ICT PROJECT 258109
Monitoring Control for Remote Software Maintenance

Project number: 258109
Project Acronym: FastFix
Project Title: Monitoring Control for Remote Software Maintenance
Call identifier: FP7-ICT-2009-5
Instrument: STREP
Thematic Priority: Internet of Services, Software and Virtualization
Start date of the project: June 1st, 2010
Duration: 33 months

D4.5: 1st iteration prototype of the pattern mining module

Lead contractor: S2
Author(s): Sergio Zamarripa, Maria del Carmen Calle, Emitzá Guzmán, Tobias Röhm, Dennis Pagano, Walid Maalej
Submission date: May 2012
Dissemination level: Public

Project co-founded by the European Commission under the Seventh Framework Programme

© S2, TUM, LERO, INESC ID, TXT, PRODEVELOP
Abstract: This document reports on the implementation of the 1st prototype of the pattern mining module, a subcomponent specialized in the identification of event patterns that allow for failure prevention and performance degradation. The idea of the pattern mining feature is centered on the design of a mechanism to acquire new software maintenance knowledge about existing errors, from the events flowing in the system. This module is strongly connected to the event correlation system, since the kind of knowledge that the system aims to acquire will feed the event correlation system back, taking advantage of the new mined patterns to detect new errors, that the FastFix system was not able to detect previously with the existing knowledge.

This document has been produced in the context of the FastFix Project. The FastFix project is part of the European Community's Seventh Framework Program for research and development and is, as such, funded by the European Commission. All information in this document is provided "as is" and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability. For the avoidance of all doubts, the European Commission has no liability in respect of this document, which is merely representing the authors view.
Contents

1 Introduction ...........................................................................................................................................4

2 Overview ..............................................................................................................................................6
  2.1 Code Metrics .....................................................................................................................................7
  2.2 Used Libraries – Open source ........................................................................................................8

3 Pattern Mining Tool ..........................................................................................................................10
  3.1 Regular application behaviour training .........................................................................................11
    3.1.1 Detachment ................................................................................................................................11
    3.1.2 Data preparation .......................................................................................................................13
    3.1.3 Algorithm execution ..................................................................................................................16
  3.2 Error pattern recognition ...............................................................................................................18
  3.3 Pattern metadata settings .............................................................................................................19

4 Running the prototype .......................................................................................................................22
  4.1 Infrastructure for FastFix Development .........................................................................................22
  4.2 Build and Run FastFix ...................................................................................................................22
    4.2.1 Build .......................................................................................................................................22
    4.2.2 Run existing bundles ...............................................................................................................24

5 Next steps ............................................................................................................................................25

6 Bibliography .......................................................................................................................................27
1 Introduction

One of the major challenges and useful features to be provided by FastFix to software maintenance teams is the development of a tool, to be able to acquire new knowledge associated with patterns of error or degradation trends, providing an automated tool oriented to machine learning of new symptoms of failure which, otherwise, would go unnoticed through the system.

Concerning the pattern mining module as part of the FastFix system, it must be seen as part of the set of automated tools supporting software maintenance tasks. Software maintenance is a field of knowledge strongly supported on engineer’s experience, especially regarding the different situations of error occurrence. Software engineers tend to focus on previous gained experience on multiple cases, usually by mentally classifying the current issue and looking for concrete causes associated with the resulting classification. Existing knowledge has been gained through the years coping with each error, classifying it from the existing symptoms, analyzing its causes, and looking for solutions that can be applied to recover from the effects produced by the cause, or even, in the best case, to remove the cause itself. Nevertheless, this process historically lacks from automation [13], both for pattern acquisition to error detection. The conceptual model of the event processor and pattern recognition, as described in D4.3e [1] is based on providing automated tools for detection errors, taking advantage of a representation of patterns in an ontology and acquiring new patterns using machine learning techniques.

