Event_Log_Analyzer

Release 1.0.0

Bernhard Zosel

Dec 01, 2021

CONTENTS:

1 event_log_analyzer package **3**
- 1.1 event_log_analyzer.importer module . . . 3
- 1.2 event_log_analyzer.adapter module . . . 4
- 1.3 event_log_analyzer.event_log module . . . 6
- 1.4 event_log_analyzer.validate module . . . 8
- 1.5 event_log_analyzer.pattern_library package . . . 8
  - 1.5.1 event_log_analyzer.pattern_library.pattern module . . . 8
  - 1.5.2 event_log_analyzer.pattern_library.manufacturing_scheduling_pattern module . . . 9
  - 1.5.3 event_log_analyzer.pattern_library.pattern_structure module . . . 13
- 1.6 event_log_analyzer.utils module . . . 14

2 Indices and tables **15**

Python Module Index **17**

Index **19**

Event Log Analyzer for importing event logs and classifying them according to constraint patterns.

CHAPTER

ONE

EVENT_LOG_ANALYZER PACKAGE

1.1 event_log_analyzer.importer module

This module contains thr functionality to import new event logs.

event_log_analyzer.importer.**import_csv_file**(*config*)
Import the event log from the file format into a pandas dataframe

Parameters **config_file** – the path to a JSON configuration file

Returns the pandas dataframe without any modification (only the string timestamps are converted to real timestamps)

Return type raw_dataframe

event_log_analyzer.importer.**import_event_log**(*config_file*, *storage_type=StorageType.COLUMN_BASED_AT_ONCE*)
Import and validate an event log into the internal representation of an EventLogStorage object

Parameters

- **config_file** – the path to a JSON configuration file
- **storage_type** (StorageType) – the storage type of the database where we want to import the log, default StorageType.COLUMN_BASED_AT_ONCE

Returns EventLogStorage object

Return type log

event_log_analyzer.importer.**import_xes_file**(*config*)
Import the event log from the file format into a pandas dataframe with timestamps

Parameters **config_file** – the path to a JSON configuration file

Returns the pandas dataframe without any modification

Return type raw_dataframe

1.2 event_log_analyzer.adapter module

This module contains all functionality related to the adaption process to transform new event logs in a unified format

class event_log_analyzer.adapter.**ActivityInstanceAdder**
Bases: *event_log_analyzer.adapter.Adapter*

adds an activity instance, if not already present (only applicable to logs in atomic format with 'Timestamp' attribute -> TimestampRenamer needs to be applied before)

transform(*cfg*, *df*)
transforms the given dataframe and returns it

Parameters

- **cfg** (*json*) – the configuration file
- **df** (*pandas.DataFrame*) – the event log as a data frame that should be transformed

Returns returns the transformed data frame

Return type pandas.DataFrame

class event_log_analyzer.adapter.**Adapter**
Bases: abc.ABC

Adapter interface which every adapter needs to implement

abstract transform(*cfg*, *df*)
transforms the given dataframe and returns it

Parameters

- **cfg** (*json*) – the configuration file
- **df** (*pandas.DataFrame*) – the event log as a data frame that should be transformed

Returns returns the transformed data frame

Return type pandas.DataFrame

class event_log_analyzer.adapter.**ColumnRenamer**
Bases: *event_log_analyzer.adapter.Adapter*

maps all attributes names as specified in the config file (this adapter should be applied after all other adapters that need the original attribute names have been applied)

transform(*cfg*, *df*)
transforms the given dataframe and returns it

Parameters

- **cfg** (*json*) – the configuration file
- **df** (*pandas.DataFrame*) – the event log as a data frame that should be transformed

Returns returns the transformed data frame

Return type pandas.DataFrame

class event_log_analyzer.adapter.**EventToIntervalLog**
Bases: *event_log_analyzer.adapter.Adapter*

if a log is given in atomic format, this adapter converts it to interval event log format (only applicable if log contains 'Job', 'Machine', 'Timestamp' and 'Transaction_Type' attributes)

transform(*cfg*, *df*)
transforms the given dataframe and returns it

Parameters

- **cfg** (*json*) – the configuration file
- **df** (*pandas.DataFrame*) – the event log as a data frame that should be transformed

