A Measurement Framework for Evaluating Emulators for Digital Preservation

MARK GUTTENBRUNNER, Secure Business Austria
ANDREAS RAUBER, Vienna University of Technology

Accessible emulation is often the method of choice for maintaining digital objects, specifically complex ones such as applications, business processes, or electronic art. However, validating the emulator's ability to faithfully reproduce the original behavior of digital objects is complicated.

This article presents an evaluation framework and a set of tests that allow assessment of the degree to which system emulation preserves original characteristics and thus significant properties of digital artifacts. The original system, hardware, and software properties are described. Identical environment is then recreated via emulation. Automated user input is used to eliminate potential confounders. The properties of a rendered form of the object are then extracted automatically or manually either in a target state, a series of states, or as a continuous stream. The concepts described in this article enable preservation planners to evaluate how emulation affects the behavior of digital objects compared to their behavior in the original environment. We also review how these principles can and should be applied to the evaluation of migration and other preservation strategies as a general principle of evaluating the invocation and faithful rendering of digital objects and systems. The article concludes with design requirements for emulators developed for digital preservation tasks.

Categories and Subject Descriptors: H.3.7 [**Information Storage and Retrieval**]: Digital Libraries—*Standards; systems issues; user issues*; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces—*Input devices and strategies*; I.6.6 [**Simulation and Modeling**]: Simulation Output Analysis; I.6.8 [**Simulation and Modeling**]: Types of Simulation—*Continuous; discrete event; gaming*

General Terms: Reliability, Standardization, Documentation

Additional Key Words and Phrases: Emulation, digital preservation, characteristics of rendered content, system properties, automated testing, preserving interactive content

ACM Reference Format:
Guttenbrunner, M. and Rauber, A. 2012. A measurement framework for evaluating emulators for digital preservation. ACM Trans. Inf. Syst. 30, 2, Article 14 (May 2012), 28 pages.
DOI = 10.1145/2180868.2180876 http://doi.acm.org/10.1145/2180868.2180876

Work in this article has been partially funded by COMET K1, FFG – Austrian Research Promotion Agency and by European Community under the Information Society Technologies (IST) Programme of the 6th FP for RTD – Project IST-033789/PLANETS and the 7th FP for RTD – Project ICT-269940/TIMBUS. The authors are solely responsible for the content of this article. It does not represent the opinion of the European Community, and the European Community is not responsible for any use that might be made of data appearing therein.
Authors' addresses: M. Guttenbrunner (corresponding author), SBA Research gGmbH, Favoritenstr. 16, A-1040 Vienna, Austria; email: guttenbrunner@ifs.tuwein.ac.at; A. Rauber, Vienna University of Technology, Vienna, Austria.

DOI 10.1145/2180868.2180876 http://doi.acm.org/10.1145/2180868.2180876

1. INTRODUCTION

Digital preservation deals with the issue of keeping digital data usable even after the original environment is obsolete. Two main strategies have evolved in previous years: migration and emulation. While migration is a strategy to convert data to a newer, not obsolete format, emulation keeps the data in the original format and recreates the rendering environment for the digital object in a different form. A digital object in this context can be any kind of digital content in static, dynamic, or even interactive form. Dynamic and interactive digital artifacts that have to be considered are video games, interactive art, interactive documents, and other applications such as business processes and numerical analysis.

Preservation planning is used to evaluate what strategy and what tool is suited in a specific preservation scenario. In preservation planning the effects of applying different strategies and/or tools to a corpus of digital objects are evaluated. This is usually done by analyzing the properties of a digital object, identifying those that are relevant in the scenario, and inspecting the properties after applying the preservation action (i.e., after migrating the object to a different format or rendering the object in an emulated environment). Once the effects of the preservation action on the digital objects are known, a decision can be taken on one of the alternatives as the optimal one in the setting. This decision takes not only into account how well the significant properties are preserved, but also considers other factors, for example, the cost, legal, and infrastructure properties of the evaluated alternatives [Becker et al. 2009].

