1 Introduction and Background

During the execution of software, execution data can be recorded. With the development of process mining techniques on the one hand, and the growing availability of software execution data on the other hand, a new form of software analytics comes into reach. That is, applying process mining techniques to analyze software execution data. To enable process mining for software, event logs should be capable of capturing software-specific data.

A software event log is typically recorded at the method call level during software execution. Events generated at this level reference a specific point in the software source code. The Software Event extension captures this event location information, together with some basic runtime information related to this location.

Extension definition

- **Name:** Software Event
- **Prefix:** swevent
- **URI:** http://www.xes-standard.org/swevent.xesext
- **XML:**

```
```

The remainder of this extension is organized as follows. In Section 2 we explain some basic terminology. In Sections 3 and 4 we detail how a method execution is recorded. In addition, Section 5 covers how to record data associated with a method execution. Via Section 6 we allow the user to describe the executed software, making the event log more self-contained. Section 7 provides minimal support for runtime information, and Section 8 covers exception handling data. In Section 9 we touch upon the relation with the micro event extension. Section 10 provides an example, and the XES Extension definition is given in Section 11. Finally, in Section 12 provides a reference glossary.
2 Terminology

In this extension, we will use some software-specific terminology. We provided a reference glossary in Section 12. In this section, we will explain some key terminology using the example code snippet shown in Listing 1.

Listing 1: Example code snippet showing when which event types are triggered.

```java
1 class A {
2     void f(int y) {
3         try {
4             int r = b.g(12, y);
5         } catch (Exception e) {
6             ...
7         }
8     }
9 }
10 class B {
11     int g(int x, int y) {
12         return x / y;
13     }
14 }
```

When recording a software event at the method call level, we refer to the method being called or invoked as the callee. Optionally, we can track the context method where a method is triggered from. We call this context method the caller. That is, the caller method invokes the callee method. In Listing 1, we can see how the callee and caller role can change over the course of a software execution:

<table>
<thead>
<tr>
<th>Line</th>
<th>Callee</th>
<th>Caller</th>
<th>Line</th>
<th>Callee</th>
<th>Caller</th>
<th>Line</th>
<th>Callee</th>
<th>Caller</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>A.f</td>
<td>–</td>
<td>4</td>
<td>B.g</td>
<td>A.f</td>
<td>13</td>
<td>B.g</td>
<td>–</td>
</tr>
<tr>
<td>10</td>
<td>A.f</td>
<td>–</td>
<td>6</td>
<td>B.g</td>
<td>A.f</td>
<td>15</td>
<td>B.g</td>
<td>–</td>
</tr>
</tbody>
</table>

3 Event Type and Lifecycle

Software events can be triggered at different states during a software execution. To indicate when an event was triggered, we record an indicative event type attribute, as defined below. In Listing 1, these event types are shown on the right in the context of an example code snippet.

<table>
<thead>
<tr>
<th>Level</th>
<th>Key</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>type</td>
<td>string</td>
<td>Software event type, indicating at what point during execution this event was generated. The possible values are enumerated below.</td>
</tr>
</tbody>
</table>
The possible software event type values we recognize are:

- **call**: The start of a method block
- **return**: The normal end of a method block
- **throws**: The end of a method block in case of an uncaught exception
- **handle**: The start of an exception handle catch block
- **calling**: The start of calling / invoking another method
- **returning**: The end of returning a called method

The software event type values describe transitions in a transactional model for the lifecycle of a method execution. In Figure 1, the state machine for the method execution transactional model is given. As an example, these are the states for the two methods in Listing 1:

