kLFA User Manual

Contents

1 Introduction **3**

2 Installing and Compiling KLFA **5**

2.1 Installing a compiled version of KLFA 5

2.2 Compiling KLFA from a source distribution 6

2.3 Compiling KLFA from CVS 7

2.4 Installing SLCT 7

3 Tools **9**

3.1 Monitoring 9

3.2 Model Generation 9

3.3 Failure Analysis 11

4 Examples **13**

4.1 Glassfish deployment failure 13

4.1.1 Monitoring 13

4.1.2 Model Generation 13

4.1.3 Failure analysis 20

Bibliography **26**

Chapter 1

Introduction

Log files are commonly inspected by system administrators and developers to detect suspicious behaviors and diagnose failure causes. Since size of log files grows fast, thus making manual analysis impractical, different automatic techniques have been proposed to analyze log files. Unfortunately, accuracy and effectiveness of these techniques are often limited by the unstructured nature of logged messages and the variety of data that can be logged.

KLFA is a tool that automatically analyzes log files and retrieves important information to identify failure causes. KLFA automatically identifies dependencies between events and values in logs corresponding to legal executions, generates models of legal behaviors and compares log files collected during failing executions with the generated models to detect anomalous event sequences that are presented to users.

Experimental results show the effectiveness of the technique in supporting developers and testers to identify failure causes.

kLFA has been described in [CPMP07] and [MP08].

Figure 1.1 shows the three steps of the technique, while Figure 1.2 focus on the model generation. Detailed information about the technique can be found in [MP08].

Following chapters describe for every step of the technique the tools involved and give examples of the usage of the tools.

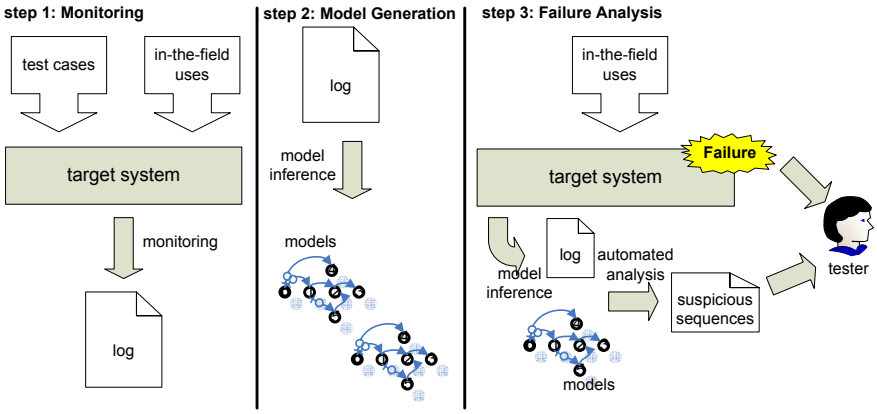

Figure 1.1: Automated log analysis.

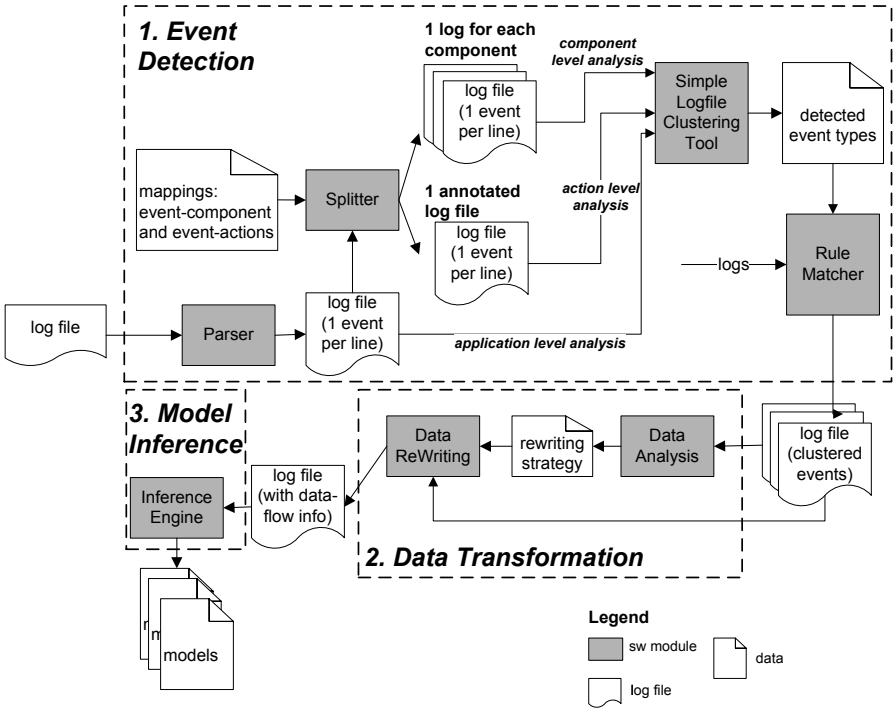

Figure 1.2: Model generation.

Chapter 2

Installing and Compiling KLFA

2.1 Installing a compiled version of KLFA

If you received the KLFA distribution zip (something like klfa-201010141601.zip), just uncompress it in the location you prefer, e.g. `/home/fabrizio/Programs/klfa201010141601`.

