
94              activity.removeDialog(UseActivity.DIALOG_ACTION_ID);
95      });
97
98      return dialog;
99  }
```

Gambar 5.40 Kode Dialog *Action Device*

Gambar 5.40 merupakan kode program fungsi untuk membuat kotak dialog *Action*. Fungsi ini mengambil tiga parameter yaitu `activity`, `chatApplication` dan *`device`*_id. Parameter *`device`*_id digunakan untuk menentukan *action list* apa saja yang harus ditampilkan karena *action list* tiap peralatan berbeda. Fungsi ini membuat kotak dialog dengan tampilan yang ada pada *file* useactiondialog.xml (baris 62). Kotak dialog *action* berisi *state* dan *action list* dari peralatan yang dipilih. State dari peralatan didapatkan dengan memanggil fungsi `getState` dan menampilkan kembalian dari fungsi ke `TextView useState` (baris 64-65). *Action list* didapat dari pemanggilan fungsi `getActionList` yang ada pada *file* ChatApplication (baris 72). Hasil kembalian dari pemanggilan fungsi berbentuk *array* kemudian di-*convert* menjadi bentuk `arrayAdapter` agar dapat ditampilkan pada `ListView` (baris 72-78). Jika *user* melakukan klik pada salah satu *action* yang ada pada `ListView` maka program menjalankan kode baris ke 80-87 yaitu memanggil fungsi `newLocalMessage` untuk mengirimkan pesan *action* kepada Raspberry Pi dengan format `C-`*`<device_id>`*`-`*`<action_id>`*. Jika *user* melakukan klik pada *button Cancel* maka program akan menjalankan kode program baris ke 90-95 yaitu menghilangkan kotak dialog *action*.

```
145
146     public Dialog createEventDialog(final Activity activity, final ChatApplication application) {
147         Log.i(TAG, "createEventDialog() Variabel StateString : "+application.getEvent());
148         final Dialog dialog = new Dialog(activity);
149         dialog.requestWindowFeature(dialog.getWindow().FEATURE_NO_TITLE);
150         dialog.setContentView(R.layout.useeventdialog);
151
152         TextView errorText = (TextView)dialog.findViewById(R.id.eventDescription);
153         errorText.setText(application.getEvent());
154
155         Button yes = (Button)dialog.findViewById(R.id.stateOk);
156         yes.setOnClickListener((view) -> {
158                 activity.removeDialog(UseActivity.DIALOG_DEVICE_EVENT);
159                 dialog.cancel();
160         });
162
163         return dialog;
164     }
165 }
166
```

Gambar 5.41 Kode Dialog Pesan *Event*

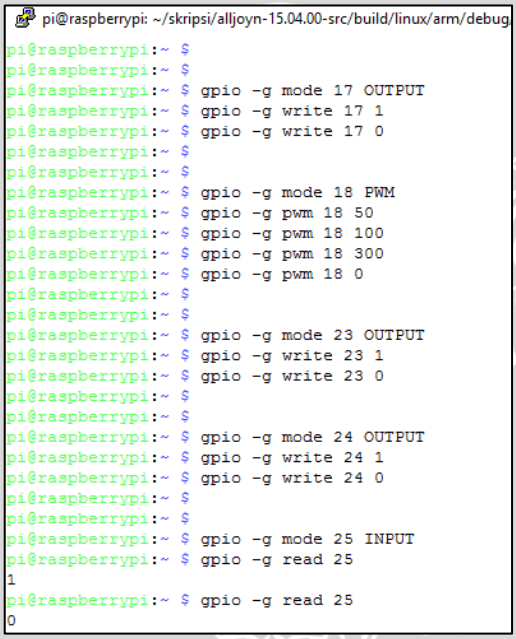
Gambar 5.41 merupakan kode program fungsi untuk membuat kotak dialog *Event*. Fungsi ini mengambil dua parameter yaitu `activity`, dan `chatApplication`. Fungsi ini membuat kotak dialog dengan tampilan yang ada pada *file* useeventdialog.xml (baris 150). Kotak dialog *event* berisi *state* dari peralatan yang mengalami perubahan *state* akibat perubahan *output* sensor. State tersebut didapatkan dengan memanggil fungsi `getEvent` dan menampilkan kembalian dari fungsi ke `TextView` *event*`Description` (baris 152-153). Jika *user* melakukan klik pada *button Cancel* maka program akan menjalankan kode program baris ke 155-160 yaitu menghilangkan kotak dialog *event*.

BAB 6 PENGUJIAN DAN ANALISIS

Bab ini akan membahas mengenai hasil pengujian beserta analisisnya untuk membuktikan sistem telah berhasil dibuat.

