Demonstration of an Integrated System for Platform-independent Description of Human-Machine Interactions

Oleg Stobbe, Klaus Rechert and Dirk von Suchodoletz
Albert-Ludwigs University Freiburg
Hermann-Herder Str. 10
79104 Freiburg i. B., Germany

1. MOTIVATION

When using emulation to render digital objects, a dedicated system environment is required. This environment typically consists of a set of software, i.e. an emulator, replicating the original hardware, operating system, hardware drivers, application as well as tools and utilities. Typically such technical meta-adata is modeled using a view-path. Configuration and operating knowledge, however, is also required and needs to be described and preserved to secure deterministic future environment and workflow replication.
One possible solution is to capture and replay human-machine interaction in an abstract way. A model for recording and capturing interactions between human users and an emulated machine has already been proposed [2]. With the integration of such a system in an Emulation-as-a-Service service model [1], usability has been improved significantly by integration the capturing and replay into EaaS workflows. This demo's purpose is to demonstrate the system's usability and utility for digital preservation tasks like automation, documentation and replication of interactive tasks.

2. ARCHITECTURE & IMPLEMENTATION

To capture and replay any user-interaction either directly with the running emulated system or with the emulator, an interaction workflow description (IWD) recorder is added to EaaS's emulation components. Emulation components abstract each emulator's individual complexity and provide unified interfaces for interaction with the emulated environment. In contrast to so-called macro-recorder, the IWD-system does not interact directly with the emulated operating system and thus, is platform independent and extensible to cover new, upcoming interaction paradigms.
The basic idea of IWD is to simulate a human user's behavior: before executing a single interaction, e.g. mouse movement, mouse click, keyboard input, the system needs to be in the appropriate state, i.e. providing a proper context for a certain action and a potential previous event has to be processed completely. More formally, a single event is described through a precondition, i.e. the system has to be in a specific, pre-defined state pc, an action a, and the expected outcome eo of the user-action ($ev_i :=< pc, a >_i \rightarrow eo_i$). Both, pre- and postcondition of each interaction are verified by using the emulator's visual output and the emulator's internal machine state. Furthermore, we assume that each postcondition is also precondition for the next event.
In the current version synchronization is implemented using visual/graphical output only. Before executing the next action, the system waits for the screen to reach a state "similar enough" to the one, at the time of the action's recording. Pre- and post-conditions are rather simply automatically generated by fingerprinting the emulators output.

2.1 Recording Architecture

When using EaaS to instantiate a dedicated emulated system environment the user is able to simply enable recording of the session's interactions. Recording is performed on the server-side running as a background task independently of the way the user interacts with the emulator (e.g. using a web client or a dedicated VNC, RDP, etc.). An abstract *interceptor* interface allows to capture, filter and manipulate any message sent between the user's client and the emulation component. On the server-side, two worker threads analyze the user's stream of input events and the emulator's output. Specific sync-instructions and timestamps are used for synchronization of both streams. To support visual synchronization, the emulator's visual output is reconstructed

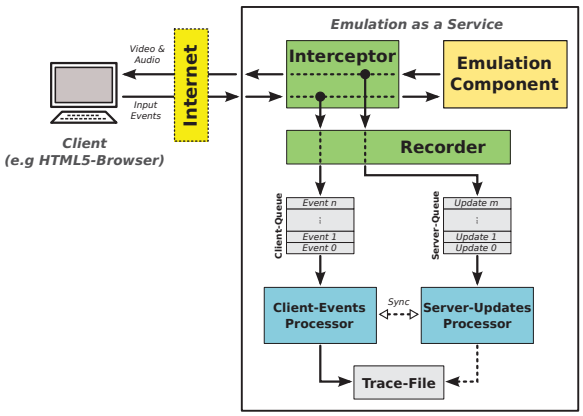

Figure 1: Recording of human-machine interactions using an EaaS setup.

and drawn on the server-side. For instance, all screen updates are processed before a mouse-click event. From this, a snapshot of $n \times m$ pixels surrounding the mouse cursor is written to the trace file. This also design provides some simple by-products such as the creation of screenshots as well as screencast movies both annotated with the emulation's context information. The resulting IWD trace file has been reworked from previous versions making it both human readable (text-based) and efficient to parse and execute.

```
IWD := blocks{trace, meta-data, index}
trace :=  {<timestamp>|<instr_len>|<instr>}*
instr :=  <op_len>.<op>,<arg_length>.<arg>,...;
```

A *trace* block contains the session's events as an abstract description, such as the user's input-events, emulator output and synchronization data. For instance,
`520274|24|5.mouse,3.251,3.782,1.0;`
describes a mouse movement instruction (to 251,782) which occurred 520274*ns* after the start of the recording. The length of the instruction is 24 chars. For synchronization, events like:
`6226615|6434|5.vsync,2.13,2.77,2.40,2.30,6400.[...];`
describe pre- respectively postconditions. In this case 40×30 pixels at position $(13, 77)$ are expected to be similar to the Base64-encoded bitmap.
The trace file's *meta-data* section contains simple key/value pairs providing information regarding the trace file's creation context, e.g. a reference to the environment and emulator used as well as descriptive meta-data to be displayed. The trace file ends with an auto-generated *index* section that provides technical information for efficient parsing.