Once you uncompressed it you just need to do the following commands:

1) (if using Linux or OSX) make scripts executables e.g.

```
chmod a+x /home/fabrizio/Programs/klfa-201010141601/bin/*
```

2) (for any OS) set the environment variable KLFA_HOME to point to the folder where you installed klfa, e.g. `/home/fabrizio/Programs/klfa-201010141601/`

If you are using Linux or OSX with the BASH shell you could add the following line to file `$HOME.bashrc`:

```
export KLFA_HOME=/home/fabrizio/Programs/klfa-201010141601/
```

Change the path according to your KLFA installation path.

3) (for any OS) add the bin folder in KLFA_HOME to the PATH environment variable.

If you are using Linux or OSX with the BASH shell you could add the following line to file .bashrc (change the path according to your path):

```
export PATH=$PATH:/home/fabrizio/Programs/klfa-201010141601/bin/
```

You can check if the previous command succeeded by running the following command and checking that you have an output similar to the one reported below:

```
$ which klfaCsvAnalysis.sh
/home/fabrizio/Programs/klfa-201010141601/bin//klfaCsvAnalysis.sh
```

Check if klfa is correctly installed by running:

```
$ klfaCsvAnalysis.sh
```

The command will output KLFA command help. Like in the following paragraph:

```
This program builds models of the application behavior by analyzing a trace
file. The trace file must be a collection of lines, each one in the format
COMPONENT,EVENT[,PARAMETER].
Multiple traces can be defined in a file, to
separate a trace from another put a line with the | symbol.
Usage :
it.unimib.disco.lta.alfa.klfa.LogTraceAnalyzer [options] <analysisType> <phase>
<valueTranformersConfigFile> <preprocessingRules> <traceFile>
```

KLFA includes several programs and utilities described in the following Sections. The most common utilities can be run by using the shell scripts in `KLFA_HOME/bin`

We suggest to go through the examples in folder `KLFA_HOME/examples` to understand how to use KLFA. Some examples are described in Chapter 4, others are described in the file `README.txt` that you find in each example folder.

2.2 Compiling KLFA from a source distribution

If you received a source distribution zip of KLFA (something like klfa-src-201010141601.zip), uncompress it in the location you prefer, e.g. /home/fabrizio/Programs/klfa-src201010141601.

In order to compile an installable version of klfa from sources run the following command within the folder where you uncompressed klfa:

```
ant distribution
```

so you could do:

```
cd  /home/fabrizio/Programs/klfa-src-201010141601
ant distribution
```

The command will create the KLFA distribution zip in the dist folder. e.g. `/home/fabrizio/Programs-/klfa-src201010141601/dist/klfa201010141601.zip`

After creating the distribution zip you can follow the commands described in Section **??**.

2.3 Compiling KLFA from CVS

In order to install the head version of klfa stored on the UniMiB CVS repository you need to download the following CVS modules:

- LogFileAnalysis-LFA
- BCT (you need to download the TPTPIntegration branch)

LogFileAnalysis-LFA is klfa. BCT provides the libraries to infer automata.

The first step is the compilation of klfa dependencies. To do so run

```
ant buildDependencies
```

The command will create the library *bct.jar* in folder *lib*.

Next step is to run the command

```
ant distribution
```

This command builds the klfa distribution zip. Follow the instructions described in Section 2.1 to install KLFA.

Other klfa ant compilation options are described by the build.xml help. To see the other compilation options just run

```
ant
```

2.4 Installing SLCT

In order to identify event types AVA uses SLCT [Vaa03]. In order to install SLCT you need to change your current directory to `src-native/slct-0.5` and compile slct.

If you use Linux or OsX you can run the following commands

```
cd $AVA_HOME/../src-native/slct-0.5
gcc -o slct -O2 -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64 slct.c
sudo mkdir /opt/slct-0.5
sudo mv slct /opt/slct-0.5
```

Chapter 3

Tools

3.1 Monitoring

In the monitoring phase the user is supposed to collect log files relative to correct system executions. These log files can be collected at testing time during functional system tests or during correct runs of the system. We do not provide any logging tool because the system can work with any logging systems.

3.2 Model Generation

In this phase the log files collected are analyzed by the system to derive a model that generalizes the application behavior. In this phase the initial logs files are preprocessed with different tools in order to:

- contain a complete event in a single line;
- automatically detect event types and associated parameters;
- detect rewriting strategies for parameters;
- infer a model of the log file structure;

Figure 3.1 shows the components involved in this phase. All the components must be called from command line and the user has to set parameters according to the analysis type and the log file analyzed. Following sections describe the functionality of each component.

