TESIS – KI142502

PERBAIKAN PROSES BISNIS YANG TERPOTONG DALAM KASUS YANG MENGANDUNG AKTIVITAS TIDAK TERLIHAT DAN PILIHAN TIDAK BEBAS PADA TERMINAL PETI KEMAS

KELLY ROSSA SUNGKONO

05111650010045

DOSEN PEMBIMBING
Prof. Drs.Ec. Ir. Riyanarto Sarno, M.Sc.,Ph.D
19590803 198601 1 001

PROGRAM MAGISTER
BIDANG KEAHLIAN MANAJEMEN INFORMASI
JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI DAN KOMUNIKASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2017

TESIS – KI142502

RECOVERING TRUNCATED BUSINESS PROCESS IN THE CASE OF INVISIBLE TASKS AND NON-FREE CHOICE OF PORT CONTAINER TERMINAL

KELLY ROSSA SUNGKONO

05111650010045

SUPERVISOR
Prof. Drs.Ec. Ir. Riyanarto Sarno, M.Sc.,Ph.D
19590803 198601 1 001

MASTER PROGRAM
INFORMATION MANAGEMENT EXPERTISE
DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2017

Tesis disusun untuk memenuhi salah satu syarat memperoleh gelar
Magister Komputer (M.Kom.)
di
Institut Teknologi Sepuluh Nopember Surabaya

oleh:
Kelly Rossa Sungkono
Nrp. 05111650010045

Dengan judul :
Perbaikan Proses Bisnis yang Terpotong dalam Kasus yang Mengandung Aktivitas Tidak Terlihat dan Pilihan Tidak Bebas pada Terminal Petikemas

Tanggal Ujian : 28-12-2017
Periode Wisuda : 2017 Gasal

Disetujui oleh:

Prof.Ir.Drs.Ec. Riyanarto Sarno, M.Sc, Ph.D
NIP. 195908031986011001
(Pembimbing 1)

Tohari Ahmad, S.Kom, MIT, Ph.D
NIP. 197505252003121002
(Penguji 1)

Dr. Ir. Raden Venantius Hari Ginardi, M.Sc
NIP. 196505181992031003
(Penguji 2)

Bagus Jati Santoso, S. Kom., Ph. D.
NIP. 1986201711051
(Penguji 3)

Dekan Fakultas Teknik Informasi dan Komunikasi (FTIK),

Dr. H. Agus Zainal Arifin, S.Kom., M.Kom
NIP. 197208091995121001

PERBAIKAN PROSES BISNIS YANG TERPOTONG DALAM KASUS YANG MENGANDUNG AKTIVITAS TIDAK TERLIHAT DAN PILIHAN TIDAK BEBAS PADA TERMINAL PETIKEMAS

Nama mahasiswa : Kelly Rossa Sungkono
NRP : 05111650010045
Pembimbing : Prof. Drs.Ec. Ir. Riyanarto Sarno, M.Sc.,Ph.D.

ABSTRAK

Algoritma-algoritma pembentukan model proses akan berjalan maksimal apabila log data yang diolah adalah log data yang lengkap. Pada kenyataannya, terdapat proses-proses yang tidak lengkap diakibatkan pengambilan data pada interval waktu tertentu. Untuk memberikan hasil maksimal bagi algoritma, maka diperlukan perbaikan terhadap proses yang terpotong dengan menjadikan model proses sebagai acuan.

Model proses deklaratif terdiri dari peraturan-peraturan yang menjelaskan relasi proses secara fleksibel untuk mempermudah analisa dan modifikasi. Akan tetapi, model proses deklaratif tidak menggambarkan *control-flow pattern* secara mendetail, sehingga relasi proses tidak tergambar secara langsung. *MinerFul* adalah algoritma yang mengkonversi model deklaratif menjadi model proses imperatif, model yang menggambarkan *control-flow pattern*. Akan tetapi, *MinerFul* tidak dapat menggambarkan relasi *non-free choice* pada kondisi proses yang melibatkan *invisible task*. Kemudian, model proses imperatif tidak memperhatikan data tambahan yang disebut atribut, yang digunakan untuk penentuan aktivitas pengganti pada relasi pilihan dalam perbaikan proses terpotong.

Penelitian ini membentuk model proses imperatif *tree* yang mampu menggambarkan *non-free choice* dan *invisible task* dengan cara menggabungkan *control-flow pattern* yang terbentuk dalam *Linear Temporal Logic* (LTL). LTL dipilih karena ia merupakan bahasa formal dalam penggambarakn relasi komponen yang berkaitan dengan waktu dan LTL dapat dikonversi menjadi model *tree* yang merupakan representasi model proses imperatif. Kemudian, model proses imperatif akan diberi tambahan atribut dari data tambahan. Atribut adalah data selain aktivitas yang ditambahkan di log data, yang dalam penelitian ini adalah detail lampiran aktivitas. Data ini akan ditambahkan pada aktivitas di relasi pilihan model proses imperatif. Kemudian, proses terpotong akan diperbaiki dengan menggolongkan proses terpotong dan anomali dan pemberian aktivitas pengganti berdasarkan urutan *node* dengan kondisi khusus. Kondisi khusus adalah aktivitas pengganti pada relasi pilihan yang dipilih adalah aktivitas yang memiliki atribut yang sesuai dengan atribut pada proses terpotong.

Kemampuan penggambaran *non-free choice* dan *invisible task* pada model proses imperatif menghasilkan nilai presisi dan *simplicity* lebih tinggi dari hasil *MinerFul*, dengan rata-rata nilai presisi adalah 0,861 dan nilai *simplicity* adalah 0,878. Kemudian, hasil perbaikan proses yang terpotong dengan memperhatikan atribut menunjukkan bahwa akurasi lebih tinggi dibandingkan algoritma *Heuristic Linear Approach* terutama dalam proses yang terpotong di awal, dimana rata-rata akurasi yang didapat adalah 82,16%.

Kata kunci: aktivitas yang hilang, *invisible task*, *linear temporal logic* (LTL), model proses tree, *non-free choice*, proses yang terpotong

RECOVERING TRUNCATED BUSINESS PROCESS IN THE CASE OF INVISIBLE TASKS AND NON-FREE CHOICE OF PORT CONTAINER TERMINAL

Name	: Kelly Rossa Sungkono
Student ID	: 05111650010045
Supervisor	: Prof. Drs.Ec. Ir. Riyanarto Sarno, M.Sc.,Ph.D.

ABSTRACT

Process discovery algorithms run optimally if an event log is a complete log. In fact, there is a truncated process due to data retrieval at certain time intervals. To provide maximum results for the algorithms, recovering the truncated process is necessary.

A declarative process model consists of several rules that explain the relations of process flexibly. It leads to the ease in analyzing or modifying the process. However, a declarative model does not describe control-flow patterns in detail, so the relations of process are not drawn directly. MinerFul can convert a declarative model to an imperative model, a model that describes control-flow patterns. The disadvantage of MinerFul is an inability to describe non-free choice relations in the processes involving invisible tasks. Furthermore, the imperative model does not consider additional data called attributes, which are used for the determination of replacement activities of choice relations in the truncated process.

This research forms an imperative model, in the form model tree, that involves non-free choice and invisible tasks by combining control-flow pattern in Linear Temporal Logic (LTL). LTL is chosen because it is a formal language for describing the relation of components related to time and it can be converted into a tree model as the imperative model. Afterwards, the imperative model will be given additional attributes. Attribute is data, other than activity, added in the event log. In this research, attributes are detail attachments of activities in the event log. This data will be added to the activity in the choice relation of the imperative model. Then, the truncated processes will be corrected by classifying the truncated and anomalous and then providing replacement activity based on the order of nodes in the preorder and reverse preorder with a special condition. A special condition is a chosen activity on the choice relation is an activity that has an attribute that matches the attribute of the truncated process.

Non-free choice and invisible task capability by proposed method obtains higher precision and simplicity values than MinerFul results, with an average precision value of 0.861 and a simplicity value of 0.878. Then, the result of recovering truncated processes with respect to the attribute by proposed method shows higher accuracy than the result of Heuristic Linear Approach, especially in the truncated process at the beginning. The average accuracy of the recovered results by proposed method is 82.16%.

Keywords: invisible task, linear temporal logic (LTL), missing events, non-free choice, tree process model, truncated processes

(*halaman ini sengaja dikosongkan*)

KATA PENGANTAR

Puji dan syukur penulis panjatkan kehadirat Tuhan Yesus atas berkat dan rahmat -Nya sehingga penyusunan tesis ini dapat diselesaikan. Tesis ini dibuat sebagai salah satu syarat dalam menyelesaikan Program Studi Magister di Institut Teknologi Sepuluh Nopember Surabaya. Penulis menyadari bahwa tesis ini dapat diselesaikan karena dukungan dari berbagai pihak, baik dalam bentuk dukungan moral dan material.

Melalui kesempatan ini dengan kerendahan hati penulis mengucapkan terima kasih dan penghargaan setinggi-tingginya kepada semua orang untuk semua bantuan yang telah diberikan, antara lain kepada:

1. Ibu dan Bapak tercinta yang tiada henti selalu mendukung anaknya, tetap sabar mendengar keluhan anaknya, selalu mendoakan anaknya yang terbaik dan selalu menjadi panutan yang baik.
2. Bapak Prof. Drs.Ec. Ir. Riyanarto Sarno, M.Sc.,Ph.D selaku pembimbing yang memberikan arahan kepada penulis.
3. Teman seperjuangan, Yutika A. Effendi, Afina Lina, Abd. Charis F., Dewi Rahmawati, serta Andy dan Rey yang membantu dalam tesis ini. Terima kasih juga untuk dukungan dan doa sahabat serta teman-teman lainnya.
4. Bapak Daniel Oranovora Siahaan, S.Kom., M.Sc., PD.Eng, Bapak Dr. Ir. Raden Venantius Hari Ginardi, M.Sc, serta Bapak Bagus Jati Santoso, S.Kom., Ph.D. sebagai tim Penguji Tesis yang memberikan masukan dan kritik yang membangun untuk Tesis ini.

Akhirnya dengan segala kerendahan hati penulis menyadari masih banyak terdapat kekurangan pada tesis ini. Oleh karena itu, segala tegur sapa dan kritik yang sifatnya membangun sangat penulis harapkan demi kesempurnaan Tesis ini.

Surabaya, Desember 2017

Penulis

(*halaman ini sengaja dikosongkan*)

DAFTAR ISI

Halaman

(halaman ini sengaja dikosongkan)

DAFTAR ISTILAH

AND	:	relasi yang menunjukkan bahwa seluruh aktivitas pada relasi ini harus dijalankan semua.
Case	:	proses
Control-flow pattern	:	bentuk potongan model prose yang menggambarkan relasi tertentu di model proses imperatif
Event log	:	Log Data
Fitness	:	nilai kecocokan model proses
Invisible Task	:	aktivitas tidak terlihat (aktivitas yang tidak terdapat dalam log data, tetapi dibutuhkan di model proses)
Non-free choice	:	pilihan tidak bebas
OR	:	relasi yang menunjukkan bahwa seluruh aktivitas pada relasi tersebut dapat dijalankan semua atau dipilih beberapa untuk tiap proses
Node	:	unit dalam graf atau model *tree*
Simplicity	:	nilai kualitas kesederhanaan model
Process Discovery	:	pembentukan model proses
Process Mining	:	penggalian proses
Running Time	:	waktu yang digunakan algoritma dalam suatu program untuk mendapatkah hasil sesuatu
Skip activity	:	aktivitas yang hilang
SOP	:	*Standard Operating Procedure*
Trace	:	varian proses pada log data
Truncated Process	:	proses yang terpotong
XOR	:	relasi yang menunjukkan bahwa seluruh aktivitas pada relasi ini harus dipilih salah satu untuk dijalankan
YAWL	:	*Yet Another Workflow Language*

(halaman ini sengaja dikosongkan)

DAFTAR GAMBAR

Halaman

DAFTAR TABEL

Halaman

(*halaman ini sengaja dikosongkan*)

BAB 1.
PENDAHULUAN

1.1 Latar Belakang

Log data berisi informasi mengenai bisnis proses yang berjalan (Wil M P Van Der Aalst, 2011). 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 (Burattin, Sperduti, & van der Aalst, 2012). 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 *heuristic* (percobaan berdasarkan data yang ada untuk mencari hasil yang mendekati solusi terbaik) serta membangun model proses sebagai acuan perbaikan. Pendekatan *heuristic* bisa dilakukan dengan cara mencari relasi aktivitas yang memiliki probabilitas tertinggi sebagai relasi perbaikan proses yang terpotong. Pendekatan *heuristic* 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.

Model deklaratif (Maggi, Mooij, & Van Der Aalst, 2011) adalah model yang menggambarkan relasi proses secara implisit. Relasi implisit adalah relasi proses yang tidak menggambarkan penanda kondisi sekuensial atau paralel, seperti XOR, OR, dan AND, akan tetapi menggambarkan ciri dari relasi tersebut. Penggambaran relasi pada

model deklaratif membuat model ini menjadi fleksibel, sehingga analis dapat menganalisa atau memodifikasi lebih mudah. Sebagai contoh dari fleksibilitas model deklaratif, relasi XOR dapat digambarkan dengan relasi *chain_response* dan relasi *exclusive_choice* sedangkan relasi AND dapat digambarkan hanya dengan relasi *chain_response*. Apabila analis ingin memodifikasi model dari relasi XOR menjadi relasi AND, analis cukup menyembunyikan relasi *exclusive_chain* dari model dan apabila relasi XOR kembali dibutuhkan, maka relasi *exclusive_chain* dapat ditampilkan kembali. Akan tetapi, penggambaran secara implisit ini akan menyulitkan perbaikan proses yang terpotong karena metode perbaikan membutuhkan relasi yang digambarkan secara eksplisit untuk menentukan aktivitas yang akan dipilih.

Algoritma *MinerFul* adalah algoritma yang dapat mengkonversi model deklaratif menjadi model proses dengan relasi yang tergambar secara eksplisit (model proses imperatif). Terdapat berbagai macam bentuk model proses imperatif, seperti YAWL, Petri Net, model proses *tree*. Algoritma *MinerFul* menggambarkan dalam bentuk Petri Net. Kelemahan dari algoritma ini adalah ketidakmampuan membentuk *invisible task* dan relasi *non-free choice*. *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 melewati, berpindah atau mengulang.

Kemudian, dari sisi perbaikan proses yang terpotong, terdapat beberapa algoritma perbaikan proses tidak lengkap telah diusulkan, yaitu *Heuristic Linear Approach* (Song, Xia, Jacobsen, Zhang, & Hu, 2015) dan algoritma *Repair Logs* (Wang, Song, Zhu, Lin, & Sun, 2016). Kedua algoritma tersebut memanfaatkan model proses untuk menentukan aktivitas pengganti dari aktivitas yang hilang, dimana pembentukan model proses menggunakan algoritma lain. Pada algoritma *Heuristic Linear Approach*, model proses yang digunakan adalah model proses dalam bentuk Petri Net yang dapat dibentuk menggunakan algoritma Alpha. 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.

Melihat permasalahan-permasalahan tersebut, penelitian ini mengusulkan metode untuk perbaikan proses yang terpotong pada log data yang mengandung *non-free choice* dan *invisible tasks* dengan membentuk model proses acuan. Model proses yang dibentuk adalah model prsoses yang dibentuk dari penggabungan *control-flow pattern* dalam bentuk *Linear Temporal Logic* (LTL). Dalam hal ini, LTL (Wil M.P. Van der Aalst, De Beer, & Van Dongen, 2005) 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. *Control-flow pattern* akan dibentuk dari hubungan *rules* pada model deklaratif. Pembentukan model deklaratif menggunakan algoritma *Declare Miner*.

Bentuk model proses yang digunakan sebagai bentuk penggabungan *control-flow pattern* adalah model proses *tree*. Model ini dipilih karena model ini banyak digunakan untuk menggambarkan penguraian formula matematika, yang memiliki kemiripan dengan formula LTL, sehingga mempermudah penggabungan *control-flow pattern* dalam bentuk LTL. Model *tree* tersebut akan digunakan untuk mendeteksi proses yang terpotong dan anomali serta memperbaiki proes yang terpotong. Selain itu, atribut dari log data ditambahkan pada model *tree* untuk menambah acuan dalam penentuan aktivitas pengganti di relasi pilihan pada saat perbaikan proses yang terpotong.

Kemudian, perbaikan proses yang terpotong akan dilakukan dengan menambahkan aktivitas pengganti dari aktivitas yang hilang sesuai dengan urutan aktivitas dalam *node* pada model *tree*. Pada proses yang terpotong di bagian akhir, urutan *node* dalam bentuk *pre-order*, dan proses yang terpotong di bagian awal menggunakan model urutan *reverse* dari *pre-order*. Akan tetapi, aktivitas yang dipilih tidak hanya berdasarkan urutan melainkan berdasarkan *node* operator pada model *tree*. *Node* operator akan menentukan aktivitas yang dipilih. Hasil dari penelitian ini adalah

proses terpotong yang telah diperbaiki. Dengan adanya perbaikan proses yang terpotong ini, maka log data mengandung proses-proses yang lengkap sehingga algoritma-algoritma penggalian proses dapat melakukan analisa dengan baik menggunakan log data tersebut.

1.2 Perumusan Masalah

Dari latar belakang diatas didapatkan permasalahan sebagai berikut :

1. Bagaimana metode usulan membentuk model proses imperatif yang dapat menggambarkan *non-free choice* dan *invisible task* dari *rules* pada model deklaratif?
2. Bagaimana metode usulan memperbaiki proses yang terpotong?
3. Bagaimana hasil evaluasi metode usulan dalam pembentukan model proses imperatif yang mengandung *non-free choice* dan *invisible task*?
4. Bagaimana hasil evaluasi metode usulan dalam perbaikan proses yang terpotong?

1.3 Tujuan dan Manfaat

1.3.1 Tujuan

Tujuan dari penelitian ini adalah membangun model proses dari kumpulan *control-flow patterns* yang dibentuk dari model deklaratif, membedakan proses yang terpotong dan anomali, serta memperbaiki proses yang terpotong dari log data.

1.3.2 Manfaat

Penelitian ini diharapkan dapat membentuk model proses dari kumpulan *control-flow patterns* yang dibentuk dari model deklaratif serta memperbaiki proses yang terpotong sehingga meghasilkan hasil yang lengkap dan analis dapat menganalisa dengan baik menggunakan algoritma-algoritma penggalian proses.

1.3.3 Kontribusi Penelitian

Kontribusi dari penelitian ini adalah perbaikan proses yang terpotong dengan dengan membangun model proses yang merupakan gabungan *pattern* yang dibentuk dalam notasi LTL serta membedakan proses yang mengandung anomali dengan proses

yang terpotong. Untuk membedakan proses, penelitian akan memanfaatkan model proses yang terbentuk dari *control-flow pattern* dalam bentuk LTL. Kemudian, untuk memperbaiki proses, terdapat tiga tahapan. Tahapan pertama adalah pembentukan *pattern* berdasarkan *rules* pada model deklaratif. Tahapan kedua, *pattern* tersebut akan digabungkan dalam bentuk model proses *tree* yang digunakan sebagai acuan perbaikan proses yang terpotong dan juga penambahan atribut dalam model proses *tree* untuk bahan acuan dalam pemilihan aktivitas pada relasi pilihan. Tahapan ketiga adalah pembedaan anomali dan proses yang terpotong serta perbaikan proses yang terpotong berdasarkan model proses yang dibentuk.

1.3.4 Batasan Masalah

Untuk memfokuskan permasalahan penelitian ini, batasan masalah yang ditentukan adalah sebagai berikut:

- 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.
- Penelitian ini tidak membedakan macam-macam anomali yang muncul.
- Proses yang terpotong pada kasus ini karena pengambilan data dalam interval waktu tertentu.
- Pemilihan proses yang terpotong untuk diperbaiki tidak memperhitungkan adanya anomali pada aktivitas yang akan datang.
- Log data yang digunakan dalam penelitian memiliki data tambahan selain aktivitas, waktu dan pelaku untuk digunakan sebagai atribut.

(*halaman ini sengaja dikosongkan*)

BAB 2.
KAJIAN PUSTAKA DAN DASAR TEORI

2.1 Kajian Pustaka

Kajian Pustaka yang telah dilakukan adalah dengan melakukan ujicoba penelitian sebelumnya dan melakukan kajian terhadap literatur yang relevan. Penelitian yang dilakukan adalah pembentukan model proses dari model deklaratif sebagai acuan dalam perbaikan proses yang terpotong. Pembentukan model proses dilakukan dengan menggabungkan *control-flow pattern* yang dibentuk dari hubungan antar *rules* pada model deklaratif. Pembentukan model proses dari model deklaratif memberikan dua tipe model, yaitu model proses secara garis besar yang dipaparkan oleh model deklaratif dan model dengan relasi aktivitas secara rinci yang dipaparkan oleh model proses yang dibentuk pada penelitian ini.

Algoritma *MinerFul* (Di Ciccio, Schouten, De Leoni, & Mendling, 2015) adalah algoritma yang mampu mengkonversi model deklaratif menjadi model proses dengan relasi eksplisit (model proses imperatif). Algoritma ini mencari hubungan *rules* pada model deklaratif untuk membentuk relasi sekuensial dan relasi paralel XOR, AND. Algoritma ini menggambarkan model proses dari model deklaratif dalam bentuk Petri Net. Kelemahan dari algoritma *MinerFul* adalah ketidakmampuan untuk menggambarkan relasi *non-free choice* dan *invisible task*.

Untuk perbaikan proses yang memiliki aktivitas hilang, terdapat dua penelitian tentang itu, yaitu *Heuristic Linear Approach* (Song et al., 2015) dan algoritma *Repair Logs* (Wang et al., 2016). Algoritma *Repair Logs* (Wang et al., 2016) adalah metode yang memperbaiki proses yang memiliki aktivitas yang hilang berdasarkan model proses SOP (Prosedur Standar Operasi) dengan mencari jumlah aktivitas pengganti terkecil. Algoritma ini tidak hanya memperbaiki aktivitas yang hilang, akan tetapi juga mengestimasi waktu ekseskusi dari aktivitas pengganti aktivitas yang hilang tersebut. Akan tetapi, algoritma ini memiliki kesulitan dalam menentukan aktivitas pengganti apabila terdapat lebih dari satu kemungkinan pilihan (yang terjadi pada relasi pilihan XOR atau OR) yang memiliki jumlah aktivitas yang sama. Algoritma *Heuristic Linear Approach* adalah metode yang memiliki kinerja yang sama dengan algoritma *Repair*

Logs, yaitu menggunakan model proses SOP untuk memprediksi aktivitas pengganti dari aktivitas yang hilang. Metode ini mengembangkan algoritma *Repair Logs* dengan memanfaatkan probabilitas kemunculan rangkaian aktivitas pengganti pada log data apabila terdapat lebih dari satu kemungkinan pilihan. Akan tetapi, pengunaan probabilitas ini memiliki kendala apabila proses yang memiliki aktivitas hilang (dalam hal ini proses terpotong) merupakan proses yang tidak banyak muncul dalam proses lengkap di log data. Pengujian yang dilakukan hanya menggunakan algoritma *Heuristic Linear Approach* karena algoritma ini mengembangkan algoritma *Repair Logs* dengan memperhatikan proses yang memiliki probabilitas tinggi pada saat perbaikan proses yang memiliki aktivitas hilang.

Melihat permasalahan yang timbul dari algoritma-algoritma yang ada sebelumnya, penelitian ini akan membangun metode untuk membentuk model proses dari model deklaratif yang dapat menangani *non-free choice* dan *invisible task* serta metode perbaikan proses yang terpotong berdasarkan model proses yang dibentuk dengan tambahan keterangan atribut pada model proses serta pengelompokkan proses yang terpotong dengan anomali. Penambahan keterangan atribut diusulkan untuk memberikan tambahan acuan dalam pemilihan aktivitas pengganti pada relasi pilihan dan pengelompokkan proses yang terpotong dengan anomali diusulkan untuk memberikan proses lengkap yang tidak mengandung *noise* (data yang salah).

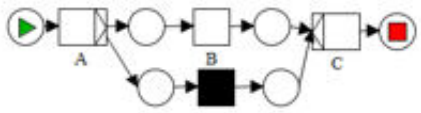
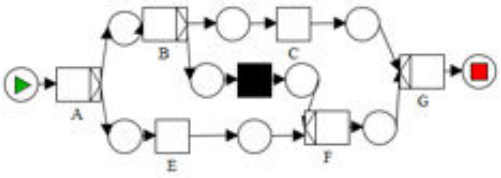
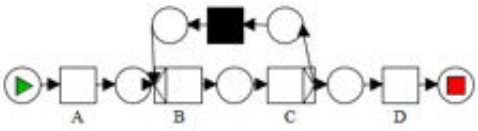
2.2 Dasar Teori

2.2.1 *Invisible Task*

Invisible Task (Aktivitas Tak Terlihat) 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 prime task* yaitu Melewati, Mengulang, dan Berpindah.

