ITS
Institut
Teknologi
Sepuluh Nopember

TUGAS AKHIR – KI1502

POLA PENCOCOKAN WORKFLOW UNTUK MEMULIHKAN ANOMALI PADA PROSES BISNIS

RIZAL SEPTIARAKHMAN
NRP 5114 100 180

Dosen Pembimbing I
Prof. Drs. Ec. Ir. Riyanarto Sarno, M.Sc.,Ph.D

Dosen Pembimbing II
Dwi Sunaryono, S.Kom.,M.Kom.

DEPARTEMEN INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya 2018

[Halaman ini sengaja dikosongkan]

TUGAS AKHIR – KI1502

POLA PENCOCOKAN WORKFLOW UNTUK MEMULIHKAN ANOMALI PADA PROSES BISNIS

RIZAL SEPTIARAKHMAN
NRP 5114 100 180

Dosen Pembimbing I
Prof. Drs. Ec. Ir. Riyanarto Sarno, M.Sc.,Ph.D.

Dosen Pembimbing II
Dwi Sunaryono, S.Kom.,M.Kom.

DEPARTEMEN INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya 2018

[Halaman ini sengaja dikosongkan]

FINAL PROJECT – KI1502

WORKFLOW PATTERN MATCHING FOR RECOVERING BUSINESS PROCESS ANOMALY

RIZAL SEPTIARAKHMAN
NRP 5114 100 180

Supervisor I
Prof. Drs. Ec. Ir. Riyanarto Sarno, M.Sc.,Ph.D.

Supervisor II
Dwi Sunaryono, S.Kom.,M.Kom.

DEPARTMENT OF INFORMATICS
Faculty of Information and Communication Technology
Institut Teknologi Sepuluh Nopember
Surabaya 2018

[Halaman ini sengaja dikosongkan]

LEMBAR PENGESAHAN

POLA PENCOCOKAN WORKFLOW UNTUK MEMULIHKAN ANOMALI PADA PROSES BISNIS

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Rumpun Mata Kuliah Manajemen Informasi
Sistem Studi S-1 Departemen Informatika
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember

Oleh
RIZAL SEPTIARAKHMAN
NRP. 5114 100 180

Disetujui oleh Dosen Pembimbing Tugas Akhir:

Prof. Drs. Ec. Ir. RIYANARTO SARNO, M.Sc., Ph.D.
NIP: 19590803 198601 1 001 ..... (pembimbing 1)

DWI SUNARYONO, S.Kom.,M.Kom.
NIP: 19860823 201504 1 004 ..... (pembimbing 2)

SURABAYA
JANUARI, 2018

[Halaman ini sengaja dikosongkan]

POLA PENCOCOKAN WORKFLOW UNTUK MEMULIHKAN ANOMALI PADA PROSES BISNIS

Nama : **Rizal Septiarakhman**
NRP : **5114100180**
Departemen : **Informatika – FTIK**
Dosen Pembimbing I : **Prof. Drs. Ec. Ir. Riyanarto Sarno, M.Sc.,Ph.D.**
Dosen Pembimbing II : **Dwi Sunaryono, S.Kom.,M.Kom.**

Abstrak

Belakangan ini, banyak perusahaan yang membutuhkan program untuk memodelkan log data mereka. Dikarenakan ini dapat memiliki banyak keuntungan salah satunya untuk mengecek error dari tiap aktivitas yang ada pada log data. Masalahnya log data biasanya sangatlah besar dan tidak dapat dimodelkan manual karena akan sangat membuang waktu dan energi. Maka dari itu, tugas akhir ini ditujukan untuk memunculkan model bisnis proses dari Linear Temporal Logic dan membandingkannya dengan model yang diciptakan langsung dari event log menggunakan metode Pattern Matching. Tugas akhir kali ini menunjukkan kelanjutan dari algoritma declarative miner, untuk membuat pola dalam Linear Temporal Logic (LTL) model dari log data yang dimiliki perusahaan, memiliki performa yang sangat bagus. Itu dibuktikan dari besarnya akurasi perbandingan antara LTL dan Event Log dengan menggunakan Neo4j.

Kata kunci: ***Control Flow Pattern, Neo4j, Linear Temporal Logic, Graph Database, Pattern Matching*****.**

[Halaman ini sengaja dikosongkan]

WORKFLOW PATTERN MATCHING FOR RECOVERING BUSINESS PROCESS ANOMALY

Student Name	**: Rizal Septiarakhman**
NRP	**: 5114100180**
Major	**: Informatics – FTIK**
Supervisor I	**: Prof. Drs. Ec. Ir. Riyanarto Sarno, M.Sc.,Ph.D.**
Supervisor II	**: Dwi Sunaryono, S.Kom.,M.Kom.**

Abstract

Nowadays, many companies need a program to model their data logs. As this can have many benefits such as to check the errors of every activity in the data log. The problem is log data is usually very large that it can't be modelled manually because it will be such a waste of time and energy. Therefore, this research generate a business process model from Linear Temporal Logic and compare it to model generated directly from event log using Pattern Matching. This research shows that an extended algorithm of declarative miner, to create pattern in Linear Temporal Logic (LTL) rule model from Log Data owned by the company, has a very good performance. It is proven by the high accuracy of the matching of LTL and event log using graph in Neo4j.

Keywords: ***Control Flow Pattern, Neo4J, Linear Temporal Logic, Graph Database, Pattern Matching*****.**

[Halaman ini sengaja dikosongkan]

KATA PENGANTAR

Segala puji syukur penulis kepada Allah SWT atas segala nikmat dan karunia-Nya, sehingga tugas akhir berjudul "Pola Pencocokan Workflow untuk Memulihkan Anomali Pada Proses Bisnis" ini dapat selesai sesuai dengan waktu yang telah ditentukan.

Pengerjaan tugas akhir ini menjadi sebuah sarana untuk penulis memperdalam ilmu yang telah didapatkan di Institut Teknologi Sepuluh Nopember Surabaya, khususnya dalam disiplin ilmu Teknik Informatika. Terselesaikannya buku tugas akhir ini tidak terlepas dari bantuan dan dukungan semua pihak. Pada kesempatan kali ini penulis ingin mengucapkan terima kasih kepada:

1. Mama, papa dan keluarga yang selalu memberikan dukungan berupa doa dan nutrisi selama proses pengerjaan Tugas Akhir.
2. Bapak Riyanarto Sarno dan Bapak Dwi selaku dosen pembimbing yang telah bersedia meluangkan waktu selama proses pengerjaan Tugas Akhir.
3. Bapak dan Ibu dosen Jurusan Teknik Informatika ITS yang banyak memberikan ilmu dan bimbingan bagi penulis.
4. Seluruh staf dan karyawan FTIf ITS yang banyak memberikan kelancaran administrasi akademik kepada penulis.
5. Teman-teman 'Rantau' dan TC Angkatan 2014 yang selalu mendukung selama proses pengerjaan tugas akhir.
6. Serta semua pihak yang turut membantu penulis dalam menyelesaikan tugas akhir ini.

Penulis menyadari bahwa tugas akhir ini masih memiliki banyak sekali kekurangan. Dengan kerendahan hati, penulis mengharapkan kritik dan saran dari pembaca untuk perbaikan ke depan.

Surabaya, Januari 2018

[Halaman ini sengaja dikosongkan]

DAFTAR ISI

[Halaman ini sengaja dikosongkan]

DAFTAR GAMBAR

[Halaman ini sengaja dikosongkan]

DAFTAR TABEL

[Halaman ini sengaja dikosongkan]

DAFTAR KODE SUMBER

[Halaman ini sengaja dikosongkan]

DAFTAR SINGKATAN

LTL	:	*Linear Temporal Logic.*
CSV	:	*Comma Separated Value*.
ProM	:	*Process Mining.*

[Halaman ini sengaja dikosongkan]

BAB I
PENDAHULUAN

Pada bab ini akan dipaparkan mengenai garis besar Tugas Akhir yang meliputi latar belakang, tujuan, rumusan dan batasan permasalahan, metodologi pengerjaan Tugas Akhir, dan sistematika penulisan Tugas Akhir.

1.1 Latar Belakang

Log data adalah yang berisi informasi mengenai bisnis proses yang berjalan [1]. Log data ini akan digunakan oleh algoritma-algoritma pembentukan model proses untuk menghasilkan model proses secara langsung. Algoritma-algoritma pembentukan model proses akan berjalan maksimal apabila log data yang diolah adalah log data yang lengkap. Pada kenyataanya, tidak semua log data menyimpan proses yang lengkap [1]. Pada Terminal Peti Kemas, terdapat puluhan kontainer yang diperiksa dimana setiap pemeriksaan membutuhkan waktu lebih dari satu hari. Hal ini memunculkan adanya proses-proses yang keseluruhan aktivitas belum selesai dilaksanakan pada saat algoritma pembentukan model proses dijalankan pada rentang waktu tertentu. Proses tersebut disebut sebagai proses yang terpotong. Proses yang terpotong dapat memicu hasil yang salah apabila diolah secara langsung oleh algoritma pembentukan model proses.

Untuk mengatasi permasalahan tersebut, maka dibutuhkan perbaikan proses yang terpotong. Terdapat beberapa cara dalam perbaikan proses, yaitu menggunakan pendekatan heuristik (percobaan berdasarkan data yang ada untuk mencari hasil yang mendekati solusi terbaik) serta membangun proses model sebagai acuan perbaikan. Pendekatan heuristik bisa dilakukan dengan cara mencari relasi aktivitas yang memiliki probabilitas tertinggi atau mencari jumlah aktivitas terkecil sebagai relasi perbaikan proses yang terpotong. Pendekatan heuristik tidak dapat dipilih karena pendekatan ini tidak melihat keterkaitan antar aktivitas, dimana

hal tersebut diperlukan pada kasus *non-free choice*. Oleh karena itu, penelitian ini akan menggunakan model proses sebagai acuan perbaikan proses yang terpotong.

Beberapa algoritma perbaikan proses tidak lengkap telah diusulkan, yaitu *Heuristic Linear Approach* [2] dan algoritma *Repair Logs* [2]. Kedua algoritma tersebut menggunakan kemungkinan rangkaian proses yang diambil dari model proses untuk menentukan aktivitas pengganti dari aktivitas yang hilang. Kedua metode ini tidak memiliki algoritma pembentukan model proses, sehingga metode ini tidak dapat digunakan apabila model proses tidak ada. Dalam kasus Terminal Peti Kemas, data yang didapatkan adalah log data, tanpa model proses rinci kegiatan Terminal Peti Kemas. Kemudian, kedua algoritma perbaikan tersebut tidak membedakan proses yang akan diperbaiki. Akan tetapi, pada kenyataannya terdapat beberapa proses yang memiliki ciri seperti proses yang terpotong, akan tetapi proses tersebut adalah anomali. Anomali tersebut adalah *skip sequence* dan *wrong pattern*. *Skip sequence* adalah proses yang memiliki aktivitas-aktivitas hilang di tengah-tengah proses. Sedangkan, *wrong pattern* adalah proses yang memiliki aktivitas yang berjalan tidak sesuai proses seharusnya. Apabila algoritma pendeteksian hanya berfokus pada aktivitas awal dan akhir proses, maka kedua anomali ini akan terdeteksi sebagai proses yang terpotong. Oleh karena itu, diperlukan algoritma pembentukan model proses dalam hal perbaikan proses yang terpotong serta pembedaan proses yang tidak lengkap, dikarenakan proses yang terpotong atau anomali.

Dalam algoritma pembentukan model proses, terdapat dua hal yang menjadi perhatian, yaitu *non-free choice* dan pengunaan invisible task. *Non-free choice* adalah keterkaitan aktivitas pada relasi pilihan dengan aktivitas pada relasi pilihan lainnya. Sedangkan, pengunaan invisible task pada beberapa model proses untuk menggambarkan adanya kondisi *skip* atau relasi paralel yang bertumpuk. Untuk memberikan hasil yang terbaik, maka

algoritma pembentukan model proses harus memperhatikan kedua hal tersebut.

Penelitian ini fokus pada pembuatan metode untuk perbaikan proses yang terpotong pada log data yang mengandung *invisible tasks* yang bertipe redo dengan membentuk model proses acuan. Berbagai macam bentuk model proses, seperti YAWL, Petri Net, dan BPMN. Model-model proses tersebut memiliki bentuk yang berbeda-beda, dimana bagi orang yang awam akan kesulitan untuk memahami proses yang tergambar dalam model proses tersebut. Untuk meminimalkan ambiguitas, bahasa formal diperlukan dalam penggambaran proses. Dalam hal ini, Linear Temporal Logic (LTL) [1] dipilih karena ia merupakan bahasa formal yang menggambarkan relasi aktivitas berkaitan dengan waktu. Waktu yang dimaksud disini adalah waktu pelaksanaan aktivitas terhadap aktivitas lain, seperti suatu aktivitas dijalankan tepat setelah aktivitas lain selesai dijalankan atau suatu aktivitas dijalankan kapan saja, dengan ketentuan setelah suatu aktivitas dijalankan.

Agar suatu model dapat digunakan kembali atau dapat didaur ulang dalam membentuk model baru, maka dibentuklah *pattern* (potongan model proses) dari log data yang ada. Berbagai macam *pattern* model proses telah didefinisikan [1], dimana *pattern* tersebut dibentuk dalam model Petri Net. Penelitian ini akan menerapkan LTL untuk menggambarkan *pattern* tersebut, dimana kumpulan *pattern* ini akan dibentuk dalam model *tree*. Model *tree* dipilih karena mampu merepresentasikan *logical operator* seperti yang ada pada LTL (Linear Temporal Logic) sehingga mempermudah penggabungan *control-flow pattern* dalam bentuk LTL. Hasil dari LTL tersebut kemudian akan dimodelkan kebentuk graf dengan menggunakan applikasi Neo4j yang merupakan applikasi berbasis bahasa cipher.

Kinerja metode yang diusulkan berupa model graf pada Neo4j yang dihasilkan melalui proses Declarative Miner akan dibandingkan dengan model graf yang terbentuk dari log data yang langsung diolah menggunakan bahasa cipher. Perbandingan

menggunakan algoritma Pattern Matching pada Neo4j dengan bantuan APOC dan algoritma Brute Force pada Java.

1.2 Rumusan Permasalahan

Rumusan masalah yang diangkat dalam Tugas Akhir ini dapat dipaparkan sebagai berikut:

1. Bagaimana cara mengimplementasikan data non-relational dalam Neo4j?
2. Bagaimana mengimplementasikan Pattern Matching untuk membandingkan model bisnis proses?
3. Bagaimana cara membuat model bisnis proses pada Neo4j?
4. Bagaimana menentukan relasi antara XOR Split dan XOR Join?
5. Bagaimana membentuk *pattern* dalam bentuk LTL (Linear Temporal Logic) berdasarkan log data?

1.3 Batasan Permasalahan

Permasalahan yang dibahas dalam Tugas Akhir ini memiliki beberapa batasan, di antaranya sebagai berikut:

1. lmplementasi model bisnis proses dengan database Neo4j.
2. Data masukan berupa *event log* dalam bentuk Excel yang diproses menggunakan *declarative miner* pada Java sehingga membentuk suatu model.
3. Hasil dari Pattern Matching berupa persentase kesamaan antar model.
4. Metode perbaikan proses yang terpotong belum berjalan dengan baik apabila log data yang didapatkan memiliki variasi proses yang sedikit. Sebagai contoh, metode perbaikan proses yang terpotong berjalan sempurna pada relasi XOR yang melibatkan 3 aktivitas apabila terdapat tiga variasi proses yang terekam di log data.

