Evaluating an Emulation Environment: Automation and Significant Key Characteristics

Mark Guttenbrunner
Secure Business Austria
Vienna, Austria
mguttenbrunner@sba-research.org

Andreas Rauber
Vienna University of Technology
Vienna, Austria
rauber@ifs.tuwien.ac.at

ABSTRACT

Evaluating digital preservation actions performed on digital objects is essential, both during the planning as well as quality assurance and re-use phases to determine their authenticity. While migration results are usually validated by comparing object properties from before and after the migration, the task is more complex: as any digital object becomes an information object only in a rendering environment, the evaluation has to happen at a rendering level for validating its faithfulness. This is basically identical to the situation of evaluating the performance in an emulation setting.

In this paper we show how previous conceptual work is applied to an existing emulator, allowing us to feed automated input to the emulation environment as well as extract properties about the rendering process. We identify various significant key characteristics that help us evaluate deviations in the emulator's internal timing compared to the original system and how we can find out if the emulation environment works deterministically, an important characteristic that is necessary for successful comparison of renderings. We show the results of rendering different digital objects in the emulator and interpret them for the rendering process, showing weaknesses in the evaluated emulator and provide possible corrections as well as generalized recommendations for developing emulators for digital preservation.

1. INTRODUCTION

Preserving digital information for the long term means to adapt it to be accessible in a changed socio-technological environment. But applying a preservation action like migration or emulation on a digital object changes elements in the so-called view-path. This includes not only the object but also secondary digital objects needed to render it, i.e. the viewing application, operating system, hardware or rendering devices. To strengthen the trust in these digital preservation actions we have to validate the rendered form of the object (where "rendering" means any form of deploying an information object, being rendering it on a screen or on any other form of output, including acoustic, physical actuators, output on data carriers or TCP/IP ports, etc.) Thus, migration and emulation, usually perceived to be drastically different approaches in digital preservation, actually become rather similar in their principles of evaluating the rendering of the object.

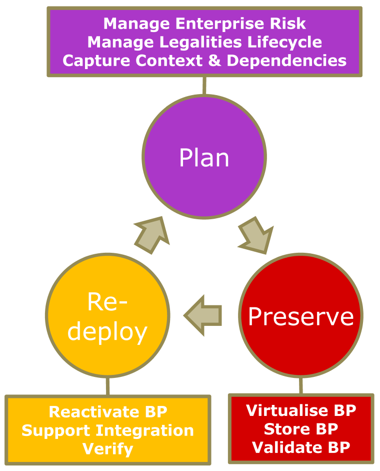


Figure 1: Process for Digital Preservation of Business Processes (BP) in TIMBUS.

The principles are the same: devise a way to capture information from a rendering environment (which we will, without limiting it's general applicability, refer to as "emulator" for the remainder of this paper, and where we will use a concrete emulator as a compact form of a system comprising key elements of the rendering environment providing access to a digital object). We devised a formal framework to evaluate the rendering of digital objects in [8] that is applicable to all kinds of objects, from static files to dynamic processes. In this paper we will validate this framework and show a detailed evaluation of the rendering process.

Evaluating digital preservation actions performed on digital objects becomes a necessity when doing preservation planning to support the decision for the most suitable strategy and tool to perform the action. Similarly the validity of the preservation action has to be checked when preserving the object by executing the preservation action on it, as well as when validating the object once its re-deployed for future execution in a new environment. The different stages as defined in the TIMBUS[1] project are shown in Figure 1, and are explained in detail in [1]. To compare the renderings in these different stages of an object's life cycle, we have to extract characteristics about the rendering process as well as data rendered during this process from the environment. But to reliably compare two different renderings of a digital object it is necessary to avoid side-effects from manual input and other non-deterministic aspects, so we need to automate the evaluation process.

In-depth information about the rendering process is only known inside of the rendering environment. In the case of emulation this is inside the emulator. Based on this we argue that emulators used for digital preservation have to offer functionality to support their evaluation. Based on the theoretical work on the features we would expect emulators to offer [8], we show how we implemented some of these in an existing emulator. We also show how these features are used to automate input to the emulation environment to support automated and repeatable testing uncoupled from exactly timed manual user input. We describe significant key characteristics that we extract from the log files created by the emulator about the rendering processes of various different digital objects. Theses characteristics are analyzed and used to improve the emulator.

While applicable to all kind of preservation actions, in this paper we focus on emulation. We picked an open-source emulator of a home-computer environment as an example of sufficient but still manageable complexity. Using two types of applications with different characteristics and complexity, namely a game as well as a simple, early business application allowing the management of income and expenses, we will validate the key characteristics and feasibility of the proposed approach, and show how these extend to more generic business or eScience processes of generic object renderings.

