BAB 3
ANALISIS

Pada bab ini akan dibahas mengenai proses analisis terhadap kebutuhan sistem, proses bisnis sistem, analisis *use case*, dan pemodelan data. Penjelasan-penjelasan pada bab ini merupakan hasil dokumentasi atas tahapan *communication* dan *modeling* terhadap analisis. Tahap *communication* merangkum kebutuhan fungsional dan *scope* dari sistem, sedangkan tahap *modeling* mencakup penjelasan detil tiap fungsi guna memenuhi kebutuhan fungsional untuk kemudian digambarkan dalam *use case diagram* dan *business process diagram.*

3.1 Analisis Kebutuhan Sistem

Subbab ini membahas mengenai analisis kebutuhan sistem TRuST yang meliputi kebutuhan fungsional dan *non*-fungsional sistem, *actor glossary, use case glossary,* dan *use case diagram.*

Kebutuhan sistem dapat dibagi menjadi dua yaitu kebutuhan fungsional dan kebutuhan *non*-fungsional. Kebutuhan fungsional adalah *input, output,* proses, dan data yang dibutuhkan untuk memenuhi tujuan pengembangan sistem dan harus disediakan oleh sistem. Sedangkan, kebutuhan *non*-fungsional adalah fitur-fitur, karakteristik dan batasan lain yang mendefinisikan kepuasan sistem [WHI04].

Kebutuhan fungsional :

1. *Get* RS Terdekat
   Input : posisi pasien, rekam medis
   Proses : pasien menekan tombol 'RS Terdekat'

   Output : sistem menampilkan daftar rumah sakit yang terdekat dan paling sesuai

   Stored Data : posisi pasien, informasi RS
2. *View* Map
   Input : posisi pasien, posisi RS
   Proses : Bagian dari proses Get RS Terdekat

Output : peta daerah sekitar RS dan pasien

Stored Data : posisi pasien, posisi RS

3. *Healthcare Reservation*

   Input : *Healthcare reservation signal*

   Proses : pasien menekan tombol "Pesan" dan sistem mengirimkan *message* kepada WCF *service* RS yang terpilih sebelum menampilkan form untuk mengisi kondisi saat itu

   Output : *warning signal* di sistem RS

   Stored Data : -

4. *Urgent Signal*

   Input : posisi dan rekam medis *user*

   Proses : pasien menekan tombol "urgent" dan sistem mengirimkan *urgent signal* ke WCF *service* RS yang memiliki spesialis yang dibutuhkan dan yang paling dekat

   Output : *Warning urgent signal* di sistem rumah sakit

   Stored Data : -

5. *Warning healthcare*

   Input : *message healthcare* dari *device*

   Proses : sistem menampilkan *warning signal*

   Output : sistem menampilkan *warning* adanya *urgent signal* dan *healthcare reservation signal*

   Stored Data : data *healthcare reservation*

6. *Update* rekam medis

   Input : data rekam medis pasien

   Proses : admin memasukan rekam medis pasien yang baru berdasarkan data yang didapatkan dari RS

   Output : rekam medis yang baru

Stored Data : rekam medis

7. *Get New* Rekam medis

   Input : id rekam medis, ip *address* WCF *service* rumah sakit

   Proses : pasien menekan tombol 'Update Rekam medis' dari rumah sakit tertentu.

   Output : rekam medis yang baru.

   Stored Data : rekam medis yang baru

8. *View* rekam medis

   Input : -

   Proses : pasien memilih menu untuk melihat rekam medis yang disimpan di *device*

   Output : rekam medis pasien

   Stored Data : -

9. *Edit* biodata

   Input : biodata pasien

   Proses : pasien memilih menu untuk mengubah biodata miliknya yang disimpan di *device*

   Output : biodata pasien yang baru.

   Stored Data : biodata pasien

Kebutuhan non-fungsional :

1. Peta yang diperlihatkan hanya bagian peta yang menunjukkan di mana posisi *user* dan rumah sakit.
2. Daftar rumah sakit yang ditampilkan adalah rumah sakit yang masuk dalam radius jarak tertentu.

3.1.1 Batasan Sistem

Batasan sistem TRuST antara lain*:*

1. *Scope* lokasi yang digunakan untuk *prototype e-health context aware system* yang penulis kembangkan hanya untuk wilayah DKI Jakarta dan Depok, lebih tepatnya lagi hanya diimplementasikan untuk 8 rumah sakit yang berada di daerah tersebut, sedangkan *context* penyakit yang digunakan adalah ontologi dari penyakit kronis pada sistem pernapasan, sistem sirkulasi, dan saraf. Contoh dari penyakitnya misalkan penyakit jantung, asma atau hipertensi.

2. GPS yang digunakan pada sistem ini adalah sebuah *fake* GPS (*software* dari Microsoft untuk mensimulasikan kerja GPS *device*) yang membaca data NMEA GPS *message*. Data NMEA tersebut di-*generate* dari sebuah GPS emulator.

3. Diasumsikan ada standar rekam medis yang sama untuk setiap rumah sakit.

4. Diasumsikan setiap pasien yang berbeda memiliki id yang unik meskipun berbeda rumah sakit.

5. Sistem ini hanya dapat digunakan apabila kondisi jaringan internet diasumsikan tidak sedang mengalami gangguan. Modul-modul yang ada pada sistem ini merupakan versi awal yang nantinya akan menjadi dasar bila ternyata akan dilakukan pengembangan sistem lebih lanjut.

6. Pada menu *Urgent Signal* selalu diasumsikan terdapat rumah sakit yang *available* untuk memberikan pertolongan.

7. Untuk membatasi penggunaan memori di *mobile device* dalam menyimpan rekam medis, maka *mobile device* hanya menyimpan 20 data rekam medis yang paling baru.

8. Penyimpanan rekam medis hanya dilakukan pada *mobile device* dan *database* rumah sakit. Menurut Undang-undang Republik Indonesia Nomor 29 Tahun 2004 pasal 47 dan diperkuat dengan Peraturan Menteri Kesehatan Republik Indonesia Nomor 269/MENKES/PER/III/2008, dokumen rekam medis merupakan milik rumah sakit, sedangkan isi dari

rekam medis adalah milik pasien. Hal di atas menjadi salah satu alasan mengapa tidak ada sebuah server pusat yang menyimpan semua rekam medis pasien.

3.1.2 Kebutuhan Infrastruktur

Kebutuhan infrastruktur minimal untuk membuat dan menjalankan sistem ini adalah:

- Koneksi jaringan Internet antara *device*, server, dan WCF *service* rumah sakit.
- *Mobile device* dengan sistem operasi Windows Mobile 6 yang memiliki kemampuan GPS.
- Sebuah server sebagai server pusat.
- Sebuah server sebagai WCF *service* RS.

3.1.3 *Actor Glossary*

Tabel 3 di bawah ini adalah tabel yang berisikan deskripsi peranan dari suatu tipe pengguna pada sistem TRuST.

Tabel 3. ***Actor Glossary***

No.	*Actor*	*Description*
1	Pasien	Aktor ini adalah orang yang berperan sebagai pasien yang membutuhkan *healthcare*, yang dapat mengaktifkan aplikasi TRuST sesuai dengan rekam medisnya yang tersimpan di *device*.
2	Admin RS	Aktor ini adalah administrator dari masing-masing sistem di rumah sakit, yang memiliki wewenang untuk meng-*update* rekam medis pasien dan melihat adanya *warning signal healthcare reservation* dan *urgent signal.*
3	*Device*	Aktor ini adalah sebuah *mobile device* yang dapat mengirimkan *signal healthcare reservation* dan *urgent signal* kepada WCF rumah sakit setelah pasien melakukan aktifasi terhadap *signal healthcare*

No.	*Actor*	*Description*
		reservation dan *urgent signal.*
4	Server	Aktor ini adalah sebuah server yang dapat melakukan penghitungan jarak antara rumah sakit dengan *pasien*.

3.1.4 *Use Case Glossary*

Tabel 4 di bawah ini adalah tabel yang berisikan gambaran kebutuhan-kebutuhan sistem TRuST secara umum.

Tabel 4. *Use Case Glossary*

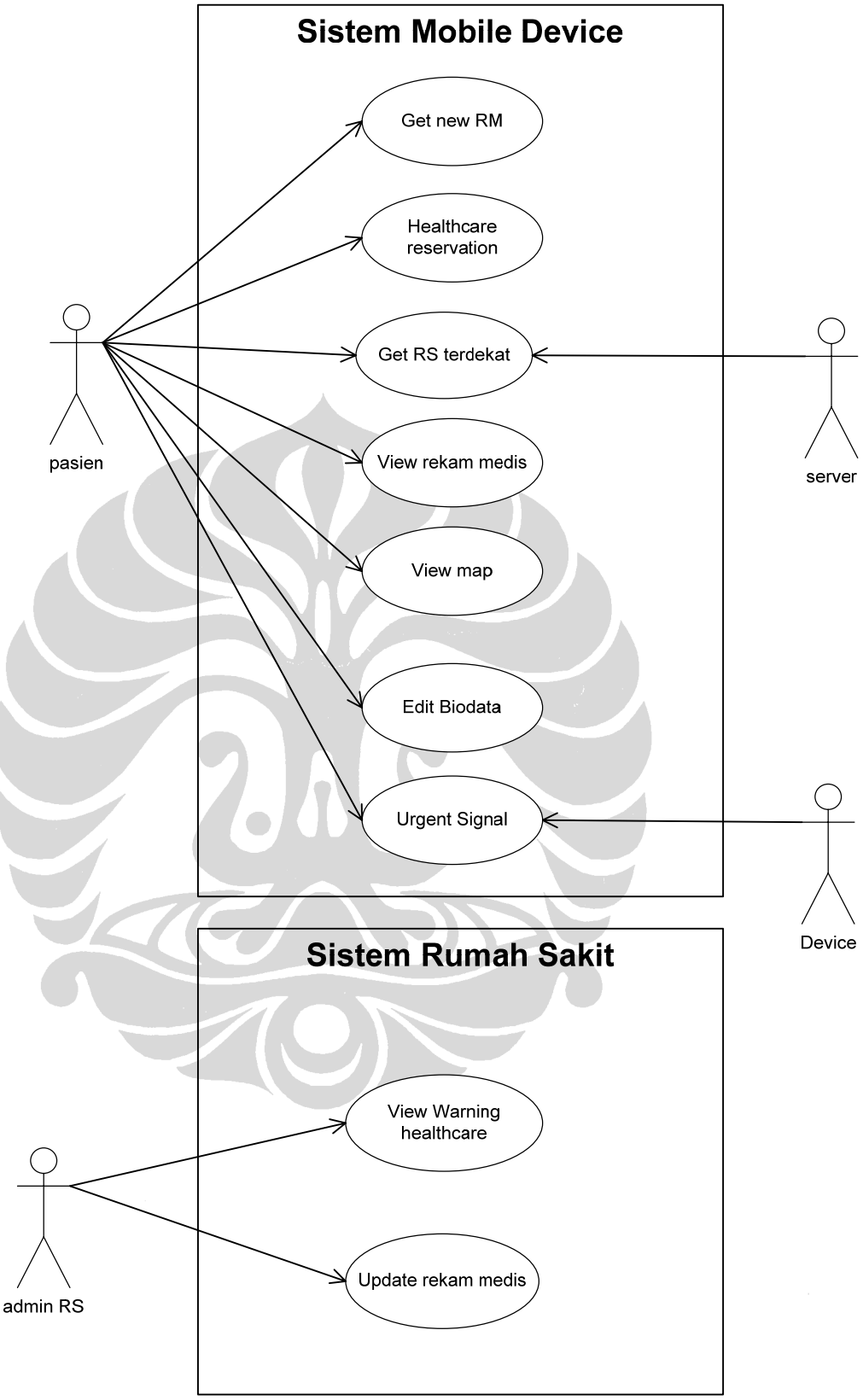
No	Nama *Use case*	Dekripsi *Use case*	Pelaku & Perannya
1	Get RS Terdekat	*Mobile device* dapat membantu pasien dalam memberikan info tentang rumah sakit terdekat dalam radius jarak tertentu yang mampu melayani pasien dengan penyakit seperti yang diderita pasien serta menampilkan detil spesialis dokter yang bersesuaian yang *available* saat itu.	Pasien, server
2	*Healthcare reservation*	Pasien dapat memberikan notifikasi bahwa ia akan berobat di salah satu rumah sakit dengan memilih salah satu rumah sakit yang masuk ke dalam daftar rumah sakit terdekat untuk melakukan pemesanan layanan.	Pasien
3	*View Map*	*Mobile device* dapat memperlihatkan letak posisi pasien dan posisi rumah sakit yang terdekat.	Pasien
4	*Urgent Signal*	*Pasien* dapat mengirimkan *urgent signal* kepada rumah sakit yang memiliki spesialis yang dibutuhkannya dan memiliki jarak paling dekat agar dikirimkan tim medis ke lokasinya.	Pasien, *Device*

No	Nama *Use case*	Dekripsi *Use case*	Pelaku & Perannya
5	*Update* rekam medis	Memperbaharui rekam medis *pasien* dengan data kesehatan yang terbaru, dimasukan setelah *pasien* melakukan pemeriksaan.	Admin RS
6	*View* rekam medis	*Mobile device* dapat menampilkan rekam medis pasien yang disimpan dalam *mobile device*.	*Pasien*
7	*Get new* Rekam medis	Pasien dapat meng-*update* rekam medis miliknya yang tersimpan di *database* rumah sakit tertentu.	Pasien
9	*Edit* biodata	Pasien dapat mengubah biodata miliknya yang tersimpan dalam mobile device.	Pasien
8	*View Warning healthcare*	Admin RS dapat melihat *warning* adanya orang yang akan membutuhkan *healthcare* di rumah sakit tersebut.	Admin RS

3.2 Analisis *Use Case*

Dalam menganalisis kebutuhan, penulis menggunakan *use-case diagram* untuk memodelkan hasil analisis yang diperoleh. Dengan diagram ini, penulis dapat mengkomunikasikan kepada narasumber mengenai fungsi yang harus dibuat penulis beserta aktor-aktor yang terlibat. *Use-case* diagram adalah suatu diagram yang menggambarkan siapa yang akan menggunakan sistem dan dengan cara apa pengguna diharapkan untuk berinteraksi dengan sistem [WHI04].

Kebutuhan-kebutuhan dari sistem secara umum dapat digambarkan dalam diagram *use case* pada **Gambar 9** berikut ini:



Gambar 9. *Use Case* Diagram

3.3 *Business Process*

Untuk memberikan gambaran tentang keterkaitan proses antara elemen sistem, penulis menggambarkannya dengan menggunakan sebuah diagram *business process*.

Bussiness process adalah kerja, prosedur, dan aturan yang dibutuhkan untuk menyelesaikan tugas bisnis, independen terhadap teknologi yang digunakan untuk mengotomasi atau mendukung mereka [WHI04].

Business process sistem yang penulis kembangkan dapat dilihat pada **Lampiran A**.

3.4 Pemodelan

Dalam pengembangan sistem TRuST dilakukan pemodelan terhadap data yang ada di rumah sakit maupun yang tersimpan di *mobile device* dan server pusat serta pemodelan ontologi yang digunakan pada sistem ini.

3.4.1 Pemodelan Data

Setelah kebutuhan-kebutuhan sistem terhimpun dalam *use case diagram*, penulis selanjutnya melakukan pemodelan terhadap data-data yang akan digunakan dalam sistem. *Database* yang nantinya dibuat adalah *database* yang ada pada sistem rumah sakit yang terintegrasi dengan data yang ada pada WCF *service* rumah sakit.

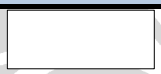
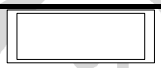
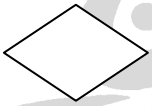
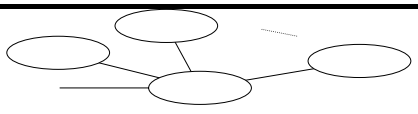
Dalam tahapan ini, penulis mencoba menggambarkan *database* TRuST dengan menggunakan *Entity Relationship Diagram* (ERD) sebagai bentuk dari pemodelan data.

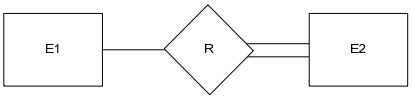
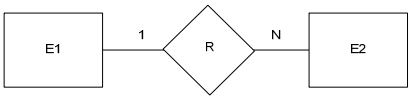
Entity Relationship Diagram (ERD) adalah salah satu pemodelan data menggunakan beberapa notasi untuk menggambarkan data dalam bentuk entitas-entitas dan relasi-relasi yang digambarkan oleh data tersebut. Entitas adalah sebuah *class* dari orang, tempat, objek, *event*, atau konsep tentang apa yang dibutuhkan untuk mengambil dan menyimpan data. Setiap entitas memiliki atribut, yaitu suatu properti yang deskriptif atau karakteristik dari suatu entitas [WHI04].

Entitas dibedakan menjadi dua tipe, yaitu *strong entity* dan *weak entity*. *Strong entity* adalah tipe entitas yang memiliki atribut *key*, dan keberadaannya tidak tergantung pada entitas lain. Sedangkan *weak entity* merupakan entitas yang tidak memiliki atribut *key* sendiri. Atribut *key* adalah atribut yang nilainya berbeda untuk masing-masing entitas, dan bisa digunakan untuk mengidentifikasi masing-masing entitas secara unik [ELM00]. Salah satu jenis atribut lainnya adalah *composite attribute*, yaitu atribut yang dapat dibagi menjadi bagian-bagian yang lebih kecil yang merepresentasikan lebih banyak atribut dasar dengan keterkaitan makna [ELM00].

Notasi ERD yang digunakan dapat dilihat pada **Tabel 5**.

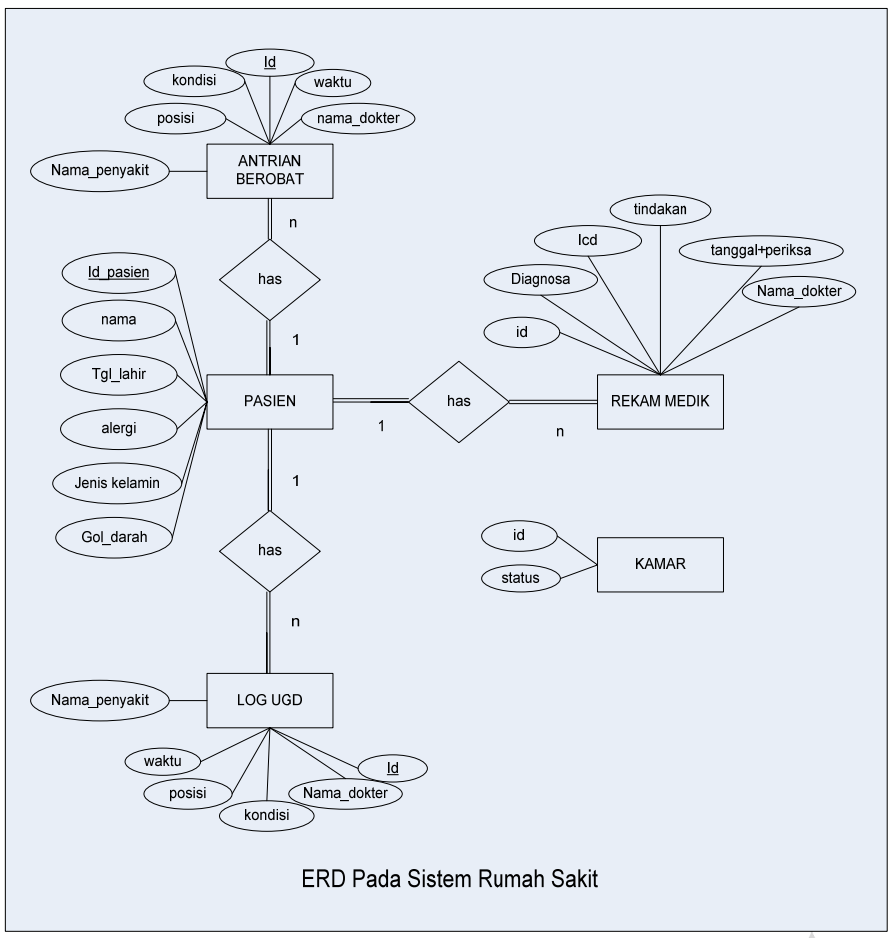
Tabel 5. Notasi ERD [WHI04]

Notasi	Keterangan
	*Strong entity*
	*Weak entity*
	Relasi
	Atribut
	Atribut *key*
	*Multivalue attribute*
	*Composite attribute*

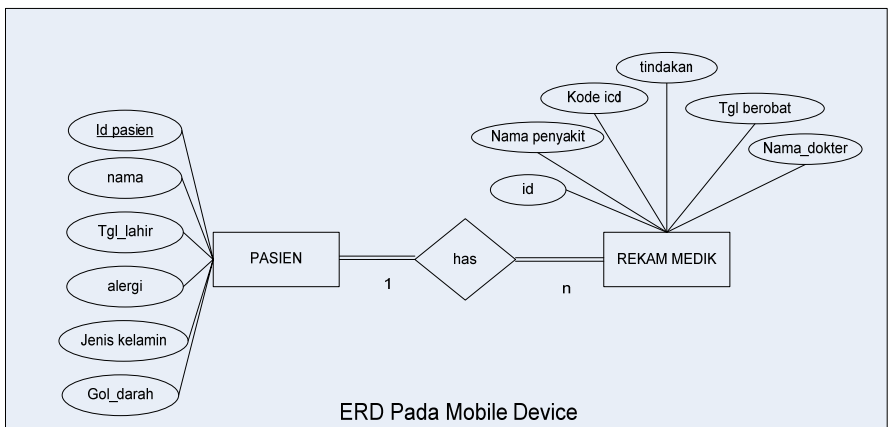
Notasi	Keterangan
	Partisipasi total dari E2 pada R
	Kardinalitas 1:N untuk E1:E2 pada R

Relasi adalah suatu hubungan atau keterkaitan antara satu atau lebih entitas [WHI04]. Ada dua tipe *constraint* pada relasi yaitu kardinalitas dan partisipasi. Suatu relasi biner memiliki kardinalitas yang menspesifikasikan jumlah keterhubungan di mana suatu entitas berpartisipasi. Sedangkan partisipasi suatu entitas pada relasi ada dua macam yaitu total, di mana semua anggota entitas terlibat pada relasi tersebut; dan relasi parsial, di mana tidak semua anggota entitas terlibat dalam relasi tersebut [ELM00].

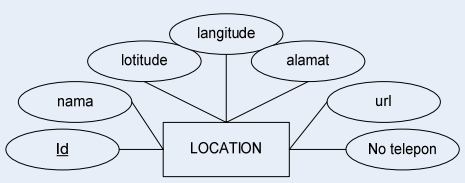
Gambar 10 berikut ini adalah gambar dari ERD yang merepresentasikan pemodelan data yang digunakan pada sistem rumah sakit di sistem TRuST.