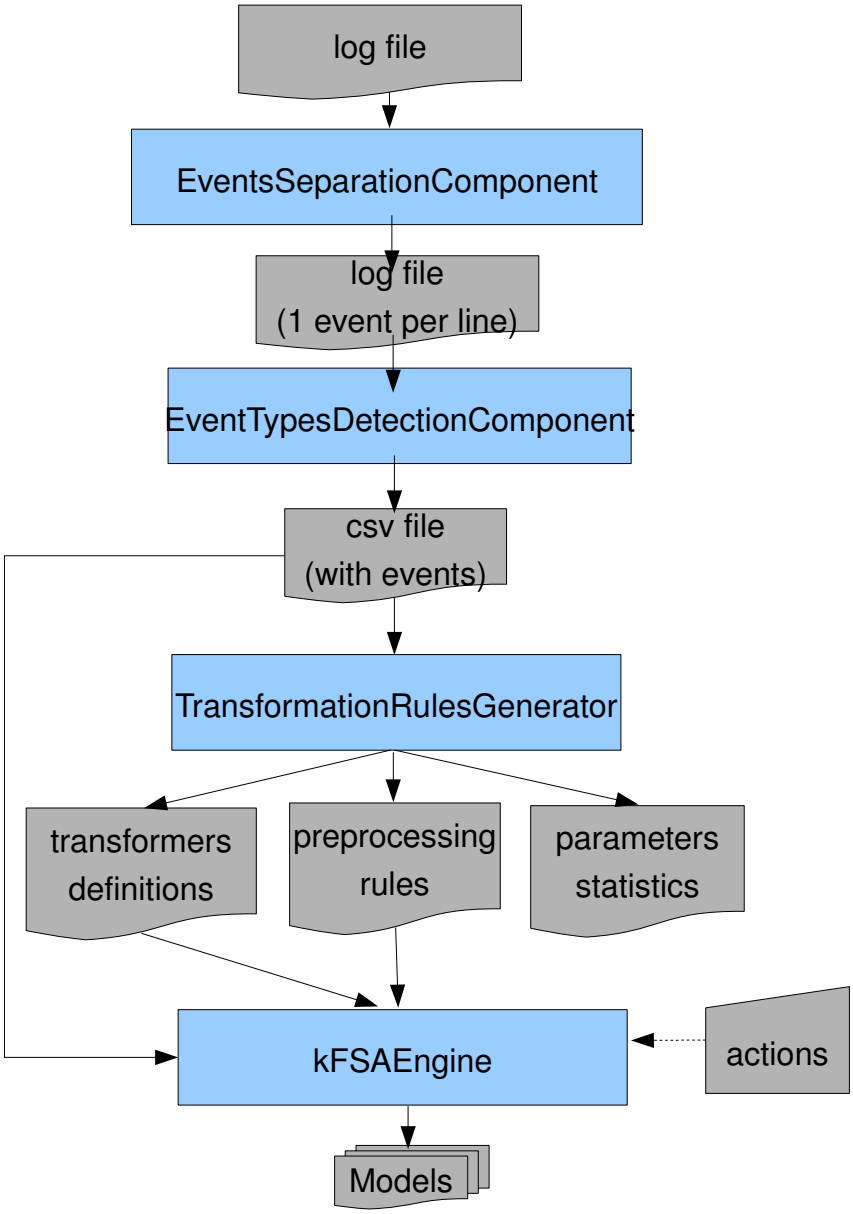

Figure 3.1: Components involved in the model generation phase.

3.3 Failure Analysis

In this fail the logs recorded during faulty executions are first preprocessed following the criterion adopted in the model inference phase and then are compared with the inferred models.

Figure 4.1 shows the components involved in this phase.

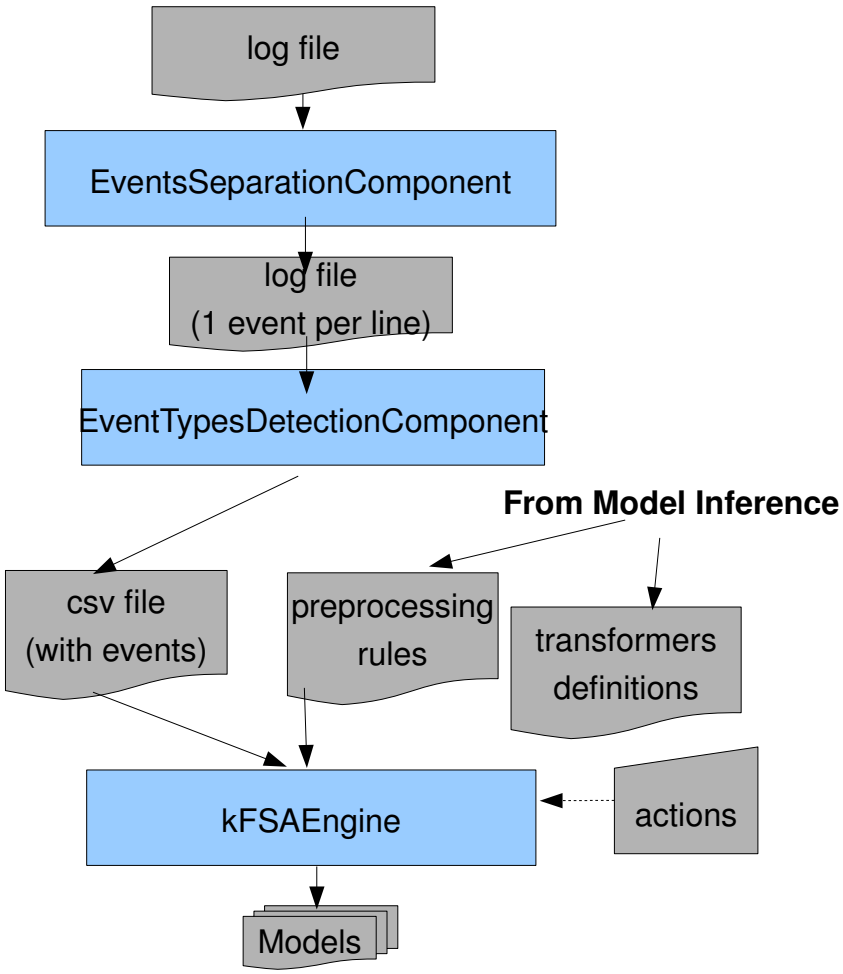

Figure 3.2: Components involved in the failure analysis phase.