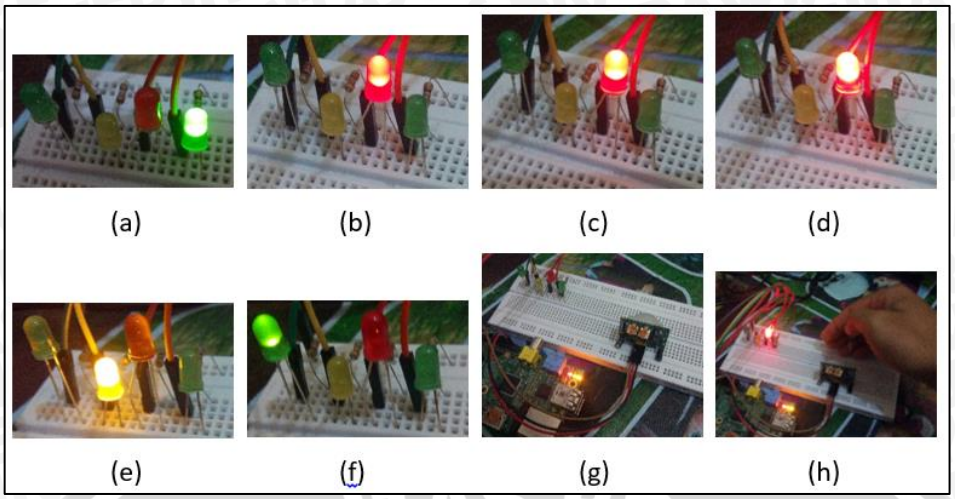
6.1 Pengujian Fungsional Sensor dan LED

Pengujian ini bertujuan untuk memastikan Sensor Pir dan keempat LED dapat berfungsi dengan baik. Pengujian ini juga untuk memastikan GPIO pada Raspberry Pi dapat bekerja dengan baik. Komponen sensor dan LED harus dipastikan terpasang pada pin yang sesuai. Sensor Pir sebagai komponen *input* pada sistem harus dapat mendeteksi adanya gerakan di area deteksi sensor. Raspberry Pi juga harus dapat menampilkan status dari deteksi Sensor Pir, apakah sedang mendeteksi adanya obyek atau tidak. LED harus dapat mengeluarkan *output* berupa menyala dan *output* dimming. *Output* dimming pada LED terdiri dari nyala redup, nyala remang dan nyala terang.



Gambar 6.1 Pengujian Fungsional LED dan Sensor

Pengujian dilakukan dengan menggunakan perintah `gpio` pada *terminal* Raspbian seperti pada Gambar 6.1. Untuk mengatur mode pin digunakan perintah `gpio -g mode <no pin> <pin mode>`. Untuk menyalakan dan mematikan LED digunakan perintah `gpio -g write <no pin> <nilai>`. Sedangkan untuk memberi nilai pwm digunakan perintah `gpio -g pwm <no pin> <nilai pwm>`. Untuk membaca *input* sensor digunakan perintah `gpio -g read <no pin>`.



Gambar 6.2 Hasil Pengujian Fungsional LED dan Sensor

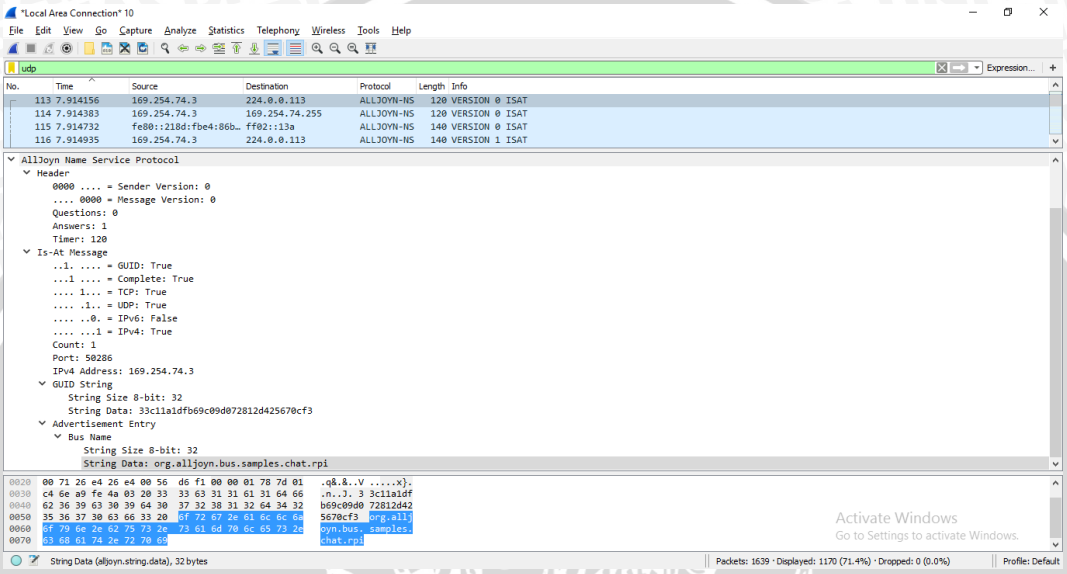
Pengujian menghasilkan LED dan Sensor berfungsi dengan baik seperti pada Gambar 6.2. No pin pada program juga sesuai dengan nyala LED yang diinginkan. Program yang dijalankan pada Raspberry Pi juga menyertakan kode untuk mengecek fungsionalitas LED dan sensor pada bagian awal program seperti pada Gambar 6.3. Sehingga fungsionalitas LED dan sensor dapat dilihat ketika menjalankan program.