ERD Pada Sistem Rumah Sakit



ERD Pada Mobile Device



ERD Pada Server Pusat

Gambar 10. ERD *Database* Sistem TRuST

Entitas yang ada pada *database* sistem rumah sakit terdiri dari:

Rekam medik adalah entitas yang merepresentasikan rekam medis pasien. Atribut yang dimiliki oleh entitas ini adalah:

id	Atribut yang berisikan nomor identitas unik dari rekam medis
diagnosa	Atribut yang berisikan nama penyakit yang diderita *user*
kode ICD	Atribut yang berisikan kode ICD penyakit yang diderita *user*
nama dokter	Atribut yang berisikan nama dokter yang memeriksa *user*
waktu berobat	Atribut yang berisikan tanggal *user* melakukan pemeriksaan
tindakan	Atribut yang berisikan tindakan/operasi yang dilakukan

Pasien adalah entitas yang merepresentasikan data pribadi pasien. Atribut yang dimiliki oleh entitas ini adalah:

nama	Atribut yang berisikan nama dari *user*
id	Atribut yang berisikan no id dari *user*
tanggal lahir	Atribut yang berisikan tanggal lahir *user*
jenis kelamin	Atribut yang berisikan jenis kelamin dari *user*
golongan darah	Atribut yang berisikan golongan darah dari *user*
alergi	Atribut yang berisikan alergi yang diderita *user*

Antrian UGD adalah entitas yang merepresentasikan daftar urutan pemesanan UGD. Atribut yang dimiliki oleh entitas ini adalah:

id	Atribut yang berisikan id log UGD
waktu	Atribut yang berisikan waktu pemesanan UGD
kondisi	Atribut yang menyatakan kondisi pasien saat itu.
posisi	Atribut yang menyatakan posisi pasien saat itu.
ICD	Atribut yang menyatakan kode penyakit yang diderita.
nama_penyakit	Atribut yang menyatakan nama penyakit yang diderita

Antrian Berobat adalah entitas yang daftar urutan antrian berobat. Atribut yang dimiliki oleh entitas ini adalah:

id	Atribut yang berisikan id antrian berobat
waktu	Atribut yang berisikan waktu saat adanya *user* yang mendaftar untuk berobat
kondisi	Atribut yang menyatakan kondisi pasien saat itu.
posisi	Atribut yang menyatakan posisi pasien saat itu.
nama_dokter	Atribut yang menyatakan nama dokter yang spesialis yang sanggup menangani pasien dengan penyakit tertentu
ICD	Atribut yang menyatakan kode penyakit yang diderita.
nama_penyakit	Atribut yang menyatakan nama penyakit yang diderita

Kamar adalah entitas yang daftar kamar yang dimiliki rumah sakit. Atribut yang dimiliki oleh entitas ini adalah:

id	Atribut yang berisikan id kamar di rumah sakit
status	Atribut yang berisikan status dari kamar tersebut *available* atau tidak

Relasi yang ada pada *database* tersebut adalah sebagai berikut:

HAS

- Merupakan relasi antara Pasien dan Rekam medik
- *Total Participation of* Pasien in HAS
- *Total Participation of* Rekam medik in HAS
- *Cardinality Ratio* 1:n *for* Pasien : Rekam medik

HAS ANTRIAN

- Merupakan relasi antara Pasien dan Antrian Berobat
- *Total Participation of* Pasien in HAS ANTRIAN

- *Total Participation of* Rekam medik in HAS ANTRIAN
- *Cardinality Ratio* 1:n *for* Pasien : Antrian Berobat

HAS LOG UGD

- Merupakan relasi antara Pasien dan Antrian UGD
- *Total Participation of* Pasien in HAS LOG UGD
- *Total Participation of* Antrian UGD in HAS LOG UGD
- *Cardinality Ratio* 1:n *for* Pasien : Antrian UGD

Untuk *database* yang di simpan di *mobile device* dibuat 2 buah tabel berikut ini yang kesemuanya adalah data pribadi seorang pasien per setiap *mobile device*.

Rekam medik adalah entitas yang merepresentasikan rekam medis pasien. Atribut yang dimiliki oleh entitas ini adalah:

id	Atribut yang berisikan nomor identitas unik dari rekam medis
diagnosa	Atribut yang berisikan nama penyakit yang diderita *user*
kode ICD	Atribut yang berisikan kode ICD penyakit yang diderita *user*
nama dokter	Atribut yang berisikan nama dokter yang memeriksa *user*
waktu berobat	Atribut yang berisikan tanggal *user* melakukan pemeriksaan
tindakan	Atribut yang berisikan tindakan/operasi yang dilakukan

Pasien adalah entitas yang merepresentasikan data pribadi pasien. Atribut yang dimiliki oleh entitas ini adalah:

nama	Atribut yang berisikan nama dari *user*
tanggal lahir	Atribut yang berisikan tanggal lahir *user*
jenis kelamin	Atribut yang berisikan jenis kelamin dari *user*
golongan darah	Atribut yang berisikan golongan darah dari *user*
alergi	Atribut yang berisikan alergi yang diderita *user*

Relasi yang ada pada *database* tersebut adalah sebagai berikut:

HAS

- Merupakan relasi antara Pasien dan Rekam medik
- *Total Participation of* Pasien in HAS
- *Total Participation of* Rekam medik in HAS
- *Cardinality Ratio* 1:n *for* Pasien : Rekam medik

Sedangkan di server pusat terdapat *database* yang menyimpan sebuah tabel tunggal, yakni **Location**.

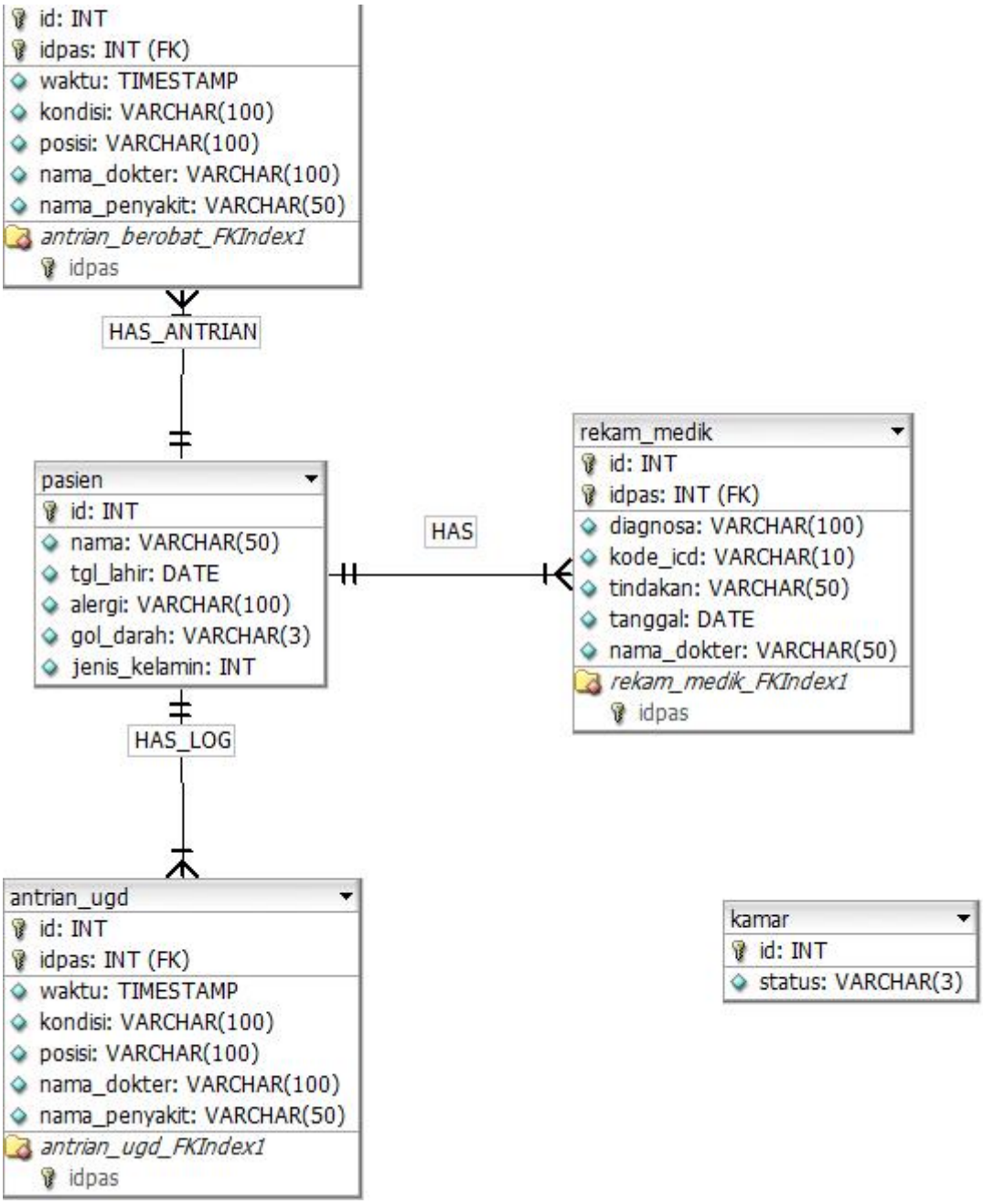
Location adalah entitas yang merepresentasikan daftar rumah sakit yang telah mendaftarkan WCF *service* nya. Atribut yang dimiliki oleh entitas ini adalah:

id	Atribut yang berisikan nomor identitas unik dari WCF *service* rumah sakit
nama	Atribut yang berisikan nama rumah sakit
latitude	Atribut yang berisikan posisi *latitude* rumah sakit
longitude	Atribut yang berisikan posisi *longitude* rumah sakit
alamat	Atribut yang berisikan alamat rumah sakit
telp	Atribut yang berisikan no telepon rumah sakit
url	Atribut yang berisikan alamat url WCF *service* rumah sakit

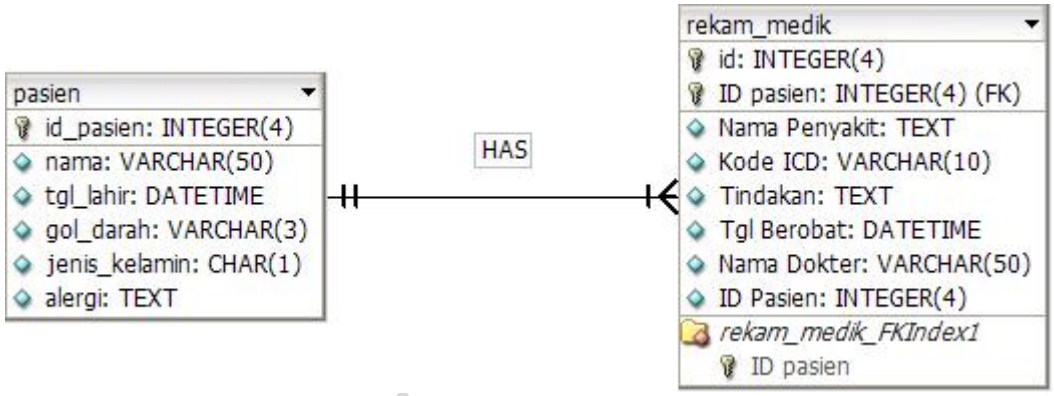
ERD yang berhasil dibentuk kemudian dianalisis lebih jauh oleh tim pengembang dengan melakukan kegiatan pemetaan. Setelah melakukan pemetaan tabel dan normalisasi sampai tahap normalisasi 3NF, dihasilkan 5 tabel untuk *database* di rumah sakit, 2 tabel untuk *database* yang tersimpan di mobile device dan 1 tabel yang tersimpan di server pusat. Pada diagram hasil pemetaan ERD dapat dilihat hubungan yang lebih jelas dari satu Entitas dengan Entitas lainnya dan bagaimana struktur fisik dari *database* TRuST yang direpresentasikan dengan menggunakan sebuah skema basis data (*database schema*).

Database schema adalah model fisik atau sebuah cetak biru untuk sebuah basis data. *Database schema* merepresentasikan implementasi teknis dari *logical data model*. *Database schema* mendefinisikan struktur basis data menurut tabel, *key*, *index*, dan aturan-aturan integritas.

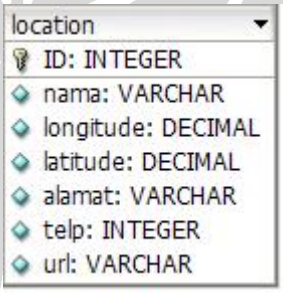
Database schema yang telah dibuat dapat dilihat pada **Gambar 11**, **Gambar 12** dan **Gambar 13** di bawah ini.



Gambar 11. *Database Schema* Sistem Rumah Sakit TRuST



Gambar 12. *Database Schema Mobile Device* Sistem TRuST



Gambar 13. *Database Schema* Server Pusat Sistem TRuST

Penjelasan untuk masing-masing atribut tabel baik yang ada di *mobile device*, server pusat maupun sistem rumah sakit dapat dilihat di **Lampiran B**.

3.4.2 Pemodelan Ontologi

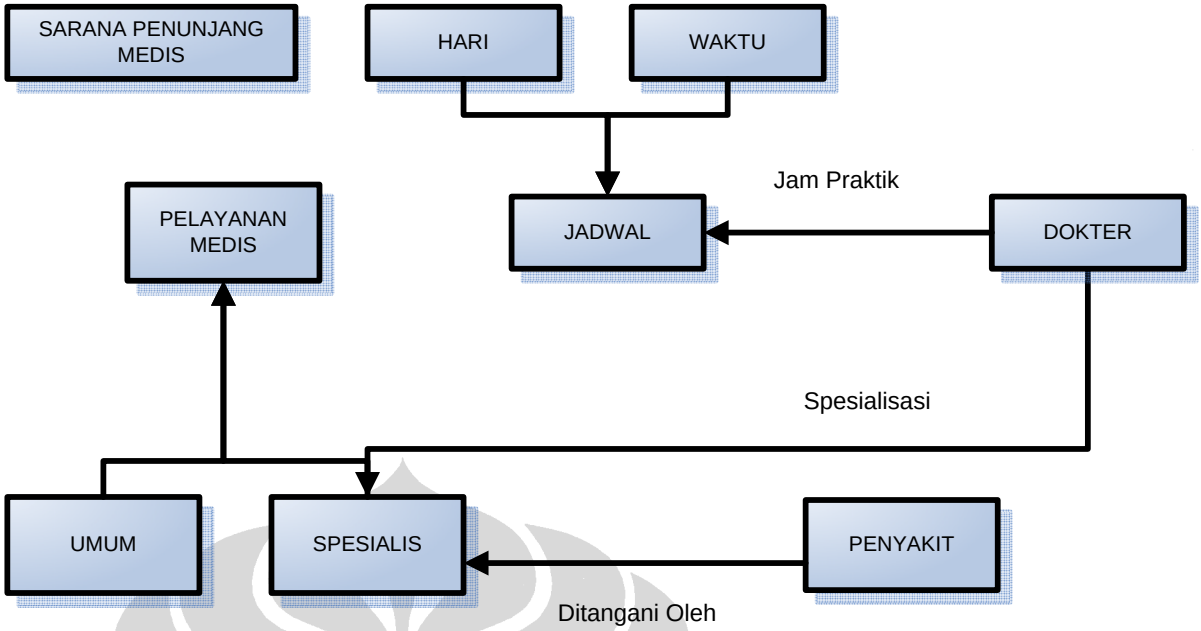
Dalam tahap analisis ini, penulis mencoba mendaftarkan konsep (untuk membangun ontologi) apa saja yang diperlukan dalam pengembangan sistem ini agar setiap elemen sistem yang satu dengan yang lainnya dapat berkomunikasi dan mengerti dengan konsep yang sama.

Tabel 6 berikut ini adalah daftar konsep utama yang dibutuhkan dalam pengembangan sistem.

Tabel 6. Daftar Konsep Ontologi

No	Konsep	Keterangan
1	PENYAKIT	Adalah konsep tentang klasifikasi penyakit yang diderita oleh *user*. Dalam pengembangan sistem ini, penulis mencoba membuat taksonomi jenis penyakit kronis, yakni jenis penyakit yang sifatnya menahun, lebih spesifik lagi, untuk jenis penyakit sirkulasi, saraf, dan pernapasan.
2	PELAYANAN MEDIS	Adalah konsep tentang jenis layanan medis apa yang dimiliki oleh sebuah rumah sakit
3	JADWAL	Adalah konsep tentang waktu avaibilitas JADWAL dokter tertentu.
4	SARANA PENUNJANG MEDIS	Adalah konsep tentang fasilitas rumah sakit yang dapat menunjang layanan medis yang disediakan rumah sakit untuk meningkatkan kenyamanan dan kesehatan pasien.
5	DOKTER	Adalah konsep tentang DOKTER yang memiliki spesialisasi tertentu yang menangani pasien dengan penyakit yang termasuk dispesialisasinya.

Keterkaitan antara konsep yang satu dengan yang lainnya, dapat dilihat pada **Gambar 14** berikut ini.



Gambar 14. Keterkaitan Antara Konsep Ontologi

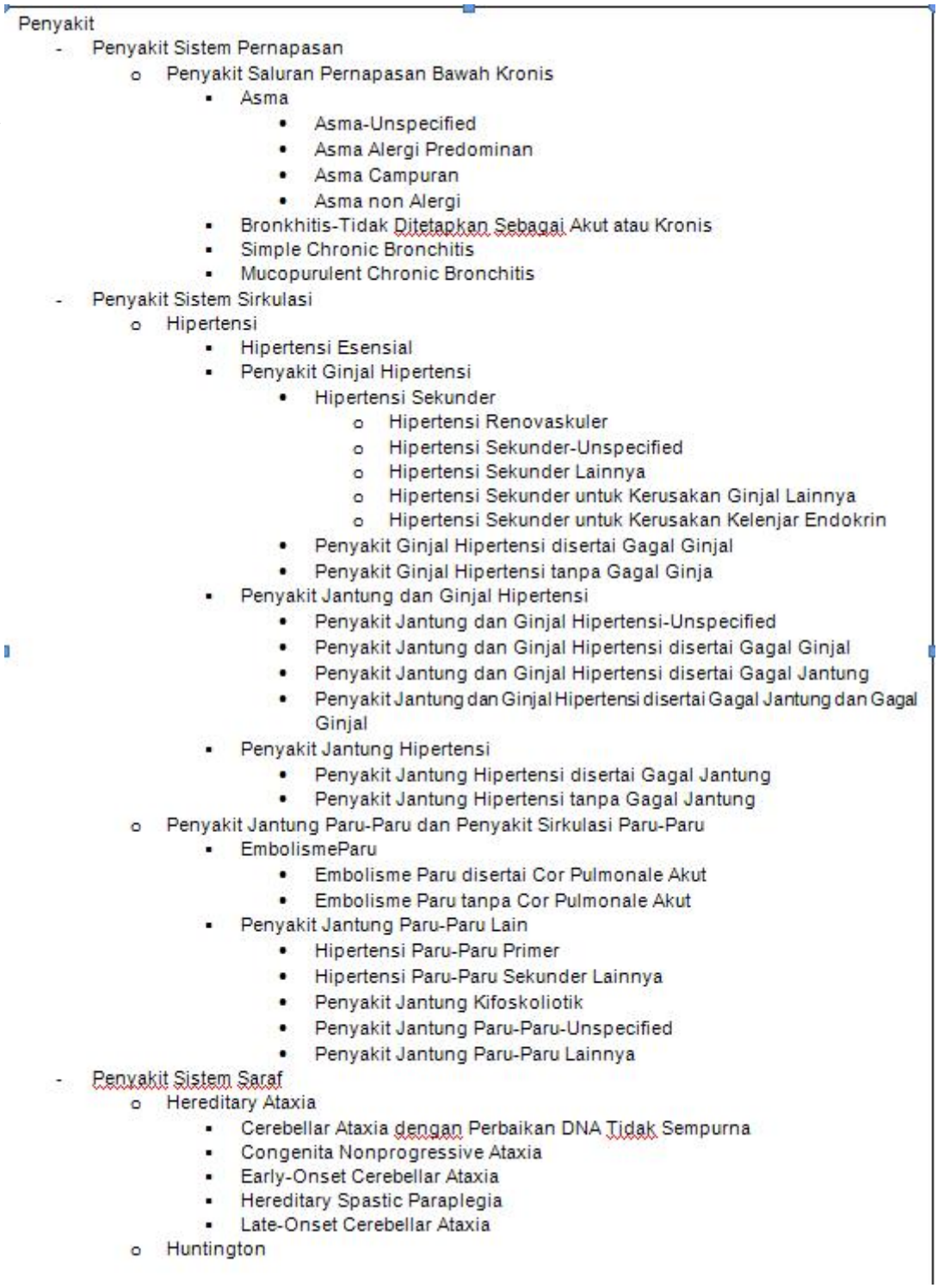
DOKTER nantinya akan berhubungan dengan **JADWAL** untuk mendapatkan jadwal praktik dari dokter sedangkan **SPESIALIS** dan **PENYAKIT** masing-masing saling berhubungan satu dengan yang lainnya. **DOKTER** akan menjadi **SPESIALIS** tertentu dan **PENYAKIT** tertentu dapat ditangani oleh **SPESIALIS** tertentu pula.

Pada tahap ini juga dilakukan klasifikasi pada konsep **PENYAKIT**, **JADWAL**, **PELAYANAN MEDIS**, dan **SARANA PENUNJANG MEDIS.**

Berikut ini adalah klasifikasi yang telah dibuat.

1. Klasifikasi penyakit

Konsep penyakit yang diklasifikasikan adalah 3 jenis penyakit kronis yang paling banyak terjadi yakni, penyakit sistem pernapasan, penyakit sistem saraf, dan penyakit sistem sirkulasi. Taksonomi ketiga jenis kelompok penyakit itu yang diperlihatkan pada **Gambar 15** berikut mengacu pada *International Statistical Classification of Diseases and Related Health Problems 10th Revision Version for* 2007 (ICD 10) [SCH08]



Gambar 15. Klasifikasi Penyakit [SCH08]

2. Klasifikasi Pelayanan Medis

Konsep pelayanan medis juga dibagi menjadi dua, yaitu pelayanan medis yang melayani penyakit-penyakit yang harus ditangani oleh spesialis tertentu dan pelayanan medis umum. Pelayanan medis umum melayani penyakit yang sifatnya umum. Sedangkan pelayanan medis yang sifatnya spesialis diklasifikasikan lagi menjadi spesialis yang menangani penyakit dalam seperti spesialis paru-paru dan jantung serta spesialis yang menangani saraf.

3. Klasifikasi Sarana Penunjang Medis

Sarana penunjang medis diklasifikasikan menjadi unit-unit yang menangani bagian UGD, farmasi, kamar operasi, *ambulance* dan unit laboratorium. Sedangkan unit laboratorium dibagi lagi menjadi laboratorium serologi, laboratorium parasitologi, laboratorium histopatologi, laboratorium patalogi, laboratorium anatomi, laboratorium imunologi, laboratorium klinik, laboratorium biochemistry, laboratorium sitologi, laboratorium mikrobiologi, dan laboratorium hematologi.

4. Klasifikasi Jadwal

Jadwal diklasifikasikan terdiri atas komponen hari dan jam. Komponen hari didapatkan dari nama-nama hari dalam satu minggu yakni, Senin, Selasa, Rabu, Kamis, Jumat, Sabtu, dan Minggu. Sedangkan komponen jam dibagi menjadi renatang jam dalam satu hari per satuan jam.

Implementasi dan penjelasan ontologi dalam sistem ini, yang terdiri atas *class*, *subclass*, slot atau *property* dan data *type* tiap *property*, dapat dilihat di bagian Implementasi Ontologi pada Bab 5.

BAB 4
PERANCANGAN

Pada bab ini akan berisi penjelasan mengenai proses perancangan sistem yang meliputi perancangan arsitektur dan implementasi ontologi yang digunakan pada sistem dengan menggunakan RDF. Proses perancangan ini dilakukan setelah tahap analisis diselesaikan.