<table>
<thead>
<tr>
<th>Line</th>
<th>Event</th>
<th>Type</th>
<th>A.f</th>
<th>B.g</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>call</td>
<td>A.f</td>
<td>InProcess</td>
<td>–</td>
</tr>
<tr>
<td>4</td>
<td>calling</td>
<td>B.g</td>
<td>Suspended</td>
<td>–</td>
</tr>
<tr>
<td>13</td>
<td>call</td>
<td>B.g</td>
<td>Suspended</td>
<td>InProcess</td>
</tr>
<tr>
<td>15</td>
<td>return</td>
<td>B.g</td>
<td>Suspended</td>
<td>Completed</td>
</tr>
<tr>
<td>15</td>
<td>throws</td>
<td>B.g</td>
<td>Suspended</td>
<td>Aborted</td>
</tr>
<tr>
<td>10</td>
<td>returning</td>
<td>B.g</td>
<td>InProcess</td>
<td>Completed</td>
</tr>
<tr>
<td>7</td>
<td>handle in A.f</td>
<td></td>
<td>InProcess</td>
<td>Aborted</td>
</tr>
<tr>
<td>10</td>
<td>return</td>
<td>A.f</td>
<td>Completed</td>
<td>Closed</td>
</tr>
<tr>
<td>10</td>
<td>throws</td>
<td>A.f</td>
<td>Aborted</td>
<td>Closed</td>
</tr>
</tbody>
</table>

Figure 1: State machine for the method execution transactional model.

The software event type values can be related to values in the standard lifecycle transactional model, defined in the lifecycle extension. We suggest the following mapping (note that the standard lifecycle transactional model cannot correctly support the **handle** transition):

<table>
<thead>
<tr>
<th>Type</th>
<th>Lifecycle</th>
<th>Type</th>
<th>Lifecycle</th>
<th>Type</th>
<th>Lifecycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>call</td>
<td>start</td>
<td>calling</td>
<td>start</td>
<td>handle</td>
<td>reassign</td>
</tr>
<tr>
<td>return</td>
<td>complete</td>
<td>returning</td>
<td>complete</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4 Event Location

Software events are triggered at a particular, uniquely identifiable location in the software source code. With this location, each event is traceable back to the source code location where it was generated.

Each event has a callee location. Events with the calling or returning event type also have an additional caller location. Each location is described by the following attributes, with a callee- or caller-key prefix:

<table>
<thead>
<tr>
<th>Level Key</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event</td>
<td>package</td>
<td>The package in the software code architecture to which the method belongs.</td>
</tr>
<tr>
<td>event</td>
<td>class</td>
<td>The class to which the method belongs.</td>
</tr>
<tr>
<td>event</td>
<td>method</td>
<td>The referenced method.</td>
</tr>
<tr>
<td>event</td>
<td>returnSig</td>
<td>The return signature of the referenced method.</td>
</tr>
<tr>
<td>event</td>
<td>isConstructor</td>
<td>If the referenced method is a class constructor.</td>
</tr>
<tr>
<td>event</td>
<td>instanceId</td>
<td>The instance id of the corresponding class instance. The absence of an instance id is represented by the value “0”.</td>
</tr>
<tr>
<td>event</td>
<td>filename</td>
<td>The file name of the corresponding source code artifact.</td>
</tr>
<tr>
<td>event</td>
<td>lineNr</td>
<td>The line number of the executed source code statement.</td>
</tr>
</tbody>
</table>

The software event callee values can be related to the concept name, defined in the concept extension. We suggest the use the concatenation of callee package, class, method and paramSig as a concept:name (e.g., demo.A.f(int)). This way, the concept name is the unique, canonical name for the executed method.

Example event location information for some events from Listing 1:

<table>
<thead>
<tr>
<th>callee-package</th>
<th>Line 2</th>
<th>caller-package</th>
<th>Line 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>demo</td>
<td>demo</td>
<td>demo</td>
<td>demo</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>f</td>
<td>g</td>
<td>f</td>
<td>g</td>
</tr>
<tr>
<td>(int)</td>
<td>(int,int)</td>
<td>(int)</td>
<td>(int,int)</td>
</tr>
<tr>
<td>void</td>
<td>int</td>
<td>void</td>
<td>int</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>1074593562</td>
<td>660017404</td>
<td>1074593562</td>
<td>660017404</td>
</tr>
<tr>
<td>source/A.java</td>
<td>source/B.java</td>
<td>source/A.java</td>
<td>source/B.java</td>
</tr>
<tr>
<td>2</td>
<td>13</td>
<td>2</td>
<td>13</td>
</tr>
<tr>
<td>demo.A.f(int)</td>
<td>demo.B.g(int,int)</td>
<td>demo.A.f(int)</td>
<td>demo.B.g(int,int)</td>
</tr>
</tbody>
</table>

5 Method Data

At runtime, when a method is called or returns, data can be passed along. A return or returning event can define a return data value. A call or calling
event can define multiple parameter data values.