1.4 Tujuan

Tujuan dari Tugas Akhir ini diharapkan dapat membentuk model proses dari kumpulan *control-flow patterns* yang dibentuk

dari model deklaratif dan log data kemudian membandingkannya. Sehingga menghasilkan output yang menunjukkan bahwa antar model tersebut terjadi sebuah anomali atau tidak.

1.5 Manfaat

Penelitian ini diharapkan dapat bermanfaat bagi masyarakat luas terutama pada perusahaan-perusahaan yang ingin merancangkan model bisnis proses dari sebuah log data yang berukuran sangat besar. Model bisnis proses yang telah diciptakan diharapkan dapat mengurangi error pada pengecekan aktivitas tiap log.

1.6 Metodologi

Langkah-langkah yang ditempuh dalam pengerjaan Tugas Akhir ini yaitu sebagai berikut:

1. **Studi literatur**

   Dalam pembuatan Tugas Akhir ini telah dipelajari tentang hal-hal yang dibutuhkan sebagai ilmu penunjang dalam penyelesaiannya. Ilmu penunjang utama pada Tugas Akhir ini adalah permodelan menggunakan Neo4j, Declarative Miner. Selain itu, terdapat literatur lain yang menunjang proses penyelesaian Tugas Akhir ini.

2. **Pembuatan Metode**

   Pada tahap ini penulis menjabarkan cara pemecahan masalah yang terdapat dalam rumusan masalah.

3. **Analisis dan Perancangan Algoritma**

   Tahap ini meliputi perancangan algoritma berdasarkan studi literatur dan pembelajaran konsep teknologi dari perangkat lunak yang ada. Tahap ini mendefinisikan alur dari implementasi. Langkah-langkah yang dikerjakan juga didefinisikan pada tahap ini.

4. **Implementasi**

   Pada tahap ini dilakukan pembuatan perangkat lunak yang menghasilkan dua graf dengan menggunakan neo4j yang terbentuk dari hasil declarative miner dan input langsung dari event log, kemudian membandingkan kedua hasil tersebut.

5. **Pengujian dan evaluasi**

   Pada tahap ini dilakukan pengujian terhadap elemen perangkat lunak dengan menggunakan data uji yang telah dipersiapkan. Pengujian dan evaluasi perangkat lunak dilakukan untuk mengevaluasi jalannya perangkat lunak, mengevaluasi fitur utama, mengevaluasi fitur-fitur tambahan, mencari kesalahan yang timbul pada saat perangkat lunak aktif, dan mengadakan perbaikan jika ada kekurangan. Tahapan-tahapan dari pengujian adalah sebagai berikut:

   a. Model proses yang dihasilkan dari declarative miner diharuskan memiliki relasi yang sama sesuai hasil LTL.
   b. Hasil permodelan kedua graf dalam hal kesamaan node dan relasi.

6. **Penyusunan buku Tugas Akhir**

   Pada tahap ini dilakukan pendokumentasian dan pelaporan dari seluruh konsep, dasar teori, implementasi, proses yang telah dilakukan, dan hasil-hasil yang telah didapatkan selama pengerjaan Tugas Akhir.

1.7 Sistematika Penulisan

Buku Tugas Akhir ini bertujuan untuk mendapatkan gambaran dari pengerjaan Tugas Akhir ini. Selain itu, diharapkan dapat berguna untuk pembaca yang tertarik untuk melakukan

pengembangan lebih lanjut. Secara garis besar, buku Tugas Akhir terdiri atas beberapa bagian seperti berikut ini.

Bab I Pendahuluan

Bab ini berisi latar belakang masalah, tujuan dan manfaat pembuatan Tugas Akhir, permasalahan, batasan masalah, metodologi yang digunakan, dan sistematika penyusunan Tugas Akhir.

Bab II Dasar Teori

Bab ini membahas beberapa teori penunjang yang berhubungan dengan pokok pembahasan dan yang menjadi dasar dari pembuatan Tugas Akhir ini.

Bab III Metode Pemecahan Masalah

Bab ini membahas cara penulis memecahkan masalah yang ada. Penjelasan tentang algoritma yang dikembangkan penulis dan langkah-langkahnya sehingga dapat memecahkan masalah yang ada.

Bab IV Analisis dan Perancangan Algoritma

Bab ini membahas mengenai perancangan algoritma. Perancangan algoritma meliputi perancangan alur, dan perancangan proses.

Bab V Implementasi

Bab ini berisi implementasi dari perancangan perangkat lunak perangkat lunak dan implementasi fitur-fitur penunjang perangkat lunak.

Bab VI Pengujian dan Evaluasi

Bab ini membahas pengujian dengan metode pengujian subjektif untuk mengetahui penilaian aspek kegunaan (*usability*) dari perangkat lunak dan pengujian fungsionalitas yang dibuat dengan memperhatikan keluaran yang dihasilkan serta evaluasi terhadap fitur-fitur perangkat lunak.

Bab VII Kesimpulan

Bab ini berisi kesimpulan dari hasil pengujian yang dilakukan. Bab ini membahas saran-saran untuk pengembangan sistem lebih lanjut.

Daftar Pustaka

Merupakan daftar referensi yang digunakan untuk mengembangkan Tugas Akhir.

Lampiran

Merupakan lampiran yang digunakan untuk menjabarkan hasil pengujian algoritma.

BAB II
DASAR TEORI

Pada bab ini akan dibahas mengenai teori-teori yang menjadi dasar dari pembuatan Tugas Akhir.

2.1 *Process Discovery*

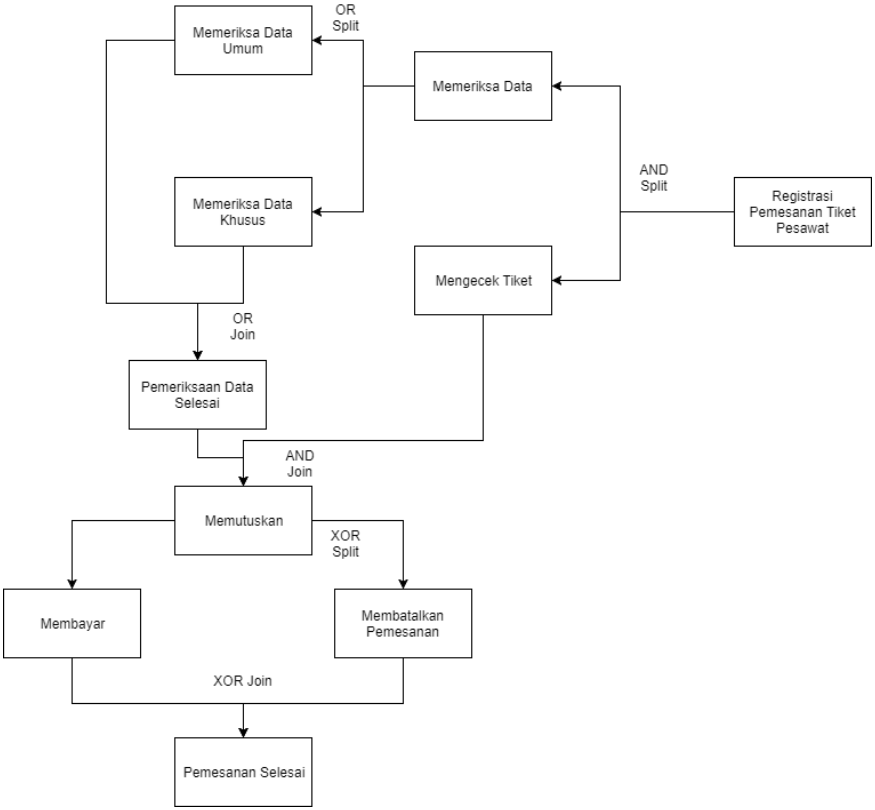
Proses mining adalah suatu ilmu yang menggabungkan antara pembelajaran mesin serta *data mining* pada satu sisi dan memodelkan serta menganalisa proses pada sisi yang lain. Ide dasar dari *process mining* adalah menemukan, memantau serta meningkatkan proses sebenarnya dengan mengambil pengetahuan dari *event log* yang ada pada suatu sistem [1].

Proses mining dibagi menjadi tiga bagian, salah satunya adalah *process discovery*. *Process discovery* adalah suatu teknik yang memanfaatkan *event log* dalam penentuan proses model tanpa menggunakan informasi-informasi lainnya. *Process discovery* yang ada diantaranya algoritma *Alpha#* [2] serta *Heuristics Miner* [3].

Proses model yang benar adalah proses model yang mampu menampilkan konkurensi serta *control-flow pattern* [1]. Konkurensi berarti aktivitas pada proses model tidak ditampilkan secara redundan (tidak ada aktivitas yang sama ditampilkan pada proses model) [1]. *Control-flow pattern* adalah penanda relasi antar aktivitas. Terdapat empat relasi antar aktivitas yaitu relasi *sequence*, relasi XOR, relasi AND and relasi OR [1].

Relasi *sequence* adalah relasi yang terjadi untuk menghubungkan satu aktivitas dengan satu aktivitas lainnya. Relasi XOR adalah relasi yang terjadi yang hanya memperbolehkan salah satu aktivitas yang terjadi pada suatu proses. Relasi OR adalah relasi yang terjadi apabila beberapa aktivitas yang terjadi pada suatu proses. Sedangkan relasi AND adalah relasi yang terjadi dimana semua aktivitas harus dijalankan semuannya. Apabila aktivitas pilihan pada relasi XOR, OR dan AND adalah aktivitas sebelum dari aktivitas lainnya, maka *control-flow pattern* dari relasi tersebut adalah penambahan kata “Split” pada relasi tersebut [1]. Sedangkan,

apabila aktivitas pilihan pada relasi XOR, OR dan AND adalah aktivitas setelah dari aktivitas lainnya, maka *control-flow pattern* dari relasi tersebut adalah penambahan kata “Join” pada relasi tersebut [1]. Gambar 2.1 merupakan contoh penggunaan relasi pada proses model.



Gambar 2.1 Contoh Model Proses

2.2 *Streaming Event Log*

Event log merupakan catatan kejadian dari suatu proses bisnis yang berjalan [1]. Sebuah kejadian menunjukkan aktivitas ataupun langkah dari suatu proses. *Event log* menunjukkan proses tunggal sehingga sebuah kejadian termasuk dalam sebuah proses, yang biasa disebut dengan *case* [1]. *Event log* dapat menyimpan informasi yang berhubungan dengan kejadian, seperti *timestamp* atau *resources* [1]. *Timestamp* adalah waktu terjadinya suatu kejadian. Sedangkan *resources* adalah pelaku yang mengeksekusi suatu kejadian. Definisi lengkap dari *event log* dapat dilihat pada Gambar 2.2 [1].

Definisi *Event log*

Event log terdiri dari rangkaian aktivitas dan waktu yang didefinisikan dengan $L_{A,TD} = (E, C, \alpha, \gamma, \beta, \succ)$ dimana:

- E adalah kumpulan kejadian.
- C adalah kumpulan case.
- $\alpha: E \rightarrow A$ adalah fungsi yang menghubungkan setiap kejadian dengan sebuah aktifitas.
- $\gamma: E \rightarrow TD$ adalah fungsi yang menghubungkan setiap kejadian dengan sebuah waktu kejadian (*timestamp*).
- $\beta: E \rightarrow C$ adalah fungsi yang menghubungkan sebuah kejadian dengan sebuah case.
- $\succ \subseteq ExE$ adalah *relation succesion*, yang merupakan total suatu kejadian yang dilanjutkan dalam kejadian lain dan termasuk dalam E. $e_2 \succ e_1$ adalah notasi singkat untuk $(e_2, e_1) \in \succ$. Kumpulan kejadian yang berkaitan dalam sebuah case disebut sebagai *trace*.

Gambar 2.2 Definisi *Event Log*

Sedangkan *streaming event log* merupakan *event log* yang diobservasi satu per satu berdasarkan waktu dilaksanakannya kejadian pada *event log* tersebut [6]. Sehingga, *streaming event log* adalah *event log* yang tercatat secara *real-time*.

2.3 *NoSQL*

No Structured Query Language merupakan istilah yang digunakan pertama kali oleh Carlo Strozzi pada tahun 1998 untuk basis data relasional Strozzi NoSQL opensource-nya. Database ini bersifat relasional, namun istilahnya telah berevolusi dan sejak 2009 lebih dikenal dengan database non-relasional atau non-ACID. Database NoSQL bervariasi dari toko dengan nilai kunci sederhana hingga sistem seperti SQL yang kompleks.

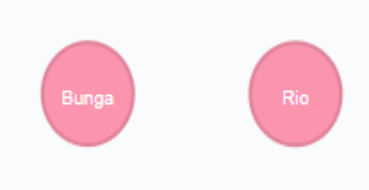
2.4 *NEO4j*

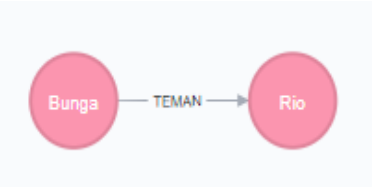
Neo4j adalah aplikasi gratis NoSQL graf database yang di implementasikan di java dan scala dengan pembuatan mulai 2003, dan telah tersedia untuk umum sejak 2007. Neo4j sekarang digunakan oleh ratusan ribu perusahaan dan organisasi di hampir semua industri. Use case yang meliputi matchmaking, manajemen jaringan, analisis perangkat lunak, penelitian ilmiah, routing, manajemen organisasi dan proyek, rekomendasi, jejaring sosial dan lainnya. Neo4j menerapkan model grafik properti secara efisien dalam tingkat penyimpanan. Dikarenakan Neo4j merupakan salah satu NoSQL database, Neo4j memiliki beberapa keunggulan dibandingkan dengan database SQL pada umumnya. Tabel X menjelaskan kelebihan Neo4j dibanding SQL.

Neo4j dapat menyimpan 2 hal yang terdiri dari: *Node Label* dan *Relatonship Types*. Node merupakan kumpulan data yang berisi informasi-informasi yang tersimpan dalam sebuah data. Contoh: Sebuah database Person yang menampung informasi seperti Nama dan Asal. Sedangkan Relationship merupakan kumpulan data yang berisi hubungan / relasi antar node. Contoh: dari Gambar 2.3 dibuat sebuah relasi yang menyatakan Bunga dan Rio adalah teman. Gambar 2.4 menunjukan relasi tersebut.

Tabel 2.1 Perbandingan Neo4j dengan SQL Database

Pembanding	Neo4j	SQL Database
Waktu yang diperlukan untuk menampilkan ~2500 data	0.01 detik	0.016 detik
Waktu yang diperlukan untuk menampilkan ~11.000 data	0.168 detik	30.267 detik
Fleksibilitas	Database fleksibel	Database bergantung dengan tabel yang dibuat
Visual	Memiliki tampilan	Hanya berbentuk tabel
Relasi antar data	Dapat membentuk relasi	Hanya dapat membentuk foreign key





Gambar 2.3 Contoh Node dalam *Neo4j* & Gambar 2.4 Contoh Relationship dalam *Neo4j*

2.5 *Declarative Miner*

Penemuan model deklaratif secara umum dan model Declare secara khusus dapat digunakan untuk secara efektif menyelesaikan dua kelemahan penting dari teknik penemuan proses yang ada:

- Model yang dihasilkan cenderung besar dan kompleks terutama di lingkungan yang fleksibel dimana eksekusi proses melibatkan banyak alternatif

- Menawarkan kemungkinan terbatas untuk memandu proses penambangan menuju sifat-sifat menarik yang spesifik

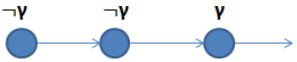
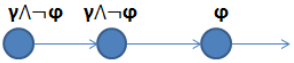
Dengan menggunakan model deklaratif, perilaku proses digambarkan sebagai seperangkat aturan yang harus dipenuhi selama proses eksekusi [6]. Penemuan model deklaratif dapat dengan mudah dipandu dalam kerangka aturan.