Invisible task tipe Melewati digunakan untuk menggambarkan aktivitas yang dapat dilewati. *Invisible task* tipe melewati baru terjadi apabila kondisi aktivitas yang

dilewati tersebut lebih dari 5% kemunculan di log data. Hal ini untuk membedakan proses dengan kondisi melewati dan anomali berupa *skip sequence.* Pemberian *threshold* 5% dikarenakan anomali adalah kejadian yang tidak sesuai dengan proses yang seharusnya dan kemunculannya sangat jarang dalam log data. *Invisible task* tipe Mengulang digunakan untuk menggambarkan aktivitas yang dapat diulang. *Invisible task* tipe Berpindah digunakan untuk perpindahan eksekusi pada beberapa percabangan. Contoh dari *invisible task* dalam bentuk YAWL digambarkan pada Gambar 2.1. Lingkaran hijau digunakan sebagai awal dan lingkaran merah digunakan sebagai akhiran. *Split pattern* hanya memiliki lingkaran hijau dan *Join pattern* memiliki lingkaran merah untuk menunjukkan bahwa *split pattern* harus ditempatkan pada awal *join pattern* untuk menggambarkan hubungan paralel. Kotak yang berwarna hitam merupakan *invisible task.*

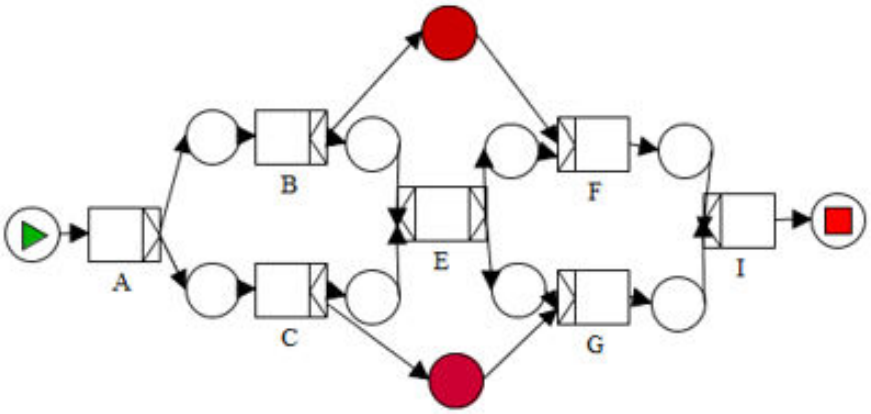
Condition	Event Log	Process Model (YAWL)
Melewati	[ABC], [AC], [AC]	
Berpindah	[ABCG], [ABFG], [AEFG], [AEFG]	
Mengulang	[ABCBCD], [ABCBCBCD]	

Gambar 2.1 Contoh *Invisible Task*

Penelitian ini mengkonstruksi *control-flow pattern* berdasarkan template deklarasi (*rules*) sebagai hasil algoritma Declare Miner. Sejalan dengan template deklarasi yang terbentuk dalam Linear Temporal Logic (LTL), penelitian ini mengusulkan *control-flow patterns* pada Linear Temporal Logic (LTL).

2.2.2 *Non-Free Choice*

Non-Free Choice adalah relasi tambahan pada model proses untuk menggambarkan aktivitas pilihan yang harus dijalankan setelah aktivitas tertentu telah dijalankan. Contoh sederhana dari *non-free choice* ditunjukkan pada relasi aktivitas A ke aktivitas I dan aktivitas B ke aktivitas F pada Gambar 2.2. Dari gambar tersebut, dapat dilihat bahwa sebenarnya aktivitas F dan aktivitas G adalah aktivitas pilihan, akan tetapi pengeksekusian kedua aktivitazs tersebut bergantung pada aktivitas tertentu.



Gambar 2.2 Model Proses yang memiliki relasi *non-free choice*

Algoritma pertama yang berfokus dalam menampilkan non-free choice di model proses adalah algoritma Alpha++ (Wen, van der Aalst, Wang, & Sun, 2007). Algoritma ini telah diimplementasikan pada kakas bantu bernama ProM dan aturan pemunculan non-free choice dari algoritma Alpha++ telah diadaptasi oleh algoritma lainnya, yaitu algoritma Alpha$ (Guo, Wen, Wang, Yan, & Yu, 2015).

2.2.3 Proses Yang Terpotong dan Anomali

Baik buruknya kualitas log data dipengaruhi oleh beberapa hal. Permasalahan-permasalahan utama yang menyebabkan menurunnya kualitas log data adalah (Bose, Mans, & van der Aalst, 2013):

- *Missing data* : terdapat beberapa informasi (seperti aktivitas, waktu dan atribut) yang tidak terekam dalam log data.

- *Incorrect data* : terdapat beberapa informasi yang salah terekam, seperti aktivitas yang salah, relasi antar aktivitas yang salah, atau nilai atribut yang salah.
- *Imprecise data* : terdapat beberapa informasi yang tidak dijelaskan secara spesifik pada log data sehingga menyulitkan penggalian informasi dari data tersebut.
- *Irrelevant data* : terdapat beberapa informasi yang tidak sesuai untuk digunakan oleh algoritma penggalian proses, sehingga untuk mengatasi permasalahan ini dilakukan *filtering* terlebih dahulu pada log data.

Salah satu penyebab dari adanya *missing data* adalah proses yang terpotong. Proses yang terpotong ini adalah proses yang aktivitas-aktivitas awal atau aktivitas-aktivitas akhir hilang pada log data akan tetapi terjadi pada kenyataannya. Contohnya pada proses ABC. Aktivitas A dan B berjalan pada tanggal 30 September dan aktivitas C berjalan pada tanggal 1 Oktober. Apabila algoritma penggalian proses mengolah log data bulan September, aktivitas C tidak ikut dalam log yang diolah sehingga proses ini termasuk dalam proses yang terpotong. Penelitian ini akan menggambarkan formalitas dari proses yang terpotong. Formalisasi tersebut dapat dilihat pada Gambar 2.3.

Proses yang Terpotong (*Truncated Process*)

Jika T_{SOP} adalah proses pada model proses, T_{trunc} adalah proses yang terpotong, serta k dan l adalah aktivitas dimana $k(0)$ adalah aktivitas awal dan $k(t)$ adalah aktivitas akhir, maka proses yang terpotong terjadi apabila $k \in T_{trunc}, l \in T_{SOP}, k(0) \neq l(0) \vee k(t) \neq l(t) \wedge k(0), k(t) \in T_{SOP}$

Gambar 2.3 Formalisasi Proses yang Terpotong

Perbedaan dari proses yang terpotong dan anomali, berupa *skip acitivity* terletak pada keberadaan *missing data*. Proses yang terpotong adalah proses yang mengalami *missing data* pada bagian akhir dan awal, akan tetapi tidak mengalami *missing data* pada bagian tengah proses. Sedangkan, anomali berupa *skip activity* adalah proses yang

memiliki *missing data* pada bagian tengah proses. Kemudian, perbedaan proses yang terpotong dengan anomali berupa *wrong pattern* adalah proses yang memiliki alur aktivitas yang terbalik atau tidak sesuai dengan model proses. Apabila algoritma perbaikan proses yang terpotong hanya memperhatikan aktivitas awal dan akhir, maka proses yang termasuk *wrong pattern* bisa dikategorikan sebagai proses yang terpotong. Contoh dari proses yang terpotong, *skip sequence*, dan *wrong pattern* dapat dilihat pada Tabel 2.1. Pada contoh proses yang lengkap, aktivitas A sampai D dilakukan pada bulan Januari, akan tetapi aktivitas E, F, dan G dilakukan pada bulan Februari. Proses yang terbentuk dalam proses terpotong, *skip sequence,* dan *wrong pattern* adalah proses yang terjadi di bulan Januari.

Tabel 2.1 Contoh Proses Terpotong, *Skip Sequence,*dan *Wrong Pattern*

Jenis Proses	**Contoh Proses**
Proses Lengkap	A, B, C, D, E, F, G
Proses Terpotong	A, B, C, D
Skip Sequence	A, D
Wrong Pattern	A, C, B, D

2.2.4 Algoritma *Heuristic Linear Approach*

Heuristic Linear Approach adalah algoritma yang digunakan untuk memperbaiki proses yang memiliki aktivitas hilang dengan memanfaatkan model proses Petri Net. Algoritma yang membentuk model proses dalam bentuk Petri Net adalah algoritma Alpha. Algoritma *Heuristic Linear Approach* membentuk *traces* dari model proses dan kemudian memilih aktivitas pengganti untuk aktivitas yang hilang dengan mencari jumlah aktivitas pengganti paling kecil serta probabilitas kemunculan aktivitas pengganti yang besar di log data.

Langkah-langkah dari *Heuristic Linear Approach* adalah sebagai berikut:

1. Model proses dipecah menjadi *traces* (rangkaian aktivitas yang dapat dibentuk dari model proses) sebagai acuan dalam perbaikan proses.
2. Apabila aktivitas awal *case* bukan merupakan aktivitas awal di model proses, maka aktivitas pengganti adalah rangkaian aktivitas sebelum aktivitas awal *case* yang

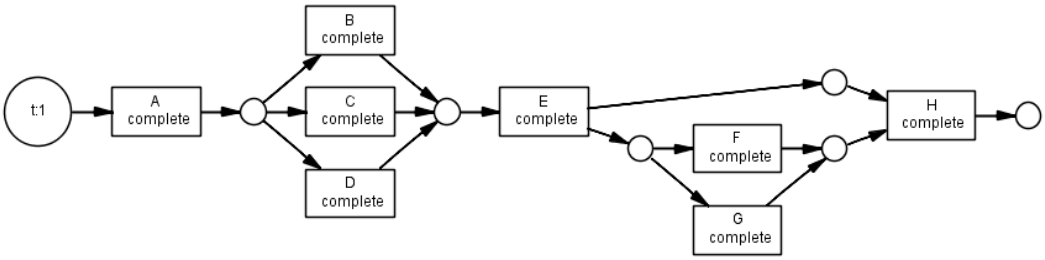
memiliki jumlah aktivitas paling sedikit atau jumlah probabilitas kemunculan setiap aktivitas rangkaian tersebut terhadap log data adalah jumlah terbesar.

3. Apabila aktivitas akhir *case* bukan merupakan aktivitas akhir di model proses, maka aktivitas pengganti adalah rangkaian aktivitas setelah aktivitas akhir *case* yang memiliki jumlah aktivitas paling sedikit atau jumlah probabilitas kemunculan setiap aktivitas rangkaian tersebut terhadap log data adalah jumlah terbesar.
4. Kemudian, dimulai dari aktivitas pertama pada *case*, apabila aktivitas tersebut dan aktivitas setelahnya di *case* tidak tergambar di semua *traces* acuan, maka *traces* yang digunakan sebagai acuan adalah *traces* yang mengandung kedua aktivitas tersebut dimana aktivitas setelah tidak muncul sebelum aktivitas tersebut di *traces*. Rangkaian aktivitas yang dipilih adalah rangkaian aktivitas sebelum aktivitas awal *case* yang memiliki jumlah aktivitas paling sedikit atau jumlah probabilitas kemunculan setiap aktivitas rangkaian tersebut terhadap log data adalah jumlah terbesar.

Dengan menggunakan probabilitas kemunculan aktivitas serta jumlah aktivitas, algoritma *Heuristic Linear Approach* memiliki kendala, yaitu:

1. Kesulitan dalam Penentuan Aktivitas Pengganti pada relasi *non-free choice* yang mengandung *invisible task*

   Pembentukan *non-free choice* pada relasi yang mengandung *invisible task* menjadi kesulitan dalam algoritma Alpha. Ketidakmampuan pembentukan *non-free choice* ini akan berpengaruh pada perbaikan proses dalam algoritma *Heuristic Linear Approach*. Sebagai contoh, model yang menjadi acuan algoritma *Heuristic Linear Approach* tergambar dalam Gambar 2.4, dimana log data berisi *trace* [A, B, E, F, H] yang muncul sebanyak 4 kali, *trace* [A, C, E, G, H] yang muncul sebanyak 6 kali, *trace* [A, D, E, H] sebanyak 10 kali.



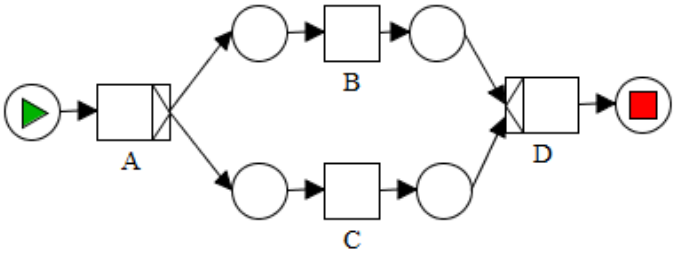
Gambar 2.4 Model Proses Acuan Algoritma *Heuristic Linear Approach*

Terdapat *case* [A, B] yang merupakan proses yang terpotong di bagian akhir dimana *case* lengkapnya adalah [A, B, E, F, H]. Berdasarkan algoritma *Heuristic Linear Approach*, terdapat 3 *trace* yang menjadi acuan dari 9 *trace* yang terbentuk berdasarkan model proses. Tiga *trace* tersebut adalah [A, B, E, F, H], [A, B, E, G, H], dan [A, B, E, H]. Karena rangkaian dengan jumlah aktivitas paling sedikit dan juga probabilitas kemunculan tiap aktivitas besar adalah rangkaian [E, H], maka *case* [A, B] akan diperbaiki menjadi [A, B, E, H]. Berdasarkan *case* lengkap, hasil dari algoritma *Heuristic Linear Approach* tidak benar. Apabila relasi *non-free choice* antara aktivitas B dan F tergambar, maka *trace* yang dijadikan acuan hanya [A, B, E, F, H] sehingga hasil perbaikan proses akan menjadi benar. Oleh karena itu, pembentukan model proses yang mampu menangani relasi *non-free choice* dalam relasi yang mengandung *invisible task* diperlukan.

2. Kesulitan dalam Penentuan Aktivitas Pengganti yang Memiliki Probabilitas Rendah

Terdapat *case* [A] yang memiliki aktivitas hilang di bagian akhir, dimana *case* lengkapnya adalah [A, C, D]. Terdapat model proses seperti pada Gambar 2.5 dengan log berisi *trace* [A, B, D] yang muncul sebanyak 12 kali dan *trace* [A, C, D] yang muncul sebanyak 8 kali. Dengan menggunakan algoritma *Heuristic Linear Approach*, terdapat dua rangkaian aktivitas sebagai pilihan, yaitu [B, D] dan [C, D]. Probabilitas [B, D] adalah probabilitas kemunculan B + probabilitas kemunculan D = 0,6 + 1 = 0,8 dan probabilitas [C, D] adalah 0,4 + 1 = 0,7. Dikarenakan algoritma *Heuristic Linear Approach* memilih probabilitas terbesar, maka [B, D] dipilih sehingga hasil perbaikan adalah [A, B, D]. Jika dibandingkan dengan *case* lengkap, hasil tersebut adalah salah. Oleh karena itu, diperlukan informasi lain untuk acuan

perbaikan proses, dimana dalam penelitian ini menggunakan atribut (tambahan informasi pada log data selain aktivitas).



Gambar 2.5 Model Proses Acuan yang Mengandung Relasi XOR

3. Tidak membedakan Proses Terpotong dan Anomali dalam Perbaikan Proses

Tabel 2.2 menunjukkan contoh dari proses yang lengkap dan Tabel 2.3. Apabila proses lengkap merupakan *trace* yang digunakan sebagai acuan, maka proses yang terpotong akan diperbaiki menjadi [A, B, C, D, E, F].

Untuk anomali, berdasarkan langkah ke-empat dari algoritma *Heuristic Linear Approach*, maka aktivitas C ditambahkan diantara aktivitas B dan D, dan berdasarkan langkah ke-tiga maka rangkaian aktivitas [D, E, F] ditambahkan di akhir proses. Hasil akhir dari perbaikan tersebut adalah [A, B, C, D, C, D, E, F]. Jika dikomparasi dengan proses lengkap pada Tabel 2.2, hasil perbaikan anomali ini tidak benar dan akan memberikan hasil yang salah apabila diikutkan sebagai bahan analisa proses. Oleh karena itu, diperlukan pemisahan proses terpotong dan anomali sebelum algoritma *Heuristic Linear Approach* dijalankan, sehingga proses yang diperbaiki adalah proses yang terpotong.

Tabel 2.2 Contoh Kasus Proses

Proses Lengkap		
Case ID	**Activity**	**Time**
440	A	1/31/2017 19:20:05
440	B	1/31/2017 21:35:10
440	C	1/31/2017 22:43:00
440	D	1/31/2017 23:59:13
440	E	2/01/2017 02:03:56
440	F	2/01/2017 04:00:00

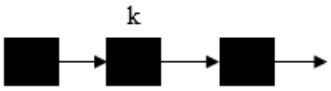
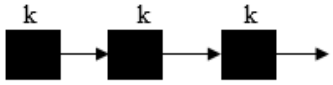
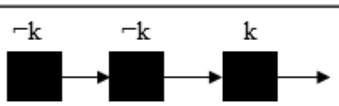
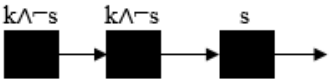
Tabel 2.3 Contoh Proses Terpotong dan Anomali berdasarkan Tabel 2.2

Jenis Proses	***Case***
Proses Terpotong	A B C D
Anomali (*Wrong Pattern*)	A B D C

2.2.5 *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 (¬, ∨, ∧, →) dan operator modal temporal. Terdapat empat operator modal temporal yang digunakan pada *Linear Temporal Logic*. Penjelasan mengenai operator tersebut dapat dilihat pada Tabel 2.4.

Tabel 2.4 Operator *Linear Temporal Logic* (LTL)

Tekstual	**Simbol**	**Penjelasan**	**Diagram**
Xk	Ok	k harus dilaksanakan sebagai *state* selanjutnya	
Gk	□k	k harus terjadi pada seluruh *state* selanjutnya	
Fk	◊k	k berada pada *state* selanjutnya	
kUs	kUs	k harus dilaksanakan sampai s muncul	

Ilustrasi penggunaan operator diberikan untuk memperjelas pemahaman. Contoh operator pertama adalah Registration -> X (Go In The Venue), Dikatakan bahwa setiap pengunjung harus pergi ke tempat acara tepat setelah mendaftar. Contoh operator kedua adalah G ((Registration -> X (Go to The Venue)). Dikatakan bahwa registrasi aktivitas dan kemudian pergi ke venue bisa terjadi pada proses pertama, pada proses tengah, atau akhir proses. Contoh operator ketiga adalah Registration -> F (Enter the Meeting Room). Dikatakan bahwa setelah melakukan registrasi, setiap pengunjung dapat melakukan aktivitas lain sebelum masuk ruang pertemuan atau langsung masuk

ke ruang pertemuan. Contoh dari operator terakhir adalah Registration U Event Opening. Pernyataan ini memberitahukan bahwa pendaftaran akan terus dibuka sampai acara dimulai.

2.2.6 *Control-Flow Patterns*

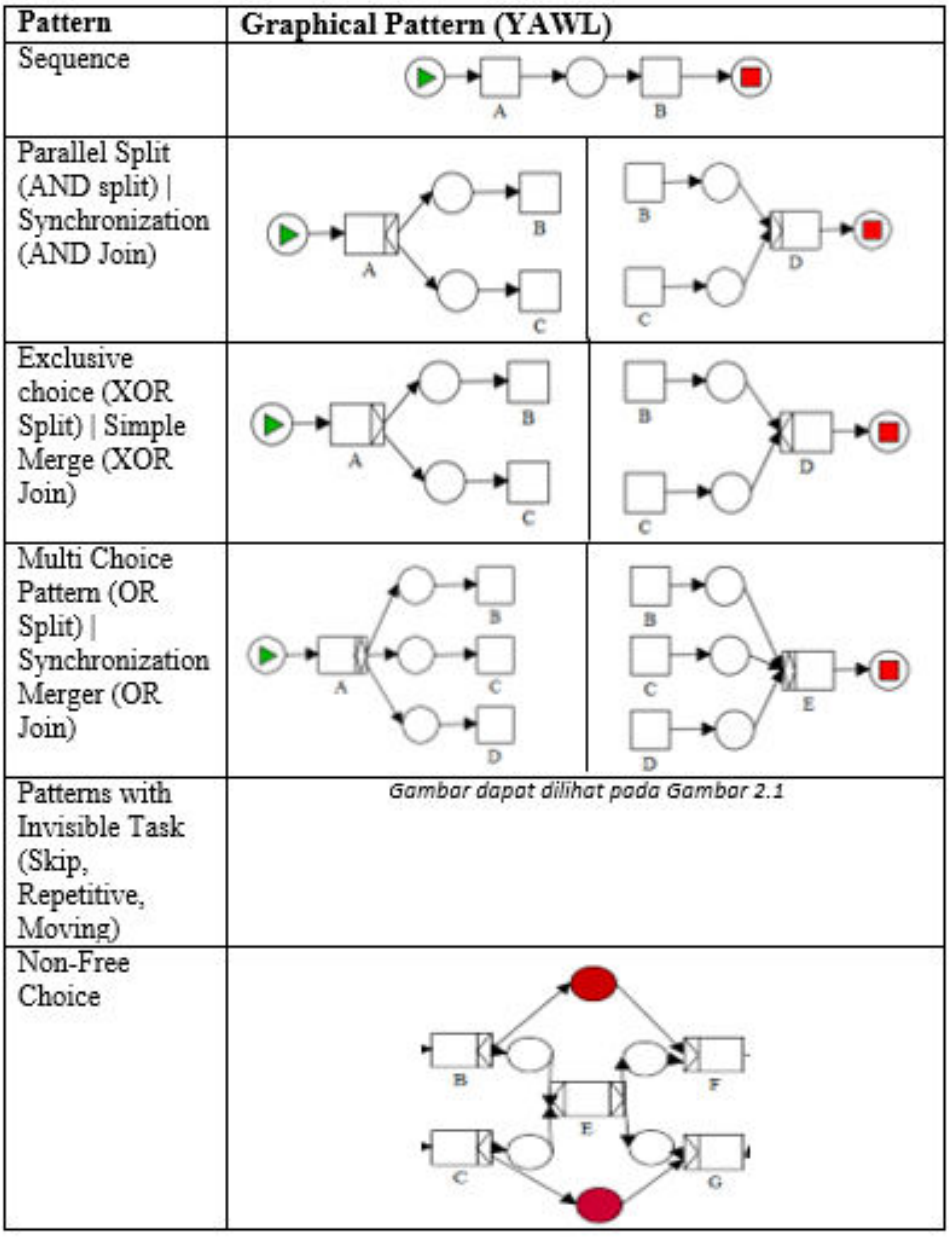
Standarisasi *pattern* yang menggambarkan pengekseskusian aktivitas dalam relasi sekuensial dan paralel telah banyak didefinisikan (Mulyar, Russell, ter Hofstede, & van der Aalst, 2006; Russell, N., ter Hofstede, A.H.M, van der Aalst, W.M.P, Mulyar, 2006). Dari dua puluh *pattern* yang diperkenalkan, ada beberapa *pattern* yang sering diterapkan pada model proses. *Pattern* tersebut adalah *parallel split, exclusive choice, multi-choice, sequence, simple merge, synchronization*, and *synchronization merge*.

Sequence pattern menunjukkan aktivitas yang berjalan berurutan. *Parallel split* dan *synhronization* digunakan untuk menggambarkan aktivitas secara paralel atau menggambarkan AND relation dalam model proses. *Simple merge* dan juga *executive choice* digunakan untuk menandakan pelaksanaan salah satu kegiatan pilihan atau menggambarkan hubungan XOR dalam model proses. Kemudian, *multi-choice* dan *synchronization merge* digunakan untuk menggambarkan pelaksanaan lebih dari satu aktivitas pilihan atau menggambarkan hubungan OR dalam model proses. Sekumpulan *pattern* tersebut, selain digunakan sebagai potongan model, berfungsi sebagai pemeriksa kesalahan pada model proses yang digambarkan dari log data dengan membangun *pattern* dari model proses Operasional Prosedur Standar (SOP) (Sarno, Wibowo, Kartini, Effendi, & Sungkono, 2016).

Selain *pattern* yang sudah umum, pada penelitian ini ditambahkan dua *pattern*, yaitu *pattern* untuk relasi *non-free choice* dan *invisible task*. Kesemua *pattern* tersebut digambarkan dalam format YAWL yang dapat dilihat pada Gambar 2.6. Lingkaran hijau dilambangkan sebagai awal dari model proses dan lingkaran merah dilambangkan sebagai akhir dari model proses. Pemberian lingkaran hijau pada *split pattern* dan pemberian lingkaran merah pada *join pattern* untuk menunjukkan bahwa *split pattern* harus ditempatkan sebelum *join pattern* dalam menggambarkan hubungan paralel. *Pattern* yang menggambarkan relasi *non-free choice* tidak memiliki lingkaran hijau atau

lingkaran merah karena *pattern* ini perlu digabung dengan *pattern* lain untuk menggambarkan relasi utuh.

Penelitian ini membentuk *control-flow pattern* berdasarkan *rules* pada model deklaratif sebagai hasil dari algoritma Declare Miner (Maggi et al., 2011). Sejalan dengan *rules* pada model deklaratif yang dibangun dalam Linear Temporal Logic (LTL), penelitian ini akan membentuk *control-flow pattern* dalam bentuk Linear Temporal Logic (LTL).



Gambar 2.6 *Control-Flow Pattern*

2.2.7 Algoritma *Declare Miner*

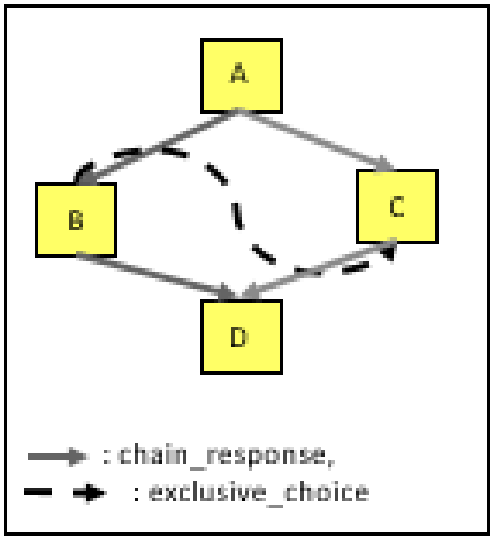
Algoritma *Declare Miner* (Maggi et al., 2011) adalah algoritma yang membentuk model deklaratif dari log data. Model deklaratif adalah gabungan berbagai *rules* untuk menggambarkan relasi aktivitas secara implisit. Relasi yang implisit adalah relasi yang tidak menggambarkan secara langsung kondisi sekuensial atau paralel, seperti XOR, OR, dan AND, akan tetapi menggambarkan ciri-ciri terjadinya kondisi-kondisi tersebut. Penggambaran *rules* secara implisit itu memberikan fleksibilitas pada model. *Rules* yang digunakan disebut juga *rules (template)* Declare. Beberapan *rules* declare, yang mana digunakan pada penelitian ini, dapat dilihat pada Tabel 2.5. Setiap *rules* dilengkapi dengan format LTL (Linear Temporal Logic). Format ini digunakan untuk menentukan apakah *rules* sesuai dengan keadaan proses dalam log data.

