BAB 2
TINJAUAN PUSTAKA

2.1 Alarm

Jam alarm, dalam bentuk apa pun, adalah penanda cara mengatur waktu dan seringkali mengendalikan hidup kita, dari banyak kewajiban yang dimiliki orang dalam masyarakat kompleks di mana waktu memainkan peran yang sangat penting. Kita memulai hari dengan jam alarm membangunkan kita dan menghabiskan sisa hari itu sering dengan jam dari satu jenis atau yang lain terlihat dengan waktu di pikiran kita dan membentuk perilaku kita. Bagi banyak orang, jam alarm telah digantikan oleh alarm di smartphone mereka [1]

2.2 Global Positioning System

GPS adalah singkatan dari Global Positioning System, yang merupakan sistem navigasi dengan menggunakan teknologi satelit yang dapat menerima sinyal dari satelit. Cara kerja GPS secara logik ada 5 langkah:

1. Memakai perhitungan “triangulation” dari satelit.
2. Untuk perhitungan “triangulation”, GPS mengukur jarak menggunakan travel time sinyal radio.
3. Untuk mengukur travel time, GPS memerlukan memerlukan akurasi waktu yang tinggi.
4. Untuk perhitungan jarak, kita harus tahu dengan pasti posisi satelit dan ketingian pada orbitnya.
5. Terakhir harus menggoreksi delay sinyal waktu perjalanan di atmosfer sampai diterima reciever.

Sistem ini menggunakan 24 satelit yang mengirimkan sinyal gelombang mikro ke bumi. Sinyal ini diterima oleh alat penerima (receiver) di permukaan, dimana GPS receiver ini akan mengumpulkan informasi dari satelit GPS. Sebuah GPS receiver harus mengunci sinyal minimal tiga satelit untuk menghitung posisi 2D (latitude dan longitude) dan track pergerakan. Jika GPS reciever dapat menerima empat atau lebih satelit, maka GPS reciever akan dapat menghitung

posisi 3D (latitude, longitude dan altitude). Jika sudah dapat menentukan posisi user, selanjutnya GPS dapat menghitung informasi lain, seperti kecepatan, arah yang dituju, jalur, tujuan perjalanan, jarak tujuan, matahari terbit dan matahari terbenam dan lain-lain. Sinyal yang dikirimkan oleh satelit ke GPS akan digunakan untuk menghitung waktu perjalanan (travel time). Waktu perjalanan ini sering juga disebut sebagai Time of Arrival (TOA). Sesuai dengan prinsip fisika, bahwa untuk mengukur jarak dapat diperoleh dari waktu dikalikan dengan cepat rambat sinyal. Dari beberapa pemakaian GPS di atas dikategorikan menjadi:

1. Waktu. GPS receiver menerima informasi waktu dari jam atom yang mempunyai keakurasian sangat tinggi.
2. Lokasi. GPS memberikan informasi lokasi, latitude, longitude, dan altitude
3. Kecepatan. Ketika berpindah tempat, GPS dapat menunjukkan informasi kecepatan berpindah tersebut.
4. Arah perjalanan. GPS dapat menunjukkan arah tujuan. Simpan lokasi. Tempat-tempat yang sudah pernah atau ingin dikunjungi bisa disimpan oleh GPS receiver.
5. Komulasi data. GPS receiver dapat menyimpan informasi track, seperti total perjalanan yang sudah pernah dilakukan, kecepatan rata-rata, kecepatan paling tinggi, kecepatan paling rendah, waktu/jam sampai tujuan dan sebagainya.
6. Tracking. Membantu untuk memonitoring pergerakan obyek. Membantu memetakan posisi tertentu dan perhitungan jaringan terdekat[11].

2.3 *LBS*

Sistem layanan berbasis lokasi atau yang biasa dikenal *dengan location based services* (LBS), menggabungkan antara proses dari layanan *mobile* dengan posisi geografis dari penggunanya. LBS mampu mendeteksi lokasi pengguna berada sehingga dapat memberikan layanan sesuai dengan lokasi pengguna tersebut. Lokasi geografis pengguna ditentukan dengan menggunakan layanan terpisah seperti misalnya *Global Positioning System* (GPS), yang mana GPS adalah sistem yang berfungsi sebagai sistem navigasi global yang dapat menerima

informasi dari sistem satelit. Satelit GPS ini memancarkan sinyal yang memungkinkan penerima sinyal GPS untuk mendapatkan informasi berupa posisi, kecepatan, maupun waktu [4].

2.4 SQLite

SQLite adalah mesin database SQL tertanam. Tidak seperti kebanyakan database SQL lainnya, SQLite tidak memiliki proses server yang terpisah. SQLite membaca dan menulis langsung ke file disk biasa. Database SQL lengkap dengan beberapa tabel, indeks, pemicu, dan tampilan, terkandung dalam satu file disk. Format file basis data adalah lintas-platform - Anda dapat dengan bebas menyalin database antara sistem 32-bit dan 64-bit atau antara arsitektur big-endian dan little-endian. Fitur-fitur ini menjadikan SQLite pilihan populer sebagai Format File Aplikasi. File database SQLite adalah format penyimpanan yang direkomendasikan oleh US Library of Congress. Pikirkan SQLite bukan sebagai pengganti untuk Oracle tetapi sebagai pengganti untuk fopen().