```
pi@raspberrypi: ~/skripsi/alljoyn-15.04.00-src/build/linux/arm/debug/di
ples/chat $
pi@raspberrypi:~/skripsi/alljoyn-15.04.00-src/build
ples/chat $ sudo ./skripsi2 -j RaspberryPi
Set Pinmode 17 set 1
Set Pinmode 18 set 3
Set Pinmode 23 set 1
Set Pinmode 24 set 4
Set Pinmode 25 set 2
Set digitalWrite pin 17 acts 0
Set digitalWrite pin 17 acts 1
Set digitalWrite pin 17 acts 0
Set digitalWrite pin 17 acts 1
Set digitalWrite pin 17 acts 0
Set pwmWrite pin 18 acts 0
Set pwmWrite pin 18 acts 75
Set pwmWrite pin 18 acts 150
Set pwmWrite pin 18 acts 300
Set pwmWrite pin 18 acts 0
Set digitalWrite pin 23 acts 0
Set digitalWrite pin 23 acts 1
Set digitalWrite pin 23 acts 0
Set digitalWrite pin 23 acts 1
Set digitalWrite pin 23 acts 0
Set digitalWrite pin 24 acts 0
Set digitalWrite pin 24 acts 1
Set digitalWrite pin 24 acts 0
Set digitalWrite pin 24 acts 1
Set digitalWrite pin 24 acts 0
Sensor 25 digitalRead()
Sensor tidak mendeteksi objek
Sensor 25 digitalRead()
Sensor tidak mendeteksi objek
Sensor 25 digitalRead()
Sensor tidak mendeteksi objek
Sensor 25 digitalRead()
Sensor tidak mendeteksi objek
Sensor 25 digitalRead()
Sensor tidak mendeteksi objek
GPIO Sudah Diatur
```

Gambar 6.3 Pengujian Fungsional Pada Program

6.2 Pengujian Alljoyn *Service*

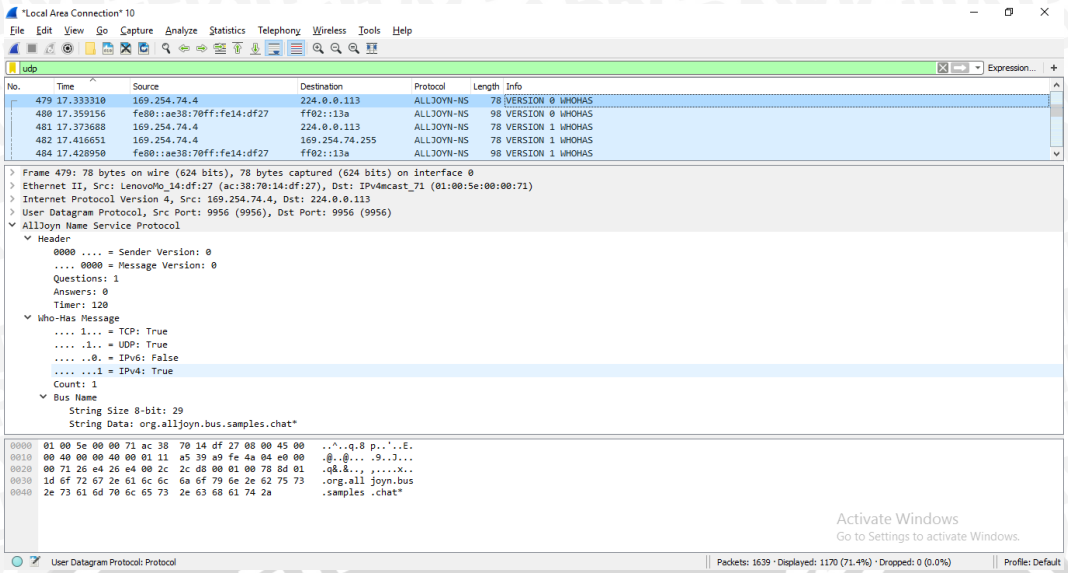
Pengujian ini bertujuan untuk memastikan *framework* Alljoyn berhasil diterapkan pada sistem dan dapat bekerja dengan baik. Pengujian dilakukan dengan menjalankan program Alljoyn pada Raspberry Pi. Kemudian dilakukan pengamatan menggunakan Wireshark pesan-pesan apa saja yang dikirim dan diterima dalam satu jaringan. Jika Alljoyn berhasil ditanamkan pada Raspberry Pi, maka akan terdapat protokol Alljoyn *Name Service* pada Wireshark yang melakukan *advertise service* yang dimiliki program.



