**SDRANK: *ADAPTABLE SERVICE SELECTION FOR INTERNET OF THINGS***

**DRAFT TESIS**

**BAB 1, 2, dan 3**

**Oleh**

**Deddy Christoper Kakunsi**

**NIM: 23516047**

**(Program Studi Magister Informatika)**

****

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**2017**

# LEMBAR PENGESAHAN

**SDRANK: *ADAPTABLE SERVICE SELECTION FOR INTERNET OF THINGS***

**DRAFT TESIS**

**BAB 1, 2, dan 3**

**Oleh**

**Deddy Christoper Kakunsi**

**23516047**

**Program Studi Magister Informatika**

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

Telah disetujui sebagai Draft Tesis Bab 1, 2, dan 3

di Bandung, pada …………………… 2017

|  |
| --- |
| **Pembimbing,** |
| Dr. Techn. Muhammad Zuhri Catur Candra ST., MT. NIP: 197709212010121002 |

# DAFTAR ISI

[LEMBAR PENGESAHAN i](#_Toc505196458)

[DAFTAR ISI iii](#_Toc505196459)

[DAFTAR TABEL iv](#_Toc505196460)

[DAFTAR GAMBAR v](#_Toc505196461)

[BAB I PENDAHULUAN 1](#_Toc505196462)

[I.1 Latar Belakang 1](#_Toc505196463)

[I.2 Rumusan Masalah 3](#_Toc505196464)

[I.3 Tujuan Penelitian 3](#_Toc505196465)

[I.4 Hipotesis 4](#_Toc505196466)

[BAB II STUDI LITERATUR 5](#_Toc505196467)

[II.1. Protokol *Internet of Things* 5](#_Toc505196468)

[II.2. *Service Discovery* pada *Web Services* 7](#_Toc505196469)

[II.3. *Service Discovery* pada *Internet of Things* 9](#_Toc505196470)

[II.4. *Service Ranking* 15](#_Toc505196471)

[II.5. *System Adaptability* 17](#_Toc505196472)

[BAB III ANALISIS DAN PERANCANGAN 21](#_Toc505196473)

[III.1. Analisis Sistem 21](#_Toc505196474)

[III.1.1. IoT Service Selection 21](#_Toc505196475)

[III.1.2. Kebutuhan Sistem 23](#_Toc505196476)

[III.2. Perancangan Sistem 25](#_Toc505196477)

[III.2.1. Rancangan Arsitektur Sistem 26](#_Toc505196478)

[III.2.2. Rancangan Behavioral Sistem 29](#_Toc505196479)

[III.2.3. Proses Ranking 32](#_Toc505196480)

[DAFTAR PUSTAKA 33](#_Toc505196481)

# DAFTAR TABEL

[Tabel 1. Perbandingan kualitatif protokol *service discovery* (Cabrera, Palade, & Clarke, 2017) 5](#_Toc505196482)

[Tabel 2. Perbandingan *service discovery* pada *web services* 8](#_Toc505196483)

[Tabel 3. Perbandingan *service discovery* pada IoT 14](#_Toc505196484)

[Tabel 4. Daftar aktor yang berinteraksi dengan SDRank 23](#_Toc505196485)

[Tabel 5. Kebutuhan fungsional SDRank 24](#_Toc505196486)

[Tabel 6. Kebutuhan non-fungsional SDRank 25](#_Toc505196487)

[Tabel 7. Komponen sistem 28](#_Toc505196488)

# DAFTAR GAMBAR

[Gambar 1. Pendekatan *clustering* untuk *service discovery* (Elhaster, 2016) 10](#_Toc505196489)

[Gambar 2. Arsitektur Digcovery (Jara, dkk. 2013). 10](#_Toc505196490)

[Gambar 3. Arsitektur resource discovery (Datta, 2015) 11](#_Toc505196491)

[Gambar 4. Arsitektur P2P perangkat IoT (Liu, 2013) 12](#_Toc505196492)

[Gambar 5. Mekanisme *service discovery* pada arsitektur P2P (Cirani, 2014) 13](#_Toc505196493)

[Gambar 6. Arsitektur WSExpress (Zhang, 2010) 16](#_Toc505196494)

[Gambar 7. Fungsionalitas *autonomic manager* (IBM, 2005) 17](#_Toc505196495)

[Gambar 8. *Service advertisement* dari sisi *gateway* (Cirani, 2014) 19](#_Toc505196496)

[Gambar 9. *Service advertisement* dari sisi perangkat (Cirani, 2014) 20](#_Toc505196497)

[Gambar 10. *IoT Service Model* berdasarkan *Physical Service Model* 22](#_Toc505196498)

[Gambar 11. Proses *service selection* 23](#_Toc505196499)

[Gambar 12. Diagram *Use Case* SDRank 26](#_Toc505196500)

[Gambar 13. Arsitektur SDRank 27](#_Toc505196501)

[Gambar 14. Proses *service selection* 29](#_Toc505196502)

[Gambar 15. Proses *adaptability* untuk penambahan perangkat 30](#_Toc505196503)

[Gambar 16. Proses *adaptability* untuk kegagalan perangkat 31](#_Toc505196504)

# BAB I PENDAHULUAN

## I.1 Latar Belakang

*Internet of things* (IoT) adalah interkoneksi antar obyek dalam suatu jaringan (Xia, Yang, Wang, & Finel, 2012). Penerapan IoT memungkinkan integrasi antara dunia nyata dan dunia digital melalui jaringan internet. Perangkat-perangkat yang memiliki kemampuan komputasi dapat diatur untuk memperoleh data dari obyek tertentu (*sensor*) dan bereaksi terhadap obyek tersebut (*actuator*). Data yang diperoleh dapat diakses oleh pengguna melalui jaringan internet, sehingga memungkinkan pemantauan dan pengendalian obyek dari jarak jauh. Perkembangan teknologi komunikasi memungkinkan interaksi antar perangkat IoT melalui jaringan yang lemah, sehingga berdampak positif terhadap pemanfaatan IoT. Saat ini, IoT dimanfaatkan pada bidang transportasi, *building automation*, *smart home*, *smart city*, dan bidang-bidang lainnya. Tingkat penerapan IoT terus meningkat dan diperkirakan akan mencapai jumlah 50 miliar perangkat pada tahun 2020 (Vandana & Chikkamannur, 2016).

Ketersediaan perangkat dalam jumlah besar membutuhkan mekanisme dan metode pengelolaan yang efektif dan efisien. Sehingga, keberadaan suatu perangkat harus dapat diketahui dan digunakan. Selain itu, layanan yang disediakan harus tersedia dan dapat diakses, sehingga memudahkan pengguna dalam berkomunikasi dengan perangkat bahkan dalam skenario komunikasi antar mesin yang minim keterlibatan manusia.

Banyaknya jumlah perangkat yang tersedia memungkinkan adanya sejumlah perangkat dengan kemampuan yang mirip, meskipun berbeda domain maupun aplikasinya. Kemiripan tersebut memberikan pilihan perangkat yang dapat digunakan, tergantung pada relevansi dan kualitas perangkat terhadap kebutuhan. Kemiripan antar perangkat dapat ditinjau dari data dan operasi yang disediakan, lokasi perangkat, dll. Sejumlah perangkat dengan data yang sama mungkin berdekatan secara geografis, namun di antara pernagkat-perangkat tersebut ada salah satu yang paling relevan untuk kebutuhan tertentu. Pada penerapannya, pengguna tidak memiliki akses terhadap semua layanan perangkat IoT yang tersedia. Sebuah penelitian (Vandana & Chikkamannur, 2016) menyatakan bahwa saat ini dibutuhkan mekanisme untuk memilih layanan perangkat IoT berdasarkan urutan relevansinya terhadap kebutuhan pengguna.

Ketersediaan perangkat IoT bersifat dinamis (Ngu, Gutierrez, Metsis, Nepal, & Sheng, 2016). Hal ini disebabkan oleh beberapa faktor antara lain kemampuan komputasi, sumber energi, dan jaringan komunikasi yang terbatas. Perangkat IoT tersedia pada waktu tertentu namun tidak tersedia pada waktu yang lain, dan sebaliknya. Selain itu, keadaan perangkat IoT dapat berubah setiap waktu. Perangkat yang sudah ada dapat mengalami kegagalan yang disebabkan oleh kerusakan maupun perawatan, serta perangkat baru dapat ditambahkan ke dalam jaringan IoT. Hal tersebut menyebabkan konfigurasi jaringan IoT dapat berubah sewaktu-waktu. Akibatnya, pengguna mungkin saja menggunakan layanan yang kurang relevan, sementara tersedia layanan dengan relevansi yang lebih baik untuk kebutuhannya.

Domain *software engineering* mengelompokkan *adaptability* ke dalam kebutuhan non-fungsional (Elhaster, Elgazzar, & Mertin, 2016). Secara umum, *adaptability* adalah kemampuan sistem untuk menyesuaikan diri terhadap perubahan pada lingkungan operasinya. Terkait dengan ketersediaan perangkat IoT yang dinamis, dibutuhkan mekanisme pemilihan layanan yang mampu beradaptasi terhadap perubahan ketersediaan pada perangkat IoT. Sehingga pengaruhnya terhadap relevansi layanan dalam jaringan perangkat IoT dapat diidentifikasi. Apabila tingkat relevansi berubah, pengguna dapat menerima pemberitahuan terkait perubahan tersebut.

Mekanisme pemilihan layanan untuk perangkat IoT yang ada saat ini belum menyediakan kemampuan untuk menangani permasalahan di atas, yaitu pemilihan layanan berdasarkan relevansi terhadap kebutuhan pengguna, serta adaptasi terhadap perubahan ketersediaan perangkat IoT pada lingkungan dengan jumlah perangkat yang banyak dan terus bertambah.

## I.2 Rumusan Masalah

Permasalahan yang dihadapi dalam penerapan IoT saat ini berkaitan dengan jumlah perangkat yang tersedia. Jumlah perangkat yang banyak mengakibatkan penentuan layanan yang paling sesuai untuk suatu kebutuhan menjadi rumit, karena pengguna tidak memiliki akses terhadap semua perangkat. Dibutuhkan suatu mekanisme yang memungkinkan pencarian perangkat secara otomatis berdasarkan kebutuhan pengguna.

Jumlah perangkat IoT yang tersedia akan terus bertambah, dan beberapa perangkat dapat mengalami kegagalan atau diganti dengan perangkat lain. Hal tersebut berpengaruh terhadap urutan layanan yang pernah dibuat sebelumnya. Dalam kasus penambahan perangkat, relevansi layanan yang baru harus dihitung untuk menentukan urutan relevansinya. Sehingga, diperlukan suatu mekanisme adaptasi yang mampu menjamin pengguna memperoleh data dari layanan yang paling relevan. Sedangkan dalam kasus pengurangan perangkat karena kegagalan maupun penggantian, sistem harus mampu menyediakan layanan alternatif sebagai sumber data.

**Permasalahan.** Bagaimana merancang dan membangun sistem yang mampu menentukan layanan yang paling relevan untuk kebutuhan tertentu dan dapat beradaptasi terhadap perubahan ketersediaan perangkat?

## I.3 Tujuan Penelitian

Tujuan penelitian ini adalah menghasilkan sistem IoT, beserta penerapan dan evaluasinya, yang menyediakan mekanisme pemilihan layanan (*service selection)*. Sistem yang diusulkan mampu mengidentifikasi perangkat-perangkat dengan layanan yang relevan terhadap kebutuhan tertentu, serta mengurutkannya berdasarkan kualitasnya. Selain itu, sistem mampu mengidentifikasi perubahan ketersediaan perangkat, serta dampaknya terhadap urutan relevansi layanan yang sudah dibuat sebelumnya. Kinerja sistem akan diukur berdasarkan *response time* ketika menggunakan *ranking* dibandingkan dengan tidak menggunakan *ranking*, baik untuk kemampuan *adaptability* maupun *reusability*.

## I.4 Hipotesis

Permasalahan *service selection* telah menjadi masalah utama dalam lingkungan *web services*. *Service selection* terkait erat dengan kemampuan *service discovery* dan *service rating*. Pertama untuk menemukan layanan yang relevan dan yang kedua untuk mengurutkan layanan tersebut berdasarkan relevansinya. Beberapa pendekatan *search engine* telah diterapkan sebagai solusi *service discovery* pada *web services* (Ngu, Gutierrez, Metsis, Nepal, & Sheng, 2016). Rating menggunakan aspek non-fungsional dapat digunakan untuk mengurutkan layanan berdasarkan kebutuhan pengguna (Jin, Chun, Jung, & Lee, 2014). Selain itu, pendekatan *Zero Configuration* telah digunaan untuk *self-configuring network*, sehingga perangkat baru dapat diidentifikasi (Cirani, et al., 2014). Oleh karena itu, penerapan *service selection* dalam domain IoT yang mampu beradaptasi terhadap perubahan dapat dikembangkan. Hasil penelitian ini mampu meningkatkan *usability* dari perangkat IoT, terutama pencarian layanan secara otomatis. Sehingga, data yang digunakan untuk kebutuhan pengguna adalah data dari perangkat yang paling relevan yang tersedia.