4.1 Arsitektur Sistem

Arsitektur untuk sistem TRuST terbagi atas 3 yakni arsitektur untuk *mobile device*, arsitektur pada server pusat dan arsitektur untuk rumah sakit. Masing-masing arsitektur merujuk pada arsitektur AWARENESS PROJECT. Tidak semua fungsi dan *service* pada AWARENESS penulis adaptasikan. Misalnya seperti autentifikasi *service* atau *security mechanism service*. Elemen yang kami gunakan adalah *connectivity* dan *context management*. Selain itu beberapa elemen pada *application layer* juga penulis gunakan, seperti GUI atau *database*.

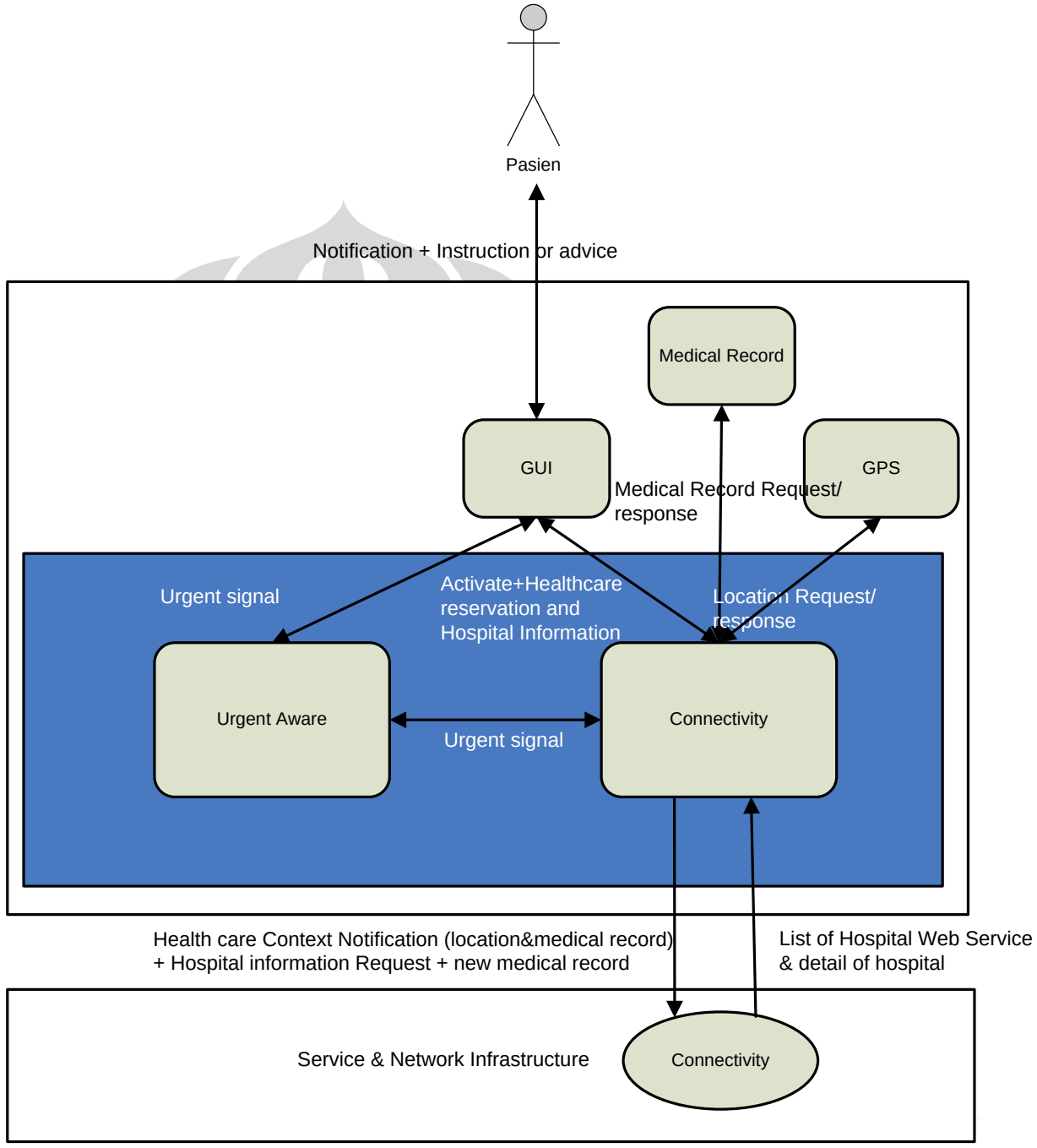
Masing-masing arsitektur akan dijelaskan pada subbab selanjutnya.

4.1.1 Arsitektur Mobile Device

Aplikasi (*context aware system*) yang berjalan di *mobile device* berada di atas *layer network* serta infrastruktur yang juga digunakan pada aplikasi WCF *service* rumah sakit serta WCF *service* pusat. Arsitektur untuk *mobile device* dapat dilihat pada **Gambar 16** di bawah ini. Di aplikasi ini terdapat beberapa komponen diantaranya:

- GUI: yang akan menjadi *interface* bagi *user* agar dapat mengaktifkan sistem TRuST ini. Dengan adanya bagian ini, *user* dapat meminta *device* untuk dapat mencarikan daftar rumah sakit terdekat dengan menekan tombol tertentu, selain itu detail spesialis yang dimiliki oleh rumah sakit yang masuk ke dalam himpunan rumah sakit terdekat juga akan ditampilkan. Notifikasi tentang *healthcare reservation* serta *urgent signal* pada rumah sakit tertentu juga di-*handle* pada bagian ini.

- GPS: yang akan memberikan *response context location* dari pasien jika ada elemen lain yang me-*request* lokasi dari pasien. Pada sistem TRuST, GPS yang digunakan adalah sebuah fake GPS yang akan membaca GPS data yang di-*generate* oleh sebuah generator NMEA0183 *message* berjenis RMC.



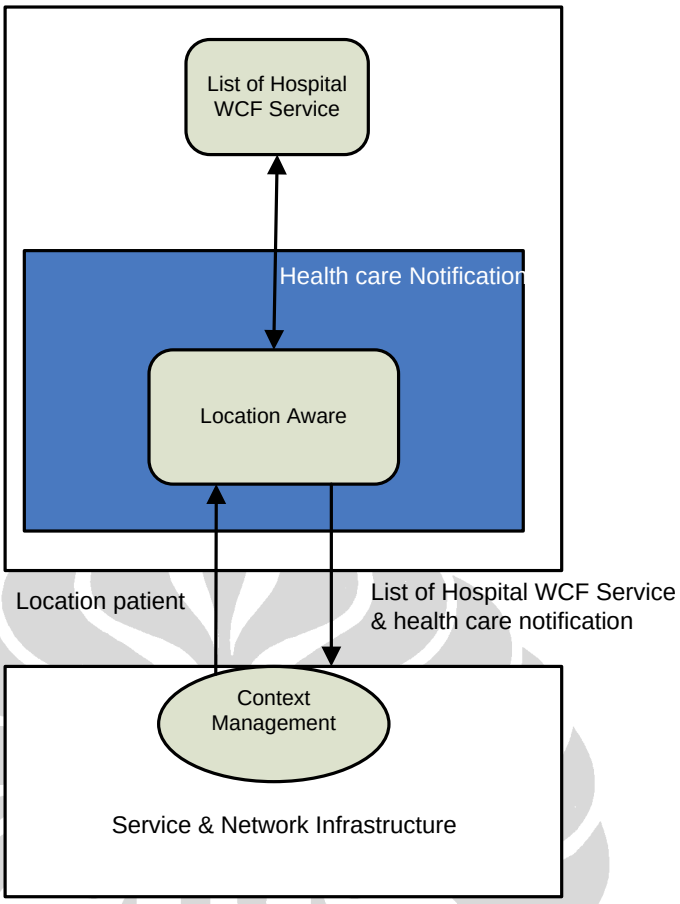
Gambar 16. Arsitektur *Mobile Device*

- *Medical Record*: berisi riwayat medis serta biodata singkat dari pasien. Bagian ini akan memberikan *response* atas *medical record user* jika ada elemen lain yang me-*request medical record* dari *user*.
- *Connectivity*: bagian yang menghubungkan antara koneksi aplikasi di *mobile* dengan aplikasi di server pusat ataupun WCF *service* rumah sakit.
- *Urgent Aware*: adalah bagian yang melakukan *reasoning* untuk mendapatkan satu rumah sakit paling dekat dan *available* untuk dikirimkan *healthcare notification*. Hal ini terjadi hanya saat *user* memilih menu *urgent* pada aplikasi di *mobile*.

4.1.2 Arsitektur Server

Server pusat adalah sebuah WCF *service* yang akan menjadi 'jembatan' untuk mempertemukan antara *mobie device* dan sistem rumah sakit, khususnya WCF *service* rumah sakit. Arsitektur untuk server pusat dapat dilihat pada **Gambar 17** di bawah ini. Pada server pusat terdapat beberapa elemen diantaranya:

- *List of* WCF *Hospital service*: adalah bagian yang menyimpan daftar WCF *service* rumah sakit yang sudah mendaftarkan *service*-nya untuk aplikasi ini. Daftar tersebut menyimpan nama, alamat, no telepon, posisi, dan *IP address* dari WCF *service* rumah sakit.
- *Location Aware*: adalah bagian yang meng-*handle context location* yang dikirimkan oleh *mobile device* dan akan menentukan beberapa rumah sakit yang terdekat dengan pasien. Bagian ini akan berhubungan dengan *context management* yang berada pada *service* dan *network layer*.
- *Context Management*: adalah bagian pada *service* dan *network* infrastuktur yang menangkap *context* yang diberikan oleh aplikasi lain dan memberikan *context* yang diminta oleh aplikasi lain.



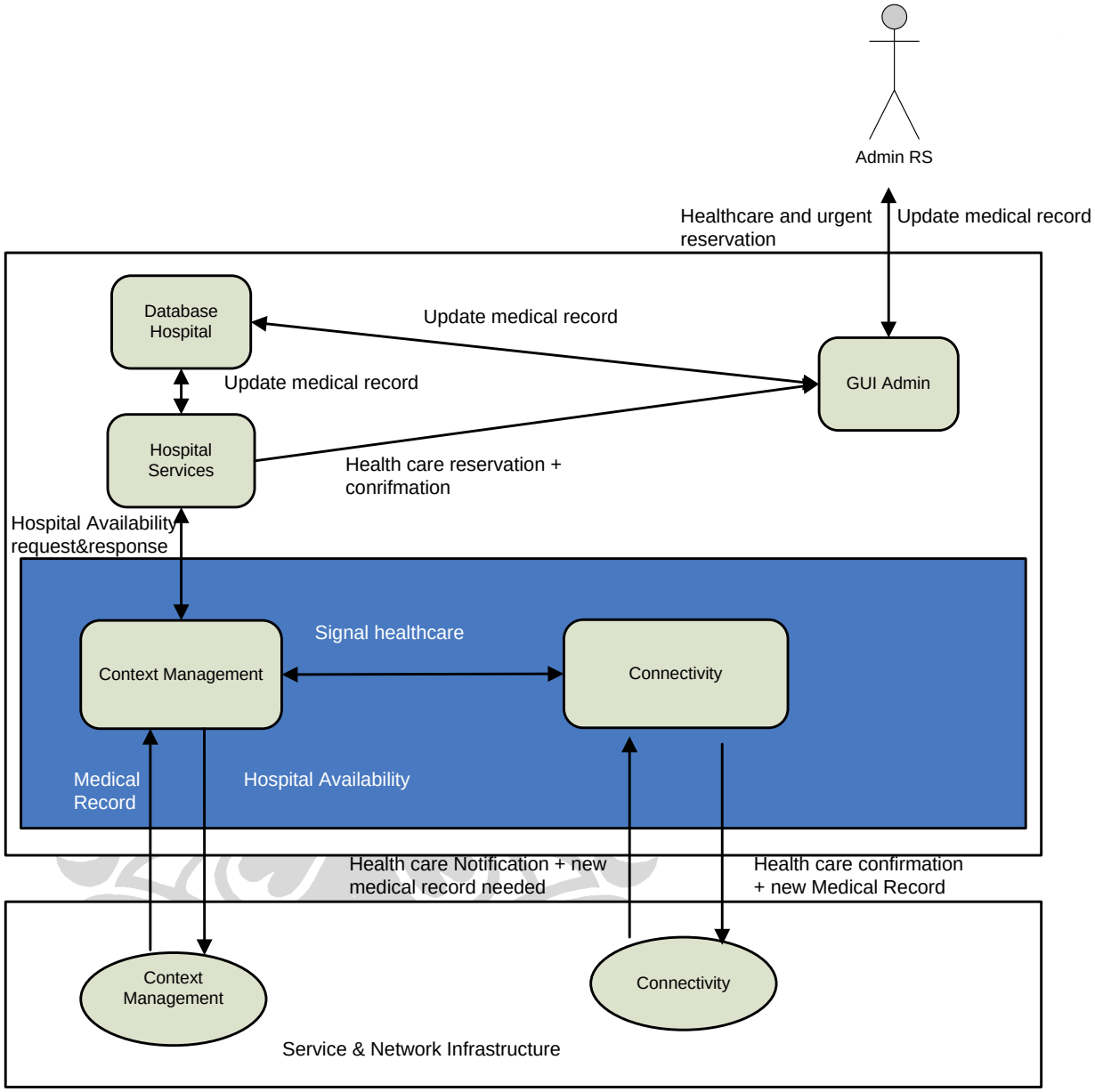
Gambar 17. Arsitektur Server Pusat

4.1.3 Arsitektur Rumah Sakit

Sistem yang berada di rumah sakit merupakan aplikasi penujang pada sistem TRuST yang secara garis besar terdiri atas sebuah WCF *service* dan aplikasi *desktop* yang saling terhubung. Arsitektur untuk sistem di rumah sakit dapat dilihat pada **Gambar 18.** Pada sistem rumah sakit terdapat beberapa komponen seperti berikut ini:

- *Hospital service*: adalah sebuah WCF *service* yang memberikan *service* untuk memberikan *response* atas *request* availabilitas dari rumah sakit untuk dapat menangani pasien serta untuk meng-*handle* adanya *request* rekam medis yang ada pada *database hospital*.
- *Database hospital* adalah bagian yang menyimpan *database* rumah sakit yang dapat membantu dalam menunjang rumah sakit untuk dapat meng-*handle*

request akan *healthcare reservation* dan *urgent signal*, serta jika ada *response* untuk *update* rekam medis (*medical record*).



Gambar 18. Arsitektur Sistem Rumah Sakit

- GUI: adalah bagian yang akan menampilkan *healthcare reservation* dan *urgent signal* yang terlebih dahulu telah ditangkap oleh *hospital service*. Selain itu pada bagian ini juga menjadi *interface* bagi seorang admin untuk dapat memasukan data rekam medis terbaru dari pasien yang nantinya tersimpan dalam database *hospital* untuk sewaktu-waktu dapat dikembalikan

oleh *hospital service* jika ada *user* yang ingin mendapatkan rekam medis terbarunya.

- *Preference aware*: yang akan mengolah *context* rekam medis pasien dengan *context hospital service* dan nantinya akan dikembalikan informasi availabilitas dokter spesialis yang dimiliki rumah sakit yang paling sesuai dengan kesehatan pasien.
- *Connection*: yang akan mengatur masalah koneksi antara WCF *service* rumah sakit dengan aplikasi pada *mobile device* atau dengan server pusat.
- *Context Management*: adalah bagian pada *service* dan *network* infrastuktur yang menangkap *context* yang diberikan oleh aplikasi lain dan memberikan *context* yang diminta oleh aplikasi lain.

4.2 Implementasi RDF

Hasil tahapan analisis tentang konsep-konsep ontologi yang akan menjadi *knowledge based* pada sistem TRuST pada tahap ini akan diimplementasikan dengan menggunakan dengan menggunakan sebuah RDF skema.

Pembuatan ontologi menggunakan suatu *tool* bernama Protégé (penjelasan *tool* dapat dilihat di bagian implementasi). Visualisasi ontologi dapat dilihat pada **Tabel 7** berikut ini.

Tabel 7. Visualisasi RDF

Notasi	Penjelasan
	sebuah *class* atau *subclass*
	memiliki atribut yang bernilai *class* dari *node superclass* yang dituju panah
	memiliki atribut yang bernilai *instance* dari *class* yg dituju

Notasi	Penjelasan
	*Subclass* dari *class* yang dituju
	*Instance* dari *class* yang dituju
Nama_Atribut*	Nilai atributnya bisa lebih dari satu

Langkah-langkah yang dilakukan dalam mengimplementasikan sebuah ontologi mencakup tahapan seperti disebut di bawah ini [NOY08].

1. Mendefinisikan *class* atau konsep di dalam ontologi.
2. Memasukan setiap *class* ke dalam taksonomi hierarki.
3. Mendefinisikan slot dan menentukan *value* dari slot.
4. Mengisikan *value* dari slot untuk membuat sebuah *instance.*

Berikut ini akan dipaparkan hasil dari setiap tahap pengembangan ontologi yang digunakan pada sistem ini.

4.2.1 Definisi class

Class yang ada pada ontologi [NOY08] terdiri atas *class* berikut ini yang terbagi atas 3 *level class*, yakni, *top level, middle level* dan *bottom level*.

- *Top level* adalah *level* di mana sebuah *class* adalah sebuah *specific class.*
- *Middle level* adalah *level* di mana sebuah *class* yang menjadi *subclass* dari *class* tertentu dan memiki *class-class* lain yang menjadi *subclass* nya.

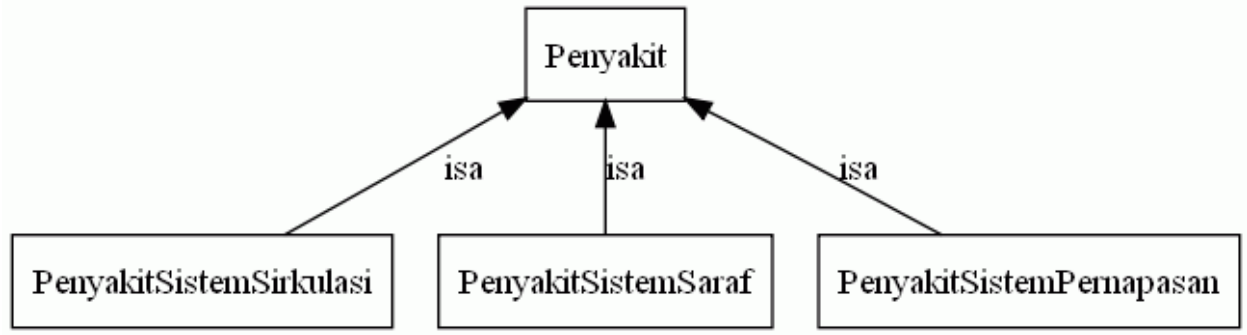
- *Bottom Level* adalah *level* di mana diletakkan *general class*.

Daftar *class-class* yang masuk ke dalam *Top, Middle* atau *Bottom Class* dapat dilihat di **Lampiran C.**

4.2.2 Taksonomi Hierarki

Setelah mengklasifikasikan tipe dari masing-masing *class*, tahapan selanjutnya adalah memasukannya ke dalam taksonomi hierarki yang bersesuaian. Berikut ini adalah gambar taksonomi hierarki yang dibuat pada sistem TRuST.

1. Taksonomi Penyakit

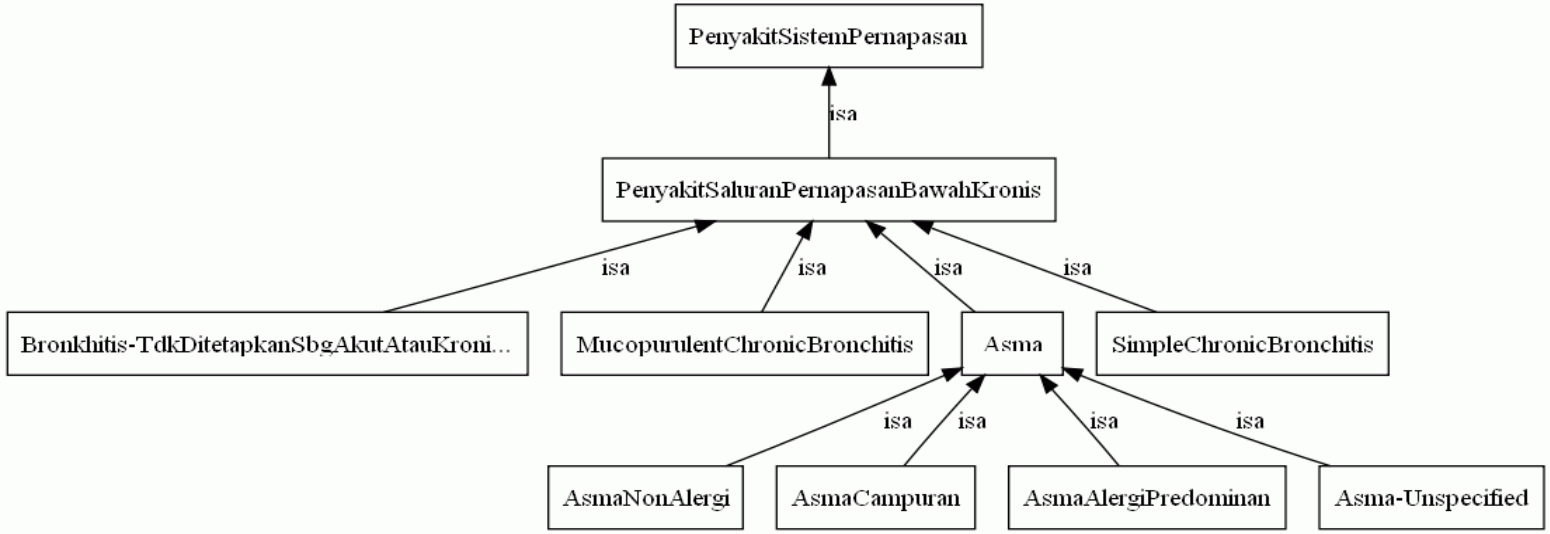


Gambar 19. Taksonomi Penyakit

Class penyakit di klasifikasikan menjadi 3 besar yakni **Penyakit Sistem Sirkulasi**, **Penyakit Sistem Saraf** dan **Penyakit Sistem Pernapasan**.

Gambar 19 di atas menyatakan bahwa *class* **Penyakit Sistem Sirkulasi**, *class* **Penyakit Sistem Saraf** dan *class* **Penyakit Sistem Pernapasan** adalah *subclass* dari *class* **Penyakit**. Masing-masing taksonomi akan dijelaskan pada gambar selanjutnya.

