A framework for Automated Fault Identification

Béchir Ktari
Hashem Waly

Département d'informatique et de génie logiciel
Faculté des Sciences et de Génie
Université Laval, Québec, Canada

December 8, 2010
Montréal, Canada

Objectives

- Automating the detection of suspecious behaviors, performance degradation, and software bugs, in LTTng traces.
- Avoid to affect the performance of the system being analyzed.
- Integration within a software development environment (*Eclipse*).

Expected results

- An Eclipse plug-in editor for the definition of scenarios.
- An Eclipse plug-in engine that automatically detect faults in an abstracted version of LTTng traces.

Work progress

- Continuous reviews of the state of the art.
- Learning and exploring Eclipse plug-in framework (exchanges with Ericsson team).
- Definition of a framework for the detection of scenario-based properties :
  - Rigourous definition of a language.
  - Definition of typing rules.
  - Use of ANTLR parser generator.
  - Implementation of an editor for the definition of scenarios.
  - Implementation of a checker and a detection engine (GUI).
  - The developed plug-in is connected with Ericsson's one.
  - We propose an interface for building plug-ins on top of our framework.
- Rq: The developed framework is general as it could be used with any types of input traces. Furthermore, the proposed scenario language can be used with properties of different levels of abstraction. Hence, we anticipate to reuse this framework in the second phase of the project, i.e. high-level properties identification.

Outline

Outline

Characteristics

1. Core langage :
   - Filter predicates : The language is composed of atomic parts (predicates). The smallest predicate is the filter which filters a specific field in the event (*channel*, *process name*, etc). Users can specify several filters on the different fields of the event related using relational operators.
   - Event filters : Grouping filters into the event filter. Tagging the filter with an *id* so it can be referenced by other filters.
   - Scenarios : Combining event filters together into a scenario.
   - Scenarios as abstract events : Through the parameters of a scenario, it could be used as a *type* in other scenarios.
   - Group of scenarios : Combining scenarios into a group of scenarios.

Methodology (cont.)

2. Beyond the core language : The scenario and statement *options*, are a set of optional clauses used to give directions to the different modules of the system. We have implemented the following default options, but there is no constraint that the options could be extensible :
   - Statement *transition* : it contains four different values (default, consuming, non-consuming, and un-winding).
   - Priority : it represents the importance of the statement in the scenario being detected. It takes three values: low, medium, and high. It is currently used as part of the IDMEF messages.
   - Description : textual description of the scenario or the statement that is being detected. It is used as part of IDMEF messages.
   - Instances Limit : maximum number of created scenario instances.

Methodology (cont.)

3. Actions rules : The *action* clause, is used to define the appropriate action to be taken when a given group, scenario, or event is detected. User can define any external functions using the technique explained later, and use them as an action to a specific detection fact. Along with the external functions, we have implemented the following default functions:
   - *Display:* to display the detected events graphically in the user interface.
   - *IDMEF:* to create XML-based IDMEF messages, and send them to a remote host.
   - *Command:* to execute a given Linux commands on the shell, it could be useful for example to shut down the computer, or close a specific port in a response to a detected attack.

Outline

BNF of the proposed language

$$
\begin{array}{lll}
\textit{sfile} & ::= & \textit{include}^* \ \textit{ext}^* \ \textit{type}^* \ \textit{spec}? \ \textit{group}^* \ \textit{action}^* \\
\textit{include} & ::= & \textbf{include} \ \textit{file} \ ; \\
\textit{ext} & ::= & \textbf{extern} \ \tau \ f \ (\tau_1, \ldots, \tau_n) \ ; \\
\textit{group} & ::= & \textbf{group} \ g \ \{ \ \textit{spec} \ \} \\
\textit{spec} & ::= & \textit{var}^* \ \textit{pred}^* \ \textit{evtdef}^* \ \textit{scenario}^* \\
\textit{type} & ::= & \textbf{type} \ t = \tau \ ; \\
\textit{var} & ::= & \textbf{var} \ x = \textit{exp} (, y = \textit{exp})^* \ ; \\
\textit{pred} & ::= & \textbf{predicate} \ p \ (x_1, \ldots, x_n) \ \{ \ \textit{exp} \ \} \ ; \\
\textit{evtdef} & ::= & \textbf{eventdef} \ e \ \textbf{where} \ (\textit{exp}) \ ; \\
\textit{scenario} & ::= & \textbf{scenario} \ s \ [\textit{option}^*]? \ \ (x_1, \ldots, x_n) \ \textbf{as} \ (\textit{exp}_1, \ldots, \textit{exp}_n) \ \{ \ \textit{statement}^* \ \} \\
\textit{statement} & ::= & (!)? \ \textbf{event} \ e \ [\textit{option}^*]? \ \ (: \textit{etype})? \ \textbf{where} \ (\textit{exp})? \ ; \\
 & | & \textbf{within} \ [\textit{option}^*]? \ ((\textit{exp}_1, \textit{exp}_2) \mid (\textit{exp})) \ \{ \ \textit{statement}^* \ \} \\
 & | & \textbf{repeat} \ [\textit{option}^*]? \ (\textit{exp}) \ \{ \ \textit{statement}^* \ \} \\
\textit{option} & ::= & (o_1 = v_1, \ldots, o_n = v_n) \\
\textit{action} & ::= & \textbf{action} \ (\textit{exp}) \ \{ f \ (\textit{exp}_1, \ldots, \textit{exp}_n)^* \} \\
\textit{exp} & ::= & c \mid x \mid \textit{exp}.l \mid (\textit{exp}) \mid f \ (\textit{exp}_1, \ldots, \textit{exp}_n) \mid \textit{unop} \ \textit{exp} \mid \textit{exp}_1 \ \textit{binop} \ \textit{exp}_2 \\
\textit{unop} & ::= & ! \mid - \\
\textit{binop} & ::= & \textit{exp}_1 \ \textit{logop} \ \textit{exp}_2 \mid \textit{exp}_1 \ \textit{eqop} \ \textit{exp}_2 \mid \textit{exp}_1 \ \textit{relop} \ \textit{exp}_2 \mid \textit{exp}_1 \ \textit{arop} \ \textit{exp}_2 \\
 & | & \textit{exp}_1 + \textit{exp}_2 \\
\textit{logop} & ::= & \&\& \mid \| \\
\textit{eqop} & ::= & == \mid != \\
\textit{relop} & ::= & = \mid <= \mid > \mid < \\
\textit{arop} & ::= & - \mid * \mid /
\end{array}
$$