The results of this phase are a set of extended models and an anomaly file.

Column name	Description
Component	Name of the component that presents this anomaly.
Anomaly	Anomaly type, can be branch, tail or final state.
Line	Position in the trace in which the anomaly starts. This number corresponds to the position of the event in the trace named checking_<componentName>.trace
State	State of the component FSA in which the anoamly has been found
StateType	State type, can be *existing* if it is a state present in the component FSA, or *new*if it is a state added during a previous extension
Event	Sequence of anomalous preprocessed events observed
Original log line	Position in the original log
Original log event	Sequence of anomalous events observed
To state	State in which the anomaly ends (makes sense only if it is a branch added anomaly).
Branch length	Lenght of the added branch.
Expected	Expected event going out from the anomalous state
Expected incoming	Events expected before state "To state"

Chapter 4

Examples

4.1 Glassfish deployment failure

This section describe a real case study in which we analyzed log files generated by the Glassfish J2EE application server to detect the cause of a failure while deploying the Petstore [Sun10a] web application.

In this case study we collected the log files produced by glassfish during system tests, derived models from the log files (we applied the three different approaches), and compared the log file produced during the failure. This log file was provided by a user of the system who was not able to deploy the Petstore web application using Netbeans [Gla].

All the files described in this example can be found in folder `examples/glassfishForumUserIssue/`.

4.1.1 Monitoring

In the monitoring phase we collected log files produced by Glassfish while it was performing different functionalities: start-up, shutdown, web application deploy, and response to web application requests.

The log files were recorded with the default log verbosity. Log files are stored in folder `examples/glassfishForumUserIssue/correctLogs`.

4.1.2 Model Generation

In the model generation phase we preprocess the original log files in order to generate a model of the correct log file format.

Table 4.1: RegexBasedRawEventsSeparator parameters.

Parameters	description
$-eventsStartExpression$" \ [# \ |.*	indicates that log messages start with [# \ |
$../correctLogs/server.log*$	expands to all the correct log files

Raw Events Separation

Glassfish records logs in the Uniform Log Format [Sun10b]. Logging messages witten in this format start with `[#` and end with `|]` and can span over different lines. For this reason we need to preprocess the original log files in order to obtain a file in which each log message is recorded in a line.

In order to do this we descend into folder `examples/glassfishForumUserIssue/analysis/` and run RegexBasedRawEventsSeparator with the following command (all in a line):

```
java -cp
path/to/klfa
preprocessing.rawEventsSeparation.RegexBasedRawEventsSeparator
-eventStartExpression "\[#\|2008.*" ../correctLogs/server.log*
events.correct.txt
```

From `examples/glassfishForumUserIssue/analysis/` you can simply run `../bin/runRawEvent`
Table 4.1.2 explains the options used.

Events Types Detection

Event types detection is performed using the `AutomatedEventTypesDetector` tool, which uses SLCT to detect the event types and then parses the given log to produce a final csv file in which component names, events and parameters are separated in different columns.

The usage of the AutomatedEventTypesDetector depends on the kind of analysis you want to perform on your log file. Following Sections list the different options used for the distinct analysis.

Component Level Analysis

```
java -cp
path/to/klfa
```

```
it.unimib.disco.lta.alfa.preprocessing.eventTypesDetection.
AutomatedEventTypesDetector
-slctExecutablePath path/to/slct
-replacement "CORE5076: Using.*" "Using Java" -replacement
".*/domains/domain1/config/" "/domains/domain1/config/" -replacement
"service:jmx:rmi:///jndi/rmi://.*:8686/jmxrmi" "" -replacement
"service:jmx:rmi:///jndi/rmi://.*:8686/jmxrmi" "" -replacement
"\|INFO\|" "" -replacement "\|FINE\|" "" -replacement "\|DEBUG\|" ""
-replacement "\|FINEST\|" "" -replacement "\|FINER\|" ""
-dataExpression "\[#\|2008.*\|.*\|.*\|.*\|.*\|(.*)\|#\]"
-componentExpression "\[#\|2008.*\|.*\|.*\|(.*)\|.*\|.*\|#\]"
-exportRules rules.properties -workingDir trainingCsvGen
-componentsDefinitionFile components.training.properties
events.correct.txt events.correct.csv
```

From `examples/glassfishForumUserIssue/analysis/` you can simply run `../bin/runComponen` Table 4.2 explains the parameters used.

Table 4.2: AutomatedEventsDetector parameters.

Parameters	description
-slctExecutablePath path/to/slct	Path to the SLCT executable
-replacement "CORE5076: Using.*" "Using Java"	Replaces all messages of this type with a default message. We need to replace this message because it causes a false positive due to the different versions of VM used during training and checking, thus we removed the info about the VM.
-replacement ".*/domains/domain1/config/" "/domains/domain1/config/"	Removes the part of the path that generates a false positive.