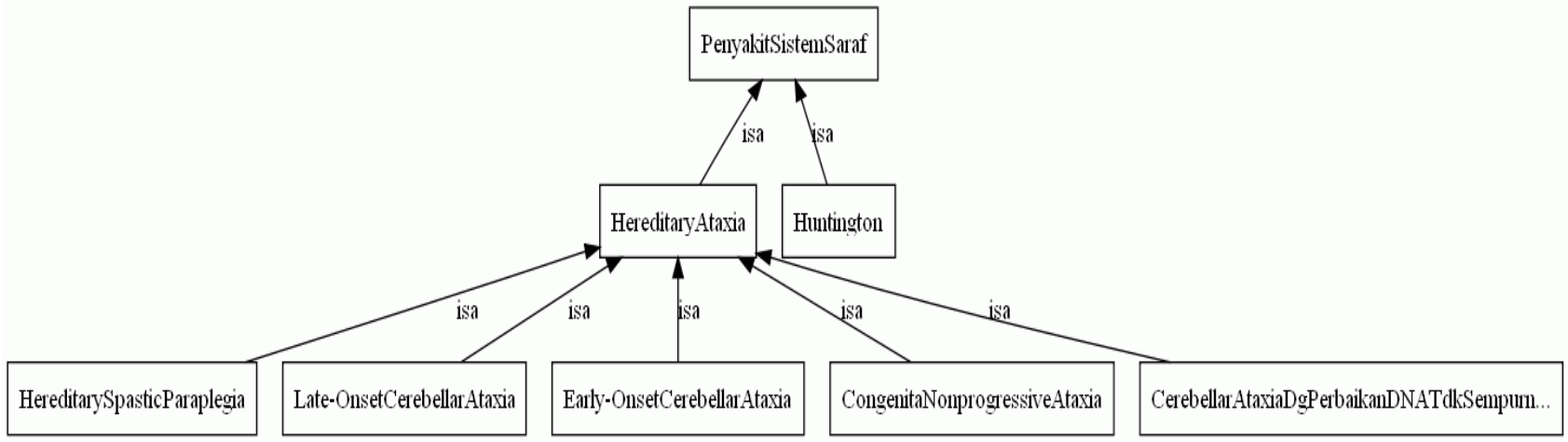
2. Taksonomi Penyakit Sistem Pernapasan



Gambar 20. Taksonomi Penyakit Sistem Pernapasan

Gambar 20 di atas adalah taksonomi yang dilakukan pada *class* **PenyakitSistemPernapasan**. *Class* **AsmaNonAlergi**, **AsmaCampuran**, **Asma AlergiPredominan** dan **Asma-Unspecified** merupakan *subclass* dari *class* **Asma**. Sedangkan *class* **Asma**, *class* **BronkhitisTidakDitetapkanSebagaiAkutAtau Kronis**, *class* **MucopurulentChronicBronchitis** dan *class* **SimpleChronic Bronchitis** adalah *subclass* dari *class* **PenyakitSaluranPernapasanBawah Kronis** yang menjadi *subclass* dari *class* **PenyakitSistemPernapasan**.

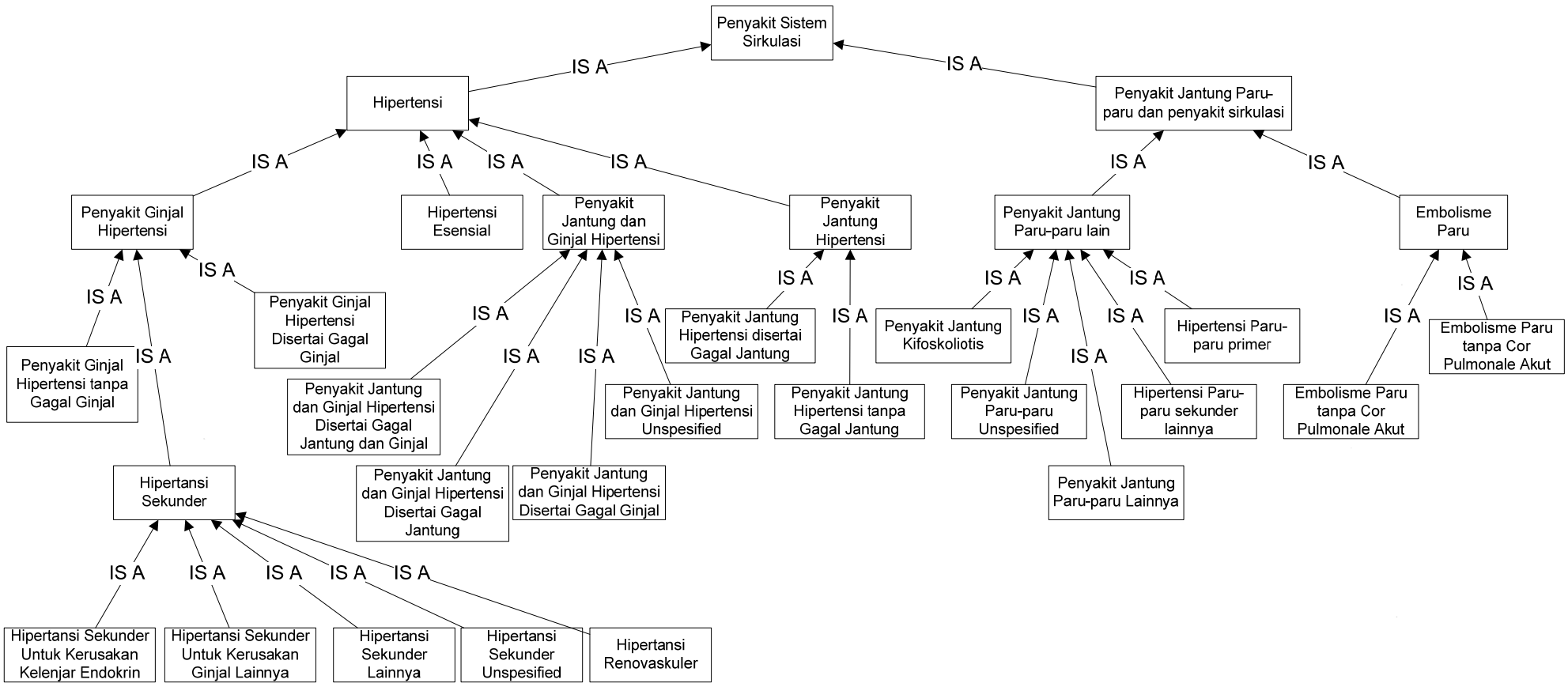
3. Taksonomi Penyakit Sistem Saraf



Gambar 21. Taksonomi Penyakit Sistem Saraf

Gambar 21 di atas adalah taksonomi yang dilakukan pada *class* **PenyakitSistemSaraf**. *Class* **Late-OnsetCerebellarAtaxia**, *class* **HereditarySpasticParaplegia**, *class* **Early-OnsetCerebellarAtaxia**, *class* **CongenitaNonprogressiveAtaxia**, dan ***class*** **CerebellarAtaxiadengan PerbaikanDNATidakSempurna** masing-masing adalah *subclass* dari *class* **HereditaryAtaxia**. Sedangkan *class* **HereditaryAtaxia** dan *class* **Huntington** sendiri adalah *subclass* dari *class* **PenyakitSistemSaraf**.

4. Taksonomi Penyakit Sistem Sirkulasi

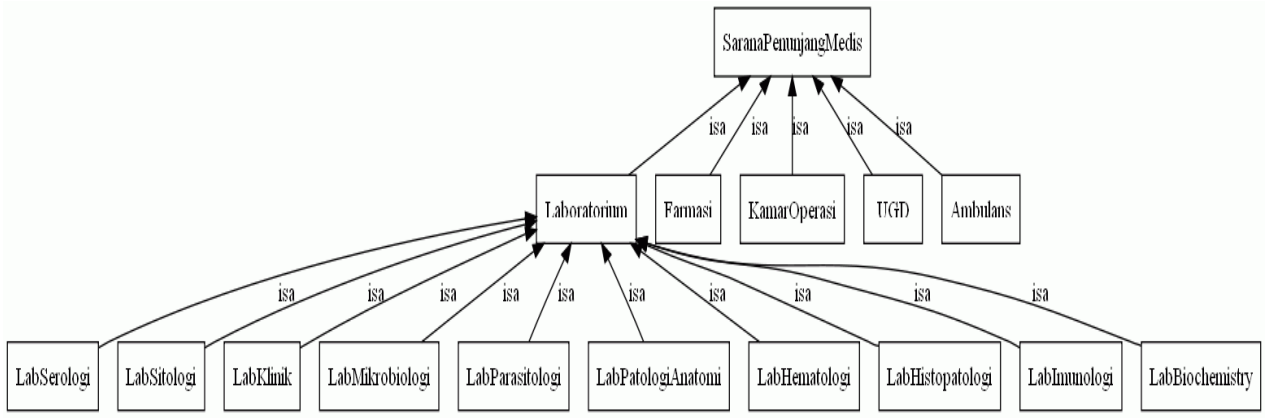


Gambar 22. Taksonomi Penyakit Sistem Sirkulasi

Gambar 22 di atas adalah taksonomi yang dilakukan pada *class* **PenyakitSistemSirkulasi**. Taksonomi **PenyakitSistemSirkulasi** terbagi menjadi 2 *class* besar yakni *class* **PenyakitJantungParu-ParudanSirkulasi** dan *class* **Hipertensi**. *Class* **PenyakitJantungParu-ParudanSirkulasi** menjadi super dari *class* **EmbolismeParu** dan *class* **PenyakitJantungParu-ParuLain**. *Class* **EmbolismeParuDisertaiCorpulmonaleAkut** dan *class* **EmbolismeParuTanpa DisertaiCorpulmonaleAkut** merupakan sub *class* dari *class* **Embolisme**, sedangkan *class* **JantungKifoskoliotik**, *class* **HipertensiParu-ParuPrimer**, *class* **HipertensiParu-ParuSekunderLainnya**, *class* **PenyakitJantungParu-ParuLainnya** dan *class* **JantungParu-ParuUnspecified** merupakan *subclass* dari *class* **PenyakitJantungParu-ParuLain.**

Untuk taksonomi *class* **Hipertensi** terdiri atas *subclass* **JantungHipertensi**, **GinjalHipertensi**, **JantungdanGinjal Hipertensi** dan *class* **Hipertensi Esensial**. *Class* **JantungdanGinjalHipertensiDisertaiGagalGinjal**, *class* **JantungdanGinjalHipertensiDisertaiGagalJantung**, *class* **JantungdanGinjal HipertensiDisertaiGagalJantungdanGagalGinjal** dan *class* **Jantungdan GinjalHipertensiUnspecified** menjadi *subclass* dari *class* **JantungdanGinjal Hipertensi**. *Class* **JantungHipertensiDisertaiGagalJantung** dan *class* **Jantung HipertensiTanpaDisertaiGagalJantung** merupakan *subclass* dari *class* **Jantung Hipertensi**. Sedangkan *class* **HipertensiSekunder**, *class* **GinjalHipertensi DisertaiGagalGinjal** dan *class* **GinjalHipertensiTanpaDisertaiGagalGinjal** adalah *subclass* dari *class* **GinjalHipertensi**. Dan *class* **Hipertensi Renovaskuler**, *class* **HipertensiSekunderUntukKerusakanKelenjar**, *class* **HipertensiSekunderLainnya**, *class* **HipertensiSekunderUnspecified dan** *class* **HipertensiSekunderUntukKerusakanGinjalLain** adalah *subclass* dari *class* **HipertensiSekunder.**

5. Taksonomi Sarana Penunjang Medis

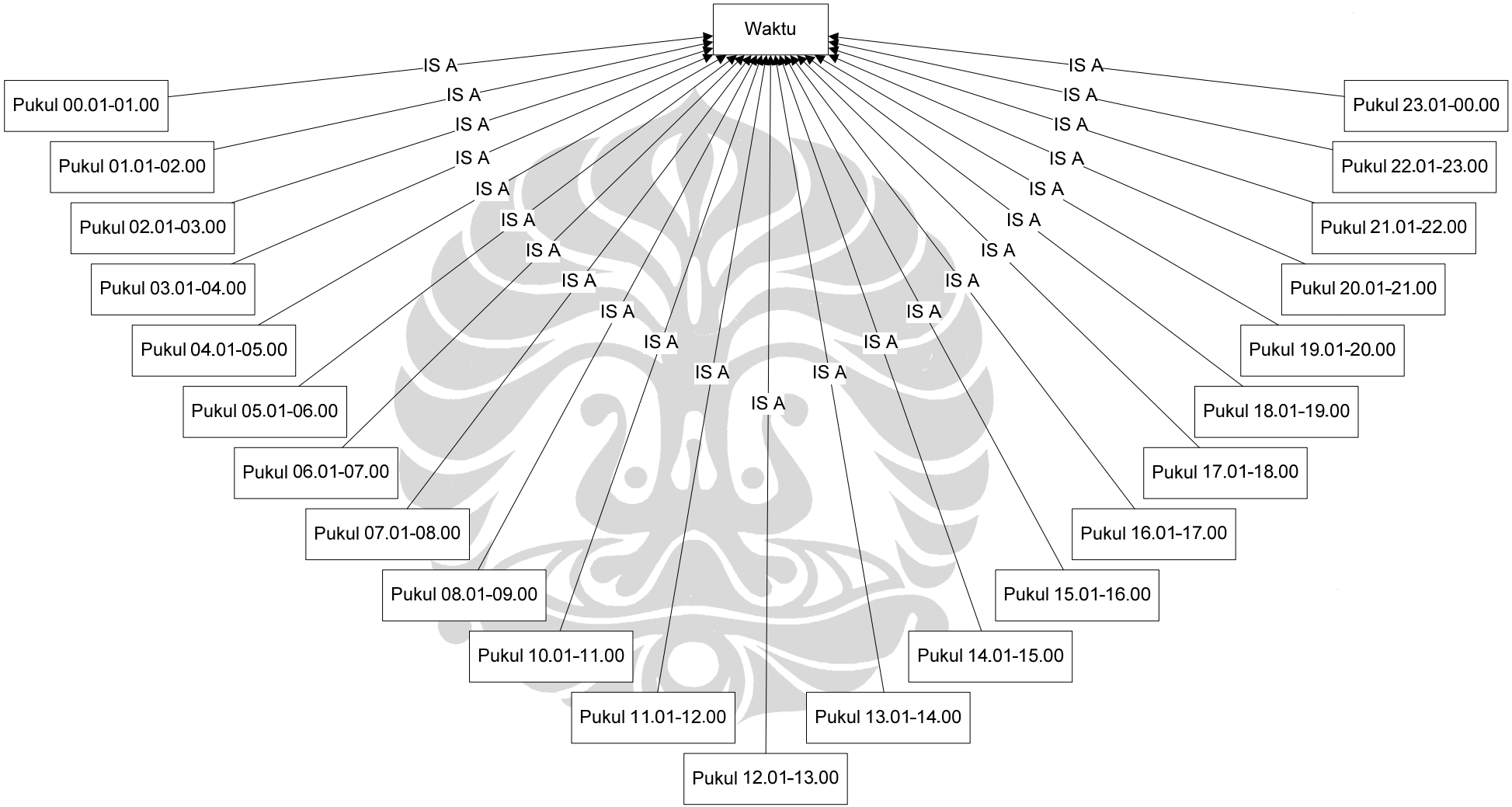


Gambar 23. Taksonomi Sarana Penunjang Medis

Gambar 23 di atas adalah taksonomi yang dilakukan pada *class* **SaranaPenunjangMedis**. *Class* **SaranaPenunjangMedis** menjadi super *class* dari *class* **Laboratorium**, *class* **Farmasi**, *class* **Kamar Operasi**, *class* **UGD** dan *class* **Ambulance**.

Class **Laboratorium** sendiri terdiri atas beberapa *subclass* yakni, *class* **LabSerologi**, *class* **LabParasitologi**, *class* **LabHistopatologi**, *class* **LabPatologiAnatomi**, *class* **LabImunologi**, *class* **LabKlinik**, *class* **LabBiochemistry**, *class* **LabSitologi**, *class* **LabMikrobiologi** dan**LabHematologi.**

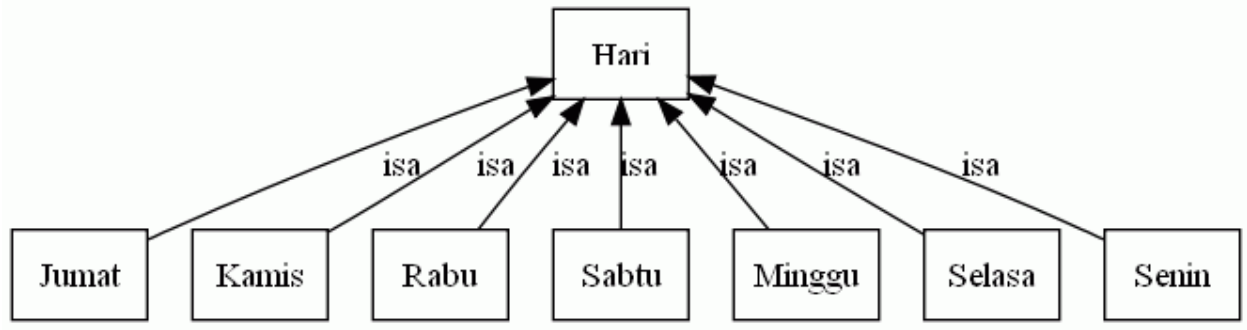
6. Taksonomi Jam



Gambar 24. Taksonomi Jam

Gambar 24 di atas adalah taksonomi yang dilakukan pada *class* **Jam**. Taksonomi yang dilakukan pada *class* **Jam** dilakukan dengan membuat *subclass* **Pukul** dalam satu hari yakni dari pukul 00.00 sampai dengan pukul 24.00 dengan rentang waktu masing-masing 1 jam yang menyebabkan *class* **Jam** memiliki 24 *subclass*

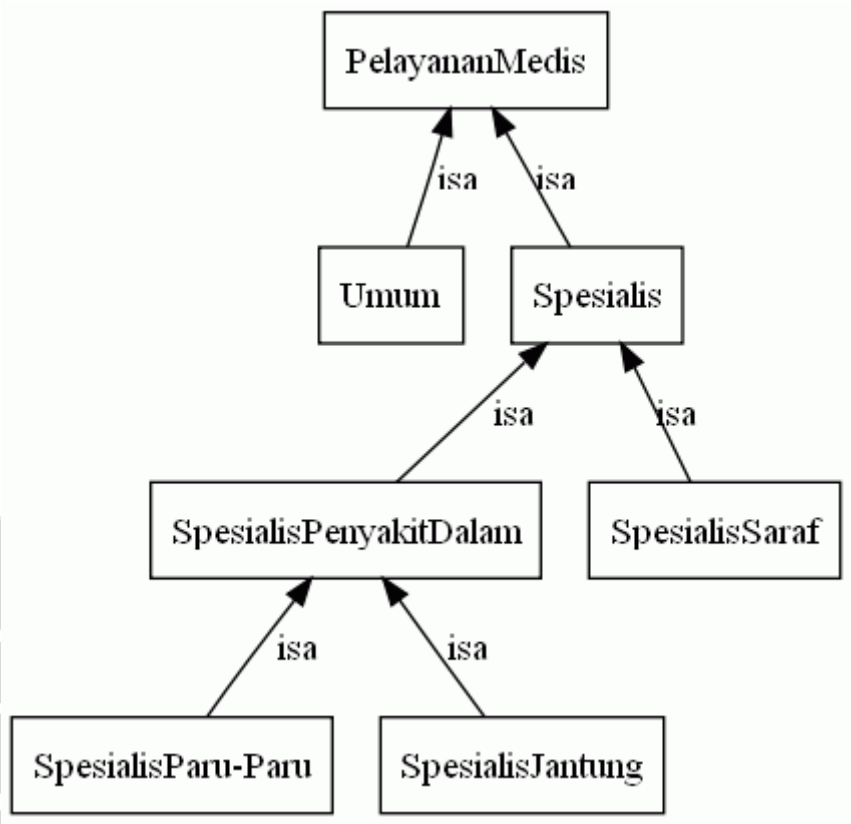
7. Taksonomi Hari



Gambar 25. Taksonomi Hari

Gambar 25 di atas adalah taksonomi yang dilakukan pada *class* **Hari**. *Class* **Hari** memiliki 7 *subclass* yang merupakan nama-nama hari dalam satu minggu yakni *class* **Senin**, *class* **Selasa**, *class* **Rabu**, *class* **Kamis**, *class* **Jumat**, *class* **Sabtu**, dan *class* **Minggu**.

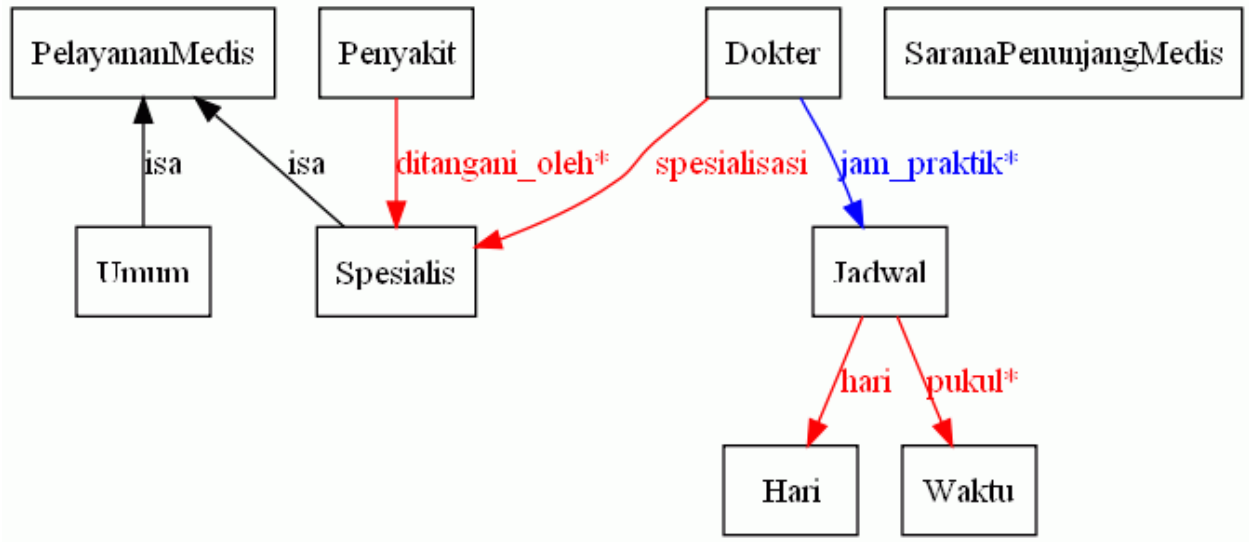
8. Taksonomi Pelayanan Medis



Gambar 26. Taksonomi Pelayanan Medis

Gambar 26 di atas adalah taksonomi yang dilakukan pada *class* **Pelayanan Medis**. *Class* **PelayananMedis** memiliki 2 *subclass* yaitu *class* **Spesialis** dan *class* **Umum**. *Class* **Umum** tidak memiliki *subclass* apapun sedangkan *class* **Spesialis** memiliki 2 *subclass* yakni *class* **SpesialisPenyakitDalam** dan *class* **SpesialisSaraf**. Dan *class* **SpesialisPenyakitDalam** dibagi lagi menjadi *subclass* **SpesialiParu-Paru** dan *class* **SpesialisJantung**.