# BAB II STUDI LITERATUR

## II.1. Protokol *Internet of Things*

*Internet of Things* (IoT) dapat memberikan *real world service* untuk berbagai domain aplikasi. Dengan jumlah ketersediaan perangkat yang sangat banyak IoT memberikan pilihan layanan yang bervariasi. Sehingga dibutuhkan mekanisme *service discovery* perangkat IoT untuk memudahkan proses pencarian layanan. Beberapa mekanisme telah diajukan sebagai bagian dari protokol komunikasi IoT. Sebuah penelitian (Cabrera, Palade, & Clarke, 2017) mencoba mengevaluasi protokol *service discovery* yaitu, CoAP-SD, DNS-SD, mDNS-SD, dan DDS-SD. Evaluasi dilakukan secara kualitatif dan kuantitatif, baik menggunakan arsitektur terpusat maupun terdistribusi. Hasil evaluasi pada data heterogen membuktikan bahwa presisi pencarian tidak lebih dari 50% dan pada pencarian recall 20% - 30% layanan yang relevan tidak diambil. Hal ini disebabkan karena metode *discovery* menggunakan *exact string matching*. Selain itu, beberapa protokol menunjukkan masalah skalabilitas, karena pencarian dilakukan terhadap seluruh isi register. Perbandingan kualitatif antar protokol disajikan pada tabel 1.

Tabel . Perbandingan kualitatif protokol *service discovery* (Cabrera, Palade, & Clarke, 2017)

| **Metrik Kualitatif** | | **Protokol** | | | |
| --- | --- | --- | --- | --- | --- |
| **Fitur** | **Metrik** | **CoAP-SD** | **DNS-SD** | **mDNS-SD** | **DDS-SD** |
| *Hetero*-*geneity* | Interopera-bilitas dengan protokol lain | Dapat digunakan bersama dengan DNS-SD, mDNS-SD, dan HTTP | Dapat digunakan Bersama dengan mDNS-SD | Dapat digunakan Bersama dengan DNS-SD | Tidak interoperable |
| *Security* | Mekanisme keamanan yang disediakan | Menggu-nakan Datagram Transport Layer Security (DTLS) | Tidak tersedia | Tidak tersedia | Tersedia versi beta (OMG) |
| *QoS Support* | Penyertaan parameter QoS dalam deskripsi layanan | Dapat disertakan pada deskripsi layanan | Dapat dideskrip-sikan pada file TXT | Dapat dideskrip-sikan pada file TXT | Beberapa parameter disediakan oleh DDS |
| Penggunaan parameter QoS saat pencarian layanan | Digunakan sebagai parameter pencarian | Tidak dapat digunakan | Tidak dapat digunakan | Digunakan untuk memban-dingkan 2 partisipan |
| *Context Management* | Dukungan terhadap manajemen konteks dalam deskripsi layanan | Dapat disertakan pada deskripsi layanan | Dapat dideskrip-sikan pada file TXT | Dapat dideskrip-sikan pada file TXT | Tidak tersedia |
| Pemanfaatan informasi konteks saat pencarian layanan | Dapat disertakan sebagai parameter pencarian | Tidak dapat digunakan | Tidak dapat digunakan | Tidak tersedia |

## II.2. *Service Discovery* pada *Web Services*

*Service discovery* telah menjadi permasalahan pada domain *Web Services* sejak lama. Pada lingkungan sistem berbasis layanan, *Web Services* adalah bagian yang paling krusial. Pemanfaatan *Web Services* menjamin skalabilitas sistem dan penggunaan kembali fungsionalitas sistem. Hal ini menyebabkan banyak layanan dengan fungsionalitas yang sama telah tersedia. Oleh karena itu, beberapa mekanisme diajukan untuk menentukan layanan berdasarkan relevansi, salah satunya dengan menggunakan *search engine*. Beberapa penelitian yang dilakukan terkait dengan pemanfaatan s*earch engine* pada proses *service discovery* antara lain Woogle (Dong, Haley, Madhavan, Nemes, & Zhang, 2004), Titan (Wu, Chen, Xie, & Zheng, 2012), WSExpress (Zhang, Zheng, & Lyu, 2010), dan ServiceXplorer (Ngu, Ma, Sheng, Yao, & Julian, 2014). Dalam penerapannya Woogle dan Titan menggunakan pendekatan *clustering*, WSExpress menggunakan akumulasi kemiripan fungsional dan non-fungsional, sedangkan ServiceXplorer memanfaatkan *indexing* basis data serta algoritma *Earth Mover’s Distance* (EMD). Perbandingan layanan *service discovery* disajikan pada tabel 2.

*Service clustering* adalah pengelompokkan *Web Services* berdasarkan kemiripan karakteristik, sehingga layanan-layanan yang mirip berada pada *cluster* yang sama. Woogle adalah *search engine* untuk proses *service discovery* berdasarkan kemiripan layanan. Metode yang diusulkan dalam Woogle mampu mengelompokkan nama parameter pada operasi *web services* menjadi konsep yang mengandung makna *semantic*, sehingga mudah digunakan untuk menemukan kemiripan antar operasi berdasarkan parameter *input* dan *output,* serta menentukan kemiripan antar operasi pada *web services*. *Clustering* yang digunakan adalah pengembangan dari *agglomerative clustering*, yang menggabungkan setiap *term* berdasarkan tingkat asosiasi antar *term*. Kemiripan antar *web service* ditentukan oleh kemiripan antara parameter *input/output*, dan *operation* yang ditawarkan. Kemiripan tersebut dihitung dengan pendekatan *Term Frequency – Inverse Document Frequency* (TF-IDF).

Titan memanfaatkan *clustering* sebagai bagian utama dari solusi yang diusulkan, sehingga komponen *search engine* dapat melakukan pencarian pada *cluster* yang relevan. Dalam proses pembentukan *cluster*, Titan menggunakan pendekatan *Information Retrieval* untuk memperoleh karakteristik layanan berupa *content*, *type*, *message*, *port*, dan *name*. Karakteristik tersebut berguna dalam penentuan kemiripan antar layanan (*feature-based similarity*). Untuk meningkatkan relevansi layanan, Titan menggunakan *tag* pada *web service*. *Tag* ditentukan berdasarkan kata kunci yang paling sering muncul dalam deskripsi *web service* menggunakan TF-IDF. Penentuan *cluster* ditetapkan berdasarkan kemiripan *tag* pada setiap *web service* (*tag-based similarity*). Penentuan posisi layanan terhadap *cluster* ditentukan berdasarkan *global similarity*, yaitu akumulasi dari *feature-based similarity* dan *tag-based similarity*.

WSExpress berusahan menemukan layanan yang sesuai dengan kebutuhan pengguna berdasarkan kemiripan fungsional dan non-fungsional (kualitas) *web services* dan *query*. Web services mungkin saja mirip secara fungsional, namun dapat berbeda secara kualitas. Bagian fungsional *web services* ditentukan oleh vektor Rf = (rk, rin, rout). Kemiripan ditentukan berdasarkan *cosine similarity* antara vektor Rf terhadap *query* dan vektor Rf terhadap *web services*. Sedangkan vektor Rq = (C, W) menentukan atribut kualitas *web services* dan *query*. Sehingga kemiripan antara *web services* dan *query* secara keseluruhan adalah R = (Rf, Rq).

Tabel . Perbandingan *service discovery* pada *web services*

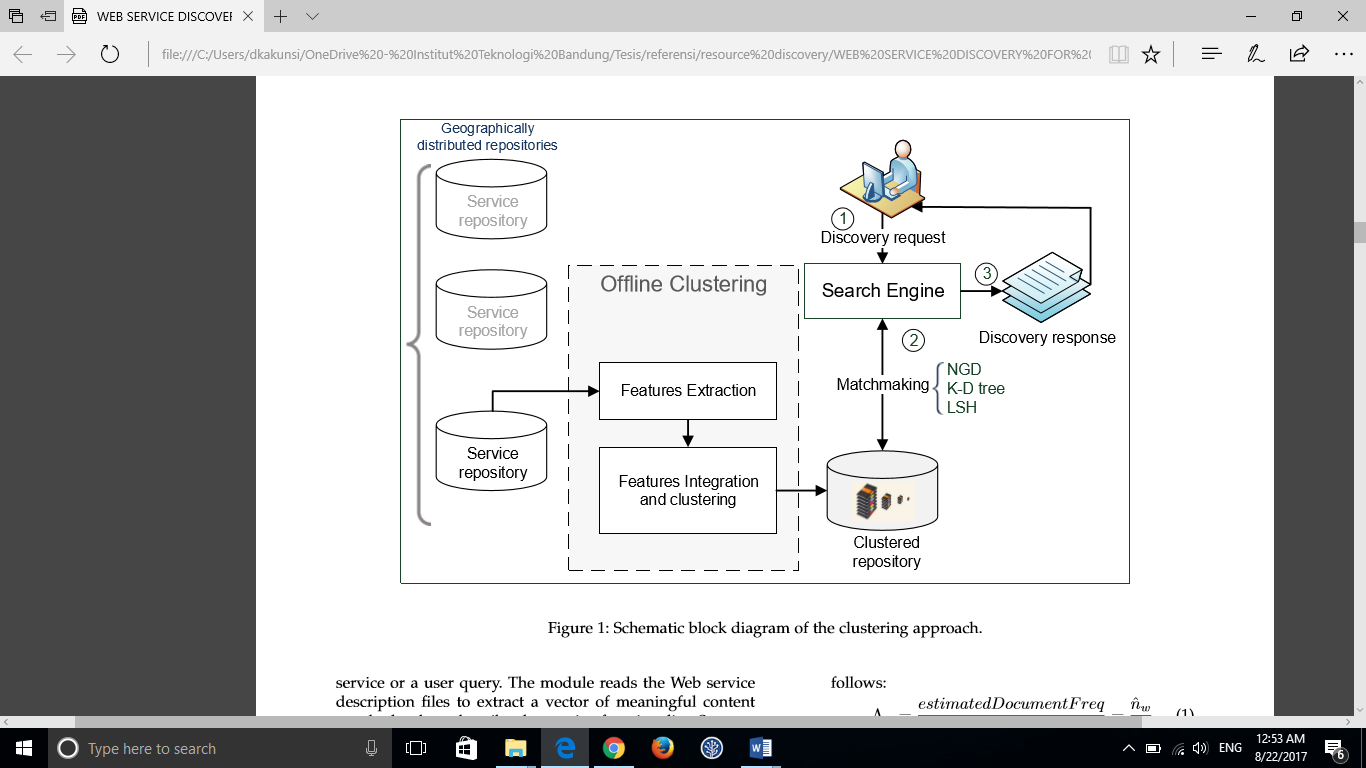
| **Fitur** | **Woogle** | **Titan** | **WSExpress** | **ServiceXplorer** |
| --- | --- | --- | --- | --- |
| *Search* *strategy* | Pencarian kata kunciberdasarkan kelompok parameter | Kombinasi kata kunci & *tag* | Kombinasi kata kunci & atribut kualitas | Kesesuaian parsial |
| *Similarity lev*el | Operasi & nama parameter | Fitur & *tag* | Fitur | Fitur |
| *Similarity* *measure* | *Cosine similarity* | *Cosine similarity* | *Cosine similarity* | EMD *similarity* |
| *Quality* *attribute* | Tidak tersedia | Tidak tersedia | Digunakan untuk mengurutkan layanan | Tersedia dalam basis data |

ServiceXplorer mencoba menyelesaikan permasalahan *service discovery* dengan menghindari perbandingan kata kunci (*exact keyword matching*), karena beberapa *term* berbeda namun memiliki konsep yang sama tidak akan ditemukan. Pendekatan yang digunakan adalah EMD *similarity*, yang dapat menemukan kesesuaian parsial antara *web services* dan *query*. Informasi layanan diekstrak dari dokumen WSDL dan disimpan di dalam basis data untuk memudahkan proses pencarian kemiripan. Selain itu, indeksbasis data dapat digunakan menyaring layanan yang akan diproses oleh EMD dan mengurutkan atribut kualitas.