This paper is structured as follows. First we provide related work on the evaluation of digital preservation actions. Then we give a brief overview of the emulator we chose for evaluation in Section 3. For the remainder of the paper we present the theoretical work on evaluation and how it is implemented in the emulator: We first show in Section 4 how we implemented an event-log. Then we show in Section 5 how we used this log for automated execution of the emulator. In Section 6 we describe how the created logs can be used to extract characteristics about the rendering process and how those can be used for evaluating an emulator. In Section 7 we describe the experiments we performed on different digital objects in the emulator and describe the findings in the rendering logs. Finally, we show our conclusions and give an outlook to future work.

2. RELATED WORK

Choosing the right preservation action for digital objects is a challenging task. To give the team responsible for performing digital preservation activities a certain level of certainty about the digital preservation actions performed, it is necessary to validate the effects of these actions on the significant properties of digital objects.

[1]http://timbusproject.net/

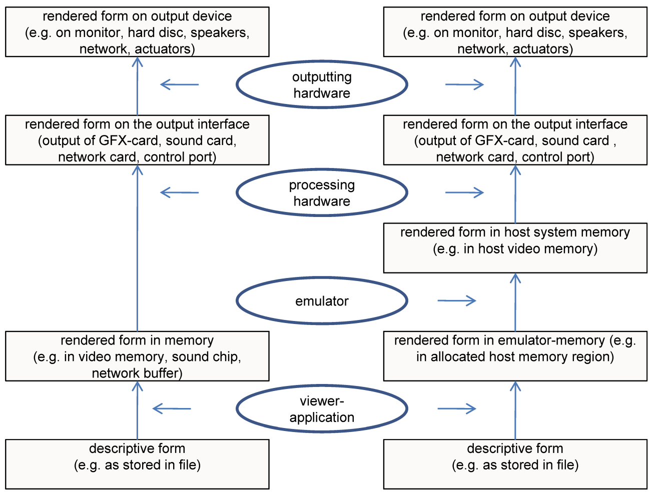
In [2] a preservation planning workflow that allows for repeatable evaluation of preservation alternatives, including migration and emulation strategies, is described. This workflow is implemented in the preservation planning tool *Plato* [3]. As part of the preservation planning automatic characterization of migrated objects can be performed. Tools like Droid [5] are used to identify files. Migration results can be validated automatically supported by the eXtensible Characterisation Languages (XCL) [4]. The original and migrated objects are hierarchically decomposed and represented in XML. These representations can be compared to measure some of the effects of the migration on the digital object. It is, however, not immediately obvious if all the significant properties of a digital object are sufficiently reproduced once it is rendered in a new rendering environment. This new rendering environment can be either different software used to render the migrated file or, in the case of emulation, a new environment in which the original file is rendered.

Comparing rendering results to evaluate the outcome of a rendering process was proposed in [12] as separating the information contained within a file from the rendering of that information. The information stored in the file can, for example, be the coordinates of text or descriptive information about the font to use while the rendering displays the text on a specific point on the screen and uses either a font built into the system or a font stored within the file, which in turn is also rendered in a way specific to the application used for displaying the document. This is described as the *look & feel* aspect of an object. In [9] case studies of interactive objects comparing the rendering outcomes of different rendering environments using the aforementioned characterization language XCL on the level of screenshots of renderings are presented.

Most approaches to evaluate the validity of emulators as a preservation strategy are currently based on manually reviewing the emulation results. In the CAMiLEON project [10] users compared objects preserved by different strategies including emulation. The emulated environments were evaluated by the users as subjective experience with the preserved digital object. A case study to compare different approaches to preserve video games with one of the approaches being emulation was also reported in [6] on a human-observable and thus also to some extent subjective level.

A manual comparison of original and emulated environment is a very time consuming process, that would have to be repeated whenever a new emulator or a new version of an emulator is introduced in an archive due to the necessity of a digital preservation action, e.g. if the hardware platform used for the previous emulator gets obsolete or if any other element in the viewpath (including any system changes on the host environment running the emulator) or on the level of output devices used for the rendering of an object that may have an effect on the result of performing/rendering an information object, change.