2.6 Linear Temporal Logic (LTL)

Linear Temporal Logic (LTL) (Raim, 2014) adalah bahasa yang menggambarkan logika temporal yang mengacu pada waktu. Linear Temporal Logic dibangun dari konstanta (benar dan salah), sekelompok proposisi atomic, operator logika ($\neg$, $\vee$, $\wedge$, $\rightarrow$) dan operator modal temporal. Terdapat empat operator modal temporal yang digunakan pada Linear Temporal Logic. Penjelasan mengenai operator tersebut dapat dilihat pada Tabel 2.2.

Pada penggalian proses, LTL dibentuk menjadi relasi aktivitas yang disebut *template* Declare (Fabrizio M. Maggi, Mooij, & Van Der Aalst, 2011). Contoh *template* declare dapat dilihat pada Tabel 2.3. *Template* ini yang digunakan oleh algoritma *Declarative Miner* (Fabrizio Maria Maggi, 2013) untuk membentuk model menggunakan log data. Model tersebut berupa graf yang memiliki relasi seperti pada *template* Declare. LTL dibutuhkan dikarenakan pada algoritman *Declarative Miner*, aktivitas pada event log akan dibentuk relasinya dengan sebuah *template* Declare yang kemudian akan menghasilkan sebuah logika dalam format LTL. Format LTL ini yang nantinya akan dibentuk sebuah model bisnis proses.

Tabel 2.2 Operator Model Temporal (LTL)

Tekstual	Simbol	Penjelasan	Diagram
Xγ	◯γ	γ harus dilaksanakan sebagai *state* selanjutnya	
Gγ	□γ	γ harus berada pada seluruh *state* berikutnya	
Fγ	◊γ	γ harus berada pada tempat lain di dalam *path*.	
γUφ	γ∪φ	γ harus dilaksanakan sampai φ muncul.	

Tabel 2.3 Potongan *Template* Declare

***Template* Declare**	**LTL**	**Deskripsi**
Init(K)	K	K adalah aktivitas pertama yang terekam dalam proses.
Chain Response(K,R)	□ (K → O (R))	Jika K terekam, maka aktivitas yang langsung terekam selanjutnya adalah R
Exclusive Choice (K,R)	(◊ (K ∨ R) ∧ ! (◊ (K ∧ R)))	K atau R akan terekam dalam proses, akan tetapi kedua aktivitas tersebut tidak dapat terekam pada proses yang sama.
Response(K,R)	□ (K → ◊ (R))	Aktivitas R hanya boleh terekam apabila aktivitas K sudah terekam dalam proses

Existences2(K)	◊ (K ∧ O (◊ (K)))	Aktivitas K muncul sekurang-kurangnya dua kali di proses

2.7 Pattern Matching

Dalam ilmu komputer, pencocokan pola adalah sebuah tindakan untuk memeriksa urutan suatu aktivitas yang diberikan dari beberapa pola. Berbeda dengan pengenalan pola *(Pattern Recognition)*, perbandingan pada pengenalan pola biasanya diharuskan tepat / persis [11]. Pola umumnya memiliki bentuk urutan atau struktur berupa pohon. Perbandingan yang dilakukan akan mencakup 3 hal:

- Node asal: merupakan sebuah node, dimana node berisi informasi tentang suatu aktivitas yang kemudian akan berhubungan dengan suatu aktivitas tujuan.
- Relasi: merupakan sebuah informasi yang menyimpan detail hubungan antar node asal dan node tujuan.
- Node tujuan: merupakan sebuah node, dimana node berisi informasi tentang suatu aktivitas yang terhubung dengan node asal melalui relasi tertentu.

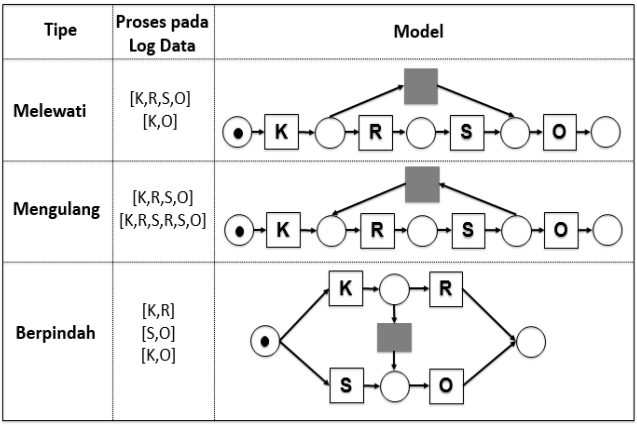
2.8 Anomali pada Bisnis Proses

Pada model bisnis proses terdapat proses-proses yang terpotong, dalam arti lain tidak terbentuk dengan baik. Hal tersebut merupakan sebuah anomali, salah satu jenis anomali adalah *skip sequence. Skip sequence* adalah proses yang memiliki aktivitas-aktivitas hilang di tengah-tengah proses. Hal tersebut dapat terjadi dikarenakan pada pembentukan model proses terbentuk sebuah *Invisible Task* (Aktivitas Tak Terlihat).

Invisible Task adalah aktivitas tambahan yang tidak muncul di log data tetapi ditampilkan dalam model proses (Wen, Wang, van der Aalst, Huang, & Sun, 2010). Kegunaan dari invisible task adalah untuk membantu menggambarkan proses secara sebenarnya dalam model proses. Invisible task dibagi menjadi dua macam, yaitu invisible prime task dan invisible non-prime task. Invisible prime

task adalah invisible task yang ditangani oleh algoritma Alpha# (Wen et al., 2010). Terdapat tiga macam invisible task prime yaitu Melewati, Mengulang, dan Berpindah.

Invisible task tipe Melewati digunakan untuk menggambarkan aktivitas yang dapat dilewati. Invisible task tipe Mengulang digunakan untuk menggambarkan aktivitas yang dapat diulang. Invisible task tipe Berpindah digunakan untuk perpindahan eksekusi pada beberapa percabangan [5]. Contoh dari Invisible task dalam bentuk Petri Net digambarkan pada Gambar 2.5. Kotak yang berwarna abu-abu merupakan contoh *Invisible task.*



Gambar 2.5 Contoh Anomali pada Bisnis Proses

[Halaman ini sengaja dikosongkan]

BAB III
METODE PEMECAHAN MASALAH

Pada bab ini akan dibahas mengenai metodologi pemecahan masalah yang digunakan sebagai dasar solusi dari pembuatan Tugas Akhir. Metodologi tersebut menerangkan langkah demi langkah proses hingga dapat menghasilkan proses model dari suatu *event log*.

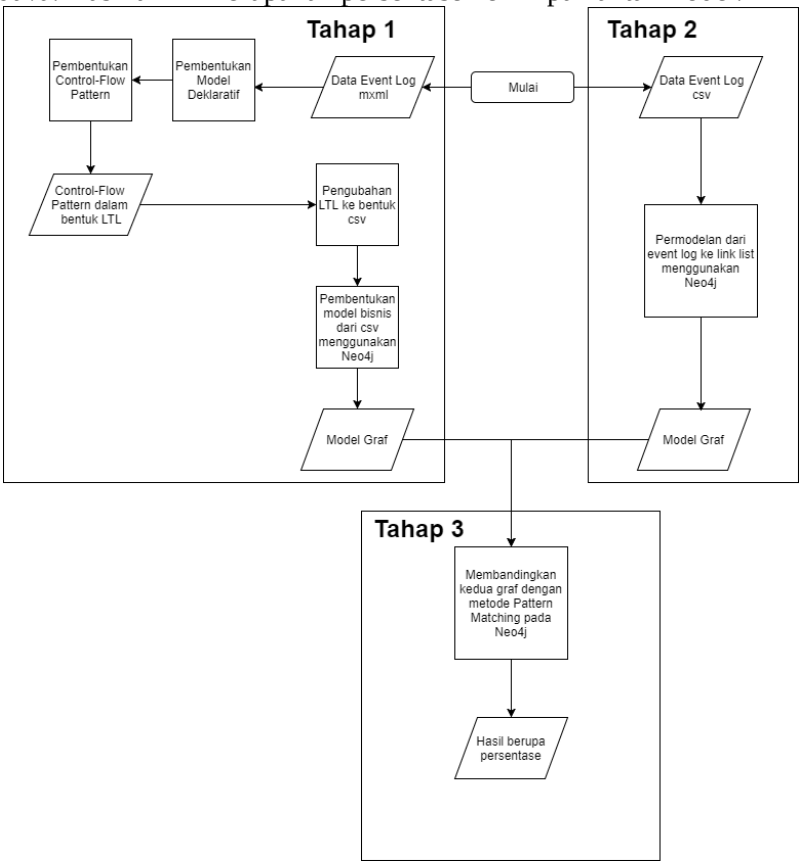
3.1 Cakupan Permasalahan

Permasalahan utama yang diangkat pada pembuatan Tugas Akhir ini adalah menemukan proses model yang tepat (*process discovery*) dari *streaming event log* yang memiliki proses yang terpotong yang merupakan sebuah anomali. Anomali yang dapat dipulihkan merupakan anomali *skip sequence* yang bertipe redo (mengulang).

Permasalahan pertama yang dipecahkan dalam Tugas Akhir ini adalah menggambarkan model proses beserta relasinya yang dihasilkan dari Declarative Miner. Permasalahan kedua yang dipecahkan pada Tugas Akhir ini adalah memodelkan proses beserta relasinya dengan input event log yang merupakan sebuah file dalam bentuk comma delimiter (csv). Permasalahan ketiga yang dipecahkan pada Tugas Akhir ini adalah *Pattern Matching*, yaitu membandingkan node dan relasi antar graf yang terbentuk dengan declarative miner dengan graf yang terbentuk langsung dari event log.

Gambar 3.1 adalah gambaran umum alur proses pengerjaan Tugas Akhir. Tahapan pertama adalah dari sebuah event log kemudian diubah menjadi mxml untuk masukkan dari *Declarative Miner*. Hal ini dilakukan untuk mendapatkan log data yang dapat diolah dalam algoritma pemodelan proses. Kemudian, log tersebut akan dioalah untuk didapatkan *control-flow patterns*. Pengolahan *control-flow patterns* melibatkan *rules* atau template pada model Deklaratif yang dihasilkan oleh algoritma *Declare Miner*. Kemudian, *control-flow patterns* yang telah terbentuk akan diubah

ke dalam bentuk csv, dan mengimportnya ke dalam Neo4j sehingga terbentuk model proses dalam bentuk graf. Tahapan yang kedua adalah memasukkan event log langsung ke dalam bentuk link list pada Neo4j, setelah itu baru dibentuk *control-flow patterns* dari model tersebut sehingga terbentuk suatu model proses. Tahap yang terakhir adalah membandingkan model bisnis proses antara model bisnis proses dari hasil *declarative* dan event log dengan bahasa Java. Hasil akhir merupakan persentase kemiripan antar model.



Gambar 3.1 Alur Proses Pengerjaan Tugas Akhir

3.2 Masukan

3.2.1 Data

Dalam penelitian ini, log data yang digunakan adalah log data mengenai proses impor barang di Terminal Peti Kemas Surabaya. Log data yang didapatkan merupakan hasil transformasi data proses impor barang mulai dari proses yang berjalan dari bulan Januari sampai proses yang berakhir di bulan Maret 2016. Log data yang digunakan mengandung ID_Case (nomor proses), nama aktivitas dan nama *message*, originator (pelaku aktivitas), sender reciever (pelaku *message*), waktu pelaksanaan, lampiran, detail lampiran, serta biaya pelaksanaan aktivitas. Tabel 3.1 merupakan potongan data aktivitas pada Terminal Peti Kemas. Untuk data keseluruhan dapat dilihat pada halaman Lampiran.

Tabel 3.1 Potongan Data Aktivitas Pada Terminal Peti Kemas

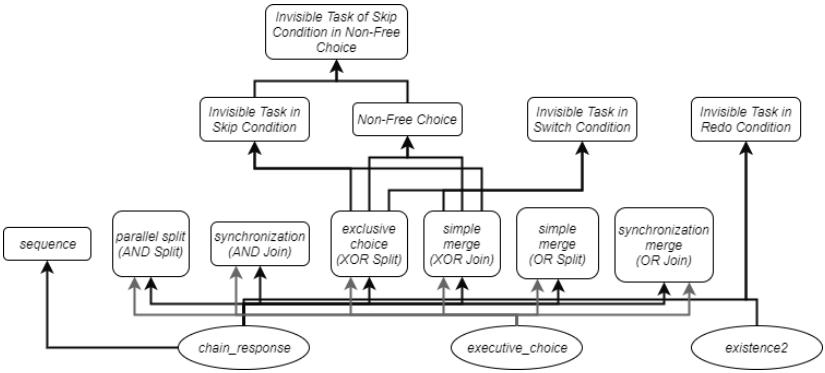
ID	Activity
1	{"Name":"Bring Back To Yard from Quarantine"}
2	{"Name":"Determine Category"}
3	{"Name":"Stack Container in Yard"}
4	{"Name":"Create SPPB"}
5	{"Name":"Create Job Order Document Delivery"}
6	{"Name":"Create document SPPB"}
7	{"Name":"Prepare Tools"}
8	{"Name":"Lift on Container Truck"}
9	{"Name":"Decide Task Before Lift Container"}
10	{"Name":"Unplug Reefer"}

3.3 Pembentukan Model Bisnis Proses

3.3.1 Pembentukan Model Bisnis Proses dari Model Deklaratif

Pada penggambaran *control-flow patterns*, model deklaratif yang digunakan menggunakan *rules* atau *template* berupa *chain_response, executive_choice, response,* dan *existence(2)*. Tidak semua *control-flow patterns* ditentukan hanya berdasarkan *rules*

pada model deklaratif, melainkan juga berdasarkan *control-flow patterns* lain yang sudah terbentuk. Gambar 3.2 menunjukkan alur pembentukan *control-flow patterns*. Rules *executive_choice* digunakan untuk menentukan relasi AND Split, AND Join, XOR Split, XOR Join, OR Split, OR Join. Rules *chain_response* digunakan untuk memodelkan semua yang dapat dimodelkan oleh *executive_choice* ditambah dengan relasi *sequence.* Rules *existence2* digunakan untuk menentukan *Redo Condition* dan juga *chain_response*. Hasil relasi XOR Join dan XOR Split kemudian digunakan untuk menentukan relasi *Invisible Task* dan *Non-Free Choice.*



Gambar 3.2 Alur Hirarki Pembentukan *Control-Flow Patterns*

Metode pembentukan *control-flow patterns* dapat dilihat pada Tabel 3.2. Di dalam metode, terdapat beberapa variabel tambahan. Variabel-variabel tersebut adalah total_after(act), total_before(act), total_ex_after(act), total_ex_before(act), total_chain_after(act), dan total_chain_before(act).