FastFix provides one of the best landscapes to the pattern mining task, since it is meant to gather events in the system, a feature that is enhanced through the development of new sensors. Each new event in the system contains relevant information, especially when it is involved in the occurrence of an error or a degradation of the operation that the system is intended for. When the system is facing a new kind of error, different from the errors associated to the patterns the system already knows, the pattern mining module is responsible for recognizing unnoticed sequences of error and showing them to the end user, which is the member of the software maintenance team.
Regarding the kind of information to be mined, at a global level, the pattern mining module is meant to discover hidden knowledge from multiple data streams. Taking into account the event correlation operation process described in section 2.1 of “D4.4 1st iteration prototype of the event processor” [2], the pattern mining procedure can feed the system with the new discovered patterns, as already introduced in section 2.2 of the same document.

The event processor prototype is sustained on the representation of existing error patterns in ontologies, in order to detect errors in real time, using the information and concepts represented in the ontology. Hence, the final purpose of the pattern mining module should take into account the insertion of the new recognized patterns as gained knowledge, as soon as they are consolidated (e.g. validated to avoiding false positives). Hence, the FastFix system is continuously evolving, learning from the information in the system, being ready for new situations of error, minimizing human intervention during the process and favoring the automation of knowledge acquisition from the continuously flowing data across the system.

This document aims to give an overview of the prototype as a companion to the source code of the pattern mining tool in the FastFix platform. The following sections detail the design and usage of these components.
2 Overview

The pattern mining module is one of the common bundles, since it might be potentially used by several server components like the self-healing and the event correlation bundles, although originally it is meant to provide patterns to the server event correlation component. The current version of the pattern mining bundle starts analyzing a list of all the events in the communication infrastructure for a configurable period of time. Hence, the expected input for the prototype is a large amount of events, with information of the concrete time of occurrence of each event, its type and the content of its properties. In order to recognize patterns of error within all this amount of events, the FastFix Pattern Mining procedure is structured in three stages:

1. **Regular application behavior training.** First, we take advantage of machine learning algorithms, not to find sequences of error directly, but to find the sequences of normal application and system behavior, and storing these sequences as reference values for next steps. Hence, the outcomes at the end of this step are a list of regular and non-erroneous sequences of events (normal patterns) of the application and the system, which results in an overview of what is expected. These patterns are first computed with high support on normal executions, on what we can be understood as a training phase.

2. **Error pattern recognition.** This second step is meant to discover the sequences of error (error patterns), using the outcomes of the first step. Whenever a sequence occurs and breaks the regular behavior sequences, it indicates that there might be a problem behind that sequence. Once these sequences are identified, the system gathers information about the content of the events of each sequence. Hence the outcomes of this step are candidates of error patterns, which are sequences of different types of events and the properties’ content of each event.

3. **Pattern metadata settings.** Once we have the possible error sequences, what we have at this stage is a sequence of different types of events and some concrete content. Nevertheless, since we need to provide a relevant pattern, that can be identified the next time it occurs, we must abstract away from the concrete
emergence of this error and we have to generalize what we found. If the same type of error occurs again, maybe the concrete user input or the stack traces will be different, but the error is the same. Hence, we have to provide a mechanism to provide the pattern to further identify this kind of error when it happens again. We have to define what part of this pattern setting procedure will be automated and what part might be supported, even with the help of the user expertise. The final pattern, conveniently structured, is what is supplied to the event correlation system as gained knowledge, which allows detecting the newly discovered pattern any time that an error of the same type occurs, from now on.

This approach contrasts with our original approach, where the pattern mining system directly tried to find patterns of error, as part of the first step. Nevertheless, with the original approach, the results were highly inefficient, and it took considerable time to find bugs, since these errors had to occur with considerable frequency to meet the minimum support of machine learning algorithms. The advantage of our three-step approach is that we don’t need to see an error several times before we can have a characterization of what is wrong. The drawback of the final approach is that we need a set of “normal” traces on which to mine patterns in the first place. Nevertheless, this is reasonable in the scope of the project, since FastFix is oriented to support software teams in the maintenance phase, where a lot of regular application behavior can be gathered coming from testing or monitored from the users’ environment through the FastFix Client.