Outline

Main characteristics of the core language

- Filters :

```
(timestamp == 26126.7872610);
(channel == "kernel") && (pname == "chroot");
```

- Event filters :

```
event e where (timestamp < 26126 &&  pname == "firefox*" || channel == "kernel");
```

- Scenarios :

```
scenario test {
    event e1 where (channel == "fs" && type == "open" );
    event e2 where (channel == "fs" && type == "close" && pid == e1.pid);
}
```

Main characteristics of the core language (cont.)

- Scenarios :

```
scenario test {
    event e1 where (channel == "fs" && type == "open" );
    event e2 where (channel == "fs" && type == "close" && pid == e1.pid);
}
```

- Abreviations :

```
eventdef open where (channel == "fs" && type == "open" );
eventdef close where (channel == "fs" && type == "close");

scenario test {
    event e1 : open;
    event e2 : close where (pid == e1.pid);
}
```

Main characteristics of the core language (cont.)

- Scenarios as atomic proposition :

```
extern bool file(string s);

eventdef syscall_entry where (channel == "kernel" &&  type == "syscall_entry");
eventdef syscall_exit where (channel == "kernel" &&  type == "syscall_exit");

eventdef open_core where (channel == "fs" &&  type == "open");
eventdef chroot_core where (channel == "kernel" &&  type == "chroot");

scenario open (filename, pid, return_code) as
              (e1.content.filename, e3.pid, e3.content.return_code) {
    event e1 : syscall_entry where (e1.content.syscall_id == 5);
    event e2 : open_core where (e2.pid == e1.pid);
    event e3 : syscall_exit where (e3.pid == e2.pid && e3.content.return_code != -1);
}

scenario chroot_jail {
    event chroot1 : syscall_entry where (chroot1.content.syscall_id == 61);
    event chroot2 : chroot_core where (chroot2.pid == chroot1.pid);
    event chroot3 : open where (chroot3.pid == chroot2.pid && file(chroot3.filename));
}
```

Main characteristics of the core language (cont.)

- Other constructs :

```
eventdef socket_create where (channel == "net" &&  type == "socket_create");
eventdef socket_connect where (channel == "net" && type == "socket_connect");
eventdef socket_send where (channel == "net" && type == "socket_call" && content.call == 9);
eventdef socket_receive where (channel == "net" && type == "socket_call" &&
                               content.call == 10);

var timeout = 1.0;

scenario syn_flood  {
    event syn1 : socket_create;
    event syn2 : socket_connect where (syn2.pid == syn1.pid);
    event syn3 : socket_send where (syn3.pid == syn2.pid);
    within(timeout) {
        !event e : socket_receive;
    }
}
```

Main characteristics of the core language (cont.)

- Other constructs :

```
var threshold = 50;
var timeWindow = 2.0;

eventdef fs_write where (channel == "fs" && type == "write");
eventdef fs_access where (channel == "kernel" && type == "syscall_entry" &&
                          content.syscall_id == 33);

scenario inefficient_IO  {
    event e1 : open;
    within (timeWindow){
        repeat (threshold){
            event e : access_file;
        }
    }
}
```

Main characteristics of the core language (cont.)

- Other constructs :

```
var admFiles = [...];

scenario test {
    event e where (e.content.filename in admFiles);
}
```

Main characteristics of the core language (cont.)

- Group of scenarios :

```
include "sysFiles.scn";

group LocalPolicy {

    scenario s1 {
        ...
    }

    scenario s2 {
        ...
    }

    ...
}
```

Main characteristics of the advanced part

- Consuming and nonconsuming + attributes :

```
eventdef socket_create where (channel == "net" &&  type == "socket_create");
eventdef socket_connect where (channel == "net" && type == "socket_connect");
eventdef socket_send where (channel == "net" && type == "socket_call" && content.call == 9);
eventdef socket_receive where (channel == "net" && type == "socket_call" &&
                               content.call == 10);

var HIGH = 2, MEDIUM = 1, LOW = 0;

scenario syn_flood
[priority = MEDIUM; descr = "Syn␣flood␣attack"; iLimit = 5]
{
    event syn1[nc] : socket_create;
    event syn2[c] : socket_connect where (syn2.pid == syn1.pid);
    event syn3[c] : socket_send where (syn3.pid == syn2.pid);
    within(2.0) {
        !event[c] e : socket_receive;
    }
}
```