## II.3. *Service Discovery* pada *Internet of Things*

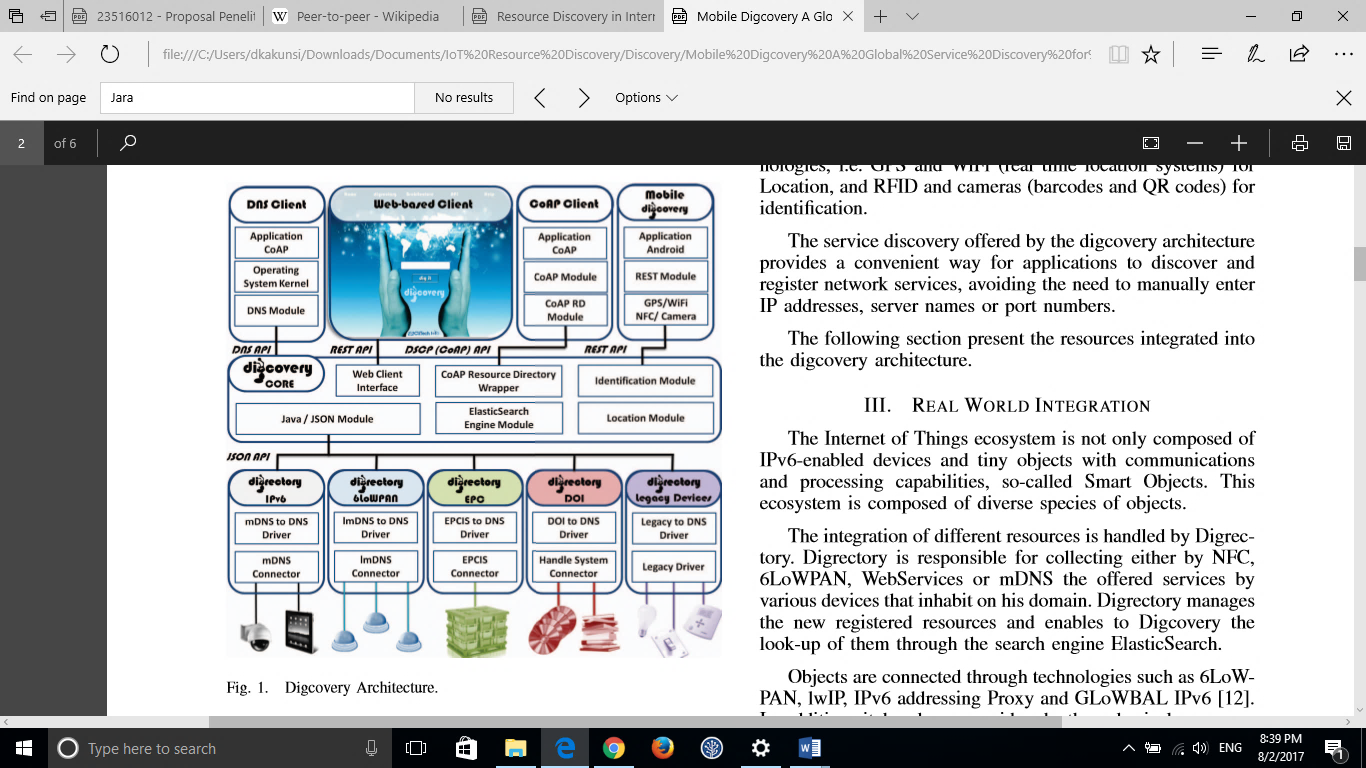
Perangkat IoT menyediakan *resource* melalui *service*, sehingga service discovery menjadi salah satu permasalahan. Selain itu, jumlah yang banyak memungkinkan lebih banyak layanan memiliki karakteristik yang mirip. Pendekatan *service discovery* telah diusulkan dalam beberapa penelitian, antara lain dengan memanfaatkan *search engine*, dan P2P *overlay* untuk menjamin skalabilitas.

Elhaster, dkk. memanfaatkan *clustering* untuk mendapatkan performa yang lebih efisien dengan hasil yang akurat. *Clustering* digunakan untuk mengelompokkan layanan berdasarkan fungsionalitas yang ditawarkan. Fungsionalitas layanan dapat didefinisikan berdasarkan karakteristik yang diekstrak dari dokumen WSDL, yakni *content word*, *operations*, *messages*, *types*, dan *name*. Sistem yang diajukan menggunakan sejumlah *service repository* untuk menampung deskripsi layanan, berdasarkan lokasi *repository* terdekat (gambar 1). *Cluster* layanan dibentuk pada setiap *repository* secara *offline* untuk mengurangi waktu tunggu saat pengguna mengirimkan *request*. Dengan adanya *cluster*, proses *discovery* dapat dibatasi pada *cluster* yang relevan terhadap *request*. Sehingga *search space* lebih kecil dan meningkatkan efisiensi *discovery*.



Gambar . Pendekatan *clustering* untuk *service discovery* (Elhaster, 2016)

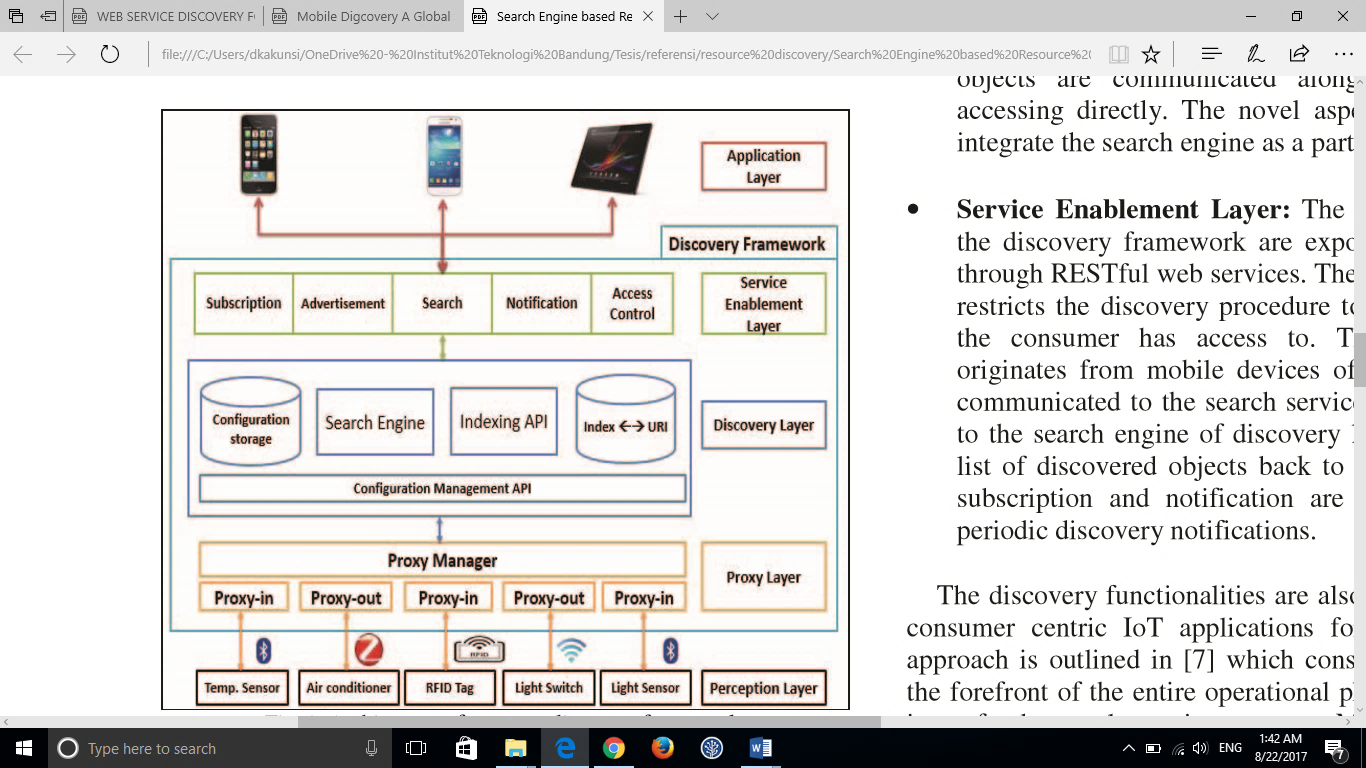
Proses pencarian *web service* dilakukan terhadap *repository* yang sudah terbagi dalam *cluster* melalui *search engine*. Seperti *search engine* pada umumnya, pencarian melibatkan proses *indexing* untuk memaksimalkan proses pencarian dari segi efisiensi dan akurasi. Proses *matchmaking* dilakukan dilakukan pada saat *runtime* menggunakan mekanisme *tree* dan *bruteforce*. Pemanfaatan sejumlah *repository* memungkinkan distribusi *cluster*, sehingga penambahan jumlah *service* tidak akan mempengaruhi proses *discovery* secara signifikan.



Gambar . Arsitektur Digcovery (Jara, dkk. 2013).

Sebuah penelitian (Jara, et al., 2013) memperkenalkan arsitektur Digcovery untuk menangani proses *service discovery* secara global. Dalam arsitekturnya (gambar 2), Jara memperkenalkan komponen yang disebut Digrectory, yaitu sebuah *directory* layanan yang dapat bertindak sebagai *proxy* terhadap obyek. Digrectory menangani proses integrasi layanan ke dalam sistem dan menyimpan deskripsi serta mempublikasikan layanan sehingga dapat ditemukan melalui *search engine*. Penerapan Digrectory memberikan skalabilitas yang baik, karena deskripsi layanan terdistribusi pada setiap Digrectory.

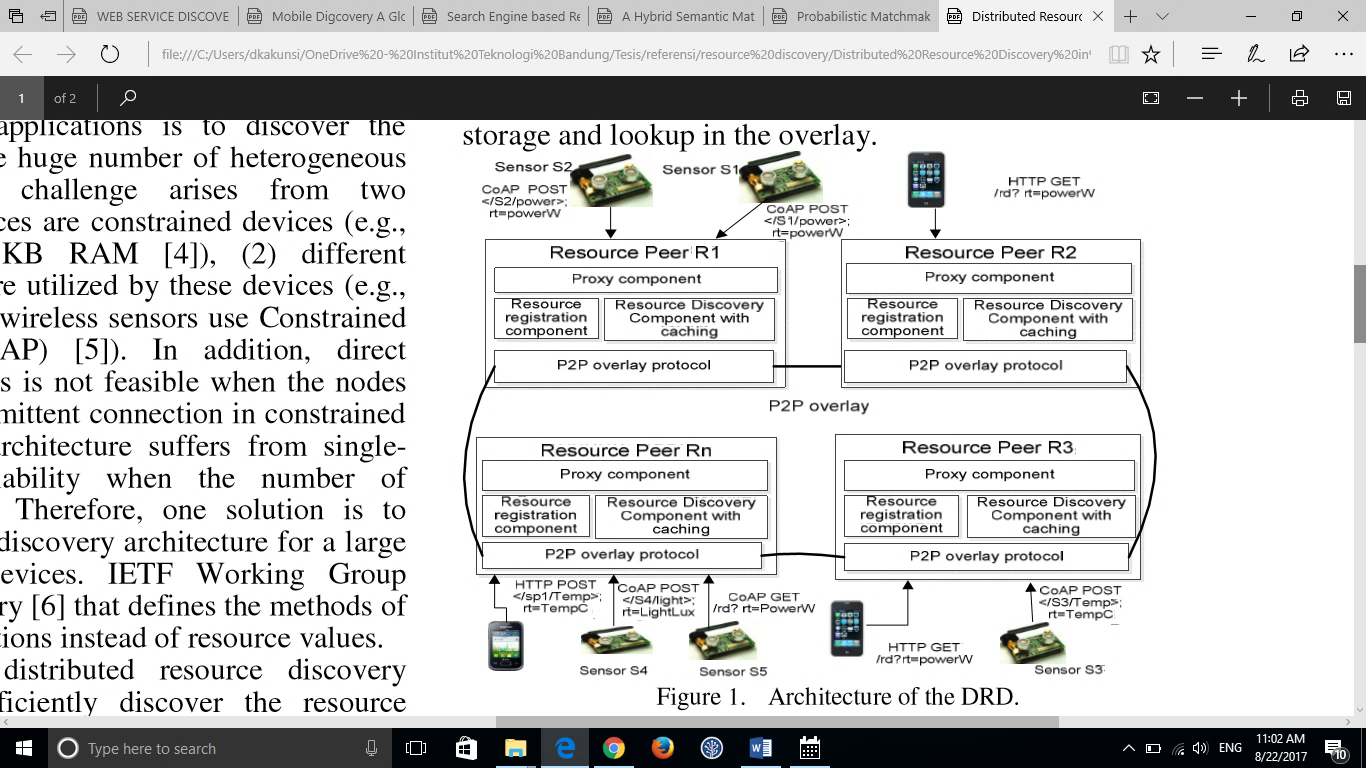
Penelitian lainnya (Datta & Bonnet, 2015) mencoba menerapkan *serach engine* untuk framework *service discovery* pada platform IoT. Arsitektur sistem ditunjukkan pada gambar 3. Pencarian dilakukan terhadap deskripsi layanan yang tersimpan dalam *registry* terpusat dan telah diindeks. Arsitektur menggunakan *layer* *proxy* untuk mengintegrasikan obyek yang berbeda-beda ke dalam sistem. Penggunaan *proxy* dan Digcovery memungkinkan integrasi perangkat-perangkat yang berbeda ke dalam sistem, serta menjamin skalabilitas dan *robustness* sistem karena deskripsi dan akses layanan terdistribusi pada beberapa komponen.



Gambar . Arsitektur resource discovery (Datta, 2015)

Penerapan *service discovery* berbasis *search engine* memaksa akses pengguna melalui satu sumber saja, sehingga mengharuskan penggunaan solusi terpusat. Salah satu keterbatasan solusi terpusat adalah *single point of failure* (Liu, et al., 2013). Sebuah kesalahan pada komponen sistem akan berakibat pada keseluruhan sistem.