SQLite adalah pustaka yang ringkas. Dengan semua fitur yang diaktifkan, ukuran pustaka bisa kurang dari 600KiB, tergantung pada platform target dan pengaturan optimisasi kompiler. (Kode 64-bit lebih besar. Dan beberapa optimisasi kompiler seperti fungsi agresif inlining dan loop unrolling dapat menyebabkan kode objek menjadi jauh lebih besar.) Ada tradeoff antara penggunaan memori dan kecepatan. SQLite umumnya berjalan lebih cepat semakin banyak memori yang Anda berikan. Namun demikian, kinerja biasanya cukup baik bahkan di lingkungan dengan memori rendah. Bergantung pada bagaimana ia digunakan, SQLite bisa lebih cepat daripada I/O sistem file langsung.[12]

2.5 Pengertian dan Konsep dari Model *Waterfall*

Metode Waterfall adalah sebuah metode pengembangan sistem dimana antar satu fase ke fase yang lain dilakukan secara berurutan. Dalam proses implementasi metode Waterfall ini, sebuah langkah akan diselesaikan terlebih dahulu dimulai dari tahapan yang pertama sebelum melanjutkan ke tahapan yang

berikutnya. Adapun keuntungan menggunakan metode waterfall ini yaitu requirement harus didefinisikan lebih mendalam sebelum proses coding dilakukan, selain itu proses implementasinya dilakukan secara bertahap dari tahap pertama hingga tahap terakhir secara berurutan. Disamping itu, metode Waterfall ini juga memungkinkan sedikit mungkin perubahan yang dilakukan oleh proyek berlangsung.

Adapun metode Waterfall menurut Ian Sommerville [9], memiliki tahapan utama dari waterfall model yang mencerminkan aktifitas pengembangan dasar. Terdapat 5 (lima) tahapan pada metode Waterfall, yaitu Requirements Definition, System and Software Design, Implementation and Unit Testing, Integration and SystemTesting, Operation and Maintenance.

Adapun penjelasan dari tahapan-tahapan metode waterfall menurut Ian Sommerville tersebut sebagai berikut.

1. Requirement Analysis and Definition adalah tahapan penetapan fitur, kendala dan tujuan sistem melalui konsultasi dengan pengguna sistem. Semua hal tersebut akan ditetapkan secara rinci dan berfungsi sebagai spesifikasi sistem.
2. Pada Tahap System and Software Design ini akan dibentuk suatu arsitektur sistem berdasarkan persyaratan yang telah ditetapkan. Sekain itu juga, dilakukan identifikasi dan penggambaran terhadap abstraksi dasar sistem perangkat lunak beserta hubungan-hubungannya.
3. Dalam tahapan Implementation and Unit Testing ini, hasil dari desain perangkat lunak akan direalisasikan sebagai satu set program atau unit program. Setiap unit akan diuji apakah sudah memenuhi spesifikasinya.
4. Dalam tahap Integration and System Testing, setiap unit program akan diintegrasikan satu sama lain dan diuji sebagai satu sistem yang utuh untuk memastikan sistem sudah memenuhi persyaratan yang ada. Setelah itu sistem akan dikirim ke pengguna sistem.
5. Dalam tahap Operation and Maintenance ini, sistem diinstal dan mulai digunakan. Selain itu juga memperbaiki error yang tidak ditemukan pada tahap pembuatan. Dalam tahap ini juga dilakukan pengembangan sistem seperti penambahan fitur dan fungsi baru.

2.6 Android

Android merupakan sistem operasi *mobile* berbasis linux yang dirancang untuk perangkat yang memiliki kemampuan layar sentuh seperti *smartphone* maupun *tablet*. Android dibeli Google pada tahun 2005 dan pada tahun 2008 tepatnya pada tanggal 22 Oktober 2008 diluncurkan *smartphone* berbasis Android pertama dengan nama HTC Dream. Dikarenakan Android bersifat *open source* perkembangannya sangat pesat sekali. Sekarang sistem operasi Android tidak hanya dapat berjalan pada *smartphone* maupun *tablet* saja tetapi dapat berjalan pada perangkat seperti laptop, TV maupun *smart watch*.[5]

2.7 Android Studio

Android Studio adalah Lingkungan Pengembangan Terpadu – Integrated Development Environment (IDE) untuk pengembangan aplikasi Android, berdasarkan IntelliJ IDEA[12]. Selain merupakan editor kode IntelliJ dan alat pengembang yang berdaya guna, Android Studio menawarkan fitur lebih banyak untuk meningkatkan produktivitas pengembang saat membuat aplikasi Android, misalnya:

1. Sistem versi berbasis Gradle yang fleksibel.
2. Emulator yang cepat dan kaya fitur.
3. Lingkungan yang menyatu untuk pengembangan bagi semua perangkat Android.
4. Instant Run untuk mendorong perubahan ke aplikasi yang berjalan tanpa membuat APK baru.
5. Template kode dan integrasi GitHub untuk membuat fitur aplikasi yang sama dan mengimpor kode contoh.
6. Alat pengujian dan kerangka kerja yang ekstensif.
7. Alat Lint untuk meningkatkan kinerja, kegunaan, kompatibilitas versi dan masalah-masalah lain.
8. Dukungan C++ dan NDK
9. Dukungan bawaan untuk Google Cloud Platform, mempermudah pengintegrasian Google Cloud Messaging dan App Engine.