Total_next(act) adalah jumlah *chain_response*(act, aktivitas lainnya) yang memiliki nilai *support* lebih dari 0. Sedangkan total_before(act) adalah jumlah *chain_response*(aktivitas lainnya, act) yang memiliki nilai *support* lebih dari 0. Total_chain_before(act) adalah jumlah *chain_response*(aktivitas sebelum act, aktivitas sebelum act) yang memiliki nilai *support* lebih

dari 0. Total_chain_after(act) adalah jumlah *chain_response*(aktivitas setelah act, aktivitas setelah act) yang memiliki nilai *support* lebih dari 0. Total_ex_after(act) adalah jumlah *executive_choice* (aktivitas setelah act, aktivitas setelah act) yang memiliki nilai *support* lebih dari 0 dan rotal_chain_before(act) adalah jumlah *executive_choice*(aktivitas sebelum act, aktivitas sebelum act) yang memiliki nilai *support* lebih dari 0.

Sebagai contoh, *chain_response*(K,R), *chain*_response(K,S), dan *chain_response*(T,S) tergambar dalam model deklaratif. Dari *rules* pada model deklaratif tersebut, total_next(K) dan total_before(S) adalah dua, dan total_next(T) dan total_before(R) adalah satu, serta total_ex_after(K) adalah satu.

Tabel 3.2 Metode untuk membangun *control-flow patterns*

Pattern	**Peraturan untuk membangun *control-flow patterns***
Sequence	`if total_next(act) == 1:` `act -> O ( y )`
Parallel Split (AND Split)	`If total_next(act) > 1 and total_ex(act) == 0:` `act -> ◊ ( ( y1 ∧ y2 …. ∧ yn ) )`
Synchronization (AND Join)	`If total_before(act) > 1 and total_ex(act) == 0:` `◊( ( y1 ∧ y2 …. ∧ yn ) ) -> O ( act )`
Exclusive choice (XOR Split)	`If total_next(act)>1 and total_ex_after(act)>0 and (total_chain_after(act) < total_after(act)):`

	act -> O ((y1 ∨ y2 …. ∨ yn))
Simple Merge (XOR join)	If total_before(act)>1 and total_ex_before(act)>0 and (total_chain_before(act)< total_next(act)): O ((y1 ∨ y2 …. ∨ yn)) -> O (act)
Multi Choice Pattern (OR Split)	If total_next(act)>1 and total_ex_after(act)>0 and (total_chain_after(act)>= total_next(act)): act -> ◊ ((y1 ∨ y2 …. ∨ yn))
Synchronization Merger (OR Join)	If total_before(act)>1 and total_ex_before(act)>0 and (total_chain_before(act)>= total_next(act)):
	◊ ((x1 ∨ x2 …. ∨ xn)) -> O (act)

Setelah mendapatkan LTL kemudian kita ubah ke dalam bentuk csv. Hal ini diperlukan karena untuk mendapatkan sebuah graf model dari Neo4j, input file yang dapat digunakan hanya berupa ekstensi file csv. Pada Gambar 3.3 ditunjukkan cara mengubah LTL ke dalam bentuk csv. Alur pengubahannya yang pertama adalah, dari relasi XORSplit yang didapatkan dari hasil LTL, dilakukan perulangan sebanyak jumlah data dan diambil data hanya yang berupa *activity* dari *node,* dalam kata lain relasi tiap *node* diabaikan.

Tahap selanjutnya, seluruh *activity* yang telah didapatkan dimasukkan ke dalam suatu file dengan nama LTLXORSPLIT dengan format csv. Proses yang dilakukan untuk relasi XORJoin dan Sequence sama dengan proses pada XORSplit, hanya saja untuk XORJoin relasi yang dibaca pada LTL ialah relasi XORJoin dan hasil file dibentuk dengan nama LTLXORJOIN. Untuk Sequence, relasi yang dibaca pada LTL ialah relasi Sequence kemudian hasil file yang terbentuk diberi nama LTLSequence.

```
// get LTLXORSPLIT
new fileWriter = LTLXORSPLIT.csv
fileWriter.append(FILE_HEADER)
for i=0 in all LTLXORSPLIT
        (
        for j=1 in all LTLXORSPLIT.get(i)
                fileWriter.append(LTLXORSplite.get(i).get(0))
                fileWriter.append(LTLXORSplite.get(i).get(j))
fileWriter.close
// LTLXORJOIN
new fileWriter = LTLXORJOIN.csv
fileWriter.append(FILE_HEADER)
for i=0 in all LTLXORJoin
        (
        for j=1 in all LTLXORJoin.get(i)
                fileWriter.append(LTLXORJoin.get(i).get(j))
                fileWriter.append(LTLXORJoin.get(i).get(0))
fileWriter.close
// LTLSequence
new fileWriter = LTLSequence.csv
fileWriter.append(FILE_HEADER)
println LTLSequence
for i=0 in all LTLSequence
        for j=0 in all LTLSequence.get(i)
                fileWriter.append(LTLSequence.get(i).get(j)
fileWriter.close
```

Gambar 3.3 Algoritma Pengubahan LTL ke Bentuk Csv

Hasilnya digunakan sebagai input untuk menjalankan bahasa cypher pada Neo4j. Gambar 3.4 ini menunjukkan setiap file diubah menjadi relasi yang berbeda. Pada penerapannya setiap aktivitas pada file csv dengan urutan ganjil akan dijadikan node asal, sementara urutan genap akan dijadikkan node tujuan. Jenis relasinya dibuat berdasarkan pada nama file csv yang digunakan sebagai input. File XORSplit.csv digunakan sebagai input untuk membuat node dengan relasi XOR Split, XORJoin.csv akan membuat relasi XOR Join, sementara Sequence.csv akan membuat relasi NEXT pada model bisnis proses.

```
//untuk mendeteksi XORSplit
LOAD CSV WITH HEADERS FROM "file XORSPLIT" AS line
MERGE (n:load { Case_ID:line.Case_ID, Activity:line.activityname }) WITH COLLECT(n) AS
activities
FOREACH (m IN RANGE(0, SIZE(activities)/2-1) |
FOREACH (prev IN [activities[m*2]] |
FOREACH (next IN [activities[m*2+1]] |
MERGE (prev)-[:NEXT]->(next) MERGE (prev)-[:XORSPLIT]->(next))))
//untuk mendeteksi XORJoin
LOAD CSV WITH HEADERS FROM "file XORJOIN" AS line
MERGE (n:load { Case_ID:line.Case_ID, Activity:line.activityname }) WITH COLLECT(n) AS
activities
FOREACH (m IN RANGE(0, SIZE(activities)/2-1) |
FOREACH (prev IN [activities[m*2]] |
FOREACH (next IN [activities[m*2+1]] |
MERGE (prev)-[:NEXT]->(next) MERGE (prev)-[:XORJOIN]->(next))))
//untuk mendeteksi Sequence
LOAD CSV WITH HEADERS FROM "file Sequence" AS line
MERGE (n:load { Case_ID:line.Case_ID, Activity:line.activityname }) WITH COLLECT(n) AS
activities
FOREACH (m IN RANGE(0, SIZE(activities)/2-1) |
FOREACH (prev IN [activities[m*2]] |
FOREACH (next IN [activities[m*2+1]] |
MERGE (prev)-[:NEXT]->(next))))
```

Gambar 3.4 Algoritma Memasukkan Hasil Deklaratif ke Neo4j

3.3.2 Pembentukan *Model Bisnis Proses* dari Event Log

Berbeda dengan pembentukan Model bisnis proses menggunakan Deklaratif yang relasinya sudah diketahui dan kita hanya perlu memasukkannya ke Neo4j. Pada pembentukkan Model bisnis proses dari Event Log langsung, process yang dilakukan sedikit berbeda. Input yang digunakan merupakan data event log murni tanpa relasi yang berkestensi csv. Sehingga kita harus menginputkan data tersebut ke dalam link list dan membuat algoritma sendiri untuk membuat relasinya. Hal ini ditunjukkan pada Gambar 3.5.

File event log dimasukkan ke Neo4j sehingga terbentuk link list dalam bentuk *graf data*. Line Case_ID, Activity, Start_Stamp, End_Stamp dari setiap aktivitas akan dibentuk sebuah node yang berisi CaseId, Name, StartTime, EndTime secara berurutan. Kemudian antar node pada satu case yang sama akan dibentuk relasi NEXT. Aktivitas yang telah terbuah pada suatu case tidak akan dibentuk ulang walaupun terdapat aktivitas tersebut di case lain.

```
//Load CSV
LOAD CSV with headers FROM "file event LOG" AS line
Merge (:Activity
{CaseId:line.Case_ID,Name:line.Activity,StartTime:line.Start_Stamp,EndTime:line.End_Stamp })
LOAD CSV with headers FROM "file event LOG" AS line
Merge (:CaseActivity {Name:line.Activity })
//Connect the activity
Match (c:Activity)
WITH COLLECT(c) AS Caselist
UNWIND RANGE(0,Size(Caselist) - 2) as idx
WITH Caselist[idx] AS s1, Caselist[idx+1] AS s2
match (b:CaseActivity),(a:CaseActivity)
WHERE s1.CaseId = s2.CaseId AND s1.Name = a.Name  AND s2.Name = b.Name
MERGE (a)-[r:NEXT]->(b)
```

Gambar 3.5 Algoritma Memasukkan Event Log ke Neo4j

Algoritma pada Gambar 3.5 hanya membentuk relasi antar node tanpa mengetahui *Control Flow Pattern* yang terjadi diantaranya. Sehingga langkah selanjutnya adalah menjalankan algoritma pada Gambar 3.6 untuk membentuk *Control Flow Pattern*. Apabila pada suatu link list terdapat lebih dari satu relasi yang keluar dari node asal dan pada node tujuan hanya terdapat satu relasi keluar dan relasi masuk maka akan terbentuk relasi XORSplit. Sebaliknya jika jumlah relasi masuk dan relasi keluar pada node asal berjumlah hanya satu dan relasi masuk dari node tujuan berjumlah lebih dari satu maka akan terbentuk relasi XORJoin. Untuk relasi AND Split akan terbentuk apabila jumlah relasi keluar dari node asal lebih dari satu kemudian jumlah relasi masuk dari node tujuan lebih dari dua. Relasi yang terakhir adalah relasi ANDJoin, relasi ini akan terbentuk apabila jumlah relasi masuk dari node asal hanya berjumlah satu, kemudian jumlah relasi keluar dari node tujuan berjumlah lebih dari satu.

```
ir = incoming relation
or = outgoing relation
// for XOR SPLIT relation
Match n-(Next)->a
Where n.or > 1 and (a.ir and a.or) = 1
Merge n-(XORSplit)->a
Return distinct XORSplit, n.name, a.name
// for XOR JOIN relation
Match n-(Next)->a
Where (n.or and n.ir) = 1 and a.ir > 1
Merge n-(XORJoin)->a
Return distinct XORJoim, n.name, a.name
// for AND SPLIT relation
Match n-(Next)->a
Where n.or > 1 and a.ir > 2
Create n-(ANDSPLIT)->a
Return distinct AND, n.name, a.name
// for AND JOIN relation
Match n-(Next)->a
Where n.ir = 1 and a.or > 1
Create n-(ANDJOIN)->a
Return distinct AND, n.name, a.name
```

Gambar 3.6 Algoritma Pembuatan *Control-Flow Pattern* pada Neo4j

3.3.3 Pattern Matching

Dalam tahap ini kita akan membandingkan kedua Model bisnis proses yang telah terbentuk dengan menggunakan algoritma *Pattern Matching*, yaitu mengambil semua node dan relasi yang ada dari suatu proses model dan membandingkannya dengan node dan relasi dari proses model yang lainnya. Berikut Gambar 3.7 yang menjelaskan pseudocodenya. Input pada algoritma *Pattern Matching* berupa hasil *eksport* dari pembentukan Model bisnis proses yang telah terbentuk pada Neo4j. Kemudian dibentuk tiga string yang akan

menyimpan node asal, relasi, dan node tujuan dari kedua proses model. Dilakukan iterasi sebanyak n x m, dimana n merupakan banyaknya relasi dari hasil proses model dari model deklaratif dan m merupakan banyaknya relasi dari hasil proses model dari event log. Apabila terdapat kesamaan antara node asal, relasi, dan node tujuan antara kedua proses model maka persentase akan meningkat. Program akan berhenti setelah iterasi berjalan sebanyak n x m atau persentase telah mencapai 100 %.

```
// JAVA COMPARISION

new br = BufferedReader(csvfromLTL)
while (line1 = br.readLine) != null
        node1 = line1.split(",")
        nodefrom1[a] = node1[0]
        relation1[a] = node1[1]
        nodeto1[a] = node1[2]
        a++
new cr = BufferedReader(csvfromEventLog)
while (line2 = cr.readLine) != null
        node2 = line2.split(",")
        nodefrom2[a] = node2[0]
        relation2[a] = node2[1]
        nodeto2[a] = node2[2]
        a++
for c=0 to a
        compare(nodefrom1,nodefrom2),(relation1,relation2),(nodeto1,nodeto2)
                match++
print percentage = (match/a) * 100 %
```

Gambar 3.7 Algoritma Pattern Matching

3.4 Keluaran

3.4.1 Metode Pengujian

Pada hasil dari tugas akhir ini terdapat 2 graf yang menggambarkan model bisnis proses yaitu graf yang berasal dari LTL, dan graf yang berasal dari Event Log. Untuk menguji baik buruk nya metode graf database ini, maka kita mengkomparasi antara 2 graf ini dengan metode pattern matching untuk mengetahui apakah graf yang terbentuk merupakan graf yang sama atau tidak.

3.4.2 Skenario Uji Coba

Skenario uji coba berdasarkan metode pengujian yang dijelaskan pada bab 3.4.1. Tedapat dua *model bisnis proses* yang akan digunakan untuk pengujian, yaitu model proses hasil dari model deklaratif dan model proses dari LTL. Untuk pengujian hasil perbaikan model proses yang telah mengandung *control-flow pattern* tersebut akan saling dikomparasi dengan metode pattern matching, baik menggunakan fungsi neo4j maupun dengan brute force. Hasil dari pengujian ini adalah perbandingan apakah kedua metode tersebut menghasilkan hasil yang *valid* atau tidak. Berikut merupakan skenario yang akan digunakan.

a) Skenario 1:
   - Pembuatan model bisnis proses dari model deklaratif (menggunakan Declarative Miner).
   - Mendeteksi anomali proses yang berulang (invisible task bertipe redo) dari model deklaratif

b) Skenario 2:
   - Pembuatan model bisnis proses dari event log (tanpa menggunakan Declarative Miner)
   - Mendeteksi anomali proses yang berulang (invisible task bertipe redo) dari event log langsung.

c) Skenario 3:
   - Pattern Matching menggunakan fungsi Neo4j.
   - Pattern Matching menggunakan *Brute Force Algorithm* pada Java

BAB IV
IMPLEMENTASI

Bab ini membahas tentang implementasi dari perancangan sistem. Bahasa pemrograman yang digunakan adalah bahasa pemrograman Python.

