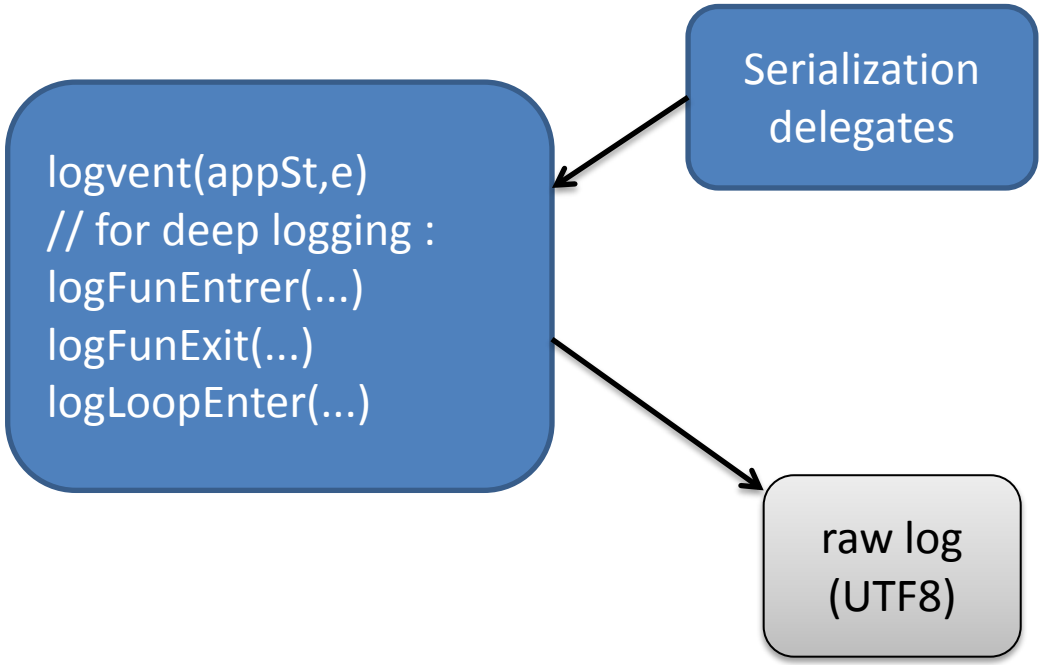
Logging Utils (AS)

- Provides a set of logging functions; call them to log something.



To log an object o, LogUtils relies on a "serialization scheme"

1. It works like this:
    1. If o implements Serializable, then it has a method serializeSelf.
    2. Else check delegates mapping, which is a map of this type:

Class of o → (Object → void)

(a function that given an object knows how to serialize it)

2. There is a separation between:
    1. Specifying projection
    2. Formatting

Example of class implementing Serializable

```
class Pair implements Serializable {
    public var fst : * ;
    public var snd : * ;

    ....
   public function serializeSelf(s : Serializer) : void {
          s.beginObject(this,Object(MYTYPE)) ;
          s.storeField(Object(FST),fst) ;
          s.storeField(Object(SND),snd) ;
          s.endObject() ;
     }
}
```

a Serializer will take care for the formatting.

this function specifies how to make the projection

Example of serialization delegate

```
function collectionSerializationDelegateFunction(c : Object, s : Serializer) : void {
    s.beginObject(c,COLL_TY_NAME) ;
    for each (var o:* in c)  s.storeField("elem",o) ;
    s.endObject() ;
}
```

Automation framework

- Listens to flash events, decide which ones are worth “recording”.
- On a relevant event e, it will produce a “RECORD” event, that will in turn contain pointers to:
  - e (actually, something else representing it)
  - its parameters
  - its target (e.g. which button e interacts on).
- Logging is attached simply by adding a handler to this RECORD event, and this handler will call “logEvent(...)” from LogUtils, passing to it:
  - the RECORD event itself (which in turn contain the above infos)
  - a projection of application state. Someone has to provide this projection function ---the logger cannot invent one on its own.