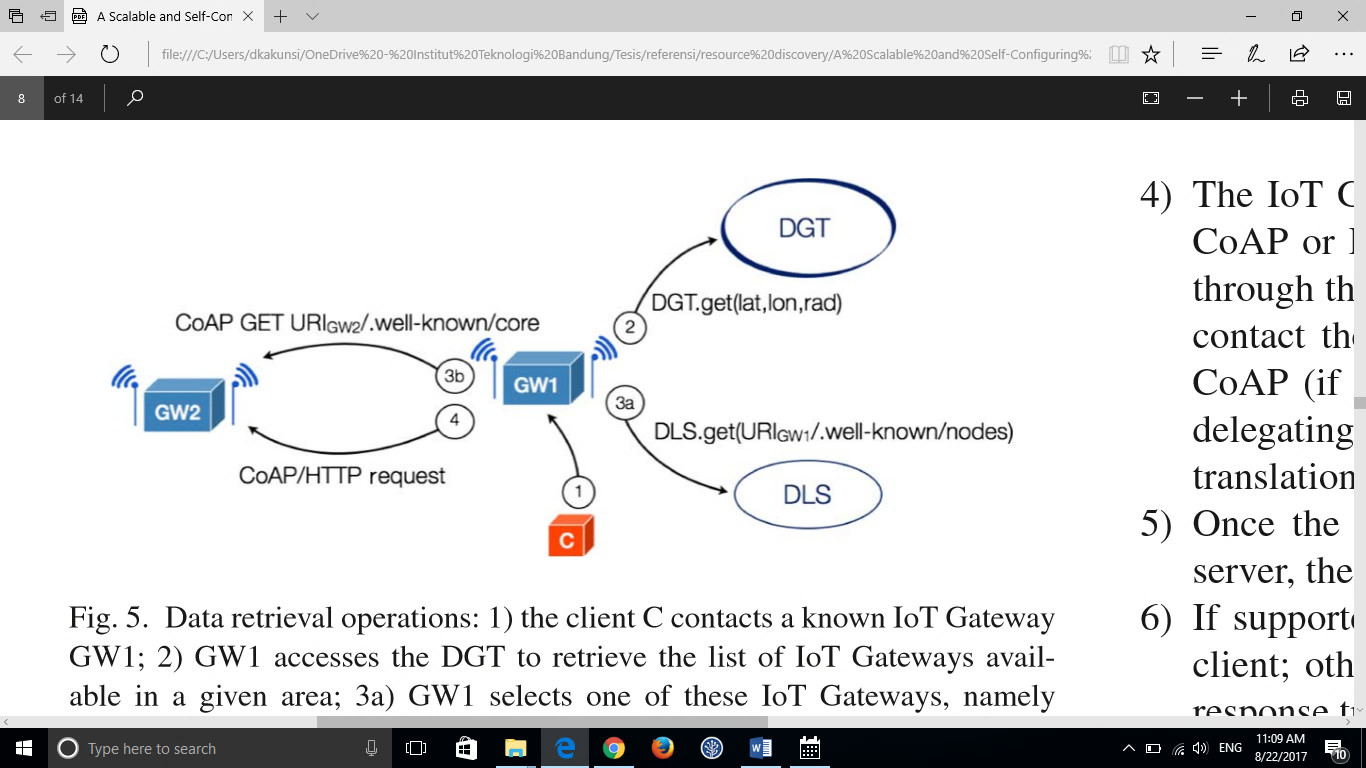
Beberapa penelitian dalam domain IoT telah dilakukan untuk mengelola perangkat secara terdistribusi, baik untuk melayani *end user* maupun perangkat lainnya dalam skenario *machine-to-machine* (M2M). Liu, dkk. (2013) mengusulkan solusi terdistribusi untuk proses *service discovery* pada skenario M2M. Sistem dibangun di atas arsitektur *Peer-to-Peer* (P2P) *overlay*, dan masing-masing *peer* mengelola perangkat-perangkat IoT yang terdaftar (gambar 4). Setiap *peer* mengelola proses registrasi dan identifikasi layanan dari perangkat yang dikelola, serta melakukan *discovery* layanan lain yang berada pada *peer* berbeda. Selain itu, *peer* bertindak sebagai *proxy* antara pengguna dan perangkat yang menyediakan data.



Gambar . Arsitektur P2P perangkat IoT (Liu, 2013)

Pendekatan terdistribusi menggunakan P2P *overlay* dapat meningkatkan skalabilitas sistem (Cirani, et al., 2014). Arsitektur yang diusulkan mampu menangani jumlah perangkat yang besar dan tersebar secara geografis. Pada penerapannya, setiap *peer* bertindak sebagai *gateway* terhadap perangkat-perangkat yang dikelola. *Peer* dapat melakukan translasi protokol sehingga dapat diakses oleh semua pengguna, baik menggunakan CoAP maupun HTTP. Selain menyimpan deskripsi layanan, *peer* juga bertanggung jawab mempublikasikan deskripsi pada jaringan P2P *overlay* menggunakan *Distributed Location Service* (DLS), sehingga dapat diakses oleh *peer* yang lain. *Peer* mengelola perangkat yang berada pada lokasi geografis yang sama. Agar dapat dicari berdasarkan lokasi geografis, *peer* mempublikasikan keberadaannya dalam P2P *overylay* pada *Distributed Geographic Table* (DGT). Sehingga setiap *peer* dapat mengidentifikasi *peer* lainnya berdasarkan lokasi geografis. Proses *service discovery* ditunjukkan pada gambar 5.

Pendekatan terdistribusi menggunakan arsitektur P2P memberikan jaminan skalabilitas, sehingga peningkatan jumlah perangkat yang terdaftar dalam sistem tidak akan membebani sistem. Selain itu, setiap *peer* bertindak sebagai *gateway* sekaligus *proxy*, sehingga akses terhadap perangkat dapat dikendalikan serta data dapat diproses sebelum dikirimkan kepada pengguna.



Gambar . Mekanisme *service discovery* pada arsitektur P2P (Cirani, 2014)

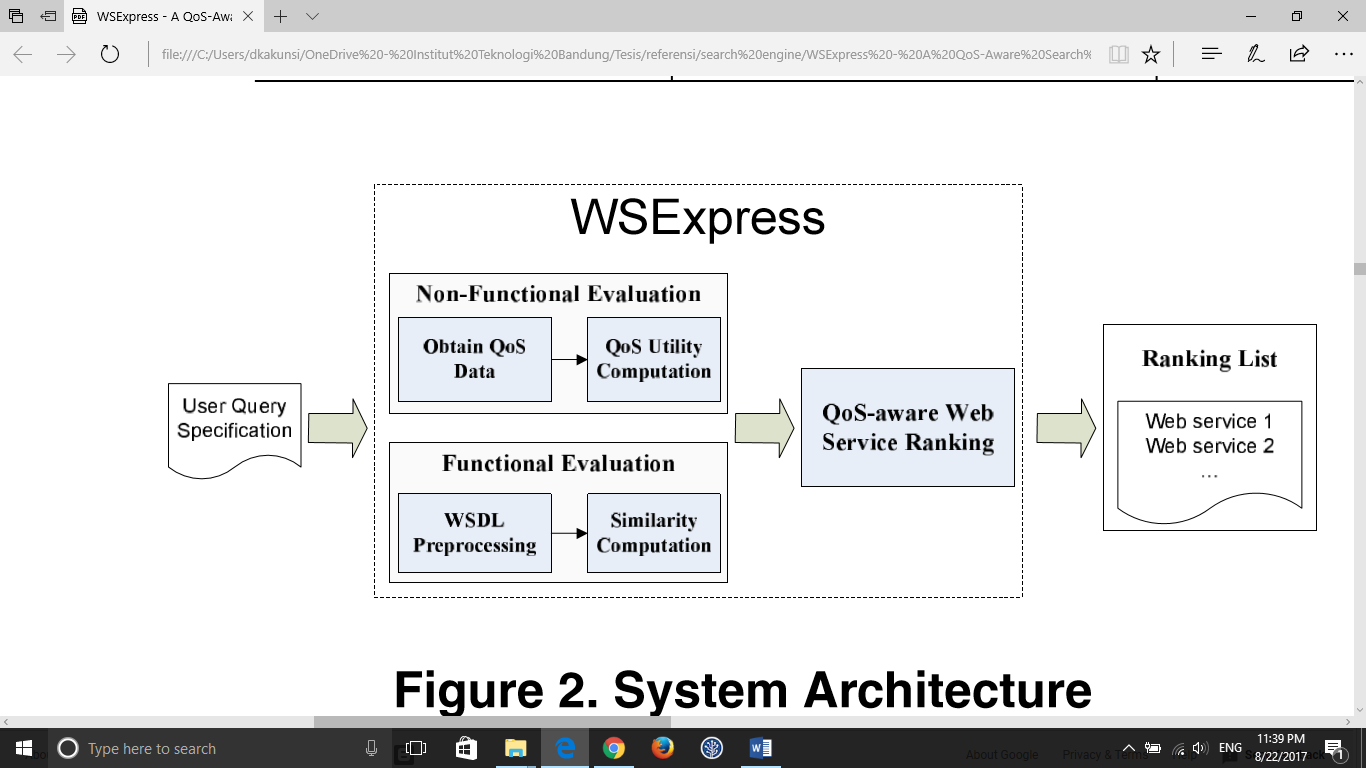
Tabel . Perbandingan *service discovery* pada IoT

| Fitur | *Discovery* | *Scalability* | *Heterogeneity* | *Locality* |
| --- | --- | --- | --- | --- |
| Jara, et al. | Memungkinkan *discovery* berbasis *ElasticSearch* | Menggunakan Digrectory, - *service* *repository* terdistribusi | Memungkinkan integrasi perangkat melalui Digrectory | Menggunakan atribut lokasi pada deskripsi dan *mobile client* |
| Datta, at al. | Memungkinkan *discovery* berbasis *search engine* terhadap *service repository* dan indeks terpusat | Tidak tersedia | Memungkinkan integrasi perangkat melalui *proxy* tertentu sesuai teknologi komunikasi | Tidak tersedia |
| Elhaster, et al. | Menggunakan model statistik dan indeks terhadap *service repository* yang sudah dikelompokkan dan terdistribusi | Menggunakan *service repository* yang terdistribusi berdasarkan lokasi geografis | Tidak tersedia | Ditentukan berdasarkan *service repository* |
| Cirani, et al. | Menggunakan 2 *overlay*, yaitu DLS dan DGT untuk mempublikasikan *service* dan *peer* | Menggunakan arsitektur P2P, setiap *peer* mengelola perangkat yang terdaftar | Tidak tersedia | Ditentukan berdasarkan lokasi *peer* |
| Liu, et al. | Menggunakan P2P *overlay* | Menggunakan arsitektur P2P, setiap *peer* mengelola perangkat yang terdaftar | Tidak tersedia | Tidak tersedia |

## II.4. *Service Ranking*

Pembagian layanan menjadi kelompok *cluster* mampu meningkatkan efisiensi pencarian layanan, karena layanan yang memiliki kemiripan karakteristik telah dikumpulkan menjadi satu kelompok, sehingga area pencarian tidak terlalu luas. Akan tetapi, pemilihan layanan yang terbaik untuk suatu *request* belum sepenuhnya terjawab hanya dengan *clustering*. Diperlukan mekanisme lain untuk menentukan relevansi layanan-layanan dalam suatu *cluster* terhadap request tertentu. Elhaster, dkk. (Elhaster, Elgazzar, & Mertin, 2016) membandingkan 4 jenis algoritma pencocokan, yaitu *Normalized Google Distance* (NGD), *brute force search*, *K-Dimensional tree*, dan *Locality Sensitive Hashing* (LSH). Hasil evaluasi menunjukkan bahwa algoritma berbasis indeks dapat menemukan layanan dengan cepat, sedangkan NGD dan *brute force* menemukan layanan lebih akurat namun waktu proses lebih lama.

Beberapa penelitian lain mencoba menerapkan perbandingan secara fungsional maupun non-fungsional, antara lain (Jin, Chun, Jung, & Lee, 2014) dan (Zhang, Zheng, & Lyu, 2010). Jin, dkk. mengusulkan seleksi layanan berdasarkan kualitas *physical* *service*. Kriteria yang digunakan dalam seleksi yaitu *available time* (AT), *service area* (SA), *processing time* (PT), dan *reputation* (RP). Penghitungan nilai dilakukan pada setiap kriteria dengan model *rating* masing-masing. Penghitungan PT dan RP dilakukan pada *design time*, sedangkan AT dan SA dihitung saat *runtime* karena bergantung pada *request* pengguna. Urutan layanan dapat berubah sesuai dengan preferensi pengguna.