Main characteristics of the rule part

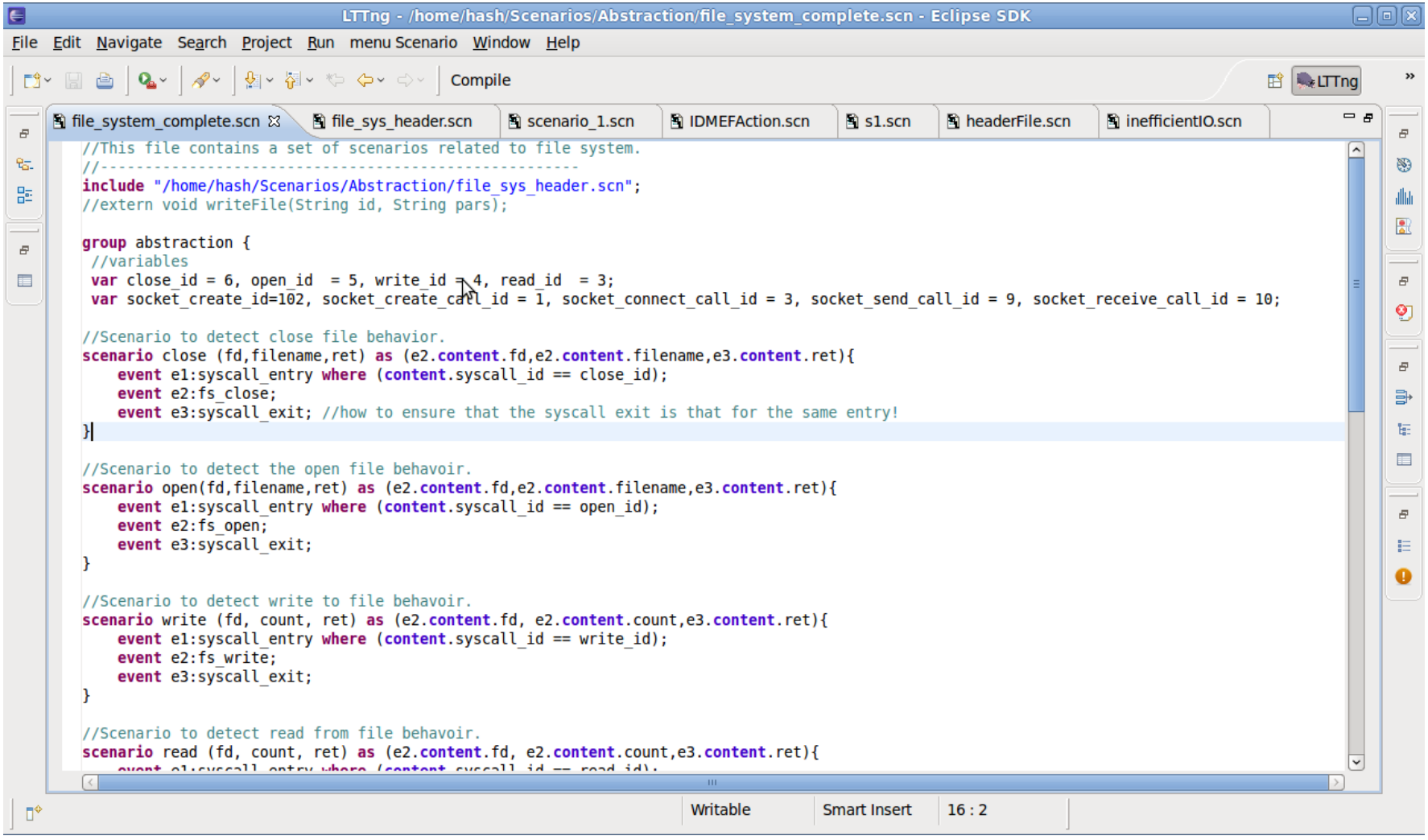
- Rules :

```
action (testing.inefficientIO && security.chroot_jail) {
    display (testing.inefficientIO.*);
}

action (security.syn_flood.$x) {
    idmef ($x.pid);
}

action (security.*.*) {
    command "mailto: ..."
}
```

Scenario Editor - An eclipse plugin



Outline

Type checker module

$$\tau \quad ::= \quad int \mid string \mid bool \mid time \mid \tau_1 \times \ldots \times \tau_n \rightarrow \tau \mid \{\, l_1 : \tau_1, \ldots, l_n : \tau_n \,\}$$

- Typing environments :
  - $\mathcal{E} = [x_1 : \tau_1, \ldots, x_n : \tau_n]$
  - $\mathit{TypeOf} = [\mathrm{INT} : \mathit{int}, \mathrm{IDENT} : \mathit{string}, \mathrm{TIME} : \mathit{time}]$
  - $\mathcal{E}'_0 = [\mathrm{lttng} : \{\mathrm{trace} : \mathit{string}, \mathrm{type} : \mathit{string}, \mathrm{channel} : \mathit{string}, \mathrm{cpu} : \mathit{int}, \mathrm{timestamp} : \mathit{time}\}]$
  - $\mathcal{E}_0 = \mathcal{E}'_0 \dagger (\mathit{flat}(\mathcal{E}'_0(\mathrm{lttng})))$
- Sequents :
  - $\mathcal{E} \vdash \mathit{exp} : \tau$
  - $\mathcal{E} \vdash o : \mathcal{E}'$