2.2 Replay

To replay an IWD, the trace-file is fed directly to the emulation component. If requested, the user is able to observe the emulator's visual output. For replay, also two worker threads are used, one for processing the trace file and another one for processing the emulator's output.
When replaying user interactions, it is possible that the emulator may drop events, e.g. if it cannot keep up with input processing. Furthermore, an action may take a varying amount of time for complete execution, such that recorded timestamps of events are not useable for input synchronization. Hence, the replay system has to adapt to the emulator's behavior. Since most input events produce a number of corresponding screen-updates[1], these updates, respectively the update patterns, are used for input synchronization and flow-control, i.e. delay processing of the next events until expected screen-regions are updated, hence the action's outcome is visible. Since screen-updates are not deterministic both by size or position, an expected update pattern is considered as successfully received if it covers the updated screen region. This way, visual synchronization is also available for environments without direct mouse input. Furthermore, this method is computationally efficient since no screen content has to be processed. Fig. 2 shows a recording of a console-based session. The yellow rectangle marks the screen area expected to be an outcome of an previous input action.

[1] For efficiency reasons only a set of tiles, covering changes on the screen are transferred.

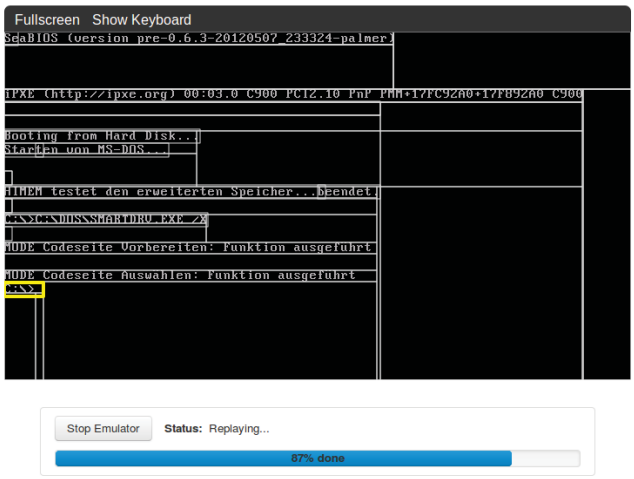

Figure 2: Replay of a console based user interaction with visualized screen-updates (grey) and visual synchronization (yellow).

3. RESULTS, ISSUES & OUTLOOK

With a recording and replay system integrated into the EaaS framework, this system can be used with all available system environments. Yet, the system is usable for simple automation tasks as our tests still exposed unresolved issues, like failed or dropped mouse events (e.g. the mouse event had no effect and the window did not close). Furthermore, some animations, system-clock widgets caused problems due to non-deterministic screen-update events. These issues will be addressed by both implementing more robust pre- and post-condition checks as well as incorporating non-visual feedback from the emulation component (e.g. cpu and I/O status).
Other issues found in our tests, like unexpected error messages e.g. due to networking issues, missing menu-options or files, need to be addressed by the recording user.
Despite these yet unresolved issues, the presented system and its integration into EaaS workflows provides a stable base for new features for the digital preservation community to automate and document tasks on interactive systems. Furthermore, by unifying the communication with EaaS's emulation components, first steps for emulator-independent replay of interactions have been made. With this, captured interactions with today's emulators can be replayed with a future emulator hosting the same system environment.

4. REFERENCES

[1] K. Rechert, I. Valizada, D. von Suchodoletz, and J. Latocha. bwFLA – A Functional Approach to Digital Preservation. *PIK – Praxis der Informationsverarbeitung und Kommunikation*, 35(4):259–267, 2012.

[2] K. Rechert, D. von Suchodoletz, R. Welte, M. van den Dobbelsteen, B. Roberts, J. van der Hoeven, and J. Schroder. Novel workflows for abstract handling of complex interaction processes in digital preservation. In *Proceedings of the 6th International Conference on Preservation of Digital Objects (iPRES2009)*, pages 155–161, 2009.