### 2.1 Code Metrics

Since the FastFix platform is based on the OSGi framework, the pattern mining module has been organized as an OSGi bundle, which is hosted at [https://repository.fastfixproject.eu/svn/fastfix](https://repository.fastfixproject.eu/svn/fastfix) (authenticated access):

- **eu.fastfix.common.event.pattern.mining**: this bundle performs pattern mining over the events in the FastFix system during a period of time. It provides a list of recognized patterns over a minimum support, based on association
rules and sequential pattern mining algorithms (19 classes, 43 methods, 805 lines of code [LOC]).

2.2 Used Libraries – Open source

In order to reuse existing open source technologies in the field of machine learning, we have been evaluating several options in order to be integrated in the first version of our pattern mining prototype.

The execution of pattern mining algorithms starts after the pattern mining bundle has formatted conveniently the information about events (type, timestamp, content) into an input file, ready to be analyzed by any pattern mining algorithm.

The best way to evaluate these machine learning technologies consisted of installing them and using the formatted files that the pattern mining bundle builds after acquiring the events, mapping them with type identifiers, determining groups of events based on the time of occurrence from their timestamps (which we will refer as detachment).

One of the first open source ML technologies analyzed is Apache Mahout [6], which provides a set of scalable machine learning libraries build in Java. Focusing on pattern mining algorithms, it only provided Parallel Frequent Pattern Growth [7], which is good to find which types of events usually come together, but it does not include sequential pattern mining. Regarding our approach, introduced in section 2, the desired outcome of the first step (regular application behaviour training) should be a list of sequences, associated to the normal behaviour of the system.

Looking for other alternatives that could take into account sequential pattern mining, we tested R, which includes a complete set of association rules and sequential pattern mining algorithms (e.g. SPADE: Sequential PAttern Discovery using Equivalence classes [8]). SPADE provided satisfactory results (some of them were part of the first tests described in the chapter 4 Evaluation). Even though R was a good alternative because of the set of algorithms it provides and the availability of sequential pattern mining algorithms, one of the
criteria to be selected was also the possibility to be integrated as part of our Java - OSGi architecture. Nevertheless, R source is written in C, so the integration as part of a Java OSGi bundle, although it would have been possible, it would have required a remarkable use of resources.

Hence, the open source machine learning alternative to be integrated should provide a set of sequential pattern mining algorithms as well as association rules algorithms and if their source is Java, it would make integration easier and FastFix could also contribute to this open source code with new features or algorithms.

SPMF [5] is an open-source data mining framework written in Java, and distributed under the GPL v3 license. It was originally a sequential pattern mining framework, but currently it also includes implementations of association rule mining, sequential rule mining and frequent itemset mining algorithms [9]. These algorithm implementations are especially useful for the pattern mining bundle, since they can analyze the list of events that are part of long runs of FastFix, after mapping these events and analyzing their contents, to propose possible patterns of error.
3 Pattern Mining Tool

The pattern mining bundle is structured in the following packages:

eu.fastfix.common.event.pattern.mining.datapreparation:
This package provides the classes for the implementation of the pre-processing of the input data.

eu.fastfix.common.event.pattern.mining.model
This package provides the classes required in the system to build itemsets and sequences.

eu.fastfix.common.event.pattern.mining.metadata
This package provides the classes for the implementation of the metadata analysis.

eu.fastfix.common.event.pattern.mining.sequence
This package provides the classes for the implementation of the determination and analysis of sequences, to find relevant sequences breaking the regular behaviour of the system.

capfv.spmf.sequentialpatterns
This package is part of the open source code of the SPMF Framework [5], it includes the set of sequential pattern mining algorithms.

capfv.spmf.associationrules
This package is part of the open source code of the SPMF Framework [5], it includes the set of association rule mining algorithms.

capfv.spmf.frequentpatterns
This package is part of the open source code of the SPMF Framework [5], it includes the set of frequent pattern mining algorithms.

These packages are used through different steps of the pattern mining bundle, which we describe in detail in the following sections, showing some evaluation results for this iteration:
3.1 Regular application behaviour training

We start from the initial situation, where we received a large set of events from the context system. As already introduced in section 2, our first objective is to find sequences of normal application and system behavior, and storing these sequences as reference values which results in an overview of what is expected.