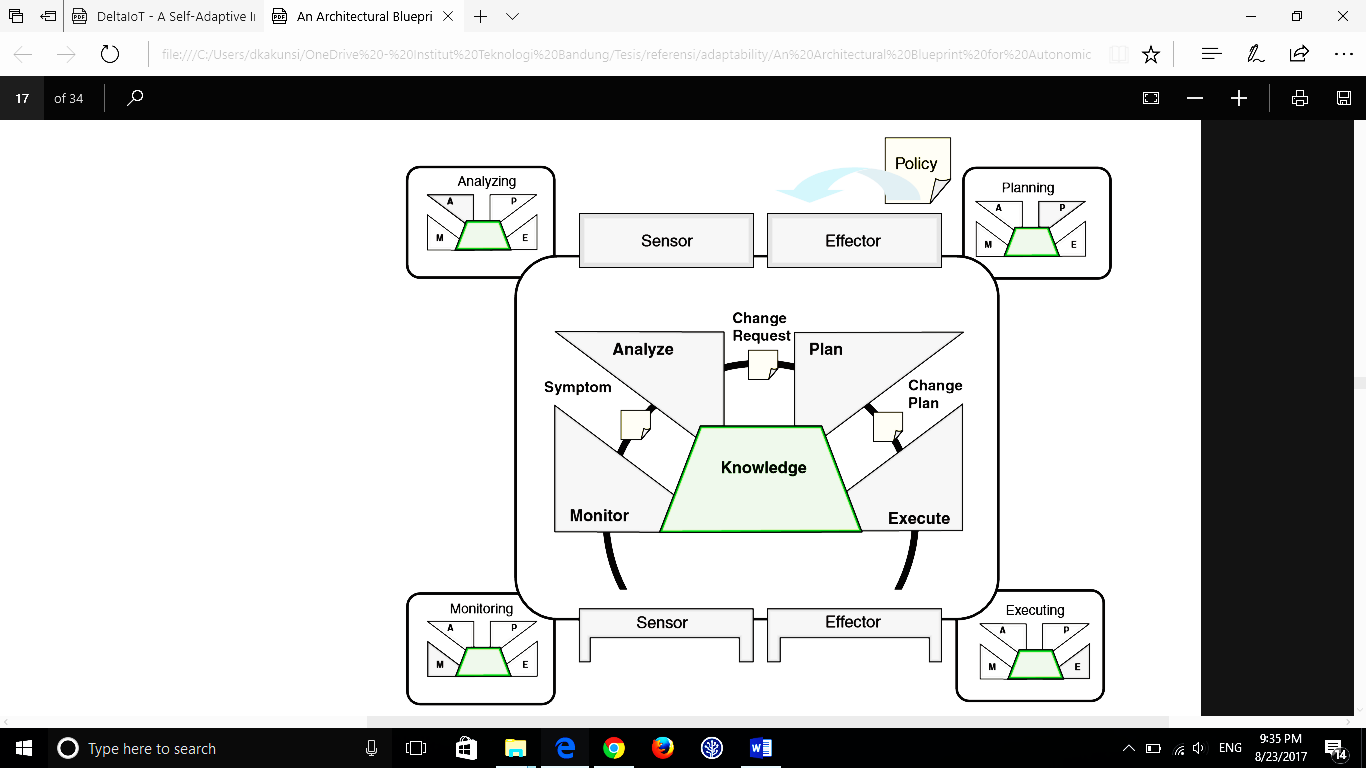
Gambar . Arsitektur WSExpress (Zhang, 2010)

Zhang, dkk. dalam penelitian WSExpress menggunakan kombinasi nilai fungsional dan non-fungsional untuk memilih layanan yang paling sesuai (gambar 6). Penghitungan nilai fungsional menggunakan *cosine similarity* berdasarkan pengukuran TF-IDF terhadap vektor (*keyword, input, output*). Penghitungan nilai non-fungsional menggunakan *utility* *function* terhadap matriks kualitas *web* *service*. Kriteria kualitas yang digunakan adalah kriteria umum seperti *response time*, *throughput*, dan *availability*, serta kriteria bisnis seperti *penalty rate* dan *price*.

Penilaian berdasarkan algoritma *matching* yang diusulkan Elhaster, dkk. pada dasarnya membandingkan karakteristik fungsional layanan, sehingga urutan yang dihasilkan berkaitan dengan kemiripan fungsional. Pada prakteknya, perangkat dengan kemiripan fungsional yang tinggi namun kualitas layanan buruk seharusnya tidak dapat menjadi layanan alternatif. Pendekatan non-fungsional yang diusulkan Jin, dkk. memungkinkan pengurutan berdasarkan kualitas layanan. Urutan dapat ditentukan berdasarkan preferensi pengguna. Akan tetapi, penelitian tidak mempertimbangkan kemiripan fungsional layanan. Selain itu, aspek reputasi layanan mengharuskan pengguna berinteraksi secara langsung dengan perangkat agar dapat memberikan *feedback*. *Adaptability* sistem terhadap ketersediaan perangkat sulit dicapai jika pengguna berinteraksi langsung dengan perangkat. Zhang, dkk. menggabungkan penilaian fungsional dan non-fungsional. Pendekatan yang digunakan mencoba membandingkan semua layanan yang tersedia terhadap *request* pengguna, sehingga membutuhkan waktu yang lama. Seiring dengan penambahan jumlah layanan, beban terhadap sistem *discovery* akan menjadi semakin besar. Selain itu, dalam lingkungan operasional IoT waktu proses tidak boleh terlalu lama sebab dapat mempengaruhi faktor *timeliness* layanan, serta kemampuan pemrosesan perangkat IoT tidak sebesar *server* pada umumnya.

## II.5. *System Adaptability*

Lingkungan operasional IoT bersifat dinamis, dimana ketersediaan perangkat dapat berubah-ubah kapan saja. Sistem IoT harus dapat beradaptasi terhadap perubahan tersebut sehingga dapat memberikan hasil terbaik, terutama berkaitan dengan fungsionalitas *service discovery*. Agar dapat memberikan layanan yang paling relevan kepada pengguna, sistem harus dapat mengidentifikasi perubahan konfigurasi perangkat, serta memperhitungkan pengaruhnya terhadap *service ranking*.

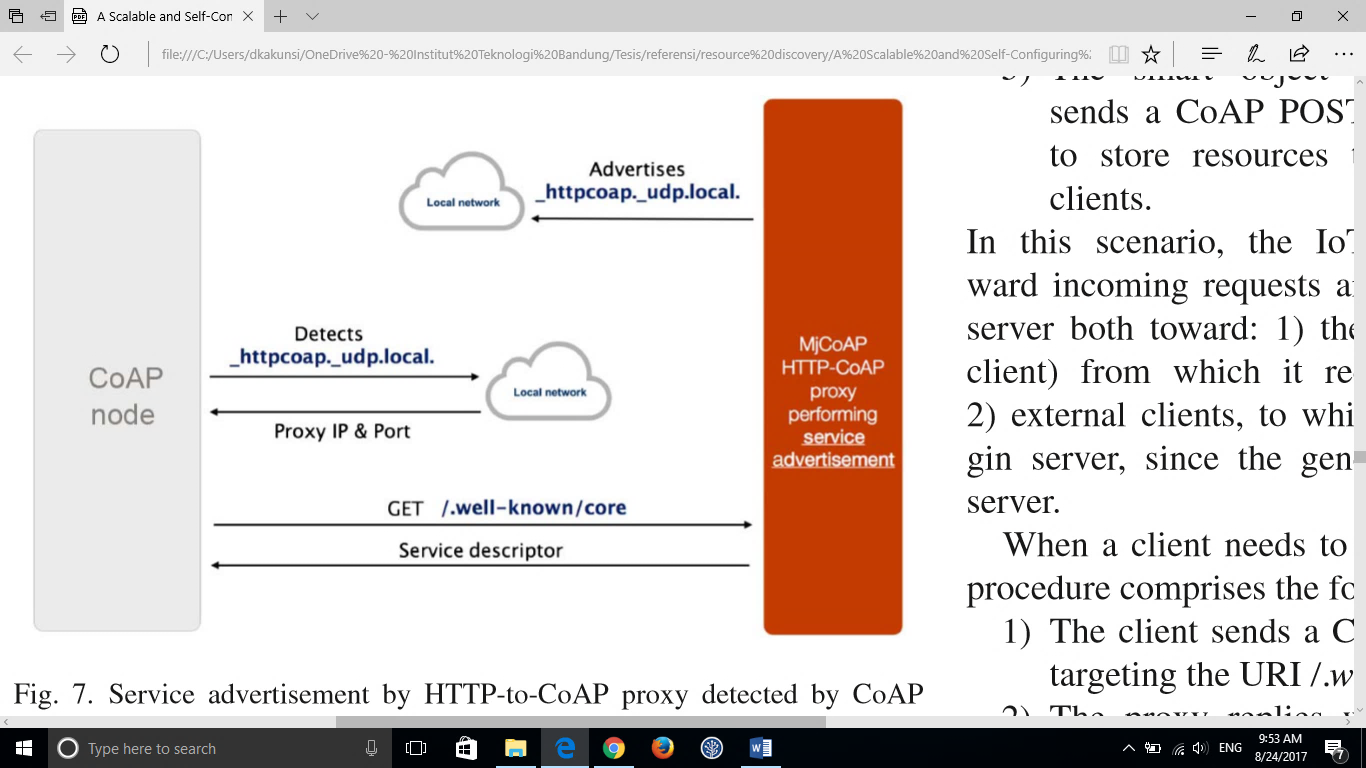


Gambar . Fungsionalitas *autonomic manager* (IBM, 2005)

Dalam domain sistem terotomasi dikenal istilah *autonomic computing*. IBMmembuat suatu rancangan *autonomic computing* yang menjelaskan tentang konsep, arsitektur, tingkat adopsi, serta standar yang digunakan (IBM, 2006). Tujuan *autonomic computing* adalah sistem mampu mengelola komponen operasional sesuai dengan situasi dan perubahan yang terjadi pada lingkungannya atau yang disebut *self-management*. IBM mendefinisikan 4 kategori *self-management*, yaitu *self-configuring*, *self-healing*, *self-optimizing*, dan *self-protecting*. *Self-configuring system* mampu beradaptasi terhadap perubahan konfigurasi jaringan perangkat IoT, sehingga memungkinkan identifikasi layanan baru dan kegagalan layanan. Dalam rancangan arsitektur *autonomic computing*, diperkenalkan komponen *autonomic manager* yang berfungsi untuk mengelola proses otomasi sistem, dengan fungsionalitas seperti pada gambar 7.

Fungsionalitas *autonomic manager* didefinisikan menjadi 4 bagian yang disebut MAPE, yaitu *monitoring*, *analyze*, *plan*, dan *execute*. *Monitoring* bertanggung jawab untuk mengumpulkan, menyaring, dan melaporkan perubahan yang terjadi pada obyek yang dikelola. *Analyze* berfungsi memperhitungkan dan memodelkan situasi atau keadaan berdasarkan hasil monitoring. *Plan* bertanggung jawab membangun rencana aksi yang dibutuhkan untuk mencapai tujuan berdasarkan hasil analisis sebelumnya. *Execute* berfungsi mengendalikan pelaksanaan rencana. Keempat bagian menggunakan *knowledge* yang sama dan didukung oleh *policy*. Meskipun dirancang sebagai suatu kesatuan untuk melaksanakan fungsi otomasi, namun setiap bagian dari MAPE dapat beroperasi secara terpisah.

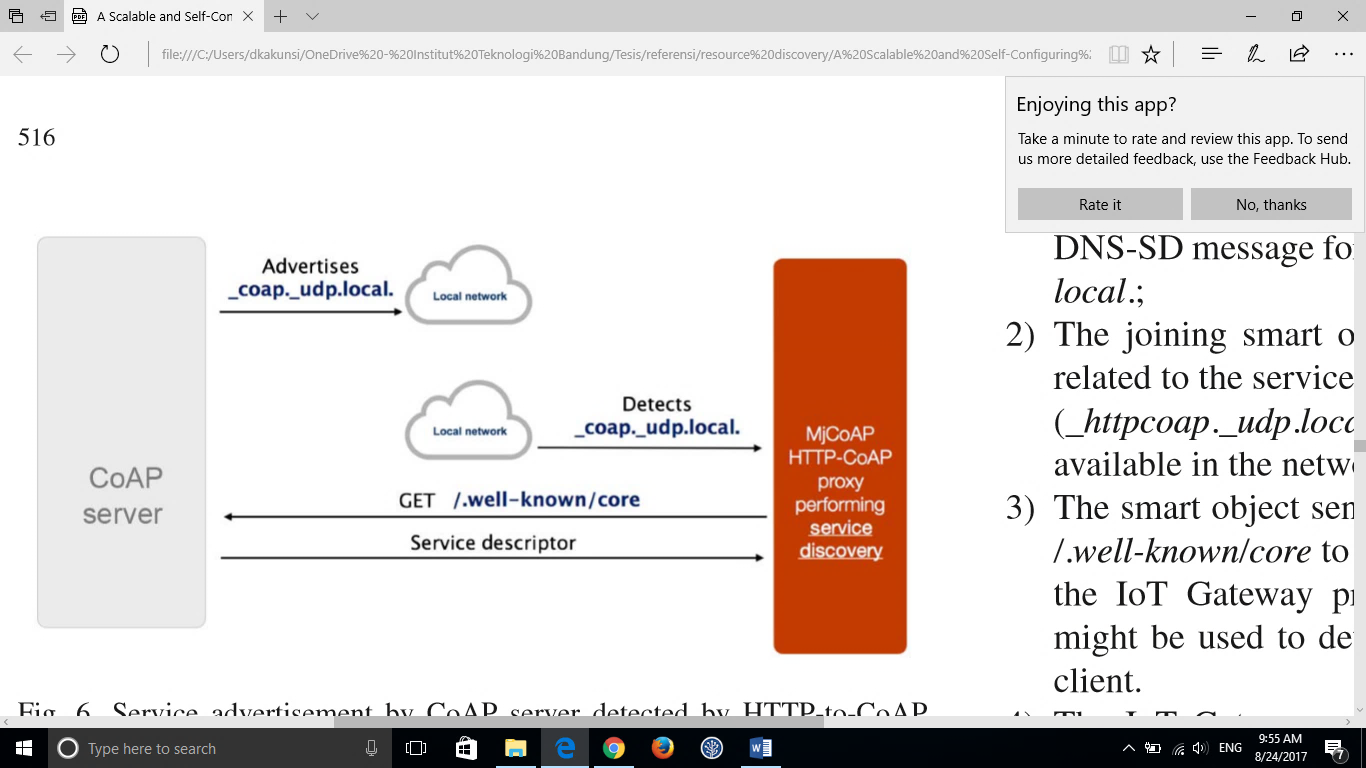
Cirani, dkk. membuat rancangan arsitektur *self-configuring* untuk mendukung *service discovery*, terutama pada proses *service advertisement*. Arsitektur memungkinkan layanan perangkat yang baru ditambahkan ke dalam sistem dipublikasikan secara otomatis pada *gateway* dan P2P *overlay* agar dapat ditemukan melalui jaringan lokal dan global. Penerapan arsitektur ini meminimalisir keterlibatan manusia dalam proses konfigurasi dan perawatan perangkat. Selain pada tingkat perangkat, *self-configuration* juga berlaku pada tingkat *gateway* yang berperan sebagai *peer* dalam jaringan P2P.



