Trace analysis, a preliminary study

Contributors:

Rogerio Aparecido Da Silva	LIG	Author
Lydie du Bousquet	LIG	Author
Roland Groz	LIG	Reviewer
Tareq Sbai	LIG	Reviewer
Yves Ledru	LIG	Reviewer
Yoann Blein	LIG	Reviewer
Arnaud Clère	MinMaxMedical	Reviewer
Fabrice Bertrand	Blue Ortho	Reviewer

Partners:

Partly funded by:

Contents

1. Introduction

WP5 investigates the design of tools that exploit information from a corpus of traces.

When disposing of traces representing several executions, it is possible to analyze them as a whole set and report about the MCPS usages. This is a complementary approach to monitoring, where each trace is analyzed separately.
The objective of this work package is to offer several tools to present the different executions in a easy-to-understand way. We choose to propose a language that allows carrying out some queries, in order to select a set of traces or some data from a set of traces. This language is built on top of the DSL defined in WP1.

The contribution is based on similar works in the domain of Complex Event Processing (CEP). We use the convention of SQL-like languages such as ESQL in BeepBeep 3 [7] and Cayuga Query Language [4].

The language is based on two main operators. The first one aims at reducing a set of traces with the idea to keep only those that satisfy a property. Trace files are not modified during this operation. It is called “filtering” in the following. The second main operator aims at reducing the set of information inside each trace file. It is called “projection” in the following.

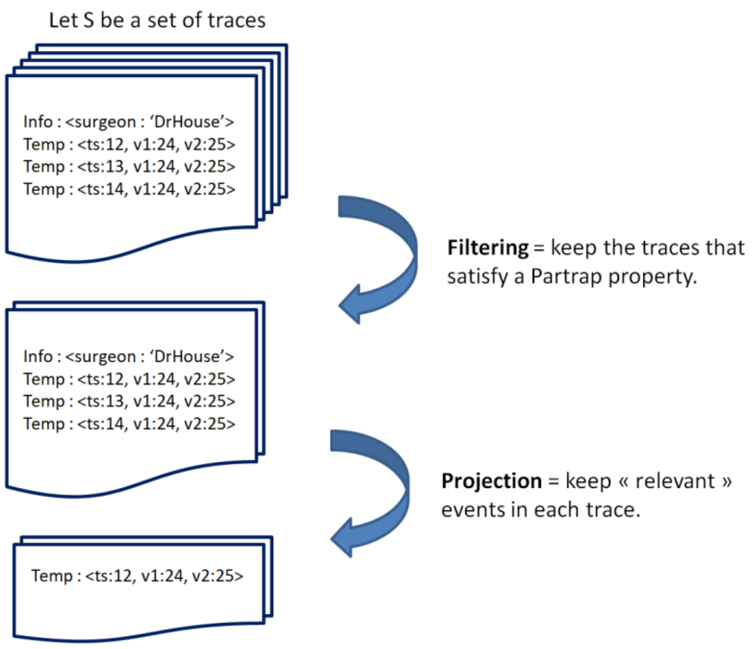
Let S be a set of traces
Info : <surgeon : 'DrHouse'>
Temp : <ts:12, v1:24, v2:25>
Temp : <ts:13, v1:24, v2:25>
Temp : <ts:14, v1:24, v2:25>
Filtering = keep the traces that satisfy a Partrap property.
Info : <surgeon : 'DrHouse'>
Temp : <ts:12, v1:24, v2:25>
Temp : <ts:13, v1:24, v2:25>
Temp : <ts:14, v1:24, v2:25>
Projection = keep « relevant » events in each trace.
Temp : <ts:12, v1:24, v2:25>

Fig. 1 a global view of the approach

2. Informal presentation of the language

2.1 Filtering

The first purpose of WP5 is to offer the possibility to select the traces that satisfy a property from a “repository”. The general syntax to do that is the following:

```
FROM <trace_set>
FILTER <Property>
```

Clause **FROM** defines from where the traces are taken.
Clause **FILTER** defines a property that is evaluated in order to decide whether a trace is selected or not. This property has to be expressed in the Partrap language.

Example 1. Select the traces where the surgeon is Dr. House.

```
FROM  trace_set
FILTER occurrence_of Info i where i.surgeon == 'Dr House'
```

The process of filtering consists in considering each trace belonging to the trace set defined in the From clause. By default, all events that are expressed in the Filter condition are supposed to belong to the trace under consideration.