4.1 Lingkungan Implementasi

Lingkungan implementasi merupakan lingkungan dimana sistem ini dibangun, dimana akan dijelaskan mengenai kebutuhan perangkat yang diperlukan untuk membangun sistem ini. Lingkungan implementasi dibagi menjadi dua, yaitu lingkungan implementasi terhadap perangkat keras dan lingkungan implementasi terhadap perangkat lunak.

4.1.1. Perangkat Keras

Implementasi dilakukan pada sebuah laptop dengan spesifikasi sebagai berikut:

- Merk : ASUS.
- Seri : N Series 750.
- Prosesor : AMD A8-5545M APU @ 1.70 GHz.
- RAM : 8 GB.

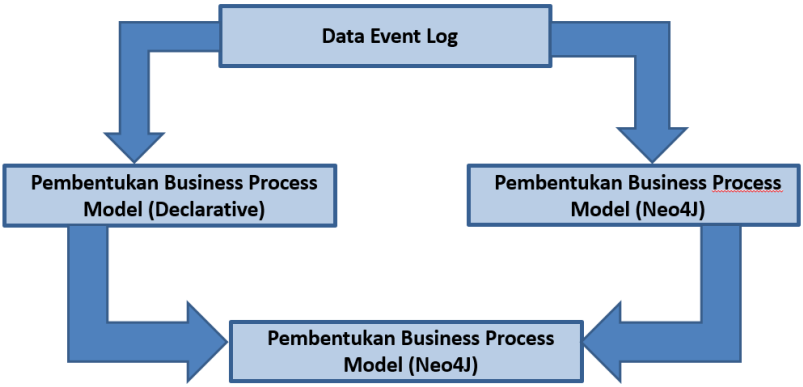
4.1.2. Perangkat Lunak

Perangkat lunak yang mendukung fungsionalitas sistem adalah:

1. Sistem Operasi Windows
   Sistem operasi yang digunakan adalah Microsoft Windows 8.1 Single Languange 64-bit.
2. Java
3. Eclipse
4. ProM
5. Neo4j

4.2 Penjelasan Implementasi

Pada sub bab ini dijelaskan implementasi setiap metode yang dijelaskan pada bab metode pemecahan masalah, sehingga terbentuk suatu perangkat lunak yang mengimplementasi metode-metode pada bab metode pemecahan masalah. Gambar 4.1 merupakan gambaran alur implementasi.



Gambar 4.1 Alur Implementasi

4.2.1 Implementasi

Sub bab ini membahas implementasi untuk pembentukkan sebuah model bisnis proses, baik dari model deklaratif maupun dari event log.

4.2.1.1 Implementasi Pembentukan Model Bisnis Proses

Pada implementasinya, kita perlu untuk membentuk 2 model bisnis proses yang terbagi menjadi 2, yaitu: Model bisnis proses dari model deklaratif dan dari event log.

4.2.1.1.1 Implementasi Pembentukan Business Process Model dari Model Deklaratif

Untuk membentuk sebuah Model bisnis proses dari Model Deklaratif, hal pertama yang dibutuhkan ialah plugin ProM pada Eclipse. Kemudian modifikasi kode sumber pada kelas declare miner, tambahkan kode dibawah ini. Kode Sumber lengkap pada tiap tahap dapat dilihat pada halaman Lampiran. Pada pembentukan model deklaratif terdapat 4 tahapan.

a) Tahap 1

Tahap pertama adalah melakukan perulangan sebanyak total aktivitas, dan digunakan untuk mencari nilai keempat variabel, yaitu: total_before, total_after, total_before_resp, total_after_resp. Kode Sumber 4.1 merupakan potongan program Declarative Miner pada tahap pertama. Pada penerapannya terdapat empat *case*, yaitu:

a.1. Mencari nilai total_after

Apabila declareminerоutput.get(j).get(0) sama dengan aktivitas dan declareminerоutput.get(j) merupakan chain response maka total after bertambah satu.

a.2. Mencari nilai total_before

Apabila declareminerоutput.get(j).get(1) sama dengan aktivitas dan declareminerоutput.get(j) merupakan chain response maka total before bertambah satu.

a.3. Mencari nilai total_after_resp

Apabila declareminerоutput.get(j).get(0) sama dengan aktivitas dan declareminerоutput.get(j) merupakan sebuah response maka total_after_resp bertambah sejumlah satu.

a.4. Mencari nilai total_before_resp

Apabila declareminerоutput.get(j).get(1) sama dengan aktivitas dan declareminerоutput.get(j) merupakan response maka total_before_resp bertambah satu.

```
if(declareMinerOutput.getConstraintParametersMap().get(j
).get(0).equals(act) &&
declareMinerOutput.getTemplate().get(j).equals(DeclareTe
mplate.Chain_Response)){
                        totalafter++;
                    } else
if(declareMinerOutput.getConstraintParametersMap().get(j
).get(1).equals(act) &&
declareMinerOutput.getTemplate().get(j).equals(DeclareTe
mplate.Chain_Response)){
                        totalbefore++;
                    } else
if(declareMinerOutput.getConstraintParametersMap().get(j
).get(0).equals(act) &&
declareMinerOutput.getTemplate().get(j).equals(DeclareTe
mplate.Response)){
                        totalafterresp++;
                    } else
if(declareMinerOutput.getConstraintParametersMap().get(j
).get(1).equals(act) &&
declareMinerOutput.getTemplate().get(j).equals(DeclareTe
mplate.Response)){
                        totalbefresp++;
                    }
```

Kode Sumber 4.1 Potongan Declarative Miner Pada Tahap Pertama

b) Tahap 2

Tahap kedua dijalankan untuk menentukan semua aktivitas yang memiliki relasi Sequence, dan menentukan aktivitas awal dan aktivitas akhir dari suatu proses model. Pertama dilakukan perulangan sebanyak total aktivitas, kemudian terdapat tiga *case*. Kode Sumber 4.2 menunjukan potongan Declarative Miner pada tahap kedua.

b.1. Mencari relasi Sequence

Apabila ada nilai total_after yang bernilai satu, maka dibuat koneksi Sequence antara aktivitas tersebut dengan aktivitas sebelumnya.

b.2. Mencari aktivitas akhir

Apabila ada suatu aktivitas yang memiliki nilai total_after dan total_after_resp yang bernilai 0 (nol)

maka aktivitas tersebut dikategorikan sebagai aktivitas akhir.

b.3. Mencari aktivitas awal

Apabila ada suatu aktivitas yang memiliki nilai total_before dan total_before_resp yang bernilai 0 (nol) maka aktivitas tersebut dikategorikan sebagai aktivitas awal.

```
if(declareMinerOutput.getConstraintParametersMap().get(j
).get(0).equals(act) &&
declareMinerOutput.getTemplate().get(j).equals(
       DeclareTemplate.Chain_Response)){

       y.add(declareMinerOutput.getConstraintParametersM
ap().get(j).get(1));
              }
       }
if(total_after.get(act)==1){
              LTLtemp.add(act);
              LTLtemp.add(y.get(0));
              LTLSequence.add(LTLtemp);
       }
if(total_after.get(act)==0 &&
total_after_resp.get(act)==0){
              LastAct = act;
       } else if(total_before.get(act)==0 &&
total_before_resp.get(act)==0){
              FirstAct = act;
       }
}
```

Kode Sumber 4.2 Potongan Declarative Miner Pada Tahap Kedua

c) Tahap 3

Tahap ketiga dijalankan untuk menentukan nilai 6 variabel, yaitu: totalEXsplit, totalCRsplit, totalCRnext, total EXjoin, totalCRjoin, totalCRbef. Kode Sumber 4.3 menunjukan potongan Declarative Miner pada tahap ketiga. Step yang pertama adalah dibuat variabel “y” yang menyimpan aktivitas dari node tujuan, dan variabel “x” yang menyimpan aktivitas

dari node asal. Kemudian pada setiap perulangan akan dicari keenam variabel.

c.1. Mencari nilai totalEXsplit

Apabila declaremineroutput.get(k) merupakan exclusive choice maka totalEXsplit bertambah satu.

c.2. Mencari nilai totalCRsplit

Apabila declaremineroutput.get(k) merupakan chain response maka totalCRsplit bertambah satu.

c.3. Mencari nilai totalCRnext

Apabila declaremineroutput.get(k).get(0) sama dengan y.get(j) dan declaremineroutput.get(k).get(1) sama dengan aktivitas dan declaremineroutput.get(k) merupakan Chain Response maka totalCRnext bertambah satu.

c.4. Mencari nilai totalEXjoin

Jika diketahui declaremineroutput.get(k) merupakan exclusive choice maka totalEXjoin bertambah satu.

c.5. Mencari nilai totalCRjoin

Apabila declaremineroutput.get(k) merupakan chain response maka totalCRjoin bertambah satu.

c.6. Mencari nilai totalCRbef

Jika declaremineroutput.get(k).get(1) sama dengan y.get(j) dan declaremineroutput.get(k).get(0) sama dengan aktivitas dan declaremineroutput.get(k) merupakan Chain Response maka totalCRbef bertambah satu.

```
if(declareMinerOutput.getTemplate().get(k).equals(Declar
eTemplate.Exclusive_Choice)){
                                        totalEXsplit++;
                                    } else
if(declareMinerOutput.getTemplate().get(k).equals(Declar
eTemplate.Chain_Response)){
                                        totalCRsplit++;
                                    }
```

```
        if(declareMinerOutput.getConstraintParametersMap(
).get(k).get(0).equals(y.get(j)) &&
declareMinerOutput.getConstraintParametersMap().get(k).g
et(1).equals(act) &&
declareMinerOutput.getTemplate().get(k).equals(DeclareTe
mplate.Chain_Response)){
                                        totalCRnext++;
                                }
if(declareMinerOutput.getTemplate().get(k).equals(Declar
eTemplate.Exclusive_Choice)){
                                            totalEXjoin++;
                                        } else
if(declareMinerOutput.getTemplate().get(k).equals(Declar
eTemplate.Chain_Response)){
                                            totalCRjoin++;
                                        }
                                }
if(declareMinerOutput.getConstraintParametersMap().get(k
).get(1).equals(x.get(j)) &&
declareMinerOutput.getConstraintParametersMap().get(k).g
et(0).equals(act) &&
declareMinerOutput.getTemplate().get(k).equals(DeclareTe
mplate.Chain_Response)){
                                        totalCRbef++;
                                }
                        }
                }
        }
```

Kode Sumber 4.3 Potongan Declarative Miner Pada Tahap Ketiga

d) Tahap 4

Tahap keempat sekaligus tahap terakhir merupakan tahapan untuk pembentukan relasi AND, XOR, OR, dan menentukan baik relasi itu Split atau Join. Kode Sumber 4.4 menunjukan potongan Declarative Miner pada tahap keempat. Jika diketahui suatu aktivitas memiliki totalEXsplit sama dengan 0 (nol) maka aktivitas itu berelasi AND split dengan aktivitas berikutnya. Apabila totalCRnext < total_after dari suatu aktivitas maka aktivitas tersebut berelasi XOR split dengan aktivitas berikutnya, jika kedua kondisi tersebut tidak terpenuhi maka aktivitas tersebut berelasi OR split dengan

aktivitas berikutnya. Untuk kondisi untuk relasi Join adalah sebagai berikut: Jika suatu aktivitas memiliki totalEXjoin sama dengan 0 (nol) maka bersifat AND join dengan aktivitas sebelumnya. Apabila totalCRbef < total_before.get(act) maka memiliki relasi XOR join dengan aktivitas sebelumnya. Dan yang terakhir jika dua kondisi join tidak terpenuhi maka aktivitas tersebut memiliki relasi OR join dengan aktivitas sebelumnya.

```
// AND Split
if(total_after.get(act)>1 && totalCRnext==0){
       LTLtemp.add(act);
       if(totalEXsplit==0){
              for(int j=0;j<y.size();j++){
                     LTLtemp.add(y.get(j));
              }
              LTLOtherSplit.add(LTLtemp);
       }
else {
// XOR Split
       if(totalCRnext<total_after.get(act)){
              for(int j=0;j<y.size();j++){
                     LTLtemp.add(y.get(j));
              }
              LTLXORSplit.add(LTLtemp);
       } else {
// OR Split
              for(int j=0;j<y.size();j++){
                     LTLtemp.add(y.get(j));
              }
              LTLOtherSplit2.add(LTLtemp);
              }
       }
}
// AND Join
if(LTLSequence.get(k).get(1).equals(act) &&
LTLSequence.get(k).get(0).equals(x.get(j))){

       LTLSequence.remove(k);
              break;
            }
                     LTLOtherJoin.add(LTLtemp);
```

```
                } else{
// XOR Join
if(LTLSequence.get(k).get(1).equals(act) &&
LTLSequence.get(k).get(0).equals(x.get(j))){

       LTLSequence.remove(k);
              break;
              }
                     LTLXORJoin.add(LTLtemp);
// OR Join             }
else {        if(LTLSequence.get(k).get(1).equals(act) &&
LTLSequence.get(k).get(0).equals(x.get(j))){

       LTLSequence.remove(k);
              break;
                     }
       LTLOtherJoin2.add(LTLtemp);
                            }
```

Kode Sumber 4.4 Potongan Declarative Miner Pada Tahap Keempat

Setelah terbentuk data Linear Temporal Logic (LTL), data harus di proses lebih lanjut agar terbentuk sebuah data eventlog dengan format csv. Hal ini diperlukan dikarenakan untuk mengoutputkan hasil relasi / mengubah eventlog ke dalam link list (bentuk graf Neo4j) diperlukan input dengan format csv (comma delimiter). Perlu diketahui nantinya dalam penelitian kali ini, akan ada pembentukan graf dari 2 eventlog yang berbeda. Eventlog yang pertama yaitu eventlog yang telah diproses terlebih dahulu menggunakan declare miner sehingga semua relasi di dalamnya sudah diketahui. Yang kedua merupakan eventlog yang hanya berisi informasi Case, Activity, dan Waktu (Start Time dan End Time) sehingga agar model bisnis proses yang terbentuk di graf nantinya merupakan suatu process yang benar diperlukan suatu algoritma pada Neo4j untuk mencari relasi tiap activity.