This phase consists of the following tasks:

1. Detachment: The timestamp of the different types of events is the property that is used to represent and detach the sequences contained in the set of events. Hence, the component will convert the list of events as a set of sequences depending on the time of occurrence and its type, since it is based on separating groups of events associated to an action, since they are closer in time.

2. Data preparation: All the information coming from the context system must be structured and converted into a format that can be understood by a machine learning algorithm. In other words, we define the different types of events that are going to be analyzed by the pattern mining algorithm and the pattern mining system maps each type of event with an identifier.

3. Algorithm execution: As soon as the input is formatted and detached, the pattern mining algorithms are executed over this input, providing a list of mined sequences that satisfy a minimum support.

3.1.1 Detachment

Our first objective is to make a first structure of event groups in order to define what are the sequences of our input data. Before starting with the whole process, we must introduce some concepts about sequence representation. An itemset is a collection of one or more items that is represented between brackets (e.g. \{Bread, Butter\}). When we represent sequences, in other words an ordered list of itemsets, we will represent them as lists of itemsets separated by commas (e.g. (\{Bread, Butter\},\{Milk\})) in other words: {Bread,Butter} → {Milk}).
Hence, we need a mechanism to determine itemsets and sequences, and also a criterion to determine when the sequence ends and we must consider the next element as part of a different sequence. The purpose of this section is to build these input sequences as faithfully as possible, in order to prepare the best scenario for the pattern mining algorithms to find remarkable patterns.

We are using two different criteria to define itemsets and sequences:

Itemset: We consider that some events are part of the same itemset, based on its closeness in time (using the timestamp, which identifies the moment of occurrence of each event).

Sequences: We consider that some itemsets are part of the same sequence as long as the sequence is not containing a sequence end-point. Additionally, other time criterion is established (e.g. if there is no sequence end-point for a certain period of time, in our case, 5 seconds, we consider that the sequence is complete).

A sequence end-point is a special type of event that might be understood as the end of an action as a whole. There are several types of events that can be understood as sequence end-points (e.g. for a web application: HttpResponse, for a desktop: WindowRepaint). Conceptually, a sequence end-point allows to define the border lines between an action and the next one.

With these criteria in mind, the SessionGenerator class is the responsible for building the sequences, first building each itemset and after that, using the end-point criterion to find out when the sequence is done. This criterion uses of existing information about every type of events, since there are some types that can be understood as end-points of a sequence. For example, most of sequences of events will be understood as cycles like the following: {TextInput}→ {PressButton}→ {HttpRequest}→ {HttpResponse}. In this example, the HttpResponse can be understood as a sequence end-point, in other words, as an end of cycle.

Thus, the SessionGenerator takes this information into account in order to build the sequences, considering all the itemsets as part of a sequence until it finds a sequence end-point type. Another type of event that must be considered as a sequence end-point is the strong symptom.
3.1.2 Data preparation

This section describes the data preparation process, which are the actions completed by the pattern mining bundle to convert events data to a formatted input that is understandable for the machine learning algorithms.

Although the integrated version will be integrated with the context system, the first iteration of this prototype is not integrated. Instead, we are using a list of events that are printed into a file, with all the relevant information of each event, like its type, start and end timestamps, associated user and properties names and values. Here is an example of the information within this file (after the detachment phase):

http://www.fastfixproject.eu/ontologies/MonitoringOntology.owl#TextInput,1334148073129;hasText=Department 1
http://www.fastfixproject.eu/ontologies/MonitoringOntology.owl#PressButton,1334148074431;hasLabel=List procedures
http://www.fastfixproject.eu/ontologies/MonitoringOntology.owl#HttpRequest,1334148074827;
http://www.fastfixproject.eu/ontologies/MonitoringOntology.owl#HttpResponse,1334148074982
...

As can be seen, the information is not understandable for an algorithm that should try to look for relevant sequences, based on the types of event.

This information is first converted with the PrintoutConverter class to an structure with a list of the relevant elements regarding pattern mining, in other words, we keep the type, the start timestamp and the metadata content (e.g. text input, button label, error text, exception stacktrace,...).