<table>
<thead>
<tr>
<th>Level</th>
<th>Key</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>log</td>
<td>hasData</td>
<td>boolean</td>
<td>If method data is recorded for this log.</td>
</tr>
<tr>
<td>event</td>
<td>returnValue</td>
<td>string</td>
<td>The return value for the returning method.</td>
</tr>
<tr>
<td>meta</td>
<td>params</td>
<td>list</td>
<td>List of parameters for the called method.</td>
</tr>
<tr>
<td>meta</td>
<td>paramValue</td>
<td>string</td>
<td>A parameter value in the list params.</td>
</tr>
<tr>
<td>meta</td>
<td>valueType</td>
<td>string</td>
<td>The runtime value type for a return or parameter value.</td>
</tr>
</tbody>
</table>

6 Application Information

At runtime, the software events are generated by a particular application or process instance. Users can annotate events with application information, indicating which application instance generated the events. Not only makes this the event log more self-contained, it also can help in tracing events in a distributed or multi process setup.

<table>
<thead>
<tr>
<th>Level</th>
<th>Key</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event</td>
<td>appName</td>
<td>string</td>
<td>The user defined application name.</td>
</tr>
<tr>
<td>event</td>
<td>appTier</td>
<td>string</td>
<td>The user defined application tier.</td>
</tr>
<tr>
<td>event</td>
<td>appNode</td>
<td>string</td>
<td>The user defined application node.</td>
</tr>
<tr>
<td>event</td>
<td>appSession</td>
<td>string</td>
<td>The user defined application session.</td>
</tr>
</tbody>
</table>

7 Runtime Information

At runtime, the software events in an application process are triggered on a particular thread. Such a thread can be identified by its thread id.

The nano time attribute is used to measure elapsed time in software. It does not have to be related to any other notion of system or wall-clock time. The value represents nanoseconds since some fixed but arbitrary time (perhaps in the future, so values may be negative).

<table>
<thead>
<tr>
<th>Level</th>
<th>Key</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event</td>
<td>threadId</td>
<td>string</td>
<td>The thread id on which the event was generated.</td>
</tr>
<tr>
<td>event</td>
<td>nanotime</td>
<td>int</td>
<td>The elapsed nano time since some fixed but arbitrary time (see description above.)</td>
</tr>
</tbody>
</table>
8 Exception Information

An exception in software indicates an error. When an exception is thrown from a particular location / method, it can be caught / handled at a particular location.

<table>
<thead>
<tr>
<th>Level</th>
<th>Key</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>log</td>
<td>hasException</td>
<td>boolean</td>
<td>If exception data is recorded for this log.</td>
</tr>
<tr>
<td>event</td>
<td>exThrown</td>
<td>string</td>
<td>The <em>thrown exception type</em> for a <em>throws</em> or <em>handle</em> event.</td>
</tr>
<tr>
<td>event</td>
<td>exCaught</td>
<td>string</td>
<td>The <em>caught exception type</em> for a <em>handle</em> event.</td>
</tr>
</tbody>
</table>

9 Relation to the Micro Event Extension

Note that the events software events generated on the same execution thread implicitly describe a call graph structure. For example in Listing 1, the event calling B.g on line 4 happens during the execution of A.f. That is, the event calling B.g on line 4 is a nested event of the events call A.f on line 2 and return A.f on line 10. This nested call graph relation can be explicitly logged using the *micro event* extension.

10 Example

In Listing 2 an example XES trace is given for the code snippet in Listing 1, called with f(0). In Figure 2, a message sequence diagram is given as a visual aid. We have annotated each event in Listing 2 with the corresponding even type, similar to the annotations in Listing 1. Note that, in this example, we assume that Line 14 in Listing 1 throws an ArithmeticException when y equals zero. This example log consists of 1 trace with 6 events.