Returns returns the transformed data frame

Return type pandas.DataFrame

transform_without_pm4py(*cfg*, *df*)
this method shows how the transform() method could be implemented without pm4py

class event_log_analyzer.adapter.**IntervalToEventLogTransformer**
Bases: *event_log_analyzer.adapter.Adapter*

if a log is given in interval format, this adapter converts it to atomic event log format (two timestamps 'Start' and 'Complete' are required -> TimestampRenamer needs to be applied before)

transform(*cfg*, *df*)
transforms the given dataframe and returns it

Parameters

- **cfg** (*json*) – the configuration file
- **df** (*pandas.DataFrame*) – the event log as a data frame that should be transformed

Returns returns the transformed data frame

Return type pandas.DataFrame

class event_log_analyzer.adapter.**RowIDAdder**
Bases: *event_log_analyzer.adapter.Adapter*

maps a given row id to the Row_ID attribute, if nothing is specified a new row column id is added to the log

transform(*cfg*, *df*)
transforms the given dataframe and returns it

Parameters

- **cfg** (*json*) – the configuration file
- **df** (*pandas.DataFrame*) – the event log as a data frame that should be transformed

Returns returns the transformed data frame

Return type pandas.DataFrame

class event_log_analyzer.adapter.**Sorter**
Bases: *event_log_analyzer.adapter.Adapter*

sorts the event log if this is not already done by default

transform(*cfg*, *df*)
transforms the given dataframe and returns it

Parameters

- **cfg** (*json*) – the configuration file
- **df** (*pandas.DataFrame*) – the event log as a data frame that should be transformed

Returns returns the transformed data frame

Return type pandas.DataFrame

class event_log_analyzer.adapter.**TimestampModifier**
Bases: *event_log_analyzer.adapter.Adapter*

transforms all timestamps, indicated in the config file, to a common format (must be done before timestamps are renamed)

get_sec(*time_str*)
converts a timestamp in HH:mm:ss format to seconds

Parameters **time_str** (*str*) – a string in the following format HH:mm:ss, where the hours are continuous, i.e. this timestamp can be considered as duration

Returns duration in seconds

Return type int

transform(*cfg*, *df*)
transforms the given dataframe and returns it

Parameters

- **cfg** (*json*) – the configuration file
- **df** (*pandas.DataFrame*) – the event log as a data frame that should be transformed

Returns returns the transformed data frame

Return type pandas.DataFrame

class event_log_analyzer.adapter.**TimestampRenamer**
Bases: *event_log_analyzer.adapter.Adapter*

maps all timestamp attribute names as specified in the config file (in atomic event log, we map them to 'Timestamp'; in interval log, we map them to 'Start' and 'Complete' timestamps)

transform(*cfg*, *df*)
transforms the given dataframe and returns it

Parameters

- **cfg** (*json*) – the configuration file
- **df** (*pandas.DataFrame*) – the event log as a data frame that should be transformed

Returns returns the transformed data frame

Return type pandas.DataFrame

1.3 event_log_analyzer.event_log module

This module is responsible for storing and managing the event log data.

class event_log_analyzer.event_log.**EventLogStorage**(*cfg*, *storage_type=StorageType.ROW_BASED*)
Bases: object

The EventLogStorage object stores all information about a given event log. To itialize the object, the dataframe that stores the event log already needs to be in a correct format, adaption needs to be done before!

_con
the database connection

Type Connection

_dataframe
the actual event log data

_config
the configuration file

Type json

_storage_type
specifies the data layout used in the event log storage

Type *StorageType*

add_new_dataframe(*new_df*)
validates the log and then adds it into the EventLogStorage database and the database is optimized (e.g. by creating indexes)

Parameters **new_log** (*pandas.DataFrame*) – the event log needs to be in the goal format, i.e. in atomic event log form with one timestamp "Timestamp" in a pandas timestamp format

create_interval_log()
if possible construct an interval log out of the stored atomic event log and save it into the database as a separate table

get_event_log()
loads the event log from the database

Returns **dataframe** – the whole event log from the data base as a data frame

Return type pandas.DataFrame

get_interval_log()
loads the interval log from the database

Returns **dataframe** – the whole interval log from the data base as a data frame