In [8] we presented a framework which allows one to determine the effects of an emulated environment on the rendering of objects in a methodical way and suggest methods



rendered form on output device (e.g. on monitor, hard disc, speakers, network, actuators)
outputting hardware
rendered form on output device (e.g. on monitor, hard disc, speakers, network, actuators)
rendered form on the output interface (output of GFX-card, sound card, network card, control port)
processing hardware
rendered form on the output interface (output of GFX-card, sound card , network card, control port)
rendered form in host system memory (e.g. in host video memory)
emulator
rendered form in memory (e.g. in video memory, sound chip, network buffer)
rendered form in emulator-memory (e.g. in allocated host memory region)
viewer-application
descriptive form (e.g. as stored in file)
descriptive form (e.g. as stored in file)

Figure 2: Different forms of a digital object in a system's memory. On the left the layers of an original system are shown, on the right the layers of the system hosting the emulator are shown.

to automate the process of evaluation to some extent. We described the different methods to automate input to the rendering environment to ensure that changes in manual handling of the digital object can be ruled out as a cause for changes in the rendering process. We also described the levels on which information can be extracted from the emulation environment as shown in Figure 2.

In this paper we show how we apply some of the concepts presented in the theoretical work on an existing emulator we presented in [7]. We implement automated input and extraction of data from the emulated environment. We identify key characteristics of the rendering process which can be measured automatically to not only evaluate the emulation environment but also to help improving the emulator.

3. VIDEOPAC EMULATOR O2EM

The emulator we chose for implementing features for evaluation, O2EM[2], was previously described in [7]. It emulates a home-computer system introduced in 1978 as well as an updated version of the system released in 1983. The original system is both usable as a video game console by inserting cartridges with games, but due to its built-in keyboard it was also used as a home-computer with a special BASIC-cartridge. In this home-computer-mode the system was able to run BASIC programs and also load and save data to an external tape recorder.

In our previous work we implemented features in the emulator to make it usable for digital preservation purposes from a user's point of view (e.g. data exchange between the emulated and the host system). To actually be able to use an emulator in a digital archive, however, we need the possibility to evaluate the rendering process of digital objects more objectively and in an automated way. Based on our theoretical work in [8] we decided to implement the following features:

Event-Log The original system can be controlled by using either the keyboard of the system or joysticks. In interactive applications (and especially video games) timeliness and type of input usually have a major influence on the behavior and thus resulting rendering of the digital object. Besides recording the points and type of input, we also wanted to log other events like file access (reading / writing to files in home-computer-mode) and the start of drawing an image frame (i.e. the start of the Vertical Blank period on the original system), to allow us to make statements about the correct timing of the emulator compared to the original system. Additionally, we recorded user-driven events in the emulator such as triggering a screenshot or a memory dump.

Automated Input The previously created event-log was defined in a form that is usable also as a command-file for the emulator, allowing us to automatically apply input to the system as well as create screenshots and memory dumps at specified times.

Memory Dumps We also implemented a feature to trigger memory dumps of the different memory regions in the system, including the hardware registers of the multimedia processors. This allows us to not only rely on screenshots of the emulator or files saved in the home-computer-mode as a way to extract data from the rendering process.

The next sections describe in detail the design decisions taken when implementing these features.

4. RECORDING OF EVENTS

The migration of an object lets us to some extent draw conclusions about the digital preservation action taken by comparing the object's properties before and after migration. Yet we need to draw conclusions on the rendering process of a digital object. We have to extract that information from the rendering environment and not from the object. To allow this, we need to implement an event-log of the rendering process in the rendering environment, e.g. an emulator or the new viewer application. We decided to include the following information in the log-file:

Executed Cycles The system emulated in the rendering environment usually runs at a different clock speed than the host system. Therefore we decided on the number of executed cycles as the main indicator of timing of when an event appears. This adds value for automated testing, as during an unsupervised test the emulator can be run without any speed limits, thus reducing the time needed for testing.

Elapsed Time As an additional time measurement we also record in the log-file the actual elapsed time since the rendering process was started. This measurement gives us an indication of how the emulator speed is perceived by a user of the emulator and may be used to normalize for timed events driven by a clock-based system rather than execution cycles based timing.

Drawn Frame As an additional timing measurement we record for every event in which 'frame' (unique consecutive image produced by the video hardware of the system) the

[2]O2EM Sourceforge Page - http://o2em.sourceforge.net/
O2EM DP version - http://www.ifs.tuwien.ac.at/dp/o2em

event was registered. (Note: For the purpose of this study we focus on the screen rendering for ease of presentation. Other forms of output rendering, such as acoustic behavior or output on other devices such as storage units, are considered in a similar manner.)

Recorded Event For each event we record the type of event as a code and as full text (for easier human readability).

Additional Infos Additional information on the recorded event is included, e.g. the key that has been pressed, the file that has been accessed etc.

To easily import the resulting file in spreadsheet applications for further processing and analysis we decided to use a comma separated value (CSV) format escaping commas that form part of the input in the log. When starting the emulator the event-log file that should be created can be specified as an extra parameter.