-replacement "service:jmx:rmi:///jndi/rmi://.*:8686/jmxrmi" ""	Remove this information because the path is system dependent and we do not have enough tests to permit SLCT to understand that the service string is a parameter.
-replacement "service:jmx:rmi:///jndi/rmi://.*:8686/jmxrmi" ""	Same as above.
-replacement "\ \ |$DEBUG$ \ |" ""	Removes the information about the logging granularity. We remove this information not because it introduces false positives, but because make events regular expressions less readable.
-replacement "\ \ |$FINE$ \ |" ""	Same as above.
-replacement "\ \ |$FINER$ \ |" ""	Same as above.
-replacement "\ \ |$FINEST$ \ |" ""	Same as above.
-replacement "\ \ |$INFO$ \ |" ""	Same as above.
-dataExpression "[# \ |2008. $*$ \ \ |. $*$ \ \ |. $*$ \ \ |. $*$ \ \ |. $*$ \ \ |(.$*$) \ |#]"	Tells KLFA where the useful information about the event is positioned using regex grouping.
-componentExpression "[# \ |2008. $*$ \ \ |. $*$ \ \ |. $*$ \ \ |(.$*$) \ |. $*$ \ \ |. $*$ \ \ |#]"	Tells KLFA where the component name is positioned in the log line using regex grouping.
-exportRules rules.properties	Export the patterns detected by SLCT to file `rules.properties` (in the current dir).
-workingDir trainingCsvGen	Generates component files in folder `trainingCsvGen`.
-componentsDefinitionFile components.training.properties	save components ids to file `components.training.properties`.
events.correct.txt events.correct.csv	Original log file (the one that we generated in the previous step). The destination file.

Application Level Analysis and Action Level Analysis

```
java -cp
path/to/klfa
it.unimib.disco.lta.alfa.preprocessing.eventTypesDetection.
AutomatedEventTypesDetector
-dontSplitComponents
-replacement "CORE5076: Using.*" "Using Java" -replacement
".*/domains/domain1/config/" "/domains/domain1/config/" -replacement
"service:jmx:rmi:///jndi/rmi://.*:8686/jmxrmi" "" -replacement
"service:jmx:rmi:///jndi/rmi://.*:8686/jmxrmi" "" -replacement
"\|INFO\|" "" -replacement "\|FINE\|" "" -replacement "\|DEBUG\|" ""
-replacement "\|FINEST\|" "" -replacement "\|FINER\|" ""
-dataExpression "\[#\|2008.*\|.*\|.*\|.*\|.*\|(.*)\|#\]"
-componentExpression "\[#\|2008.*\|.*\|.*\|(.*)\|.*\|.*\|#\]"
-exportRules rules.properties -workingDir trainingCsvGen
-componentsDefinitionFile components.training.properties
events.correct.txt events.correct.csv
```

From `examples/glassfishForumUserIssue/analysis/` you can simply run `../bin/run-ActionLevelEventsDetectionTraining.sh`.

As you can see for both Application and Action Level Analysis the options are the same of the Component Level Analysis except from the additional parameter `-dontSplitComponents`. This happens because the log file format is the same so the parsing options do not change, the only difference is in the way events are detected, in this case we do not need to detect events for components separately.

Transformation Rules Generation

The next step is the automatic detection of the rewriting strategies to be used with the engine. This is achieved by running TransformationRulesGenerator.

```
 java -cp
path/to/klfa
```

Parameters	description
-patterns rules.properties	load events regex from file rules.properties.
-signatureElements 0,1	do not threat columns 0 and 1 as parameters.
events.correct.csv	name of the csv file to analyze.

Table 4.3: TransformationRulesgenerator options

```
it.unimib.disco.lta.alfa.parametersAnalysis.TransformationRulesGenerator
-patterns rules.properties -signatureElements 0,1 events.correct.csv
```

From `examples/glassfishForumUserIssue/analysis/` you can simply run `../bin/run-TransformationRulesGeneration.sh`.

If you already had a CSV file and for this reason you did not run class `EventTypesDetector`, you can generate the transformation rules by running:

```
 java -cp
path/to/klfa
it.unimib.disco.lta.alfa.parametersAnalysis.TransformationRulesGenerator
-signatureElements 0,1 events.correct.csv
```

Table 4.1.2 explains the options used.

Inference of the models

Model inference is done using the LogTraceAnalyzer tool. It first applies the data transformation rules detected by the TransformationRulesGenerator. Then it builds models using the kBehavior inference engine [MP07].

The analysis type is selected by the user providing the corresponding parameters to the LogTraceAnalyzer. In the following paragraphs we explain how to do the different analysis.

Component Level Analysis

```
 java -cp path/to/klfa
tools.kLFAEngine.LogTraceAnalyzer -separator "," -minimizationLimit
100 componentLevel training transformersConfig.txt
```

Parameters	description
-separator ","	separator char used in the csv file.
-minimizationLimit 100	do not minimize FSA if they have more than 100 states.
componentLevel	do component level analysis.
training	learn the models.
transformersConfig.txt	file with the rewriting rules defined for the different data clusters.
preprocessingRules.txt	file with the association between the different instances of rewriting strategies and the different parameters.
events.correct.csv	csv file to load data from.