Return type pandas.DataFrame

get_interval_sequence(*attr*, *needed_columns=[]*)
groups the interval log database and returns a generator object of the sequence of the given attribute as a generator object

Parameters

- **attr** (*str*) – the attribute by which the event log should be grouped
- **needed_columns** (*List[str], optional*) – a list of attributes/columns that need to be accessed, by default [] (the empty list stands for all attributes)

Yields *pandas.DataFrame* – The next interval sequence as a dataframe.

get_sequence(*attr*, *needed_columns=[]*)
groups the event log database and returns a generator object of the sequence of the given attribute as a generator object

Parameters

- **attr** (*str*) – the attribute by which the event log should be grouped
- **needed_columns** (*List[str], optional*) – a list of attributes/columns that need to be accessed, by default [] (the empty list stands for all attributes)

Yields *pandas.DataFrame* – The next sequence as a dataframe.

print_event_log()
prints the event log on the console

save_adapted_event_log(*file_name='adapted_event_log.csv'*)
saves the adapted event log that is internally stored in the database in the given file

Parameters **file_name** (*str, optional*) – file name where the log should be saved, by default "adapted_event_log.csv"

class event_log_analyzer.event_log.**StorageType**(*value*)
Bases: enum.Enum

The storage types specify the data layout that is used internally to store the event logs.

COLUMN_BASED = 1

COLUMN_BASED_AT_ONCE = 3

ROW_BASED = 2

1.4 event_log_analyzer.validate module

This module is responsible for validating event logs before storing them in the database as well validating the config files.

event_log_analyzer.validate.**validate**(*event_log*)
checks whether the log is sorted by the timestamp column

Parameters

- **event_log** (*DataFrame*) – the dataframe of the event log in atomic event log format ('Timestamp' column required)
- **Raises** –

ValueError if the timestamps are incorrect, i.e. no timestamp column is present or if the log is not sorted by this timestamp

event_log_analyzer.validate.**validate_config**(*json_schema_path*, *config_file*)
validates the given config_file, whether it is valid according to the given schema

Parameters

- **json_schema** (*str*) – the path to the JSON schema, where the config file format has been specified
- **config_file** (*dict*) – the JSON object that was parsed

Raises **ValueError if the config file is invalid** –

1.5 event_log_analyzer.pattern_library package

1.5.1 event_log_analyzer.pattern_library.pattern module

This module implements contains the abstract pattern class that must be extended by any pattern.

class event_log_analyzer.pattern_library.pattern.**Pattern**
Bases: abc.ABC

The abstract Pattern class describes single constraint patterns with all its dependencies.

name
a unique name, with which each pattern can be identified

Type str

dependencies
dependencies to other patterns (identified by their name) given as a key value pair where the key stands for the type of dependency (for example necessary conditions)

Type Dict[str, List[str]]

applies
True if the pattern applies, False if not, None if it has not been checked yet

Type bool

add_dependencies(*pattern_structure*)
adds all dependencies to other patterns in the pattern structure

Parameters **pattern_structure** (PatternStructure) – the Pattern Structure object in which the dependencies should be linked

check_dependencies()
checks whether the pattern already applies without further checking based on dependencies with other classes and logs it

Returns True if the pattern already applies without further checking, False if it cannot apply anymore, or otherwise None if no statement can be made

Return type bool

abstract property dependencies

abstract property name

abstract pattern_applies(*event_log*) → bool
checks whether the pattern applies

Parameters **event_log** (EventLogStorage) – the EventLogStorage object of the log on which the conditions of the pattern should be checked

Returns True if the pattern applies, False if it does not apply

Return type bool

print_name()
prints the unique name

1.5.2 event_log_analyzer.pattern_library.manufacturing_scheduling_pattern module

This module contains all patterns, that have the Manufacturing Scheduling Pattern as a prerequisite

class event_log_analyzer.pattern_library.manufacturing_scheduling_patterns.**DistinguishableResource**
Bases: *event_log_analyzer.pattern_library.pattern.Pattern*

In the Distinguishable Resource Pattern, when processing a number of operations, exactly one resource from a set of distinguishable resources needs to be present for each of the operations.