Gambar 6.4 Pengujian Alljoyn Pada Sisi Raspberry Pi

Gambar 6.4 menunjukkan bahwa *service* org.alljoyn.bus.samples.chat.rpi diumumkan atau di *advertise* kepada alamat *broadcast* dengan menggunakan protokol Alljoyn *Name Service*. *Service* ini berada pada *bus name* org.alljoyn.bus.samples.chat. Pesan *advertise* yang dikirimkan adalah sebesar 32 bit String.

Kemudian pada sisi aplikasi. Ketika aplikasi dijalankan, maka aplikasi melakukan *discovery* pada jaringan adakah *service* yang berada pada *bus* yang sama. Hal ini dibuktikan dengan Gambar 6.5. Pada gambar tersebut aplikasi yang memiliki IP 169.254.74.4 melakukan *discovery* dengan menggunakan protokol Alljoyn Name Service dengan jenis pesan WHOHAS. Pesan ini dikirimkan ke alamat *broadcast*. Pesan ini menanyakan siapa yang memiliki *service* pada *bus name* org.alljoyn.bus.samples.chat. Ukuran dari pesan ini adalah 78 bit.



Gambar 6.5 Pengujian Alljoyn Pada Sisi Aplikasi

Dari kedua hasil di atas dapat disimpulkan pada Alljoyn telah diimplementasikan pada Raspberry Pi dan aplikasi pada *smartphone*. Fungsi Alljoyn yang diimplementasikan adalah fungsi *Advertisement* dan *Discovery Service* pada jaringan.

6.3 Pengujian Kontrol *Device* Melalui Aplikasi

Pengujian ini bertujuan untuk memastikan kontrol yang diberikan *user* melalui aplikasi sesuai dengan kontrol yang dilakukan oleh Raspberry Pi terhadap peralatan-peralatan yang terhubung pada GPIO. Pengujian juga dilakukan untuk memastikan bahwa *state* yang ditampilkan pada aplikasi sesuai dengan *state* pada peralatan.

Pengujian dilakukan dengan melakukan kontrol melalui aplikasi. Kemudian dilakukan pengecekan pada peralatan. Apakah *output* yang dihasilkan sudah sesuai atau belum. Jika *output* sesuai dengan kontrol, kemudian dilanjutkan dengan pengecekkan *state* peralatan yang ada pada kotak dialog *action*. Jika *state* sesuai maka sistem kontrol pada rumah cerdas dapat bekerja dengan baik.

Tabel 6.1 Pengujian Kontrol *Device*

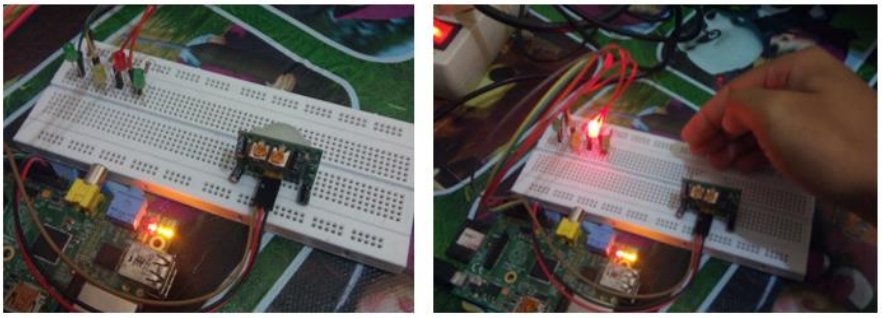
No	Kontrol	State Aplikasi	*Device* Terkontrol	Gambar 6.5
1	Matikan TV	TV Mati	Terkontrol	(a)
2	Nyalakan TV	TV Nyala	Terkontrol	(b)
3	Matikan lampu	Lampu Mati	Terkontrol	(c)
4	Redupkan lampu	Lampu Redup	Terkontrol	(d)
5	Remangkan lampu	Lampu Remang	Terkontrol	(e)
6	Terangkan lampu	Lampu Terang	Terkontrol	(f)
7	Matikan AC	AC Mati	Terkontrol	(g)
8	Nyalakan AC	AC Nyala	Terkontrol	(h)

Detail hasil pengujian dijelaskan pada Tabel 6.1. Pengujian ini menunjukkan bahwa kontrol dapat dilakukan melalui aplikasi. Hasil *output* pada peralatan sesuai dengan kontrol yang dilakukan pada aplikasi. *State* juga berubah sesuai dengan keadaan peralatan. Perubahan *output* peralatan terjadi bersamaan dengan kontrol yang dilakukan pada aplikasi. Hasil pengujian dibuktikan dengan Gambar 6.6.