The following different types of events were defined for the system emulated in the emulator:

4.1 Controlling the Environment

To be able to evaluate the rendering process reliably, we have to make sure that the rendering is always exactly the same under the same conditions applied to the rendering environment, i.e. the emulator is deterministic in its behavior. Lamport et al. describe deterministic algorithms as „algorithms in which the actions of each process are uniquely determined by its local knowledge" ([11]). This means that for any object rendered in the environment relying on external input to the rendering environment (e.g. user input, network activity, access to files on the host system) the type of input as well as the actual input data have to be stored to be able to provide the same data on a re-run for evaluation purposes.

The emulator O2EM (and the original system it emulates) supports user input in the form of key presses and joystick input. The hook-point for recording these events for the event-log is the interface in the emulator between the emulated environment and the host environment, i.e. when the emulator detects that the emulated process is trying to access the hardware registers that usually store the input values and provides the host system input instead. By recording the exact cycles already executed in the rendering when accessing this information, we are able to provide the same information when re-running the rendering process.

Reading files in home-computer-mode as a different type of providing external data to the rendering environment was recorded in the event-log, to let the digital archivist know that for later evaluation of the emulator these files have to be present besides the actual digital object, as they also potentially influence the rendering process.

4.2 Extraction of Data

As a basis for comparing the results of the emulation process, it is necessary to extract not only events but actual data as a result of the rendering. In Figure 2 we show different levels on which a rendered object exists during the rendering process. From inside the emulator we have access to two different forms of rendered information: the form in the (emulated) memory of the system (e.g. hardware registers of the multimedia processor, usually triggering an output on the original system) as well as the form that is already translated to the host system (e.g. a rendered screen based on hardware registers of the emulated system's video hardware).

In O2EM a feature to save screenshots of the currently displayed image was already present. We enhanced this feature to create an event-log entry including (as every log entry) the executed cycles up until the point in the rendering the screenshot was taken. Additionally, we implemented a feature that works similar to saving screenshots that lets the user save the different emulated memory regions of the host system: memory internal to the processor, main system memory external to the processor, multimedia hardware registers memory and, if available, the emulated home-computer-mode memory. Additionally, in home-computer-mode files can be stored externally, which also influences the rendering process. The process of writing these files was also recorded in the event-log.

Under the assumption that the emulator works as a deterministic process, extracting data under the same external conditions (e.g. the exact same input applied) at the same point in the rendering process should provide the exact same result files.

4.3 Additional Events

In addition to the events described above, we also defined two other special event types for the log:

Vertical Blank The vertical blank is the period before the drawing of a new frame is started. It was an important event used to synchronize events on the screen to a fixed timing. We implemented this event to let us draw additional conclusions about how the number of cycles executed and the frames being drawn relates to the original system's timing.

Emulation Start For O2EM we record information about the cartridge image file that was rendered (filename and a checksum), as well as name and version number of the emulator and the date and time the log was created. This metadata gives us additional information about the rendering process for which the log was recorded.

Emulation Stop The information that the rendering process was stopped, the total number of cycles executed, the number of frames drawn and the elapsed time is recorded in the event-log.

5. AUTOMATED EXECUTION

Recording the events of a rendering process is only the first step in validation and verification of the digital preservation action. Especially if the rendering environment changes between execution of the digital preservation action and the re-deployment of the digital object at a later point in time, it is necessary to verify the correct rendering of the object in the new environment.

To be able to compare the rendering between validation (the time the digital preservation action was initially performed) and verification we need to make sure that the external conditions influencing the execution are unchanged. This means that any manual input or external data applied to the rendering environment has to be the same as when the preservation action was initially validated. By recording these external events in a rendering environment and applying them at a later point in time to the new environment, we can compare the outcome of the rendering process.

In the emulator O2EM we implemented a feature to use the earlier described event-logs as command files. All external events and triggered data export actions recorded in the event-log file are automatically provided to the emulator using the command file. Actions are read from the command file and applied to the emulator when the specified number of cycles have been executed. In a deterministic emulator this means that the relevant actions are applied at the same time in the rendering process as they initially had been recorded.

In detail the following actions where implemented:

Keyboard and Joystick Input The manually recorded input events are applied at the exact same cycle count as initially recorded. The action from the command file is (similarly to the recording of the input for the event-log) interpreted once the emulator invokes the interface in which the emulated system tries to receive input from the host system. In a deterministic emulator the number of cycles executed until this check is performed does not change between renderings of the same digital object.