Table 4.4: LogTraceAnalyzer Component Level Analysis options

```
preprocessingRules.txt events.correct.csv
```

From `examples/glassfishForumUserIssue/analysis/` you can simply run ../bin/runComponentLevelInference.sh.

Table 4.1.2 explains the options used.

Action Level Analysis

```
 java -cp path/to/klfa
tools.kLFAEngine.LogTraceAnalyzer -separator ","
-splitActionLines -actionLines
actions.correct.properties -minimizationLimit
100 actionLevel training transformersConfig.txt
preprocessingRules.txt events.correct.csv
```

From `examples/glassfishForumUserIssue/analysis/` you can simply run ../bin/runActionLevelInference.sh.

Application Level Analysis

```
 java -cp path/to/klfa
tools.kLFAEngine.LogTraceAnalyzer -separator "," -minimizationLimit
100 applicationLevel training transformersConfig.txt
preprocessingRules.txt events.correct.csv
```

From `examples/glassfishForumUserIssue/analysis/` you can simply run ../bin/runApplicationLevelInference.sh.

4.1.3 Failure analysis

Once the failure occurs the faulty log file can be compared with the inferred models to detect anomalies. To do this we have to process the faulty log file in a similar manner as in the model inference phase. Figure 4.1 shows the required steps.

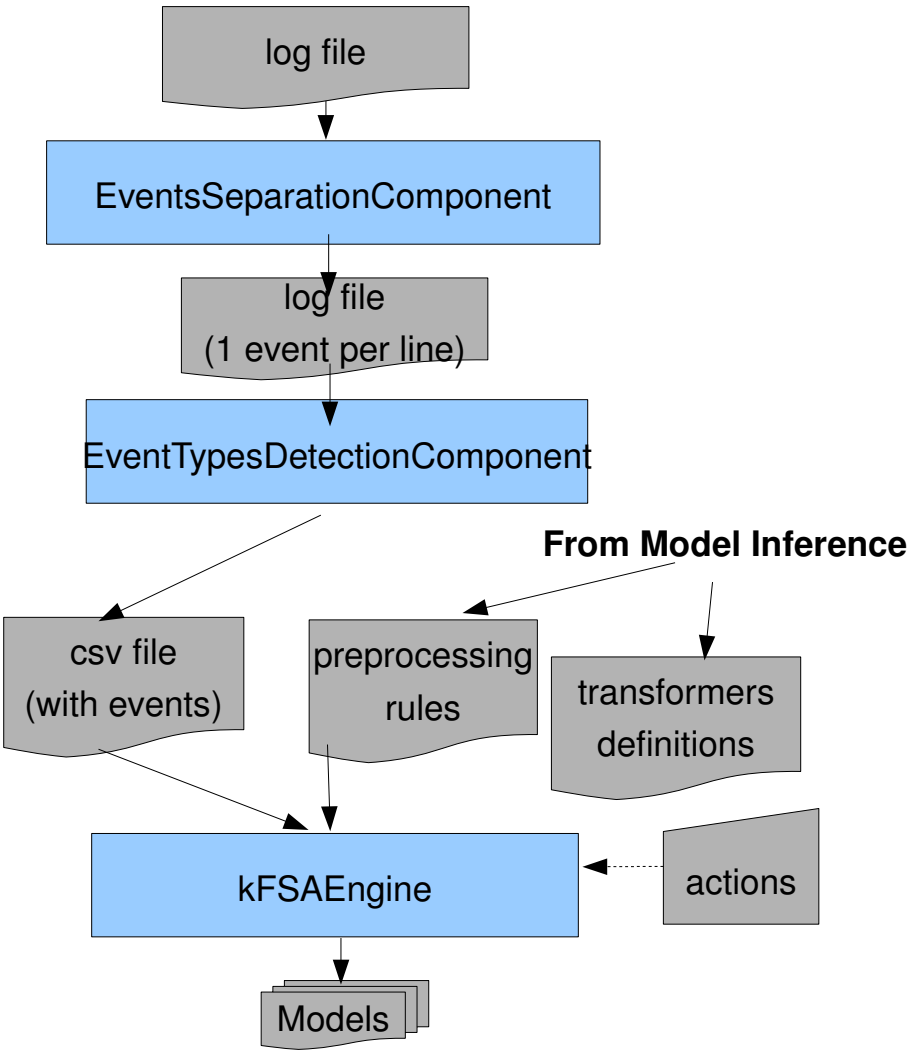

Figure 4.1: Components involved in the failure analysis phase.

Raw Events Separation

The command used to separate raw events is the same as in the ModelGeneration phase except from the input and output parameters.

```
java -cp
path/to/klfa
preprocessing.rawEventsSeparation.RegexBasedRawEventsSeparator
-eventStartExpression "\[#\|2008.*" ../faultyLogs/server.fail.log
events.fail.txt
```

From `examples/glassfishForumUserIssue/analysis/` you can simply run `../bin/run-RawEventsSeparationChecking.sh`.

Events Types Detection

The command is similar as in the Model Generation phase except from the fact that we tell the tool to use the component and rules ids used in the Model Generation phase.