Tujuan dibuatnya Android Studio adalah untuk mempercepat pengembangan dan membantu pengembang membuat aplikasi berkualitas tinggi untuk setiap perangkat Android. Menawarkan alat bantu yang dibuat khusus untuk pengembang Android, meliputi pengeditan kode yang lengkap, debugging, pengujian dan alat pembuatan profil.

2.8 Java

Java adalah bahasa pemrograman yang berorientasi objek (OOP) dan dapat dijalankan pada berbagai platform sistem operasi. Perkembangan java tidak hanya terfokus pada satu sistem operasi, tetapi juga dikembangkan untuk berbagai sistem operasi dan bersifat *open source* [6]. Bahasa ini awalnya dibuat oleh James Gosling saat masih bergabung di Sun Microsystems saat ini merupakan bagian dari Oracle dan dirilis tahun 1995. Bahasa ini banyak mengadopsi sintaksis yang terdapat pada C dan C++ namun dengan sintaksis model objek yang lebih sederhana. Aplikasi-aplikasi berbasis Java umumnya dikompilasi ke dalam p-code (bytecode) dan dapat dijalankan pada berbagai Mesin Virtual Java (JVM).

Java merupakan bahasa pemrograman yang bersifat umum/non-spesifik (*general purpose*), dan secara khusus didisain untuk memanfaatkan dependensi implementasi seminimal mungkin. Karena fungsionalitasnya yang memungkinkan.

Aplikasi Java mampu berjalan di beberapa platform sistem operasi yang berbeda, Java dikenal pula dengan slogannya, "Tulis sekali, jalankan di mana pun". Saat ini Java merupakan bahasa pemrograman yang paling populer digunakan, dan secara luas dimanfaatkan dalam pengembangan berbagai jenis perangkat lunak aplikasi ataupun aplikasi berbasis web.

Java menggunakan model pengamanan tiga lapis (*three-layer security model*) untuk melindungi sistem dari *untrusted* Java code. Pertama, bytecode *verifier* membaca bytecode sebelum dijalankan dan menjamin bytecode memenuhi aturan-aturan dasar bahasa Java. Kedua, *class loader* menangani pemuatan kelas Java ke *runtime interpreter*. Ketiga, manajer keamanan menangani keamanan tingkat aplikasi dengan mengendalikan apakah program

berhak mengakses sumber daya seperti sistem file, port jaringan, proses eksternal dan sistem window.

Java termasuk bahasa *Multithreading*. *Thread* adalah untuk menyatakan program komputer melakukan lebih dari satu tugas di satu waktu yang sama. Java menyediakan kelas untuk menulis program *multithreaded*, program mempunyai lebih dari satu thread eksekusi pada saat yang sama sehingga memungkinkan program menangani beberapa tugas secara konkuren.

Program Java melakukan *garbage collection* yang berarti program tidak perlu menghapus sendiri obyek-obyek yang tidak digunakan lagi. Fasilitas ini mengurangi beban pengelolaan memori oleh pemrogram dan mengurangi atau mengeliminasi sumber kesalahan terbesar yang terdapat di bahasa yang memungkinkan alokasi dinamis.

Java mempunyai mekanisme *exception-handling* yang ampuh. *Exception handling* menyediakan cara untuk memisahkan antara bagian penanganan kesalahan dengan bagian kode normal sehingga menuntun ke struktur kode program yang lebih bersih dan menjadikan aplikasi lebih tegar. Ketika kesalahan yang serius ditemukan, program Java menciptakan *exception. Exception* dapat ditangkap dan dikelola program tanpa resiko membuat sistem menjadi *down*. Program Java mendukung native method yaitu fungsi ditulis di bahasa lain, biasanya C/C++.

Adapun kelebihan dari Java yaitu:

Java mempunyai beberapa keunggulan dibandingkan dengan Bahasa pemrograman lainnya. Keunggulan bahasa pemrograman Java antara lain:

1. Berorientasi obyek

   Java adalah bahasa pemrograman yang berorientasi pada obyek. Java membagi program menjadi obyek - obyek serta memodelkan sifat dan tingkah laku masing-masing dalam menyelesaikan suatu masalah.