Type rules

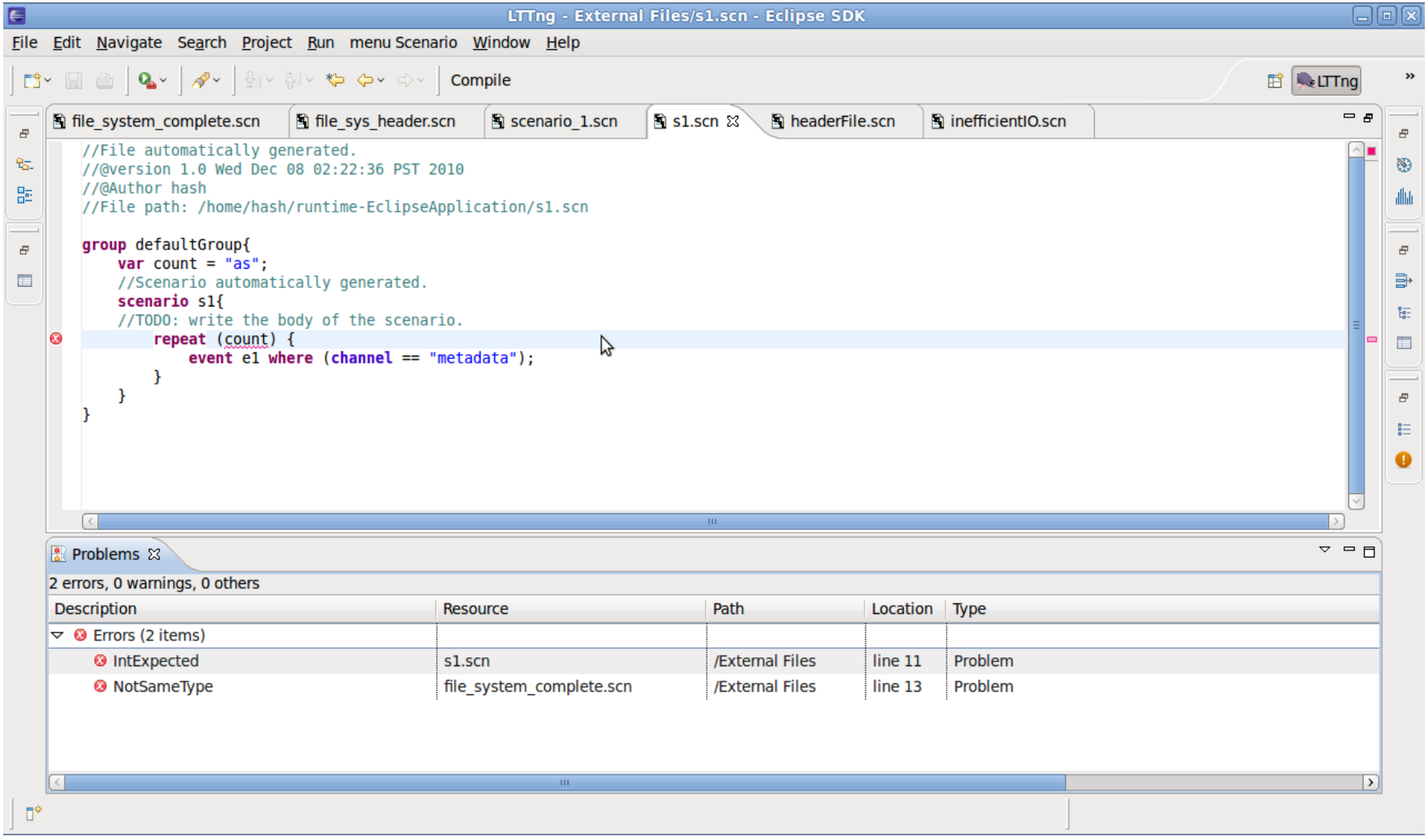
$$(\mathrm{evt}_1)\ \frac{\mathcal{E} \dagger [\mathit{flat}(\mathcal{E}(\mathit{etype}))] \vdash \mathit{exp} : \mathit{bool} \quad e \notin \mathit{dom}(\mathcal{E})}{\mathcal{E} \vdash (!)?\ \textbf{event}\ e\ (: \mathit{etype})?\ (\textbf{where}\ \mathit{exp})? : \mathcal{E} \dagger [e : \{\mathrm{timestamp} : \mathit{time}\} \oplus \mathcal{E}(\mathrm{etype})]}$$

$$(\mathrm{evt}_2)\ \frac{\mathcal{E} \vdash \mathit{exp} : \mathit{bool} \quad e \notin \mathit{dom}(\mathcal{E})}{\mathcal{E} \vdash (!)?\ \textbf{event}\ e\ (\textbf{where}\ \mathit{exp})? : \mathcal{E} \dagger [e : \mathcal{E}(\mathrm{lttng})]}$$

$$(\mathrm{scn})\ \frac{\mathcal{E} \vdash \mathit{evtfilter}^* : \mathcal{E}' \quad \mathcal{E}' \vdash \mathit{exp}_1 : \tau_1 \ \ldots\ \mathcal{E}' \vdash \mathit{exp}_n : \tau_n \quad s \notin \mathit{dom}(\mathcal{E})}{\mathcal{E} \vdash \textbf{scenario}\ s\ (x_1, \ldots, x_n)\ \textbf{as}\ (\mathit{exp}_1, \ldots, \mathit{exp}_n)\ \{\ \mathit{evtfilter}^*\ \} : \mathcal{E} \dagger [\, s : \{\, x_1 : \tau_1, \ldots, x_n : \tau_n \,\}\,]}$$

$$(\mathrm{edef})\ \frac{\mathcal{E} \vdash \mathit{exp} : \mathit{bool} \quad e \notin \mathit{dom}(\mathcal{E})}{\mathcal{E} \vdash \textbf{eventdef}\ e\ \textbf{where}\ \mathit{exp} : \mathcal{E} \dagger [\, e : \mathcal{E}(\mathit{lttng})\,]}$$

Scenario Editor - An eclipse plugin



Detection Engine

- The inputs to the detection engine are the *pre-treated* scenario instances and the standardized low-level events.
- There is a difference between scenario prototypes which is the original format of the given scenario, and the scenario instances which is the executable running instance of the scenario.
- For reasons of performance, the LTTng events are treated as block of events.
- For the evaluation of different expressions, a complete environment is used to keep the values of different statements detected as well as the scenario parameters.

Detection Engine : How it works?