Gambar . *Service advertisement* dari sisi *gateway* (Cirani, 2014)

Pada tingkat lokal, peneliti mengajukan mekanisme baru berbasis Zeroconf sebagai solusi *service discovery* dengan pendekatan *self-configuration*. Mekanisme tersebut berhubungan dengan *service advertisement* yang dilakukan secara otomatis dari perangkat ke *gateway*. Mekanisme ini memungkinkan setiap perangkat dalam jaringan lokal beradaptasi terhadap lingkungannya atau *gateway* yang memonitor ketersediaan setiap perangkat. Pada skenario pertama, perangkat IoT yang baru ditambahkan ke dalam jaringan lokal melakukan proses *advertisement* ke *gateway* (gambar 8). Pada skenario ini, *gateway* mempublikasikan keberadaannya secara periodik agar dapat dideteksi oleh perangkat. Skenario kedua perangkat yang harus mempublikasikan keberadaanya agar terdeteksi oleh *gateway* (gambar 9). Dua skenario tersebut menggunakan pesan DNS-SD untuk mempublikasikan diri sehingga dapat terdeteksi satu sama lain.

Pendekatan MAPE mampu menganalisis perubahan dan beradaptasi dengan perubahan tersebut. Pada penerapannya, terdapat 2 jenis sistem yang diimplementasikan yaitu sistem yang dapat mengalami perubahan (*managed system*) dan sistem yang mengelola perubahan (*managing system*). Agar sistem *self-management* dapat berfungsi, kedua sistem harus ada. Sedangkan pada kasus IoT, perubahan yang terjadi berkaitan dengan ketersediaan perangkat. *Managing system* tidak dapat melaksanakan fungsinya apabila *managed system* tidak tersedia. Pendekatan ZeroConf yang diterapkan oleh Cirani, dkk. dapat mendeteksi penambahan perangkat secara otomatis. Namun, pengaruh perubahan tersebut tidak terlihat secara jelas karena aspek *discovery* yang ditekankan hanya memungkinkan layanan ditemukan melalui publikasi *service description*. Selain itu, perubahan yang terjadi karena kegagalan perangkat tidak dibahas pada penelitian tersebut.



Gambar . *Service advertisement* dari sisi perangkat (Cirani, 2014)

# BAB III ANALISIS DAN PERANCANGAN

Bab ini gambaran sistem yang akan dibangun secara keseluruhan, terkait dengan kebutuhan dan arsitektur *service selection*, rancangan sistem, dan skenario adaptasi.

## III.1. Analisis Sistem

Berdasarkan studi literatur, ditemukan bahwa penerapan *service selection* yang ada saat ini belum menyertakan aspek *adaptability* terhadap perubahan ketersediaan perangkat (*availability*) pada jaringan perangkat IoT. Oleh karena itu, penelitian ini berupaya mengintegrasikan aspek *adaptability* ke dalam proses *service selection*, sehingga pengguna memperoleh data dari sumber yang paling relevan pada waktu tertentu. Analisis terhadap *service selection* diperlukan untuk menentukan fungsionalitas yang dibutuhkan, beserta pendekatan yang akan diterapkan. Selain itu, diperlukan analisis kebutuhan terkait interaksi dengan eksternal sistem dan kebutuhan berdasarkan aspek kualitas *service selection*.

### III.1.1. IoT Service Selection

*Service* adalah *building block* yang penting untuk membangun sistem yang kompleks. Dalam *service-oriented computing*, *service* adalah abstraksi fungsionalitas yang dapat dilakukan dengan menyediakan API dan deskripsinya, sehingga dapat digunakan oleh pengguna.

Berkaitan dengan domain IoT, model layanan dapat digambarkan dengan menggunakan *Physical Service Model* seperti pada gambar 10 (Jin, Chun, Jung, & Lee, 2014). Dalam penerapannya, perangkat IoT (*device*) melekat pada obyek tertentu. Setiap *device* memiliki *resource,* yaitu elemen komputasi yang mampu berinteraksi dengan obyek. *State* dari obyek dapat diakses dan dimanipulasi melalui *resource*. *Resource* diekspos oleh *service* yang disediakan oleh perangkat sehingga dapat diakses oleh pengguna. Setiap *service* terikat dengan sebuah deskripsi yang menjelaskan segala hal tentang *service* tersebut, yang disebut *service description*.



Gambar . *IoT Service Model* berdasarkan *Physical Service Model*

*IoT service selection* adalah pemilihan layanan yang disediakan oleh perangkat berdasarkan relevansi dan kualitas layanan terhadap kebutuhan pengguna. Hasil dari *service selection* adalah perangkat IoT yang memenuhi aspek fungsional yang dibutuhkan dan memiliki kualitas layanan yang paling baik berdasarkan preferensi pengguna. Kemampuan *service selection* dapat disediakan melalui 2 sub-proses, yaitu *discovery* dan *ranking* (gambar 11). *Service* *discovery* bertujuan untuk mempublikasikan layanan-layanan yang tersedia pada jaringan IoT agar dapat ditemukan dan diakses oleh pengguna. *Service discovery* adalah salah satu bagian penting dalam menyediakan *service selection*, karena layanan hanya dapat digunakan jika pengguna mampu menemukan layanan tersebut. Proses *discovery* yang baik pada lingkungan IoT harus mampu mendeteksi penambahan perangkat baru dan menambahkan *resource* yang disediakan ke dalam daftar *service*. *Ranking* bertujuan untuk mengurutkan layanan-layanan yang relevan terhadap kebutuhan pengguna berdasarkan kualitasnya. Perangkat yang dipilih adalah perangkat yang berada pada urutan teratas pada *ranking*, sesuai dengan jumlah yang dibutuhkan. Selain itu, *ranking* dapat digunakan untuk menyediakan layanan alternatif ketika terjadi kegagalan perangkat.



Gambar . Proses *service selection*

Adaptasi terhadap perubahan ketersediaan perangkat dibutuhkan karena lingkungan operasional IoT sangat dinamis. Dalam jaringan perangkat IoT dapat terjadi kegagalan perangkat yang tersedia dan penambahan perangkat baru. Keduanya dapat mempengaruhi aspek *availability* layanan. Proses adaptasi terhadap kegagalan perangkat dapat ditanggulangi dengan adanya *ranking*. Apabila terjadi kegagalan, perangkat pada urutan berikutnya pada *ranking* dijadikan sebagai alternatif. Sedangkan untuk penambahan perangkat, adaptasi bergantung pada kemampuan *discovery* dalam mendeteksi dan menambahkan layanan secara otomatis (*automatic discovery*). Apabila *automatic discovery* mampu dilakukan, proses selanjutnya adalah penyesuaian *ranking* dengan menyisipkan layanan yang baru ke dalam *ranking* yang tersedia.

### III.1.2. Kebutuhan Sistem

Kebutuhan SDRank dapat diidentifikasi dari aspek fungsional dan non-fungsional. Aspek fungsional dapat ditinjau dari interaksi sistem dengan lingkungan eksternal atau aktor, sedangkan aspek non-fungsional ditinjau dari faktor kualitas. Jenis interaksi sistem dengan aktor dapat dilihat pada tabel 4.

Tabel . Daftar aktor yang berinteraksi dengan SDRank

| **No.** | **Aktor** | **Deskripsi** |
| --- | --- | --- |
| A-01 | Pengguna | Pengguna dapat berupa orang maupun mesin. Interaksi terjadi ketika pengguna membutuhkan layanan dan data dari perangkat, melalui sistem sebagai *proxy*. |
| A-02 | *Proxy* | *Proxy* dapat berinteraksi dengan sistem dalam menyediakan *resource* dan mendaftarakan layanan saat pertama kali ditambahkan ke dalam jaringan lokal, serta mengakses layanan dan data dalam skenario M2M. |

Sistem dapat berinteraksi langsung dengan pengguna maupun perangkat. Fungsionalitas utama sistem adalah memilih layanan yang paling relevan terhadap kebutuhan pengguna dan menyediakan akses terhadap perangkat tersebut. Sedangkan berdasarkan aspek non-fungsional, sistem harus menyediakan kemampuan *scalability*, *reusability*, dan *robustness*. Faktor *scalability* harus diperhatikan karena terkait dengan jumlah perangkat yang banyak dan terus bertambah. Aspek *reusability* berkaitan dengan sumber daya yang dimiliki sistem, yaitu layanan yang terdaftar dan *ranking*. Sumber daya tersebut harus dapat digunakan kembali apabila kebutuhan pengguna sama. *Robustness* berkaitan dengan lokalisasi kegagalan. Kegagalan pada perangkat tidak boleh mempengaruhi keseluruhan sistem dalam menyediakan fungsionalitasnya. Detil kebutuhan fungsional dan non-fungsional SDRank dijabarkan pada tabel 5 dan tabel 6.

Tabel . Kebutuhan fungsional SDRank

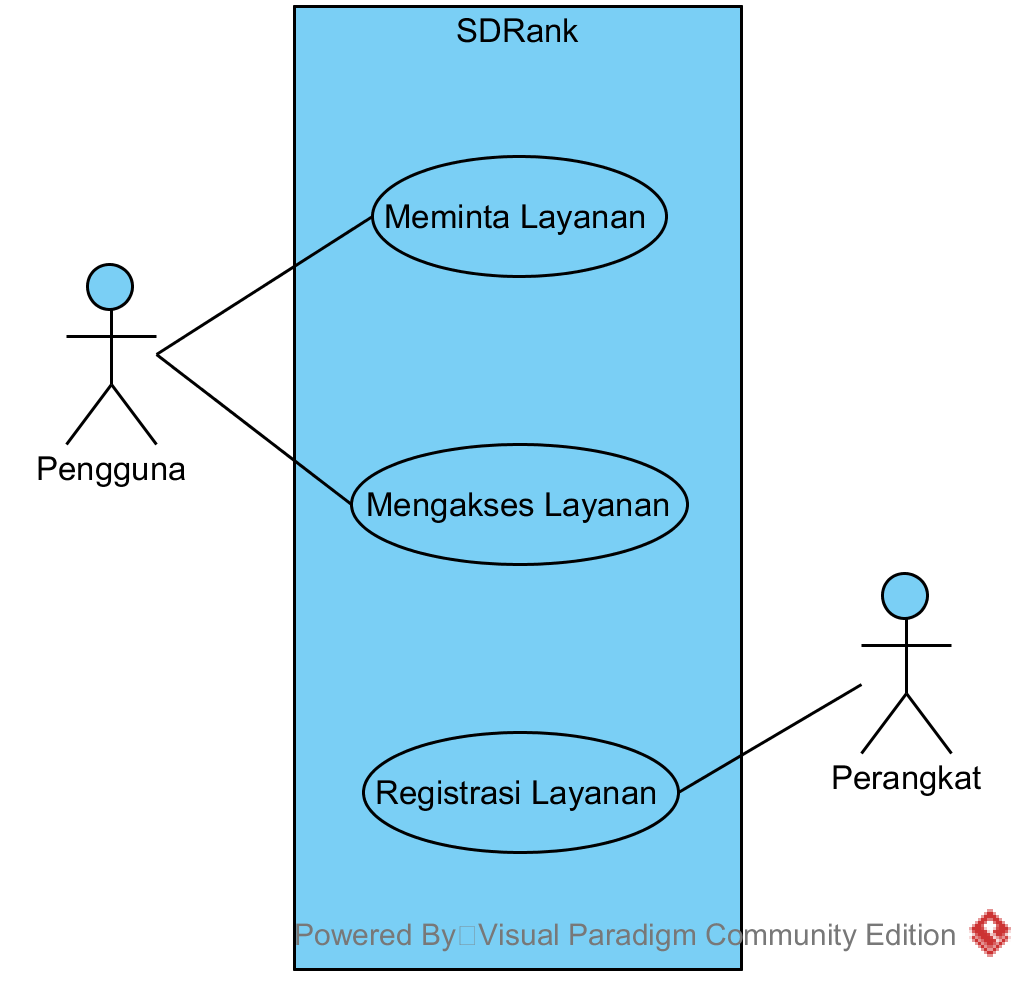
| **No.** | **Kebutuhan Fungsional** |
| --- | --- |
| F-01 | Sistem mampu memilih layanan untuk kebutuhan pengguna sesuai dengan relevansi layanan terhadap kebutuhan dan kualitas kinerja layanan. |
| F-02 | Sistem mampu menemukan layanan yang disediakan perangkat dalam jaringan IoT secara otomatis. |
| F-03 | Sistem mampu mendaftarkan layanan yang disediakan oleh perangkat ke dalam repositori layanan. |
| F-04 | Sistem mampu mengurutkan layanan yang memenuhi kebutuhan pengguna berdasarkan relevansi dan kualitas. |
| F-05 | Sistem mampu berinteraksi dengan perangkat. |
| F-06 | Sistem mampu menggunakan kembali *ranking* yang tersedia apabila *ranking* tersebut relevan terhadap permintaan layanan dari pengguna. |
| F-07 | Sistem mampu menyimpan daftar layanan yang disediakan perangkat. |
| F-08 | Sistem mampu mengidentifikasi penambahan perangkat. |
| F-09 | Sistem mampu menyediakan layanan alternatif saat terjadi kegagalan perangkat. |
| F-10 | Sistem mampu mengubah urutan *ranking* ketika terjadi penambahan perangkat yang mempengaruhi *ranking*. |
| F-11 | Sistem mampu mengubah layanan yang digunakan pengguna apabila layanan baru menempati urutan pertama dalam *ranking*. |