2. Java bersifat *multiplatform*

   Java dirancang untuk mendukung aplikasi yang dapat beroperasi di lingkungan jaringan berbeda. Untuk mengakomodasi hal tersebut, Java compiler membangkitkan bytecodes (sebuah format yang tidak tergantung

pada arsitektur tertentu yang didesain untuk mengirimkan kode ke banyak platform perangkat keras dan perangkat lunak secara efisien). Java dapat dijalankan oleh banyak platform seperti Linux, Unix, Windows

3. Java bersifat *multithread*

   Multithreading adalah kemampuan suatu program komputer untuk mengerjakan beberapa proses dalam suatu waktu. Thread dalam Java memiliki kemampuan untuk memanfaatkan kelebihan multi prosessor apabila sistem operasi yang digunakan mendukung multi prosessor.

4. Dapat didistribusi dengan mudah

   Java memiliki *library* rutin yang lengkap untuk dirangkai pada *protocol* TCP/IP (seperti HTTP dan FTP) dengan mudah. Kemampuan networking Java lebih kuat dan lebih mudah digunakan. Java memudahkan tugas pemrograman jaringan yang sulit seperti membuka dan mengakses sebuah soket koneksi.

5. Bersifat dinamis

   Java dirancang untuk beradaptasi dengan lingkungan yang sedang berkembang. Java bersifat dinamis dalam tahap *linking*. Class yang ada dapat di-*link* sebatas yang diperlukan, apabila diperlukan modul kode yang baru dapat di-*link* dari beberapa sumber

Kekurangan Java yaitu:

1. Tulis sekali

   Masih ada beberapa hal yang tidak kompatibel antara *platform* satu dengan *platform* lain. Untuk J2SE, misalnya SWT-AWT bridge yang sampai sekarang tidak berfungsi pada Mac OS X.

2. Mudah didekompilasi

   Dekompilasi adalah proses membalikkan dari kode jadi menjadi kode sumber. Ini dimungkinkan karena kode jadi Java merupakan bytecode yang menyimpan banyak atribut bahasa tingkat tinggi, seperti nama-nama kelas, metode, dan tipe data. Hal yang sama juga terjadi pada Microsoft .NET *Platform*. Dengan demikian, algoritma yang digunakan program akan lebih sulit disembunyikan

3. Penggunaan memori yang banyak

   Penggunaan memori untuk program berbasis Java jauh lebih besar daripada bahasa tingkat tinggi generasi sebelumnya seperti C/C++ dan Pascal (lebih spesifik lagi, Delphi dan Object Pascal). Biasanya ini bukan merupakan masalah bagi pihak yang menggunakan teknologi terbaru (karena *trend* memori terpasang makin murah)

2.9 API

Application programming interface (API) adalah serangkaian prosedur, fungsi, protokol, dan aturan. API menentukan bagaimana komponen perangkat lunak atau paket yang berbeda harus berinteraksi satu sama lain. API seperti halnya penghubung dua orang yang sedang bekerja sama dengan dengan menggunakan bahasa yang sama sekali berbeda.[15]

2.10 Google Direction API

Google Direction API adalah layanan yang menghitung arah antar lokasi menggunakan permintaan HTTP. *Google Direction API* memiliki beberapa parameter yang di butuhkan untuk meminta lama perjalanan. Daftar dari parameter yang ada dapat dilihat pada tabel 2.1 Daftar parameter yang dibutuhkan *Google Direction API* [7]

Tabel 2.1 Daftar parameter yang dibutuhkan Google Direction API

Nama Parameter	Masukan
Origin	Koordinat atau Nama lokasi
Destination	Koordinat atau Nama lokasi
key	Kunci API

Parameter yang Terdapat pada Tabel 2.1 merupakan parameter yang dibutuhkan untuk Google Maps Google Maps Direction API untuk dapat digunakan parameter *Origin* merupakan parameter untuk Titik Awal perhitungan, parameter *Destination* Merupakan parameter untuk titik akhir perhitungan,

parameter Key merupakan kunci API yang digunakan pada aplikasi, *Google Maps Direction API* sendiri memiliki beberapa parameter Opsional yang diberikan daftar dari parameter opsional yang ada dapat dilihat pada tabel 2.2 Daftar parameter opsional *Google Maps Direction API*

Tabel 2.2 Daftar parameter opsional Google Maps Direction API

Nama Parameter	Masukan
mode	*Driving,walking,bicycling,transit*
waypoints	Koordinat Lokasi atau Nama Lokasi
alternatives	*True* atau *flase*
language	Bahasa yang digunakan
region	*country code top-level domain*
Avoid	*Tolls,highway,ferries,indoor*
Units	Satuan yang digunakan
Arrival_time	*Unix Time*
Departure_time	*Unix Time,now*
Traffic_model	*Best_guess,Pessimistic,optimistic*
Transit_mode	*Bus,subway,train,tram,rail*
Transit_routing_preference	*Less_walking,fewer_transfer*

Penjelasan mengenai parameter parameter pada Tabel 2.2 adalah sebagai berikut:

1. *Mode*

   Merupakan parameter untuk menentukan mode transportasi yang akan digunakan saat menghitung arah

2. *Waypoints*

   Merupakan parameter untuk menentukan susunan lokasi perantara untuk disertakan di sepanjang rute antara titik asal dan tujuan sebagai melewati atau menghentikan lokasi. Titik jalan mengubah rute dengan mengarahkannya melalui lokasi yang ditentukan

3. *Alternatives*

menetapkan bahwa layanan arah dapat memberikan lebih dari satu alternatif rute dalam respons

4. *Language*

   Merupakan parameter bahasa yang akan digunakan untuk hasil dari *Google Maps Direction API*

5. *Region*

   Merupakan Parameter area untuk membatasi hasil dari *geocoder*

6. *Avoid*

   Merupakan Parameter yang membatasi jalur yang digunakan seperti tidak melewati toll

7. *Units*

   merupakan Parameter yang menentukan satuan jarak yang akan di gunakan pada hasil

8. *Arrival Time*

   Merupakan parameter yang digunakan untuk menspesifikasi waktu Sampai yang di inginkan namun parameter ini hanya dapat digunakan bila berkendara menggunakan transit

9. *Departure Time*

   Merupakan parameter yang digunakan untuk menspesifikasi waktu keberangkatan yang di inginkan parameter ini membutuhkan masukan Unix Time yaitu detik semenjak 1970 januari 1 *UTC* (*Coordinated universal Time)*

10. *Traffic model*

    Merupakan Parameter yang dipakai untuk asumsi perhitungan prediksi lama Perjalanan

11. *Transit model*

    Merupakan Parameter yang dipakai untuk asumsi perhitungan prediksi lama Perjalanan transit

12. *Transit routing preference*

    Merupakan parameter untuk memilih jalur transit apakah lebih banyak jalan atau berkendara

2.10.1 Google direction API Travel time prediction

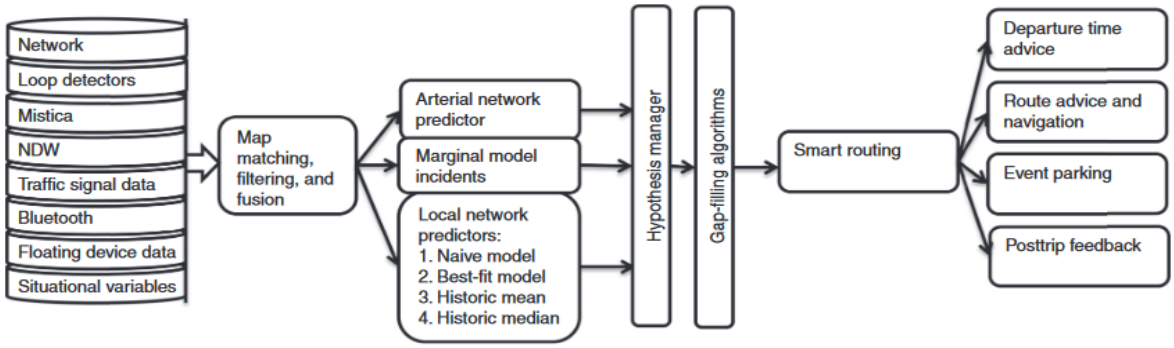
Prediksi pada Gooogle Direction API sendiri menggunakan data kondisi lalu lintas terdahulu [7] dimana data lalu lintas yang digunakan merupakan data lalu lintas waktu pada satu hari dan hari dalam satu minggu, dimana pessimistic dan optimistic hanya menggunakan data best_guess dapat pula mempertimbangkan kondisi lalu lintas yang sedang terjadi [14].

2.11 UNIX time

UNIX time adalah adalah sistem untuk menggambarkan suatu titik waktu berbasis integer dari jumlah detik tanpa lompatan yang berlalu sejak 00:00 1 Januari 1970 UTC. Meningkatnya penggunaan sistem mirip UNIX dalam aplikasi Internet dan *smartphone* berarti bahwa metode ketepatan waktu komputasi ini hampir memonopoli representasi waktu dalam sistem digital. Perangkat iOS, Android, Linux, dan MacOS semuanya menggunakan standar ini dalam perhitungan waktu[8].

2.12 Real-Time Travel Time Prediction Framework for Departure Time

Sebuah kerangka kerja model yang terdiri dari beberapa submodel untuk memprediksi waktu perjalanan door to door untuk seluruh jaringan secara real-time telah diajukan. Arsitektur yang digunakan untuk memberikan saran ditunjukkan pada Gambar 2.1 amsterdam *Practical Trial*