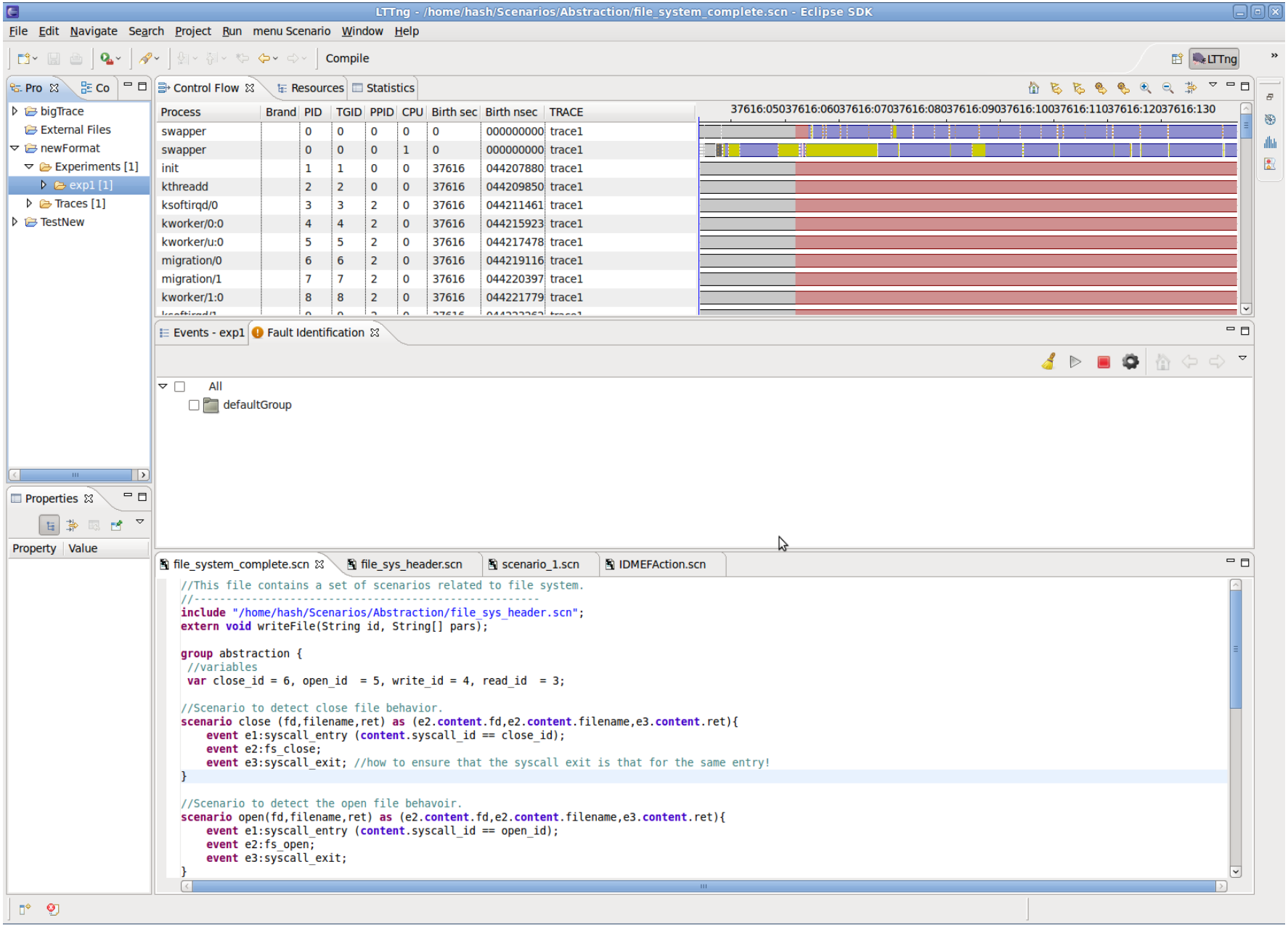
- Loop sequentially on all the events in the trace.
- Loop on all the scenario instances.
- Pass the event to the scenario instance :
  - If the expression in the statement is evaluated to true, the cursor associated with the scenario is incremented;
  - in that case, if there is an action associated with the statement, it is executed; otherwise, the event is displayed to the *GUI* or sent as an IDMEF message to a remote client.
- if all the statements in the scenario are evaluated to true :
  - the action associated with the scenario is executed;
  - if the last statement of the scenario is *non-consuming*, the scenario is deleted and removed from the array of scenarios; otherwise the cursor is initialized and the detection continues at the same scenario instance.