Berikut merupakan algoritma yang akan merubah tiap relasi LTL ke dalam bentuk relasi yang telah ditentukan (LTLXORSPLIT, LTLXORJOIN, LTLSEQUENCE) dengan format file csv. Kode Sumber 4.5 menjelaskan pengubahan ltl ke format csv pada relasi XOR Split. Program pertama akan membentuk sebuah file baru

dengan nama LTLXORSPLIT.csv, kemudian file akan ditambahkan sebuah header yang berisi Case_ID dan activityname. Berikutnya akan dilakukan *double loop* sebanyak jumlah case XOR Split, dan diambil node asal pada perulangan pertama. Dilanjutkan dengan pengambilan node tujuan pada stiap perulangan tingkat kedua. Program akan berhenti apabila index pada loop telah melebihi jumlah case pada XOR Split.

```
FileWriter fileWriter = new
FileWriter("LTLXORSPLIT.csv");
      fileWriter.append(FILE_HEADER);
      for(int i=0;i<LTLXORSplit.size();i++){
    outputStream.print(LTLXORSplit.get(i).get(0) +
" -> <>(");
      for(int
j=1;j<LTLXORSplit.get(i).size();j++){

      outputStream.print(LTLXORSplit.get(i).get(j)
);

      fileWriter.append(NEW_LINE_SEPARATOR);
      fileWriter.append(""+j);
      fileWriter.append(COMMA_DELIMITER);
      fileWriter.append(LTLXORSplit.get(i).get(0))
;           fileWriter.append(NEW_LINE_SEPARATOR);
      fileWriter.append(""+j);
      fileWriter.append(COMMA_DELIMITER);
      fileWriter.append(LTLXORSplit.get(i).get(j))
;
      if(j!=LTLXORSplit.get(i).size()-1){
            outputStream.print(" /\\ ");
            }
      }
      outputStream.println(")");}
fileWriter.close();
```

Kode Sumber 4.5 Pengubahan LTL ke Format CSV pada Relasi LTLXORSPLIT

Kode Sumber 4.6 menjelaskan pengubahan ltl ke format csv pada relasi XOR Join. Program pertama akan membentuk sebuah file baru dengan nama LTLXORJOIN.csv, kemudian file akan ditambahkan sebuah header yang berisi Case_ID dan activityname. Berikutnya akan dilakukan *double loop* sebanyak jumlah case XOR Join. Program akan mengambil node tujuan pada stiap perulangan tingkat kedua. Pada akhir index perulangan tingkat dua, program akan mengambil node asal, dilanjutkan dengan perulangan pada tingkat satu index berikutnya. Program akan berhenti apabila index pada loop telah melebihi jumlah case pada XOR Join.

```
FileWriter fileWriter = new
FileWriter("LTLXORJOIN.csv");
fileWriter.append(FILE_HEADER);
      for(int i=0;i<LTLXORJoin.size();i++){
            outputStream.print("<>(");
            for(int
j=1;j<LTLXORJoin.get(i).size();j++){

      outputStream.print(LTLXORJoin.get(i).get(j))
;

      fileWriter.append(NEW_LINE_SEPARATOR);
      fileWriter.append(""+j);
      fileWriter.append(COMMA_DELIMITER);
      fileWriter.append(LTLXORJoin.get(i).get(j));
            fileWriter.append(NEW_LINE_SEPARATOR);
      fileWriter.append(""+j);
      fileWriter.append(COMMA_DELIMITER);
      fileWriter.append(LTLXORJoin.get(i).get(0));
                  if(j!=LTLXORJoin.get(i).size()-
1){
                        outputStream.print(" /\\
");
            }
      }
```

```
      outputStream.println(") ->
O("+LTLXORJoin.get(i).get(0)+ ")");
}
fileWriter.close();
```

Kode Sumber 4.6 Pseudocode Pengubahan LTL ke Format CSV pada Relasi LTLXORJOIN

Kode Sumber 4.7 menjelaskan pengubahan ltl ke format csv pada relasi Sequence. Program pertama akan membentuk sebuah file baru dengan nama LTLSequence.csv, kemudian file akan ditambahkan sebuah header yang berisi Case_ID dan activityname. Berikutnya akan dilakukan *double loop* sebanyak jumlah case LTL Sequence, dan diambil node asal pada perulangan pertama. Dilanjutkan dengan pengambilan node tujuan pada perulangan tingkat kedua. Program akan berhenti apabila index pada loop telah melebihi jumlah case pada LTL Sequence.

```
FileWriter fileWriter = new
FileWriter("LTLSequence.csv");
fileWriter.append(FILE_HEADER);
outputStream.println("LTLSequence");
      for(int i=0;i<LTLSequence.size();i++){
            outputStream.println(LTLSequence.get(i
).get(0) + " -> O(" + LTLSequence.get(i).get(1)+
")");
            for(int
j=0;j<LTLSequence.get(i).size();j++){
      fileWriter.append(NEW_LINE_SEPARATOR);
      fileWriter.append(""+i+1);
      fileWriter.append(COMMA_DELIMITER)
      fileWriter.append(LTLSequence.get(i).get(j))
;
      }
}
fileWriter.close();
```

Kode Sumber 4.7 Pseudocode Pengubahan LTL ke Format CSV pada Relasi LTLSequence

Setelah terbentuk file dengan format csv, gunakan Kode Sumber 4.8 untuk membentuk tiap file ke bentuk link list (node, dan juga relasinya). File LTLXORSPLIT.csv akan diubah ke relasi XOR Split pada link list, file LTLXORJOIN.csv akan diubah ke relasi XOR Join pada link list, file LTLSequence.csv akan diubah ke relasi NEXT pada link list. Program akan mengambil baris genap pada tiap file untuk dijadikan node awal, kemudian node ganjil pada tiap file akan dijadikan node tujuan. Relasi yang menghubungkan node awal dan node tujuan akan bergantung pada nama file yang digunakan.

```
//for detecting XORSplit
LOAD CSV WITH HEADERS FROM "file XORSPLIT" AS line
MERGE (n:load { Case_ID:line.Case_ID, Activity:line.activityname }) WITH COLLECT(n) AS activities
FOREACH (m IN RANGE(0, SIZE(activities)/2-1) |
FOREACH (prev IN [activities[m*2]] |
FOREACH (next IN [activities[m*2+1]] |
MERGE (prev)-[:NEXT]->(next) MERGE (prev)-[:XORSPLIT]->(next))))
//for detecting XORJoin
LOAD CSV WITH HEADERS FROM "file XORJOIN" AS line
MERGE (n:load { Case_ID:line.Case_ID, Activity:line.activityname }) WITH COLLECT(n) AS activities
FOREACH (m IN RANGE(0, SIZE(activities)/2-1) |
FOREACH (prev IN [activities[m*2]] |
FOREACH (next IN [activities[m*2+1]] |
MERGE (prev)-[:NEXT]->(next) MERGE (prev)-[:XORJOIN]->(next))))
//for detecting Sequence
LOAD CSV WITH HEADERS FROM "file:///sequence.csv" AS line
MERGE (n:load { Case_ID:line.Case_ID, Activity:line.activityname }) WITH COLLECT(n) AS activities
FOREACH (m IN RANGE(0, SIZE(activities)/2-1) |
FOREACH (prev IN [activities[m*2]] |
FOREACH (next IN [activities[m*2+1]] |
MERGE (prev)-[:NEXT]->(next))))
```

Kode Sumber 4.8 Memodelkan Model Bisnis Proses dari LTL

4.2.1.1.2 Implementasi Pembentukan Model Bisnis Proses dari Event Log

Model bisnis proses lainnya yang harus dimodelkan ialah model yang langsung dari event log. Kita diharuskan membentuk file event log dari csv ke bentuk link list. Gunakan Kode Sumber 4.9 untuk memodelkan hal tersebut Program ini akan memodelkan tiap aktivitas pada csv ke bentuk node pada Neo4j. Setelah itu tiap node yang telah terbentuk pada case yang sama akan dihubungkan dengan relasi NEXT. Aktivitas yang sama

pada case yang berbeda tidak akan terbentuk untuk kedua kalinya.

```
//LOAD CSV
LOAD CSV with headers FROM "file e" AS line
Merge (:Activity
{CaseId:line.Case_ID,Name:line.Activity,StartTime:line.Start_Stamp,EndTime:line.End_Stamp })
LOAD CSV with headers FROM "file:///eventlogswitch.csv" AS line
Merge (:CaseActivity {Name:line.Activity })
//CONNECT THE ACTIVITY
Match (c:Activity)
WITH COLLECT(c) AS Caselist
UNWIND RANGE(0,Size(Caselist) - 2) as idx
WITH Caselist[idx] AS s1, Caselist[idx+1] AS s2
match (b:CaseActivity),(a:CaseActivity)
WHERE s1.CaseId = s2.CaseId AND s1.Name = a.Name  AND s2.Name = b.Name
MERGE (a)-[r:NEXT]->(b)
```

Kode Sumber 4.9 Memodelkan Model Bisnis Proses dari *Event Log*

Setelah kita mendapatkan model yang berupa node dan relasinya. Hal berikutnya yang harus dilakukan ialah menambahkan *Control Flow Pattern* kedalam proses model tersebut dengan menggunakan algoritma pada Kode Sumber 4.10. Program ini akan membentuk relasi XORSplit apabila jumlah relasi keluar yang berupa NEXT pada node awal lebih dari satu, dan jumlah relasi masuk dan keluar yang berupa NEXT pada node tujuan berupa 1. Sebaliknya relasi XORJoin akan terbentuk apabila relasi keluar yang berupa NEXT pada node awal berjumlah satu, dan relasi masuk yang berupa NEXT pada node tujuan berjumlah lebih dari 1.

```
match (n)-[r:NEXT]->(a)
where size((n)-->())>1 and (size((a)<--()) = 1 and size((a)-->()) = 1)
Merge(n)-[:XORSplit]->(a)
Return distinct 'XORSplit', n.Name, a.Name

match (n)-[r:NEXT]->(a)
where size((n)-->())=1 and (size((a)<--()) > 1)
Merge(n)-[:XORJoin]->(a)
Return distinct 'XORJoin', n.Name, a.Name
```

Kode Sumber 4.10 Membentuk *Control-Flow Pattern* pada Model Proses

4.2.1.2 lmplementasi Pattern Matching

Setelah terbentuk kedua Model bisnis proses, langkah selanjutnya ialah membandingkan keduanya. Disini terdapat 2 cara untuk membandingkan model proses.

4.2.1.2.1 Implementasi Pattern Matching Menggunakan Fungsi Neo4j

Cara yang pertama ialah dengan menggunakan fungsi yang sudah disediakan oleh Neo4j sendiri. Yaitu dengan cara mengenkripsi model proses tersebut kemudian hasil enkripsi dibandingkan dengan menggunakan fungsi phonetic. Kode Sumber 4.11 dibawah merupakan algoritma pattern matching tersebut. Program ini dimulai dengan memodelkan semua relasi yang terdapat pada Model bisnis proses dan memasukannya ke sebuah string “result”. String “result” tersebut kemudian akan dienkripsi menggunakan md5 dan disimpan ke string “finalresult”. Hasil “finalresult” yang terbuat dari Business Process Model dari Model Declaratif dan Event Log kemudian akan dibandingkan dengan menjalankan fungsi phoneticDelta pada Neo4j. Jika keluar output 4 maka hasilnya ialah sangat mirip, sebaliknya jika keluar 1 maka hasilnya sangat jauh. Fungsi phonetic sendiri memiliki kelemahan yaitu hanya dapat membandingkan model proses yang identik.

```
// Pattern Matching Using Neo4j
// return '4'  (jika sangat mirip)
// return '1' (jika sangat berbeda)
START n=node(*) MATCH (n)-[r]->(m)
WITH collect(properties(n)) AS result
WITH apoc.util.md5(result) AS finalresult
RETURN finalresult
CALL apoc.text.phoneticDelta('Data1', 'Data2')
```

Kode Sumber 4.11 Pattern Matching dengan Fungsi Neo4j

4.2.1.2.2 Implementasi Pattern Matching Menggunakan Brute Force Algorithm pada Java

Cara yang kedua ialah dengan menggunakan algoritma brute force pada Java. Untuk melakukan Pattern Matching dengan cara ini, hal pertama yang harus dilakukan ialah mengeksport data link list pada Neo4j ke format csv dengan Kode Sumber 4.12. Program ini menampilkan semua node asal, relasi, dan node tujuan, kemudian mengeksportnya ke bentuk csv.

```
START n=node(*) MATCH (n)-[r]->(m) RETURN n,r,m;
```

Kode Sumber 4.12 Membentuk CSV dari Suatu Model Proses

Setelah terbentuk data dengan format csv, kemudian jalankan algoritma Brute Force sesuai kode sumber dibawah ini. Terdapat 4 tahapan dalam *Brute Force Pattern Matching Algorithm*.

a) Tahap 1

Tahap yang pertama adalah inisialisasi, di tahap ini kita akan mengimport semua library yang dibutuhkan dan juga inisialisasi variabel baik berupa integer, double , string, dan juga inisialisasi lokasi file yang akan dibaca. File yang dibaca berupa hasil eksport ke csv dari model bisnis proses yang telah dibentuk. Kode Sumber 4.13 menunjukan algoritma Pattern Matching pada tahap pertama.

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.Date;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Locale;
```

```
public class HelloTA {

    public static void main(String[] args) {

        String csvFile1 = "C:\\XOREventLog.csv";
        String csvFile2 = "C:\\XORLTL.csv";
        String line1 = "", line2 = "";
        String cvsSplitBy = ",";
        String[] nodefrom1, nodeto1, relation1;
        String[] nodefrom2, nodeto2, relation2;
        nodefrom1 = new String[100];
        nodeto1 = new String[100];
        relation1 = new String[100];
        nodefrom2 = new String[100];
        nodeto2 = new String[100];
        relation2 = new String[100];
        int a=0, b=0, c=0, d=0;
        int flag=1;
        double match=0;
        double percentage;
```

Kode Sumber 4.13 Pattern Matching Pada Tahap Pertama

b) Tahap 2

Tahap kedua adalah menginput file csv yang terbentuk dari Business Process Model dari model deklaratif. Pada tahap ini akan dilakukan perulangan terus menerus, setiap perulangan file akan dibaca per baris, kemudian akan dipisahakan setiap terdapat tanda "," (*comma*). Hasil pemisahan yang pertama akan dimasukkan ke dalam string nodefrom1, pemisahan kedua dimasukkan ke relation1, dan pemisahan ketiga akan dimasukkan ke string nodeto1. Perulangan akan berhenti apabila baris yang dibaca dari file berupa null. Kode Sumber 4.14 menunjukan algoritma Pattern Matching pada tahap keuda.

```
try (BufferedReader br = new BufferedReader(new
FileReader(csvFile1)))
        {

            while ((line1 = br.readLine()) !=
null)
            {
                String[] country1 =
line1.split(cvsSplitBy);

                nodefrom1[a] = country1[0];
                relation1[a] = country1[1];
                nodeto1[a] = country1[2];

                a++;
            }
        }

        catch (IOException e) {
            e.printStackTrace();
        }
```

Kode Sumber 4.14 Pattern Matching Pada Tahap Kedua

c) Tahap 3

Tahap ketiga adalah menginput file csv yang terbentuk dari Business Process Model dari event log. Pada tahap ini akan dilakukan perulangan terus menerus, setiap perulangan file akan dibaca per baris, kemudian akan dipisahakan setiap terdapat tanda "," (*comma*). Hasil pemisahan yang pertama akan dimasukkan ke dalam string nodefrom2, pemisahan kedua dimasukkan ke relation2, dan pemisahan ketiga akan dimasukkan ke string nodeto2. Perulangan akan berhenti apabila baris yang dibaca dari file berupa null. Kode Sumber 4.15 menunjukan algoritma Pattern Matching pada tahap ketiga.

```
try (BufferedReader cr = new BufferedReader(new
FileReader(csvFile2)))
        {

            while ((line2 = cr.readLine()) !=
null)
            {

                String[] country2 =
line2.split(cvsSplitBy);

                nodefrom2[b] = country2[0];
                relation2[b] = country2[1];
                nodeto2[b] = country2[2];

                b++;
            }
        }

        catch (IOException e) {
            e.printStackTrace();
        }
```

Kode Sumber 4.15 Pattern Matching Pada Tahap Ketiga

d) Tahap 4

Tahap keempat sekaligus tahap terakhir adalah membandingkan hasil string yang telah dibuat pada tahap kedua dengan hasil string pada tahap ketiga. Pada tahap ini dilakukan *double loop.* Perulangan pertama untuk membaca indeks string dari hasil deklaratif, perulangan kedua untuk membaca indeks string dari hasil event log. Apabila antara "nodefrom1" dan "nodefrom2", "relation1" dan "relation2", "nodeto1" dan "nodeto2" semuanya sama, maka nilai "match" akan bertambah satu. Setelah looping berakhir nilai "match" akan dibagikan dengan total jumlah baris yang paling besar dan dikalikan dengan 100%. Hasil tersebut kemudian akan disimpan pada variabel "percentage" dan akan ditampilkan.

Kode Sumber 4.16 menunjukan algoritma Pattern Matching pada tahap keempat.