Focusing on the type, we established a categorization of different types of events that are analyzed to find out what events and what types make the pattern up (associated with the regular application behaviour).
<table>
<thead>
<tr>
<th>Event Type</th>
<th>Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>ContextEvent</td>
<td>0001</td>
</tr>
<tr>
<td>ApplicationEvent</td>
<td>0002</td>
</tr>
<tr>
<td>ApplicationCrash</td>
<td>0003</td>
</tr>
<tr>
<td>ApplicationExit</td>
<td>0004</td>
</tr>
<tr>
<td>ApplicationStart</td>
<td>0005</td>
</tr>
<tr>
<td>AttributeAccess</td>
<td>0006</td>
</tr>
<tr>
<td>ReadAccess</td>
<td>0007</td>
</tr>
<tr>
<td>WriteAccess</td>
<td>0008</td>
</tr>
<tr>
<td>ConfigurationEvent</td>
<td>0009</td>
</tr>
<tr>
<td>BrowserConfigurationEvent</td>
<td>0010</td>
</tr>
<tr>
<td>ExceptionThrow</td>
<td>0011</td>
</tr>
<tr>
<td>ExternalComponentCall</td>
<td>0012</td>
</tr>
<tr>
<td>LibraryCall</td>
<td>0013</td>
</tr>
<tr>
<td>SystemCall</td>
<td>0014</td>
</tr>
<tr>
<td>FileRead</td>
<td>0015</td>
</tr>
<tr>
<td>FileWrite</td>
<td>0016</td>
</tr>
<tr>
<td>HttpRequestEvent</td>
<td>0017</td>
</tr>
<tr>
<td>HttpRequestOnClientSide</td>
<td>0018</td>
</tr>
<tr>
<td>HttpRequestOnServerSide</td>
<td>0019</td>
</tr>
<tr>
<td>MethodCall</td>
<td>0020</td>
</tr>
<tr>
<td>HttpResponse</td>
<td>0021</td>
</tr>
<tr>
<td>ConfigurationChange</td>
<td>0022</td>
</tr>
<tr>
<td>DbSelectEvent</td>
<td>0023</td>
</tr>
<tr>
<td>DbInsertEvent</td>
<td>0024</td>
</tr>
<tr>
<td>JVMEvent</td>
<td>0025</td>
</tr>
<tr>
<td>NetworkEvent</td>
<td>0026</td>
</tr>
<tr>
<td>OSEvent</td>
<td>0027</td>
</tr>
<tr>
<td>UserEvent</td>
<td>0029</td>
</tr>
<tr>
<td>ExecuteCommand</td>
<td>0030</td>
</tr>
<tr>
<td>GUIInteraction</td>
<td>0031</td>
</tr>
<tr>
<td>PressButton</td>
<td>0032</td>
</tr>
<tr>
<td>WatchView</td>
<td>0033</td>
</tr>
<tr>
<td>Read</td>
<td>0034</td>
</tr>
<tr>
<td>TextInput</td>
<td>0035</td>
</tr>
</tbody>
</table>

**Table 1 Mapping of some event type identifiers**
Hence, each occurrence of a PressButton will be identified as “0032”, in order to simplify the input to feed the machine learning algorithms. Our sequence of example is represented as:

\{0035\}, \{0032, 0019\}, \{0021\}

Depending on the machine learning libraries and the algorithm, the input format to represent the sequences will be different. The eu.fastfix.common.event.pattern.mining.datapreparation package contains a SourceFormatter class, which is responsible for formatting the sequences of itemsets identified into an input file that can be processed by the corresponding algorithm.

Even though we are using SPMF as the integrated data mining framework, we have taken into account also the R input format, in order to compare results between other algorithms, during the training phase.

We will take the recently described sequence as reference to explain the input format:

\{0035\}, \{0032, 0019\}, \{0021\}

For example, the expected format for SPADE [11] in R:

```
0  0  1  0035
1  0  2  0032  0019
2  0  1  0021
```

In this format, each line represents an itemset, so the first column is the number of itemset, the second is the sequence number identifier, the third column represents the total number of elements in the itemset and the rest of columns are the elements of the itemset.