Automation Delegates

- To determine which events to monitor, the automation framework scans all GUI elements (called “Display Objects”) of the application.
- It is also aware when at the runtime more DOs are added, or removed.
- For each DO x, it checks a map of type:

  Class of x → AutomationDelegate

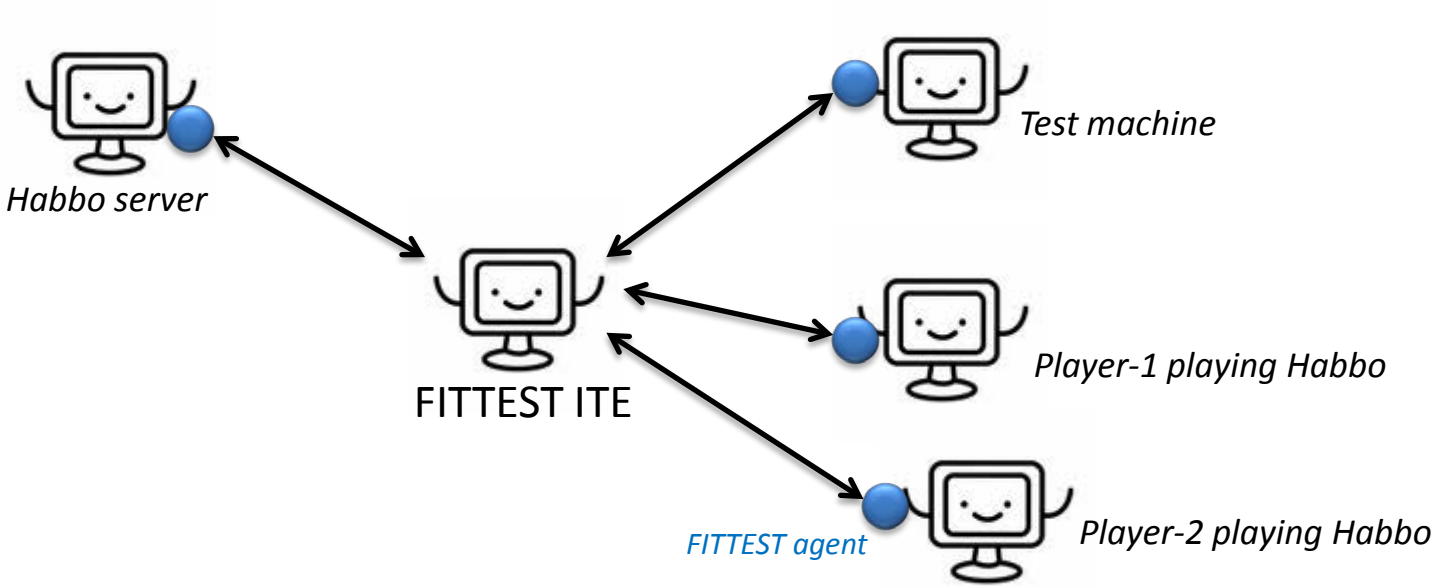
  If a delegate exists, then x will be logged; the delegate specifies which events on x will be logged.