Tabel 2.5 Potongan *Rules* Declare

***Rules* Declare**	**LTL**	**Deskripsi**
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

Langkah dari Algoritma *Declare Miner* adalah sebagai berikut:

1. Pemilihan *rules* yang akan ditampilkan dalam model deklaratif. Untuk penelitian ini, *rules* yang dipilih adalah *rules* pada Tabel 2.5.
2. Penentuan nilai minimum *support* sebagai *threshold* untuk penentuan *rules* yang terpilih. Nilai *support* adalah jumlah keadaan aktivitas di log data yang memenuhi *rules* dibagi dengan jumlah kemunculan aktivator. Sebagai contoh, terdapat proses dalam log data sebagai berikut : [A, B, D], [A, C, D], [A, B, D]. nilai *support* dari chain_response(A,B) adalah 0,67 karena [A,B] muncul dua kali dan relasi dimana A sebagai aktivitas persis sebelum aktivitas lain ada tiga kejadian. nilai *support* dari exclusive_choice(B,C) adalah 1.0 karena dari semua proses, *rules exclusive_choice,* yang menyatakan bahwa aktivitas B dan D tidak berada dalam proses yang sama, terpenuhi.
3. Setiap tipe *rules* dengan mengkombinasikan semua aktivitas dicek berdasarkan log data untuk mendapatkan nilai *support*.
4. *Rules* yang memiliki nilai *support* sama dengan atau lebih besar dari nilai minimum *support* akan dipilih dan digabungkan menjadi model deklaratif. Contoh hasil model deklaratif dapat dilihat pada Gambar 2.7.



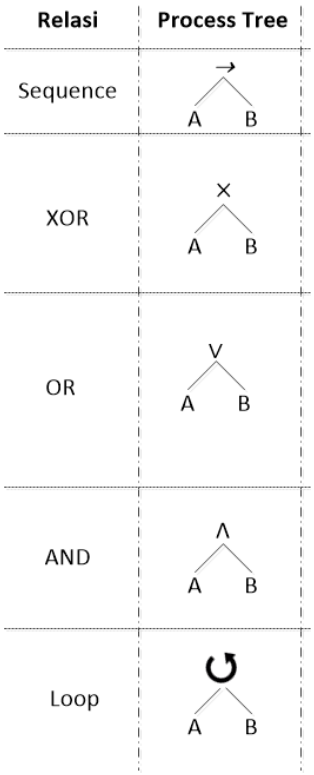
Gambar 2.7 Contoh Model Deklaratif

Model deklaratif tidak menggambarkan relasi paralel (XOR, OR, AND) maupun relasi *non-free choice* secara langung. Relasi ini disebut sebagai relasi implisit. Pada

model proses imperatif, seperti model YAWL, model Petri Net, penggambaran relasi secara langsung dengan menggunakan *control-flow pattern*. Untuk mempermudah penentuan aktivitas pengganti dalam perbaikan proses, relasi implisit dalam model deklaratif dapat dikonversi ke dalam *control-flow pattern* pada model proses imperatif.

2.2.8 Model *Tree*

Model *tree* adalah model umum yang banyak digunakan oleh beberapa algoritma *process mining*, salah satunya *Inductive Miner* . Suatu model *tree* memiliki *operator nodes* dan *leaf node*. *Operator node* melambangkan relasi antar anak cabang (Buijs, Van Dongen, & Van Der Aalst, 2012). *Operator node* terdiri dari relasi *sequence* (→), relasi AND (∧), relasi OR (∨), relasi XOR (×) dan *loop* (↺) (Buijs, van Dongen, & van der Aalst, 2012). *Leaf node* pada *process tree* merupakan aktivitas pada model proses (Buijs, van Dongen, & van der Aalst, 2012). Urutan pengeksekusian *node* pada *process tree* dimulai dari cabang paling kiri ke cabang paling kanan serta cabang paling bawah ke cabang paling atas (Buijs, van Dongen, & van der Aalst, 2012). Gambar 2.8 menunjukkan contoh model *tree* yang mendeskripsikan model proses Petri net.



Gambar 2.8 Notasi Model *Process Tree*

2.2.9 Perhitungan Kualitas Model

Terdapat empat aspek dalam mengukur kualitas model. Ke-empat aspek tersebut adalah *fitness*, presisi*, simplicity,* dan generalisasi. *Fitness* mengukur seberapa besar proses pada log data dapat tergambar di model proses. Presisi mengukur seberapa besar *trace* yang terbetuk dari model proses tergambar dalam log data. *Simplicity* mengukur kesederhanaan model proses tanpa menghilangkan proses dari log data. Sedangkan, generalisasi mengukur generalisasi model proses.

Sebuah model proses dengan *fitness* yang baik menunjukkan "kecocokan" dari model proses yang ditemukan dengan "realita" . Sebuah model proses dikatakan memiliki *fitness* sempurna jika semua *trace* di log data. dapat diwakili oleh model proses dari awal sampai akhir. Sebaliknya, jika banyak *trace* di log data tidak dapat diwakili oleh model proses dari awal sampai akhir, maka model proses disebut memiliki *fitness* yang buruk (De Cnudde, Claes, & Poels, 2014). Perhitungan *fitness* menggunakan Persamaan (1).

Presisi menyatakan bahwa suatu model proses tidak seharusnya menunjukkan proses yang cenderung berbeda dengan proses yang terlihat pada *log data*. Presisi berkaitan dengan notasi *overfitting* dalam konteks *data mining*. Suatu model proses dikatakan *overfitting* apabila model proses tersebut sangat spesifik dan berpatokan penuh pada contoh proses di log data. Semakin tinggi nilai presisi suatu model proses, maka semakin besar kecenderungan model tersebut dalam notasi *overfitting*. Perhitungan presisi menggunakan Persamaan (2).

Perhitungan *simplicity* suatu model proses dengan cara membandingkan ukuran *tree model* dari suatu model proses dengan aktivitas pada *event log*. Apabila aktivitas yang muncul pada model proses hanya sekali dan semua aktivitas pada *log data* tergambar pada *tree model*, maka model tersebut memiliki nilai *similarity* yang tinggi. Perhitungan *simplicity* menggunakan Persamaan (3).

Generalisasi menyatakan bahwa suatu model proses seharusnya menunjukkan generalisasi dari contoh proses yang terlihat pada *event log*. Generalisasi berkaitan dengan notasi *underfitting* dalam konteks *data mining*. Suatu model proses dikatakan *underfitting* apabila model proses tersebut juga menunjukkan proses yang cenderung

berbeda dengan proses yang terlihat pada *log data*. Semakin tinggi nilai generalisasi suatu model proses, maka semakin besar kecenderungan model tersebut dalam notasi *underfitting*. Perhitungan generalisasi suatu model proses memperhatikan frekuensi dari kemunculan *node* pada *process tree* berdasarkan *log data*. Perhitungan generalisasi menggunakan Persamaan (4).

$$Qf = \frac{\#casesCaptured}{\#casesLog} \quad (1)$$

$$Qp = \frac{\#traceCaptured}{\#tracesModel} \quad (2)$$

$$Qs = 1 - \frac{\#dupAct + \#misAct}{\#nodeTree} \quad (3)$$

$$Qg = 1 - \frac{\sum_1^{nodeTree}(\sqrt{\#exectNode})^{-1}}{\#nodeTree} \quad (4)$$

Keterangan:

Qf	:	nilai kualitas model proses dari sisi *fitness*
Qp	:	nilai kualitas model proses dari sisi presisi
Qs	:	nilai kualitas model proses dari sisi *simplicity*
Qg	:	nilai kualitas model proses dari sisi generalisasi
#casesCaptured	:	jumlah proses dalam *log data* yang tergambar di model proses
#casesLog	:	jumlah proses di *log data*
#tracesCaptured	:	jumlah *traces* (variasi proses yang dapat dibentuk dari model proses) yang terekam dalam proses di *log data*
#tracesModel	:	jumlah *traces* atau variasi proses yang dapat dibentuk dari model proses
#executions	:	jumlah *node* yang diimplementasikan pada *trace* di *log data*.
#dupAct	:	jumlah aktivitas yang digambarkan duplikasi dalam *tree model* serta jumlah duplikasinya.

#misAct : jumlah aktivitas pada *log data* yang tidak tergambar dalam *model tree* serta aktivitas yang tergambar tetapi tidak ada dalam *log data*.

#exectNode : jumlah *node* dalam *model tree* dilalui sesuai dengan proses dalam *log data*.

#nodeTree : jumlah *node* yang tedapat dalam *tree model*

2.2.10 Perhitungan Akurasi Metode Perbaikan Proses yang Terpotong

Metode pengujian yang dilakukan mengguunakan hasil akurasi. Terdapat dua persamaan akurasi yang dilakukan. Akurasi untuk penentuan anomali serta proses yang terpotong terdapat pada Tabel 2.6 dan Persamaan (5) , sedangkan akurasi untuk hasil perbaikan proses yang terpotong terdapat dalam Persamaan (6).

Tabel 2.6 *Confusion Matric* untuk menentukan Anomali dan Proses yang Terpotong

		Kelas Sebenarnya (dari *Expert*)	
		Proses yang Terpotong	Anomali
Kelas Prediksi	Proses yang Terpotong	TP	FP
	Anomali	FN	TN

$$Akurasi = \frac{TP + TN}{TP + TN + FP + FN} \quad (5)$$

$$Akurasi = \frac{n(PerbaikanBenar)}{n(ProsesTerpotong)} \times 100\% \quad (6)$$

Pada Tabel 2.6, kelas sebenarnya didapatkan dari *expert*, dimana *expert* memberikan *trace* dari proses yang benar. *Trace* dari proses yang benar dapat dilihat pada LAMPIRAN A. Apabila aktivitas di tengah-tengah proses tidak sesuai dengan aktivitas di tengah-tengah proses yang benar, maka proses tersebut dikategorikan sebagai anomali. Sebaliknya, apabila aktivitas di tengah-tengah proses sesuai dengan

proses yang sebenarnya dimana aktivitas awal atau akhirnya tidak terekam, maka proses tersebut disebut sebagai proses yang terpotong.

Pada Persamaan (6), n(PerbaikanBenar) merupakan jumlah dari hasil perbaikan proses yang terpotong yang sesuai dengan proses yang lengkap. Sedangkan n(ProsesTerpotong) adalah jumlah proses terpotong pada log data.

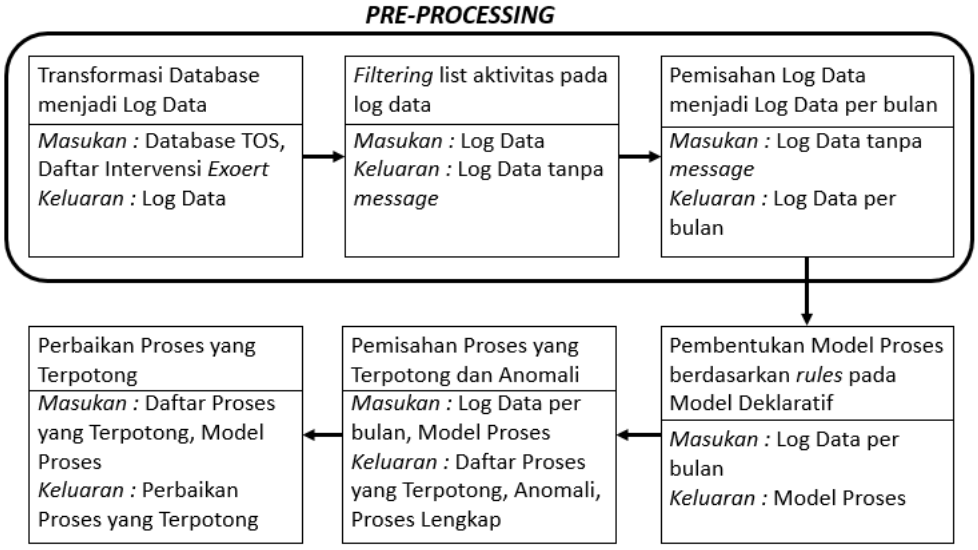
(*halaman ini sengaja dikosongkan*)

BAB 3.
METODA PENELITIAN

3.1 Metode Usulan

3.1.1 Model Sistem Metode Usulan

Metode usulan yang digunakan adalah pembentukan model proses imperatif berupa model *tree* dari *control-flow pattern* yang dibentuk berdasarkan model deklaratif, pembedaan proses yang terpotong dan anomali, serta perbaikan proses yang terpotong. Alur metode usulan dapat dilihat pada Gambar 3.1.



Gambar 3.1 Alur Metode Usulan

Tahapan pertama adalah melakukan *pre-processing*. Tahapan *pre-processing* dibagi menjadi tiga bagian, yaitu transformasi *database* ke log data, *filtering* aktivitas pada log data, dan pemisahan log data. Transformasi dilakukan untuk mengubah data di database TOS (Terminal Operating System) ke dalam log data yang berisi proses impor barang di Terminal Peti Kemas. Transformasi dilakukan untuk mendapatkan isi dari log data, seperti ID proses, aktivitas atau *message*, pelaku aktivitas atau *message*, waktu pelaksanaan, dan atribut pendukung aktivitas atau *message* (data lain yang mempengaruhi pelaksanaan suatu aktivitas atau *message*). Kemudian, proses *filtering* dilakukan setelah proses transformasi. Proses ini memisahkan *message* dari log data sehingga hanya aktivitas yang termasuk dalam log data. Hal ini dikarenakan algoritma

model proses tidak memerlukan *message*. Langkah terakhir dalam *pre-processing* adalah pemecahan log data menjadi log data per bulan. Hal ini dilakukan untuk membentuk log data yang mengandung proses yang terpotong.

Tahapan kedua adalah pembentukan model proses imperatif. Pembentukan model proses tersebut menggunakan log data per bulan. Langkah dalam memodelkan proses adalah memodifikasi algoritma *Declare Miner*, dimana modifikasi dilakukan dengan menambahkan metode untuk mengubah *rules* pada model hasil *Declare Miner* ke dalam model proses *tree*. Pengubahan tersebut memiliki dua langkah yaitu membentuk *control-flow pattern* dari *rules* dan menggabungkan *control-flow pattern* ke dalam model proses *tree*. Setelah model proses *tree* terbentuk, maka aktivitas yang berada dalam relasi pilihan diberi keterangan tambahan. Keterangan tambahan tersebut berdasarkan atribut pendukung pada log data.

Tahapan ketiga adalah perbaikan proses yang terpotong. Perbaikan proses yang terpotong dilakukan dengan cara pemisahan proses yang terpotong dengan anomali pada log data dan perbaikan proses yang terpotong. Pemisahan proses yang terpotong menggunakan metode usulan berdasarkan model proses *tree* yang sudah terbentuk. Kemudian, proses yang terpotong tersebut diperbaiki dengan menambahkan aktivitas sebagai aktivitas pengganti dari aktivitas yang hilang. Penambahan aktivitas pengganti dari aktivitas yang hilang sesuai dengan urutan aktivitas dalam *node* pada model *tree*. Pada proses yang terpotong di bagian akhir, urutan *node* dalam bentuk *pre-order*, dan proses yang terpotong di bagian awal menggunakan model urutan *reverse* dari *pre-order*. Akan tetapi, aktivitas yang dipilih tidak hanya berdasarkan urutan melainkan berdasarkan *node* operator pada model *tree. Node* operator akan menentukan aktivitas yang dipilih. Hasil akhir dari penelitian ini adalah proses terpotong yang telah diperbaiki.

3.1.2 Jadwal Kegiatan

Jadwal kegiatan dalam penyusunan penelitian dapat dilihat pada Tabel 3.1.

Tabel 3.1 Jadwal Kegiatan Penelitian

Aktivitas	**Bulan**																					
	8				**9**				**10**					**11**				**12**				
Studi Literatur																						
Analisa Perancangan																						

Implementasi Pengujian Evaluasi																						
Penyusunan Buku																						

3.1.3 Masukan

Dalam penelitian ini, log data yang digunakan adalah log data mengenai proses impor barang di Terminal Peti Kemas Surabaya. Log data merupakan hasil *pre-processing* yang diambil dari database TOS (Terminal Operating System) mengenai proses impor barang di Terminal Peti Kemas. Data yang diambil adalah data yang berjalan dari bulan Januari sampai berakhir di bulan Maret 2016. Penanda dari awal proses pada database adalah tanggal pada kolom VESEL_ATB dan penanda akhir proses adalah tanggal yang tersimpan pada kolom TRUCK_OUT.

Database TOS memiliki beberapa kolom, dimana berisi tanggal untuk menunjukkan aktivitas pada proses, serta data lainnya yang menunjang pembentukan log data. Tabel 3.2 memaparkan keterangan dari kolom database TOS yang digunakan dalam pembentukan log data.

Tabel 3.2 Keterangan Kolom Database TOS

Kolom	**Keterangan**
CONTAINER_KEY	Kode unik untuk membedakan data proses impor barang yang satu dengan yang lain
CTR_TYPE	Tipe umum barang yang di-impor. Tipe ini dibedakan ke dalam tiga macam yaitu Reefer, Dry, dan Uncontainer yang dituliskan dalam kode khusus.
YARD_BLOCK	Blok dari tempat untuk menumpuk barang (letak penumpukkan barang dalam zona vertikal)
YARD_SLOT	Slot dari tempat untuk menumpuk barang (letak penumpukkan barang dalam zona horizontal)
HAS_QUARANTINE_FLAG	Penunjuk dilakukannya proses di karantina atau tidak, yang disimbolkan “YES” dan “NO”
CUSTOM_BEHANDLE_COUNT	Penunjuk dilakukannya proses di behandle atau tidak, yang disimbolkan “0” dan “1”

3.1.4 Pre-processing

3.1.4.1 Transformasi ke Log Data berdasarkan Database

Log data yang akan digunakan adalah hasil dari transformasi dari database TOS. Log data yang digunakan berisi Case_ID, nama aktivitas atau *message*, waktu pelaksanaan, pelaku aktivitas (*originator*), pelaku *message* (sender receiver), *cost, atribut* tambahan yang mendukung aktivitas (*input, output, detail attachment*). Cara transformasi secara garis besar dapat dilihat pada Tabel 3.3.

Tabel 3.3 Cara Transformasi ke Log Data secara Garis Besar

Isi Log Data	**Cara Transformasi**
Case_ID	Mengambil data pada kolom CONTAINER_KEY
Waktu pelaksanaan	Melakukan *inverse* distribusi kumulatif normal berdasarkan database dan rentang waktu intervensi dari *expert*
Nama aktivitas dan *message*	Berdasarkan intervensi *expert*
Pelaku *message* (sender receiver)	
Pelaku aktivitas (originator)	
Cost	Biaya aktivitas didapatkan dari rumus yang mengolah data biaya kelompok aktivitas
Atribut tambahan (*input output*)	Berdasarkan list dokumen yang dibutuhkan pada proses impor barang
Atribut tambahan (*detail attachment*)	Berdasarkan kolom pada database, yaitu CTR_TYPE, HAS_QUARANTINE_FLAG, dan CUSTOM_BEHANDLE_COUNT

Database TOS menyimpan waktu dari aktivitas dan *message*, akan tetapi tidak semua kolom menyimpan waktu setiap aktivitas dan *message*. Terdapat beberapa kolom yang menunjukkan rentang waktu pelaksanaan beberapa aktivitas serta *message* sekaligus. Sedangkan, log data membutuhkan waktu setiap aktivitas atau *message*. Oleh karena itu, diperlukan perkiraan waktu untuk setiap aktivitas dan *message*.

Selain database TOS, *expert* juga memberikan perkiraan rentang waktu eksekusi setiap aktivitas. Perkiraan rentang waktu eksekusi tersebut dimanfaatkan untuk menentukan perkiraan waktu setiap aktivitas dan *message.* Penentuan waktu eksekusi setiap aktivitas menggunakan *inverse normal distribution* yang mengacu pada perkiraan

rentang waktu eksekusi dan rentang waktu beberapa aktivitas dari database. Pengunaan *normal distribution* dipilih dengan asumsi bahwa persebaran datanya adalah sebaran normal.

Persamaan untuk perhitungan waktu aktivitas dapat dilihat pada Persamaan (7) sampai Persamaan (9). Waktu aktivitas didapatkan dari penambahan waktu sebelumnya dan perkiraan waktu eksekusi aktivtias menggunakan *inverse normal distribution*. Rata-rata untuk *inverse normal distribution* adalah nilai perkiraan berdasarkan median rentang waktu eksekusi dan rentang waktu beberapa aktivitas database. Simpangan baku adalah 5% dari nilai rata-rata. Pemilihan 5% untuk memberikan persebaran yang tidak lebar dalam hasil perkiraan waktu eksekusi aktivitas.

$$time(Act\_now) = time(Act\_before) + NormInv(p, \mu, \sigma_x) \quad (7)$$

$$\mu = \frac{\widetilde{Es.Act}}{\sum_{i=0}^{n} \widetilde{Es.Act_i}} \times \overline{ActinLog} \quad (8)$$

$$\sigma_x = 0{,}05 \times \mu \quad (9)$$

dimana:

time(Act_now)	=	*waktu pelaksanaan aktivitas atau message sekarang*
time(Act_bfeore)	=	*waktu pelaksanaan aktivitas atau message sebelumnya*
NormInv	=	*inverse normal distribution untuk memperkirakan rentang waktu tiap aktivtias*
$\widetilde{Es.Act}$	=	*median dari interval rentang waktu yang didapatkan dari expert*
$\overline{ActinLog}$	=	*rata-rata dari rentang waktu kelompok aktivitas di database*
n	=	*jumlah aktivitas yang berada pada log data*
p	=	*nilai probabilitas secara random*
μ	=	*rata-rata untuk perhitungan inverse normal distribution*
σ_x	=	*simpangan baku untuk perhitungan inverse normal distribution*

Contoh dari pembentukan waktu pelaksanaan adalah pembentukan waktu aktivitas-aktivitas yang terjadi saat truk datang untuk mengambil barang (TRUCK IN) dan truk keluar dari Terminal Peti Kemas (TRUCK OUT). Perhitungan dapat dilihat pada Tabel 3.4. Batas atas dan batas bawah adalah intervensi yang diberikan oleh *expert* mengenai rentang waktu eksekusi setiap aktivitas. Kemudian, dari rentang waktu eksekusi tersebut, diambil nilai median yang ditunjukkan oleh nilai pada Median(Es.Act). Kemudian, nilai rata-rata didapatkan dari nilai pada Median(Es.Act) dibagi jumlah keseluruhan nilai media dan dikali dengan rentang waktu kelompok aktivitas pada database, yang ditunjukkan oleh nilai ActinLog. Kemudian, nilai standar deviasi menunjukkan nilai simpangan baku. Nilai rata-rata dan simpangan baku yang didapatkan kemudian diolah menggunakan Persamaan (7) untuk menghasilkan waktu setiap aktivitas atau *message*. Hasil waktu setiap aktivitas dapat dilihat pada Tabel 3.5, dimana nilai time(Act_now) berada pada kolom Time.

Tabel 3.4 Perhitungan Waktu Aktivitas untuk Log Data

Aktivitas	Batas Atas	Batas Bawah	Median(Es.Act)	Rata-rata	Standar Deviasi
Dispatch WQ Delivery to CHE	0:00:30	0:00:50	0:00:40	0:00:40	0:00:02
Determine Container Type	0:01:00	0:02:00	0:01:30	0:01:31	0:00:05
Determining Uncointaner	0:01:00	0:02:00	0:01:30	0:01:31	0:00:05
Decide Task Before Lift Container	0:01:00	0:02:00	0:01:30	0:01:31	0:00:05
Prepare Tools	0:10:00	0:15:00	0:12:30	0:12:35	0:00:38
Lift on Container Truck	0:02:00	0:03:00	0:02:30	0:02:31	0:00:08
Truck Go To Gate Out	0:01:00	0:02:00	0:01:30	0:01:31	0:00:05
Check Container before Truck out	0:01:00	0:02:00	0:01:30	0:01:31	0:00:05
		sum(median(Es.Act))	0:23:10		
		ActInLog	**0:23:20**		

Tabel 3.5 Contoh Hasil Estimasi Waktu Setiap Aktivitas

CaseID	Activity	Time	NormInv
4619882	Truck in	3/14/16 19:30:21	
4619882	Dispatch WQ Delivery to CHE	3/14/16 19:31:02	0:00:41
4619882	Determine Container Type	3/14/16 19:32:30	0:01:28
4619882	Determining Uncointaner	3/14/16 19:34:06	0:01:36
4619882	Decide Task Before Lift Container	3/14/16 19:35:36	0:01:31
4619882	Prepare Tools	3/14/16 19:48:30	0:12:54
4619882	Lift on Container Truck	3/14/16 19:51:05	0:02:35
4619882	Truck Go To Gate Out	3/14/16 19:52:34	0:01:29
4619882	Check Container before Truck out	3/14/16 19:54:13	0:01:39
4619882	Truck Out	3/14/16 22:48:25	

Kemudian, *detail attachment* pada log data berisi atribut atau data tambahan berkaitan dengan suatu aktivitas atau *message*. *Detail attachment* akan digunakan dalam penentuan aktivitas pengganti dari aktivitas hilang pada metode perbaikan proses yang terpotong. *Detail attachment* diambil dari data pada database TOS, yaitu data pada

kolom CTR_TYPE, HAS_QUARANTINE_FLAG, CUSTOM_BEHANDLE_COUNT. Pseudocode dari pengisian *detail attachment* ditunjukkan pada Gambar 3.2. Setiap data di database TOS akan dicek dan *detail attachment* akan terisi sesuai dengan aturan pada pseudocode.