Instead, the expected format for the same sequence for the PrefixSpan algorithm in SPMF is as follows:

```
0035  -1  0032  0019  -1  0021  -2
```
Here, we consider itemsets separated by “-1” and sequences separated by “-2”.

These structures, especially the one for SPMF, are the ones that feed the pattern mining algorithms to extract and discover relevant patterns of expected application behavior. In the following section, we are going to describe how this process is done.

3.1.3 Algorithm execution

Once we have the formatted source with all the sequences, we must describe how this formatted source will be analysed through the selected pattern mining algorithms. First of all, the input data is used to build a SequenceDatabase, using the SPMF resources.

Before executing the algorithms, the system must specify the main input for the algorithm execution: minimum support. This minimum support is the minimum frequency of a sequence to be considered as frequent. This parameter is one of the key elements to survey about the pattern mining operation, since it would give either less or more results than expected, depending on its value. Hence, this value is one of the main things to tune on the pattern mining prototype. Our first evaluation results are oriented, among other things, to dynamically calculate the appropriate minimum support, depending on the dataset of events.

Once we have set the input data and the minimum support, the system makes a call to run the current selected algorithm. For our first tests, we have selected the PrefixSpan algorithm [10]. As J.Pei et al. described in “Mining Sequential Patterns by Pattern-Growth”, a comprehensive performance study shows that PrefixSpan, in most cases, outperforms the apriori-based algorithm GSP, FreeSpan, and SPADE [11] (a sequential pattern mining algorithm that adopts vertical data format), and PrefixSpan is the fastest among all the tested algorithms.

Nevertheless, at this stage of our evaluation, we do not discard other algorithms for sequential pattern mining, also available in SPMF, like BIDE+ [12], as well as a combination with association rule mining algorithms. Once the system has been built to make this evaluation possible, it will be done in the next iteration of the pattern mining prototype.
**Evaluation**

In a complex task such as pattern mining, early evaluation is especially important, particularly to determine whether the approaches being made are the most appropriate.

In this direction, the main objective of this section is to show the most remarkable results of the pattern mining prototype. As important as evaluating early is to evaluate over real data, in order to analyse results from a known environment, where we already know the concrete error pattern to be found. Of course, this will not be the final scenario where the pattern mining operation will take part, but we consider that is necessary to start evaluating results in a supervised mode, in order to find what allows the system to narrow the results to the expected pattern. With this purpose, we focused on one of the current FastFix trial applications (Espigon), trying to look for relevant patterns of regular behaviour. The results in this section are provided with the PrefixSpan algorithm (already integrated as part of the pattern mining bundle) and using SPFM.

The evaluation was performed with the following values:

- Number of event traces: 4
- Average size of event traces: 320 events
- Number of distinct event types: 8

The results of the algorithm execution are a list of the mined sequences that meet the minimum support, specifying the support of each sequence (only sequences over minimum support are returned) as well as the confidence, which, taking X->Z as a sequence, measures how often items in Z appear in transactions that contain X. Example of mined patterns (regular behaviour):

```
0032 0019 -1 0021 -1 SUP: 0.4 CONF: 0.67
0035 -1 0032 0019 -1 0021 -1 SUP: 0.34 CONF: 0.52
0032 0019 -1 0023 -1 0021 -1 SUP: 0.31 CONF: 0.41
0035 -1 0032 0019 -1 0024 -1 0021 -1 SUP: 0.28 CONF: 0.34
```

Basically, the found patterns are showing normal application behaviour as expected:
At this stage, we found that the number of event types is limited, and as a consequence, the number of relevant mined patterns is also limited. For the next iteration, the number of event types will be greater, and the expected mined patterns should be richer. Additionally, the support will also be reviewed, since it will be affected by the variability of new types.