## III.2. Perancangan Sistem

Rancangan sistem dibangun berdasarkan kebutuhan fungsional dan non-fungsional yang telah diidentifikasi pada bagian analisis. Pada penelitian ini, *service selection* didekomposisi menjadi dua sub-proses, yaitu *discovery* dan *ranking* seperti pada gambar 11. Proses *discovery* dan *ranking* adalah dua proses yang beruntun, di mana keluaran dari proses *discovery* menjadi masukan untuk proses *ranking*. Berdasarkan kebutuhan sistem, interaksi sistem dengan pengguna ditampilkan pada diagram *use case* (gambar 12). Rancangan pada penelitian ini disediakan dalam dua tipe berdasarkan model UML, yaitu struktural dan *behavioral*.

Tabel . Kebutuhan non-fungsional SDRank

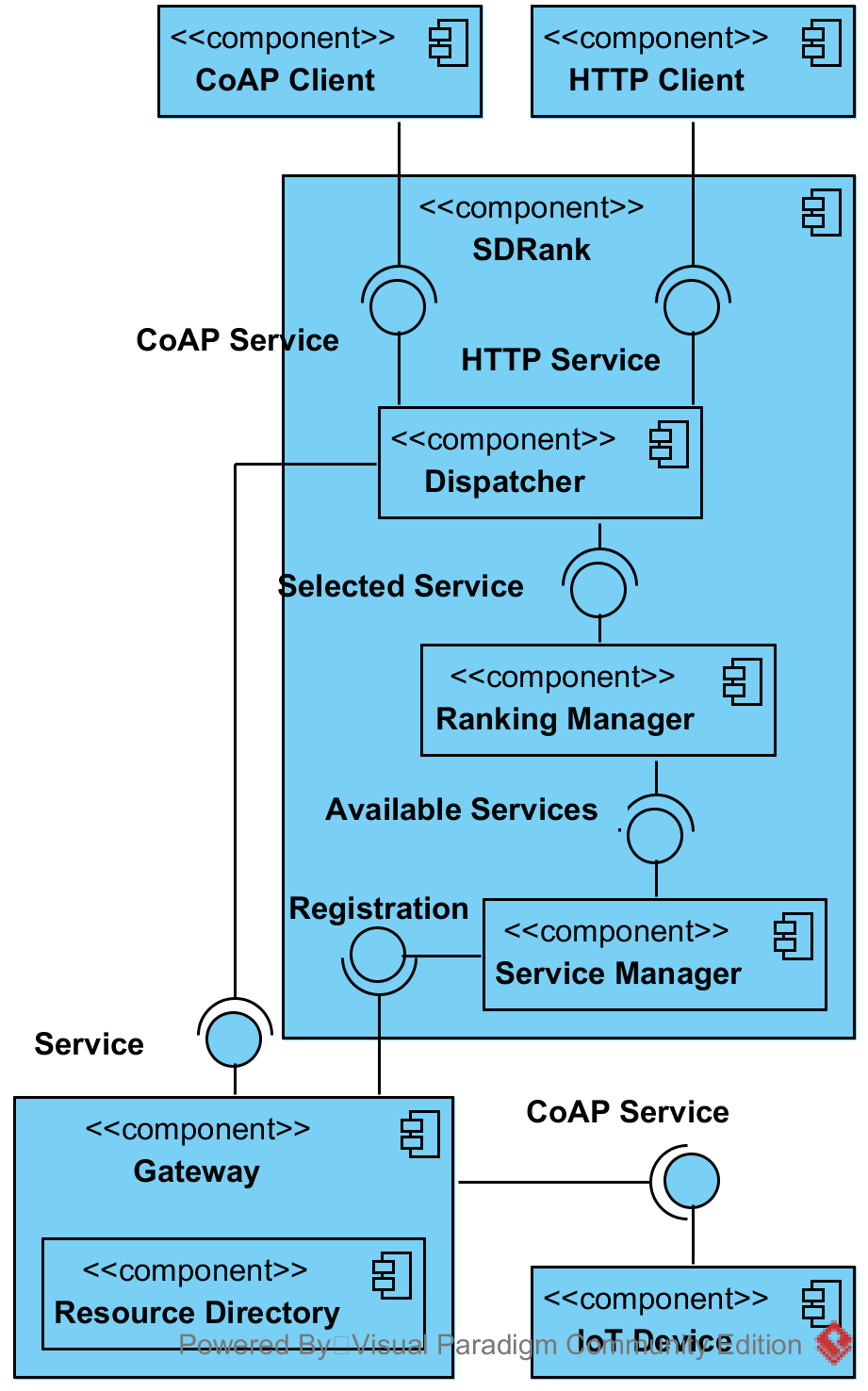
| **No.** | **Kebutuhan Non-Fungsional** |
| --- | --- |
| NF-01 | Sistem harus mampu mengakomodir penambahan perangkat tanpa penurunan kinerja. |
| NF-02 | Pemilihan layanan harus mempertimbangkan *ranking* yang tersedia. |
| NF-03 | Kegagalan perangkat tidak boleh mempengaruhi fungsionalitas sistem. |



Gambar . Diagram *Use Case* SDRank

### III.2.1. Rancangan Arsitektur Sistem

Secara umum, sistem terdiri atas beberapa *layer*, yaitu *sensing* *layer*, *discovery layer*, *service enablement* *layer*, dan *application layer*. *Sensing layer* terdiri atas perangkat-perangkat IoT yang berfungsi untuk mengambil nilai obyek melalui *resource* yang tersedia pada perangkat. *Discovery layer* berfungsi untuk mendeteksi perangkat dan *resource* yang disediakan, serta mempublikasikan layanan pada SDRank. Komponen pada *discovery layer* antara lain *proxy* dan *service manager*. *Service enablement layer* berinteraksi langsung dengan pengguna dan berfungsi untuk menyediakan fungsionalitas layanan SDRank, yaitu pemilihan perangkat dan akses terhadap perangkat. *Application layer* adalah aplikasi pengguna yang membutuhkan *resource* dari perangkat IoT. Aplikasi dapat mengakses sistem menggunakan CoAP dan HTTP sesuai dengan tipe akses. Arsitektur sistem SDRank ditunjukkan pada gambar 13.



Gambar . Arsitektur SDRank

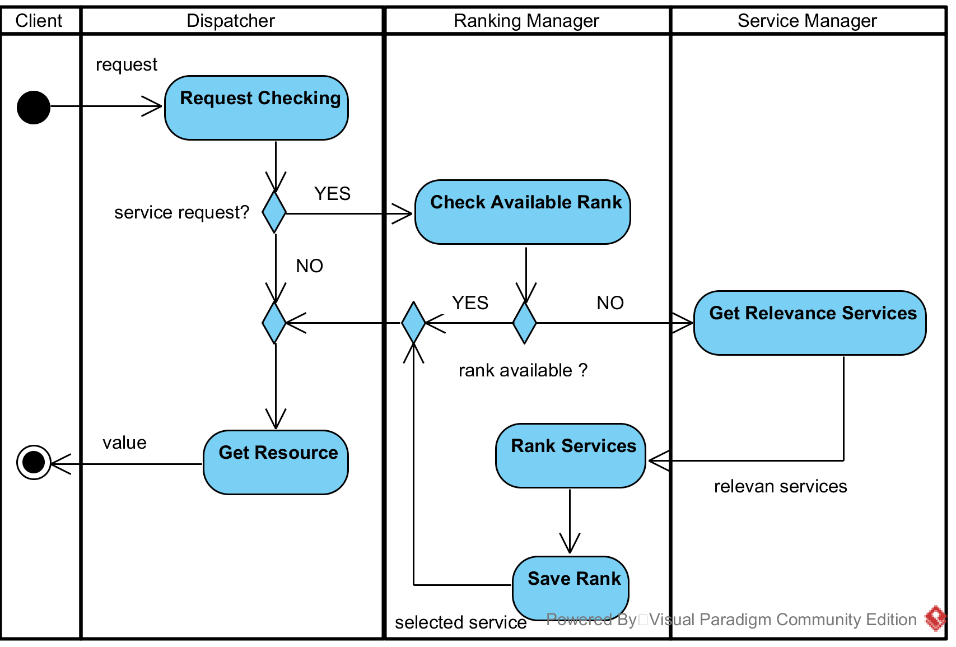
Berdasarkan gambar 13, tersedia tiga komponen besar sistem yaitu SDRank sebagai *core* sistem, *proxy* sebagai perantara perangkat IoT dan SDRank, dan jaringan perangkat IoT. Komponen *client* disediakan oleh pengguna yang akan mengakses sistem. Akan tetapi berdasarkan analisis kebutuhan sistem, ditemukan bahwa selain proses *selection* dan adaptasi, sistem membutuhkan mekanisme akses perangkat, serta registrasi layanan dan *ranking*. Oleh karena itu, selain komponen yang terkait langsung dengan *selection*, perlu disediakan komponen-komponen yang dibutuhkan untuk setiap proses atau mekanisme tersebut seperti *dispatcher*, *service manager*, dan *resource directory*. Tabel 7 menunjukkan daftar komponen yang diperlukan dalam sistem SDRank serta menjelaskan fungsionalitas dari masing-masing komponen.

Tabel . Komponen sistem

| **No.** | **Komponen** | **Deskripsi** |
| --- | --- | --- |
| 1. | SDRank | SDRank adalah komponen utama dalam sistem, berfungsi untuk menemukan layanan IoT yang tersedia, memilih layanan secara otomatis berdasarkan kebutuhan pengguna, mengakses *resource* IoT untuk pengguna, serta menyediakan alternatif layanan saat terjadi perubahan ketersediaa perangkat pada jaringan IoT. SDRank terdiri atas sub-komponen *service manager*, *ranking*, dan *dispatcher*. |
| 2. | *Service manager* | *Service manager* berfungsi untuk mengelola daftar layanan pada jaringan IoT, sehingga dapat diakses oleh pengguna. Selain itu, *service manager* menyimpan daftar *gateway* yang terkoneksi ke SDRank. |
| 3. | *Ranking Manager* | *Ranking Manager* berfungsi untuk memilih layanan-layanan yang relevan terhadap kebutuhan tertentu, serta mengurutkan layanan berdasarkan relevansi dan kualitasnya. Selain itu, *ranking* disimpan dalam *ranking repository* agar dapat digunakan kembali untuk permintaan berikutnya. |
| 4. | *Dispatcher* | *Dispatcher* berfungsi untuk melayani permintaan pengguna. *Dispatcher* menentukan apakah akan melakukan proses *ranking* atau bertindak sebagai *proxy* untuk mengakses perangkat IoT. |
| 5. | *IoT Device* | *IoT Device* berfungsi untuk menyediakan nilai obyek. |
| 6. | *Proxy* | *Proxy* bertindak sebagai *head* untuk satu jaringan IoT lokal, menjadi *proxy* dalam mengakses IoT *resource*, serta mendeteksi penambahan perangkat di dalam jaringan lokal yang dikelola dan memulai proses *discovery*. Di dalam *gateway* tersedia *Resource Directory* untuk menyimpan layanan CoAP yang tersedia dalam jaringan lokal. |
| 7. | *Client* | *Client* adalah aplikasi pada sisi pengguna yang membutuhkan data dari perangkat IoT. *Client* dapat berupa *CoAP client* dan *HTTP client*, masing-masing berdasarkan protokol yang digunakan. |

### III.2.2. Rancangan Behavioral Sistem

Fitur yang disediakan *SDRank* adalah *service selection* dan *adaptability* terhadap perubahan ketersediaan perangkat di dalam jaringan IoT. Proses adaptasi dilakukan apabila terjadi penambahan atau kegagalan perangkat dalam jaringan IoT. *Runtime* dari *service selection* ditunjukkan pada gambar 14.