3.1.4.2 Filtering List Aktivitas pada Log Data

Tahapan kedua dari *pre-processing* adalah *filtering* list aktivitas pada log data. *Filtering* dilakukan dengan cara memisahkan data *message* dari log data, sehingga log data hanya berisi data aktivitas. Hal ini dilakukan karena algoritma pemodelan proses mengolah aktivitas, bukan *message.*

```
1.  for all data in database TOS:
2.      if CTR_TYPE is “DRY”:
3.          DetailAttachmentInLog.add(“Dry”)
4.      else if CTR_TYPE is “RFR”:
5.          DetailAttachmentInLog.add(“Reefer”)
6.      else:
7           DetailAttachmentInLog.add(“Uncontainer”)
8       if CUSTOM_BEHANDLE_COUNT is “0”:
9           DetailAttachmentInLog.add(“Green Line”)
10      else:
11          DetailAttachmentInLog.add(“Red Line”)
12      if HAS_QUARANTINE_FLAG is “YES”:
13          DetailAttachmentInLog.add(“Quarantine”)
```

Gambar 3.2 Pseudocode Pengisian *Detail Attachment*

Pemisahan *message* pada log data didasarkan pada data pelaksana. Apabila pelaksana ada dua yaitu *sender* dan *receiver*, maka data itu adalah data *message*. Sedangkan apabila pelaku hanya satu, yang biasa disebut *originator*, maka data tersebut adalah data aktivitas. Pseudocode untuk penentuan *message* dan aktivitas dapat dilihat pada Gambar 3.3. Apabila data terdeteksi *message*, maka akan ditambahkan pada log_message dan dihapus dari log_data. Penambahan *message* tidak hanya melibatkan nama *message*, melainkan waktu pelaksanaan serta atribut yang berkaitan dengan *message* tersebut.

```
1.  activity_or_message = data_in_column_activity_of_Log
2.  for x in activity_or_message:
3.      if sender(x) and receiver(x) != NULL:
4.          Log_message.add(message)
5.          Log_data.erase(message)
```

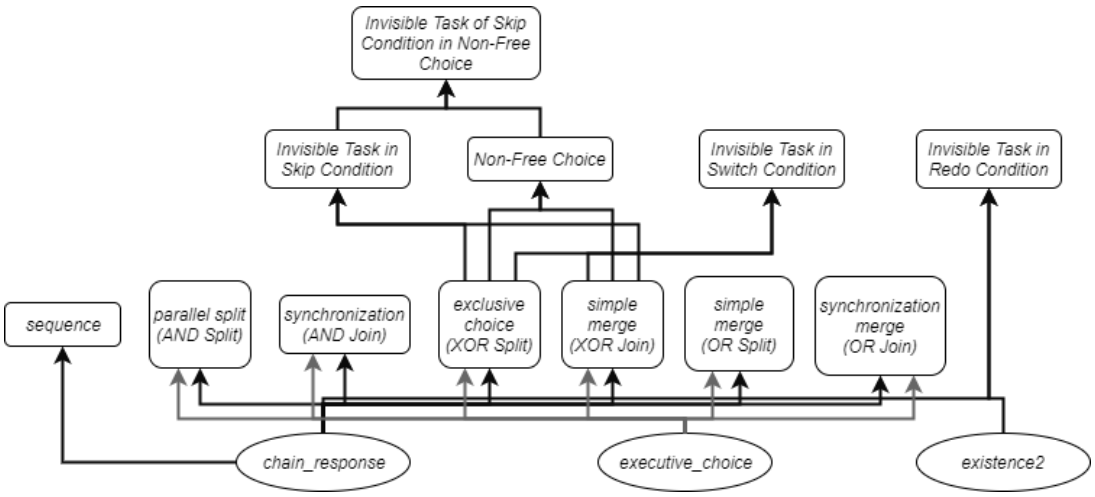
Gambar 3.3 Pseudocode pemisahan aktivitas dan *message*

3.1.4.3 Pemisahan Log Data

Proses yang terpotong dapat terjadi dikarenakan pengambilan log data dalam interval waktu tertentu. Log data yang ditransformasi dari database TOS berisi proses-proses lengkap yang berjalan di bulan Januari sampai bulan Maret. Untuk mendapatkan log data yang berisi proses yang terpotong, maka log data dipecah menjadi tiga log data, yaitu log data yang berisi aktivitas yang berjalan pada bulan Januari, log data yang berisi aktivitas yang berjalan pada bulan Februari, dan log data yang berisi aktivitas yang berjalan pada bulan Maret. Ketiga log ini akan diolah secara terpisah dalam perbaikan proses yang terpotong.

3.1.5 Pembentukan Model *Tree* dari *rules* pada Model Deklaratif

3.1.5.1 Pembentukan *Control-Flow Pattern* dari Model Deklaratif



Gambar 3.4 Alur Hirarki Pembentukan *Control-Flow Patterns*

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 3.4 menunjukkan alur pembentukan *control-flow patterns*. *Control-flow pattern* umun, yaitu *control-flow pattern* yang membentuk relasi sekuensial serta relasi paralel, seperti XOR, OR dan AND, dibentuk secara langsung dari *rules* pada model deklaratif. *Control-flow pattern* tersebut terbentuk dari *rules chain_response* dan *executive_choice*. *Control-flow pattern* lainnya, seperti *control-flow pattern* yang membentuk *invisible task* dan *non-free choice* dibentuk berdasarkan *pattern* lain yang sudah terbentuk.

Pattern yang membentuk *invisible task* pada kondisi melewati dan kondisi berpindah serta *non-free choice* dibentuk dengan memanfaatkan *pattern* lain, yaitu *pattern* yang membentuk relasi paralel XOR. Kemudian, *pattern invisible task in non-free choice* merupakan gabungan *pattern* dari *pattern* yang membentuk *invisible task* dan *non-free choice*.

Metode untuk pembangunan *control-flow pattern* tersebut dipaparkan pada Tabel 3.6. Di dalam metode, variabel dari total_next dan total_before digunakan untuk mendeteksi *pattern* umum. Kemudian, variabel total_ex dan total_chain adalah variabel yang digunakan untuk membentuk *pattern* split dan join. Sebagai ilustrasi, terdapat *chain_response*(K,R)¸ *chain_response*(A,K), *chain_response*(A,M), *exclusive_choice*(K,M) yang membentuk model deklaratif. Aktivitas K memiliki satu total_next karena aktivitas K hanya menjadi aktivitas awal di satu *chain_response* yaitu *chain_response*(K,R). Sedangkan, aktivitas A memiliki memiliki dua total_next. Dari data tersebut, *pattern* sequence ditemukan yaitu K -> O (R). Kemudian, dikarenakan aktivitas A memiliki dua total_next dan satu total_ex_after, maka terbentuk *pattern* untuk membentuk XOR Split yaitu A -> O ((K ∨ R)).

Tabel 3.6 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	`If total_before(act) > 1 and total_ex(act) == 0:`

(AND Join)	◊((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)
Non-Free Choice Relation	If act in Exclusive Choice and act in Simple Merge and total_ex_aftbef(act) < amount(act_before)^2-1: ◊((x1 ∧ act ∧ k) ∨ (x2 ∧ act ∧ k) …. ∨ (xn ∧ act ∧ k)), where x1….xn ∈ act_before of act, k = act_next in executive_choice(x, act_next) which has minimum support value
Invisible Task in Skip and Switch Condition	If act appears in Exclusive Choice before "->" and appears in Simple Merge before "->": Change O ((act ∨ y2 …. ∨ yn)) -> O (other_act) into O ((Invisible_Task ∨ y2 …. ∨ yn)) -> O (other_act) If act appears in Exclusive Choice after "->" and appears in Simple Merge after "->": Change other_act -> O ((act ∨ y2 …. ∨ yn)) into other_act -> O ((Invisible_Task ∨ y2 …. ∨ yn))
Invisible Task in Redo Condition	If act appears in LTLSequence before "->" and act not in *Existence(2)* and act_after(act) in *Existence(2):* act -> ◊ ((act_after1 ∧ O (act_after2) .... ∧ O (act_after_n)))
Invisible Taks in Non-Free Choice Relation	If act appears in Simple Merge after "->" and act appears in Non-Free Choice Relation Change ◊ ((x1 ∧ other_act ∧ act) ∨ (x2 ∧ other_act ∧k) into ◊ ((x1∧ other_act ∧ Invisible_Task) ∨ (x2 ∧ other_act ∧ k) If act appears in Simple Merge before "->" and act appears in Non-Free Choice Relation > 1:

	Change O ((y1 ∨ act …. ∨ yn)) -> O (other_act) into O ((y1 ∨ Invisible_Task …. ∨ yn)) -> O (other_act)

dimana:

Total_next(act) : jumlah *chain_response*(act, aktivitas lain)

Total_before(act) : jumlah *chain_response*(aktivitas lain, act)

Total_ex_after(act) : jumah *exclusive_choice*(aktivitas selanjutnya dari act, aktivitas selanjutnya dari act)

Total_ex_before(act) : jumlah *exclusive_choice*(aktivitas sebelumnya dari act, aktivitas sebelumnya dari act)

Total_chain_after(act) : jumlah *chain_response*(aktivitas selanjutnya dari act, aktivitas selanjutnya dari act)

Total_chain_before(act) : jumlah *chain_response*(aktivitas sebelumnya dari act, aktivitas sebelumnya dari act)

Total_ex_aftbef(act) : jumlah *exclusive_choice*(aktivitas sebelumnya dari act, aktivitas selanjutnya dari act)

Selain *control-flow pattern,* aktivitas awal dan aktivitas akhir juga diidentifikasikan bersamaan dengan *control-flow pattern* dalam bentuk Linear Temporal Logic (LTL). Aktivitas awal adalah aktivitas yang tidak memiliki *total_before* dan aktivitas akhir adalah aktivitas yang tidak memiliki *total_next*. Aktivitas awal disimbolkan dengan bentuk *FirstActivity*(aktivitas) dan aktivitas akhir disimbolkan dalam bentuk *LastActivity*(aktivitas).

3.1.5.2 Pembentukan Model *Tree* dari *Control-Flow Pattern*

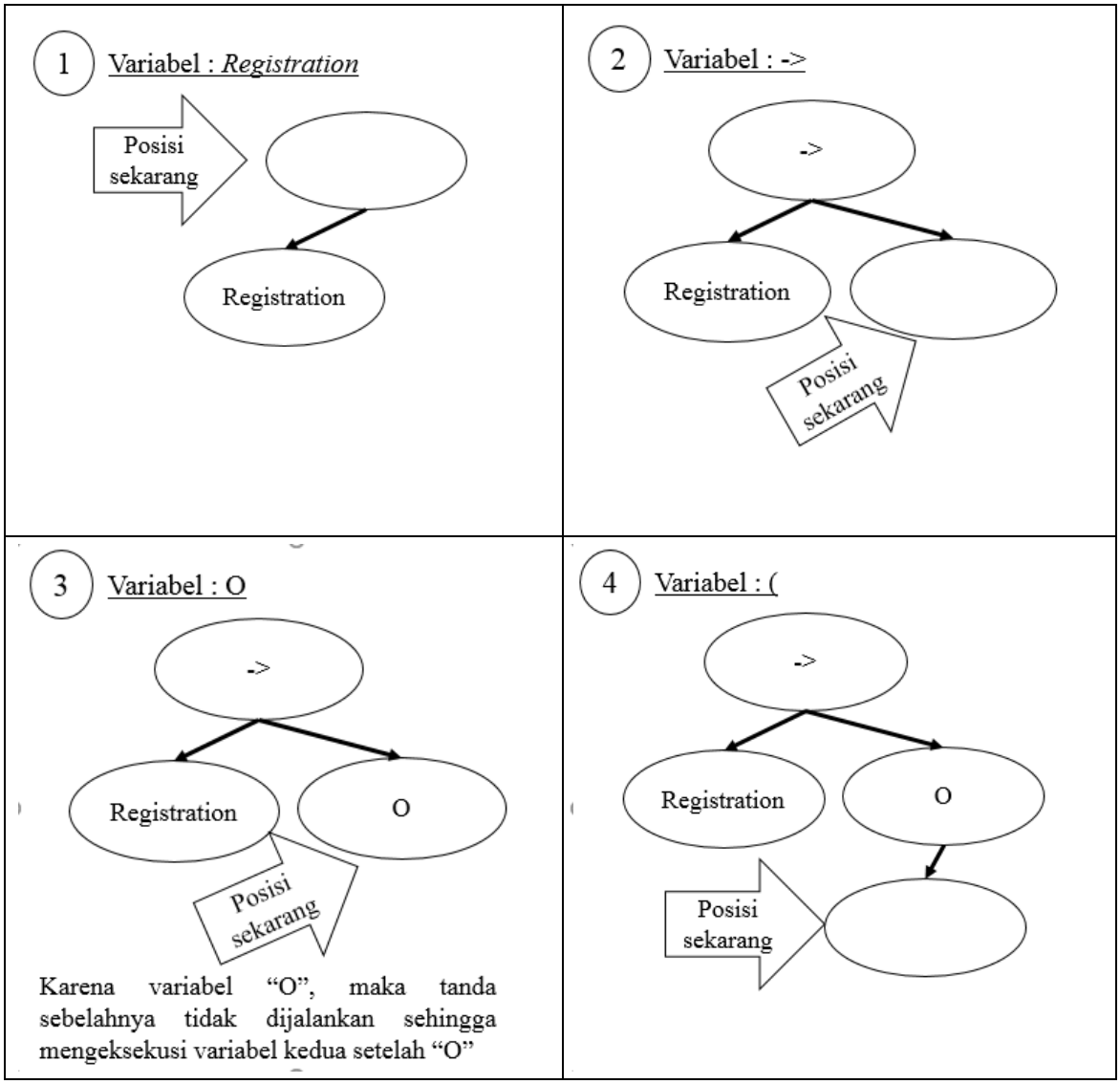
Control-Flow Patterns yang sudah terbentuk tersebut disusun dalam model *tree* yang akan digunakan untuk mendeteksi serta memperbaiki proses yang terpotong. Metode penyusunan *control-flow patterns* dapat dilihat pada Gambar 3.5. Langkah awal pada metode pembentukan model proses *tree* adalah pemilihan *control-flow pattern* dalam bentuk LTL pertama. *Control-flow pattern* pertama ini adalah *control-flow pattern* yang memiliki aktivitas yang tergolong dalam *FirstActivity*(aktivitas). Kemudian, dimulai dari *control-flow pattern* pertama, setiap *control-flow pattern* akan dipecah ke dalam beberapa bagian, yaitu simbol serta aktivitas. Pembagian dapat dilakukan dengan memecah berdasarkan spasi. Kemudian, pecahan tersebut akan

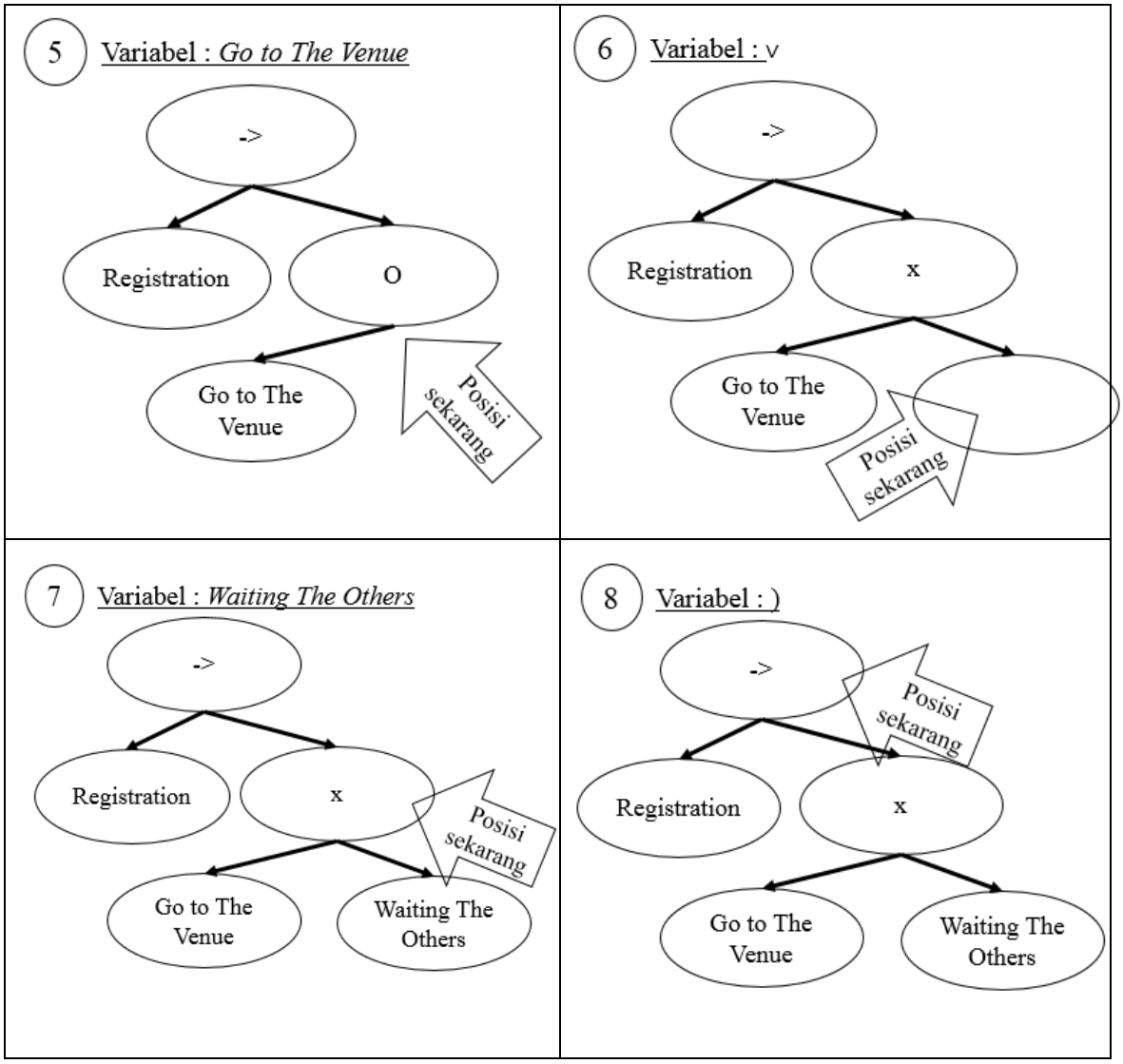
diproses mulai dari komponen aawal dalam Part_LTL sampai komponen akhir pada Part_LTL. Pemecahan sesuai langkah ke-4 sampai langkah ke-25. Apabila semua *Control-Flow Patterns* sudah dieksekusi, maka model *tree* telah terbentuk.

```
1   LTL = Control-Flow Pattern in LTL that has FirstActivity
2   While all LTL are modeled:
3       Split LTL into parts that are called part_LTL
4       For part_LTL in LTL:
5         If part_LTL is "(":
6             Make a child of node and the position is in the child of
              node
7         Else if part LTL is ")":
8             Position is in the parent of node
9         Else if part LTL is "O":
10            If node of this position is "∨":
11                node = ↺
12            Else: node = "O" and the bracket beside this part is not
              executed
13        Else if part LTL is "◊"
14            Node = ◊ and the bracket beside this part is not executed
15        Else if part LTL is "∨":
16            If node of this position is "O":
17               node = "x"
18            If node of this position is "◊":
19               node = "∨"
20        Else if part LTL is "^" or "->":
21            If node of this position is "x" or "∨":
22               Add child node and fill with "->" or  "^"
23               All of child of node "x" or "v" become child of node
                 "->" or "^"
24        Else:
25            Fill the node name with the name of part_LTL
              position is in the parents
26      LTL = LTL that has activity in previous LTL
```

Gambar 3.5 Metode Pembentukan Model *Tree*

Sebagai contoh, terdapat *control-flow pattern* berupa *Registration -> O ((Go to The Venue ∨ Waiting The Others))*. Pertama, *control-flow pattern* ini dipecah menjadi Part_LTL = [*Registration*, *->*, *O*, *(*, *(*, *Go to The Venue*, *∨*, *Waiting The Others*, *)*, *)*]. Kemudian, pembentukan model *tree* dimulai dari "Registration" sampai simbol ")". Langkah-langkahnya dapat dilihat pada Gambar 3.6. Hasil akhir adalah model proses *tree* yang ditunjukkan pada langkah terakhir, yaitu langkah 8.





Gambar 3.6 Langkah Pembentukan Model Proses *Tree* dari Contoh *Control-Flow Pattern*

3.1.5.3 Penambahan Atribut pada Model *Tree*

Selain relasi aktivitas, atribut pada log data dapat digunakan untuk menentukan aktivitas dari relasi pilihan, yang biasa disebut dengan *decision mining*. Atribut merupakan data tambahan yang terekam dalam log data, seperti *input* aktivitas, atau *detail attachment* yang terkait dengan aktivitas. Atribut-atribut tersebut biasanya terletak pada aktivitas yang merupakan aktivitas sebelum pemilihan aktivitas pada relasi pilihan (XOR atau OR).

Akan tetapi, pada log data yang digunakan sebagai pengujian, terdapat atribut untuk relasi pilihan yang terekam pada aktivitas pertama, bukan pada aktivitas sebelum

relasi. Oleh karena itu, penelitian ini memperkirakan atribut yang sesuai untuk aktivitas sebelum relasi pilihan berdasarkan peraturan yang dibentuk pada Gambar 3.7. Sebelum peraturan dijelaskan, terdapat kondisi khusus dimana proses yang terekam tidak memiliki atribut. Hal itu terjadi pada proses yang terpotong di bagian awal proses. Untuk mengatasi hal tersebut, atribut yang banyak muncul pada log data digunakan sebagai atribut pada proses tersebut.

```
List_all[act][att] = list of attributes that are related with activity
List[att] = list of index of occurance of attribute
For case in all_cases_log:
    For att in attributes:
        List[att] = index(att)
        List[act][att] = List[act][att] + 1
Grouping the attribute that has same index in one field of List_check[att]
For act in List_all[act][att]:
    For check in List_check[att]:
        For att_in_check in check:
             If List_all[act][att] > 0:
                 Var_checking = 1
        If Var_checking > 1:
             For att_in_check in check:
                 List_all[act][att_in_check] = 0
Att of act = attribut that has List/aal[act][att_in_check]=0
If act in model tree has node.parent("x") or has node.parent("V")
    add attribute in activity node
Else If node has node.parent.parent("x") or has node.parent.parent("V"):
    Add attribute in node.parent of activity
```

Gambar 3.7 Metode Penentuan Atribut pada Model *Tree*

Pada metode ini, atribut akan dikelompokkan berdasarkan urutan kemunculannya di log data. Urutan kemunculan atribut tersebut didefinisikan sebagai index(att). Kemudian, masing-masing aktivitas dicek sesuai kelompok atribut tersebut. Apabila aktivitas memiliki lebih dari satu atribut pada kelompok yang sama (didefinisikan di

langkah 13), maka seluruh atribut yang berada pada kelompok tersebut bukan atribut dari aktivitas kemudian. Terakhir, apabila aktivitas memiliki *node parent* adalah "x" atau "V", maka atribut tersebut melekat pada aktivitas di model *tree*. Akan tetapi, apabila aktivitas memiliki *"*x" atau "V" sebagai *parent* dari *node.parent* , maka atribut melekat pada *node.parent* dari aktivitas tersebut.

3.1.6 Perbaikan Proses Yang Terpotong

3.1.6.1 Penentuan Proses yang Anomali atau Proses Yang Terpotong

Penentuan proses yang anomali dan proses yang terpotong dapat dilihat pada Gambar 3.8. Pada metode ini, setiap proses di log data akan dicek apakah sudah sesuai dengan model *tree* yang dibentuk. Apabila proses tidak sesuai, maka proses tergolong sebagai anomali, sedangkan apabila proses sesuai melainkan aktivitas awal atau akhirnya tidak sama dengan aktivitas awal dan akhir di model *tree,* maka proses tersebut tergolong proses yang terpotong (*truncated processes*). Proses tidak sesuai ditunjukkan dengan check_true bernilai 0, sedangkan proses yang sesuai ditunjukkan dengan nilai check_true di atas 0.