Screenshot and Memory Data Extraction The manually triggered extraction of data that has been recorded in the event-log file is automatically executed once the executed cycles stated in the command file are reached. Additional extractions can be inserted manually. This way it is possible to extract both a screenshot and all memory regions at the same point in the rendering process.

End Emulation The initial event-log record of the emulation stop also stops the emulation in the re-run once the action is encountered in the command file. This allows for automated and unattended testing of the emulator.

By first recording external events and later applying the event-log as a command file for a new version of the emulator (or even a different emulator) it is possible to automatically test an emulator. If the resulting data extracted at significant points in the rendering process is identical, we have a strong indication that the rendering process is unchanged.

6. KEY CHARACTERISTICS OF RENDERING PROCESS

Analyzing the event-log and using the features implemented in the emulator, we identified meaningful key characteristics of the rendering process, to see how the logs can help us evaluate if the rendering stays true to the original system or how it differs between different emulators (or different versions of the same emulator).

Deterministic Rendering The most important characteristic of a rendering environment is that the rendering process must be deterministic. This means that the emulator has to perform the same rendering process under the same inputs. This is of crucial importance to the evaluation, as only a deterministic process lets us compare different renderings of the same object and the results of it.

Cycles Executed vs. Emulation Time Another characteristic we can extract from the rendering log is how many CPU cycles have been executed during the course of the emulation. If we compare these with the cycles that would have been executed on the original system (using the known clock rate of the original system), we can calculate the deviation in speed of the rendering process compared to the original system.

Executed Cycles per Frame By measuring the cycles that are executed per frame, we can see if the timing is correct. As we know the clock rate and the number of frames drawn on the original system from the systems specifications, we can evaluate any discrepancies to the original hardware.

Time Needed to Draw a Frame By evaluating the time that is needed to draw a frame and knowing how many frames are drawn per second (and thus the time the drawing of one frame should take) this characteristic also helps us evaluating the timing of the emulator.

Frames per Second Determining the frames per second we can see if the emulator is running slower than the original system is supposed to. If the emulator is in fact not fast enough, we can see from the event-log which of the drawn frames took too long to calculate and what external events happened during the slow frames.

Accessed External Sources By implementing a log for all interfaces between the emulated and the host environment, we also know which external resources (files, network, etc.) are used by a digital object. By logging the data that is transferred, we can decouple and simulate external interfaces at a re-run of the rendering process.

Using these key characteristics, we can evaluate an emulator, but also draw conclusions on the rendering process - not only in general for the rendering environment, but for specific digital objects. Re-running the same automated test in the emulator we can evaluate if the emulator works deterministic. Re-running the automated test of a deterministic emulator on a new version of the emulator we can test if the emulator still works correctly. Finally re-running the test in a different emulator for the same system, we can compare the results of these emulators.

7. EVALUATION OF O2EM

In this section we describe some of the experiments we performed on different digital objects suitable for the emulator we adapted. We describe the steps undertaken and the results of the rendering processes as well as the analysis of the resulting event-log files.

7.1 Video Game: Terrahawks

As a first digital object we chose a video game running in the standard mode of the emulator emulating a Philips Videopac G7000 running in European timing mode (PAL video stan-

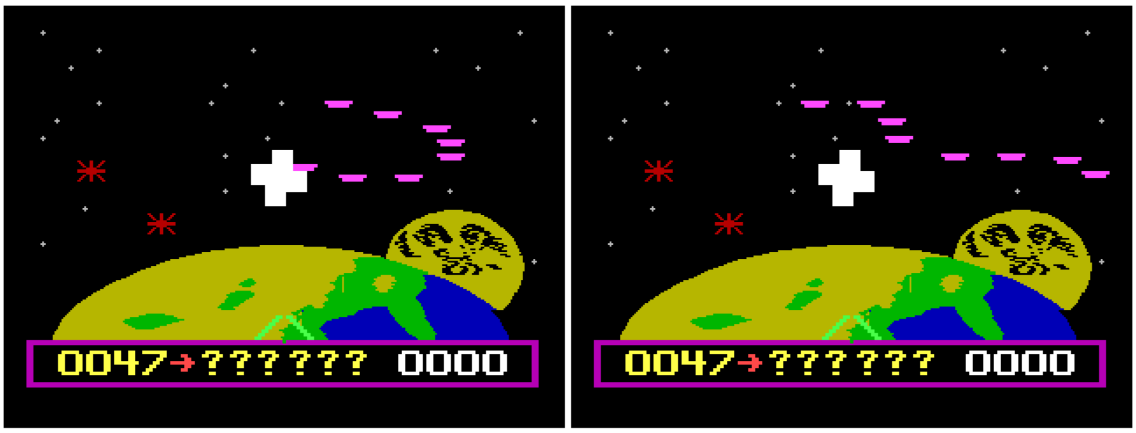


Figure 3: Non-deterministic rendering of Terrahawks - result of initial recording on the left, re-run on the right.

dard). We chose a video game as one of the objects because those are usually the most timing sensitive objects on the chosen hardware.