9. Hubungan Antar *Class*



Gambar 27. Hubungan Antar *Class*

Di antara *class-class* pada top level seperti yang diperlihatkan pada **Gambar 27** di atas, ada beberapa hubungan yang terjadi antara lain:

- *Class* **Dokter** memiliki atribut jam_praktik yang nilainya merupakan *instance* kelas **Jadwal**, dan atribut spesialisi yang nilainya merupakan kelas yang menjadi *subclass* **Spesialis**.
- *Class* **Penyakit** memiliki atribut ditangani_oleh dan nilainya merupakan kelas yang menjadi *subclass* **Spesialis.**

4.2.3 Slot *of Class* dan *Instance*

Pada ontologi ini, tidak semua *class* memiliki slot. Berikut *class-class* yang memiliki slot beserta contoh individu (atau *instance*) class yang bersangkutan:

a. Dokter

Class ini memiliki slot-slot yang menjelaskan identitas dokter seperti yang diperlihatkan pada **Tabel 8** sebagai berikut:

Tabel 8. Slot of *Class* Dokter

Nama slot	Type value	Cardinalitas	Keterangan
Nama	String	Single	Nama dari dokter

Spesialisasi	*Instance* of *Class* Spesialisasi	Multiple	Spesialis dari dokter
Jam Praktik	*Instance* of *Class* Jadwal	Multiple	Jampraktik dari dokter

Contoh *instance class* Dokter adalah sebagai berikut:

Nama: dr. Sari, Sp. PD

Spesialisasi: PenyakitDalam

Jam Praktik: Sp_21Mei_*instance*_2 (atau Senin, pukul 09.01 sampai 12.00), Sp_21Mei_*Instance*_1006 (atau Rabu, pukul 14.01 sampai 17.00), dan Sp_21Mei_*Instance*_3 (atau Senin, pukul 13.01-16.00)

_30Juni_Instance_22	
jam_praktik =	Sp_21Mei_Instance_2
	Sp_21Mei_Instance_10006
	Sp_21Mei_Instance_3
nama =	dr. Sari, Sp PD
spesialisasi =	SpesialisPenyakitDalam

Gambar 28. *Instance Class* Dokter

b. Jadwal

Class ini memiliki slot yang menjelaskan waktu jadwal dilakukan, seperti yang diperlihatkan pada **Tabel 9** berikut ini.

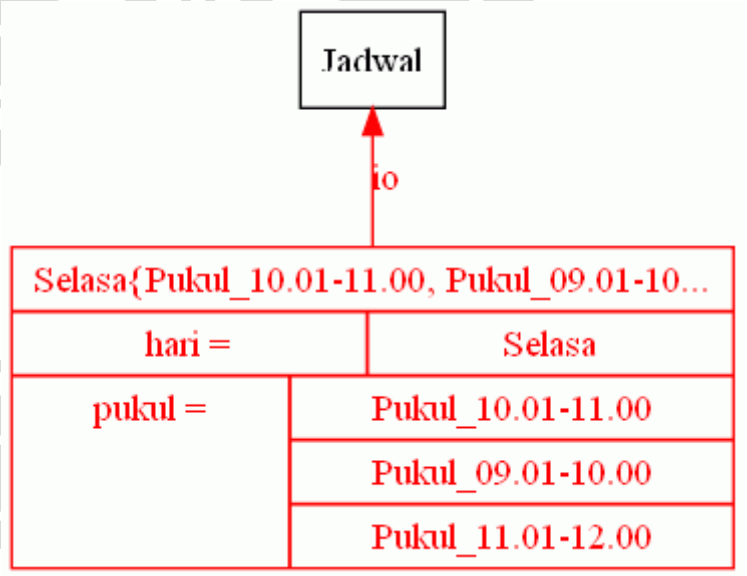
Tabel 9. Slot of *Class* Jadwal

Nama slot	*Type value*	Kardinalitas	Keterangan
Hari	*Instance* of *Class* Hari	*Multiple*	Hari pada jadwal (rentang waktu Senin-Minggu)
Pukul	*Instance* of *Class* Waktu	*Multiple*	Jam pada jadwal (rentang waktu 00.00-24.00, tiap jam)

Contoh *instance class* Jadwal adalah sebagai berikut:

Hari: Selasa

Pukul: Pukul 10.01-11.00, Pukul 09.01-10.00, Pukul 11.01-12.00



Gambar 29. *Instance Class* Jadwal

c. Penyakit

Class Penyakit memiliki slot seperti yang diperlihatkan pada **Tabel 10** sebagai berikut yaitu.

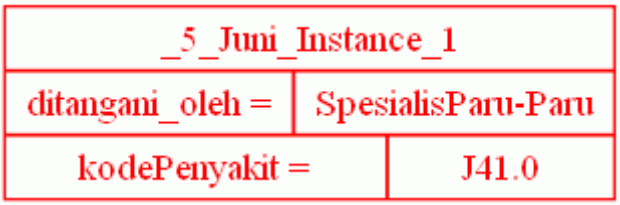
Tabel 10. Slot of *Class* Penyakit

Nama slot	Type value	Cardinalitas	Keterangan
Ditangani oleh	*Class* with super*class* Spesialis	Multiple	Apa saja spesialis yang menangani penyakit yang bersangkutan
Kode Penyakit	String	Multiple	ICD dari penyakit

Slot **Ditangani oleh** merupakan *inverse slot* menangani yang berada di *class* Spesialis. Berikut contoh *instance class* Penyakit :

Kode penyakit : J41.0

Ditangani oleh : Spesialis paru-paru



Gambar 30. *Instance Class* Penyakit

RDFS yang dibuat untuk ontologi dan contoh implementasi RDF rumah sakit dapat di lihat di **Lampiran D**.

BAB 5
IMPLEMENTASI

Bab ini menjelaskan mengenai proses implementasi dan pengujian terhadap sistem TRuST yang dikembangkan oleh penulis. Pembahasan bab ini akan dimulai dengan memaparkan solusi teknis yang diterapkan tim pengembang dalam membangun sistem TRuST yang meliputi meliputi hal-hal yang berkaitan dengan *development environment* dan *supporting tools* yang digunakan. Setelah solusi teknis, pembahasan berlanjut pada proses implementasi dan pengujian yang dilakukan terhadap sistem serta kendala yang dihadapi saat proses pengembangan.

5.1 Solusi Teknis

Pada subbab ini akan dijelaskan solusi teknis yang digunakan dan diimplementasikan pada sistem. Solusi teknis tersebut diantaranya adalah *development environment* dan *supporting tools*. *Development environment* adalah perangkat-perangkat lunak yang digunakan untuk menunjang berjalannya sistem. Sedangkan *supporting tools* adalah perangkat-perangkat lunak yang digunakan untuk menunjang proses analisis, perancangan, dan implementasi sistem. **Tabel 11** berikut merupakan rangkuman komponen-komponen yang kami gunakan sebagai *development environment.*

Tabel 11. *Development Environment*

Development Environment	Penjelasan
Protégé	*Protégé* adalah sebuah *software* yang dapat digunakan untuk pengembangan sistem berikut *Knowledge Base System*. Aplikasi yang dikembangkan oleh *protégé* digunakan dalam pemecahan masalah dan pembuatan keputusan dalam sebuah domain. *Protégé* dikembangkan oleh sebuah organisasi yang bernaung di bawah Stanford University, yang mengambil spesialisasi di bidang ontologi. *Dengan Protégé* dapat dibuat sebuah

Development Environment	Penjelasan
	domain ontologi. Selain itu pengguna juga dapat menyesuaikan *form* untuk *entry* data, dan memasukkan data ke dalam *form* tersebut. Berbagai format penyimpanannya yang disediakan adalah dalam bentuk OWL, RDF, XML dan HTML. *Protégé* menyediakan kemudahan *plug and play* yang membuatnya fleksibel untuk pengembangan *prototype*. [PRO08]
IIS 7	IIS atau *Internet Information Services* adalah sekumpulan dari Internet-*based service* untuk server yang menggunakan Microsoft Windows. IIS 7 adalah versi yang dikeluarkan bersamaan dan bersesuaian dengan Windows Vista dan Windows Server 2008. Salah satu *feature* yang dimilikinya adalah *modular architecture*. Walaupun memiliki *monolitic* server yang meng-*handle* semua *service*, IIS 7 memiliki sebuah *core web server engine*. *Modules* dengan *feature* tertentu dapat ditambahkan dalam *engine* untuk membuatnya berfungsi. Keuntungan yang dimiliki dengan adanya arsitektur ini adalah, hanya *feature* yang dibutuhkan yang dapat diakses dan fungsionalitasnya dapat di perpanjang dengan menggunakan *modules* yang di-*custom* [WIK08]
Visual Studio 2008	Microsoft® Visual Studio® 2008 adalah product yang dikeluarkan oleh Microsoft sesuai dengan misinya sebagai *smart client applications* dengan memperbolehkan pengembang untuk secara cepat membuat koneksi pada aplikasi dengan kualitas yang baik dan membuat *user* kaya akan *experiences*.

Development Environment	Penjelasan
	Dengan mengunakan Visual Studio 2008, akan lebih mudah untuk menangkap dan menganalisa informasi yang nantinya dibutuhkan untuk membuat *business decisions* yang efektif. Visual Studio 2008 juga dipilih karena lebih sesuai untuk mengembangkan aplikasi yang lebih *secure*, *manageable*, dan *reliable* jika menggunakan Windows Vista, 2007 Office *system* dan .NET 3.5 [MSD08]
SQL Server Compact	SQL Server Compact digunakan sebagai media penyimpan *database* yang di-*embedded* pada setiap *single-user client application* untuk semua aplikasi yang memiliki platform Windows Mobile, termasuk Tablet PCs, Pocket PCs, *smart phones*, dan *desktops*.
.NET 3.5 dan .NET compact framework 3.5	.NET Framework adalah *framework* yang ditawarkan oleh Microsoft dan biasanya digunakan oleh sebagian besar aplikasi baru yang dibuat dengan platform Windows.
SQL Server	Microsoft SQL Server adalah sebuah *relational database management system* (RDBMS) yang dikeluarkan oleh Microsoft. Penggunaan Microsoft SQL Server sebagai RDBMS pada server rumah sakit dan server pusat adalah karena permintaan dari *client* untuk menggunakan produk-produk yang dikeluarkan oleh Microsoft.
Avangardo	Avangardo adalah sebuah emulator GPS yang digunakan untuk men-*generate* NMEA *message.* *Software* avangardo dapat diperoleh secara cuma-cuma, dan dirasakan dapat membantu dalam men-*generate* lokasi *longitude* dan *latitude* tertentu menjadi

Development Environment	Penjelasan
	alasan bagi tim pengembang untuk menggunakan *software* ini.
SemWeb	SemWeb adalah *library* yang digunakan untuk membangun aplikasi *semantic web*. Dalam proyek mahasiswa ini, SemWeb dimanfaatkan untuk kepentingan membaca dan menulis dalam format RDF/n3 serta kemampuan untuk *reasoning*.

Berikut ini adalah sekilas penjelasan tentang GPS Emulator yang penulis gunakan. Avangardo GPS Generator adalah sebuah *software* GPS emulator yang dapat men-*generate* NMEA *message* dan dapat digunakan pada data input yang berbeda. Dengan adanya *software* ini setiap orang dapat men-*generate* kalimat NMEA yang kompleks dari berbagai data *input*. Kegunaan dari Avangardo GPS Generator pada pengembangan TRuST adalah untuk membantu men-*generate* GPS data dari lokasi pasien yang akan dibaca oleh aplikasi *mobile* TRuST untuk mengetahui posisi pasien. Lebih tepatnya, data GPS posisi pasien adalah sebuah RMC NMEA *message*.

Software ini cocok sekali digunakan ketika kita membutuhkan GPS data *input*, tetapi tidak memiliki *signal* atau GPS *receiver*. *Software* ini dapat digunakan sebagai GPS simulator menggunakan *null* modem (COM port – rs232) *cable* atau sebuah *virtual serial port software* [AVA08].

Versi dari GPS generator yang ada saat ini, dapat dioperasikan dengan dua *mode*, yaitu:

1. *Map mode*

   Dengan menggunakan *mode* ini, langkah yang dilakukan adalah kita dipersilahkan untuk memilih sebuah *file* peta, tandai koordinat (*geodetic latitude/longitude*) awal dan akhir yang dikehendaki pada peta dan tentukan kecepatannya. GPS generator kemudian akan melakukan penghitungan *heading* dan *current coordinates* dan nantinya akan di-generate atau dihasilkan sebuah data output GPS. *Current position* dan

heading yang dipilih sebagai simulasi akan ditampilkan pada peta. Selama proses simulasi berjalan *heading* dan *speed*-nya masih dapat kita ubah.

2. *File Mode*

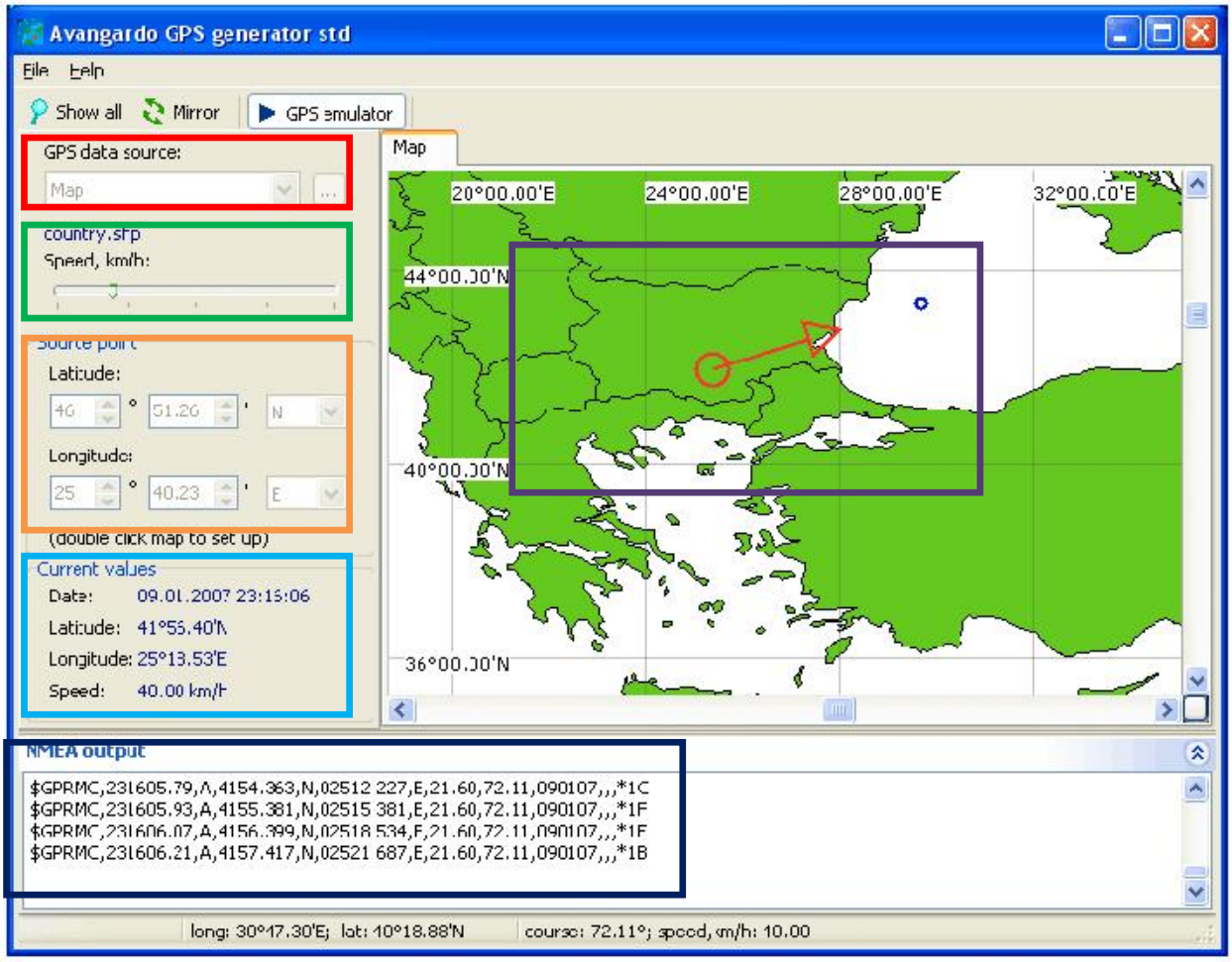
   Dengan menggunakan *mode* ini kita dapat memilih *file* NMEA yang berada di *log* dan *start position* yang ada di *file*. GPS generator akan membaca kalimat NMEA dan menunjukan posisi saat ini pada *map*. Selama proses simulasi kita dapat mengubah posisi saat ini yang dibaca pada *file* (*move forward* dan *backward*). Sehingga dengan *mode* seperti ini kita dapat mengulang GPS log data yang telah kita buat sebelumnya (termasuk GPS log, yang ditulis di *map mode*)

Kalimat NMEA 0183 yang telah dihasilkan dapat di simpan menjadi sebuah file atau mengirimkan kalimat tersebut kepada *software* atau sistem lain melalui COM *port*.

GPS simulator yang digunakan pada TRuST adalah *software* yang bersifat '*free*' yang memiliki keterbatasan sebagai berikut:

- 15 *minutes session limit*
- *splash screen during starting*
- hanya tersedia yang map mode
- hanya men-*generate* RMC NMEA *message*

Berikut ini merupakan *screenshoot* dan penjelasan tentang *software* Avangardo masing-masing.



Gambar 31. Avangardo Generator

Tabel 12. Penjelasan Avangardo

Simbol	Keterangan
	Bagian ini menunjukkan output nmea *message* yang di-*generate* oleh Avangardo.
	Bagian yang menunjukan posisi dan kecepatan angin saat ini. Nilai pada bagian ini selalu berubah-ubah sesuai dengan output NMEA saat itu.
	Bagian yang menunjukan kecepatan angin. Nilainya dapat diubah sesuai dengan keinginan kita dengan menggeser slider yang ada.

Simbol	Keterangan
	Bagian yang menunjukan *mode view* dari Avangardo.
	Bagian yang menunjukan posisi *latitude* dan *longitude*. Nilainya dapat diubah sesuai dengan keinginan dengan menentukan nilai pada box yang tersedia.
	Bagian yang menunjukan posisi lokasi yang diinginkan pada peta.

Selain development tools, penulis juga menggunakan beberapa supporting tool yang digunakan dalam membantu penulis dalam proses pemodelan dan penggambaran sistem TRuST. *Supporting tool* merupakan *tool* pembantu yang digunakan dalam pengembangan sistem. Beberapa *supporting tools* yang digunakan oleh penulis dapat dilihat pada **Tabel 13** di bawah ini.

Tabel 13. *Supporting Tools*

Supporting Tools	Penjelasan
Microsoft Visio 2003	Microsoft Visio 2003 digunakan sebagai media dalam pembuatan diagram-diagram yang dibuat dalam proses pengembangan diantaranya arsitektur 85*system*, *use case* diagram, *business process* diagram, dan *activity* diagram.
DBDesigner	DBDesigner adalah *supporting tool* yang digunakan dalam membuat skema *database*.

5.2 Prototipe Awal

Pada prototipe pertama, kami menggunakan *web service* dengan teknologi .net 2.0 sebagai implemetasi dari server pusat, dan server rumah sakit. Namun, kami mulai mengeksplorasi teknologi Microsoft yang baru yaitu *windows communication*

foundation (WCF) yang menggunakan .net 3.5, dan mulai menggunakannya pada prototipe kedua. WCF sendiri, tidak jauh berbeda konsepnya dengan *web service*. Keduanya merupakan *service oriented architecture* (SOA) dalam bentuk web dengan *service-service* yang dapat diakses dan digunakan oleh *client*. Penggunaan WCF tersebut awalnya dikarenakan permintaan *client*, dan sesuai dengan tujuan dari *student project* ini untuk mengeksplorasi teknologi terbaru dari Microsoft.

Walaupun menggunakan *business logic* yang sama seperti prototipe sebelumnya, kami merasakan beberapa kelebihan yang diberikan oleh WCF pada prototipe kedua ini. Salah satu kelebihan yang sangat dirasakan bagi developer dari WCF adalah kemudahan dalam *serialization object* yang akan dikembalikan dari WCF. *Web service* sangat bergantung pada class XmlSerializer untuk merepresentasikan data atau *object* yang akan dikembalikan dari *web service* dalam bentuk XML. Sehingga jika *web service* akan mengembalikan *complex data type*, maka pengembang harus membuat sendiri *class* untuk mendefinisikan *complex data type* tersebut menggunakan class XmlSerializer. Sedangkan WCF menawarkan tag [DataContractAttribute] dan [DataMemberAttribute] yang dapat diberikan pada *data type* ataupun *complex data type* yang akan dikembalikan WCF. WCF akan mengenali tag tersebut dan melakukan *serialization* terhadapat data yang diberikan tag tersebut. Hal ini sangat memudahkan pengembang, karena pekerjaan untuk merepresentasikan data atau *object* yang akan dikembalikan oleh WCF dikerjakan secara otomatis oleh WCF tersebut.

Kelebihan lain yang kami rasakan dari penggunaan WCF adalah kecepatan dalam mendapatkan hasil dari *service*-nya. Kami memang tidak meneliti hal ini lebih jauh, namun kami mendapatkan beberapa riset perbandingan yang menyatakan bahwa WCF lebih cepat dibandingkan *web service*. Diantara riset tersebut, terdapat percobaan berikut [GEE08].

Tabel 14. Hasil Perbandingan WCF dengan *Web Service* (asmx) [GEE08]

Percobaan	Hasil Percobaan
dimulai dengan 10 *user load* sampai maximum 250 user, user bertambah 50	WCF meng-*handle* 51.2% lebih banyak *test case* dibanding asmx,

Percobaan	Hasil Percobaan
setiap 10 detik	dengan 4.38 kali lebih cepat
dimulai dengan 10 user load sampai maximum 1000 user, user bertambah 50 setiap 10 detik	WCF meng-*handle* 56.79% lebih banyak *test case* dibanding asmx, dengan 18.82 kali lebih cepat
constant load 500 user	WCF meng-*handle* 124.11% lebih banyak *test case* per detik dibanding asmx

5.3 Implementasi

Pada subbab ini akan dijelaskan tentang proses dan cara kerja dari TRuST yang diimplementasikan untuk setiap *use case* serta *class* diagram yang di-*generat*e oleh Visual Studio berdasarkan implementasi yang telah dibuat.

5.3.1 Get RS Terdekat

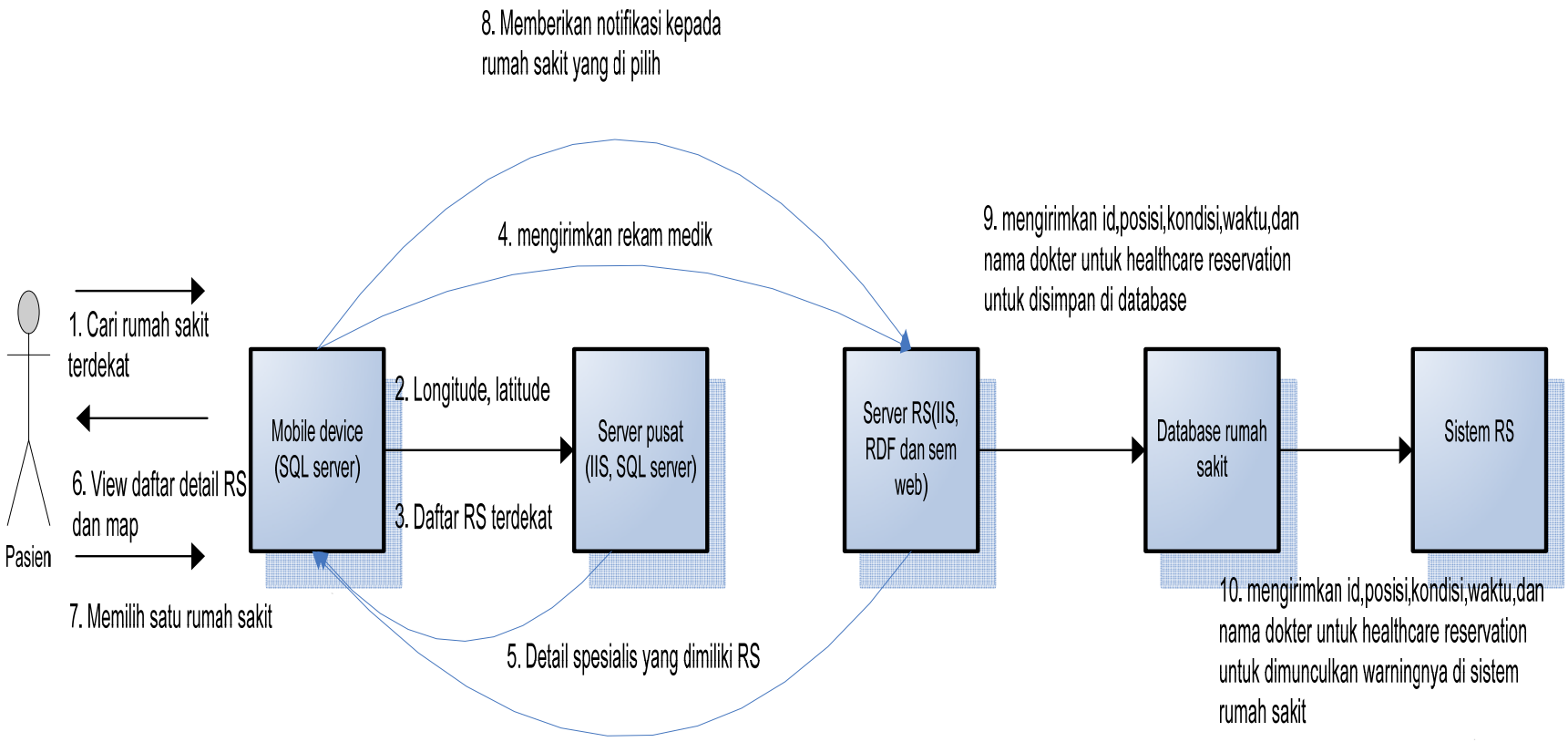
Gambar 32 adalah *flow* implementasi Get RS Terdekat yang dibuat.