![Figure 2: Message sequence diagram for the events in Listing 2](image-url)
Listing 2: Example XES log for the code snippet in Listing 1, called with f(0).

```xml
<log>
  <extension name="Concept" prefix="concept" uri="http://www.xes-standard.org/concept.xesext"/>
  <extension name="Lifecycle" prefix="lifecycle" uri="http://www.xes-standard.org/lifecycle.xesext"/>
  <extension name="Time" prefix="time" uri="http://www.xes-standard.org/time.xesext"/>

  <boolean key="swevent:hasData" value="true" />
  <boolean key="swevent:hasException" value="true" />

  <trace>
    <event>
      – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – –
      call A.f
      – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – –
      <string key="concept:name" value="demo.A.f(int)" />
      <string key="lifecycle:transition" value="start" />
      <date key="time:timestamp" value="2017-06-15T10:02:30.287Z" />
      <string key="swevent:type" value="call" />
      <string key="swevent:callee-package" value="demo" />
      <string key="swevent:callee-class" value="A" />
      <string key="swevent:callee-method" value="f" />
      <string key="swevent:callee-paramSig" value="(int)" />
      <string key="swevent:callee-returnSig" value="void" />
      <boolean key="swevent:callee-isConstructor" value="false" />
      <string key="swevent:callee-instanceId" value="1074593562" />
      <string key="swevent:callee-filename" value="source/A.java" />
      <int key="swevent:callee-lineNr" value="2" />
      <list key="swevent:params"> <values>
        <string key="swevent:paramValue" value="0">
          <string key="swevent:valueType" value="int" />
        </string>
        <string key="swevent:paramValue" value="12">
          <string key="swevent:valueType" value="int" />
        </string>
      </values> </list>
      <string key="swevent:appName" value="Demo" />
      <string key="swevent:threadId" value="1" />
      <int key="swevent:nanotime" value="493674332622147" />
    </event>
    <event>
      – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – –
      calling B.g
      – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – –
      <string key="concept:name" value="demo.B.g(int,int)" />
      <string key="lifecycle:transition" value="start" />
      <date key="time:timestamp" value="2017-06-15T10:02:30.287Z" />
      <string key="swevent:type" value="calling" />
      <string key="swevent:callee-package" value="demo" />
      <string key="swevent:callee-class" value="B" />
      <string key="swevent:callee-method" value="g" />
      <string key="swevent:callee-paramSig" value="(int,int)" />
      <string key="swevent:callee-returnSig" value="int" />
      <boolean key="swevent:callee-isConstructor" value="false" />
      <string key="swevent:callee-instanceId" value="660017404" />
      <string key="swevent:callee-filename" value="source/B.java" />
      <int key="swevent:callee-lineNr" value="13" />
      <string key="swevent:caller-package" value="demo" />
      <string key="swevent:caller-class" value="A" />
      <string key="swevent:caller-method" value="f" />
      <string key="swevent:caller-paramSig" value="(int)" />
      <string key="swevent:caller-returnSig" value="void" />
      <boolean key="swevent:caller-isConstructor" value="false" />
      <string key="swevent:caller-instanceId" value="1074593562" />
      <string key="swevent:caller-filename" value="source/A.java" />
      <int key="swevent:caller-lineNr" value="5" />
      <list key="swevent:params"> <values>
        <string key="swevent:paramValue" value="12">
          <string key="swevent:valueType" value="int" />
        </string>
        <string key="swevent:paramValue" value="0">
          <string key="swevent:valueType" value="int" />
        </string>
      </values> </list>
      <string key="swevent:appName" value="Demo" />
    </event>
  </trace>
</log>
```
<string key="swevent:appName" value="Demo" />
<string key="swevent:threadId" value="1" />
<int key="swevent:nanotime" value="4936743328738" />

<event> ---------------------------------------- call B.g </event>
<string key="concept:name" value="demo.B.g(int,int)" />
<string key="lifecycle:transition" value="start" />
<date key="time:timestamp" value="2017-06-15T10:02:30.287Z" />
<string key="swevent:type" value="call" />
<string key="swevent:callee-package" value="demo" />
<string key="swevent:callee-class" value="B" />
<string key="swevent:callee-method" value="g" />
<string key="swevent:callee-paramSig" value="(int,int)" />
<string key="swevent:callee-returnSig" value="int" />
<boolean key="swevent:callee-isConstructor" value="false" />
<string key="swevent:callee-instanceId" value="1074593562" />
<string key="swevent:callee-filename" value="source/B.java" />
<int key="swevent:callee-lineNr" value="13" />
<list key="swevent:params"> <values>
<string key="swevent:paramValue" value="12">
<string key="swevent:valueType" value="int" />
</string>
<string key="swevent:paramValue" value="0">
<string key="swevent:valueType" value="int" />
</string>
</values> </list>
<string key="swevent:appName" value="Demo" />
<string key="swevent:threadId" value="1" />
<int key="swevent:nanotime" value="493674332936044" />
<string key="swevent:exThrown" value="ArithmeticException" />
</event>