Langkah 7 sampai langkah 14 menunjukkan *pseudocode* untuk mengecek aktivitas pilihan dari relasi XOR "x", OR "v", dan AND "/ \". Variabel child menunjukkan *node* yang memiliki operator relasi sebagai *node parent*-nya dan variabel right_child menunjukkan variabel yang sesuai dengan aktivitas pada *case*. Langkah 15 sampai langkah 20 adalah mengecek aktivitas dalam relasi *sequence* yang sesuai dengan aktivitas pada *case*. Kemudian, langkah 21 sampai langkah 25 digunakan untuk menentukan kelompok *case* yaitu kelompok proses terpotong (*truncated processes)* atau kelompok anomali (*anomaly*).

```
1   First_node = first activity of case
2   Last_node = last activity of case
4   For node: first_node – last_node in preorder(tree):
5     if node == "x" or "V" or "/\":
6       check all of the child of this node that are related
        with activity in case
7       If node == "x" and n(right_child) == 1:
8             Check_true = 1
```

```
9       Else if node == “V” and n(right_child) >=1 or
        <=n(child):
10              Check_true = 1
11      Else if node == “/\” and n(right_child) == n(child):
12              Check_true = 1
13      Else:
14              Check_true = 0 and break
15    Else:
16      If node is activity:
17              If node is related with activity in case:
18                  Check_true = 1
19              Else:
20                  Check_true = 0 and break
21  If check_true == 1:
22    If first_activity is not same with first_node in tree or
      last_activity is not same with last_node in tree:
23      Case = truncated processes
24  Else:
25    Case = anomaly
```

Gambar 3.8 Metode Penentuan Anomali atau Proses yang Terpotong

3.1.6.2 Perbaikan Proses Yang Terpotong

Dari pengelompokkan proses yang merupakan proses yang terpotong, proses tersebut akan diperbaiki dengan menambahkan aktivitas pengganti dari aktivitas yang hilang. Sebelum melakukan perbaikan, proses terpotong tersebut perlu diperiksa kembali untuk mengetahui jenis proses yang terpotong. Apabila proses yang terpotong karena kehilangan aktivitas di bagian awal proses, maka perbaikan menggunakan metode untuk proses yang terpotong di depan. Apabila proses yang terpotong karena kehilangan aktivitas di bagian akhir proses, maka perbaikan menggunakan metode untuk proses yang terpotong di depan. Sedangkan apabila proses terpotong di bagian awal maupun akhir, maka kedua metode tersebut dijalankan.

```
For node in model tree:
    If node is not start_node or “Invisible_Task”:
        If node == “x” or “V”:
            child = 0
        If node.parent == “x” or “V”:
            If attribute in node is same as attribute of case:
                If node is activity:
                    if (node.parent == “x” and child<1) or (node.parent == “V” and child<3):
                        add name of the node
                        Child +=1
                Else:
                    Att_true = 1
            Else:
                Att_true = 0
        If node is activity:
            If node.parent.parent == “x” or node.parent.parent == “V”:
                If Att_true = 1:
                    Add name of the node
                Else:
                    if (node.parent == “x” and child<1) or (node.parent == “V” and child<3):
                        add name of the node
                        Child +=1
            Else:
                Add name of the node
```

Gambar 3.9 Metode Perbaikan Proses Yang Terpotong

Metode mengenai perbaikan proses yang terpotong tersebut dapat dilihat pada Gambar 3.9. Baik metode untuk perbaikan proses yang terpotong di depan maupun perbaikan proses yang terpotong di belakang, penambahan aktivitas pengganti berdasarkan urutan pengeksekusian *node* pada model proses *tree.* Kemudian, sesuai metode yang diusulkan, apabila *node* yang dieksekusi merupakan operator, maka tidak semua *node* setelahnya yang berisi aktivitas akan digunakan sebagai aktivitas pengganti. Pemilihan aktivitas pengganti tergantung pada *node* operator sebelumnya.

Untuk metode perbaikan proses yang terpotong di bagian awal, *node* pada model proses *tree* akan dijalankan sesuai *preoreder reversed* dan start_node atau *node* awal yang dijalankan pada model proses *tree* adalah *node* yang berisi aktivitas pertama dari proses tersebut. Kemudian, penambahan aktivitas pengganti diletakkan pada awal proses. Sedangkan, pada metode perbaikan proses yang terpotong di bagian belakang, *node* pada model proses *tree* akan dijalankan sesuai *preoreder* dan start_node atau *node* awal yang dijalankan pada model proses *tree* adalah *node* yang berisi aktivitas akhir dari proses tersebut. Penambahan aktivitas pengganti pada metode perbaikan proses yang terpotong di bagian belakang terletak pada akhir proses. Pada *pseudocode* metode perbaikan proses yang terpotong, variable att_true menunjukkan apakah atribut *node* di model proses sesuai dengan atribut pada proses terpotong. Variabel att_true akan bernilai 1 apabila sesuai dan akan bernilai 0 apabila tidak sesuai. Kemudian, dikarenakan adanya kemungkinan tidak adanya atribut, maka akan dipilih satu *node* aktivitas (variabel child) dari operator relasi XOR "x" dan dua *node* aktivitas dari operator relasi OR "V" dimana *node* aktivitas yang dipilih adalah *node* aktivitas dengan nilai probabilitas kemunculan di log data tertinggi.

3.2 Keluaran

Tedapat tiga log data yang akan digunakan untuk pengujian, yaitu log data pada bulan Januari, Februari, dan Maret. Untuk pengujian hasil perbaikan proses yang terpotong, proses yang lengkap didapatkan dari penggabungan keseluruhan log data. Tedapar dua pengujian, yaitu pengujian kualitas model proses yang dibentuk dan pengujian akurasi hasil perbaikan proses yang terpotong. Model proses yang dibentuk oleh metode usulan akan dikomparasikan dengan algoritma *MinerFul*. Kemudian, pada pengujian hasil akurasi proses yang terpotong, hasil dari metode yang diusulkan akan dikomprasi dengan hasil dari algoritma *Heuristic Linear approach*. Algoritma *Heuristic Linear approach* dipilih sebagai metode komparasi karena metode ini adalah pengembangan metode *Repair Log,* merupakan metode untuk perbaikan proses yang memiliki aktivitas hilang, dengan menambahkan probabilitas kemunculan aktivitas pada log sebagai pertimbangan dalam pemilihan aktivitas di relasi pilihan.

(*halaman ini sengaja dikosongkan*)

BAB 4.
HASIL PENELITIAN DAN PEMBAHASAN

4.1 Hasil Penelitian

Data *sample* diujikan dengan menggunakan laptop dengan spesifikasi processor Intel® Core™ i5-3337U CPU @ 1.80 GHz dan memori 4 GB. Terdapat dua program yang digunakan, yaitu Eclipse Java Oxygen dan Python 2.7. Eclipse Java Oxygen untuk memodifikasi algoritma Declare Miner pada ProM 6 dengan menambahkan metode pembentukan *control-flow pattern*. Python digunakan untuk mengimplemetasikan metode pembentukan model *tree,* pendeteksian anomali dan proses yang terpotong, serta perbaikan proses yang terpotong.

4.2 Pre-Processing (Transformasi ke Log Data)

Database TPS yang digunakan dapat dilihat pada Gambar 4.1. Contoh kolom yang diolah bukan sebagai waktu dari aktivitas atau *message* adalah kolom Container_Key dan Container_Type. Kolom paling kiri adalah kolom Container_Key yang digunakan sebagai Case_ID dan kolom Container_Type akan digunakan sebagai salah satu data *detail attachment* pada log data. Kemudian, log data tersebut diolah menggunakan metode yang dijelaskan pada Sub Bab 3.1.4.1. Hasil dari pengolahan log data dapat dilihat pada Gambar 4.2. Pada log data yang tergambar di Gambar 4.2, *message* maupun aktivitas masih terekam menjadi satu di kolom nama aktivitas.

CONTA	CTR_SI	CTR_TY	GROSS	VESSEL_ATB	DISC_DATE	YARD_I	YARD_S	STACK_DATE
4484646	40	DRY	29.103	1/31/16 14:35	2/1/16 20:22	M	112	2/1/16 21:00
4484654	40	DRY	28.408	1/31/16 14:35	2/1/16 20:24	I	126	2/1/16 21:14
4485330	20	DRY	24.82	1/31/16 14:35	2/1/16 21:52	S	13	2/1/16 22:13
4484680	20	DRY	22.235	1/31/16 14:35	2/1/16 21:54	S	13	2/1/16 22:17
4484670	20	DRY	23.066	1/31/16 14:35	2/1/16 21:55	S	13	2/1/16 22:18
4484683	20	DRY	23.085	1/31/16 14:35	2/1/16 22:04	S	13	2/1/16 22:22
4484679	20	DRY	27.1	1/31/16 14:35	2/1/16 22:06	S	13	2/1/16 22:24
4484669	20	DRY	22.707	1/31/16 14:35	2/1/16 22:08	S	29	2/1/16 22:26
4484663	20	DRY	13.608	1/31/16 14:35	2/1/16 22:11	S	29	2/1/16 22:27
4484862	20	DRY	29.245	1/31/16 14:35	2/1/16 1:08	M	27	2/1/16 1:22
4485109	40	DRY	23.29	1/31/16 14:35	2/1/16 1:09	M	30	2/1/16 1:30
4484857	20	DRY	23.088	1/31/16 14:35	2/1/16 1:10	M	27	2/1/16 1:23

Gambar 4.1 Database Proses Impor Barang Terminal Peti Kemas

Case ID	Sender	Origina	Input	Activity	Output	Receiv	Time	Cost
4453421		Customer	NPWP, SI	Document_Entry_via_PDE	BC 2.0		23/01/2016 7:22	0
4453421	Customer		BC 2.0	Request_Behandle	BC 2.0	SKP	23/01/2016 7:23	0
4453421		TPS		Vessel_Berthing_Process			23/01/2016 10:10	47836,7172
4453421		TPS		Discharge_Container			23/01/2016 20:32	428,1331788
4453421		TPS		Bring_Container_to_Yard			23/01/2016 20:51	50
4453421		TPS		Stack_Container_in_Yard			23/01/2016 20:54	19,34
4453421	SKP		BC 2.0	Approve_Behandle	BC 2.0	Customer	24/01/2016 11:07	1,467092317
4453421		SKP	BC 2.0	Verification_Document_Behandle	BC 2.0		25/01/2016 0:55	6,210446525
4453421		Pejabat Be	LHP, BC 2.	Create_document_SPPB	SPPB		25/01/2016 7:48	3450,546591
4453421	Pejabat Bea Cukai		SPPB	Send_SPPB_Info		Customer	25/01/2016 7:48	0
4453421		Customer	SPPB	Create_Job_Order_Document_Delivery	CEIR		25/01/2016 15:33	45,79705642
4453421	Customer			Send_Job_Order_Delivery_Info		TPS	25/01/2016 15:35	0
4453421		Customer		Truck_in			27/01/2016 15:34	11,00157762
4453421		TPS		Dispatch_WQ_Delivery_to_CHE			27/01/2016 15:35	1,330475647
4453421		TPS		Determine_Container_Type			27/01/2016 15:38	1,810399109
4453421		TPS		Determining_Dry			27/01/2016 15:40	14,79
4453421		TPS		Decide_Task_Before_Lift_Container			27/01/2016 15:42	3,166661365
4453421		TPS		Lift_on_Container_Truck			27/01/2016 15:44	15,98
4453421		Customer		Truck_Go_To_Gate_Out			27/01/2016 16:30	1,992125684
4453421		TPS		Check_Container_before_Truck_out			27/01/2016 16:31	2,398721998
4453421		Customer		Truck_Out			27/01/2016 16:33	9,516587163

Gambar 4.2 Log Data Proses Impor Barang Terminal Peti Kemas

Agar dapat digunakan untuk pemodelan, log data di-*filter* terlebih dahulu dengan memisahkan antara *message* dan aktivitas. Gambar 4.3 menunjukkan log data yang sudah di-*filter* dari *message*. Jika dilihat pada Log yang sudah di-*filter,* terdapat beberapa aktivitas seperti *Request Behandle* dan *Approve Behandle* terhapus dari log. Itu menunjukkan bahwa aktivitas-aktivitas tersebut merupakan *message*. Kemudian, log yang sudah di-*filter* tersebut akan dikelompokkan menjadi Log Data Bulan Januari, Log Data Bulan Februari, dan Log Data Bulan Maret sebagai bahan uji coba pembentukan model dan perbaikan pada proses yang terpotong.

Case ID	Sender	Origina	Input	Activity	Output	Receiv	Time	Cost
4453421		Customer	NPWP, SI	Document_Entry_via_PDE	BC 2.0		1/23/16 7:22	0
4453421		TPS		Vessel_Berthing_Process			1/23/16 10:10	47836,72
4453421		TPS		Discharge_Container			1/23/16 20:32	428,1332
4453421		TPS		Bring_Container_to_Yard			1/23/16 20:51	50
4453421		TPS		Stack_Container_in_Yard			1/23/16 20:54	19,34
4453421		SKP	BC 2.0	Verification_Document_Behandle	BC 2.0		1/25/16 0:55	6,210447
4453421		Pejabat Be	LHP, BC 2.	Create_document_SPPB	SPPB		1/25/16 7:48	3450,547
4453421		Customer	SPPB	Create_Job_Order_Document_Delive	CEIR		1/25/16 15:33	45,79706
4453421		Customer		Truck_in			1/27/16 15:34	11,00158
4453421		TPS		Dispatch_WQ_Delivery_to_CHE			1/27/16 15:35	1,330476
4453421		TPS		Determine_Container_Type			1/27/16 15:38	1,810399
4453421		TPS		Determining_Dry			1/27/16 15:40	14,79
4453421		TPS		Decide_Task_Before_Lift_Container			1/27/16 15:42	3,166661
4453421		TPS		Lift_on_Container_Truck			1/27/16 15:44	15,98
4453421		Customer		Truck_Go_To_Gate_Out			1/27/16 16:30	1,992126
4453421		TPS		Check_Container_before_Truck_out			1/27/16 16:31	2,398722
4453421		Customer		Truck_Out			1/27/16 16:33	9,516587

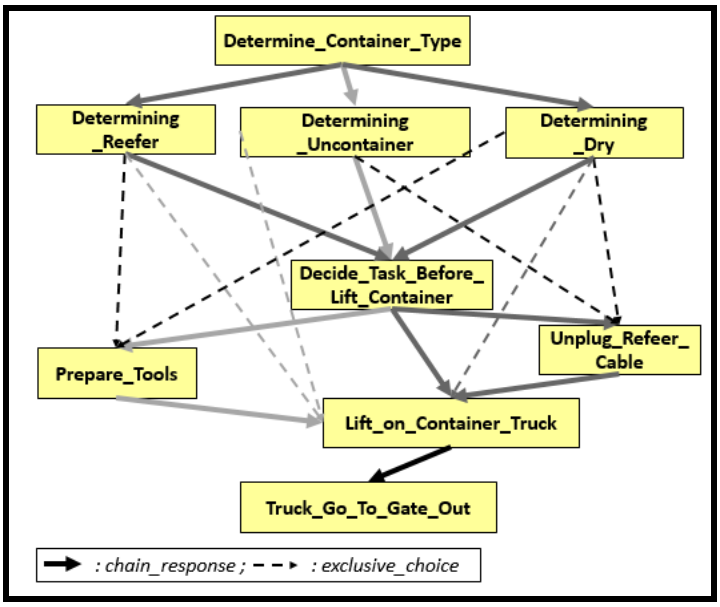
Gambar 4.3 Log Data Tanpa *Message* pada Proses Impor Barang Terminal Peti Kemas

4.3 Proses

4.3.1 Pembentukan Model Proses Imperatif dari *Rules* pada Model Deklaratif

Log data yang telah didapatkan dari *pre-processing* diproses dengan menggunakan algoritma Declare Miner (Maggi et al., 2011) untuk mendapatkan model deklaratif. Dalam pembentukan model deklaratif, terdapat penentuan minimum nilai *support* yang menjadi *threshold* untuk menentukan *rules* pembentuk model deklaratif. Pada penelitian ini, nilai *support* yang digunakan adalah 10%, 20%, dan 30%. Pemilihan nilai *support* hanya sampai 30% karena model proses imperatif yang dibentuk dari *rules* pada model deklaratif dengan nilai *support* 40% ke atas tidak dapat menggambarkan seluruh proses (nilai fitness yang didapat adalah 0). Hal ini dikarenakan *rules* pada model deklaratif yang terbentuk hanya menghasilkan relasi *sequence* pada model proses imperatif, padahal proses-proses dalam log data memiliki relasi *sequence* dan juga relasi paralel apabila dimodelkan.

Gambar 4.4 menunjukkan potongan model deklaratif yang menggunakan nilai *support* 10%. Potongan ini adalah kegiatan penentuan perlakuan pada barang impor sebelum diletakkan ke truk untuk dibawa keluar dari Terminal Peti Kemas. Dalam kegiatan ini, *rules* yang terbentuk di model deklaratif adalah *chain_response* dan *exclusive_choice*. Warna pada *rules* menggambarkan tingkat nilai *support* yang dimiliki. Warna akan semakin pudar apabila nilai *support* semakin mendekati batas minimum nilai *support* yang ditentukan.



Gambar 4.4 Potongan Hasil Model Deklaratif dari Algoritma *Declare Miner* dengan minimum *support* 10%

Rules dari model deklaratif yang telah terbentuk akan diproses oleh metode usulan untuk membentuk *control-flow pattern*. List *control-flow pattern* akan tersimpan dalam file dengan format .txt. Hasil dari *control-flow pattern* berdasarkan model deklaratif dengan nilai minimum *support* 10%, 20%, dan 30% dapat dilihat pada Tabel 4.1, Tabel 4.2, dan Tabel 4.3. Dari keseluruhan hasil *control-flow pattern* tersebut, *pattern non-free choice* terbentuk dari model deklaratif dengan nilai minimum *support* 10%. *Pattern non-free choice* digambarkan dalam *pattern* nomor 32 pada Tabel 4.1. Kemudian, *control-flow pattern* yang dibentuk berdasarkan model deklaratif dengan nilai minimum 30% tidak dapat menggambarkan beberapa aktivitas pada relasi pilihan, jika dibandingkan dengan *control-flow pattern* berdasarkan model deklaratif dengan nilai minimum 10% dan 20%. Hal ini terlihat dari jumlah *control-flow pattern* yang dihasilkan lebih sedikit dibandingkan lainnya.

Tabel 4.1 List *Control-Flow Pattern* dari Model Deklaratif 10%

No	Control-Flow Pattern
1	Firstactivity(Document_Entry_via_PDE)
2	Lastactivity(Truck_Out)
3	Document_Entry_via_PDE -> _O (Vessel_Berthing_Process)
4	Vessel_Berthing_Process -> _O (Discharge_Container)

5	Discharge_Container -> _O (Bring_Container_to_Yard)
6	Bring_Container_to_Yard -> _O (Stack_Container_in_Yard)
7	Verification_Document_Quarantine -> _O (Create_Job_Order_Document_Quarantine)
8	Create_Job_Order_Document_Quarantine -> _O (Bring_Container_from_Yard_to_Quarantine)
9	Bring_Container_from_Yard_to_Quarantine -> _O (Stack_Container_in_Quarantine_Area)
10	Stack_Container_in_Quarantine_Area -> _O (Check_Goods_Quarantine)
11	Check_Goods_Quarantine -> _O (Create_document_KH/KT)
12	Create_document_KH/KT -> _O (Send_Certificate_KH/KT_Info)
13	Send_Certificate_KH/KT_Info -> _O (Stack_Container_in_Yard_From_Quarantine)
14	Create_Job_Order_Document_Behandle -> _O (Stack_Container_in_Behandle_Area)
15	Stack_Container_in_Behandle_Area -> _O (Check_Goods_Behandle)
16	Check_Goods_Behandle -> _O (Create_document_LHP)
17	Create_document_LHP -> _O (Bring_Container_from_Yard_to_Behandle)
18	Bring_Container_from_Yard_to_Behandle -> _O (Stack_Container_in_Yard_From_Behandle)
19	Create_document_SPPB -> _O (Create_Job_Order_Document_Delivery)
20	Create_Job_Order_Document_Delivery -> _O (Truck_in)
21	Truck_in -> _O (Dispatch_WQ_Delivery_to_CHE)
22	Dispatch_WQ_Delivery_to_CHE -> _O (Determine_Container_Type)
23	Lift_on_Container_Truck -> _O (Truck_Go_To_Gate_Out)
24	Truck_Go_To_Gate_Out -> _O (Check_Container_before_Truck_out)
25	Check_Container_before_Truck_out -> _O (Truck_Out)
26	Stack_Container_in_Yard -> _O ((Verification_Document_Quarantine ⋁ Invisible_Task))
27	Verification_Document_Behandle -> _O ((Invisible_Task ⋁ Create_Job_Order_Document_Behandle))
28	Determine_Container_Type -> _O ((Determining_Dry ⋁ Determining_Refeer ⋁ Determining_Uncontainer))
29	_O ((Invisible_Task ⋁ Stack_Container_in_Yard_From_Quarantine)) -> _O (Verification_Document_Behandle)
30	_O ((Invisible_Task ⋁ Stack_Container_in_Yard_From_Behandle)) -> _O (Create_document_SPPB)

31	_O ((Invisible_Task \/ Unplug_Refeer_Cable \/ Prepare_Tools)) -> _O (Lift_on_Container_Truck)
32	<> (((Determining_Dry /\ Decide_Task_Before_Lift_Container /\ Invisible_Task) \/ (Determining_Refeer /\ Decide_Task_Before_Lift_Container /\ Unplug_Refeer_Cable) \/ (Determining_Uncontainer /\ Decide_Task_Before_Lift_Container /\ Prepare_Tools)))

Tabel 4.2 List *Control-Flow Pattern* dari Model Deklaratif 20%

No	**Control-Flow Pattern**
1	Firstactivity(Document_Entry_via_PDE)
2	Lastactivity(Truck_Out)
3	Document_Entry_via_PDE -> _O (Vessel_Berthing_Process)
4	Vessel_Berthing_Process -> _O (Discharge_Container)
5	Discharge_Container -> _O (Bring_Container_to_Yard)
6	Bring_Container_to_Yard -> _O (Stack_Container_in_Yard)
7	Verification_Document_Quarantine -> _O (Create_Job_Order_Document_Quarantine)
8	Create_Job_Order_Document_Quarantine -> _O (Bring_Container_from_Yard_to_Quarantine)
9	Bring_Container_from_Yard_to_Quarantine -> _O (Stack_Container_in_Quarantine_Area)
10	Stack_Container_in_Quarantine_Area -> _O (Check_Goods_Quarantine)
11	Check_Goods_Quarantine -> _O (Create_document_KH/KT)
12	Create_document_KH/KT -> _O (Send_Certificate_KH/KT_Info)
13	Send_Certificate_KH/KT_Info -> _O (Stack_Container_in_Yard_From_Quarantine)
14	Create_Job_Order_Document_Behandle -> _O (Stack_Container_in_Behandle_Area)
15	Stack_Container_in_Behandle_Area -> _O (Check_Goods_Behandle)
16	Check_Goods_Behandle -> _O (Create_document_LHP)
17	Create_document_LHP -> _O (Bring_Container_from_Yard_to_Behandle)
18	Bring_Container_from_Yard_to_Behandle -> _O (Stack_Container_in_Yard_From_Behandle)
19	Create_document_SPPB -> _O (Create_Job_Order_Document_Delivery)
20	Create_Job_Order_Document_Delivery -> _O (Truck_in)
21	Truck_in -> _O (Dispatch_WQ_Delivery_to_CHE)

22	Dispatch_WQ_Delivery_to_CHE -> _O (Determine_Container_Type)
23	Lift_on_Container_Truck -> _O (Truck_Go_To_Gate_Out)
24	Truck_Go_To_Gate_Out -> _O (Check_Container_before_Truck_out)
25	Check_Container_before_Truck_out -> _O (Truck_Out)
26	Stack_Container_in_Yard -> _O ((Invisible_Task ⋁ Verification_Document_Quarantine))
27	Verification_Document_Behandle -> _O ((Create_Job_Order_Document_Behandle ⋁ Invisible_Task))
28	Determine_Container_Type -> _O ((Determining_Dry ⋁ Determining_Refeer))
29	Decide_Task_Before_Lift_Container -> _O ((Invisible_Task ⋁ Unplug_Refeer_Cable))
30	_O ((Stack_Container_in_Yard_From_Quarantine ⋁ Invisible_Task)) -> _O (Verification_Document_Behandle)
31	_O ((Invisible_Task ⋁ Stack_Container_in_Yard_From_Behandle)) -> _O (Create_document_SPPB)
32	_O ((Determining_Refeer ⋁ Determining_Dry)) -> _O (Decide_Task_Before_Lift_Container)
33	_O ((Invisible_Task ⋁ Unplug_Refeer_Cable)) -> _O (Lift_on_Container_Truck)