Detection Engine : Statement transitions

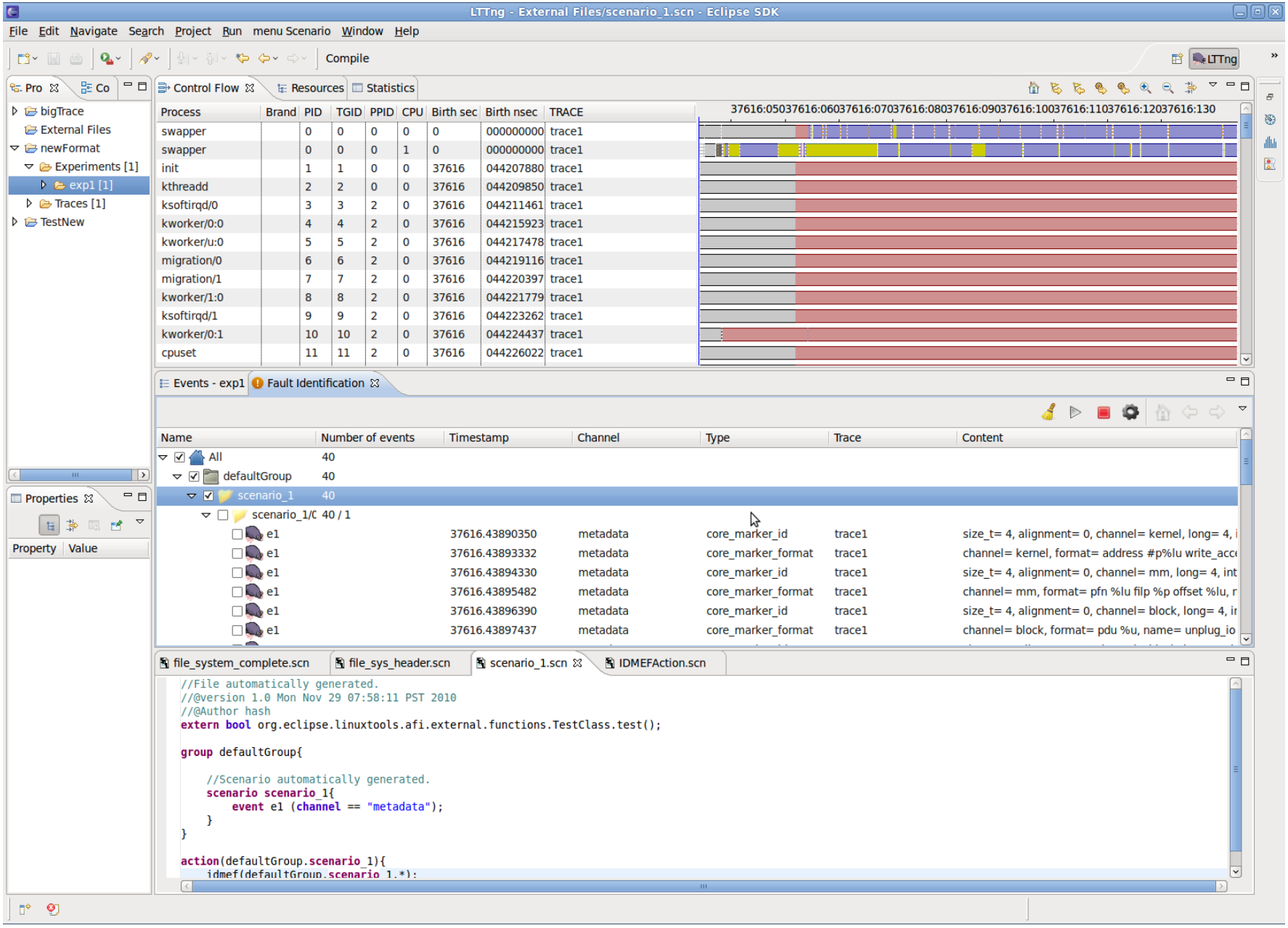
Statement transitions are given as options to the statements :

1. *Default:* The cursor is incremented, and if all the statements in the scenario are evaluated to true, there is no instance created, and the cursor is initialized to 0.
2. *Consuming transition:* the cursor is incremented in all scenario instances of the same scenario prototype. There is no scenario instance created. The cursor is incremented, and the scenario moves form the current statement to the next one.
3. *Non-Consuming:* The cursor is incremented, and another instance of the same scenario is created.
4. *Un-winding:* This is a rollback transition. The cursor of the scenario is initialized to the statement passed in the unwinding transition (for example: uw=e1); in this case the cursor is initialized to statement e1, and all the current instances of the same scenario are deleted.

Detection Engine : the *GUI*



Detection Engine : the *GUI*



Patterns

In the following, we are considering a set of undesired behaviours touching different fields like: *security*, *performance*, and *software bugs*.

More precisely, we present three examples of such properties :

1. Race conditions on files.
2. Inefficient I/O.
3. Excessive Swapping.

and propose, for each one, a specification using our language.

Outline

Race conditions on files

Race conditions is one example of *File Permissions* violations. It occurs when a system or a device assumes to perform two or more operations atomically while they are not. They are altered by external events that may be occasionally or explicitly executed by an attacker.

Race conditions on files (cont.)

```
if (access("/tmp/x", W_OK) == 0) {
    //attacker thread
    //unlink("/tmp/x");
    //attacker thread
    //symlink("/etc/passwd","/tmp/x");
    if((fd = open("/tmp/x", O_WRONLY)) == -1) {
       perror("/tmp/x");
       return(0);
    }
    //write to the file
}
```

Figure: Race conditions on files C code.

Race conditions on files (cont.)