Gambar 6.6 Pengujian Kontrol *Device* Melalui Aplikasi

6.4 Pengujian Raspberry Pi Mengirim Pesan *Event*

Pengujian ini bertujuan untuk memastikan Raspberry Pi dapat memberikan notifikasi pada aplikasi ketika peralatan *input* mengalami perubahan status. Dalam sistem ini Raspberry Pi harus dapat memberi informasi pada *user* ketika sensor PIR mendeteksi adanya obyek yang menyebabkan pintu terbuka (LED pintu menyala). Pengujian juga dilakukan untuk memastikan aplikasi dapat menampilkan pesan *event* dengan memunculkan dialog notifikasi. *State* peralatan pintu (LED pintu) juga harus berubah ketika aplikasi menerima pesan *event* dari Raspberry Pi.

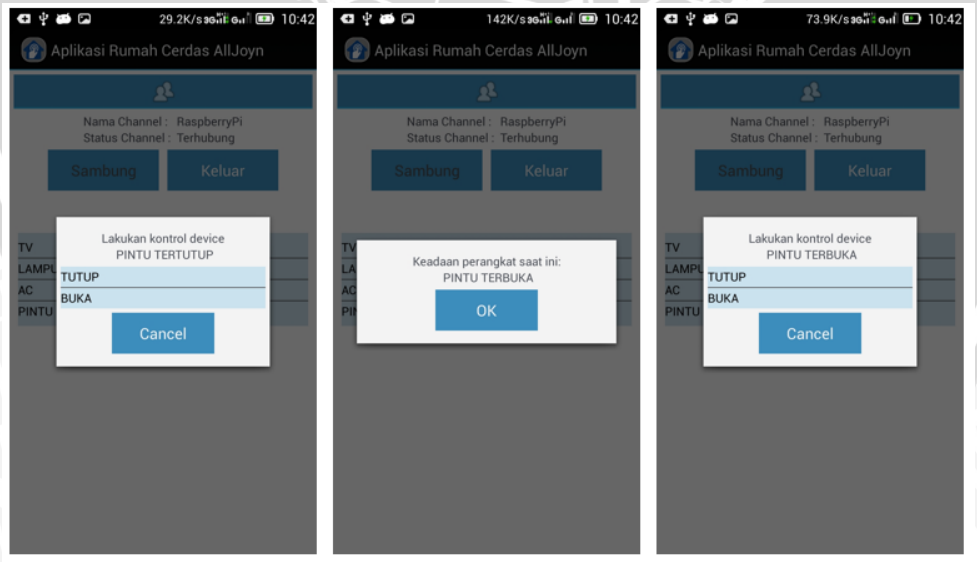


(a) (b)

Gambar 6.7 Pengujian *Event*

Pengujian dilakukan seperti pada Gambar 6.7. Pengujian dilakukan dengan melihat *state* dari pintu sebelum sensor diberi gerakan. Pintu merupakan peralatan yang *state*-nya bergantung pada *output* sensor. Kemudian sensor diberi gerakan obyek. Jika sistem berhasil, maka aplikasi akan menampilkan kotak dialog *event* yang menyatakan bahwa pintu terbuka. Kemudian perlu dilakukan pengecekkan pada kotak dialog *action* apakah *state* pintu sudah berubah atau belum. Jika *state* berubah menjadi pintu terbuka, maka sistem bekerja dengan baik.

Hasil pengujian seperti pada Gambar 6.8. Hasil pengujian menyatakan bahwa sistem dapat bekerja dengan baik. State dari pintu sesuai dengan keadaan deteksi sensor dan *output* LED indikator sensor. Ketika sensor tidak mendeteksi adanya obyek, *state* dari pintu adalah tertutup. Ketika sensor mendeteksi adanya obyek, muncul kotak dialog *event* dan *state* dari pintu menjadi terbuka.



(a) (b) (c)

Gambar 6.8 Hasil Pengujian *Event*

Hasil pengujian ditunjukkan seperti pada Tabel 6.2. Percobaan dilakukan sebanyak lima kali percobaan. Semua menunjukkan keberhasilan pengiriman informasi tentang keadaan pintu dan LED. Waktu respon yang dihasilkan adalah 1 detik dan 3 detik. Hal ini dikarenakan adanya delay dalam proses pembacaan sensor. Program memberi delay waktu 5 detik ketika sensor mendeteksi obyek dan 3 detik ketika sensor tidak mendeteksi obyek. Percobaan ketiga dilakukan ketika program masuk pada delay sensor tidak mendeteksi obyek sehingga waktu responnya menjadi bertambah.