Component Level Analysis

```
 java -cp
path/to/klfa
it.unimib.disco.lta.alfa.preprocessing.eventTypesDetection.AutomatedEventTypesDete
-replacement "CORE5076: Using.*" "Using Java" -replacement
".*/domains/domain1/config/" "/domains/domain1/config/" -replacement
"service:jmx:rmi:///jndi/rmi://.*:8686/jmxrmi" "" -replacement
"service:jmx:rmi:///jndi/rmi://.*:8686/jmxrmi" "" -replacement
"\|INFO\|" "" -replacement "\|FINE\|" "" -replacement "\|DEBUG\|" ""
-replacement "\|FINEST\|" "" -replacement "\|FINER\|" ""
-dataExpression "\[#\|2008.*\|.*\|.*\|.*\|.*\|(.*)\|#\]"
-componentExpression "\[#\|2008.*\|.*\|.*\|(.*)\|.*\|.*\|#\]"
-loadComponents components.training.properties -exportRules
rules.checking.properties -workingDir checkingCsvGen
-loadEventPatterns -patternsDir trainingCsvGen
-componentsDefinitionFile components.fail.properties events.fail.txt
events.fail.csv
```

Examples

From `examples/glassfishForumUserIssue/analysis/` you can simply run ../bin/runComponentLevelEventsDetectionChecking.sh.

Application Level Analysis and Action Level Analysis

```
 java -cp
path/to/klfa
it.unimib.disco.lta.alfa.preprocessing.eventTypesDetection.AutomatedEventTypesDete
-dontSplitComponents -replacement "CORE5076: Using.*" "Using Java"
-replacement
".*/domains/domain1/config/" "/domains/domain1/config/" -replacement
"service:jmx:rmi:///jndi/rmi://.*:8686/jmxrmi" "" -replacement
"service:jmx:rmi:///jndi/rmi://.*:8686/jmxrmi" "" -replacement
"\|INFO\|" "" -replacement "\|FINE\|" "" -replacement "\|DEBUG\|" ""
-replacement "\|FINEST\|" "" -replacement "\|FINER\|" ""
-dataExpression "\[#\|2008.*\|.*\|.*\|.*\|.*\|(.*)\|#\]"
-componentExpression "\[#\|2008.*\|.*\|.*\|(.*)\|.*\|.*\|#\]"
-loadComponents components.training.properties -exportRules
rules.checking.properties -workingDir checkingCsvGen
-loadEventPatterns -patternsDir trainingCsvGen
-componentsDefinitionFile components.fail.properties events.fail.txt
events.fail.csv
```

From `examples/glassfishForumUserIssue/analysis/` you can simply run ../bin/runApplicationtLevelEventsDetectionChecking.sh or ../bin/runActionLevelEventsDetectionChecking.sh.

Comparison against the models

Comparison against the model is done calling the LogTraceAnalyzer tool and giving the analysis type used in the model generation phase and specifying that we are now doing the comparison.

Component Level Analysis

```
java -cp path/to/klfa tools.kLFAEngine.LogTraceAnalyzer
-separator "," -minimizationLimit 100 componentLevel checking
transformersConfig.txt preprocessingRules.txt events.fail.csv
```

From `examples/glassfishForumUserIssue/analysis/` you can simply run ../bin/runComponentLevelAnomalyDetection.sh.

Action Level Analysis

```
 java -cp path/to/klfa
tools.kLFAEngine.LogTraceAnalyzer -separator "," -minimizationLimit
100 actionLevel checking transformersConfig.txt
preprocessingRules.txt events.correct.csv
```

From `examples/glassfishForumUserIssue/analysis/` you can simply run ../bin/runActionLevelAnomalyDetection.sh.

Application Level Analysis

```
 java -cp path/to/klfa
tools.kLFAEngine.LogTraceAnalyzer -separator "," -minimizationLimit
100 applicationLevel checking transformersConfig.txt
preprocessingRules.txt events.correct.csv
```

From `examples/glassfishForumUserIssue/analysis/` you can simply run ../bin/runApplicationLevelAnomalyDetection.sh.

Anomalies interpretation

In the model comparison phase the tool detects the anomalies present in the faulty log files and report them to the user by saving them in the file `klfaoutput/anomalies.csv`.

The last phase of the technique involves actively the user who has to inspect the reported anomalies, and use them as a guide to inspect correct and faulty files to detect the problem.

Table 4.1.3 shows the anomalies detected by the tool in the given case study.We imported the csv file produced by the tool, anomalies.csv, and sorted the items according to the column `Original Event Line`. In the next paragraphs we are going to interpret them to give an exhaustive explanation of the problem.

Anomaly 1 Anomaly 1 appears in line 15 of the faulty log file. The anomaly regards component `com.sun.jbi.framewor` (the id 5 correspond to this component as you can see from file `components.training.properties`). In this case the anomaly is not caused by an unexpected event, but the system detects that the events regarding component 5 stopped before expected. In fact a new final state was added to the automaton. By opening the automaton with the command `java`

Table 4.5: Anomalies detected by KLFA for the Glassfish case study

Comp.	Anomaly	Line	State	State Type	Event	Original log line	Original log event	Expected
5	FinalState	1	q2	Existing	5_R0065	15	5,R0065	5_R0064 5_R0066__0__0
0	FinalState	5	q8	Existing	0_R0055)	20	0,R0055	0_R0052) 0_R0057)
GLOBAL	Tail	13	q109	Existing	14_R0020__0)	21 14,R0020,java-petstore2.0ea5		3_R0032) λ
14	FinalState	1	q4	Existing	14_R0020__0)	21 14,R0020,java-petstore-2.0-ea5		14_R0023)
4	Tail	7	q12	Existing	4_289331648)	24	4,289331648	4_-1628344215) 4_R0073) 4_1573705168 4_R0075)
17	Tail	1	q3	Existing	17_-811928006)	25	17,-811928006	17_R0003);
3	Tail	5	q10	Existing	3_-1648356848)	27	3,-1648356848	3_R0032) 3_R0031)
23	New Component							

`-cp path/to/klfa tools.ShowFSA klfaoutput/5.fsa` we can see that many more events are expected. Furthermore by looking at the faulty log file we can see that the file is very short, so we can deduce that it was truncated by the user or the application was blocked.