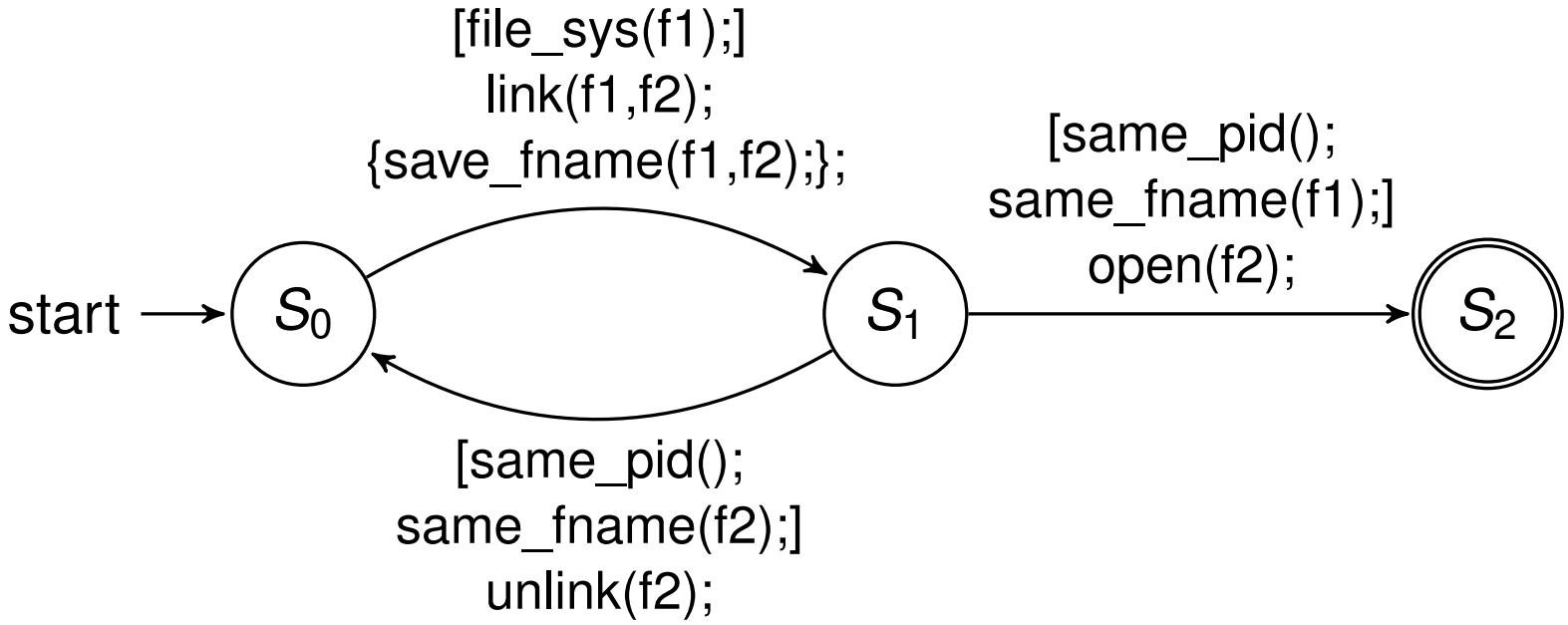


Figure: Race condition FSM.

Race conditions on files (cont.)

```
SYSCALL_DEFINE2(symlink,
  const char __user *, oldname,
  const char __user *, newname)
{
    ...
    trace_mark(kernel, syscall_link, "filename %s", newname);
    ...
}
```

Figure: Adding a marker in symlink system call.

Race conditions on files (cont.)

```
kernel.syscall_entry: 137305.5091 (./kernel_1), 7914, 7914,
  ./race_violation, , 31269, 0x0,
  SYSCALL ip = 0xb809a430,
  syscall_id = 83 [sys_symlink+0x0/0x30]

#The added marker
kernel.syscall_link: 137305.5101 (./kernel_1), 7914, 7914,
  ./race_violation, , 31269, 0x0,
  SYSCALL filename /etc/passwd

kernel.syscall_exit: 137305.5302 (./kernel_1), 7914, 7914,
  ./race_violation, , 31269,
  0x0, USER_MODE ret = 0
```

Figure: LTTng relevant link calls.

Race conditions on files script

```
extern bool file_sys (string file);

//variables definition.
var acc_file = ["read", "write", "exec"];

//Types definition.
eventdef syslink     where (channel == "kernel" && type == "link");
eventdef sysunlink   where (channel == "kernel" && type == "unlink");
eventdef acc_file    where (channel == "fs" && type in acc_file);

scenario sens_acc(){

    event e1:syslink [c] where (file_sys(content.filename));

    event e2:sysunlink[uw="e1"] where (content.filename == e1.content.file1);

    event e3:acc_file[nc] where (content.filename == e1.content.file1);
}

action (sens_acc) {
   display(sens_acc.*);
}
```

Outline

Inefficient I/O

One undesired property is having a single process written a big small chunks of data to the disk in a small interval of time.

Inefficient I/O (cont.)

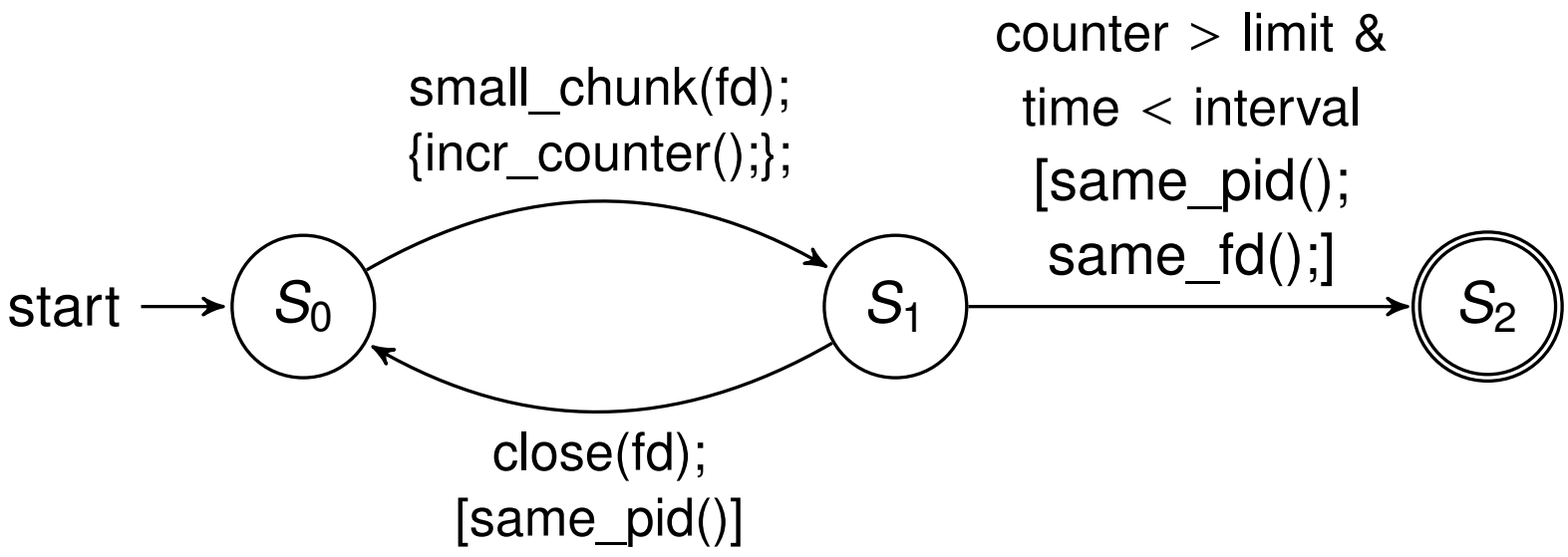


Figure: Writing small chunks of data to files.