<event> ---------------------------------------- throws B.g </event>
<string key="concept:name" value="demo.B.g(int,int)" />
<string key="lifecycle:transition" value="ate_abort" />
<date key="time:timestamp" value="2017-06-15T10:02:30.287Z" />
<string key="swevent:type" value="throws" />
<string key="swevent:callee-package" value="demo" />
<string key="swevent:callee-class" value="B" />
<string key="swevent:callee-method" value="g" />
<string key="swevent:callee-paramSig" value="(int,int)" />
<string key="swevent:callee-returnSig" value="int" />
<boolean key="swevent:callee-isConstructor" value="false" />
<string key="swevent:callee-instanceId" value="1074593562" />
<string key="swevent:callee-filename" value="source/B.java" />
<int key="swevent:callee-lineNr" value="15" />
<string key="swevent:appName" value="Demo" />
<string key="swevent:threadId" value="1" />
<int key="swevent:nanotime" value="493674332936044" />
<string key="swevent:exCaught" value="Exception" />
</event>

<event> ---------------------------------------- handle in A.f </event>
<string key="concept:name" value="demo.A.f(int)" />
<string key="lifecycle:transition" value="reassign" />
<date key="time:timestamp" value="2017-06-15T10:02:30.287Z" />
<string key="swevent:type" value="handle" />
<string key="swevent:callee-package" value="demo" />
<string key="swevent:callee-class" value="A" />
<string key="swevent:callee-method" value="f" />
<string key="swevent:callee-paramSig" value="(int)" />
<boolean key="swevent:callee-isConstructor" value="false" />
<string key="swevent:callee-instanceId" value="1074593562" />
<string key="swevent:callee-filename" value="source/A.java" />
<int key="swevent:callee-lineNr" value="7" />
<string key="swevent:appName" value="Demo" />
<string key="swevent:threadId" value="1" />
<int key="swevent:nanotime" value="493674332936044" />
<string key="swevent:exThrown" value="ArithmeticException" />
</event>
11 XES Extension

Listing 3: XES Extension - Software Event.

<extension name="Software Event" prefix="swevent"
uri="http://www.xes-standard.org/swevent.xesext">

<log>
  
  <boolean key="hasData">
    <alias mapping="EN" name="Has method data"/>
  </boolean>
  
  <boolean key="hasException">
    <alias mapping="EN" name="Has exception data"/>
  </boolean>

</log>

<event>
  
  <string key="type">
    <alias mapping="EN" name="Event Type"/>
  </string>
  
  <string key="callee-package">
    <alias mapping="EN" name="Callee - Package"/>
  </string>
  
  <string key="callee-class">
    <alias mapping="EN" name="Callee - Class"/>
  </string>
  
  <string key="callee-method">
    <alias mapping="EN" name="Callee - Method"/>
  </string>
  
  <string key="callee-paramSig">
    <alias mapping="EN" name="Callee - Parameter signature"/>
  </string>
  
  <string key="callee-returnSig">
    <alias mapping="EN" name="Callee - Return signature"/>
  </string>
  
  <boolean key="callee-isConstructor">
    <alias mapping="EN" name="Callee - Is a class constructor"/>
  </boolean>
  
  <string key="callee-instanceId">
    <alias mapping="EN" name="Callee - Instance id of class instance"/>
  </string>

</event>

</log>
12 Glossary

callees The callee is the method being called or invoked.
caller The caller is the context/method where a method is triggered from.

exception An exception in software indicates an error.
throw An exception can be thrown from a particular location / method, signaling an error.
handle An exception can be caught / handled at a particular location, indicating we are handling an error.

class/object A class / object is a conceptual unit in software, defining a combination of methods and data.

(object) instance For each class, at runtime, multiple instances can exist, be created and destroyed. Each instance has its own lifecycle during the execution of a software program.