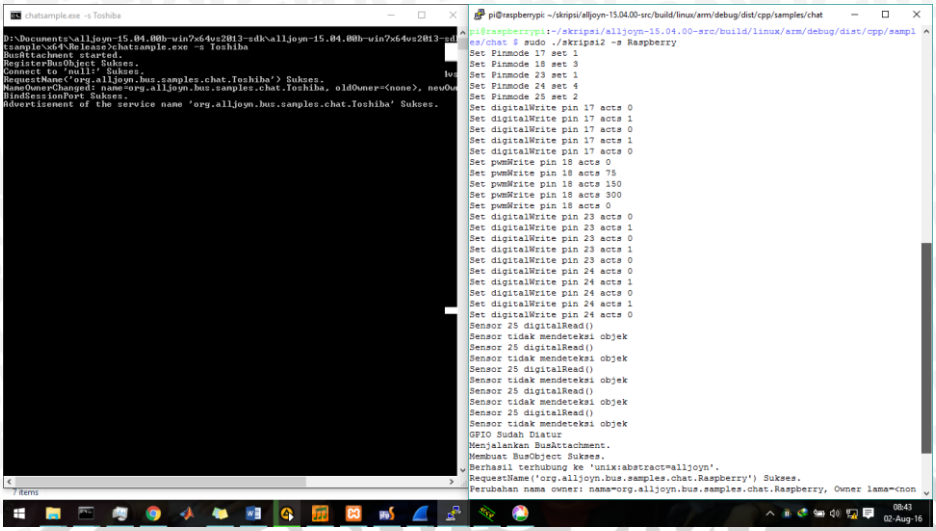
Tabel 6.2 Pengujian *Event*

Percobaan Ke	Dialog	State Aplikasi	LED	Waktu Respon (s)
1	Muncul	Pintu Terbuka	Nyala	1
2	Muncul	Pintu Terbuka	Nyala	1
3	Muncul	Pintu Terbuka	Nyala	3
4	Muncul	Pintu Terbuka	Nyala	3
5	Muncul	Pintu Terbuka	Nyala	1

6.5 Pengujian *Pervasive*

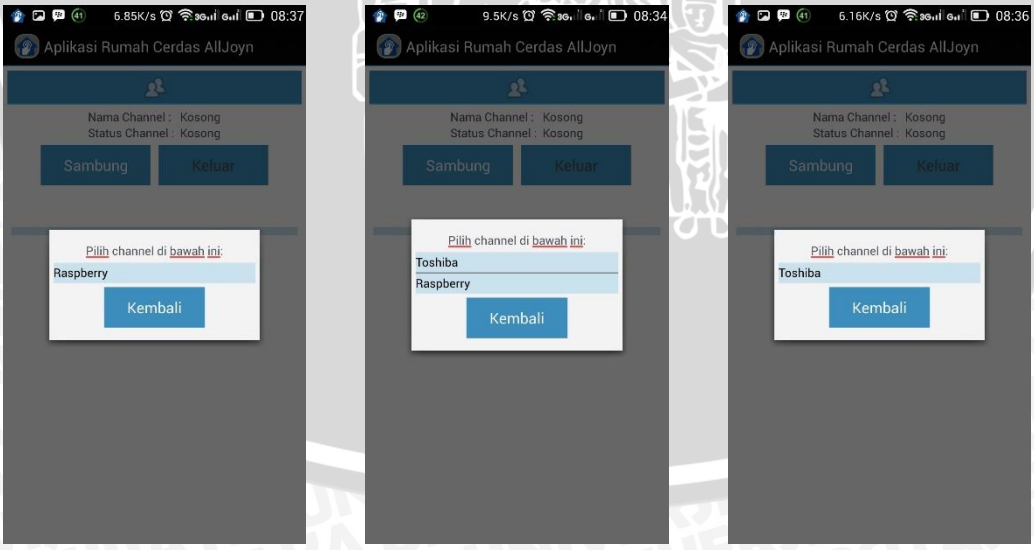
Pengujian ini dilakukan untuk memastikan sistem telah menerapkan konsep *pervasive*. Perangkat-perangkat yang menggunakan Alljoyn harus dapat berkomunikasi secara *pervasive*. Perangkat harus dapat mendeteksi perangkat lain yang berada pada satu jaringan. Jika terdapat perangkat baru bergabung pada satu jaringan maka perangkat lain harus dapat mengetahui informasi ini. Begitu juga sebaliknya, jika terdapat perangkat yang tidak lagi terhubung pada jaringan maka perangkat lain harus mengetahuinya.

Pengujian dilakukan dengan menggunakan tambahan perangkat yaitu Personal Computer (PC). PC akan bertindak seperti Rapsberry Pi yaitu sebagai *server* yang menyediakan *service* seperti pada Gambar 6.9. PC akan menjalankan program Alljoyn dengan menggunakan bahasa C. Aplikasi pada *smartphone* bertindak sebagai *client* harus dapat mendeteksi *availability* dari kedua service tersebut.