Currently the pattern only checks one resource (in the 'Resource' column), when having several resources, they need to be manufally checked one by one

dependencies = {'forces': [], 'requires': ['Manufacturing_Scheduling_Pattern']}

name = 'Distinguishable_Resource_Pattern'

pattern_applies(*event_log*)
one resource from a set of distinguishable resources must be present at every operation, but only one operation can be processed per resource (checking on atomic event log)

Parameters **event_log** (EventLogStorage) – the EventLogStorage object of the log on which the conditions of the pattern should be checked

Returns True if the pattern applies, False if it does not apply

Return type bool

class event_log_analyzer.pattern_library.manufacturing_scheduling_patterns.**FlowShop**
Bases: *event_log_analyzer.pattern_library.pattern.Pattern*

In the Flow Shop Pattern, each job needs to be processed on each of the machines in a predetermined order and the order is the same for all jobs.

cond_a(*event_log*)
every job can be processed on every machine at most once

cond_b(*event_log*)
every job consists of exactly as much operations as machines exist in total

cond_c(*event_log*)
the route of every job through the machines must be the same

dependencies = {'forces': [], 'requires': ['Job_Shop_Pattern']}

name = 'Flow_Shop_Pattern'

pattern_applies(*event_log*)
the pattern applies if all three conditions (a), (b) and (c) apply

Parameters **event_log** (EventLogStorage) – the EventLogStorage object of the log on which the conditions of the pattern should be checked

Returns True if the pattern applies, False if it does not apply

Return type bool

class event_log_analyzer.pattern_library.manufacturing_scheduling_patterns.**IndistinguishableResource**
Bases: *event_log_analyzer.pattern_library.pattern.Pattern*

In the Indistinguishable Resource Pattern, when processing a number of operations, exactly one resource from a set of indistinguishable resources needs to be present for each operation.

Currently the pattern only checks one resource (in the 'Resource' column), when having several resources, they need to be manufally checked one by one

dependencies = {'forces': [], 'requires': ['Manufacturing_Scheduling_Pattern']}

name = 'Indistinguishable_Resource_Pattern'

pattern_applies(*event_log*)
for each operation there must be exactly one resource from a set of indistinguishable resources

Parameters **event_log** (EventLogStorage) – the EventLogStorage object of the log on which the conditions of the pattern should be checked

Returns True if the pattern applies, False if it does not apply

Return type bool

class event_log_analyzer.pattern_library.manufacturing_scheduling_patterns.**JobShop**
Bases: *event_log_analyzer.pattern_library.pattern.Pattern*

In the Job Shop Pattern, each job needs to be processed on a subset of the machines in a predetermined order which might be different between the jobs.

cond_a(*event_log*)
jobs cannot be processed without a machine

vectorization is used on the whole event log to check whether there exists an event without a machine

cond_b(*event_log*)
no two operations of the same job can be processed at the same time

the condition is checked on the atomic event log

cond_b_interval(*event_log*)
no two operations of the same job can be processed at the same time

the condition is checked on the interval log

cond_c(*event_log*)
No machine can process more than one operation at the same time

the condition is checked on the atomic event log

cond_c_interval(*event_log*)
No machine can process more than one operation at the same time

the condition is checked on the interval log

dependencies = {'forces': [], 'requires': ['Manufacturing_Scheduling_Pattern']}

name = 'Job_Shop_Pattern'

pattern_applies(*event_log*)
the pattern applies if all three conditions (a), (b) and (c) apply

Parameters **event_log** (EventLogStorage) – the EventLogStorage object of the log on which the conditions of the pattern should be checked

Returns True if the pattern applies, False if it does not apply

Return type bool

class event_log_analyzer.pattern_library.manufacturing_scheduling_patterns.**ManufacturingScheduling**
Bases: *event_log_analyzer.pattern_library.pattern.Pattern*

This Pattern sets the basis for all further Scheduling patterns, it has to be checked before the other patterns can be checked to ensure that the pattern contains all needed information.

dependencies = {'forces': [], 'requires': []}

name = 'Manufacturing_Scheduling_Pattern'

pattern_applies(*event_log*)
the pattern applies if all needed attributes are available (Job, Machine, Transaction_Type) and if the transaction_types represent intervals, if the pattern applies we create an interval log which is better suited for checking the further pattern conditions