Dalam pengembangan TRuST digunakan Avangardo, sebuah emulator GPS untuk men-generate posisi *user*. Avangardo akan secara terus menerus membaca posisi *user*.

Kemudian *message* RMC yang dihasilkan oleh avangardo akan disimpan dalam bentuk text, dan dimasukkan pada *device emulator* agar dapat dibaca oleh aplikasi mobile TRuST untuk mengetahui posisi pasien. *Message* yang diambil adalah *message* yang paling akhir sesaat setelah signal healthcare diaktifkan. *Message* ini kemudian di-*parsing* untuk diambil bagian *latitude* dan *longitude*nya saja. Bagian ini kemudian dikirimkan ke server pusat.

Di server pusat sudah tersimpan beberapa posisi *latitude* dan *longitude* rumah sakit yang didapatkan dengan menggunakan Google Earth.

Lokasi yang didapatkan dari Google Earth masih dalam format posisi koordinat dalam bentuk *decimal degrees and degrees, minutes, and seconds*. Data ini harus diubah terlebih dahulu menjadi bentuk *decimal degrees.*



Gambar 32. *Flow* Implementasi Get RS Terdekat

Persamaan yang dipakai untuk mengubah format koordinat dari satuan derajat/menit ke satuan derajat adalah sebagai berikut:

```
Derajat = derajat + (menit/60)
```

Sedangkan untuk mengubah satuan derajat/menit/detik ke satuan derajat digunakan persamaan

```
Derajat = derajat + (menit/60) + (detik/3600)
```

Contoh: data yang berupa -96 *degrees*, 47 *minutes*, dan 28.0392 *seconds* diubah menjadi

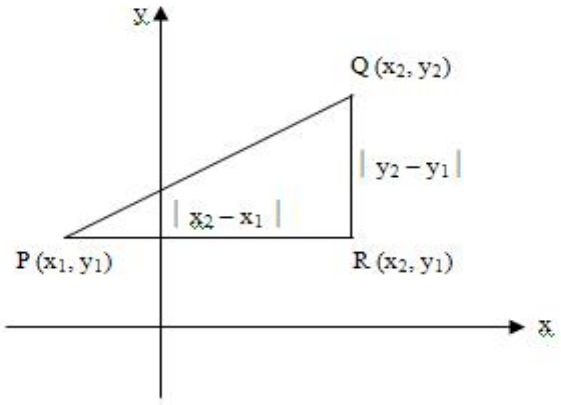
Decimal degrees = -(96 + (47 /60) + (28.0392/3600)) = -96.791122

Penanda bagian timur, barat, utara dan selatan di tandai dengan tanda bilangan positif atau negatif. Tanda negatif menunjukan arah timur dan selatan, sedangkan sisanya adalah kebalikannya.

Kemudian dilakukanlah perhitungan jarak antara posisi *user* dengan semua rumah sakit. Perhitungan jarak ini dilakukan dengan menggunakan rumus perhitungan jarak antara dua titik yang diadaptasi dari hukum phytaghoras, seperti penjelasan di bawah.

Theorema Phytagoras

$a^2 + b^2 = c^2$



Gambar 33. Teorema Phytagoras

Jarak dari titik P ke titik Q,
dengan menggunakan teorema phytagoras:

$$|PQ|^2 = |PR|^2 + |RQ|^2$$
$$= |x_2 - x_1|^2 + |y_2 - y_1|^2$$
$$d(PQ) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Misalkan Titik P adalah posisi *longitude* dan *longitude* pasien dan Q adalah posisi *latitude* dan *longitude* rumah sakit. Maka jarak antara dua titik tersebut dapat dicari dengan mencari d(PQ) yang bernilai angka dan memiliki satuan derajat. Setiap satu derajat di equator bernilai 69,2 miles, dan satu miles bernilai sekitar 1.6093 KM [AKA08].

Walaupun sebenarnya untuk menentukan jarak di permukaan bumi diperhitungkan juga tentang bentuk bumi yang tidak sepenuhnya bulat. Namun, karena jarak yang dicari cukup dekat, maka hal itu bisa diabaikan.

Untuk mendapatkan rumah sakit yang masuk ke dalam radius jarak tertentu diimplementasikan dengan *store procedure* sebagai berikut.

```
USE [location]
```

```
GO

/****** Object:  StoredProcedure [dbo].[CalculateDistance]    Script Date: 07/28/2008 14:08:02 ******/

SET ANSI_NULLS ON

GO

SET QUOTED_IDENTIFIER ON

GO

-- =============================================

-- Author:                rap

-- Create date:

-- Description:

-- =============================================

ALTER PROCEDURE [dbo].[CalculateDistance]

        -- Add the parameters for the stored procedure here

        @p1 decimal(18,5) = 0,

        @p2 decimal(18,5) = 0,

  @p3 integer = 0

AS

BEGIN

        -- SET NOCOUNT ON added to prevent extra result sets from

        -- interfering with SELECT statements.

        SET NOCOUNT ON;


  -- Insert statements for procedure here

        SELECT

  City.nama,City.alamat, City.telp,

  SQRT                ((((@p1-City.Latitude)*(@p1-City.Latitude))+((@p2-City.Longitude)*(@p2-City.Longitude)))*69.2)*1.6093
```

```
   AS kilometer, City.url, City.Latitude, City.Longitude

   FROM

   dbo.rumah_sakit City

         WHERE

         City.latitude is not null and

         City.longitude is not null and

   SQRT          ((((@p1-City.Latitude)*(@p1-City.Latitude))+((@p2-City.Longitude)*(@p2-
City.Longitude)))*69.2)*1.6093 <(@p3)

         and City.url is not null

   GROUP BY City.nama, City.latitude, City.longitude, City.alamat, City.telp, City.url

   ORDER BY kilometer

END
```

Daftar rumah sakit yang didapatkan kemudian dikirimkan kembali ke *mobile device*. Data yang dikirimkan adalah:

- Nama rumah sakit
- Alamat rumah sakit
- No telepon rumah sakit
- URL WCF *service* rumah sakit
- *Latitude* posisi rumah sakit
- *Longitude* posisi rumah sakit
- Serta jarak antara rumah sakit dengan pasien

Dari daftar yang didapatkan, *mobile device* menghubungi semua WCF *service* rumah sakit sekaligus mengirimkan kode icd penyakit yang diderita oleh pasien yang tersimpan di *mobile device*. Default kode penyakit yang dikirim adalah kode penyakit yang paling baru dia derita, namun pasien juga dapat memilih mencari

rumah sakit terdekat yang memiliki dokter spesialis tertentu sesuai dengan penyakit yang pernah dia derita. Hasil implementasi untuk pemilihan rumah sakit terdekat berdasarkan spesialis penyakit tertentu dapat dilihat pada **Gambar 34**.

Gambar 34. *Interface* Pemilihan Rumah Sakit Berdasarkan Penyakit

Setiap WCF *service* rumah sakit akan mengembalikan daftar spesialis yang dimiliki oleh rumah sakit tersebut yang bersesuian dengan penyakit yang diderita pasien. Untuk mendapatkan daftar spesialis itu, WCF rumah sakit melakukan *reasoning* terhadap *context* penyakit *user* dengan menggunakan *semantic web* yang akan dijelaskan berikut ini.

Mekanisme dalam membaca RDFS untuk melakukan *reasoning* pada *semantic web* terdiri atas dua bagian, yakni dengan menggunakan *reasoning* rdfs dan

reasoning euler backward chaining. Berikut ini penjelasan dari masing-masing mekanisme.

Reasoning RDFS digunakan untuk mengambil informasi mengenai nama-nama dokter yang tersedia dan mengetahui penyakit yang diderita pasien berdasarkan kode penyakit yang diberikan. Sintaks *reasoning* tersebut diperlihatkan pada bagian di bawah ini.

```
Public string namaDari(Entity dok)
        {
            Resource[]    nama2    =    storeRDFS.SelectObjects(dok,
bernama);
            Literal nama = (Literal)nama2[0];
            return (string)nama.ParseValue();
        }
        public Entity getPenyakit(string kodePenyakit)
        {
            Entity[] penyakit =
toreRDFS.SelectSubjects(memilikiKodePenyakit,  new
Literal(kodePenyakit));
            return penyakit[0];
        }
```

Sedangkan mekanisme *euler backward chaining* digunakan untuk menentukan jawaban dari pertanyaan apakah seorang dokter yang ada di rumah sakit tersebut dapat menangani suatu penyakit tertentu atau tidak. Faktor yang mempengaruhi jawaban atas pertanyaan tersebut adalah:

1. Ketersediaaan spesialis untuk penyakit tersebut
2. Jadwal dokter spesialis tersebut

Sedangkan *rules* yang digunakan pada mekanisme *euler* dapat dilihat pada bagian di bawah ini.

```
1. Seorang dokter dapat menangani suatu penyakit jika dokter itu
merupakan seorang spesialis yang khusus menangani penyakit
tersebut.

2. Seorang dokter bisa menangani penyakit hari ini jika memiliki
jadwal di hari ini.

3. Seorang dokter bisa menangani penyakit di jam ini jika memiliki
jadwal di jam ini.

4. Seorang dokter juga bisa menangani sebuah penyakit yang menjadi
sub class dari class spesialis penyakit yang ditanganinya.

5. Jika sebuah class spesialis memiliki sebuah sub class yang
memiliki sub class lain, maka berlaku hubungan transitif. Jika A
subclass B, dan B subclass C, maka A adalah subclass C.

6. Istilah spesialisasi dan specialization adalah sinonim
```

Sedangkan sintaks untuk *rules* di atas dapat dilihat pada bagian berikut

```
string rules =

 "@prefix : <http://www.cs.ui.ac.id/sp_ehealth#>.\n" +

 "@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.\n" + "\n"
+

 "{ ?a :jam_praktik ?b . ?b :hari ?c . } => { ?a :jam_praktik ?c }
.\n" +

 "{ ?a :jam_praktik ?b . ?b :pukul ?c . } => { ?a :jam_praktik ?c
} .\n" +

 "{ ?a :ditangani_oleh ?b . ?c :spesialisasi ?b } => { ?a
:ditangani_oleh ?c } .\n" +

 "{ ?a :ditangani_oleh ?b . ?b rdfs:subclassOf ?c . ?d
:spesialisasi ?c } => { ?a :ditangani_oleh ?d }  .\n" +

 "{ ?a rdfs:subclassOf ?b . ?b rdfs:subclassOf ?c } => { ?a
rdfs:subclassOf ?c } .\n";

"{ ?a :specialization ?b } => { ?a :spesialisasi ?b } .\n";
```

Sebelum melakukan *reasoning* euler, dilakukanlah *reasoning* RDFS yang digunakan untuk mendaftarkan nama semua dokter yang ada dan juga mengambil daftar penyakit berdasarkan kode yang diberikan. Setelah itu baru dilakukan

langkah *euler* untuk menentukan dokter yang dapat menangani penyakit tersebut. Langkah selanjutnya, *reasoning* RDFS akan mengambil semua informasi tentang dokter yang didapatkan pada langkah sebelumnya.

Informasi yang disimpan di setiap WCF rumah sakit mungkin sangat beragam. Contohnya apabila di rumah sakit Brimob untuk menyatakan spesialis menggunakan kata specialization sementara di rumah sakit Pertamina menggunakan kata spesialisasi. Perbedaan penggunaan istilah tersebut dapat dilihat pada potongan code berikut.

```
<sp_ehealth:Dokter rdf:about="&sp_ehealth;_5_Juni_Instance_13"
     sp_ehealth:nama="Dr. Malik"
     rdfs:label="_5_Juni_Instance_13">
    <sp_ehealth:jam_praktik rdf:resource="&sp_ehealth;Sp_21Mei_Instance_3"/>
    <sp_ehealth:jam_praktik rdf:resource="&sp_ehealth;Sp_21Mei_Instance_7"/>
   <sp_ehealth:specialization rdf:resource="&sp_ehealth;SpesialisSaraf"/>
</sp_ehealth:Dokter>

sp_ehealth:Dokter rdf:about="&sp_ehealth;_30Juni_Instance_21"
    sp_ehealth:nama="dr. Intan, Sp J"
    rdfs:label="30Juni_Instance_21">
   <sp_ehealth:jam_praktik rdf:resource="&sp_ehealth;Sp_21Mei_Instance_6"/>
   <sp_ehealth:jam_praktik rdf:resource="&sp_ehealth;Sp_21Mei_Instance_7"/>
  <sp_ehealth:spesialisasi rdf:resource="&sp_ehealth;SpesialisJantung"/>
/sp_ehealth:Dokter>
```

Namun menggunakan semweb dua kata ini bisa dimaknai sebagai sinonim dengan adanya rule berikut ini pada *euler reasoning*.

```
"{ ?a :specialization ?b } => { ?a :specialization ?b } .\n";
```

Setiap WCF *service* rumah sakit akan melakukan *reasoning* seperti langkah dan metode di atas. Selanjutnya informasi jawaban dari setiap rumah sakit akan dikirimkan ke *mobile device*. *Mobile device* akan mem-*filter* informasi tersebut, sehingga hanya daftar rumah sakit dengan dokter spesialis yang dibutuhkan pasien yang akan tampil pada *interface* di *mobile device*.

Gambar 35 di bawah ini adalah *interface* yang telah diimplementasikan untuk memulai menu Get RS Terdekat.

Gambar 35. *Interface* Get RS Terdekat

Untuk memulai aktivivasi menu Get RS terdekat, dapat dilakukan dengan 2 cara, yakni secara langsung meng-klik RS Terdekat atau melalui Menu. Setelah itu, mobile device akan menampilkan pilihan jarak radius rumah sakit terdekat yang ingin dicari. *Interface* yang dibuat untuk fungsi tersebut dapat dilihat di pada **Gambar 36** berikut ini.

Gambar 36. *Interface* Pilih Radius Jarak Rumah Sakit

Sedangkan **Gambar 37** di bawah ini adalah *interface* saat telah didapatkan daftar rumah sakit terdekat.

Gambar 37. *Interface* Hasil Get RS Terdekat

Jika tidak terdapat rumah sakit dalam radius yang diinginkan, maka *mobile device* akan menampilkan *message* seperti pada **Gambar 38** berikut.

Gambar 38. *Interface* Tidak Terdapat Rumah Sakit Pada Radius Yang Diinginkan

Sedangkan jika terdapat rumah sakit yang berada pada radius jarak yang diinginkan namun rumah sakit tersebut tidak memiliki dokter spesialis yang dibutuhkan, maka *mobile device* akan menampilkan *message* seperti pada **Gambar 39** berikut ini.

Gambar 39. *Interface* Tidak Terdapat Rumah Sakit Yang Memiliki Dokter Spesialis

5.3.2 *Healthcare Reservation*

Setelah daftar rumah sakit beserta detail spesialis yang dimiliki ditampilkan pada *interface mobile device*, pasien dapat memilih salah satu rumah sakit tersebut untuk melakukan *healthcare reservation*.

Gambar 40 berikut adalah *interface* yang telah di implementasikan untuk menu *healthcare reservation*.

Gambar 40. *Interface Healthcare Reservation*

Untuk mengirim *signal healthcare reservation* ini *user* dapat meng-klik menu Pesan dengan terlebih dahulu menentukan satu rumah sakit yang dipilih.

Setelah itu akan dimunculkan sebuah *display* untuk mengisi kondisi saat ini dari pasien. **Gambar 41** adalah *interface* yang diimplementasikan untuk mengisi kondisi pasien saat ini.

Gambar 41. *Interface* Memasukan Kondisi Saat Ini

Untuk melanjutkan pasien dapat memilih menu Pesan

Sehingga saat terjadinya pemesanan, *mobile device* mengirimkan data berupa:

- ID pasien
- Kondisi pasien
- Posisi pasien
- Serta mengembalikan daftar dokter spesialis yang dikirimkan oleh WCF *service*

Data ini kemudian diterima oleh WCF *service* untuk kemudian disimpan dalam *database* rumah sakit. *Database* yang sama terhubung dengan sistem rumah sakit

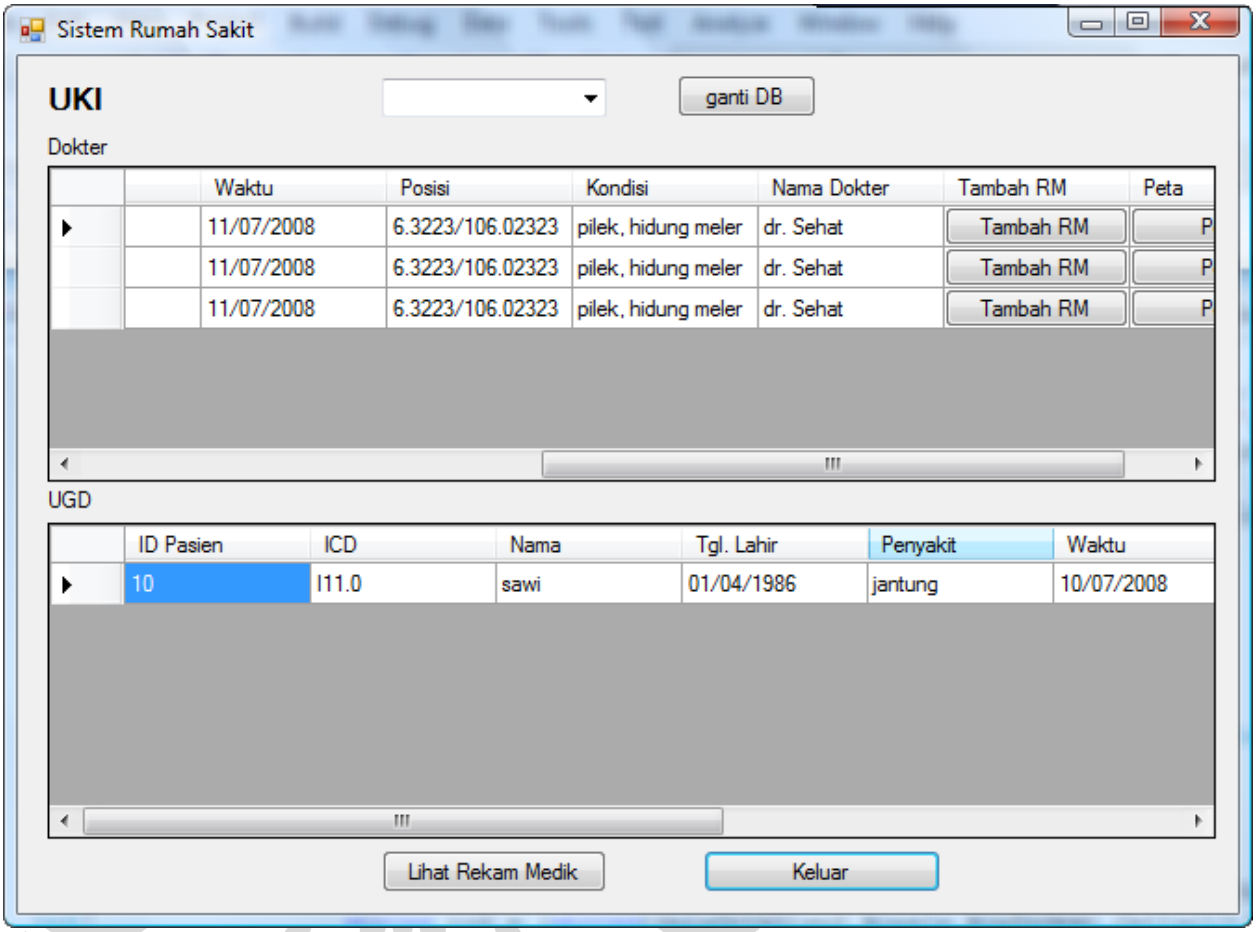
dari WCF bersangkutan, yang akan selalu melakukan pengecekan terhadap *database* tentang adanya *healthcare reservation* yang terjadi.

Jika ada *healthcare reservation* yang terjadi, maka sistem rumah sakit akan menampilkan *window* yang memberikan *warning signal*, dan meng-*update* tampilan daftar *healthcare reservation*.

Sistem rumah sakit memiliki fungsi untuk memberikan peringatan kepada pihak rumah sakit apabila ada pasien di lokasi tertentu yang memerlukan pertolongan. Untuk memenuhi kebutuhan tersebut, sistem rumah sakit selalu melakukan proses pengecekan untuk memastikan ada pasien yang meminta pertolongan atau tidak. Untuk mengimplementasikan hal tersebut, kami menggunakan *thread*. Apabila ternyata ada pasien yang meminta bantuan ke rumah sakit sistem rumah sakit akan memberikan peringatan kepada petugas rumah sakit dalam bentuk bunyi sirine yang hanya akan berhenti apabila petugas merespon terhadap permintaan tolong tersebut. *Warning* tersebut menginformasikan bahwa pasien segera datang ke rumah sakit untuk konsultasi dengan spesialis dokter yang ada.