We chose the game *Terrahawks* that creates a random impression using external events to change the gameplay on every execution, to see if repeated execution of the game will produce the same rendering results, i.e. if the rendering process can be made deterministic.

As first step the emulator was started and one gameplay was recorded, both input using joystick but also some key presses (to enter letters for the highest score). A screenshot was taken after the game resulted in the player loosing his life (at which point the game just restarts, showing the new highest score on the bottom). In a second step the emulator was restarted with the event-log file given as a command-file. The previously recorded input was applied automatically during the rendering process. However the resulting screenshot taken at the same point in the rendering process as the original screenshot differed from the initial run of the emulator as shown in Figure 3.

A closer look on the emulator source code revealed that the emulation process was not entirely deterministic (i.e. independent from external factors), as the emulation of one of the hardware components, a voice synthesis module, was actually simulated using sound samples. The status check in the emulated code of this component was connected to the actual completion of playing the sample of the host system, an event the emulated environment had no control over. By deactivating the voice component, the emulation process was made deterministic and when the experiment was repeated, the results were identical on each re-run.

As timing in video games (especially action games) is crucial for the game experience, we used the rendering log to compare the timing of the real hardware (known due to the original system's schematics) to the values measured in the log as described in Section 6. The measured values as well as the expected values calculated from the original system's specification can be seen in Table 1.

Characteristic	Calculated	Measured
executed cycles per frame	7882	7259
executed cycles per second	394100	435540
frames per second	50	60
seconds per frame	0,02	0,0165

Table 1: Calculated versus measured key characteristics taken from the event-log of running Terrahawks in O2EM.

Based on these results it can be seen that due to the evaluation log we detected another error in the emulator. Even though the emulator was executed with the timing set to European TV-standard PAL timing (50 frames per second), the emulator was still rendering 60 frames per second as in the North American TV standard NTSC. The time taken for each frame was consistently 1/60 of a second, which is correct based on NTSC timing. The emulator was running fast enough to render every frame in less time than the original system would have needed, keeping the subjective feeling of speed for the user steady. Furthermore, it can be seen in Table 1 that the timing inside the emulator is not cycle-correct, thus timing-sensitive applications would not run correctly.

The findings about the incorrect timing were used to fix the errors in O2EM and improve the timing in the emulator, thus helping us to get a better rendering environment.

7.2 Application: Cassa

As a second example we chose not a video game but an application that runs in the home-computer mode of the system. We chose an application that allowed us to save data to the external tape drive and reload the data and render it during later use. The application was a BASIC-program distributed with the system in Italy, allowing the user to keep track of income/spendings per month over a year. We started the computer in home-computer mode, loaded the program, entered various fictitious data and saved the data in the program. For the actual evaluation we recorded the following process in the event-log: starting up the emulator in home-computer mode, loading the program (of which we

Characteristic	limited	no limit
total executed cycles	49201503	49201503
total frames drawn	6778	6778
total emulation time	136.426	10.512

Table 2: Characteristics for testing the application Cassa with original (=limited) and unlimited speed.

took a screenshot as seen on the left in Figure 4), loading the data into the program and displaying the data as also shown on the right in Figure 4. So not only the recorded user input but actual data loaded from an external drive influenced the rendering (i.e. what was shown on the screen).

To test the emulator in home-computer mode for determinism, we not only recorded screenshots (as due to the missing random element in the application those would most probably be similar), but also save the memory content of all different memory regions along with the screenshot of the displayed data (i.e. after the data was loaded into memory). We ran the test under two different settings in the emulator, first with speed limited as a user would usually experience it, and a second time without speed limit, simulating a verification where the test should be performed as fast as possible. We compared all the exported data files (screenshot and memory) with the result, that in all cases the files where exactly the same. As for the timing of the different runs as shown in Table 2, we can see that on our system the unlimited test executed the exact same test in only 7.7% of the time needed for a correctly timed emulation while creating the same results.

In the event-log we could also see the external files that had been loaded. These included not only the application Cassa itself, but also the file used for storing user entered data. In a real-life scenario this would enable us to identify which resources had been accessed and keep (or simulate) the necessary data for a later verification of the rendering of the preserved application.