Additionally, in terms of performance measure of the current algorithm execution, the system shows information about the time spent in the calculation, as well as the number of sequences with the minimum support and memory in use:

```
============= Algorithm - STATISTICS =============
Total time ~ 380 ms
Frequent sequences count : 19
Max memory (mb) : 1.01854687519
===================================================
```

### 3.2 Error pattern recognition

After the regular behavior training phase, the outcomes are a list of regular behavior patterns, after being executed over normal executions. These patterns are the references for the error pattern recognition phase, with the aim of finding cases where these sequences are incomplete or broken, which is an indication of unexpected behavior and, as a consequence, a possible error pattern.

Recovering our example, at this stage, we have the following pattern of normal execution:

```
0032 0019 -1 0023 -1 0021 -1 SUP: 0.31 CONF: 0.41
```

Converting it as human-readable:
Now, with the help of the SequenceManager, each sequence containing a minimum percentage of events of any normal mined sequence is analyzed and if it is not satisfying the mined sequence, all the events (and more relevant contents) of the current sequence are stored as an error pattern candidate.

PressButton – hasLabel= “Listado de transeúntes”

HttpRequest – hasRequestURL=/espigonvalencia/perfil.do

DbSelectEvent – hasSqlStatement=SELECT * from cod_pue where select in ()

DbErrorEvent – hasErrorText=ERROR: error syntax at or near ”)

HttpResponse

The sequence satisfies the first three events of the mined normal sequence, but there is an event in between that breaks the sequence.

The SequenceManager proposes the current sequence as an error pattern candidate, storing it to be proposed to the user.

3.3 Pattern metadata settings

In the previous step, the error pattern candidates are stored. Our objective is to abstract away from the concrete error pattern and try to generalize what we found. If the same type of error occurs again, maybe the concrete user input or the stack traces will be different, but the error is the same. Hence, we have to provide a mechanism to provide the pattern to further identify this kind of error when it happens again.

At this stage we could wonder if we can automate a process, but we consider that this task is too dependent on the user expertise, so the system proposes the pattern which identifies the broken normal sequence:
PressButton – hasLabel= “Listado de transeúntes”
HttpRequest – hasRequestURL=/espigonvalencia/perfil.do
DbSelectEvent – hasSqlStatement=SELECT * from cod_pue where select in ()
DbErrorEvent – hasErrorText=ERROR: error syntax at or near ”)"

Among the next steps, we are adding a user interface to propose this pattern to the user, so he can select all or part of the pattern. This feature is not part of this iteration of the prototype, but will be available in the second iteration. Coming from expertise, the user would understand that the error is produced because of the empty parenthesis in a SQL query, which produces a syntax error. Hence the user would keep the following pattern:

PressButton
HttpRequest
DbSelectEvent – hasSqlStatement=”()”
DbErrorEvent – hasErrorText=”error syntax at or near ”)”

The validated and improved pattern is internally converted (as an ontological representation of a sequence with property restrictions) will all the information:

PressButton AND
hasNext SOME(
    HttpRequest AND
    hasNext SOME(
        DbSelectEvent AND hasContents SOME “()” AND
        hasNext SOME(
            DbErrorEvent AND hasContents SOME “error syntax at or near \“)\””))))
Additionally, the user could add more information, like the information to be represented in a FastFix Error Report, once the error is detected.

If we send this information through OSGi services to the event correlation bundle, the system will react against the error, detecting it, taking advantage of the newly gained knowledge, as explained in the pattern mining section of D4.3e [1].
4 Running the prototype

Currently the FastFix platform is run as a set of OSGi components. The best way to test the components described in this document is in the context of the FastFix development environment.