Parameters **event_log** (EventLogStorage) – the EventLogStorage object of the log on which the conditions of the pattern should be checked

Returns True if the pattern applies, False if it does not apply

Return type bool

class event_log_analyzer.pattern_library.manufacturing_scheduling_patterns.**NoWait**
Bases: *event_log_analyzer.pattern_library.pattern.Pattern*

In the No Wait Pattern, when a number of jobs are to be processed on a set of machines, jobs are not allowed to wait between operations.

dependencies = {'forces': [], 'requires': ['Flow_Shop_Pattern']}

name = 'No_Wait_Pattern'

pattern_applies(*event_log*)
jobs are not allowed to wait between operations

Parameters **event_log** (EventLogStorage) – the EventLogStorage object of the log on which the conditions of the pattern should be checked

Returns True if the pattern applies, False if it does not apply

Return type bool

class event_log_analyzer.pattern_library.manufacturing_scheduling_patterns.**OneBlocking**
Bases: *event_log_analyzer.pattern_library.pattern.Pattern*

In the 1-Blocking Pattern, when processing a number of jobs on machines under the Flow Shop pattern, there must at any time be at most one job queuing in front of a machine, so the buffer between any two machines has capacity 1.

dependencies = {'forces': [], 'requires': ['Permutation_Pattern']}

name = '1-blocking_Pattern'

pattern_applies(*event_log*)
at any time there cannot be more than one job queing in front of a machine

Note: The pattern is not implemented yet, therefore the pattern never applies!!!

Parameters **event_log** (EventLogStorage) – the EventLogStorage object of the log on which the conditions of the pattern should be checked

Returns True if the pattern applies, False if it does not apply

Return type bool

class event_log_analyzer.pattern_library.manufacturing_scheduling_patterns.**Permutation**
Bases: *event_log_analyzer.pattern_library.pattern.Pattern*

In the Permutation Pattern, when processing a number of jobs on machines under the Flow Shop Pattern no job is allowed to overtake another job, so the order in which jobs are processed on a machine is the same for all machines.

dependencies = {'forces': [], 'requires': ['Flow_Shop_Pattern']}

name = 'Permutation_Pattern'

pattern_applies(*event_log*)
check whether the Permutation Pattern applies, i.e. jobs cannot overtake each other, all machines process the jobs in the same order

Parameters **event_log** (EventLogStorage) – the EventLogStorage object of the log on which the conditions of the pattern should be checked

Returns True if the pattern applies, False if it does not apply

Return type bool

class event_log_analyzer.pattern_library.manufacturing_scheduling_patterns.**ResourceSetupTimes**

Bases: *event_log_analyzer.pattern_library.pattern.Pattern*

In the Resource Setup Times Pattern, a number of operations are to be processed by some distinguishable resources (Distinguishable Resources Pattern needs to apply) and each resource needs a certain amount of time between two operations, where the time depends on the type of the two tasks.

Note: The pattern is not implemented yet, therefore the pattern never applies!!!

dependencies = {'forces': [], 'requires': ['Distinguishable_Resource_Pattern']}

name = 'Resource_Setup_Times_Pattern'

pattern_applies(*event_log*)

each resource needs a certain amount of time between two operations

Parameters **event_log** (EventLogStorage) – the EventLogStorage object of the log on which the conditions of the pattern should be checked

Returns True if the pattern applies, False if it does not apply

Return type bool

1.5.3 event_log_analyzer.pattern_library.pattern_structure module

This module contains the functionality that organizes the pattern checking process by structuring them in a graph

class event_log_analyzer.pattern_library.pattern_structure.**PatternStructure**

Bases: object

A PatternStructure object contains all patterns in a structured way (an aycyclic directed graph), so that the event log can be checked efficiently pattern by pattern.

dependency_graph

a networkx directed graph object in which the pattern objects are stored

Type networkx.DiGraph

topological_order

a list of all patterns in a topological order

Type List[*Pattern*]

applying_pattern_list()

returns a list of all patterns that apply

Returns **pattern_list** – list of all patterns that apply

Return type List[*Pattern*]