8. CONCLUSIONS AND FUTURE WORK

In this paper we presented how previous conceptual work on evaluating emulators can be applied to evaluate the rendering process of different types of digital objects in an existing emulator. We first introduced the event-log of the rendering process with different properties that allow us to re-run a rendering in the same environment and potentially also in different ones. We showed the different kinds of events that have to be recorded depending on the original system. The different types of external data that can influence the rendering process have been explained as well as the different types of data that can be exported from the rendering environment for a comparison of different rendering processes. We then explained how the event-log can be used to automate the process of applying the same input data to the emulator to ensure a deterministic rendering of the digital object. After introducing the different key characteristics of the rendering process we identified in the event-log, we evaluated two different digital objects in the emulator O2EM and explained how the event-logs helped us to identify flaws in the rendering process.

Theoretical work we presented in [8] was successfully implemented in the emulator O2EM.

We rendered different objects in the emulator and analyzed the event-log files, which led us to the following conclusions:

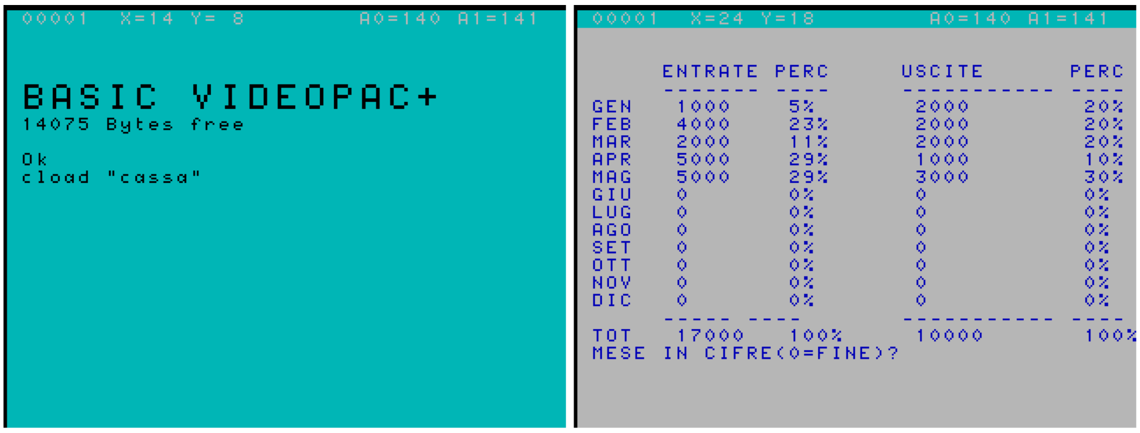
Deterministic Emulation Automatically evaluating emulators by comparing the rendering results at different points in the rendering requires that the rendering environment behaves the same provided with the same external data. In the case of the game 'Terrahawks' evaluated in Section 7.1 the emulation was initially not deterministic, leading to different results of the rendering process, even though the obvious external data (user input) was kept constant. Only by making the rendering process deterministic, we could successfully compare the renderings in consecutive runnings of the emulator. This would also be the basis for later comparison of the rendering to later emulator versions or even other emulators.

External Data The external data needed to create a deterministic rendering is the one that is passed up from the host environment into the emulated environment. By recording the data that is transferred on these interfaces, we can apply the same data at the same point in the rendering process at a later time ensuring a deterministic rendering process. With the application 'Cassa' we showed that the external events (file access and user input) can be tracked in the event-log. External resources can then either be stored for a re-run for validation purposes or even simulated if the resources are no longer available (e.g. an external Web services).

Key Characteristics Using the key characteristics about the rendering process which we extracted from the event-log we were able to draw conclusions on the correctness of the emulation process. Especially deviations in handling the timing in the emulator were detected, assisting the emulator authors in improving the rendering process. Obviously when extending the described characteristics to more complex systems, additional characteristics could be found. Additionally to the time needed to draw a frame on the screen, similar measures could be captured for other output devices, e.g. port communications etc., where the timing of events needs to be captured, normalized and compared.

Automation of Evaluation Applying the external data to the rendering process not only gives us a possibility of creating a deterministic rendering, we can also automate the process of evaluating a rendering environment by applying the user input to a digital object automatically. This way interactive digital objects could be tested automatically on re-deployment in a new environment to see if the rendering is the same as at the time they have been preserved. We also showed that for this automated evaluation we not necessarily have to run the rendering process at the original system's speed, as all the automation is based not on time passed but on CPU cycles executed in the rendering environment, thus massively speeding up the process of the validation.