Gambar 2.1

Di sisi kiri gambar, berbagai sumber data terapan ditampilkan. Sumber data berisi data historis dan waktu-nyata. Algoritma pencocokan peta digunakan untuk mencocokkan data dengan peta. Model prediksi dijalankan dengan data yang menyatu ini. Hasil model prediksi digabungkan oleh *Hypotesis Manager*(HM). Pada bagian berikut dijelaskan bagaimana HM menggabungkan hasil dari berbagai prediktor *local arterial*, dan *incident*. Untuk area *Arterial*, hasil dari *Arterial Network Predictor* (ANP) selalu digunakan kecuali prediktor ini ditolak oleh hasil *Marginal Model Incident* (MMI), yang diterapkan apabila terjadi insiden yang tidak terduga. Prediksi gabungan untuk *Local Network Predictors* (LNP), yang diterapkan pada jalan tingkat bawah, digunakan untuk jenis jalan lainnya. Langkah-langkah ini hanya dapat dilakukan untuk jalan yang datanya tersedia. Untuk dapat memberikan saran rute, waktu perjalanan yang realistis dan prediksi waktu perjalanan juga harus tersedia untuk jalan lain. Oleh karena itu, kombinasi algoritma pengisian celah diimplementasikan. Hasilnya adalah bahwa setiap 5 menit, prediksi jangka pendek baru (3 jam ke depan) untuk semua jalan dalam jaringan menjadi tersedia untuk algoritma rute pintar yang digunakan untuk memberikan saran yang berbeda. Pencocokan peta, penggabungan data, strategi saran yang berbeda, dan rute yang cerdas bukanlah subjek tulisan ini; Namun, masing-masing model prediksi dijelaskan dalam detail yang lebih besar

2.12.1 Local Network Predictors

LNP untuk saat ini ada empat model prediksi, tetapi dia dirancang untuk sejumlah model yang berubah ubah. Karena penerapan kerangka keseluruhan menuntut prediksi waktu-nyata yang cepat dan kuat, model yang diterapkan dalam LNP sengaja dibuat sederhana. Model yang diterapkan adalah model naif, model paling cocok, rata-rata historis, dan median bersejarah

2.12.2 Hypotesis Manager

HM dikembangkan untuk mengevaluasi kualitas prediksi dari masing-masing model dari LNP dan menggabungkan hasil dari kombinasi berbagai

variabel prediksi. Prediksi dari masing-masing metode prediksi dievaluasi dan parameter berikut dibedakan:

1. Kategori jalan. Tiga kategori (A, B, dan C) didefinisikan berdasarkan batas kecepatan maksimum; jalan tingkat tinggi, sedang, dan rendah dikategorikan berdasarkan batas kecepatan maksimum dengan nominal ≥100 km / jam, 70–80 km / jam, dan ≤50 km / jam, masing-masing;
2. Hari di minggu ini. Masing-masing dari 7 hari dalam seminggu secara individual;
3. Waktu hari. Dalam blok agregat 5 menit;
4. Cakrawala prediksi. Dari 0 mnt (kondisi saat ini) selama selang 5 menit hingga 3 jam sebelumnya.

Untuk setiap kombinasi dari variabel prediksi, prediksi dibuat oleh masing-masing metode prediksi dalam LNP. Setiap prediksi dievaluasi oleh prediktor HM terhadap data pelatihan menggunakan mean absolute error (E) karena ukuran kesalahan model awalnya dilatih untuk tahun 2013 dengan menggunakan data ekstensif yang dikumpulkan dalam *database National Data Warehouse for Traffic Information* (NDW) di Belanda

2.12.3 Arterial Network Predictor

ANP diterapkan untuk prediksi waktu perjalanan pada rute arteri utama yang kualitas datanya baik secara konsisten tersedia dalam waktu dan ruang. Jalan raya di wilayah Amsterdam yang lebih besar hampir seluruhnya dilengkapi dengan loop induksi ganda sebagai bagian dari sistem pemantauan Casco (MONICA) dan karenanya tersedia untuk prediksi menggunakan ANP. Sebagai bagian dari saran rute, ANP memprediksi waktu perjalanan dengan dua tingkat: jangka panjang (lintas hari) dan jangka pendek (30 menit hingga 3 jam sebelumnya)

Prediksi jangka panjang diterapkan pada prediksi yang dilakukan satu hari atau lebih sebelum perjalanan dilakukan. Karena alasan ini prediksi tidak memperhitungkan status lalu lintas saat ini; melainkan, ia menggunakan negara bagian yang diharapkan berdasarkan pola lalu lintas bersejarah

Prediksi jangka pendek dilakukan secara real time menggunakan gabungan data keadaan lalu lintas heterogen, seperti yang awalnya dijelaskan oleh Treiber dan Helbing, Treiber et al., Dan Van Lint dan Hoogendoorn. Metode ini menggunakan pola spasial temporal dalam arus lalu lintas yang dikombinasikan dengan gelombang kinematik yang didefinisikan dalam teori arus lalu lintas. Prediksi dibuat melalui penyebaran pola lalu lintas spatiotemporal ini ke masa depan untuk menggambarkan keadaan lalu lintas masa depan.