Raises **ValueError** – If the patterns have not been checked yet.

check_all_patterns(*event_log*)

check all patterns that are initialized in the pattern structure in a topological order and log whether they apply

Parameters **log** (EventLogStorage) – the event log on which the patterns should be classified

plot_pattern_structure(*file='pattern_structure.pdf'*)

plots the pattern structure in the given file as a matplotlib figure

Parameters **file** (*str, optional*) – The file name of the output where the graphic should be plotted (default is pattern_structure.pdf)

print_topological_ordering()
prints the topological order in a list object

topological_ordering()
finds the topological ordering of the initialized pattern structure and stores it in the topological_order attribute

1.6 event_log_analyzer.utils module

This module contains various functions that are needed in our tool, e.g. initialization of logging.

event_log_analyzer.utils.**log_time**(*logger*, *text*)
this function can be used as a decorator, which writes the time needed for the decorated function together with a text to the specified logger

Parameters

- **logger** (*Logger*) – the logger in which we want to write the information
- **text** (*str*) – a text printed together with the time in seconds that the decorated method took to execute

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

e

event_log_analyzer.adapter, 4
event_log_analyzer.event_log, 6
event_log_analyzer.importer, 3
event_log_analyzer.pattern_library.manufacturing_scheduling_patterns, 9
event_log_analyzer.pattern_library.pattern, 8
event_log_analyzer.pattern_library.pattern_structure, 13
event_log_analyzer.utils, 14
event_log_analyzer.validate, 8

INDEX

Symbols

_con (*event_log_analyzer.event_log.EventLogStorage attribute*), 6
_config (*event_log_analyzer.event_log.EventLogStorage attribute*), 7
_dataframe (*event_log_analyzer.event_log.EventLogStorage attribute*), 6
_storage_type (*event_log_analyzer.event_log.EventLogStorage attribute*), 7

A

ActivityInstanceAdder (*class in event_log_analyzer.adapter*), 4
Adapter (*class in event_log_analyzer.adapter*), 4
add_dependencies() (*event_log_analyzer.pattern_library.pattern.Pattern method*), 9
add_new_dataframe() (*event_log_analyzer.event_log.EventLogStorage method*), 7
applies (*event_log_analyzer.pattern_library.pattern.Pattern attribute*), 9
applying_pattern_list() (*event_log_analyzer.pattern_library.pattern_structure.PatternStructure method*), 13

C

check_all_patterns() (*event_log_analyzer.pattern_library.pattern_structure.PatternStructure method*), 13
check_dependencies() (*event_log_analyzer.pattern_library.pattern.Pattern method*), 9
COLUMN_BASED (*event_log_analyzer.event_log.StorageType attribute*), 8
COLUMN_BASED_AT_ONCE (*event_log_analyzer.event_log.StorageType attribute*), 8
ColumnRenamer (*class in event_log_analyzer.adapter*), 4
cond_a() (*event_log_analyzer.pattern_library.manufacturing_scheduling_patterns.FlowShop method*), 10
cond_a() (*event_log_analyzer.pattern_library.manufacturing_scheduling_patterns.JobShop method*), 11
cond_b() (*event_log_analyzer.pattern_library.manufacturing_scheduling_ method*), 10
cond_b() (*event_log_analyzer.pattern_library.manufacturing_scheduling_ method*), 11
cond_b_interval() (*event_log_analyzer.pattern_library.manufacturing_ method*), 11
cond_c() (*event_log_analyzer.pattern_library.manufacturing_scheduling_ method*), 10
cond_c() (*event_log_analyzer.pattern_library.manufacturing_scheduling_ method*), 11
cond_c_interval() (*event_log_analyzer.pattern_library.manufacturing_ method*), 11
create_interval_log() (*event_log_analyzer.event_log.EventLogStorage method*), 7