```
for (c = 1; c < a; c++)
        {
            for (d=1; d < b; d++)
            {
                  if (
(nodefrom1[c].equals(nodefrom2[d])) &&
(relation1[c].equals(relation2[d])) &&
(nodeto1[c].equals(nodeto2[d])) )
                  {
                        match++;

      percentage = ((match/(a-1))*100);

                        break;
                  }
            }
        }
        percentage = ((match/(a-1))*100);
        System.out.println("Percentage = " +
percentage + " %");

    }

}
```

Kode Sumber 4.16 Pattern Matching Pada Tahap Keempat

BAB V
PENGUJIAN DAN EVALUASI

Bab ini membahas hasil dan pembahasan pada aplikasi yang dikembangkan. Pada bab ini akan dijelaskan tentang data yang digunakan, hasil yang didapatkan dari penggunaan perangkat lunak dan uji coba yang dilakukan pada perangkat lunak yang telah dikerjakan untuk menguji apakah fungsionalitas aplikasi telah diimplementasikan dengan benar dan berjalan sebagaimana mestinya.

5.1 Lingkungan Uji Coba

Lingkungan uji coba menjelaskan lingkungan yang digunakan untuk menguji implementasi pembuatan sistem pada tugas akhir ini. Lingkungan uji coba meliputi perangkat keras dan perangkat lunak yang dijelaskan sebagai berikut:

1. Perangkat keras
   a. Prosesor : AMD A8-5545M APU @ 1.70GHz.
   b. Memori (RAM) : 8 GB.
   c. Tipe sistem : 64-bit sistem operasi.
2. Perangkat lunak
   a. Sistem operasi : Windows 10.
   b. Kakas bantu : Java.

5.2 Hasil Implementasi Pembentukan Model Bisnis Proses

Pada bab ini akan dijelaskan hasil implemetasi dari pembentukan model bisnis proses. Proses model yang terbentuk terbagi menjadi 2 yaitu hasil proses model dari model deklaratif,dan hasil proses model dari event log.

5.2.1 Hasil Implementasi Pembentukan Model Bisnis Proses dari Model Deklaratif

Didapatkan hasil implementasi dari metode yang dijalankan. Terdapat 5 jenis LTL yang terdiri dari LTLFirstLast untuk mendapatkan aktivitas pertama dan terakhir suatu model. LTL Sequence untuk mendapatkan aktivitas yang saling terhubung dalam arti lain berurutan. LTL XorSPLIT untuk mendapatkan aktivitas yang mengandung relasi XOR Split, LTL XorJoin untuk mendapatkan aktivitas yang mengandung relasi XOR join, LTL NonFreeChoice untuk mendapatkan aktivitas yang mengandung relasi non free choice. Hasil dari LTL tersebut dapat dilihat dari Tabel 5.1.

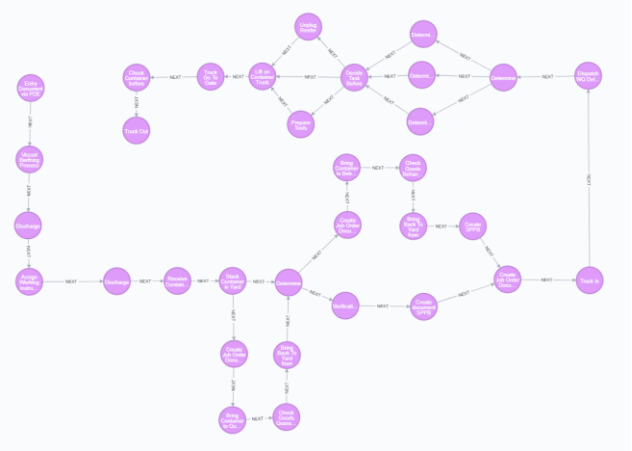
Tabel 5.1 Hasil LTL

Jenis LTL	Isi LTL
LTLFirstLast	Firstactivity(Entry Document via PDE-complete) Lastactivity(Truck Out-complete)
LTLSequence	Discharge Yard Planning-complete -> O(Assign Working Instruction to CC and Yard-complete) Entry Document via PDE-complete -> O(Vessel Berthing Process-complete) Bring Container to Behandle Area-complete -> O(Check Goods Behandle-complete) Create Job Order Document Delivery-complete -> O(Truck in-complete) Truck Go To Gate Out-complete -> O(Check Container before Truck out-complete) Receive Container-complete -> O(Stack Container in Yard-complete) Assign Working Instruction to CC and Yard-complete -> O(Discharge Container-

Jenis LTL	Isi LTL
	complete) Lift on Container Truck-complete -> O(Truck Go To Gate Out-complete) Check Goods Behandle-complete -> O(Bring Back To Yard from Behandle-complete) Bring Back To Yard from Behandle-complete -> O(Create SPPB-complete) Vessel Berthing Process-complete -> O(Discharge Yard Planning-complete) Create Job Order Document Behandle-complete -> O(Bring Container to Behandle Area-complete) Truck in-complete -> O(Dispatch WQ Delivery to CHE-complete) Verification Document-complete -> O(Create document SPPB-complete) Discharge Container-complete -> O(Receive Container-complete) Create Job Order Document Quarantine-complete -> O(Bring Container to Quarantine Area-complete) Check Goods Quarantine-complete -> O(Bring Back To Yard from Quarantine-complete) Bring Container to Quarantine Area-complete -> O(Check Goods Quarantine-complete) Check Container before Truck out-complete -> O(Truck Out-complete) Dispatch WQ Delivery to CHE-complete -> O(Determine Container Type-complete)
LTLXORSplit	Determine Category-complete -> O(Create Job Order Document Behandle-complete ∨

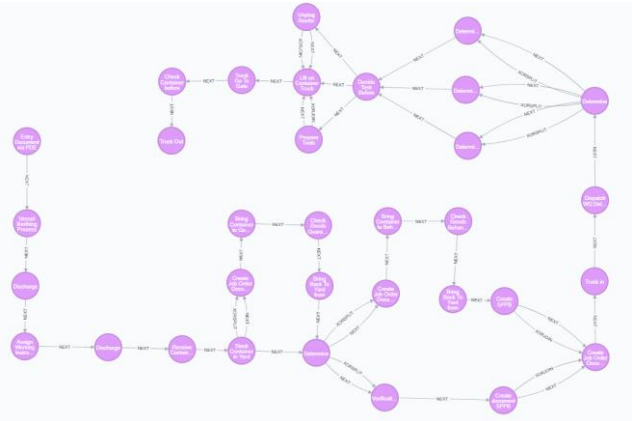
Jenis LTL	Isi LTL
	Verification Document-complete) Determine Container Type-complete -> O(Determining Reefer-complete \/ Determining Dry-complete \/ Determining Uncontainer-complete) Stack Container in Yard-complete -> O(Invisible_Task \/ Create Job Order Document Quarantine-complete)
LTLXORJoin	O(Create SPPB-complete \/ Create document SPPB-complete) -> O(Create Job Order Document Delivery-complete) O(Bring Back To Yard from Quarantine- complete \/ Invisible_Task) -> O(Determine Category-complete) O(Prepare Tools-complete \/ Invisible_Task \/ Unplug Reefer-complete) -> O(Lift on Container Truck-complete)
LTLNonFreeChoice	<>((Determining Uncontainer-complete /\ Decide Task Before Lift Container- complete /\ Prepare Tools-complete) \/ (Determining Reefer-complete /\ Decide Task Before Lift Container-complete /\ Unplug Reefer-complete) \/ (Invisible_Task /\ Decide Task Before Lift Container-complete /\ Determining Dry- complete))

Setelah mendapatkan hasil LTL, kemudian diubah ke format csv. Proses selanjutnya yang dijalankan ialah memasukkan csv tersebut ke dalam neo4j dalam bentuk link list. Hasilnya bisa dilihat pada Gambar 5.1.



Gambar 5.1 Hasil Model Bisnis Proses dari LTL

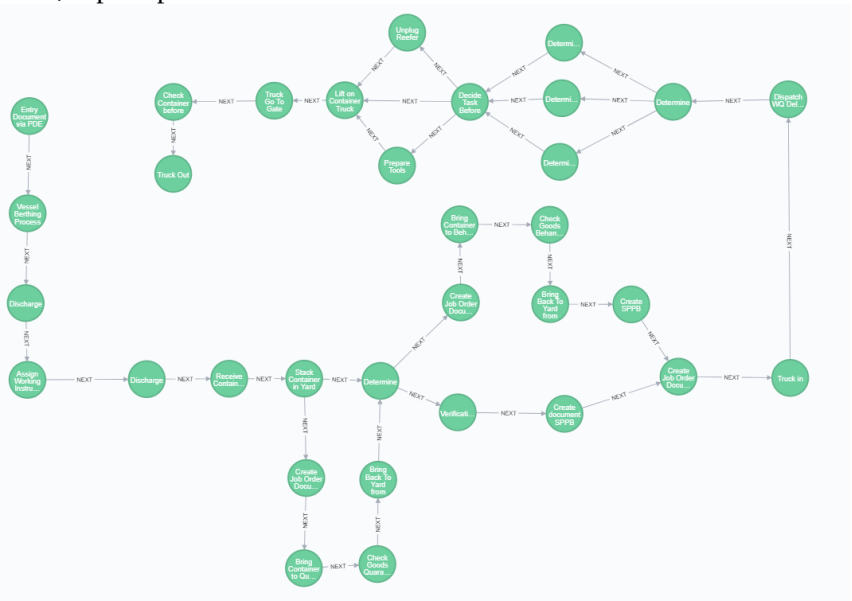
Hasil tersebut belum merupakan hasil model bisnis proses yang lengkap. Hal tersebut dikarenakan model tersebut belum memiliki *control flow pattern*, sehingga langkah berikutnya yang harus dilakukan ialah menambahkan relasi ke dalam model. Hasil akhir dari model bisnis proses dari model deklaratif dapat dilihat pada Gambar 5.2.



Gambar 5.2 Membentuk *Control-Flow Pattern* dari Model Deklaratif

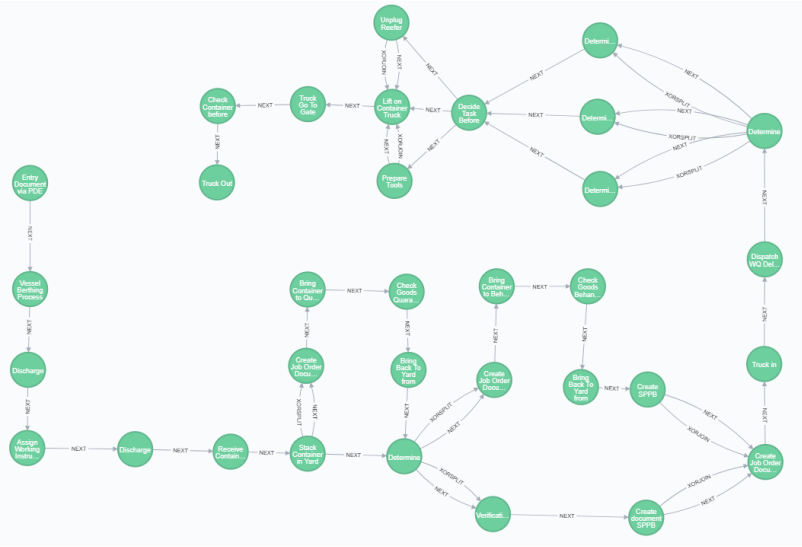
5.2.2 Hasil Implementasi Pembentukan Model Bisnis Proses dari Event Log

Pada implementasi pembentukan mode bisnis proses dari event log. Cara yang dilakukan hampir sama, pertama yaitu memasukkan event log ke Neo4j sehingga berbentuk link list. Dari link list tersebut sudah dapat terlihat semua node dan relasi antar node, seperti pada Gambar 5.3.



Gambar 5.3 Hasil Model Bisnis Proses dari Event Log

Setelah terbentuk link list (node dan relasinya), hal berikutnya adalah membentuk *control flow pattern* dari model proses tersebut. Hasil dari pembentukkan *control flow pattern* dapat terlihat pada Gambar 5.4 dibawah ini.



Gambar 5.4 Membentuk *Control-Flow Pattern* dari Event Log

5.2.3 Hasil Implementasi Pembentukan Model Bisnis Proses untuk Case Anomaly dari Model Deklaratif

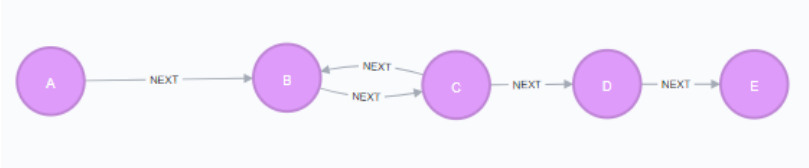
Berikut merupakan hasil percobaan untuk membentuk sebuah proses model yang mengandung sebuah anomali dari model deklaratif. Gambar 5.5 merupakan event log yang mengandung anomali, dapat dilihat untuk trace yang terjadi pada event log tesebut adalah sebagai berikut: A-B-C-B-C-D-E dan A-B-C-B-C-B-C-D-E. Hal tersebut merupakan sebuah anomali dengan jenis Mengulang atau *Re-do*. Sebuah proses model bersifat konkurensi, sehingga aktivitas yang sama tidak boleh ditampilkan secara berulang. Gambar 5.6 merupakan hasil pemulihan dari event log proses model yang mengandung anomali *Re-do* menggunakan algoritma deklaratif. Gambar 5.7 merupakan hasil pembentukan model prosesnya, terlihat pada gambar sudah tidak ada proses yang redundan.

Case_ID	Activity	Start_Stamp	End_Stamp
PP1	A	3/8/2016 10:32	3/8/2016 10:35
PP1	B	3/8/2016 10:35	3/8/2016 10:38
PP1	C	3/8/2016 10:38	3/8/2016 10:41
PP1	B	3/8/2016 10:41	3/8/2016 10:44
PP1	C	3/8/2016 10:44	3/8/2016 10:47
PP1	D	3/8/2016 10:47	3/8/2016 10:50
PP1	E	3/8/2016 10:50	3/8/2016 10:53
PP2	A	3/8/2016 10:53	3/8/2016 10:56
PP2	B	3/8/2016 10:56	3/8/2016 10:59
PP2	C	3/8/2016 10:59	3/8/2016 11:02
PP2	B	3/8/2016 11:02	3/8/2016 11:05
PP2	C	3/8/2016 11:05	3/8/2016 11:08
PP2	B	3/8/2016 11:08	3/8/2016 11:11
PP2	C	3/8/2016 11:11	3/8/2016 11:14
PP2	D	3/8/2016 11:14	3/8/2016 11:17
PP2	E	3/8/2016 11:17	3/8/2016 11:20

Gambar 5.5 Event Log yang Mengandung Anomali

Case_ID	activityname
1	A
1	B
2	B
2	C
3	C
3	B
4	C
4	D
5	D
5	E

Gambar 5.6 Hasil Pemulihan Event Log yang Mengandung Anomali dengan Menggunakan Declarative Miner



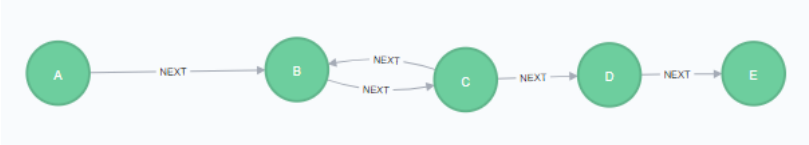
Gambar 5.7 Hasil Model Proses yang Mengandung Anomali dari Model Deklaratif

5.2.4 Hasil Implementasi Pembentukan Model Bisnis Proses untuk Case Anomaly

Berikut merupakan hasil percobaan untuk membentuk sebuah proses model yang mengandung sebuah anomali langsung dari event log. Gambar 5.8 merupakan event log yang mengandung anomali, dapat dilihat untuk trace yang terjadi pada event log tesebut adalah sebagai berikut: A-B-C-B-C-D-E dan A-B-C-B-C-B-C-D-E. Hal tersebut merupakan sebuah anomali dengan jenis Mengulang atau *Re-do*. Sebuah proses model bersifat konkurensi, sehingga aktivitas yang sama tidak boleh ditampilkan secara berulang. Gambar 5.9 merupakan hasil pemulihan dari proses model yang mengandung anomali *Re-do*. Pada gambar sudah terlihat sudah tidak ada proses yang redundan.

Case_ID	Activity	Start_Stamp	End_Stamp
PP1	A	3/8/2016 10:32	3/8/2016 10:35
PP1	B	3/8/2016 10:35	3/8/2016 10:38
PP1	C	3/8/2016 10:38	3/8/2016 10:41
PP1	B	3/8/2016 10:41	3/8/2016 10:44
PP1	C	3/8/2016 10:44	3/8/2016 10:47
PP1	D	3/8/2016 10:47	3/8/2016 10:50
PP1	E	3/8/2016 10:50	3/8/2016 10:53
PP2	A	3/8/2016 10:53	3/8/2016 10:56
PP2	B	3/8/2016 10:56	3/8/2016 10:59
PP2	C	3/8/2016 10:59	3/8/2016 11:02
PP2	B	3/8/2016 11:02	3/8/2016 11:05
PP2	C	3/8/2016 11:05	3/8/2016 11:08
PP2	B	3/8/2016 11:08	3/8/2016 11:11
PP2	C	3/8/2016 11:11	3/8/2016 11:14
PP2	D	3/8/2016 11:14	3/8/2016 11:17
PP2	E	3/8/2016 11:17	3/8/2016 11:20

Gambar 5.8 Event Log yang Mengandung Anomali



Gambar 5.9 Hasil Proses Model yang Mengandung Anomali

5.3 Hasil Implementasi Pattern Matching

Dalam tahap ini, setelah terbentuk model bisnis proses dari model deklaratif dan event log. Kita akan membandingkan keduanya. Untuk membandingkannya terdapat 2 cara, yaitu menggunakan fungsi pada neo4j dan juga menggunakan algoritma brute force pada java.

5.3.1 Hasil Implementasi Pattern Matching Menggunakan Fungsi Neo4j

Cara yang pertama ialah dengan mengenkripsi suatu model bisnis proses. Gambar 5.10 berikut menunjukkan contoh hasil enkripsi tersebut. Final result merupakan suatu string yang menyimpan hasil enkripsi dari suatu model bisnis proses. Hasil dari final result ditampilkan pada baris berikutnya.

"finalresult"
"974cb07bdfea55ca90acec6eaf430f35"

Gambar 5.10 Hasil Enkripsi Model Proses

Setelah dilakukan enkripsi, digunakan suatu fungsi Neo4j yang dinamakan phonetic untuk membandingkan kedua hasil enkripsi tersebut. Output yang akan dihasilkan berupa angka 1

sampai 4. Angka 1 menandakan sangat berbeda, dan sebaliknya angka 4 menandakan sangat mirip. Gambar 5.11 berikut menunjukkan hasil dari fungsi phonetic.

"phonetic1"	"phonetic2"	"delta"
"C131"	"C131"	4

Gambar 5.11 Hasil dari Fungsi Phonetic

5.3.2 Hasil Implementasi *Pattern Matching* Menggunakan *Brute Force Algorithm* pada Java

Pada *Pattern Matching* dengan algoritma brute force, model proses harus diubah terlebih dahulu ke bentuk csv. Data yang diambil adalah "asal node", "jenis relasi", dan "node tujuan". Gambar 5.12s berikut merupakan potongan hasil perubahan model proses ke bentuk csv. Untuk data lengkap perubahan model proses ke bentuk csv dapat dilihat pada halaman Lampiran.

"n"	"r"	"m"
{"Name":"Entry Document via PDE"}	{"relation":"NEXT"}	{"Name":"Vessel Berthing Process"}
{"Name":"Vessel Berthing Process"}	{"relation":"NEXT"}	{"Name":"Discharge Yard Planning"}
{"Name":"Discharge Yard Planning"}	{"relation":"NEXT"}	{"Name":"Assign Working Instruction to CC and Yard"}
{"Name":"Assign Working Instruction to CC and Yard"}	{"relation":"NEXT"}	{"Name":"Discharge Container"}
{"Name":"Discharge Container"}	{"relation":"NEXT"}	{"Name":"Receive Container"}
{"Name":"Receive Container"}	{"relation":"NEXT"}	{"Name":"Stack Container in Yard"}
{"Name":"Stack Container in Yard"}	{"relation":"XOR Split"}	{"Name":"Determine Category"}
{"Name":"Stack Container in Yard"}	{"relation":"XOR Split"}	{"Name":"Create Job Order Document Quarantine"}

Gambar 5.12 Potongan Hasil Perubahan Model Proses ke CSV

Setelah terbentuk data node dan relasinya dalam format csv. Data tersebut kemudian akan digunakan sebagai input untuk menjalankan *Brute Force Pattern Matching Algorithm*. Hasil tiap iterasi dari algoritma *Pattern Matching* dapat diliht pada Tabel 5.2 sampai Tabel 5.6. Hasil akhir dari algoritma *Pattern Matching* merupakan sebuah persentase dari kecocokan kedua model proses. Gambar 5.13 merupakan hasil akhir dari algoritma *Brute Force*.