Gambar 6.9 Pengujian *Pervasive* Pada Raspberry Pi dan Laptop

Hasil pengujian menunjukkan ketika Raspberry Pi menjalankan service, maka aplikasi mendeteksi adanya channel Raspberry seperti pada Gambar 6.10 (a). Ketika PC menjalankan service Toshiba maka terdapat channel baru yang dideteksi oleh aplikasi Gambar 6.10 (b). Terdapat channel Toshiba pada channel list aplikasi. Ketika Raspberry Pi menghentikan service Alljoyn, maka channel yang dideteksi oleh aplikasi hanya channel Toshiba Gambar 6.10 (c). Pengujian ini menunjukkan ketika satu perangkat terputus atau terhubung maka informasi tentang perangkat tersebut akan terkirim ke perangkat lain. Sehingga hal ini menunjukkan bahwa sistem telah menerapakan konsep komunikasi perangkat secara *pervasive* dimana aplikasi dapat menerima informasi perangkat apa saja yang berada di dekatnya.



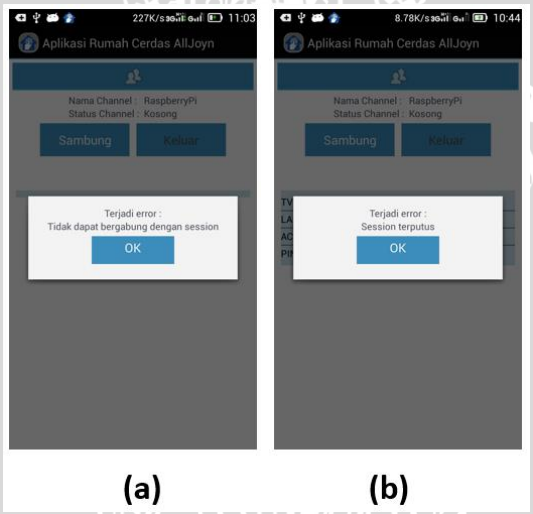
(a) (b) (c)

Gambar 6.10 Hasil Pengujian *Pervasive* Pada Aplikasi

6.6 Pengujian Kegagalan Alljoyn *Service*

Pengujian ini dilakukan untuk memastikan apa saja yang dapat menyebabkan *service* Alljoyn tidak dapat berjalan dengan baik. Pengujian dilakukan pada dua program yaitu program pada aplikasi dan program pada Raspberry Pi. Pengujian ini meliputi fungsi-fungsi Bus Listener yang ada pada Alljoyn. Pengujian dilakukan dengan melihat *output* yang dihasilkan ketika diberi dua *input* keadaan yang berbeda. Misalnya pengujian untuk memastikan fungsi Session Join Accept maka dilakukan pengujian dengan melakukan permintaan *join* ketika server aktif dan server tidak aktif.

Hasil pengujian digambarkan dengan Gambar 6.11. Gambar 6.11 (a) menunjukkan keadaan aplikasi yang menggunakan *session port* yang berbeda. Gambar 6.11 (b) menunjukkan keadaan aplikasi ketika server mati secara-tiba-tiba atau koneksi terputus. Aplikasi dapat mendeteksi adanya *channel* dalam jaringan, namun tidak dapat terhubung ke *channel* tersebut karena *session port* nya berbeda. Pengujian ini menunjukkan bahwa program dapat bekerja dengan baik jika server aktif, keadaan jaringan baik, dan parameter-parameter alljoyn pada aplikasi sama dengan yang ada pada Raspberry Pi.



Gambar 6.11 Pengujian Kegagalan Pada Alljoyn *Service*

BAB 7 PENUTUP

7.1 Kesimpulan

Berdasarkan rumusan masalah yang diangkat dan sistem telah melalui tahap perancangan, implementasi dan dilakukan pengujian dapat ditarik kesimpulan bahwa sistem ini telah berhasil memenuhi kebutuhan *user* beserta kebutuhan fungsionalnya. Berikut adalah hasil penelitian yang didapat:

1. Untuk merancang sistem peralatan rumah cerdas yang dapat dikenali secara *pervasive* dapat digunakan Alljoyn Framework. Alljoyn Framework membuat perangkat-perangkat dapat dideteksi secara *pervasive* menggunakan *advertisement* and *discovery service*. Advertisement dan discovery dilakukan dengan menggunakan *Alljoyn Name Service Protocol.*
2. Alljoyn Framework dapat diimplementasikan pada Raspberry Pi sebagai server peralatan rumah cerdas dan aplikasi pada *smartphone* Android sebagai *user interface*. Aplikasi mengirimkan pesan string untuk melakukan kontrol terhadap peralatan rumah cerdas. Raspberry Pi memberikan informasi tentang peralatan rumah cerdas kepada aplikasi. Informasi diberikan dengan prefix pesan yang berbeda-beda.