The `Event` column in this case do not represent the wrong event occurred but the last event seen. The id of this last event is R0065, which correspond to the event regex "JBIFW0010 JBI framework ready to accept requests.".

Anomaly 2 Anomaly 2 regards component `javax.enterprise.system.core`, also in this case the anomaly is caused by the premature end of messages.

Anomaly 3 Anomaly 3 regards component GLOBAL. This is not a real component, it is a keyword used to indicate the automata that describes the way components execution alternate.

The anomaly type is Tail, it indicates that an unseen tail was added to the state q109. The first anomalous event seen is 14_R0020_0, while it expected 3_R0032, 3_R0031, 13_1394096499, or 2_-2135717321 (the last three are detected following the ϵ transition). The more interesting is the first one, which indicates that a deploy message from component 3 (`javax.enterprise.system.tools.admin`) is missing from the log. We do not know if it indicates the cause of the failure (This anomaly could depend on the fact that in one case it was used the Glassfish `asadmin` tool while in the other not).

Anomaly 4 Anomaly 4 regards component 14: the component recorded less messages than expected. This is because the premature end of the log file. KLFA expected a message of type R0023 (`^(.*) AutoDeploy Disabling AutoDeployment service.`), before stopping the Glassfish server. We have an anomaly because in this log the stopping phase of the server is not recorded.

Anomaly 5 Anomaly 5 indicates that at line 24 an anomalous event 4_289331648 occurs. The event ID in this case is an hash. The AutomatedEventTypesExtractor assigns to a raw event line its hashcode as its id when the raw event is an outlier. We have an outlier when a raw event does not match any event regexp.

The occurrence of an hashcode as an anomalous event can have two meanings: the specific event was never seen in the correct logs analyzed or the event was present in the logs analyzed but its was present very few time and it was not considered an event type (by default this happens when an event occurs just once). In the first case it can be an exceptional event that appear as a consequence of a failure, or it can be a false positive caused by event regexp that do not generalize enough the data. This should happen if in the correct log files we have events in which a parameter remains constant over all their occurrences: in this case the parameter will be considered by SLCT as part of the event regex, and in case the value change in the faulty execution because of environmental reasons (e.g. domain of a web server) it will be detected as an anomaly which may be not related to

the experienced failure (pay attention it should also be the case in which in the correct execution the system was behaving correctly because of this constant value).

In this case to further inspect the anomalous event we need to take a look at the faulty log file (`events.fail.txt`), we see that there is an exception in line 24, which is related to the failure. The exception was never seen in the correct log files (search for 289331648 in the correct log).

Anomaly 6 Anomaly 6 occur at line 25, the event 17_-811928006 was unexpected. As in the previous case the hashcode-id was generate because of a message never seen before (the exception).

Anomaly 7

Anomaly 7 is detected in line 27 of the trace file. Also in this case if we take a look at the faulty log file (`events.fail.txt`), in line 27 we see that there is an exception, which is related with the failure. The technique has detected an useful information for the root cause analysis.

Anomaly 8

Anomaly 8 indicates that a new component appeared. If we open `components.fail.properties` we see that component id 23 correspond to component `com.sun.org.apache.commons.modeler.Regist`. By looking for it in the failure log we see that it appears because of an event occurred as a consequence of the failure.

Bibliography

[CPMP07] Domenico Cotroneo, Roberto Pietrantuono, Leonardo Mariani, and Fabrizio Pastore. Investigation of failure causes in workload-driven reliability testing. In *proceedings of the Fourth international workshop on Software quality assurance*, pages 78–85. ACM, 2007.

[Gla] Glassfish user forum. Glassfish configuration issue. http://forum.java.sun.com/thread.jspa?threadID=5249570, visited in 2010.

[MP07] L. Mariani and M. Pezzè. Dynamic detection of COTS components incompatibility. *IEEE Software*, 24(5):76–85, September/October 2007.

[MP08] Leonardo Mariani and Fabrizio Pastore. Automated identification of failure causes in system logs. In *Proceedings of the 19th IEEE International Symposium on Software Reliability Engineering (ISSRE'08)*, pages 117 – 126, Washington, DC, USA, 2008. IEEE Computer Society.

[Sun10a] Sun, visited in 2010. Java PetStore. http://java.sun.com/developer/releases/petstore/.

[Sun10b] Sun, visited in 2010. GlassFish v3 Application Server Administration Guide. http://docs.sun.com/doc/820-4495.

[Vaa03] R. Vaarandi. A data clustering algorithm for mining patterns from event logs. In *Proceedings of the 3rd IEEE Workshop on IP Operations and Management*, 2003.