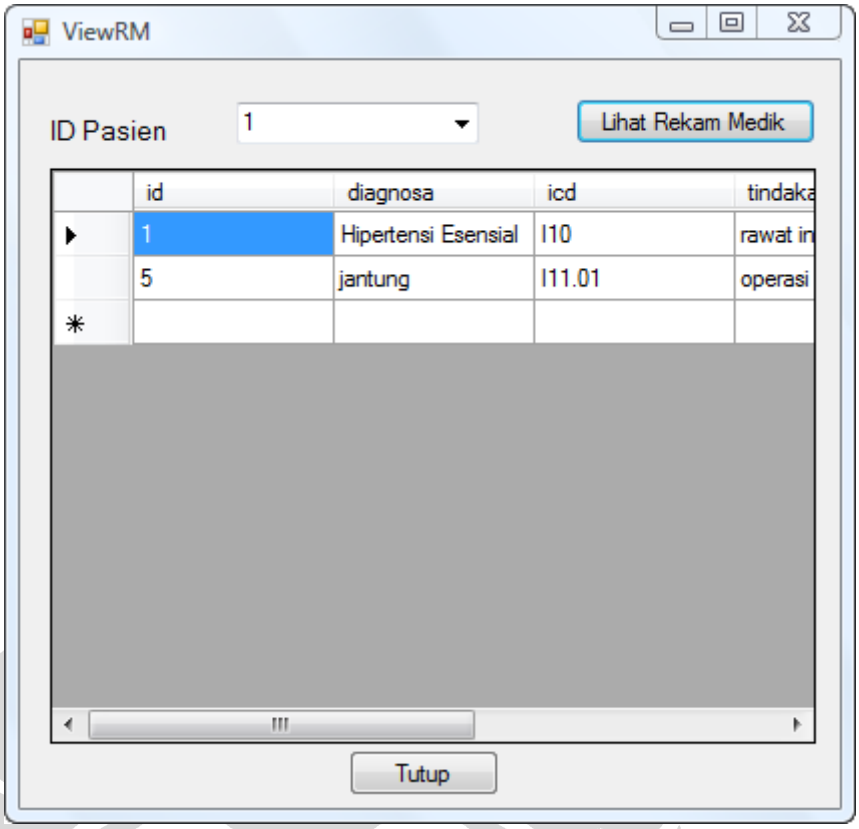
Semua peringatan yang ada, baik darurat maupun yang tidak, hanyalah status bahwa akan ada pasien yang akan dilayani.

Gambar 42 berikut ini adalah *interface* di sistem rumah sakit saat adanya *healthcare reservation*.



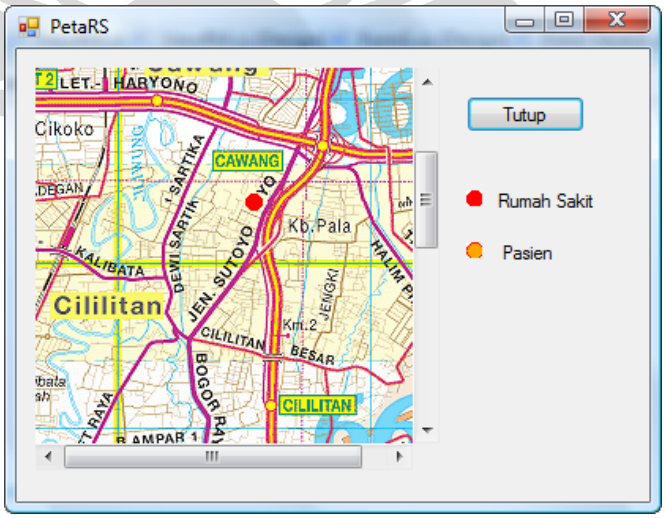
Gambar 42. *Interface* Warning Healthcare Reservation

Admin RS juga dapat melihat rekam medis pasien yang masuk ke dalam antrian *healthcare reservation* dengan menekan Lihat Rekam Medik, dan sistem rumah sakit akan menampilkan *windows* detail rekam medis pasien yang dipilih seperti pada **Gambar 43**.



Gambar 43. *View* Rekam Medis Pasien di Sistem RS

Selain itu, admin RS dapat melihat peta yang menunjukkan lokasi rumah sakit dan user yang memesan *healthcare reservation*, seperti yang dicontohkan pada **Gambar 44** berikut.



Gambar 44. *View* Peta di Sistem RS

Sedangkan di *mobile device* akan muncul tampilan pemesanan telah berhasil, seperti yang di contohkan pada **Gambar 45** berikut.

Gambar 45. ***Interface Healthcare Reservation Success***

5.3.3 *View Map*

Implementasi untuk modul *view map* dilakukan dengan menggunakan sebuah *map* buatan Gunther yang mencakup wilayah Jabodetabek (Jakarta, Bogor, Depok Tangerang, Bekasi) yang disimpan dalam *mobile device*.

Peta Gunther yang digunakan adalah peta yang dengan ukuran 3072 x 3072 *pixel*. Jika keseluruhan peta tersebut di-*load* melalui *mobile device*, maka akan sangat memberatkan kerja dari *mobile device*. Untuk mengatasi hal tersebut dan meminimalisir kerja dari *mobile device* dalam me-load peta, maka digunakanlah sebuah metode agar *mobile device* hanya me-*load* bagian peta yang dibutuhkan.

Peta gunther tersebut dipotong menjadi 144 bagian yang tersusun atas 12 bagian *horizontal* dan 12 bagian *vertical* yang masing-masing potongan peta bernilai 256 *pixel* x 256 *pixel*. Untuk mengetahui peta bagian mana yang perlu di *load*, maka kita harus mengetahui terlebih dahulu pada bagian peta yang mana letak posisi pasien, dan rumah sakit terdekat berada. Untuk mendapatkannya, perlu disimpan sebuah posisi titik pada peta untuk melakukan perbandingan. Dalam hal ini titik yang disimpan adalah titik (0,0) pixel sebagai berikut.

```
Public const double LATITUDE0 = 6.086902411764705882352941176471 ;
public const double LONGITUDE0 = 106.606276781512605042016806722 69
```

Selain informasi tentang titik pembanding itu disimpan pula sebuah konstanta perubahan *latitude* dan *longitude* untuk setiap pergeseran satu pixel. Konstanta tersebut dinamakan DELTALATITUDE dan DELTALONGITUDE

```
public const double DELTALATITUDE =
1.32647058823529411764705882352 94e-4 ;
public const double DELTALONGITUDE =
1.37840336134453781512605042016 81e-4 ;
```

Konstanta-konstanta tersebut didapatkan dengan cara membandingkan antara posisi dua buah titik di peta dan posisi *latitude* dan *longitude*-nya di bumi. Dalam hal ini digunakan posisi monas dan pancoran yang didapatkan dari Google Earth dan membandingkannya dengan posisi pixel di peta gunther.

Berikut ini adalah posisi-posisi yang didapatkan.

```
Monas
Latitude  = 6,175378
Longitude = 106,827097
Pixel x   = 66
Pixel y   = 155
Peta Bagian   = 3x7
```

```
Patung Pancoran
Latitude  = 6,243028
Longitude = 106,8435
Pixel x   = 185
Pixel y   = 153
Peta Bagian   = 5x7
```

Untuk mendapatkan DELTALATITUDE digunakanlah perhitungan ini

$$\text{DELTALATITUDE} = \frac{\text{Latitude Patung Pancoran} - \text{Latitude Monas}}{\text{Pixel Y Patung Pancoran} - \text{Pixel Y Monas}}$$

Sedangkan untuk mendapatkan DELTA*LONGITUDE* digunakanlah perhitungan ini

$$\text{DELTALONGITUDE} = \frac{\text{Longitude Patung Pancoran} - \text{Longitude Monas}}{\text{Pixel X Patung Pancoran} - \text{Pixel X Monas}}$$

Dan untuk mendapatkan posisi pixel X dan Y dari sebuah titik dengan posisi *longitude* dan *latitude* tertentu dilakukanlah perhitungan berikut ini.

```
/*menghitung perbedaan latitude yang diberikan dengan latitude
pixel 0,0 pada peta*/

            double delta = latitude – LATITUDE0;

 /*mengkalkulasikan pergeseran latitude titik peta dari titik
0,0*/

            int y = (int) Math.Round(delta / DELTALATITUDE);

/*menghitung perbedaan longitude yang diberikan dengan longitude
```

```
pixel 0,0 pada peta*/

            double delta = longitude – LONGITUDE0;

 /*mengkalkulasikan pergeseran longitude titik pada peta dari
titik 0,0*/

            int x = (int) Math.Round(delta / DELTALONGITUDE);
```

Setelah mendapatkan posisi pixel X dan Y dari posisi suatu titik, maka selanjutnya kita mencari pixel tersebut berada pada bagian peta yang mana untuk kemudian me-*load* peta tersebut ke *display mobile device*.

Untuk mengaktifkan menu View Map, pasien dapat memilih menu navigasi, yang ada saat telah dimunculkan daftar rumah sakit terdekat.

Setelah itu akan muncul bagian peta yang bersesuaian yang menampilkan posisi pasien dan rumah sakit. Contoh implementasinya dapat dlihat pada **Gambar 46** berikut ini.

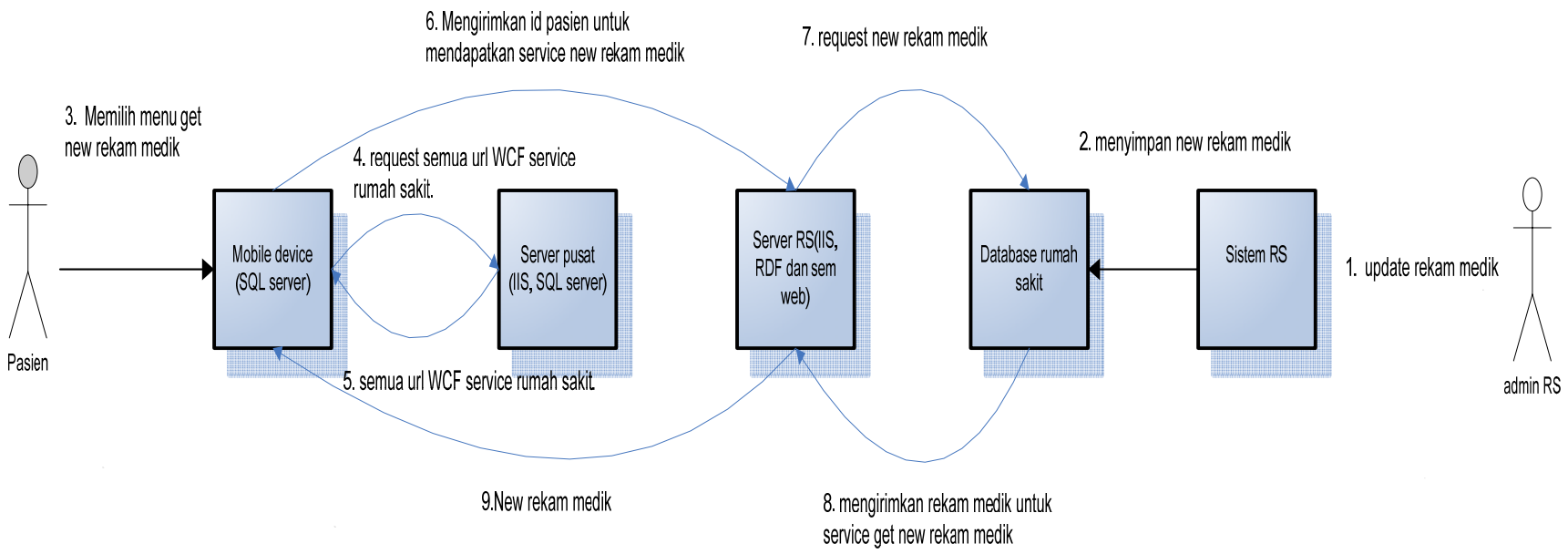
Gambar 46. *Interface View Map*

Terdapat sebuah legenda pada peta tentang warna titik yang menyatakan yang mana posisi pasien, dan yang mana posisi rumah sakit. Untuk menambahkan interaksi dibuatlah sebuah *event* yang akan memberikan informasi tentang sebuah titik jika titik tersebut di-klik. Seperti yang diimplementasikan seperti **Gambar 47** berikut.

Gambar 47. *Interface Event* di Menu *View Map*

5.3.4 *Update* dan *Get New* Rekam Medis

Gambar 48 berikut ini adalah *flow* implementasi *Update* dan *Get New* Rekam Medis yang dibuat

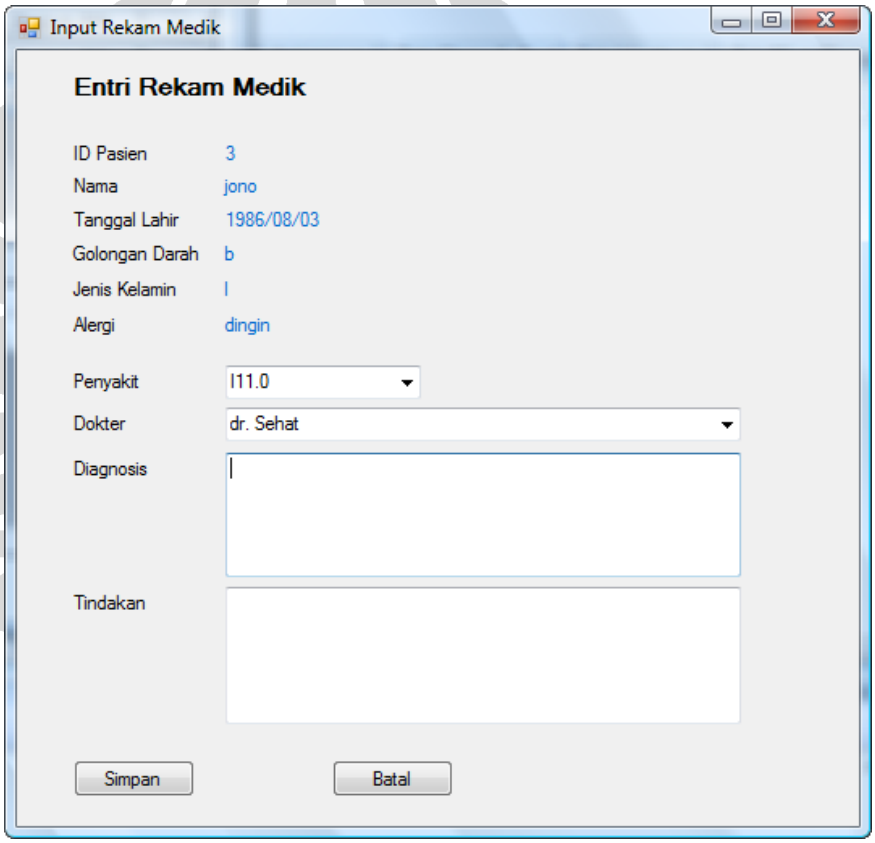


Gambar 48. *Flow* Implementasi *Update* dan *Get New* Rekam Medis

Pada modul *update* rekam medis, terdapat dua kegiatan yang terpisah namun saling berhubungan, yaitu:

- *Update* Rekam Medis, dan
- *Get New* Rekam Medis

Update rekam medis dilakukan pada sistem rumah sakit oleh admin rumah sakit. Sistem rumah sakit adalah sebuah aplikasi *desktop* yang memiliki menu untuk *update* rekam medis. **Gambar 49** berikut ini adalah *interface* untuk modul *update* rekam medis yang ada di sistem rumah sakit.



Gambar 49. *Interface Update* Rekam Medis

Sistem rumah sakit memfasilitasi *input* rekam medis baru sesuai pasien yang dilayani dan memasukkan data tersebut ke dalam basis data rumah sakit. Informasi yang termasuk di dalam data rekam medis antara lain adalah hasil diagnosis dan tindakan yang diberikan.

Untuk mengakses fungsi ini, admin RS dapat memilih menu *update* rekam medis. Dari menu tersebut akan muncul sebuah *windows* untuk memasukan data rekam medis baru pasien yang telah memesan melalui TRuST, dan menerima layanan medis sebelumnya.

Sedangkan *Get New* Rekam Medis merupakan sebuah modul yang ada pada *mobile device*. Ketika pasien memilih menu untuk *update* rekam medis. *Mobile device* menghubungi server pusat untuk mendapatkan list URL WCF *service* yang sudah mendaftarkan *service*-nya ke server pusat.

Daftar ini kemudian diterima oleh *mobile device*. Implementasi yang telah dibuat dapat dilihat pada **Gambar 50** berikut.

Gambar 50. *Interface Get New* Rekam Medis Dari Satu RS

Ketika user memilih menu Update RM dari salah satu rumah sakit, *mobile device* kemudian menghubungi WCF *service* rumah sakit tersebut dengan mengirimkan:

- ID pasien
- Tanggal dari rekam medis yang terakhir

WCF *service* rumah sakit kemudian mencari rekam medis pasien yang berada di *database* dan kemudian mengirimkan rekam medis yang terbaru, yang tanggalnya melebihi dari tanggal yang dikirimkan oleh *mobile device*. Rekam medis yang dikirimkan berupa:

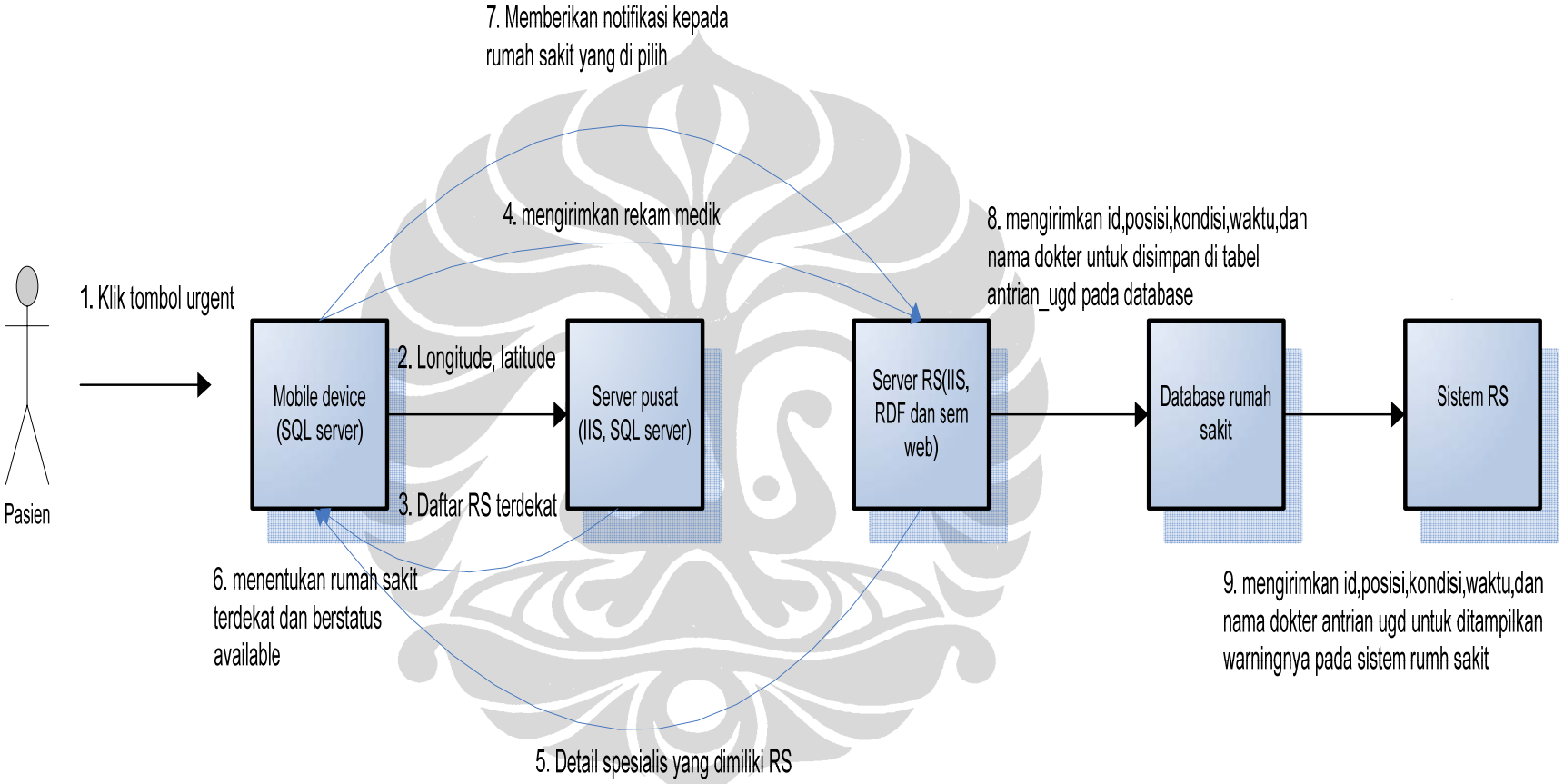
- Waktu periksa
- Diagnosa
- Kode ICD
- Nama dokter
- Serta tindakan yang dilakukan.

Data ini kemudian diterima oleh *mobile device* dan disimpan dalam *database* yang ada di *mobile device*. Apabila kapasitas di *mobile device* sudah melebihi batasnya, notifikasi bahwa penghapusan data rekam medis yang lama tampil. Implementasi untuk bagian ini dapat di lihat pada **Gambar 51** di bawah ini.

Gambar 51. *Interface* Get New Rekam Medis

5.3.5 *Urgent Signal*

Gambar 52 berikut ini adalah *flow* implementasi yang dibuat untuk modul *Urgent Signal*.



Gambar 52. *Flow* Implementasi *Urgent signal*

Implementasi yang dibuat pada menu *urgent signal* hampir menyerupai implementasi yang dilakukan pada menu "*Get* RS terdekat".

Saat pasien memilih menu 

pada *interface*, secara otomatis *mobile device* akan mencari posisi pasien saat itu dan mengirimkan *latitude* serta *longitude* pasien ke server pusat. Lalu server pusat kemudian melakukan pencarian rumah sakit yang terdekat dan mengembalikan data yang sama seperti pada menu "*Get* RS terdekat".

Setelah data tentang RS terdekat didapat, maka *mobile device* satu-persatu akan menghubungi WCF dari RS yang paling dekat untuk menanyakan apakah ada kamar UGD yang kosong atau tidak. Jika dari RS yang paling dekat memiliki kamar UGD yang kosong, maka dilakukan reservasi terhadap kamar tersebut. Sebaliknya, jika pada RS yang paling dekat tidak ada kamar UGD kosong, maka mobile device akan menghubungi WCF RS terdekat berikutnya untuk menanyakan hal yang sama.

Berbeda dengan implementasi "*Get* RS terdekat", pada bagian ini tidak dilakukan *reasoning* untuk mengetahui ketersediaan spesialis yang dibutuhkan. Bagian ini yang membedakan "*Urgent Signal*" dengan implementasi pada "*Get* RS terdekat".