Tabel 4.3 List *Control-Flow Pattern* dari Model Deklaratif 30%

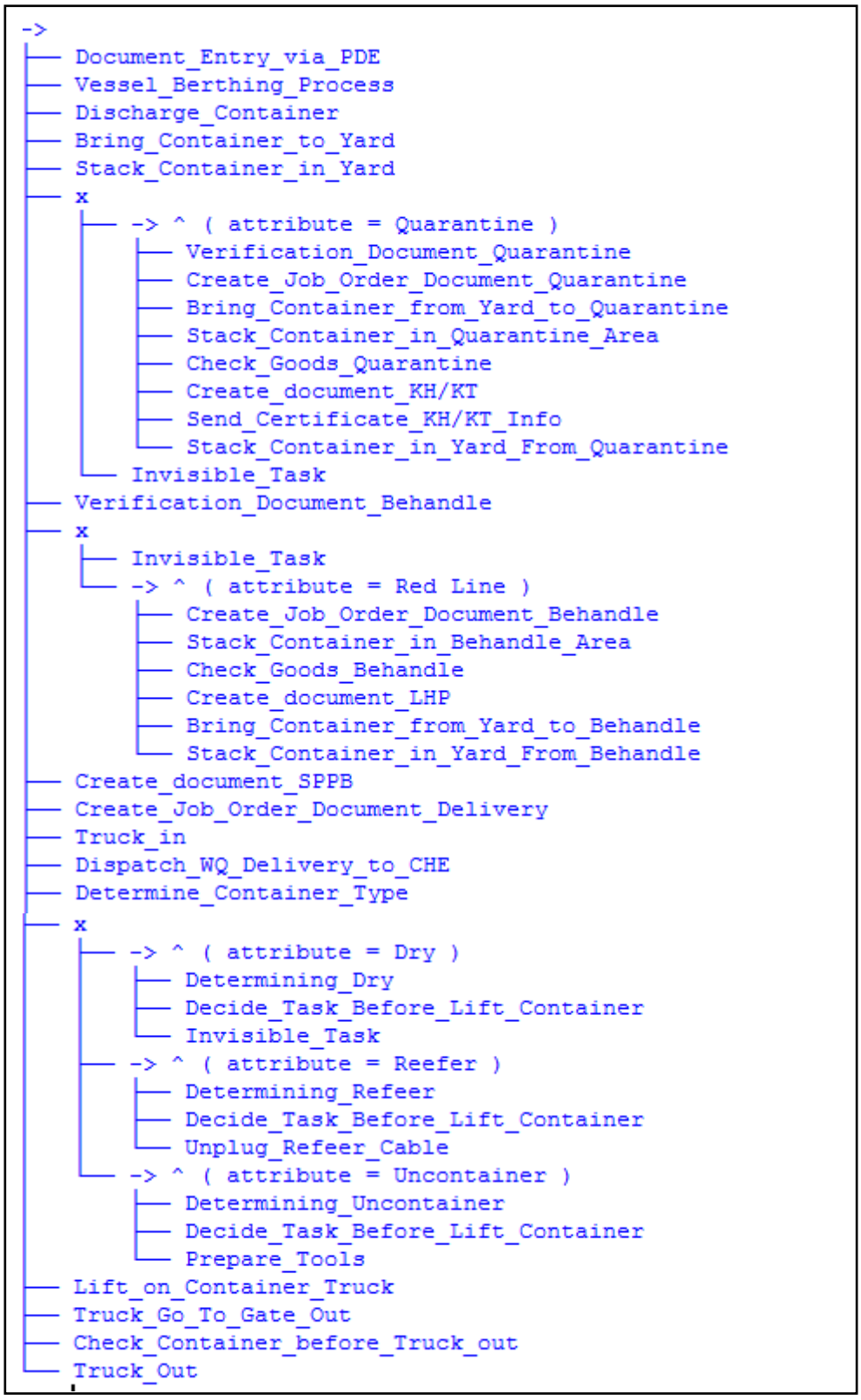
No	**Control-Flow Pattern**
1	Firstactivity(Document_Entry_via_PDE)
2	Lastactivity(Truck_Out)
3	Document_Entry_via_PDE -> _O (Vessel_Berthing_Process)
4	Vessel_Berthing_Process -> _O (Discharge_Container)
5	Discharge_Container -> _O (Bring_Container_to_Yard)
6	Bring_Container_to_Yard -> _O (Stack_Container_in_Yard)
7	Stack_Container_in_Yard -> _O (Verification_Document_Behandle)
8	Verification_Document_Behandle -> _O (Create_document_SPPB)
9	Create_document_SPPB -> _O (Create_Job_Order_Document_Delivery)
10	Create_Job_Order_Document_Delivery -> _O (Truck_in)
11	Truck_in -> _O (Dispatch_WQ_Delivery_to_CHE)
12	Dispatch_WQ_Delivery_to_CHE -> _O (Determine_Container_Type)
13	Lift_on_Container_Truck -> _O (Truck_Go_To_Gate_Out)

14	Truck_Go_To_Gate_Out -> _O (Check_Container_before_Truck_out)
15	Check_Container_before_Truck_out -> _O (Truck_Out)
16	Determine_Container_Type -> _O ((Determining_Dry ⋁ Determining_Refeer))
17	Decide_Task_Before_Lift_Container -> _O ((Unplug_Refeer_Cable ⋁ Invisible_Task))
18	_O ((Determining_Refeer ⋁ Determining_Dry)) -> _O (Decide_Task_Before_Lift_Container)
19	_O ((Unplug_Refeer_Cable ⋁ Invisible_Task)) -> _O (Lift_on_Container_Truck)

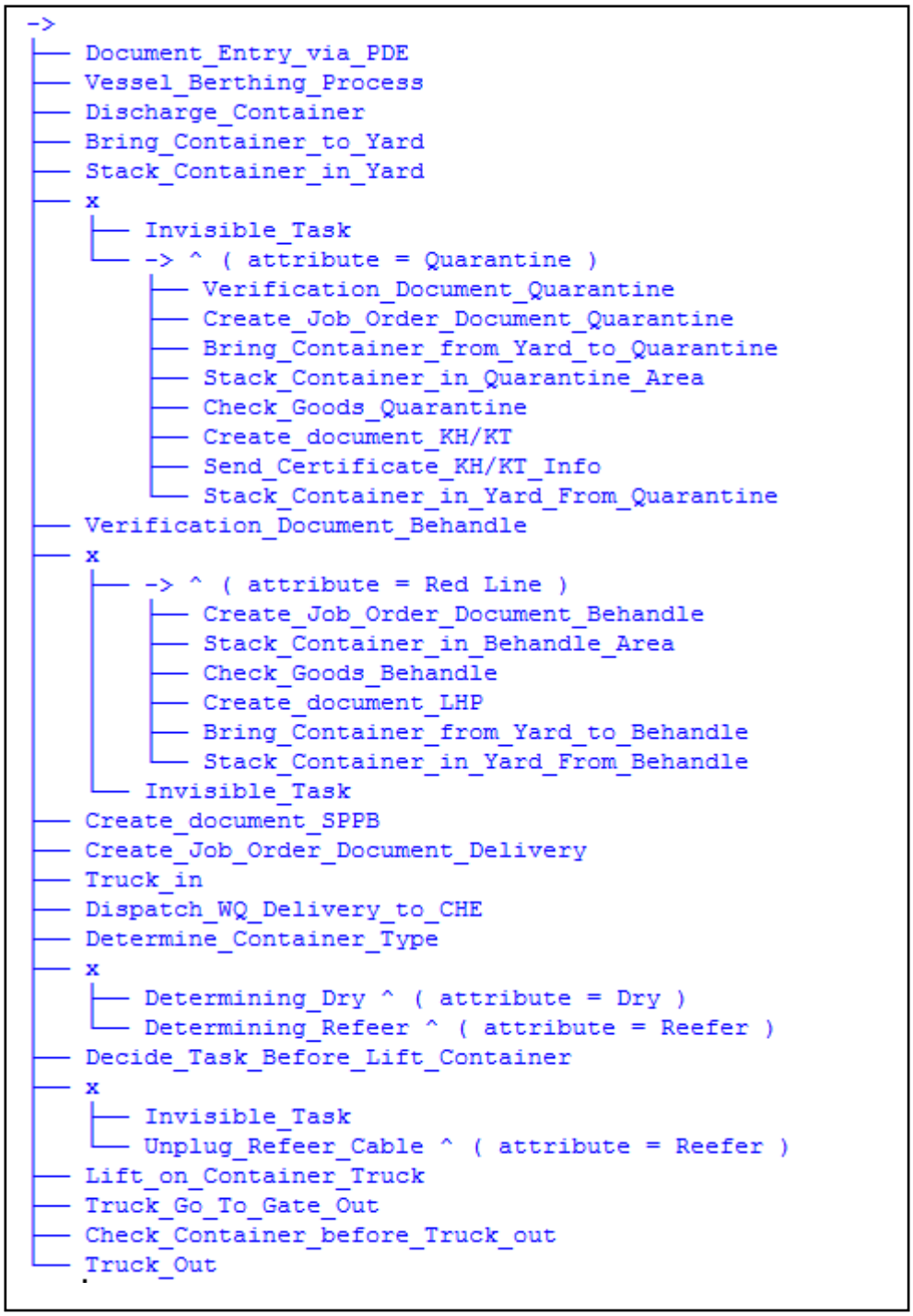
4.3.2 Pembentukan Model *Tree* dari *Control-Flow Pattern*

Model proses *tree* dibentuk dengan menggabungkan *control-flow pattern* yang telah diperoleh dan penambahan atribut pada model proses *tree*. Hasil model *tree* berdasarkan *control-flow pattern* yang didapatkan di subbab 4.3.1 dapat dilihat pada Gambar 4.5, Gambar 4.6, dan Gambar 4.7. Dari kesemua model proses *tree* tersebut, model proses *tree* paling sederhana adalah model proses yang tergambar di Gambar 4.7. Jika dibandingkan dengan model proses yang lain, model proses pada Gambar 4.7 tidak menggambarkan relasi pilihan pada kegiatan di Karantina (*Quarantine*) dan kegiatan di Behandle.

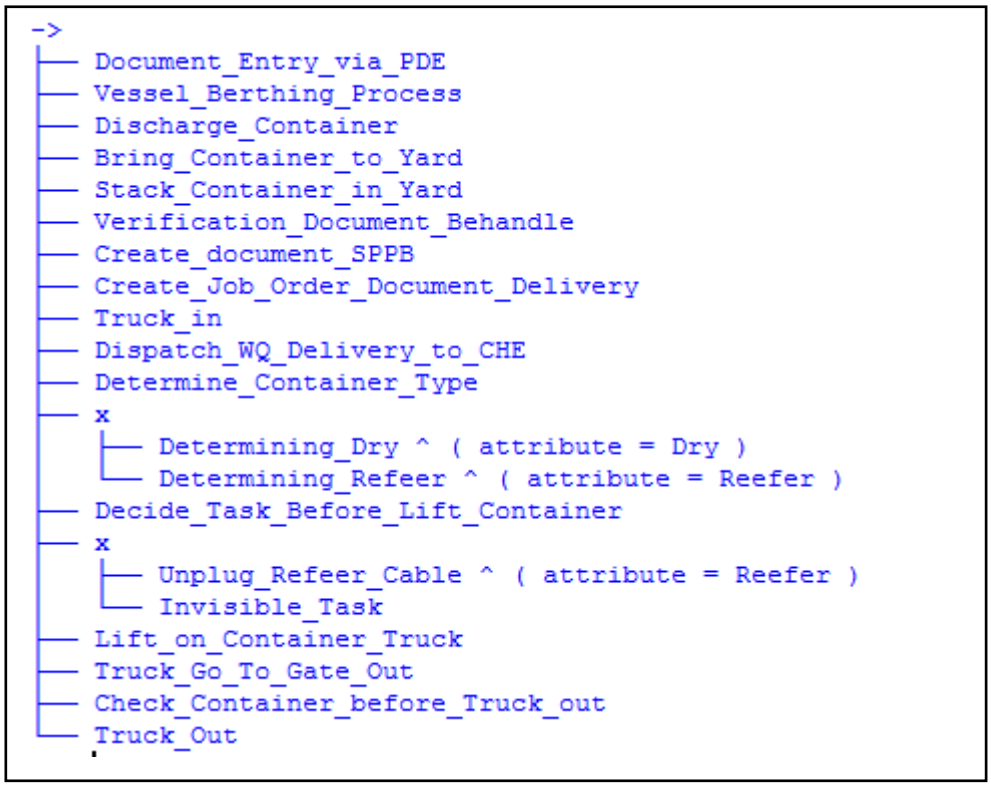
Kemudian, penambahan atribut pada model proses terlihat dari keterangan *attribute* di model proses. Penulisan atribut pada model proses tersebut pada penelitian ini berformat “ ^ (attribute = *nama_atribut*)”. Hal ini dilakukan untuk membedakan aktivitas dan atribut pendukung.



Gambar 4.5 Potongan Model *Tree* dari *Control-Flow Patterns* dengan minimum *support* 10%



Gambar 4.6 Potongan Model *Tree* dari *Control-Flow Patterns* dengan minimum *support* 20%



Gambar 4.7 Potongan Model *Tree* dari *Control-Flow Patterns* dengan minimum *support* 30%

Pada perbaikan proses yang terpotong, model proses imperatif yang dijadikan acuan adalah satu model. Pemilihan model proses yang dilakukan dengan mencari nilai kualitas tertinggi dari keseluruhan model proses tersebut. Kualitas diukur dari segi *fitness*, presisi, *simplicity*, dan generalisasi. Model proses yang dipilih adalah model proses yang memiliki nilai diatas 0,5 untuk keseluruhan kualitas. 0,5 dipilih karena nilai tersebut batas tengah dari rentang nilai pengukuran kualitas. Apabila model proses yang memenuhi kriteria tersebut lebih dari satu atau tidak memenuhi, maka akan dipilih rata-rata kualitas tertinggi dari seluruh kualitas tersebut.

Hasil kualitas model proses *tree* dapat dilihat pada Tabel 4.4. Nilai yang dicetak tebal merupakan nilai tertinggi dari setiap kualitas per bulannya. Model proses dengan nilai minimum *support* 10% memiliki nilai *fitness* dan nilai presisi tertinggi, model proses dengan nilai minimum *support* 20% memiliki nilai *simplicity* tertinggi, sedangkan model proses dengan nilai minimum *support* 30% memiliki nilai generalisasi tertinggi. Meskipun setiap model proses memiliki nilai tertinggi di bidang tertentu,

model proses dengan nilai minimum *support* 10% yang dipilih. Hal itu dikarenakan hanya model tersebut yang memiliki nilai di atas 0,5 untuk keseluruhan kualitas. Perhitungan lengkap dapat dilihat pada LAMPIRAN B.

Tabel 4.4 Hasil Kualitas Model Proses *Tree*

		Fitness (Qf)	Presisi (Qp)	*Simplicity* (Qs)	Generalisasi (Qg)
		(0,0-1,0)	**(0,0-1,0)**	**(0,0-1,0)**	**(0,0-1.0)**
Januari	10%	**0,925**	**0.92**	0.878	0.976
	20%	0,923	0.5	**0.884**	0.986
	30%	0,699	0.5	0.261	**0.990**
Februari	10%	**0.844**	**0.83**	0.878	0.973
	20%	0.842	0.5	**0.884**	0.985
	30%	0.624	0.5	0.261	**0.988**
Maret	10%	**0.914**	**0.83**	0.878	0.975
	20%	0.912	0.5	**0.884**	0.985
	30%	0.666	0.5	0.261	**0.988**

4.3.3 Penentuan Proses yang Terpotong dan Anomali

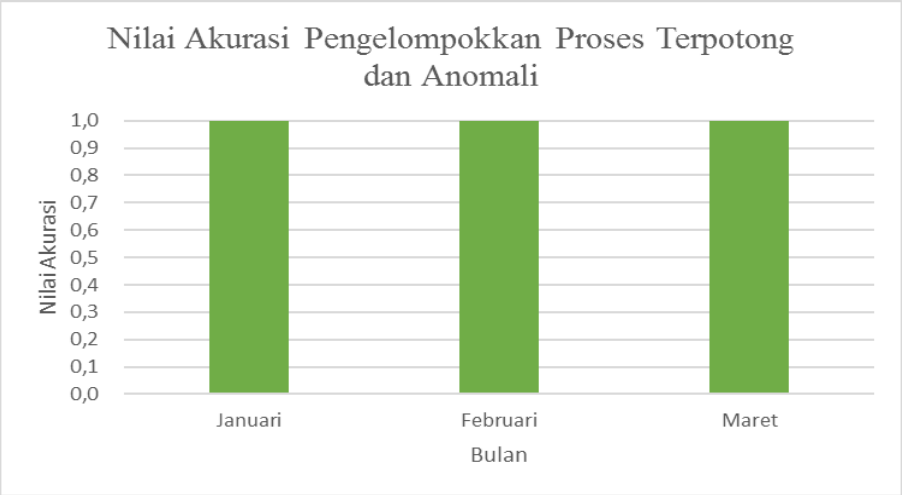
Penentuan proses yang terpotong dan anomali didasarkan pada model proses yang telah terbentuk dari penggabungan *control-flow pattern*. Metode untuk penentuan kelompok proses tersebut telah dijelaskan pada subbab 3.1.6.1. Hasil dari pengaplikasian metode tersebut dapat dilihat pada Gambar 4.8. Angka-angka yang disimpan pada kelompok proses yang terpotong dan anomali adalah case_id dari proses. Case_id tersebut akan digunakan sebagai acuan untuk memilih proses yang akan diperbaiki. Proses yang diperbaiki hanyalah proses yang termasuk dalam kelompok proses yang terpotong (list *case truncated*).

```
List case truncated :
[4453467, 4453527, 4453646, 4453782, 4453882, 4453885, 4453913, 4454007, 4454028,
 4454053, 4454063, 4454267, 4454291, 4454305, 4454309, 4454313, 4455590, 4456000,
 4456143, 4456408]
List case anomali :
[4453467, 4453527, 4453646, 4453782, 4453882, 4453885, 4453913, 4454007, 4454028,
 4454053, 4454063, 4454267, 4454291, 4454305, 4454309, 4454313, 4455590, 4456000,
 4456143, 4456408]
```

Gambar 4.8 List *case* yang termasuk proses yang terpotong (*truncated*) dan anomali

Gambar 4.9 menunjukkan hasil akurasi pengelompokkan proses yang terpotong dan anomali. Jumlah prose yang terpotong dan anomali sebenarnya didapatkan dari pengecekan proses sesuai *trace* yang diberikan oleh *expert*. List dari *trace* serta penjabaran penentuan nilai akurasi dapat dilihat pada LAMPIRAN C.

Berdasarkan Gambar 4.9, nilai akurasi dari seluruh bulan tersebut adalah 1,0. Nilai akurasi yang sangat tinggi disebabkan karena proses-proses yang terbentuk di model proses adalah proses-proses yang dikategorikan sebagai proses yang benar oleh *expert*. Oleh sebab itu, baik buruknya hasil pengelompokkan proses yang terpotong dan anomali bergantung pada model proses yang terbentuk.



Gambar 4.9 Perhitungan Akurasi Pengelompokkan Proses Terpotong dan Anomali

4.3.4 Perbaikan Proses yang Terpotong

Perbaikan proses yang terpotong berdasarkan list proses yang terpotong. Perbaikan proses yang terpotong didasarkan pada metode yang diusulkan. Metode tersebut telah dijelaskan pada subbab 3.1.6.2. Hasil dari pengaplikasian metode ini adalah urutan aktivitas yang membentuk proses yang lengkap. Contoh hasil dapat dilihat pada Gambar 4.10. *Case* 4453467 hanya memiliki aktivitas *Documen_Entry_via_PDE,Vessel_Berthing_Process, Discharge_Container, Bring_Container_to_Yard, dan Stack_Container_in_Yard* pada bulan Januari. Selain itu, *case* ini memiliki atribut yaitu “Dry, Green Line”. Berdasarkan aktivitas dan atribut

pada *case*, penambahan aktivitas sebagai pengganti aktivitas yang hilang pada Gambar 4.10 ditunjukkan mulai dari aktivitas setelah Stack_Container_in_Yard sampai aktivitas Truck_Out.

```
case : 4453467
['Document_Entry_via_PDE', 'Vessel_Berthing_Process', '
Discharge_Container', 'Bring_Container_to_Yard', 'Stack
_Container_in_Yard', 'Verification_Document_Behandle',
'Create_document_SPPB', 'Create_Job_Order_Document_Deli
very', 'Truck_in', 'Dispatch_WQ_Delivery_to_CHE', 'Dete
rmine_Container_Type', 'Determining_Dry', 'Decide_Task_
Before_Lift_Container', 'Lift_on_Container_Truck', 'Tru
ck_Go_To_Gate_Out', 'Check_Container_before_Truck_out',
'Truck_Out']
```

Gambar 4.10 Contoh Hasil Perbaikan Proses yang Terpotong

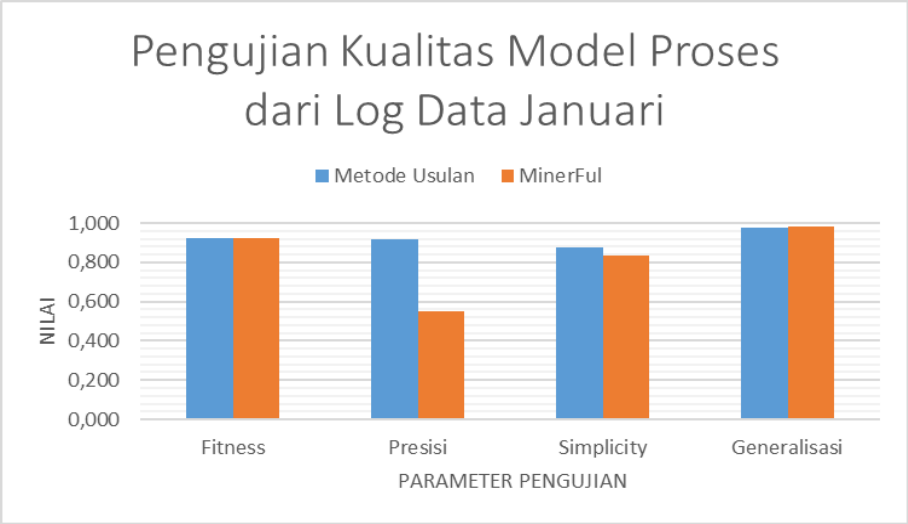
4.4 Pembahasan

4.4.1 Evaluasi Kinerja Pembentukan Model Proses

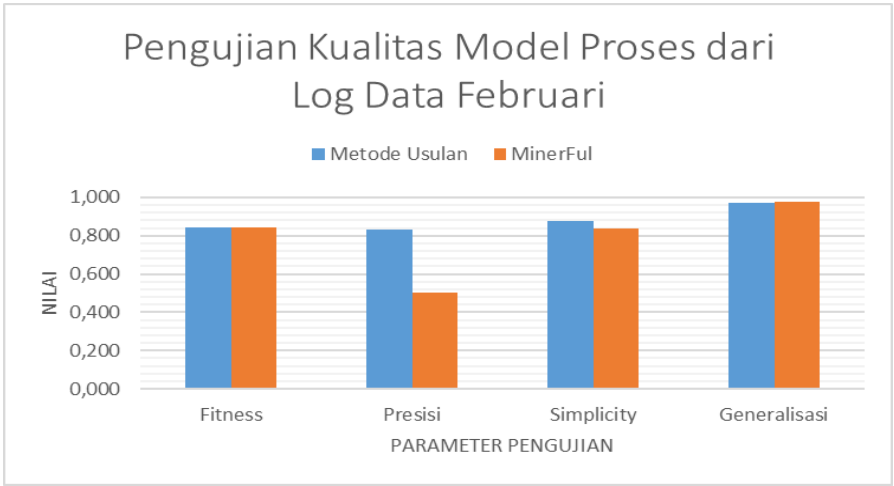
Evaluasi kinerja pembentukan model proses imperatif berdasarkan empat sisi kualitas, yaitu *fitness,* presisi, *simplicity*, dan generalisasi. Evaluasi yang dilakukan dengan membandingkan kualitas hasil metode yang diusulkan dan kualitas hasil algoritma *MinerFul*. Algoritma *MinerFul* adalah metode yang dapat mengkonversi model deklaratif menjadi model proses, yang dapat disebut sebagai model proses imperatif. Hasil metode akan semakin bagus apabila nilai kualitas yang didapatkan tinggi. Rentang nilai kualitas adalah 0,0 sampai 1,0. Untuk hasil algoritma *MinerFul* dapat dilihat pada LAMPIRAN E.