3. Alljoyn Framework mendukung komunikasi *multi-language* dan *multi-platform*. Raspberry Pi menggunakan Alljoyn Framework dengan platform sistem operasi Linux dan bahasa pemrograman C++. Aplikasi pada smartphone Android menggunakan Alljoyn Framework dengan platform sistem operasi Android dan bahasa pemrograman Java.
4. Pengujian fungsionalitas membuktikan sistem dapat mengenali perangkat secara *pervasive*. Pengujian kontrol peralatan rumah melalui aplikasi menunjukkan semua peralatan dapat dikontrol dan dapat diubah *state*-nya melalui aplikasi pada *smartphone*. Pengujian juga menunjukkan Raspberry Pi dapat mengirimkan informasi sesuai keadaan sensor kepada aplikasi pada *smartphone*.

7.2 Saran

Saran yang dapat diberikan untuk pengembangan penelitian ini antara lain :

1. Perlu dilakukan penelitian lebih lanjut tentang implementasi Alljoyn Service Framework Library.
2. Alljoyn Framework perlu diimplementasikan lebih dalam agar implementasi tidak hanya berproses pada pertukaran pesan.
3. Perlu dilakukan penelitian tentang implementasi Alljoyn Framework menggunakan *Multi Controller* dan *Multi Application*.
4. Perlu dilakukan penelitian tentang sistem rumah cerdas yang menggunakan database sebagai penyimpan data perlatan rumah tangga.

DAFTAR PUSTAKA

Aditya, F. G., 2015. *Analisis Dan Perancangan Prototype Smart Home Dengan Sistem Client Server Berbasis Platform Android Melalui Komunikasi Wireless,* Bandung: Telkom University.

Alayderous, A., 2004. *Rancang bangun dan analisa sistem rumah cerdas dengan penerapan konsep pervasive,* Jakarta: Perpustakaan Universitas Indonesia.

Alliance, A., 2015. *Alljoyn Architecture.* [Online] Tersedia: https://allseenalliance.org/framework/documentation/learn/architecture [Diakses pada 20 Januari 2016].

Alliance, A., 2015. *Alljoyn Learn.* [Online] Tersedia: https://allseenalliance.org/framework/documentation/learn [Diakses pada 20 Januari 2016].

Alliance, A., 2015. *Alljoyn Standard Core.* [Online] Tersedia: https://allseenalliance.org/framework/documentation/learn/core/standard-core [Diakses pada 15 Juli 2016].

Arena, F., 2014. *Fone Arena.* [Online] Tersedia: http://www.fonearena.com/lenovo-a859_4419.html [Diakses pada 10 July 2016].

Ebling, M. R., 2016. *Pervasive Computing and the Internet of Things,* s.l.: IEEE CS.

Ilham Megantara, M. T. B. S. N. A., 2014. *Makalah Rapberry Pi,* Bandung: Telkom University.

Jiehan Zhou, E. G. M. Y. J. R., 2010. *Pervasive Service Computing: Visions and Challenges.* Oulu, Finland, Computer Science and Engineering laboratory, University of Oulu.

Lamudi, 2014. *Lamudi Indonesia.* [Online] Tersedia: http://www.lamudi.co.id/journal/rumah-pintar-jadi-konsep-rumah-masa-depan/ [Diakses pada 20 Januari 2015].

Maryam, N. N., 2005. *Menuju konsep Pervasive pada rumah cerdas :: Studi kasus Pengontrol Akses Pintu,* Yogyakarta: Universitas Gadjah Mada.

Massimo Villari, A. C. M. F. A. P., 2014. *AllJoyn Lambda: an Architecture for the Management of Smart Environments in IoT.* Messina, Italy: DICIEAMA, University of Messina.

Onserda, 2013. *Android SDK.* [Online] Tersedia: http://www.saungit.org/2013/01/android-sdk.html [Diakses pada 20 Januari 2016].

Pi, R., 2015. *Gpio: Raspberry Pi Models A And B.* [Online] Tersedia: https://www.raspberrypi.org/documentation/usage/gpio [Diakses pada 26 Januari 2016].

Shop, G., 2013. *SDK (Software Development Kit).* [Online] Tersedia: http://gudanglinux.com/glossary/sdk-software-development-kit/ [Diakses pada 20 Januari 2016].

Sommerville, I., 2007. *Software Engineering.* 8th ed. China: Pearson Education Limited.

Spencer, B., 2012. *AllJoyn™ Overview and Integration Tips & Tricks,* s.l.: Qualcomm Innovation Center, Inc.

Tri Fajar Yurmama S, N. A., 2009. *Perancangan Software Aplikasi Pervasive Smart Home,* Yogyakarta: Universitas Nasional.