2.12.4 Marginal Model Incident

Deteksi insiden dilakukan terutama melalui deteksi penutupan jalur otomatis dari badan jalan raya Belanda (Rijk-swaterstaat) dan kedua melalui sistem registrasi insiden. Data ketersediaan jalur tersedia, bersama dengan kecepatan lalu lintas langsung dan data penghitungan, dan memberikan ketersediaan jalur dengan penundaan kurang dari 1 menit. Deteksi jalur tertutup, bagaimanapun, tidak selalu berarti bahwa suatu insiden telah terjadi karena luapan, periode puncak, dan jalur pasang surut sering ditutup pada hari ketika lalu lintas lebih tenang. Oleh karena itu diterapkan algoritma pendeteksian jalur, yang menyaring jumlah jalur yang tersedia untuk hari tertentu dan dibandingkan dengan jumlah jalur terbuka pada waktu tertentu.

2.13 Unified Modelling Language

Unified Modelling Language (UML) adalah sebuah bahasa pemodelan *visual* yang digunakan untuk memvisualisasikan, menspesifikasikan, membangun dan mendokumentasikan artefak sistem perangkat lunak [9]. UML digunakan untuk memahami, mendesain, mengeksplorasi, mengkonfigurasi, memelihara, dan mengontrol informasi tentang sistem. Hal Ini dimaksudkan untuk digunakan dengan semua metode pengembangan, tahapan siklus hidup, domain aplikasi, dan media.

UML bukanlah bahasa pemrograman, melainkan alat yang dapat menyediakan generator kode dari UML ke dalam berbagai bahasa pemrograman serta membangun model rekayasa balik dari program yang ada. UML adalah

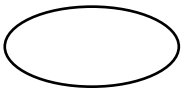
bahasa pemodelan diskrit, yang dimaksudkan untuk menjadi bahasa pemodelan umum untuk sistem diskrit seperti yang terbuat dari perangkat lunak, *firmware*, atau logika digital.

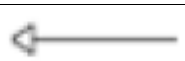
UML diadopsi dengan suara bulat oleh keanggotaan *Object Management Group* (OMG) sebagai standar pada November 1997. OMG memiliki tanggung jawab untuk pengembangan lebih lanjut dari standar UML. Bahkan sebelum adopsi terakhir, sejumlah buku diterbitkan yang menguraikan hal-hal penting dari UML.

2.14 Usecase Diagram

Use case diagram dapat digunakan selama proses analisis untuk menangkap requirements sistem dan untuk memahami bagaimana sistem seharusnya bekerja. Selama tahap desain, *usecase* diagram menetapkan perilaku (*behavior*) sistem saat diimplementasikan. Dalam sebuah model mungkin terdapat satu atau beberapa *usecase* diagram. Tabel simbol *usecase* diagram dapat dilihat pada Tabel 2.3. Simbol *Usecase* Diagram.

Tabel 2.3 Simbol Usecase Diagram

No.	Simbol	Nama	Keterangan
1		*Actor*	Merupakan orang, proses atau *system* lain yang berinteraksi dengan *system* yang akan dibuat. Jadi walaupun simbol aktor dalam diagram *usecase* berbentuk orang, namun aktor belum tentu orang.
2		*Use case*	Merupakan fungsionalitas yang disediakan sistem sebagai unit-unit yang saling berinteraksi atau bertukar pesan antar unit maupun aktor.

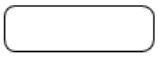
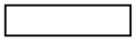
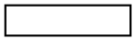
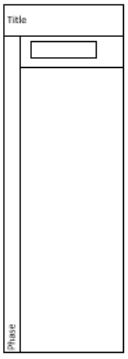
3		*Association*	Merupakan relasi yang digunakan untuk menggambarkan interaksi antara *usecase* dan aktor. Asosiasi juga menggambarkan berapa banyak objek lain yang bisa berinteraksi dengan suatu objek atau disebut *multiplicity*.
4		*Extend*	Menghubungkan antara satu objek dengan objek lain.
5		*Generalization*	Hubungan dimana objek berbagai perilaku dan struktur data dari objek yang ada di atasnya objek induk.
6		*Include*	Menspesifikasi bahwa *usecase* sumber secara eksplisit.
7		*Dependency*	Hubungan dimana perubahan yang terjadi pada suatu elemen mandiri akan mempengaruhi elemen yang bergantung padanya elemen yang tidak mandiri.
8		*System Boundary*	Menspesifikasikan paket yang menampilkan sistem secara terbatas.

2.15 Activity Diagram

Activity diagram memodelkan alur kerja (*work flow*) sebuah proses bisnis dan urutan aktivitas dalam suatu proses. Diagram ini sangat mirip dengan sebuah flowchart, karena user dapat memodelkan sebuah alur kerja dari satu aktivitas ke aktivitas lainnya atau dari satu aktivitas ke dalam keadaan sesaat (*state*). Diagram

Activity berfokus pada aktifitas-aktifitas yang terjadi yang terkait dalam suatu proses tunggal. Jadi dengan kata lain, diagram ini menunjukkan bagaimana aktifitas-aktifitas tersebut bergantung satu sama lain. Tabel simbol *activity* diagram dapat dilihat pada Tabel 2.4 Simbol *Activity* Diagram.