Hasil evaluasi kinerja dapat dilihat pada Gambar 4.11, Gambar 4.12, dan Gambar 4.13. Detail perhitungan nilai kualitas dapat dilihat pada LAMPIRAN D. Dari hasil evaluasi tersebut, hasil metode yang diusulkan memiliki nilai presisi dan *simplicity* yang lebih tinggi dari hasil algoritma *MinerFul*. Kemudian, nilai *fitness* dari hasil metode yang diusulkan maupun hasil dari algoritma *MinerFul* memiliki nilai yang tinggi. Nilai tinggi dalam presisi dan *simplicity* dipengaruhi oleh kemampuan metode yang diusulkan dalam mendeteksi relasi *non-free choice* dan *invisible task.* Algoritma *MinerFul* tidak

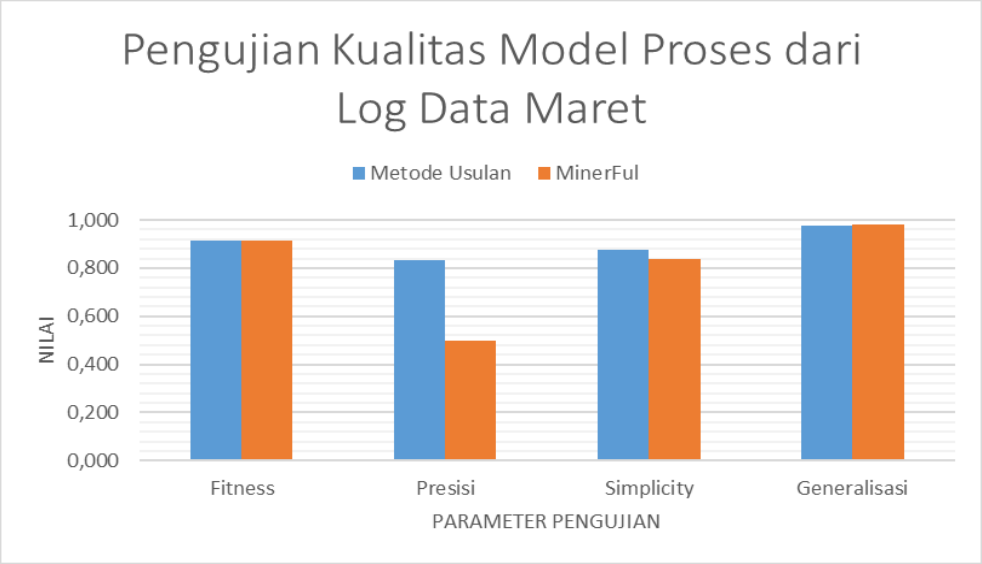
dapat mendeteksi *non-free choice* sehingga model proses yang dihasilkan memiliki *trace* yang semakin bervariasi. Pembagi dari perhitungan presisi adalah jumlah *trace* pada model proses, sehingga semakin banyak variasi *trace* yang dibentuk model proses dan tidak muncul dalam log data akan memperkecil nilai presisi. Kemudian, algoritma *MinerFul* tidak mendeteksi *invisible task* sehingga banyak aktivitas yang didefinisikan berulang (redundan). Pendefinisian aktivitas secara redundan ini mengurangi nilai *simplicity.*



Gambar 4.11 Kualitas Model Proses dari Metode yang Diusulkan dan *MinerFul* pada bulan Januari



Gambar 4.12 Kualitas Model Proses dari Metode yang Diusulkan dan *MinerFul* pada bulan Februari



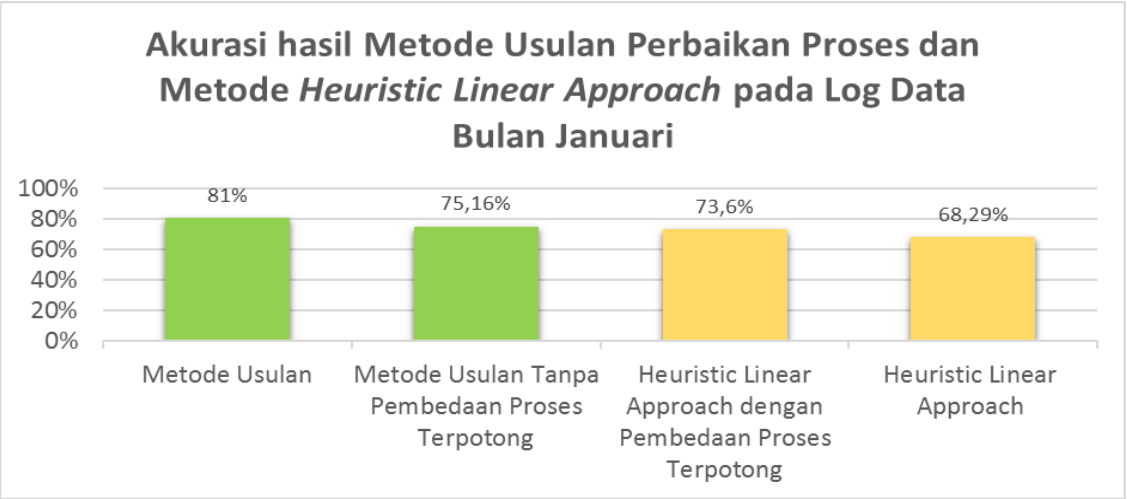
Gambar 4.13 Kualitas Model Proses dari Metode yang Diusulkan dan *MinerFul* pada bulan Maret

4.4.2 Evaluasi Kinerja Perbaikan Proses yang Terpotong

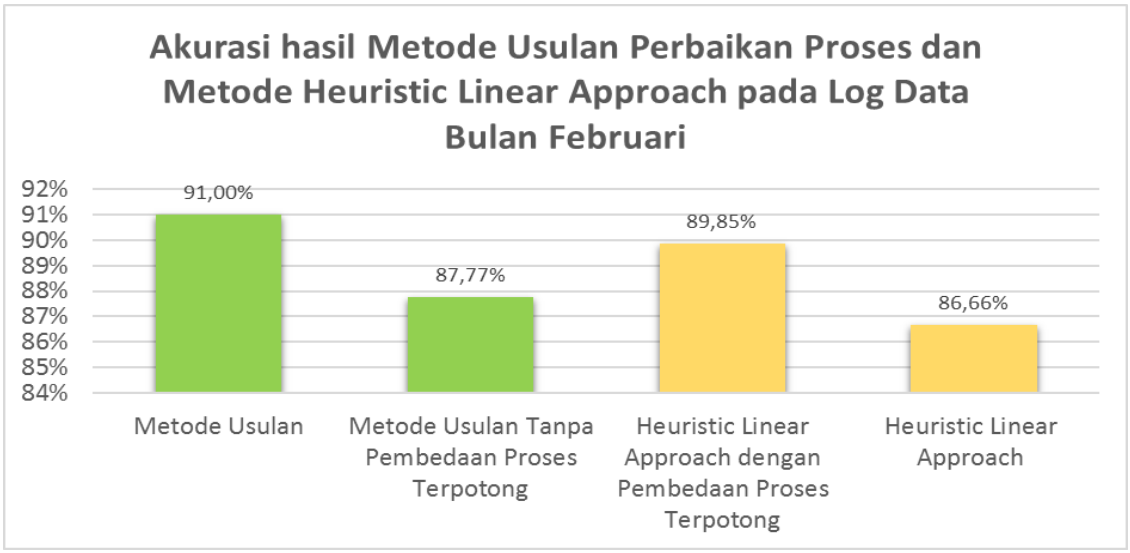
Evaluasi kinerja perbaikan proses yang terpotong dilakukan dengan cara menghitung akurasi dari hasil metode yang diusulkan. Penilaian akurasi adalah meghitung prosentase hasil yang sesuai dengan data sebenarnya. Data sebenarnya diambil dari gabungan log data, mulai dari bulan Januari sampai dengan bulan Maret. Pada bulan Januari, terdapat 1504 proses terpotong dari 21710 *case* keseluruhan. Pada bulan Februari, terdapat 3509 proses terpotong dari 23335 *case* keseluruhan. Sedangkan pada bulan Maret, terdapat 2006 proses terpotong dari 24557 *case* keseluruhan.

Hasil akurasi tersebut akan dibandingkan dengan metode pembanding, yaitu algoritma *Heuristic Linear Approach*. Hasil akurasi dapat dilihat pada Gambar 4.14, Gambar 4.15, dan Gambar 4.16. Dari keseluruhan log data tersebut, metode usulan memiliki nilai akurasi paling tinggi, yaitu 81% pada bulan Januari, 91% pada bulan Februari, dan 74,48% pada bulan Maret. Dari hasil evaluasi, didapatkan bahwa metode usulan dalam membedakan proses yang terpotong dan anomali mampu meningkatkan akurasi dari metode. Hal itu terbukti dari penambahan metode tersebut ke dalam algoritma *Heuristic Linear Approach* mengakibatkan nilai akurasi naik dibandingkan dengan hanya menggunakan algoritma *Heuristic Linear Approach*. Kemudian,

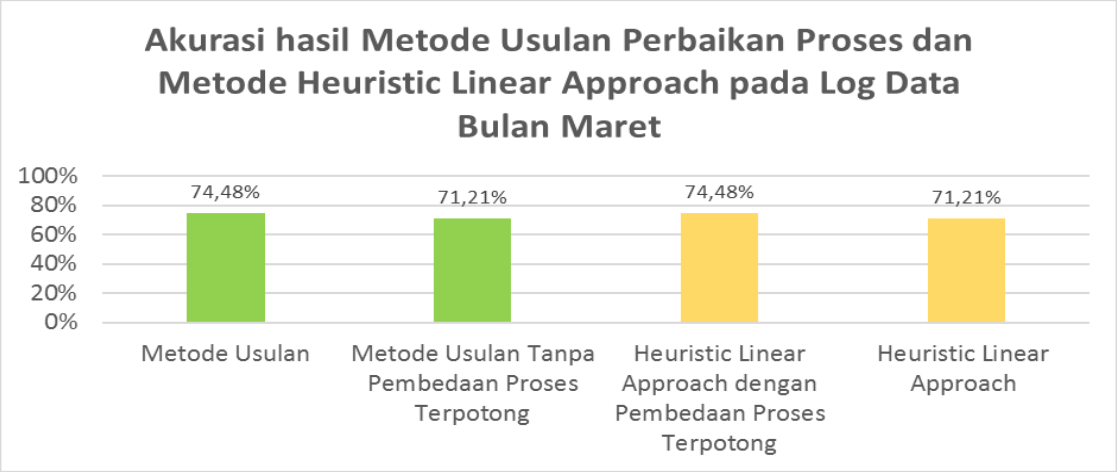
penambahan atribut pada model proses meningkatkan akurasi hasil perbaikan proses yang terpotong.



Gambar 4.14 Kualitas Hasil Perbaikan Proses Terpotong dari Metode yang Diusulkan dan *Heuristic Linear Approach* pada bulan Januari



Gambar 4.15 Kualitas Hasil Perbaikan Proses Terpotong dari Metode yang Diusulkan dan *Heuristic Linear Approach* pada bulan Februari



Gambar 4.16 Kualitas Hasil Perbaikan Proses Terpotong dari Metode yang Diusulkan dan *Heuristic Linear Approach* pada bulan Maret

Kemudian, dari sisi *running time* algoritma, metode yang diusulkan memiliki *running time* lebih kecil dibandingkan algoritma *Heuristic Linear Approach*. Baik Metode Usulan maupun algoritma *Heuristic Linear Approach* memiliki *running time* terkecil pada bulan Januari, dikarenakan bulan Januari memiliki *case* paling sedikit dan proses terpotong paling sedikit. Hasil rincian dapat dilihat pada Tabel 4.5.

Tabel 4.5 Hasil *Running Time* Metode Usulan dan Algoritma *Heuristic Linear Approach*

Log Bulan	**Algoritma**	***Running Time* (detik)**
Januari	**Metode Usulan**	**89,92**
	Metode Usulan Tanpa Pembedaan Proses Terpotong	53,30
	Heuristic Linear Approach dengan Pembedaan Proses Terpotong	103,74
	Heuristic Linear Approach	**235,62**
Februari	**Metode Usulan**	**91,49**
	Metode Usulan Tanpa Pembedaan Proses Terpotong	63,19
	Heuristic Linear Approach dengan Pembedaan Proses Terpotong	113,38
	Heuristic Linear Approach	**255,53**
Maret	**Metode Usulan**	**97,40**
	Metode Usulan Tanpa Pembedaan Proses Terpotong	64,37
	Heuristic Linear Approach dengan Pembedaan Proses Terpotong	115,97
	Heuristic Linear Approach	**267,12**

4.4.3 Analisa Hasil

Dari hasil eksperimen yang dilakukan, didapatkan bahwa model proses imperatif yang memiliki rata-rata nilai kualitas terbaik (utamanya adalah nilai *fitness* dan presisi) menggunakan nilai *threshold* minimum *support* untuk masing-masing bulan sebesar 10%. Hal ini dikarenakan semakin kecil nilai minimum *support*, maka semakin besar *rules* yang terbentuk di model Deklaratif sehingga semakin banyak masukan yang dapat digunakan oleh metode yang diusulkan untuk membentuk *control-flow patterns* sebagai bahan pembentukan moel proses. Kemudian, hasil kualitas presisi dan *simplicity* model proses dari metode yang diusulkan lebih tinggi dibandingkan hasil kualitas model proses dari algoritma *MinerFul*. Hal ini dikarenakan pembentukan *invisible task* dapat mengurangi aktivitas yang redundan, sehinggan nilai *simplicity* lebih tinggi. Kemudia, pembentukan *non-free choice* menghilangkan kemungkinan *trace* yang terbentuk tetapi tidak terdapat di log data, sehingga nilai presisi menjadi lebih tinggi.

Kemudian, pengukuran kualitas perbaikan proses yang terpotong menggunakan tiga log data, yaitu log data bulan Januari yang mengandung proses terpotong di akhir, log data bulan Februari yang mengandung proses terpotong di awal dan proses terpotong di akhir, serta log data bulan Maret yang mengandung proses terpotong di awal. Hasil akurasi terbaik perbaikan proses menggunakan metode yang diusulkan pada bulan Februari. Hal ini dikarenakan prosentase aktivitas hilang pada proses yang terpotong di awal dan prosentase aktivitas hilang pada proses yang terpotong di akhir lebih kecil jika dibandingkan log bulan lain. Hal ini terlihat pada Tabel 4.6. Kemudian, atribut yang digunakan pada metode usulan berada pada aktivitas-aktivitas awal, sehingga mteode yang diusulkan dapat memperbaiki proses yang terpotong di akhir lebih baik dibandingkan proses yang terpotong di awal. Hal ini terlihat dari akurasi yang tinggi pada log bulan Januari dan Februari. Kemudian, akurasi log bulan Maret tidak tinggi dikarenakan proses yang terpotong di depan sehingga metode menggunakan probabilitas kemunculan aktivitas untuk menanggulangi atribut yang sedikit. Penggunaan probabilitas ini yang membuat nilai akurasi metode usulan dan nilai akurasi algoritma *Heuristic Linear Approach* terhadap seluruh proses terpotong yang merupakan hasil dari metode usulan untuk pembedaan proses terpotong dan anomali memiliki hasil akurasi yang sama.

Tabel 4.6 Prosentase Aktivitas Hilang pada Setiap Log Bulan

Log Bulan	**Jenis Proses Terpotong**	**Prosentase Aktivitas Hilang (%)**
Januari	Awal	60,42
Februari	Awal	51,56
	Awal dan Akhir	66,67
	Akhir	38,35
Maret	Akhir	46,88

Pada pengukuran *running time* metode yang diusulkan dengan algoritma *Heuristic Linear Approach*, *running time* metode yang diusulkan lebih cepat dibandingkan algoritma *Heuristic Linear Approach*. Perbedaan *running time* yang cukup besar dikarenakan algoritma *Heuristic Linear Approach* melakukan pengecekan per pasangan aktivitas(aktivitas pertama dan aktivitas kedua, aktivitas kedua dan aktivitas ketiga, dst) untuk setiap *case* dalam perbaikan proses dengan jumlah *trace* acuan yang besar, yaitu 36 *trace*, sedangkan metode usulan hanya melakukan pengecekan pada aktivitas awal dan akhir pada *case* yang dikategorikan terpotong untuk perbaikan proses. *Trace* acuan yang digunakan dapat dilihat pada LAMPIRAN F. Jumlah *trace* yang besar sebagai acuan algoritma *Heuristic Linear Approach* dikarenakan model proses yang digunakan tidak dapat membentuk *non-free choice* dalam relasi yang mengandung *invisible task* (dalam kasus TPS adalah relasi dari Determine_Container_Type sampai Lift_on_Container_Truck). Kemudian, algoritma *Heuristic Linear Approach* yang menggunakan metode pembedaan anomali dan proses terpotong untuk memilih *case* yang akan diperbaiki menghasilkan *running time* lebih cepat dikarenakan *case* yang diproses hanya proses yang terpotong dan perbaikan proses membutuhkan waktu yang lebih besar dibandingkan dengan pengecekan anomali dan proses terpotong.

BAB 5.
KESIMPULAN DAN SARAN

5.1 Kesimpulan

1. Penggambaran relasi *non-free choice* dan *invisible task* dalam model proses imperatif dilakukan dengan penggambaran *control-flow pattern* sebagai bagian model proses. Relasi *non-free choice* dan *invisible task* digambarkan dengan melihat hubungan antar *pattern* untuk relasi paralel (XOR, OR, AND) yang telah terbentuk dimana *pattern* relasi paralel dibentuk dari hubungan antar *rules* pada model deklaratif.

2. Perbaikan proses terpotong dilakukan dengan membedakan proses terpotong dan anomali kemudian memberikan aktivitas pengganti dari aktivitas hilang pada proses terpotong berdasarkan urutan aktivitas dalam *node* pada model *tree*. Pada proses yang terpotong di bagian akhir, urutan *node* dalam bentuk *pre-order*, dan proses yang terpotong di bagian awal menggunakan model urutan *reverse* dari *pre-order*. Kemudian, untuk relasi paralel pilihan (XOR, OR), aktivitas yang dipilih adalah aktivitas yang sesuai dengan atribut (dalam penelitian diambil dari *detail attachment*) yang terdapat dalam log dari proses terpotong.

3. Hasil evaluasi dilakukan dengan membandingkan hasil metode usulan dan hasil dari algoritma *MinerFul*. Kemampuan penggambaran relasi *non-free choice* dan *invisible task* dalam model hasil metode usulan dapat menghasilkan nilai presisi dan *simplicity* lebih tinggi dari hasil algoritma *MinerFul.* Hasil rata-rata presisi dan *simplicity* dari metode usulan menggunakan log Terminal Peti Kemas adalah 0,861 dan 0,878, sedangkan rata-rata presisi dan *simplicity* dari algoritma *MinerFul* adalah 0,517 dan 0,837.

4. Hasil evaluasi dilakukan dengan membandingkan hasil metode usulan dan hasil dari algoritma *Heuristic Linear Approach* pada tiga log data, yaitu log data bulan Januari, log data bulan Februari dan log data bulan Maret. Log data bulan Februari memiliki hasil akurasi paling tertinggi karena log berisi proses yang terpotong di akhir dengan prosentase aktivitas yang kecil, sehingga banyak atribut

yang didapatkan untuk menunjang metode perbaikan yang diusulkan. Sedangkan, log data bulan Maret memiliki hasil akurasi paling rendah karena log berisi proses yang terpotong di awal, dimana atribut banyak terdapat pada aktivitas-aktivitas awal sehingga metode perbaikan yang diusulkan menggunakan probabilitas kemunculan aktivitas dalam penentuan aktivitas pengganti. Penggunaan probabilitas tersebut membuat akurasi metode yang diusulkan dan akurasi algoritma *Heuristic Linear Approach* dengan tambahan metode usulan untuk pembedaan proses terpotong dan anomali memiliki hasil akurasi yang sama. Untuk rata-rata akurasi pada tiga log data, metode yang diusulkan menghasilkan nilai akurasi lebih tinggi dari algoritma *Heuristic Linear Approach*, yaitu 82,16%. Sedangkan, dari sisi *running time*, metode yang diusulkan memiliki *running time* lebih cepat dibandingkan dengan algoritma *Heuristic Linear Approach* dikarenakan algoritma *Heuristic Linear Approach* mengecek pasangan aktivitas dalam setiap *case* untuk perbaikan proses, sedangkan metode usulan hanya mengecek aktivitas awal dan akhir pada setiap *case* yang sudah dikategorikan sebagai proses terpotong.

5.2 Saran

Pencarian data keterangan lain yang digunakan untuk pemilihan relasi pilihan selain keterangan pada aktivitas dokumen dapat digunakan untuk meningkatkan akurasi perbaikan proses yang terpotong, khususnya akurasi pada perbaikan proses yang terpotong di awal. Kemudian, perbaikan proses yang terpotong dapat dibentuk ke dalam log data dengan mengestimasi waktu pelaksanaannya, sehingga log yang berisi perbaikan proses yang terpotong dapat digunakan secara otomatis untuk analisa proses.

DAFTAR PUSTAKA

Anugrah, I. G., Sarno, R., & Anggraini, R. N. E. (2015). Decomposition using Refined Process Structure Tree (RPST) and control flow complexity metrics. In *2015 International Conference on Information & Communication Technology and Systems (ICTS)* (pp. 203–208). Surabaya. https://doi.org/10.1109/ICTS.2015.7379899

Bose, R. P. J. C., Mans, R. S., & van der Aalst, W. M. P. (2013). Wanna improve process mining results? In *2013 IEEE Symposium on Computational Intelligence and Data Mining (CIDM)* (pp. 127–134). https://doi.org/10.1109/CIDM.2013.6597227

Buijs, J. C. A. M., Van Dongen, B. F., & Van Der Aalst, W. M. P. (2012). On the role of fitness, precision, generalization and simplicity in process discovery. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *7565 LNCS*(PART 1), 305–322. https://doi.org/http://doi.org/10.1007/978-3-642-33606-5_19

Burattin, A., Sperduti, A., & van der Aalst, W. M. P. (2012). Heuristics Miners for Streaming Event Data. *ArXiv CoRR*. https://doi.org/10.1109/CEC.2014.6900341

De Cnudde, S., Claes, J., & Poels, G. (2014). Improving the quality of the Heuristics Miner in ProM 6.2. *Expert Systems with Applications*, *41*(17), 7678–7690. https://doi.org/10.1016/j.eswa.2014.05.055

Di Ciccio, C., Schouten, M. H. M., De Leoni, M., & Mendling, J. (2015). Declarative process discovery with MINERful in ProM. *CEUR Workshop Proceedings*, *1418*, 60–64.

Guo, Q., Wen, L., Wang, J., Yan, Z., & Yu, P. S. (2015). Mining Invisible Tasks in Non-free-choice Constructs. In *Lecture Notes in Computer Science* (pp. 109–125). Springer International Publishing. https://doi.org/10.1007/978-3-319-23063-4_7

Maggi, F. M., Mooij, A. J., & Van Der Aalst, W. M. P. (2011). User-guided discovery of declarative process models. *IEEE SSCI 2011: Symposium Series on Computational Intelligence - CIDM 2011: 2011 IEEE Symposium on Computational Intelligence and Data Mining*, 192–199. https://doi.org/http://doi.org/10.1109/CIDM.2011.5949297

Mulyar, N., Russell, N., ter Hofstede, A., & van der Aalst, W. M. P. (2006). Towards a WPSL : A Critical Analysis of the 20 Classical Workflow Control-flow Patterns. *BPM Reports*, 1–66.

Raim, M. (2014). *Discovering Declarative Process Models from Event Logs through Temporal Logic Query Checking*. University of Tartu.

Russell, N., ter Hofstede, A.H.M, van der Aalst, W.M.P, Mulyar, N. (2006). WORKFLOW CONTROL-FLOW PATTERNS A Revised View. *BPM Center Report*, *2*, 6–22. https://doi.org/http://doi.org/10.1.1.93.6974

Sarno, R., Wibowo, W. A., Kartini, Effendi, Y. A., & Sungkono, K. R. (2016).

Determining Model Using Non-Linear Heuristics Miner and Control-Flow Pattern. *TELKOMNIKA (Telecommunication, Computing, Electronics and Control)*, *14*(1), 349–360. https://doi.org/http://doi.org/10.12928/telkomnika.v14i1.3257

Song, W., Xia, X., Jacobsen, H. A., Zhang, P., & Hu, H. (2015). Heuristic Recovery of Missing Events in Process Logs. *Proceedings - 2015 IEEE International Conference on Web Services, ICWS 2015*, 105–112. https://doi.org/10.1109/ICWS.2015.24

Van Der Aalst, W. M. P. (2011). *Process Mining Discovery, Conformance and Enhancement of Business Processes*. Germany: Springer.

Van der Aalst, W. M. P., De Beer, H. T., & Van Dongen, B. F. (2005). Process mining and verification of properties: An approach based on temporal logic. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *3760 LNCS*, 130–147.