Gambar . Proses *service selection*

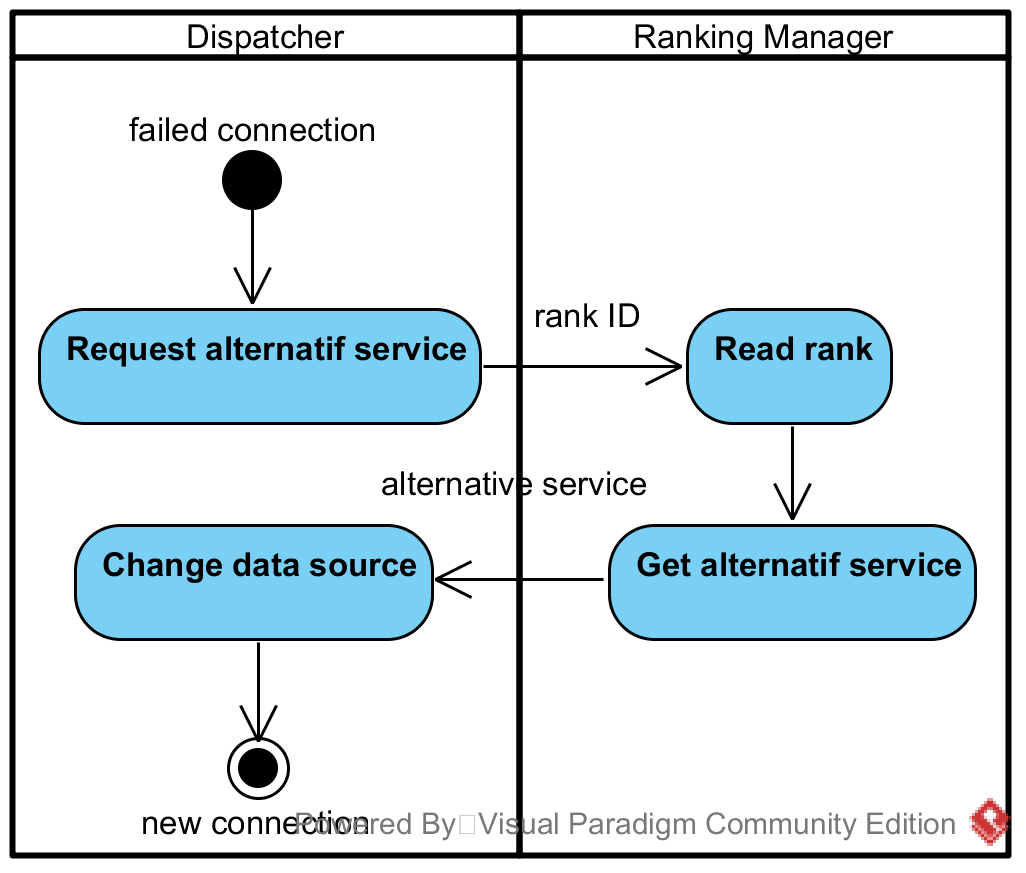
Proses *selection* melibatkan tiga komponen SDRank, yaitu *dispatcher*, *ranking manager*, dan *service manager*. *Request* dari *client* akan diterima dan diproses oleh komponen *dispatcher*. *Dispatcher* akan menentukan apakah *request* dapat langsung diteruskan ke perangkat melalui *gateway*, atau harus melakukan pengecekan pada komponen *ranking manager*.

Proses yang dilakukan oleh komponen *ranking manager* meliputi pengecekan ketersediaan dan pembuatan *ranking* yang sesuai dengan *request* pengguna. Pemeriksaan dilakukan terhadap daftar *ranking* yang sudah tersedia, berdasarkan jenis *resource* yang dibutuhkan dan lokasi perangkat. Apabila tersedia *ranking* yang sesuai dengan kebutuhan, layanan pada urutan teratas akan dikembalikan ke *dispatcher* sebagai hasil seleksi dan digunakan sebagai *data source* untuk pengguna. Apabila belum tersedia, maka operasi *service selection* akan dilakukan. Urutan yang telah terbentuk akan disimpan di dalam daftar *ranking* yang ada pada *ranking* *manager*, sehingga dapat dimanfaatkan untuk mencapai *reusability* dan *adaptability*. Apabila *availability* perangkat berubah, SDRankdapat menyediakan layanan alternatif sesuai dengan urutan *ranking*.

A screenshot of a cell phone

Description generated with high confidence

Gambar . Proses *adaptability* untuk penambahan perangkat



Gambar . Proses *adaptability* untuk kegagalan perangkat

Fungsi yang kedua adalah *adaptability* terhadap perubahan ketersediaan perangkat, yaitu penambahan perangkat baru pada jaringan IoT. Proses ini melibatkan empat komponen, yaitu *gateway*, *service manager*, *ranking manager*, dan *dispatcher*. Gambar 15 menunjukkan proses *adaptability* ketika terjadi penambahan perangkat baru dalam jaringan IoT.

Penambahan perangkat dapat dideteksi oleh *gateway* menggunakan pendekatan *Zero Configuration*. Ketika terjadi penambahan perangkat baru, *gateway* akan mengambil *resource* yang disediakan oleh perangkat tersebut dan menambahkannya ke dalam *resource directory* yang dimiliki *gateway*. *Resource* yang terdaftar dalam *resource directory* akan dipublikasikan pada komponen *service manager* sebagai *service* yang disediakan oleh *gateway*.

S*ervice manager* akan mengirimkan pemberitahuan kepada *ranking manager* untuk melakukan adaptasi terhadap perubahan tersebut. *Ranking manager* akan memeriksa *ranking* yang terpengaruh oleh penambahan layanan, dan apabila ada *ranking* yang terpengaruh, nilai dari layanan baru akan dihitung dan disisipkan ke dalam *ranking*. Dalam kasus, layanan baru menempati urutan teratas pada *ranking*, *ranking manager* akan mengirimkan pemberitahuan kepada *dispatcher* untuk mengubah *data source* yang mengacu pada *ranking* tersebut.

Adaptasi kedua adalah adaptasi terhadap kegagalan perangkat yang melibatkan komponen *dispatcher* dan *ranking manager*. Dalam kasus perangkat tidak dapat diakses, *dispatcher* akan meminta layanan alternatif dari *ranking manager*, dengan menyertakan ID *ranking* yang digunakan. *Ranking manager* akan membuka membaca *ranking* dan mengembalikan layanan pada urutan di bawahnya. *Dispatcher* akan mengubah *data source* sesuai dengan layanan alternatif. Gambar 16 menunjukkan proses *adaptability* ketika terjadi kegagalan perangkat dalam jaringan IoT.

### III.2.3. Proses Ranking

*Service ranking* bertujuan untuk mengurutkan layanan yang relevan terhadap kebutuhan pengguna berdasarkan perbandingan kualitas layanan. Pemilihan layanan dan adaptasi SDRank bergantung pada *ranking* yang dibuat, karena urutan pada *ranking* tersebut menentukan perangkat mana yang akan digunakan sebagai *resource*.

Proses *ranking* ditentukan berdasarkan atribut-atribut yang berkaitan dengan aspek kualitas layanan. Sedangkan, untuk membandingkan setiap layanan dapat menggunakan *Analytical Hierarchy Process* (AHP). AHP sering digunakan untuk pengambilan keputusan pada permasalahan-permasalahan yang melibatkan sejumlah atribut.

#### Atribut Kualitas

Atribut yang digunakan untuk proses *ranking* adalah sebagai berikut:

1. *Processing time*, adalah waktu yang diperlukan untuk melakukan suatu proses, yaitu jarak antara waktu *request* dan *response*.
2. *Coverage*, adalah jarak antara obyek dan lokasi yang diminta pengguna.
3. *Accuracy*, adalah derajat kebenaran data yang dihasilkan oleh perangkat terhadap keadaan sebenarnya.
4. *Precision*, adalah ketelitian perangkat dalam mengakuisisi data dari obyek, yaitu perbedaan antara data sebelum, saat ini, dan setelah tidak terlalu besar.
5. *Cost*, adalah biaya yang harus dikeluarkan pengguna untuk mengakses layanan.

#### Analytical Hierarchy Process (AHP)

# DAFTAR PUSTAKA

Bormann, C., Castellani, A. P., & Shelby, Z. (2012). CoAP: An Application Protocol for Billions of Tiny Internet Nodes. *IEEE Internet Computing, 16*(2), 62-67. Retrieved 8 24, 2017, from http://ieeexplore.ieee.org/document/6159216

Cabrera, C., Palade, A., & Clarke, S. (2017). *An evaluation of service discovery protocols in the internet of things*. Retrieved 8 24, 2017, from http://dblp.uni-trier.de/db/conf/sac/sac2017.html

Cheshire, S., & Krochmal, M. (2013). DNS-based service discovery. (RFC 6763).

Cheshire, S., & Krochmal, M. (2013). Multicast DNS. (RFC 6762).

Cirani, S., Davoli, L., Ferrari, G., Léone, R., Medagliani, P., Picone, M., & Veltri, L. (2014). A Scalable and Self-Configuring Architecture for Service Discovery in the Internet of Things. *IEEE Internet of Things Journal, 1*(5), 508-521. Retrieved 8 24, 2017, from http://ieeexplore.ieee.org/document/6899579

Datta, S. K., & Bonnet, C. (2015, 10 27). Search engine based resource discovery framework for Internet of Things. *Consumer Electronics (GCCE), 2015 IEEE 4th Global Conference on*, 83-85.

Dong, X., Haley, A., Madhavan, J., Nemes, E., & Zhang, J. (2004). Similarity search for web services. *The Vldb Journal*, 372-383. Retrieved 8 24, 2017, from http://dblp.uni-trier.de/db/conf/vldb/vldb2004.html

Elhaster, Y., Elgazzar, K., & Mertin, P. (2016). Web Service Discovery for Large Scale IoT Deployments. *International Journal of Services Computing, 4*(1), 55-68.

IBM. (2006, 6). An architectural blueprint for autonomic computing. *IBM White Paper, 31*, 1-34.

Jara, A. J., Lopez, P., Fernandez, D., Castillo, J. F., Zamora, M. A., & Skarmeta, A. F. (2013). *Mobile Digcovery: A Global Service Discovery for the Internet of Things*. Retrieved 8 24, 2017, from http://dblp.uni-trier.de/db/conf/aina/ainaw2013.html

Jin, X., Chun, S., Jung, J., & Lee, K.-H. (2014). *IoT Service Selection Based on Physical Service Model and Absolute Dominance Relationship*. Retrieved 8 24, 2017, from http://ieeexplore.ieee.org/document/6978172

Liu, M., Leppanen, T., Harjula, E., Ou, Z., Ylianttila, M., & Ojala, T. (2013, 10 14). Distributed resource discovery in the machine-to-machine applications. *Mobile Ad-Hoc and Sensor Systems (MASS), 2013 IEEE 10th International Conference on*, 411-412.

Ngu, A. H., Gutierrez, M., Metsis, V., Nepal, S., & Sheng, M. Z. (2016). IoT Middleware: A Survey on Issues and Enabling technologies. *IEEE Internet of Things Journal, 4*, 1-20. Retrieved 8 24, 2017, from http://ieeexplore.ieee.org/document/7582463

Ngu, A. H., Ma, J., Sheng, Q. Z., Yao, L., & Julian, S. (2014). *ServiceXplorer: a similarity-based web service search engine*. Retrieved 8 24, 2017, from http://dblp.uni-trier.de/db/conf/sigir/sigir2014.html

Vandana, C. P., & Chikkamannur, A. A. (2016). Study of Resource Discovery trends in Internet of Things (IoT). *International Journal on Advanced Networking and Application, 08*(03), 3084-3089.

Wu, J., Chen, L., Xie, Y., & Zheng, Z. (2012). *Titan: a system for effective web service discovery*. Retrieved 8 24, 2017, from http://dblp.uni-trier.de/db/conf/www/www2012c.html

Xia, F., Yang, L. T., Wang, L., & Finel, A. (2012, September 1). Internet of things. *International Journal of Communication Systems*, 1101-1102.

Zhang, Y., Zheng, Z., & Lyu, M. R. (2010). *WSExpress: A QoS-aware Search Engine for Web Services*. Retrieved 8 24, 2017, from http://ieeexplore.ieee.org/document/5552797