Overall we successfully implemented some of the concepts described in [8] in the existing emulator O2EM. This not only allowed improving the emulator for more accuracy, but also gave us a better understanding of the evaluation of ren-



```
00001  X=14 Y= 8          A0=140 A1=141

BASIC VIDEOPAC+
14075 Bytes free

Ok
cload "cassa"
```

```
00001  X=24 Y=18          A0=140 A1=141

     ENTRATE PERC     USCITE      PERC
     ------- ----     ----------- ----
GEN   1000    5%       2000        20%
FEB   4000    23%      2000        20%
MAR   2000    11%      2000        20%
APR   5000    29%      1000        10%
MAG   5000    29%      3000        30%
GIU   0       0%       0           0%
LUG   0       0%       0           0%
AGO   0       0%       0           0%
SET   0       0%       0           0%
OTT   0       0%       0           0%
NOV   0       0%       0           0%
DIC   0       0%       0           0%
     ----- ----       ----------- ----
TOT   17000   100%     10000       100%
MESE IN CIFRE(0=FINE)?
```

Figure 4: Two screenshots taken during the rendering of Cassa - before starting the loading process on the left and after displaying data on the right.

dering environments in general. We showed that it is possible to automate the process of evaluating interactive objects beyond the manual testing of emulators with human interaction.

Future work has to be done on applying the concepts to more complex rendering environments like virtual machines that are more interwoven with the host system. Input and output events would have to be defined for more complex systems to catch all the events that are needed to make the rendering environments deterministic.

9. ACKNOWLEDGMENTS

This research was co-funded by COMET K1, FFG - Austrian Research Promotion Agency and by the European Commission under the IST Programme of the 7th FP for RTD - Project ICT-269940/TIMBUS.

10. REFERENCES

[1] J. Barateiro, D. Draws, M. Neumann, and S. Strodl. Digital preservation challenges on software life cycle. In *16th European Conf. on Software Maintenance and Reengineering (CSMR2012)*, 3 2012.

[2] C. Becker, H. Kulovits, M. Guttenbrunner, S. Strodl, A. Rauber, and H. Hofman. Systematic planning for digital preservation: Evaluating potential strategies and building preservation plans. *International Journal on Digital Libraries*, 10(4):133–157, 2009.

[3] C. Becker, H. Kulovits, A. Rauber, and H. Hofman. Plato: a service-oriented decision support system for preservation planning. In *Proceedings of the ACM/IEEE Joint Conference on Digital Libraries (JCDL'08)*. ACM, June 2008.

[4] C. Becker, A. Rauber, V. Heydegger, J. Schnasse, and M. Thaller. A generic XML language for characterising objects to support digital preservation. In *Proc. 23rd Annual ACM Symposium on Applied Computing (SAC'08)*, volume 1, pages 402–406, Fortaleza, Brazil, March 16-20 2008. ACM.

[5] A. Brown. Automatic format identification using PRONOM and DROID. *Digital Preservation Technical Paper 1*, 2008. http://www.nationalarchives.gov.uk/aboutapps/fileformat/pdf/automatic_format_identification.pdf.

[6] M. Guttenbrunner, C. Becker, and A. Rauber. Keeping the game alive: Evaluating strategies for the preservation of console video games. *International Journal of Digital Curation (IJDC)*, 5(1):64–90, 2010.

[7] M. Guttenbrunner and A. Rauber. Design decisions in emulator construction: A case study on home computer software preservation. In *Proceedings of the 8th International Conference on Preservation of Digital Objects (iPres 2011)*, pages 171–180, 11 2011. Vortrag: iPres 2011 - 8th International Conference on Preservation of Digital Objects.

[8] M. Guttenbrunner and A. Rauber. A measurement framework for evaluating emulators for digital preservation. *ACM Transactions on Information Systems (TOIS)*, 30(2), 2012.

[9] M. Guttenbrunner, J. Wieners, A. Rauber, and M. Thaller. Same same but different - comparing rendering environments for interactive digital objects. In M. Ioannides, D. W. Fellner, A. Georgopoulos, and D. G. Hadjimitsis, editors, *EuroMed*, volume 6436 of *Lecture Notes in Computer Science*, pages 140–152. Springer, 2010.

[10] M. Hedstrom, C. Lee, J. Olson, and C. Lampe. The old version flickers more: Digital preservation from the user's perspective. *American Archivist*, 69:28, 2006.

[11] L. Lamport and N. Lynch. *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, chapter 18, pages 1157–1200. Elsevier Science Publishers B.V., 1990.

[12] M. Thaller. Interaction testing benchmark deliverable PC/2 - D6. *Internal Deliverable, EU Project Planets*, 2008. http://planetarium.hki.uni-koeln.de/planets_cms/sites/default/files/PC2D15_CIM.pdf.