Van der Aalst, W. M. P., Ter Hofstede, A. H. M., Kiepuszewski, B., & Barros, A. P. (2003). Workflow patterns. *Distributed and Parallel Databases*, *14*(1), 5–51. https://doi.org/http://doi.org/10.1023/A:1022883727209

Wang, J., Song, S., Zhu, X., Lin, X., & Sun, J. (2016). Efficient Recovery of Missing Events. *IEEE Transactions on Knowledge and Data Engineering*, *28*(11), 2943–2957. https://doi.org/10.1109/TKDE.2016.2594785

Wen, L., van der Aalst, W. M. P., Wang, J., & Sun, J. (2007). Mining process models with non-free-choice constructs. *Data Mining and Knowledge Discovery*, *15*(2), 145–180. https://doi.org/http://doi.org/10.1007/s10618-007-0065-y

Wen, L., Wang, J., van der Aalst, W. M. P., Huang, B., & Sun, J. (2010). Mining process models with prime invisible tasks. *Data & Knowledge Engineering*, *69*(10), 999–1021. https://doi.org/http://doi.org/10.1016/j.datak.2010.06.001

LAMPIRAN

A. Trace Lengkap dari *Expert* untuk Menguji Proses Anomali dan Proses Tidak Anomali

Nama Aktivitas	**Trace**											
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Document_Entry_via_PDE	1	1	1	1	1	1	1	1	1	1	1	1
Vessel_Berthing_Process	2	2	2	2	2	2	2	2	2	2	2	2
Discharge_Container	3	3	3	3	3	3	3	3	3	3	3	3
Bring_Container_to_Yard	4	4	4	4	4	4	4	4	4	4	4	4
Stack_Container_in_Yard	5	5	5	5	5	5	5	5	5	5	5	5
Verification_Document_ Quarantine	6	6	6	6	6	6						
Create_Job_Order_ Document_Quarantine	7	7	7	7	7	7						
Bring_Container_from_ Yard_to_Quarantine	8	8	8	8	8	8						
Stack_Container_in_ Quarantine_Area	9	9	9	9	9	9						
Check_Goods_Quarantine	10	10	10	10	10	10						
Create_document_KH/KT	11	11	11	11	11	11						
Send_Certificate_KH/KT_Info	12	12	12	12	12	12						
Stack_Container_in_ Yard_From_Quarantine	13	13	13	13	13	13						
Verification_Document_ Behandle	14	14	14	14	14	14	6	6	6	6	6	6
Create_Job_Order_ Document_Behandle	15	15	15				7	7	7			
Stack_Container_in_ Behandle_Area	16	16	16				8	8	8			
Check_Goods_Behandle	17	17	17				9	9	9			
Create_document_LHP	18	18	18				10	10	10			
Bring_Container_from_ Yard_to_Behandle	19	19	19				11	11	11			
Stack_Container_in_ Yard_From_Behandle	20	20	20				12	12	12			

Create_document_SPPB	21	21	21	15	15	15	13	13	13	7	7	7
Create_Job_Order_ Document_Delivery	22	22	22	16	16	16	14	14	14	8	8	8
Truck_in	23	23	23	17	17	17	15	15	15	9	9	9
Dispatch_WQ_Delivery_to_ CHE	24	24	24	18	18	18	16	16	16	10	10	10
Determine_Container_Type	25	25	25	19	19	19	17	17	17	11	11	11
Determining_Dry	26			20			18			12		
Determining_Refeer		26			20			18			12	
Determining_Uncontainer			26			20			18			12
Decide_Task_Before_ Lift_Container	27	27	27	21	21	21	19	19	19	13	13	13
Unplug_Refeer_Cable		28			22			20			14	
Prepare_Tools			28			22			20			14
Lift_on_Container_Truck	28	29	29	22	23	23	20	21	21	14	15	15
Truck_Go_To_Gate_Out	29	30	30	23	24	24	21	22	22	15	16	16
Check_Container_ before_Truck_out	30	31	31	24	25	25	22	23	23	16	17	17
Truck_Out	31	32	32	25	26	26	23	24	24	17	18	18

Keterangan: nomor pada tabel menunjukkan urutan aktivitas pada masing-masing *trace*

B. Data Lengkap Perhitungan Kualitas Model Proses dengan nilai minimum *Support*

Bulan	**Nilai Minimum *Support***	***Fitness* (0,0-1,0)**		
		CasesCaptured	**CasesLog**	**Qf**
Januari	10%	20089	21710	**0,925**
	20%	20032	21710	0,923
	30%	15175	21710	0,699
Februari	10%	19697	23335	**0,844**
	20%	19655	23335	0,842
	30%	14568	23335	0,624
Maret	10%	22447	24557	**0,914**
	20%	22393	24557	0,912
	30%	16359	24557	0,666

Bulan	Nilai Minimum *Support*	Presisi (0,0-1,0)		
		traceCaptured	**tracesModel**	**Qp**
Januari	10%	11	12	**0,92**
	20%	8	16	0,5
	30%	2	4	0,5
Februari	10%	10	12	**0,83**
	20%	8	16	0,5
	30%	2	4	0,5
Maret	10%	10	12	**0,83**
	20%	8	16	0,5
	30%	2	4	0,5

Bulan	Nilai Minimum *Support*	*Simplicity (0,0-1,0)*			
		dupAct	**misAct**	**nodeTree**	**Qs**
Januari	10%	3	3	49	0,878
	20%	0	5	43	**0,884**
	30%	0	17	23	0,261
Februari	10%	3	3	49	0,878
	20%	0	5	43	**0,884**
	30%	0	17	23	0,261
Maret	10%	3	3	49	0,878
	20%	0	5	43	**0,884**
	30%	0	17	23	0,261

Bulan	Nilai Minimum *Support*	Generalisasi (0,0-1,0)		
		sum(exectNode)	**nodeTree**	**Qg**
Januari	10%	1,177	49	0,976
	20%	0,601	43	0,986
	30%	0,227	23	**0,990**
Februari	10%	1,302	49	0,973
	20%	0,633	43	0,985
	30%	0,287	23	**0,988**
Maret	10%	1,235	49	0,975
	20%	0,628	43	0,985
	30%	0,278	23	**0,988**

Keterangan: variabel pada tabel sesuai dengan rumus perhitungan kualitas pada subbab 2.2.9

C. Perhitungan Akurasi Pengelompokkan Proses yang Terpotong dan Anomali

Bulan Januari		**Aktual**	
		Proses Terpotong (1504)	**Anomali (117)**
Prediksi	**Proses Terpotong (1504)**	1504	0
	Anomali (117)	0	117

Bulan Februari		**Aktual**	
		Proses Terpotong (3509)	**Anomali (129)**
Prediksi	**Proses Terpotong (3509)**	3509	0
	Anomali (129)	0	129

Bulan Maret		**Aktual**	
		Proses Terpotong (2006)	**Anomali (104)**
Prediksi	**Proses Terpotong (2006)**	2006	0
	Anomali (104)	0	104

D. Data Lengkap Perhitungan Kualitas Model Proses Metode yang Diusulkan dengan Model Proses dari Algoritma MinerFul

		***Fitness* (0,0-1,0)**		
		casesCaptured	**casesLog**	**Qf**
Januari	Metode Usulan	20089	21710	**0,925**
	MinerFul	20089	21710	**0,925**
Februari	Metode Usulan	19697	23335	**0,844**
	MinerFul	19697	23335	**0,844**
Maret	Metode Usulan	22447	24557	**0,914**
	MinerFul	22447	24557	**0,914**

		Presisi (0,0-1,0)		
		traceCaptured	**tracesModel**	**Qp**
Januari	Metode Usulan	11	12	**0,917**
	MinerFul	11	20	0,550
Februari	Metode Usulan	10	12	**0,833**

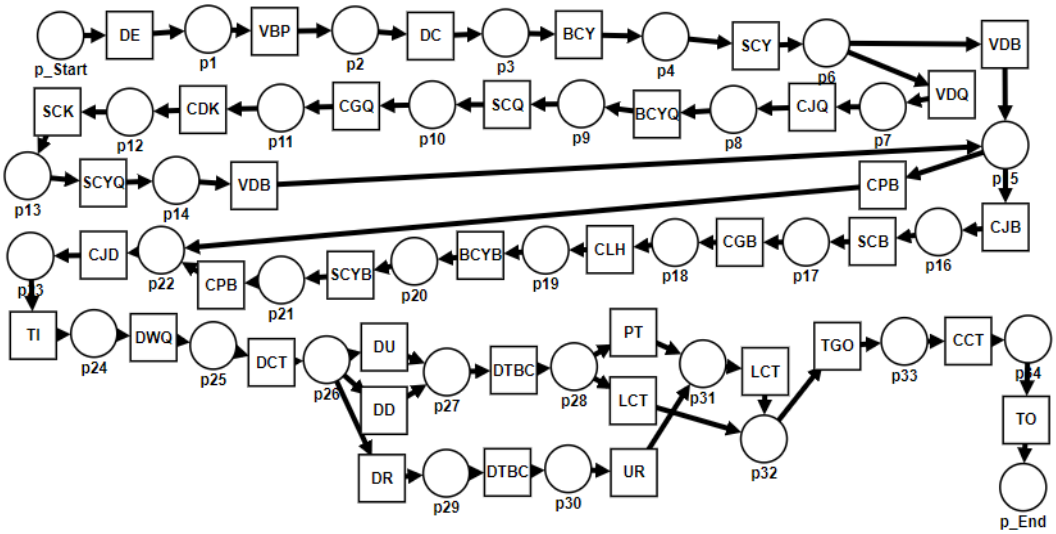
	MinerFul	10	20	0,500
Maret	Metode Usulan	10	12	**0,833**
	MinerFul	10	20	0,500

		Simplicity (0,0-1,0)			
		dupAct	**misAct**	**nodeTree**	**Qs**
Januari	Metode Usulan	3	3	49	**0,878**
	MinerFul	8	0	49	0,837
Februari	Metode Usulan	3	3	49	**0,878**
	MinerFul	8	0	49	0,837
Maret	Metode Usulan	3	3	49	**0,878**
	MinerFul	8	0	49	0,837

		Generalisasi (0,0-1,0)		
		sum(exectNode)	**nodeTree**	**Qg**
Januari	Metode Usulan	1,177	49	0,976
	MinerFul	0,934	49	**0,981**
Februari	Metode Usulan	1,302	49	0,973
	MinerFul	1,014	49	**0,979**
Maret	Metode Usulan	1,235	49	0,975
	MinerFul	0,982	49	**0,980**

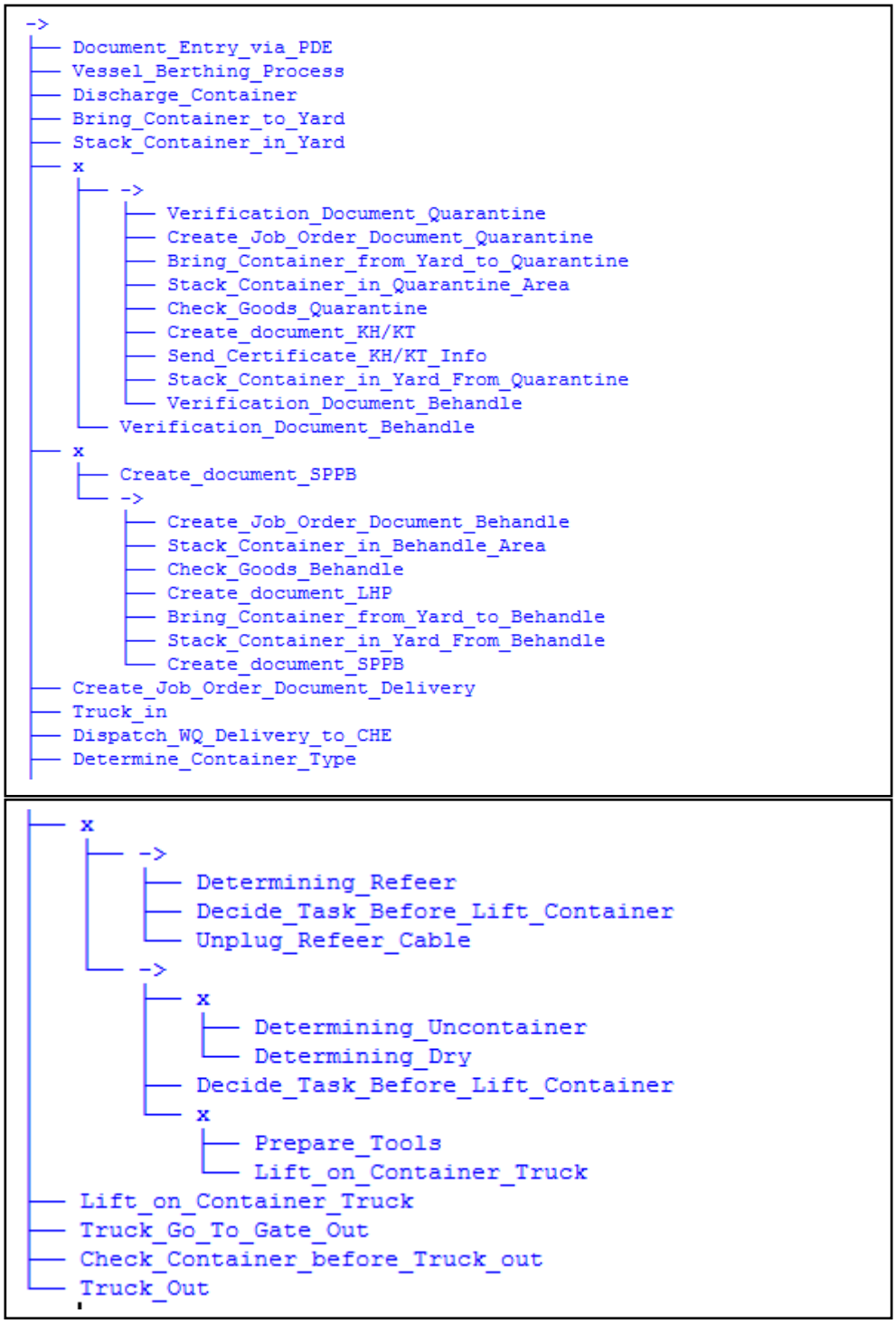
E. Hasil Model Proses dari Algoritma MinerFul

Dalam bentuk Petri Net



Nama Aktivitas	Singkatan	Nama Aktivitas	Singkatan
Document_Entry_via_PDE	DE	Bring_Container_from_ Yard_to_Behandle	BCYB
Vessel_Berthing_Process	VBP	Stack_Container_in_ Yard_ From_Behandle	SCYB
Discharge_Container	DC	Create_document_ SPPB	CPB
Bring_Container_to_Yard	BCY	Create_Job_Order_ Document_Delivery	CJD
Stack_Container_in_Yard	SCY	Truck_in	TI
Verification_Document_ Quarantine	VDQ	Dispatch_WQ_Delivery _to_CHE	DWQ
Create_Job_Order_ Document_Quarantine	CJQ	Determine_Container_ Type	DCT
Bring_Container_from_ Yard_to_Quarantine	BCYQ	Determining_Dry	DD
Stack_Container_in_ Quarantine_Area	SCQ	Decide_Task_Before_ Lift_Container	DTBC
Check_Goods_Quarantine	CGQ	Determining_Refeer	DR
Create_document_KH/KT	CDK	Unplug_Refeer_Cable	UR
Send_Certificate_KH/KT_ Info	SCK	Determining_ Uncontainer	DU
Stack_Container_in_Yard_ From_Quarantine	SCYQ	Prepare_Tools	PT
Verification_Document_ Behandle	VDB	Lift_on_Container_ Truck	LCT
Create_Job_Order_ Document_Behandle	CJB	Truck_Go_To_Gate_ Out	TGO
Stack_Container_in_Behandle_Area	SCB	Check_Container_ before_Truck_out	CCT
Check_Goods_Behandle	CGB	Truck_Out	TO
Create_document_LHP	CLH		

Dikonversi ke dalam bentuk Model Proses Tree



F. *Trace* acuan Algoritma *Heuristic Linear Approach*

Nama Aktivitas	Trace											
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Document_Entry_via_PDE	**1**	1	1	1	**1**	1	1	1	**1**	**1**	1	1
Vessel_Berthing_Process	**2**	2	2	2	**2**	2	2	2	**2**	**2**	2	2
Discharge_Container	**3**	3	3	3	**3**	3	3	3	**3**	**3**	3	3
Bring_Container_to_Yard	**4**	4	4	4	**4**	4	4	4	**4**	**4**	4	4
Stack_Container_in_Yard	**5**	5	5	5	**5**	5	5	5	**5**	**5**	5	5
Verification_Document_ Quarantine	**6**	6	6	6	**6**	6	6	6	**6**	**6**	6	6
Create_Job_Order_ Document_Quarantine	**7**	7	7	7	**7**	7	7	7	**7**	**7**	7	7
Bring_Container_from_ Yard_to_Quarantine	**8**	8	8	8	**8**	8	8	8	**8**	**8**	8	8
Stack_Container_in_ Quarantine_Area	**9**	9	9	9	**9**	9	9	9	**9**	**9**	9	9
Check_Goods_Quarantine	**10**	10	10	10	**10**	10	10	10	**10**	**10**	10	10
Create_document_KH/KT	**11**	11	11	11	**11**	11	11	11	**11**	**11**	11	11
Send_Certificate_KH/KT_In fo	**12**	12	12	12	**12**	12	12	12	**12**	**12**	12	12
Stack_Container_in_Yard_ From_Quarantine	**13**	13	13	13	**13**	13	13	13	**13**	**13**	13	13
Verification_Document_ Behandle	**14**	14	14	14	**14**	14	14	14	**14**	**14**	14	14
Create_Job_Order_ Document_Behandle	**15**	15	15	15	**15**	15	15	15	**15**			
Stack_Container_in_ Behandle_Area	**16**	16	16	16	**16**	16	16	16	**16**			
Check_Goods_Behandle	**17**	17	17	17	**17**	17	17	17	**17**			
Create_document_LHP	**18**	18	18	18	**18**	18	18	18	**18**			
Bring_Container_from_ Yard_to_Behandle	**19**	19	19	19	**19**	19	19	19	**19**			
Stack_Container_in_Yard_ From_Behandle	**20**	20	20	20	**20**	20	20	20	**20**			
Create_document_SPPB	**21**	21	21	21	**21**	21	21	21	**21**	**15**	15	15
Create_Job_Order_ Document_Delivery	**22**	22	22	22	**22**	22	22	22	**22**	**16**	16	16

Truck_in	**23**	23	23	23	**23**	23	23	23	**23**	**17**	17	17
Dispatch_WQ_Delivery_to_CHE	**24**	24	24	24	**24**	24	24	24	**24**	**18**	18	18
Determine_Container_Type	**25**	25	25	25	**25**	25	25	25	**25**	**19**	19	19
Determining_Dry	**26**	26	26							**20**	20	20
Determining_Refeer				26	**26**	26						
Determining_Uncontainer							26	26	**26**			
Decide_Task_Before_Lift_Container	**27**	27	27	27	**27**	27	27	27	**27**	**21**	21	21
Unplug_Refeer_Cable		28			**28**			28			22	
Prepare_Tools			28			28			**28**			22
Lift_on_Container_Truck	**28**	29	29	28	**29**	29	28	29	**29**	**22**	23	23
Truck_Go_To_Gate_Out	**29**	30	30	29	**30**	30	29	30	**30**	**23**	24	24
Check_Container_before_Truck_out	**30**	31	31	30	**31**	31	30	31	**31**	**24**	25	25
Truck_Out	**31**	32	32	31	**32**	32	31	32	**32**	**25**	26	26

Nama Aktivitas	**Trace**											
	13	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**
Document_Entry_via_PDE	1	**1**	1	1	1	**1**	**1**	1	1	1	**1**	1
Vessel_Berthing_Process	2	**2**	2	2	2	**2**	**2**	2	2	2	**2**	2
Discharge_Container	3	**3**	3	3	3	**3**	**3**	3	3	3	**3**	3
Bring_Container_to_Yard	4	**4**	4	4	4	**4**	**4**	4	4	4	**4**	4
Stack_Container_in_Yard	5	**5**	5	5	5	**5**	**5**	5	5	5	**5**	5
Verification_Document_Quarantine	6	**6**	6	6	6	**6**						
Create_Job_Order_Document_Quarantine	7	**7**	7	7	7	**7**						
Bring_Container_from_Yard_to_Quarantine	8	**8**	8	8	8	**8**						
Stack_Container_in_Quarantine_Area	9	**9**	9	9	9	**9**						

Check_Goods_Quarantine	10	**10**	10	10	10	**10**						
Create_document_KH/KT	11	**11**	11	11	11	**11**						
Send_Certificate_KH/KT_Info	12	**12**	12	12	12	**12**						
Stack_Container_in_Yard_ From_Quarantine	13	**13**	13	13	13	**13**						
Verification_Document_ Behandle	14	**14**	14	14	14	**14**	**6**	6	6	6	**6**	6
Create_Job_Order_ Document_Behandle							**7**	7	7	7	**7**	7
Stack_Container_in_ Behandle_Area							**8**	8	8	8	**8**	8
Check_Goods_Behandle							**9**	9	9	9	**9**	9
Create_document_LHP							**10**	10	10	10	**10**	10
Bring_Container_from_ Yard_to_Behandle							**11**	11	11	11	**11**	11
Stack_Container_in_Yard_ From_Behandle							**12**	12	12	12	**12**	12
Create_document_SPPB	15	**15**	15	15	15	**15**	**13**	13	13	13	**13**	13
Create_Job_Order_ Document_Delivery	16	**16**	16	16	16	**16**	**14**	14	14	14	**14**	14
Truck_in	17	**17**	17	17	17	**17**	**15**	15	15	15	**15**	15
Dispatch_WQ_Delivery_to_ CHE	18	**18**	18	18	18	**18**	**16**	16	16	16	**16**	16
Determine_Container_Type	19	**19**	19	19	19	**19**	**17**	17	17	17	**17**	17
Determining_Dry							**18**	18	18			
Determining_Refeer	20	**20**	20							18	**18**	18
Determining_Uncontainer				20	20	**20**						
Decide_Task_Before_Lift_ Container	21	**21**	21	21	21	**21**	**19**	19	19	19	**19**	19
Unplug_Refeer_Cable		**22**			22			20			**20**	
Prepare_Tools			22			**22**			20			20
Lift_on_Container_Truck	22	**23**	23	22	23	**23**	**20**	21	21	20	**21**	21
Truck_Go_To_Gate_Out	23	**24**	24	23	24	**24**	**21**	22	22	21	**22**	22
Check_Container_before_ Truck_out	24	**25**	25	24	25	**25**	**22**	23	23	22	**23**	23

Truck_Out	25	**26**	26	25	26	**26**	**23**	24	24	23	**24**	24

Nama Aktivitas	**Trace**											
	25	**26**	**27**	**28**	**29**	**30**	**31**	**32**	**33**	**34**	**35**	**36**
Document_Entry_via_PDE	1	1	**1**	**1**	1	1	1	**1**	1	1	1	**1**
Vessel_Berthing_Process	2	2	**2**	**2**	2	2	2	**2**	2	2	2	**2**
Discharge_Container	3	3	**3**	**3**	3	3	3	**3**	3	3	3	**3**
Bring_Container_to_Yard	4	4	**4**	**4**	4	4	4	**4**	4	4	4	**4**
Stack_Container_in_Yard	5	5	**5**	**5**	5	5	5	**5**	5	5	5	**5**
Verification_Document_ Quarantine												
Create_Job_Order_ Document_Quarantine												
Bring_Container_from_ Yard_to_Quarantine												
Stack_Container_in_ Quarantine_Area												
Check_Goods_Quarantine												
Create_document_KH/KT												
Send_Certificate_KH/KT_Info												
Stack_Container_in_Yard_ From_Quarantine												
Verification_Document_ Behandle	6	6	**6**	**6**	6	6	6	**6**	6	6	6	**6**
Create_Job_Order_ Document_Behandle	7	7	**7**									
Stack_Container_in_ Behandle_Area	8	8	**8**									
Check_Goods_Behandle	9	9	**9**									
Create_document_LHP	10	10	**10**									
Bring_Container_from_ Yard_to_Behandle	11	11	**11**									
Stack_Container_in_Yard_ From_Behandle	12	12	**12**									
Create_document_SPPB	13	13	**13**	**7**	7	7	7	**7**	7	7	7	**7**

Create_Job_Order_ Document_Delivery	14	14	**14**	**8**	8	8	8	**8**	8	8	8	**8**
Truck_in	15	15	**15**	**9**	9	9	9	**9**	9	9	9	**9**
Dispatch_WQ_Delivery_to_CHE	16	16	**16**	**10**	10	10	10	**10**	10	10	10	**10**
Determine_Container_Type	17	17	**17**	**11**	11	11	11	**11**	11	11	11	**11**
Determining_Dry				**12**	12	12						
Determining_Refeer							12	**12**	12			
Determining_Uncontainer	18	18	**18**							12	12	**12**
Decide_Task_Before_Lift_ Container	19	19	**19**	**13**	13	13	13	**13**	13	13	13	**13**
Unplug_Refeer_Cable		20			14			**14**			14	
Prepare_Tools			**20**			14			14			**14**
Lift_on_Container_Truck	20	21	**21**	**14**	15	15	14	**15**	15	14	15	**15**
Truck_Go_To_Gate_Out	21	22	**22**	**15**	16	16	15	**16**	16	15	16	**16**
Check_Container_before_ Truck_out	22	23	**23**	**16**	17	17	16	**17**	17	16	17	**17**
Truck_Out	23	24	**24**	**17**	18	18	17	**18**	18	17	18	**18**

Keterangan: ***Trace* bercetak tebal** adalah trace yang sesuai dengan *trace* lengkap di LAMPIRAN A dan aktivitas berwarna kuning menunjukkan aktivitas yang salah berdasarkan *trace* lengkap.

INDEKS

(*halaman ini sengaja dikosongkan*)

BIOGRAFI PENULIS

Kelly Rossa Sungkono lahir di Surabaya, Jawa Timur pada tanggal 9 Juni 1994. Pendidikan yang telah ditempuh adalah SDN Lawang V (2000-2006), SMPN 16 Malang (2006-2009), SMAN 7 Tangerang Selatan (2009-2012) dan S1 Teknik Informatika ITS (2012-2016). Rumpun mata kuliah (RMK) yang diambil oleh penulis adalah Manajemen Informasi serta memiliki ketertarikan di bidang *Process Mining*, Audit Sistem, Rekayasa Pengetahuan serta Basis Data. Alat komunikasi yang disediakan oleh penulis adalah surel:kelsungkono@gmail.com.

Kelly Rossa Sungkono telah membuat beberapa jurnal, *paper* seminar, dan buku. List dari publikasi-publikasi tersebut dapat dilihat pada tabel di bawah ini.

Buku	
1	Riyanarto Sarno, Abd. Charis Fauzan, Afina L. Nurlaili, Dewi Rahmawati, Kelly R. Sungkono, Yutika A. Effendi. Manajemen Proses Bisnis, Model dan Simulasi. ITS Tekno Sains. 2017. ISBN 978-602-50375-0-4
Jurnal	
1	R. Sarno and K. R. Sungkono, "Hidden Markov Model for Process Mining of Parallel Business Processes," *International Review on Computers and Software (IRECOS)*, vol. 11, no. 4, pp. 290–300, 2016. http://doi.org/10.15866/irecos.v11i4.8700
2	R. Sarno and K. R. Sungkono, "Coupled Hidden Markov Model for Process Mining of Invisible Prime Tasks," *International Review on Computers and Software (IRECOS)*, vol. 11, no. 6, pp. 539–547, 2016. http://doi.org/10.15866/irecos.v11i6.9555
Seminar	
1	R. Sarno and K. R. Sungkono, "Coupled Hidden Markov Model for Process Discovery of Non-Free Choice and Invisible Prime Tasks," Accepted *Information Systems International Conference (ISICO),* 2017. *Procedia Comput. Science*, vol. 124, pp. 134–141, 2018. http://doi.org/10.1016/j.procs.2017.12.139.
2	K. R. Sungkono, and R. Sarno, "Patterns of Fraud Detection using Coupled

	Hidden Markov Model," Accepted *International Conference on Science in Information Technology (ICSITech),* 2017.
3	K. R. Sungkono, R. Sarno, and N. F. Ariyani, "Refining Business Process Ontology Model with Invisible Prime Tasks using SWRL rules," Accepted *International Conference on Information & Communication Technology and Systems (ICTS)*. 2017
4	Y. Caesarita, R. Sarno, and K. R. Sungkono, "Identification Bottleneck and Fraud of Business Process using Alpha ++ and Heuristic Miner Algorithms (Case study: CV. Wicaksana Artha)," Accepted *International Conference on Information & Communication Technology and Systems (ICTS)*. 2017
3	K. R. Sungkono, and R. Sarno, "CHMM for Discovering Intentional Process Model From Event Logs By Considering Sequence of Activities," Accepted *International Conference on Electrical Engineering, Computer Science, and Informatics (EECSI),* 2017