4.1 Infrastructure for FastFix Development

The following software is required to develop FastFix components:

- Maven 3.0.3 (Homepage: [http://maven.apache.org/](http://maven.apache.org/))
- [http://m2eclipse.sonatype.org/sites/m2e/](http://m2eclipse.sonatype.org/sites/m2e/)
- M2Eclipse Maven Plug-In for Eclipse (Homepage: [http://m2eclipse.sonatype.org/](http://m2eclipse.sonatype.org/), Install via Updatesite [http://m2eclipse.sonatype.org/sites/m2e/](http://m2eclipse.sonatype.org/sites/m2e/); Help > Install New Software > Add... > Use Updatesite URL > Select all options)

4.2 Build and Run FastFix

4.2.1 Build

For each platform, client and server, there is a dedicated Maven Project called "parent". This is needed in order to build all required bundles for the common, the client or the server automatically with Maven. The common parent bundle is called `eu.fastfix.common.parent`. To build the common bundle

1. Right-click on the common parent project and select "Run As... -> Maven Install".
2. Wait for Maven to build the packages and generate the binaries and Manifest.MF files. As a result, you should get something similar to this:

3. Refresh your package explorer by pressing F5 or right-click on it and press "Refresh" in order to prevent "Out of sync" while running the bundles.
Perform analogous steps for the server bundles.

### 4.2.2 Run existing bundles

To run the current bundles:

4. Select one of the projects by right-clicking on it.
5. Choose "Run As -> Run Configurations...".
6. Double-Click on "OSGi Framework" item from the list placed left. This will create a new configuration file for running the bundles.
7. In the "Arguments tab", replace the proposed "Program arguments" with:
   
   
   
   1. `-os ${target.os} -ws ${target.ws} -arch ${target.arch} -nl ${target.nl} -console -clean`

8. Press "Run"
9. In the "Console" view, type "ss fastfix" and press enter in order to display all FastFix bundles. If any of the FastFix bundles has "Resolved" state, start it by typing "start xx" where "xx" is the number of the plugin which is not activated.

One only has to create a new configuration and specify arguments once. Later just hit on "Run as OSGi Framework".
5 Next steps

We have described the features of the current first iteration of the pattern mining prototype. To sum up, this bundle is able to take a list of all the gathered events during an execution of FastFix for a period of time and the pre-processing of this data in order to build a group of events that can be understood as part of the same sequence of actions (session). This structure is formatted as the algorithms expect, looking forward to start mining patterns.

The purpose of the first phase is to learn from the system, and the knowledge to learn is the expected behaviour of the application, the system and the runtime environment. When the pattern mining algorithms find the normal execution patterns, the system uses this information to look for unexpected behaviour, and it proposes error pattern candidates to the user, to allow adjusting them conveniently.

This is what is included for this first iteration, but we must define what should be the next steps and objectives for the second iteration of the prototype. As other FastFix second iterations, the next steps should be oriented to a full integration on FastFix as well as to provide other functional features that might be needed for the sake of the FastFix bundles that might need the recognized patterns.

With this spirit, these are the next steps identified as part of the second iteration prototype for pattern mining:

- Full integration with the context system. There is a need of a mechanism to build the input data of the pattern mining process in an integrated way (e.g. directly subscribing to the context bus). Additionally, this mechanism should manage the incremental addition of events in this input data, in order to provide a remarkable set of events to mine patterns efficiently, as well as controlling the size limit of this input data.

- Integration with the event correlation bundle. The event correlation bundle must be aware of the new validated patterns, adding these patterns as gained knowledge in the maintenance ontology. Hence, the ontology update procedure must be implemented through OSGi services. Additionally, the event correlation bundle must be tested with
this new pattern mining feature, in order to validate how it can detect new errors coming from mined patterns.

- As introduced in section 3.3, there is a need of a user interface to be provided to the user to adjust the setting of the error pattern candidates. Additionally, we should think about the best option to make that user interface available to the user.

- The pattern mining operation will also provide a remarkable number of false positives. Consequently, there is a need to provide a validation mechanism to ensure the minimization of these false positives, just keeping the relevant and real error patterns.

- Regarding the algorithm performance with large input data (large amount of input events), it must be further tested, since most of the evaluation has been done with a representative, but not huge, amount of events.

- Explore, test and combine other algorithms/techniques. At this point, we have evaluated the results provided by sequential pattern mining algorithms, like SPADE [11] and PrefixSpan [10]. Nevertheless, and also depending on the kind of patterns to be detected, other algorithms can provide better results in terms of performance and/or efficiency.
6 Bibliography


[12] J. Wang and J. Han “BIDE: Efficient Mining of Frequent Closed Sequences”