Berikut ini adalah *code* yang di implementasikan untuk menentukan satu pilihan rumah sakit.

```
Public String launchDarurat()
        {
            DataSet rs = getListRS();//mendapatkan semua rumah
sakit dalam radius 10km di sort berdasarkan jarak
            String res = rs.DataSetName;
            if (res.Equals("error"))
            {
                return "terjadi error menghubungi server";
            }
            else
            {

                DataTable dt = rs.Tables[0];
                int jumlah = dt.Rows.Count;
                Debug.WriteLine("jumlah"+jumlah);
                if (jumlah > 0)//jika terdapat rs dalam radius 10
km
                  {
                                                    DataRow dr =
```

```
            rekamMedikDataSet1.Rekam_Medik.Rows[0];
                    String kode = dr[2].ToString();//kode
penyakit(icd) user/pasien

                    DataRow dr2 = pasienDataSet1.Pasien.Rows[0];
                    String idp = dr2[0].ToString();//id atau no
ktp pasien
                    String penyakit = dr[1].ToString();//nama
penyakit

                    String url = "";
                    String dokter = "";
                    for (int b = 0; b < dt.Rows.Count; b++)//loop
untuk list rs terdekat, dimulai dari yang paling deket

                    {
                        url = dt.Rows[b][4].ToString();
                        if (url != "" && url != "null" && url !=
null)
                        {



                            Service1Client sc = new
Service1Client(af.CreateDefaultBinding(), new
System.ServiceModel.EndpointAddress(url));
     int kamar = int.Parse(sc.PesanUGD(idp, "darurat", posisi,
null));
                        dt.Rows[b][3].ToString()+ " kamar
"+kamar.ToString());//mencoba memesan kamar, melalui wcf service,
yang akan mengembalikan no kamar

                            if(kamar!=0) //jika pemesanan
berhasil, dengan kata lain wcf mengembalikan no kamar(no kamar
tidak mungkin0)

                                return dt.Rows[b][0].ToString() +
": kamar telah dipesan, no kamar: " +kamar +" ambulan kami akan
segera menjemput anda di tempat";



                        }
                    }
                }
                    return "tidak ada rumah sakit, dalam radius 10
km";

            }
        }
```

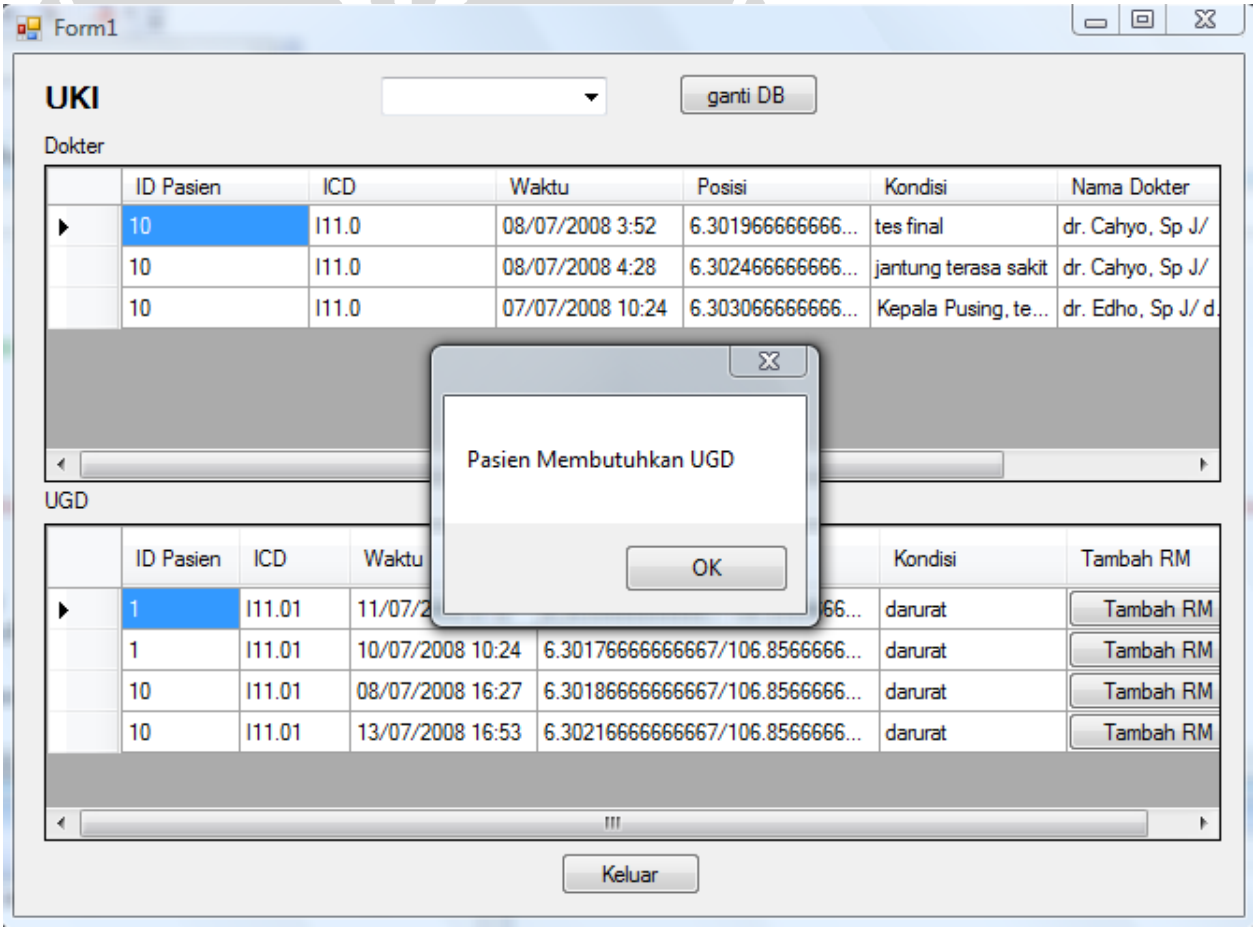
Saat terjadinya reservation, UGD *mobile device* mengirimkan data berupa:

- ID pasien

- Kondisi pasien
- Posisi pasien
- Serta mengembalikan daftar dokter spesialis yang dikirimkan oleh WCF *service*

Data ini kemudian diterima oleh WCF *service* untuk kemudian disimpan dalam *database* log UGD rumah sakit. *Database* yang sama terhubung dengan sistem rumah sakit bersangkutan yang akan selalu melakukan pengecekan terhadap *database* tentang adanya UGD *reservation* yang terjadi.

Jika ada UGD *reservation* yang terjadi, sistem rumah sakit akan menampilkan *window* yang berisi daftar reservasi UGD dan memberikan *warning signal*, seperti pada **Gambar 53** di bawah ini.



Gambar 53. *Interface Urgent Signal Warning*

Sedangkan di *mobile device* akan muncul tampilan bahwa akan ada bantuan yang datang ke pasien, seperti pada **Gambar 54** di bawah ini.

Gambar 54. *Interface Urgent Signal Success*

5.3.6 *View* Rekam Medis dan *Edit* Biodata

Modul lain yang berhasil diimplementasikan adalah *View* Rekam Medis dan *Edit* Biodata. Menu *view* rekam medis diimplementasikan untuk memenuhi kebutuhan tampilan rekam medis pasien di *mobile device* agar catatan riwayat penyakit yang dideritanya dapat dilihat. Implementasi yang dibuat untuk menu ini dapat dilihat pada **Gambar 55** di bawah ini.

Gambar 55. *View* Rekam Medis

Dan untuk memfasilitasi adanya perubahan biodata pasien, dibuatlah sebuah menu untuk meng-*edit* biodata pasien yang dapat di akses dengan memilih menu Edit Biodata pada bagian Menu, sehingga nantinya akan muncul tampilan seperti **Gambar 56** di bawah ini.

Gambar 56. *Interface Edit* Biodata

Data yang baru nantinya akan tersimpan di *database mobile device* dan akan dimunculkan setelah memilih menu Ubah Data.

Jika terdapat input yang tidak valid, maka *mobile device* akan menampilkan *message* seperti **Gambar 57** berikut.

Gambar 57. *Interface Message Input* Tidak Valid

5.3.7 *Class* Diagram

Berikut ini adalah gambar *class* digram yang di-*generate* oleh Visual Studio 2008 sesuai dengan implementasi yang telah dibuat.

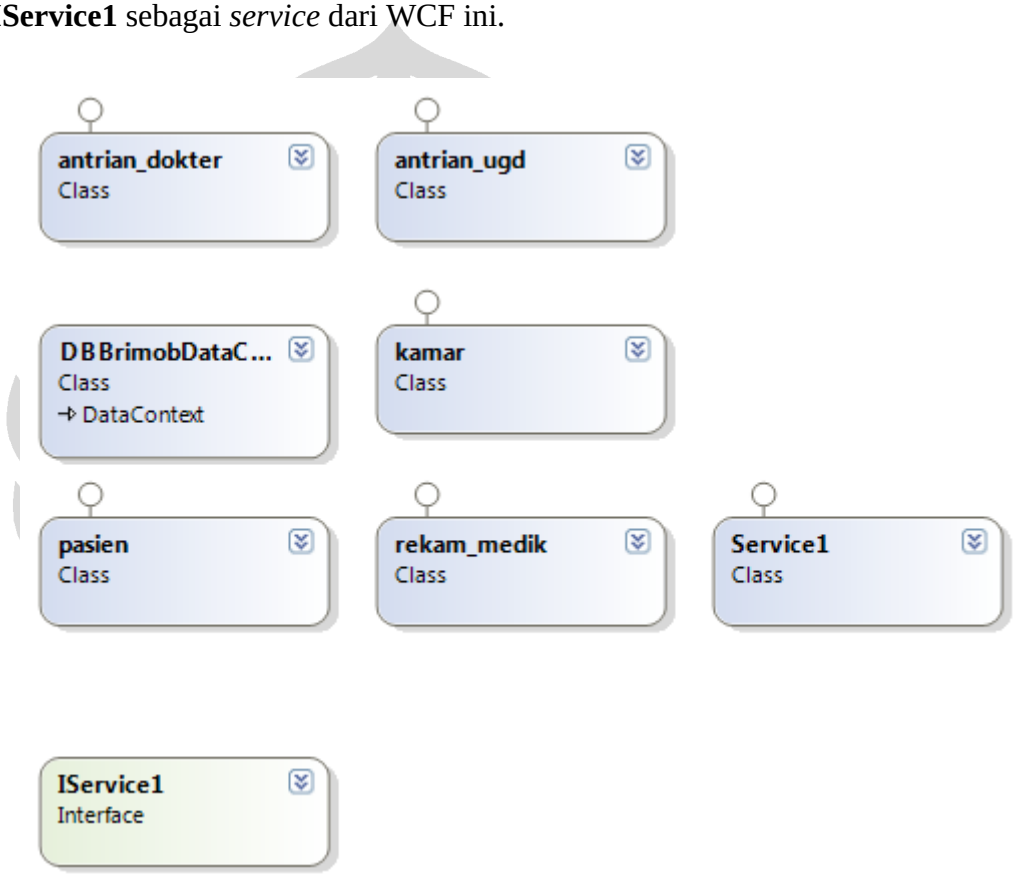
Gambar 58 adalah *class* diagram dari *solution* GPS, bagian dari modul aplikasi *mobile* yang di-*generate* oleh visual studio. Fungsi dari ***solution* GPS** ini adalah mengolah data GPS di *mobile device*, mulai dari membaca NMEA *message* hingga menerjemahkan atau mengekstrak informasi *latitude* dan *longitude* dari NMEA *message* tersebut.

Gambar 58. *Class* Diagram Solution GPS

Gambar 59 adalah *class* diagram dari modul aplikasi TRuST di *mobile device*. *Class* utama dari modul ini diantaranya adalah **RSTerdekat**, yang memuat fungsi yang akan menampilkan rumah sakit terdekat dari *user* berikut navigasi petanya. Lalu, *class* **RekamMedik** yang memiliki segala fungsi berkaitan dengan rekam medis seperti *view* rekam medis, atau *update* rekam medis. *Class* **DataDiri** memuat fungsi-fungsi berkaitan dengan *view* dan *edit* biodata user, sedangkan *class* **PetaJakarta** merupakan *class* untuk mengolah gambar yang akan ditunjukkan sebagai peta kepada *user*. *Class* **ServicePusatClient** digunakan untuk meng-*invoke* WCF dari server pusat, sedangkan **Service1Client** digunakan untuk meng-*invoke* WCF dari rumah sakit yang ada.

Gambar 59. *Class* Diagram *Mobile Device*

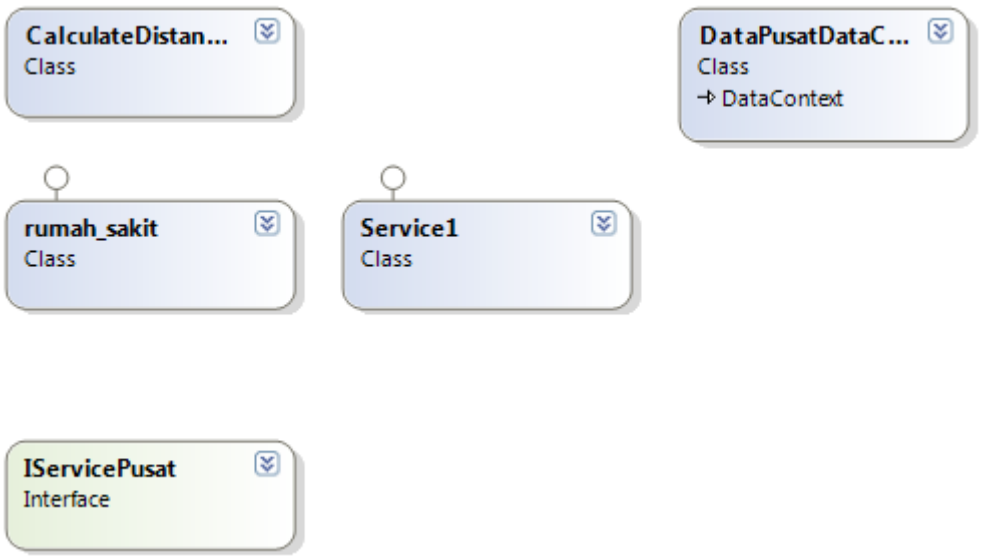
Gambar 60 adalah *class* diagram dari **WcfBrimob**, WCF yang diimplementasikan untuk rumah sakit brimob. WCF pada rumah sakit lain, memiliki *class* diagram serupa, hanya berbeda nama *class*. *Class* **DBBrimobDataContext** merupakan *class datacontext* dari Linq database rumah sakit. *Class* **antrian_dokter**, **antrian_ugd**, **kamar**, **pasien**, dan **rekam_medik** adalah *class* yang merepresentasikan tabel antrian_dokter, antrian_ugd, kamar, pasien dan rekam_medik di *database* rumah sakit bersangkutan. Sedangkan **Service1**, mengimplementasikan fungsi-fungsi yang didefinisikan di *interface* **IService1** sebagai *service* dari WCF ini.



Gambar 60. *Class* Diagram WCF *Service* Rumah Sakit

Gambar 61 adalah *class* diagram dari **WcfPusat**, WCF yang diimplementasikan untuk server pusat. Mirip seperti WCF di rumah sakit sebelumnya, **DataPusatContext** merupakan *class* **DataContext** dari linq *database* di server pusat, dan *class* **rumah_sakit** merepresentasikan tabel rumah_sakit di *database* tersebut. *Class* **CalculateDistance**, merepresentasikan *stored procedure*

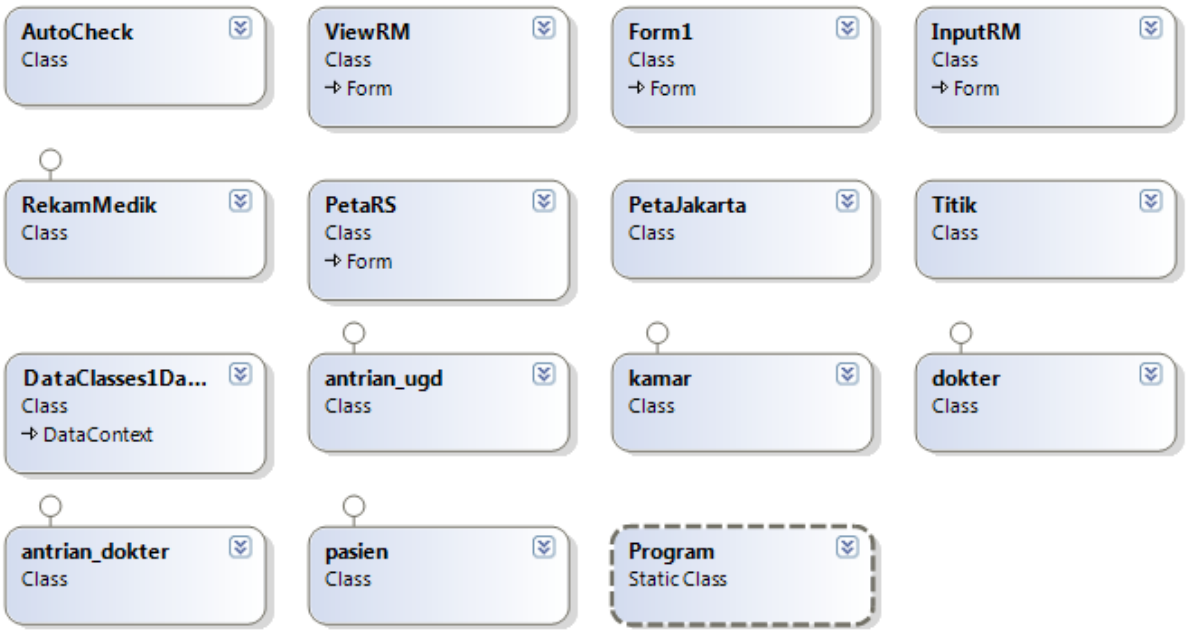
CalculateDistance yang dimiliki server pusat untuk menghitung jarak tiap rumah sakit ke posisi *user*. Sedangkan **Service1** mengimplementasikan fungsi-fungsi yang didefinisikan di *interface* **IService1**, sebagai *service* dari WCF ini.



Gambar 61. *Class* Diagram WCF Server Pusat

Gambar 62 berikut ini adalah *class* diagram dari sistem rumah sakit. Pada diagram tersebut dapat terlihat *class-class* yang terdapat pada sistem. Setiap *class* tersebut memiliki fungsi masing-masing. *Class* **Autocheck** berfungsi mengecek *database* untuk memastikan apabila ada pasien yang meminta pertolongan kepada rumah sakit. *Class* **ViewRM** berfungsi untuk menampilkan rekam medis yang tersimpan di rumah sakit. *Class* **Form1** memiliki fungsi untuk menampilkan *window* utama. *Class* **InputRM** memiliki fungsi untuk menampilkan form untuk memasukkan data rekam medis baru. *Class* **RekamMedik** adalah *class* yang berfungsi untuk menyediakan data rekam medis. *Class* **PetaRS** adalah *class* yang berfungsi untuk menampilkan peta yang menunjukkan lokasi pasien. *Class* **PetaJakarta** adalah *class* yang berfungsi menyimpan informasi peta yang akan ditampilkan oleh *class* PetaRS. *Class* **Titik** adalah *class* yang berfungsi untuk menyimpan informasi titik rumah sakit dan titik-titik pasien yang akan digambarkan pada peta. *Class* **antrian_ugd**, **kamar**, **dokter** dan **antrian_dokter** adalah *class-class* yang berfungsi mengambil informasi di *database* mengenai

kamar, dokter dan antrian pasien. Sementara *class* **Program** adalah *class* yang digunakan untuk menjalankan program pertama kali.



Gambar 62. *Class* Diagram Sistem Rumah Sakit

5.4 Metode pengujian

Pengujian yang telah dilakukan pada pengembangan TRuST ini menggunakan metode *black box testing*. *Black box testing* merupakan pengujian yang digunakan untuk memastikan bahwa sistem yang dibangun sudah melakukan semua fungsi yang diminta, menerima input secara benar, dan menghasilkan output yang sesuai serta memastikan bahwa informasi eksternal (seperti basis data) tetap terjaga. *Black box testing* mencoba menguak kesalahan-kesalahan pada fungsi yang tidak benar atau bahkan tidak ada, kesalahan antarmuka, kesalahan dalam struktur data atau akses basis data, kesalahan perilaku dan kinerja sistem, dan kesalahan inisiasi maupun akhiran suatu program dalam sistem [PRE05].

Pengujian sistem dilakukan secara bertahap, yaitu mulai dari *use-case*, hingga integrasi sistem. Untuk setiap *use-case* dibuat sebuah *Test Case* yang berisi tahapan-tahapan pelaksanaan pengujian. Dengan mengikuti alur yang dipaparkan

pada lampiran *Test Plan* tersebut, kegiatan pengujian dapat dilakukan dengan lebih teliti dan menyeluruh. *Test Plan* yang telah dilakukan dapat dilihat pada **Lampiran F**.

Pengujian TRuST dilakukan baik oleh pengembang maupun oleh pembimbing proyek mahasiswa. Ketika ditemukan masalah atau *requirement* yang baru, maka pengembang akan memperbaikinya dan melakukan penambahan agar benar-benar sesuai dengan *requirement*.

Tabel 15 berikut ini merupakan pengujian-pengujian yang telah dilakukan dengan hasil yang diharapkan dari masing-masing pengujian.

Tabel 15. Tabel Pengujian Sistem TRuST

No	Jenis pengujian	Hasil yang diharapkan	Hasil pengujian
1	*Get* RS Terdekat	Di layar *mobile device* muncul daftar rumah sakit terdekat beserta detail spesialis dokter yang dimiliki	Berhasil
2	*Healthcare reservation*	Di sistem rumah sakit yang dipilih muncul *warning healthcare reservation* atas nama pasien dan data pemesanan ini masuk ke dalam *database*.	Berhasil
3	*View Map*	Pada layar *mobile device* muncul bagian *map* yang memperlihatkan posisi pasien dan rumah sakit	Berhasil
4	*Urgent Signal*	Di sistem rumah sakit yang dipilih muncul *warning* UGD *reservation* atas nama pasien dan data pemesanan ini masuk ke dalam *database* dan di layar *mobile device* muncul *message* bahwa pertolongan akan segera	Berhasil

No	Jenis pengujian	Hasil yang diharapkan	Hasil pengujian
		sistem.	
5	*Update* rekam medis	Data rekam medis yang terbaru tersimpan ke dalam *database*.	Berhasil
6	*View* rekam medis	Pada layar *mobile device* ditampilkan rekam medis yang dimiliki oleh pasien. Data tersebut sama dengan yang tersimpan di *mobile device*.	Berhasil
7	*Get new* Rekam medis	Pada layar *mobile device* ditampilkan rekam medis terbaru pasien dari rumah sakit yang dipilih. Data tersebut sama dengan yang tersimpan di *database* rumah sakit.	Berhasil
8	*View Warning healthcare*	*Warning healthcare reservation* dan UGD *reservation* dapat dilihat oleh admin RS.	Berhasil
9	*Edit* Biodata	Biodata pasien yang tersimpan di *mobile device* tergantikan dengan data yang baru, dan data ini langsung ditampilkan pada layar *mobile device*.	Berhasil

5.5 Hambatan

Selama pelaksanaan proses ini, tim mengalami berbagai hambatan dalam memenuhi tanggung jawabnya. Berikut adalah hambatan-hambatan yang dimaksud:

1. Pengembangan sistem ini yang menggunakan Windows Mobile 6 menjadi hambatan tersendiri bagi penulis karena teknologi keluaran Microsoft tersebut tergolong baru dan belum banyak yang melakukan eksplorasi pada teknologi tersebut. Hal ini dapat dilihat di lapangan bahwa

dokumentasi yang menceritakan tentang pengembangan aplikasi *mobile* yang menggunakan J2ME lebih banyak dibandingkan Windows Mobile.

2. Sistem yang dikembangkan memiliki multidomain, selain di bidang IT, juga di bidang kesehatan. Sumber informasi yang kami dapatkan untuk bidang kesehatan dalam pengembangan sistem ini sangat terbatas, sehingga *requirement* yang dirasa cocok untuk aplikasi ini masih belum bisa didapatkan.

3. Sistem ini direncanakan dapat diimplementasikan dengan menggunakan wifi untuk koneksi internet sekaligus sebagai *location aware*, namun karena keterbatasan setting *Internet proxy* di lingkungan Universitas Indonesia dan emulator dari Windows Mobile, implementasinya hanya dapat dilakukan dengan menggunakan LAN untuk koneksi ke Internet.

4. Terbatasnya dokumentasi tentang *Semantic Web*/RDF *Library* untuk implementasi menggunakan bahasa C#/.NET yang dibuat oleh Joshua Tauberer menghambat penulis dalam mengimplementasikan *reasoning* untuk TRuST ini.

5. Tidak tersedianya *mobile device* yang asli serta sensor-sensor yang dapat digunakan untuk mendapatkan kondisi pasien yang sebenarnya pada saat itu, seperti sensor detak jantung dsb.