```
Percentage = 100.0 %
```

Gambar 5.13 Hasil Persentase dari Algoritma Pattern Matching

Tabel 5.2 Iterasi Pertama dari Algoritma Pattern Matching

Input from Event Log	Input from LTL	Matching Percentage
A XOR Split B A XOR Split C A XOR Split D A NEXT D A NEXT C A NEXT B B XOR Join E B NEXT E C XOR Join E C NEXT E D XOR Join E D NEXT E	**A XOR Split B** A NEXT B A XOR Split C A NEXT C A XOR Split D A NEXT D B NEXT E B XOR Join E C XOR Join E C NEXT E D XOR Join E D NEXT E	Match = 1 Total Data = 12 Percentage = 1/12 = 8.33 %

Tabel 5.3 Iterasi Kedua dari Algoritma Pattern Matching

Input from Event Log	Input from LTL	Matching Percentage
A XOR Split B **A XOR Split C** A XOR Split D A NEXT D A NEXT C A NEXT B B XOR Join E B NEXT E C XOR Join E C NEXT E D XOR Join E D NEXT E	**A XOR Split B** A NEXT B A XOR Split C A NEXT C A XOR Split D A NEXT D B NEXT E B XOR Join E C XOR Join E C NEXT E D XOR Join E D NEXT E	Match = 1 Total Data = 12 Percentage = 1/12 = 8.33 %

Tabel 5.4 Iterasi Ketiga dari Algoritma Pattern Matching

Input from Event Log	Input from LTL	Matching Percentage
A XOR Split B **A XOR Split C** A XOR Split D A NEXT D A NEXT C A NEXT B B XOR Join E B NEXT E C XOR Join E C NEXT E D XOR Join E D NEXT E	A XOR Split B **A NEXT B** A XOR Split C A NEXT C A XOR Split D A NEXT D B NEXT E B XOR Join E C XOR Join E C NEXT E D XOR Join E D NEXT E	Match = 1 Total Data = 12 Percentage = 1/12 = 8.33 %

Tabel 5.5 Iterasi Keempat dari Algoritma Pattern Matching

Input from Event Log	Input from LTL	Matching Percentage
A XOR Split B **A XOR Split C** A XOR Split D A NEXT D A NEXT C A NEXT B B XOR Join E B NEXT E C XOR Join E C NEXT E D XOR Join E D NEXT E	A XOR Split B A NEXT B **A XOR Split C** A NEXT C A XOR Split D A NEXT D B NEXT E B XOR Join E C XOR Join E C NEXT E D XOR Join E D NEXT E	Match = 2 Total Data = 12 Percentage = 2/12 = 16.66 %

Tabel 5.6 Iterasi Terakhir dari Algoritma Pattern Matching

Input from Event Log	Input from LTL	Matching Percentage
A XOR Split B A XOR Split C A XOR Split D A NEXT D A NEXT C A NEXT B B XOR Join E B NEXT E C XOR Join E C NEXT E D XOR Join E **D NEXT E**	A XOR Split B A NEXT B A XOR Split C A NEXT C A XOR Split D A NEXT D B NEXT E B XOR Join E C XOR Join E C NEXT E D XOR Join E **D NEXT E**	Match = 12 Total Data = 12 Percentage = 12/12 = 100.0 %

BAB VI
KESIMPULAN DAN SARAN

Pada bab ini akan diberikan kesimpulan yang diambil selama pengerjaan Tugas Akhir serta saran-saran tentang pengembangan yang dapat dilakukan terhadap Tugas Akhir ini di masa yang akan datang.

6.1 Kesimpulan

Dari hasil pengamatan selama proses perancangan, implementasi, dan pengujian perangkat lunak yang dilakukan, dapat diambil kesimpulan sebagai berikut:

1. Data non-relational dapat diimplementasikan dalam Neo4j.
2. Metode Pattern Matching dapat digunakan untuk membandingkan Model bisnis proses.
3. Neo4j dapat digunakan untuk membuat model bisnis proses.
4. Metode 1 (Pembentukan model proses dari model deklaratif) dan Metode 2 (Pembentukan model proses dari event log) sama-sama berhasil membentuk model bisnis proses dan menangani anomali *invisible task* yang bertipe redo.
5. Pola dalam bentuk LTL dapat terbentuk walaupun hanya berdasar pada log data.

6.2 Saran

Berikut merupakan beberapa saran untuk pengembangan sistem di masa yang akan datang. Saran-saran ini didasarkan pada hasil perancangan, implementasi dan pengujian yang telah dilakukan.

1. Fitur memasukkan *event log* dapat dikembangkan dengan format lain selain format Excel.

[Halaman ini sengaja dikosongkan]

DAFTAR PUSTAKA

[1] W.M.P. Van Der Aalst, "Process Mining: Discovery, Conformance and Enhancement of Business Processes," 2011.

[2] R. Sarno, Y. A. Effendi, and F. Haryadita, "Modified Time-Based Heuristics Miner for Parallel Business Processes," *Int. Rev. Comput. Softw.*, vol. 11, no. 3, pp. 249–260, 2016. https://doi.org/10.15866/irecos.v11i3.8717

[3] N. Y. Setiawan and R. Sarno, "Multi-Criteria Decision Making for Selecting Semantic Web Service Considering Variability and Complexity Trade-Off," *J. Theor. Appl. Inf. Technol.*, vol. 86, no. 2, pp. 316–326, 2016.

[4] S. R.a *et al.*, "Developing a workflow management system for enterprise resource planning," *IAENG Int. J. Comput. Sci.*, vol. 72, no. 3, pp. 412–421, 2015.

[5] R. Sarno and K. R. Sungkono, "Coupled Hidden Markov Model for Process Mining of Invisible Prime Tasks," *Int. Rev. Comput. Softw.*, vol. 11, no. 6, pp. 539–547, 2016. https://doi.org/10.15866/irecos.v11i6.9555

[6] R. Sarno and K. R. Sungkono, "Hidden Markov Model for Process Mining of Parallel Business Processes," *Int. Rev. Comput. Softw.*, vol. 11, no. 4, pp. 290–300, 2016. https://doi.org/10.15866/irecos.v11i4.8700

[7] S. Huda, R. Sarno, and T. Ahmad, "Increasing accuracy of process-based fraud detection using a behavior model," *Int. J. Softw. Eng. its Appl.*, vol. 10, no. 5, pp. 175–188, 2016. https://doi.org/10.14257/ijseia.2016.10.5.16

[8] W. Chomyat and W. Premchaiswadi, "Process mining on medical treatment history using conformance checking," 2016. https://doi.org/10.1109/ictke.2016.7804102

[9] T. Erdogan and A. Tarhan, "Process Mining for Healthcare Process Analytics," 2016. https://doi.org/10.1109/iwsm-mensura.2016.027

[10]A. S. Osses, “Business process analysis in advertising: An extension to a methodology based on process mining projects,” 2016. https://doi.org/10.1109/sccc.2016.7836000
[11]W.Matthias, "Optimising Event Pattern Matching using Business Process Models," 2011.

LAMPIRAN

[Halaman ini sengaja dikosongkan]

DAFTAR ISTILAH

AND	:	relasi yang menunjukkan bahwa seluruh aktivitas pada relasi ini harus dijalankan semua di tiap proses.
Case	:	Suatu kasus tertentu pada *event log*. Kasus tertentu tersebut dapat berupa suatu kasus dalam memproduksi suatu barang tertentu, karena *event log* dapat terdiri dari catatan dari proses eksekusi pembuatan banyak barang atau proses eksekusi dari banyak kasus proses.
Case ID	:	Nomor identitas dari kasus tertentu pada *event log.*
Event	:	Aktivitas
Event Log	:	Suatu kumpulan eksekusi proses berdasarkan data aktivitas proses bisnis yang disimpan dalam bentuk tertentu.
Iterasi	:	Proses yang digunakan secara berulang-ulang
Node	:	unit dalam graf
OR	:	relasi yang menunjukkan bahwa seluruh aktivitas pada relasi tersebut dapat dijalankan semua atau dipilih beberapa untuk tiap proses
Process Discovery	:	pembentukan model proses
Process Mining	:	penggalian proses
Pseudo-code	:	Kode yang digunakan untuk menulis sebuah algoritma

		dengan cara bebas yang tidak terikat dengan bahasa pemrograman tertentu.
Skip activity	:	aktivitas yang hilang
SOP	:	*Standard Operating Procedure*
Trace	:	rangkaian dari aktivitas.
XOR	:	relasi yang menunjukkan bahwa seluruh aktivitas pada relasi ini harus dipilih salah satu untuk dijalankan

BIODATA PENULIS

Rizal Septiarakhman, lahir pada tanggal 23 September 1996 di Tegal. Penulis merupakan seorang mahasiswa yang sedang menempuh studi di Jurusan Teknik Informatika Institut Teknologi Sepuluh Nopember (ITS). Selama menempuh pendidikan di kampus, penulis juga aktif dalam organisasi kemahasiswaan, antara lain Staff Kewirausahaan Himpunan Mahasiswa Teknik Computer-Informatika (HMTC), Staff National Seminar of Technology (NST) Schematics ITS.

[Halaman ini sengaja dikosongkan]