D

dependencies (*event_log_analyzer.pattern_library.manufacturing_schedu attribute*), 9
dependencies (*event_log_analyzer.pattern_library.manufacturing_schedu attribute*), 10
dependencies (*event_log_analyzer.pattern_library.manufacturing_schedu attribute*), 10
dependencies (*event_log_analyzer.pattern_library.manufacturing_schedu attribute*), 11
dependencies (*event_log_analyzer.pattern_library.manufacturing_schedu attribute*), 11
dependencies (*event_log_analyzer.pattern_library.manufacturing_schedu attribute*), 12
dependencies (*event_log_analyzer.pattern_library.manufacturing_schedu attribute*), 12
dependencies (*event_log_analyzer.pattern_library.manufacturing_schedu attribute*), 12
dependencies (*event_log_analyzer.pattern_library.manufacturing_schedu attribute*), 13
dependencies (*event_log_analyzer.pattern_library.pattern.Pattern attribute*), 9
dependencies (*event_log_analyzer.pattern_library.pattern.Pattern property*), 9
dependency_graph (*event_log_analyzer.pattern_library.pattern_structure attribute*), 13
DistinguishableResource (*class in*

event_log_analyzer.pattern_library.manufacturing_scheduling_patterns), 9

E

event_log_analyzer.adapter
    module, 4
event_log_analyzer.event_log
    module, 6
event_log_analyzer.importer
    module, 3
event_log_analyzer.pattern_library.manufacturing_scheduling_patterns
    module, 9
event_log_analyzer.pattern_library.pattern
    module, 8
event_log_analyzer.pattern_library.pattern_structure
    module, 13
event_log_analyzer.utils
    module, 14
event_log_analyzer.validate
    module, 8
EventLogStorage (*class in event_log_analyzer.event_log*), 6
EventToIntervalLog (*class in event_log_analyzer.adapter*), 4

F

FlowShop (*class in event_log_analyzer.pattern_library.manufacturing_scheduling_patterns*), 10

G

get_event_log() (*event_log_analyzer.event_log.EventLogStorage method*), 7
get_interval_log() (*event_log_analyzer.event_log.EventLogStorage method*), 7
get_interval_sequence() (*event_log_analyzer.event_log.EventLogStorage method*), 7
get_sec() (*event_log_analyzer.adapter.TimestampModifier method*), 6
get_sequence() (*event_log_analyzer.event_log.EventLogStorage method*), 7

I

import_csv_file() (*in module event_log_analyzer.importer*), 3
import_event_log() (*in module event_log_analyzer.importer*), 3
import_xes_file() (*in module event_log_analyzer.importer*), 3
IndistinguishableResource (*class in event_log_analyzer.pattern_library.manufacturing_scheduling_patterns*), 10
IntervalToEventLogTransformer (*class in event_log_analyzer.adapter*), 5

J

JobShop (*class in event_log_analyzer.pattern_library.manufacturing_scheduling_patterns*), 10

L

log_time() (*in module event_log_analyzer.utils*), 14

M

ManufacturingScheduling (*class in event_log_analyzer.pattern_library.manufacturing_scheduling_patterns*), 11
module
    event_log_analyzer.adapter, 4
    event_log_analyzer.event_log, 6
    event_log_analyzer.importer, 3
    event_log_analyzer.pattern_library.manufacturing_scheduling_patterns, 9
    event_log_analyzer.pattern_library.pattern, 8
    event_log_analyzer.pattern_library.pattern_structure, 13
    event_log_analyzer.utils, 14
    event_log_analyzer.validate, 8

N

name (*event_log_analyzer.pattern_library.manufacturing_scheduling_patterns.IndistinguishableResource attribute*), 10
name (*event_log_analyzer.pattern_library.manufacturing_scheduling_patterns.JobShop attribute*), 10
name (*event_log_analyzer.pattern_library.manufacturing_scheduling_patterns.ManufacturingScheduling attribute*), 11
name (*event_log_analyzer.pattern_library.manufacturing_scheduling_patterns.NoWait attribute*), 11
name (*event_log_analyzer.pattern_library.manufacturing_scheduling_patterns.OneBlocking attribute*), 12
name (*event_log_analyzer.pattern_library.manufacturing_scheduling_patterns.OpenShop attribute*), 12
name (*event_log_analyzer.pattern_library.manufacturing_scheduling_patterns.RecirculationShop attribute*), 12
name (*event_log_analyzer.pattern_library.manufacturing_scheduling_patterns.SingleMachine attribute*), 13
name (*event_log_analyzer.pattern_library.pattern.Pattern attribute*), 8
name (*event_log_analyzer.pattern_library.pattern.Pattern property*), 9
NoWait (*class in event_log_analyzer.pattern_library.manufacturing_scheduling_patterns*), 12