Example 2. Select the traces where event `KneeCenter` occurs more than 10 times, between the first occurrence of the event `HipCenter` and the last occurrence of the event `KneeCenter`:

```
FROM trace_set
FILTER between HipCenter and Kneecenter, occurrence_of 10
KneeCenter
```

It is possible to express a property with the DSL, to name it and to use the name in the “**FILTER**” clause. It is also possible to name the result, in order to reuse it after.

Example 3. Select the traces where the condition `"Temp tp where v1 of temp > 42.5"` occurs more than one hundred times, after the first occurrence of the event `Temp`. The names of the set are recorded in a file named select_2.

```
Property_A = after first Temp,
             occurrence_of 100 Temp tp where tp.v1 > 42.5

select_2 =
   FROM trace_set
   FILTER Property_A
```

2.2 Projection

In the previous examples, the full trace is selected. Let us now introduce a way to select a part of a trace. This is done thanks to the clause **"EXTRACT"**. There are two ways of selecting a part of a trace: defining a portion of the trace to considerer and/or a set of events to keep.

Let us first introduce the clause **"INSCOPE"**. It is dedicated to the definition of the portion of the trace.

Example 4. Select all the events after first `KneeCenter`, for traces where property B is true.

```
Property_B = …

FROM trace_set
FILTER Property_B
EXTRACT *
INSCOPE after first Kneecenter
```

In clause **"EXTRACT"**, '*' is used to designate all the events. If a specific (set of) event(s) has to be chosen, it must to be listed explicitly.

Example 5. Select all the events `Temp` in each trace where `property_B` holds.

```
Property_B = …

FROM trace_set
FILTER Property_B
EXTRACT Temp
```

It is also possible to select events under conditions.

Example 6. Select all the events `Temp` that are less or equals to 25 after the beginning of the surgery (materialized by "start_acquisition" event), in each trace where `property_B` holds.

```
Property_B = …

FROM trace_set
FILTER Property_B
EXTRACT Temp t where t.v1<=25 or t.v2<=25 or t.v3<=25
INSCOPE after first start_acquisition
```

3. Exemples

```
TraceSet = {T1, T2, T3, T4}
T1: [D A A A B B A C A A A]
T2: [A B C D A B C D A B C]
T3: [A A B B B D D D A A B]
T4: [B A B A B A B A B A]

TS2 = FROM TraceSet FILTER occurrence_of D
TS2 = {T1, T2, T3}

TS3 = FROM TS2 EXTRACT C
TS2 = {[C], [C C C], []}
```

4. Syntax

Defined in the DSL

```
<scope> ::= ['within' <duration>] ('after' | 'before') ('each' | 'first' | 'last') <event>
          | 'between' <event> 'and' <event>
          | 'since' <event> 'until' <event>
<event> ::= …

E::=    FROM  ( PATH | Nested_From )
         [ FILTER <Prop>]
               [ EXTRACT <event_list_cond> | '*'
                      [INSCOPE <scope>]
               ]
        | <ident>
        | Let <ident> = <E> In < E >

Nested_From::=
        '(' FROM <E>
               [FILTER <Prop>]
                      [ EXTRACT <event_list_cond> | '*'
                             [INSCOPE <scope>]
        ] ')'
    | '(' Let <ident> = <E> In < E > ')'
```

- PATH is new keyword that indicate to the path of trace(s), it will get its value when the user chooses the folder of trace(s) in the beginning of running the program.
- '(' ')' is used in Nested_From rule to avoid conflict between outer and inner 'FROM', we got this notation from the convention of SQL.

5. References

[1] Blein, Yoann. WP1/ D1: Preliminary Definition of a Domain Specific Language. Version 0.10, 2016.

[2] Hallé, Sylvain. "From Complex Event Processing to Simple Event Processing." arXiv preprint arXiv:1702.08051 (2017).

[3] Hallé, Sylvain. "When RV Meets CEP." International Conference on Runtime Verification. Springer International Publishing, 2016.

[4] Demers, Alan J., et al. "Cayuga: A General Purpose Event Monitoring System." CIDR. Vol. 7. 2007.

[5] Shafranovich, Yakov. "Common format and MIME type for comma-separated values (CSV) files." (2005).

[6] Bray, Tim. "The javascript object notation (json) data interchange format." (2014).

[7] Sylvain Hallé, Raphaël Khoury "Event Stream Processing with BeepBeep 3." RV-CuBES 2017: 81-88