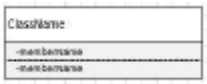
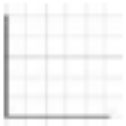
Tabel 2.4 Simbol Activity Diagram

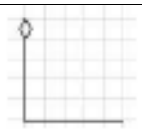
No.	Simbol	Nama	Keterangan
1.		*Action*	*State* dari sistem yang mencerminkan suatu aksi.
2.		*Decision*	Digunakan untuk menggambarkan suatu pengambilan keputusan / tindakan pada kondisi tertentu.
3.		*Initial Node*	Menunjukkan dimulainya suatu aktifitas.
4.		*Final Node*	Menunjukkan akhir dari aktifitas.
5.		*Fork Node*	Digunakan untuk menunjukkan aktifitas yang dilakukan secara paralel.
6.		*Join Node*	Digunakan untuk menunjukkan aktifitas yang digabungkan.
7.		*Swimlane*	Memisahkan organisasi bisnis yang bertanggung jawab terhadap aktifitas yang terjadi.

2.16 Class Diagram

Class diagram memberikan pandangan secara luas dari suatu sistem dengan menunjukan kelas-kelasnya dan hubungan mereka. *Class* diagram bersifat statis, menggambarkan hubungan apa yang terjadi bukan apa yang terjadi jika mereka berhubungan. Class menggambarkan keadaan (*atribut*/properti). *Class* diagram menggambarkan struktur dan deskripsi *class*, *package*, atau objek berserta hubungan satu sama lain seperti pewarisan, asosiasi, dan lain-lain. Tabel simbol *class* diagram dapat dilihat pada Tabel 2.5 Simbol *Class* Diagram.

Tabel 2.5 Simbol Class Diagram

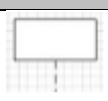
No	Simbol	Nama	Keterangan
1.		*Class*	Blok pembangunan pada pemrograman berorientasi objek. Bagian atas adalah bagian dari class. Bagian tengah mendefinisikan *property*/*atribut class*. Bagian akhir mendefinisikan *method* -*method* dari sebuah class.
2.		*Association*	*Relationship* paling umum antara 2 *class*, dan dilambangkan oleh sebuah garis yang menghubungkan antara 2 class. Garis ini bisa melambangkan tipe-tipe *relationship* dan juga dapatmenampilkan hukum-hukum multiplisitas pada sebuah relasi.
3.		*Compisition*	Jika sebuah *class* tidak bisa berdiri sendiri dan harus merupakan bagian dari *class* yang lain, maka *class* tersebut memiliki relasi *composition* terhadap *class* tempat bergantung

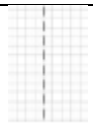
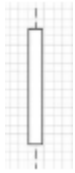
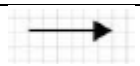
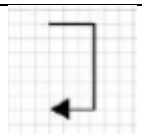
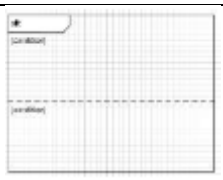
			tersebut.
4.		*Dependency*	Hubungan dimana perubahan yang terjadi pada suatu elemen mandiri (*independent*) akan mempengaruhi elemen yang bergantung pada elemen yang tidak mandiri (*independent*).
5.		*Aggregation*	Mengindikasikan keseluruhan bagian *relationship* dan biasanya disebut sebagai relasi.
6.		*Directed Association*	Relasi antar kelas dengan makna kelas yang satu digunakan oleh kelas yang lain, asosiasi biasanya disertai *multiplicity*

2.17 Sequence Diagram

Sequence Diagram digunakan untuk menunjukan aliran fungsionalitas dalam *use case*. *Sequence Diagram* merupakan salah satu diagram *Interaction* yang menjelaskan bagaimana suatu operasi itu dilakukan; message (pesan) apa yang dikirim dan kapan pelaksanaannya. Diagram ini diatur berdasarkan waktu. Obyek obyek yang berkaitan dengan proses berjalannya operasi diurutkan dari kiri ke kanan berdasarkan waktu terjadinya dalam pesan yang terurut. Tabel simbol sequenece diagram dapat dilihat pada Tabel 2.6 Simbol *Sequence* Diagram.

Tabel 2.6 Simbol Sequence Diagram

o.	**Simbol**	**Nama**	**Keterangan**
2.		partisipan	Merupakan instance dari sebuah class dan dituliskan tersusun secara horizontal.

3.		*Lifeline*	Mengindikasikan keberadaan sebuah objek dalam basis waktu.
4.		*Activation*	Mengindikasikan sebuah objek yang akan melakukan sebuah aksi.
5.		*Message*	Mengindikasikan komunikasi antar *obejct*.
6.		*Self-Message*	Mengindikasikan komunikasi kembali kedalam sebuah objek itu sendiri.
7.		*Sequence fragment*	Membungkus sebagian interaksi dengan diagram urutan yang memiliki beberapa tipe seperti *ref, assert, loop, break, alt, opt, neg, par, region*

.