Inefficient I/O script

```
//variables definition.
var timeout = 1000, count = 100, bytesCount = 10000;
var acc_file = ["write","read","exec"];

//Types definition.
eventdef file_open where (channel == "fs" && type == "open");
eventdef file_close where (channel == "fs" && type == "close");
eventdef file_acc    where (channel == "fs"  && type in acc_file);

scenario inefficient_io() {

    event e1:file_open [nc];

    event e2:file_close [uw=e1] content.fd == e1.content.fd);

    within(timeout){
        repeat(count){
            event e4:file_acc [c] where (content.count<bytesCount);
        }
    }
}

action (inefficient_io) {
   display(inefficient_io.*);
}
```

Outline

Excessive Swapping

Swapping operation is costly in terms of performance. Performing many swap operations in a small interval of time could cause "*excessive swapping*".

Excessive Swapping (cont.)

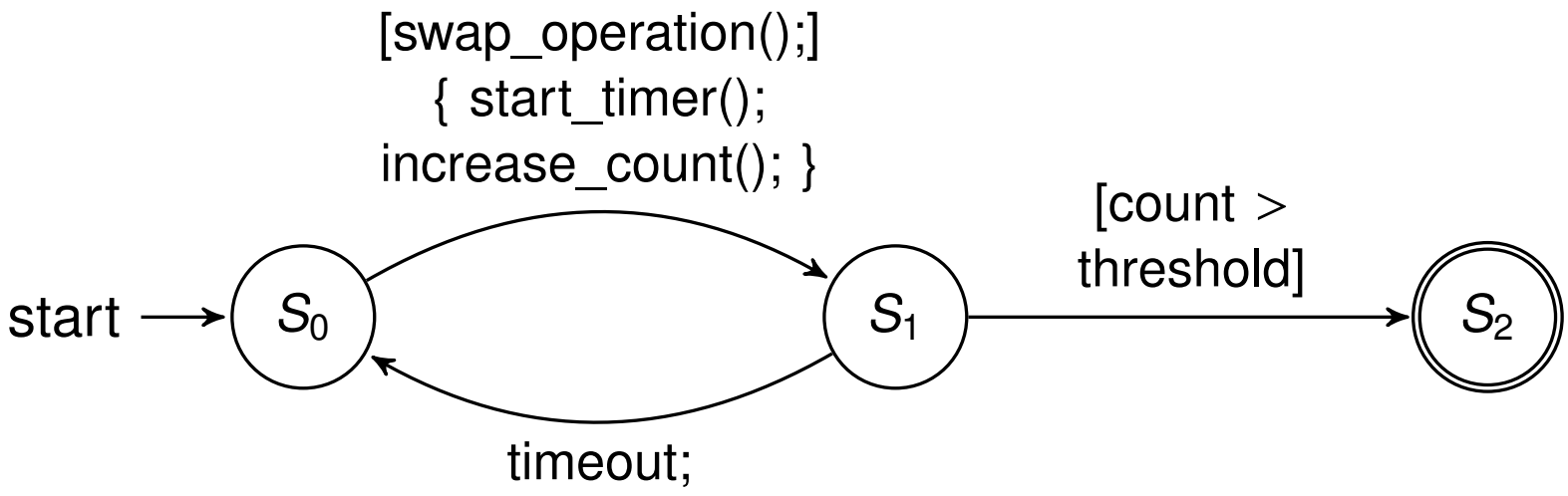


Figure: Excessive swapping FSM.

Excessive Swapping script

```
//Variables definition.
var swap_operations = ["swap_in", "swap_out"];
var allowedSwaps = 10000, timeInterval = 500;

//Types definition.
eventdef swap where (channel == "mm" & type in swap_operations);

scenario excessive_swapping {

    within(timeInterval) {
        repeat(allowedSwaps) {
            event e : swap;
        }
    }
}

action (excessive_swapping) {
   display(excessive_swapping.*);
}
```

Conclusion

1. We have presented a new *declarative scenario description language*. The declarative approach of the language is simple to use because the users are not concerned about implementation details; they are only concerned about what to be detected (events, scenarios, etc).
2. We assume that the proposed language supports the most important features needed in a scenario description language like : scenarios based on multiple events, real-time constraints, counting, knowledge acquisition (as scenarios could be used to defined other scenarios), variables definition, etc.
3. We have to consider that the presented framework is a first step toward more efficient tools for the automatic detection of problematic behaviors.
   In fact, we consider the output of our framework as what are called in the literature *low-level observation* or *facts*. On top of that framework, we can add more analysis based on such observations (*detected scenarios*).

Future Work

1. Continue the implementation of the detection engine and measure its performance against large and complex patterns.
2. Backward detection: it will be interesting if we can move forward and backward in the traces : If an interesting event is detected, move backward to see the possible reasons for the given fact.
3. Enhance the performance of the engine by adding more optimization to the scenario pre-processing or by parallelizing the detection of scenarios.
4. Synchronize with other projects, mainly *System Health project* and *Trace Abstraction project*.
5. Add to our framework streaming and live reading of a trace features.

Thank you

Questions?