O

OneBlocking (*class in event_log_analyzer.pattern_library.manufacturing_scheduling_patterns*), 12

P

Pattern (*class in event_log_analyzer.pattern_library.pattern*), 8
pattern_applies() (*event_log_analyzer.pattern_library.manufacturing_scheduling_patterns.DistinguishableResource method*), 9
pattern_applies() (*event_log_analyzer.pattern_library.manufacturing_scheduling_patterns.FlowShop method*), 10
pattern_applies() (*event_log_analyzer.pattern_library.manufacturing_scheduling_patterns.IndistinguishableResource method*), 10
pattern_applies() (*event_log_analyzer.pattern_library.manufacturing_scheduling_patterns.JobShop method*), 11
pattern_applies() (*event_log_analyzer.pattern_library.manufacturing_scheduling_patterns.ManufacturingScheduling method*), 11
pattern_applies() (*event_log_analyzer.pattern_library.manufacturing_scheduling_patterns.NoWait method*), 12
pattern_applies() (*event_log_analyzer.pattern_library.manufacturing_scheduling_patterns.OneBlocking method*), 12
pattern_applies() (*event_log_analyzer.pattern_library.manufacturing_scheduling_patterns.Permutation method*), 12
pattern_applies() (*event_log_analyzer.pattern_library.manufacturing_scheduling_patterns.ResourceSetupTimes method*), 13
pattern_applies() (*event_log_analyzer.pattern_library.pattern.Pattern method*), 9
PatternStructure (*class in event_log_analyzer.pattern_library.pattern_structure*), 13
Permutation (*class in event_log_analyzer.pattern_library.manufacturing_scheduling_patterns*), 12
plot_pattern_structure() (*event_log_analyzer.pattern_library.pattern_structure.PatternStructure method*), 13
print_event_log() (*event_log_analyzer.event_log.EventLogStorage method*), 7
print_name() (*event_log_analyzer.pattern_library.pattern.Pattern method*), 9
print_topological_ordering() (*event_log_analyzer.pattern_library.pattern_structure.PatternStructure method*), 13

R

ResourceSetupTimes (*class in event_log_analyzer.pattern_library.manufacturing_scheduling_patterns*), 12
ROW_BASED (*event_log_analyzer.event_log.StorageType attribute*), 8
RowIDAdder (*class in event_log_analyzer.adapter*), 5

S

save_adapted_event_log() (*event_log_analyzer.event_log.EventLogStorage method*), 7
Sorter (*class in event_log_analyzer.adapter*), 5
StorageType (*class in event_log_analyzer.event_log*), 8

T

TimestampModifier (*class in event_log_analyzer.adapter*), 6
TimestampRenamer (*class in event_log_analyzer.adapter*), 6
topological_ordering (*event_log_analyzer.pattern_library.pattern_structure.PatternStructure attribute*), 13
topological_ordering() (*event_log_analyzer.pattern_library.pattern_structure.PatternStructure method*), 14
transform() (*event_log_analyzer.adapter.ActivityInstanceAdder method*), 4
transform() (*event_log_analyzer.adapter.Adapter method*), 4
transform() (*event_log_analyzer.adapter.ColumnRenamer method*), 4
transform() (*event_log_analyzer.adapter.EventToIntervalLog method*), 4
transform() (*event_log_analyzer.adapter.IntervalToEventLogTransformer method*), 5
transform() (*event_log_analyzer.adapter.RowIDAdder method*), 5
transform() (*event_log_analyzer.adapter.Sorter method*), 5
transform() (*event_log_analyzer.adapter.TimestampModifier method*), 6
transform() (*event_log_analyzer.adapter.TimestampRenamer method*), 6
transform_without_pm4py() (*event_log_analyzer.adapter.EventToIntervalLog method*), 5

V

validate() (*in module event_log_analyzer.validate*), 8
validate_config() (*in module event_log_analyzer.validate*), 8