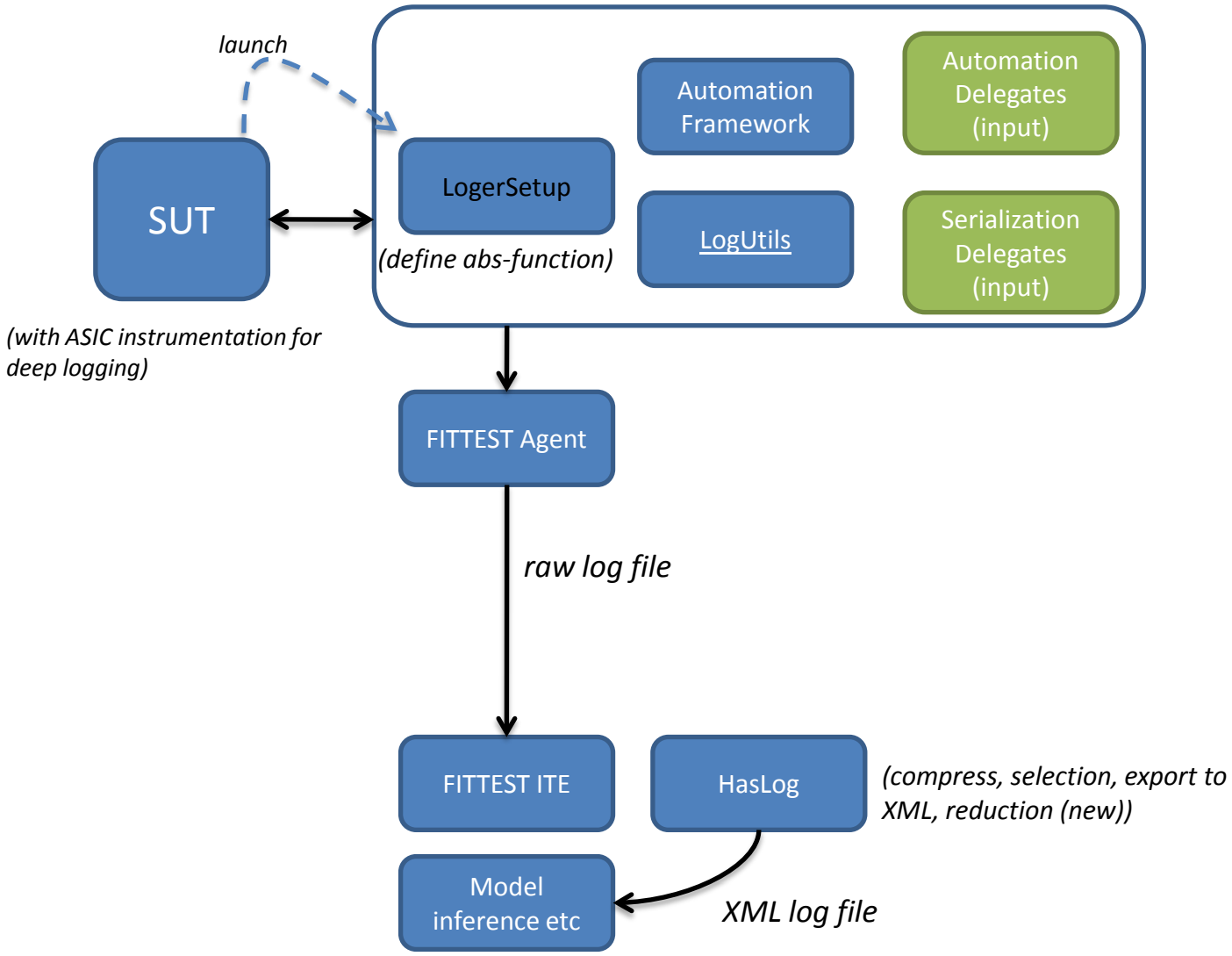
Example of Automation Delegate

```
class ClickableDelegate extends Delegate  {

  public function ClickableDelegate(x:DisplayObject){
       super(x) ;
       // this decides that only event CLICK will be recorded:
       x.addEventListener(CLICK, myHandler);
  }

  public function myHandler(ev:Event) {
     // this will dispatch RECORD-event:
     Automation.record(this, Command.create("click"));
  }

  // not needed for logging, used by replay (for test execution):
  public function click():void  {
     object.dispatchEvent(new MouseEvent(CLICK,true,false));
  }
}
```

FITTEST Integrated Test Environment (ITE)

- Allows logging to turned on/off on users' machines or SUT's server
- Collecting logs
- Launching test suite on other machines




launch
SUT
(with ASIC instrumentation for deep logging)
LogerSetup
(define abs-function)
Automation Framework
LogUtils
Automation Delegates (input)
Serialization Delegates (input)
FITTEST Agent
raw log file
FITTEST ITE
HasLog
(compress, selection, export to XML, reduction (new))
Model inference etc
XML log file