If a digital object is migrated, the information necessary to successfully render all significant properties of the object has to stay intact. It is necessary to evaluate how the migration of an object influences the ability of applications to display it in a suitable form, for example, as close to the original as possible. However, the rendering plays an even more important role in emulated environments. Not only does the object have to be rendered by its environment, but also a reproduction of the original environment has to be rendered on a host system to enable the rendering of the object. To evaluate the effects that rendering has on an object, and thus allowing a preservation planner to judge if the significant properties of an object are faithfully reproduced, it is necessary to look beyond the content of a file stream. We have to examine the rendered version of the object in the form in which it is reproduced, which usually is in its visual form on the screen and audible elements through speakers, or any other input/output behavior. Note that rendering encompasses not just the visual display of an application but any effect and interaction that a digital object has with its environment, including storage, network communication, and actuator activation. If the behavior of an object depends on external factors, these factors have to be kept identical to determine if deviations in behavior are caused by the emulation environment. Depending on the object it is important to identify from which of the diverse rendered forms of an image in the original environment or emulated environment the significant properties shall be extracted and in what state. To make the process of comparing rendering effects in emulation environments less time-consuming and more objective it is necessary to develop concepts which allow for automatable evaluation.

This document describes the integration of emulators in the preservation planning process. It describes a testing environment for evaluating the effects of emulation on digital artifacts. The remainder of this article is structured as follows: First we show the different levels on which emulation can be performed. Next we give an overview on related work in digital preservation and emulation in Section 3. We then take a closer look at the digital object and what properties with regards to determinism and significant states we have to differentiate in Section 4. Next we discuss the rendering environment in which the digital object is executed in Section 5. We explain the necessity of a reference environment to which to compare to. We discuss the view-path of the digital object and show methods to reduce the side-effects of differences in user input on the digital object. We describe the implications of these methods and how to record them in the original environment to apply them in the emulated environment. Based on the information collected about object and environment we then show in Section 6 how to recreate the viewpath for the digital object in the new rendering environment. The different rendered forms of a digital object are shown along with a discussion about how to extract properties of the rendering process. By combining the concepts in the previous sections we are then able to define a set of tests as shown in Section 7.1 that allow us to evaluate to what degree the execution of a digital object in an emulated environment is influenced by the environment's behavior. We discuss the necessary requirements for rendering environments to support automatic evaluation and long-term stability for digital preservation. In Section 8 we discuss the conclusions that can be drawn from the work presented in this article and how the concepts presented are applicable not only to emulation but to other preservation actions as well.

2. LEVELS OF EMULATION

Emulation can be done on different levels. The term *emulation* in general refers to the capability of a device or software to replicate the behavior of a different device or software. It is possible to use hardware to emulate hardware or to use software to emulate software or the hardware. While even a viewer of a document can be seen as some kind of emulation (if, e.g., fonts not stored in a document are provided by the viewer), emulation usually is seen as recreating the hardware of the system the digital object was initially rendered on and using the original software (both the operating system as well as the application) to display it. With applications distributed over various systems in a network, it is also necessary to emulate more than one system to keep a digital object in a usable state. But also social properties (subjective properties not technically measurable) like input devices or even the environment in which a system has been used can be necessary to recreate the original look and feel. The boundaries between the different kinds of emulation are usually blurred and depend on the digital object that has to be preserved.

In this article the word *emulator* is used as defined in Slats [2003] as a program that runs on one computer virtually recreating a different computer's hardware. We mainly concentrate on emulators emulating computer architectures, but the concepts introduced in this article certainly have to be considered with the different levels of emulation in mind and most can be applied to other levels of emulation as well, up to the evaluation of migration results within an appropriate rendering viewer environment.

Environments for digital objects can be emulated on different levels as shown in Figure 1. These layers can be defined as follows.

Application. Instead of using the original software application used to create and view a digital object, another piece of software, sometimes also referred to as viewer, can be used to emulate the original application. In the case of migration, again, a different viewer application is used to render a modified/transformed object.

Operating System. Program-loaders like Wine[1] allow the execution of programs on different operating systems by providing a translation layer for the operating system calls. This allows emulation on a much higher level than hardware without having to emulate the whole system. Programs written for the same hardware

[1]Wine—http://www.winehq.org/.

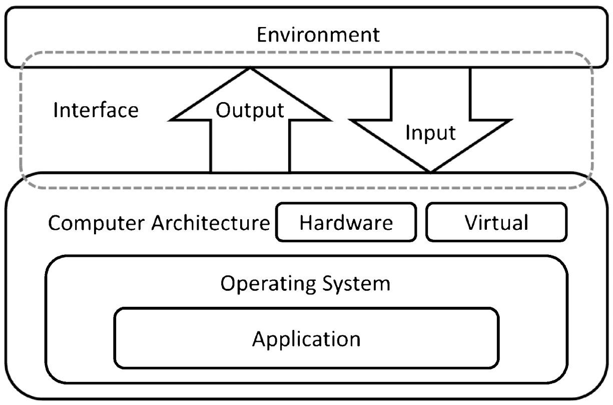

Fig. 1. Layers of emulated environments for digital objects.

but a different operating system can thus be executed in the new operating system environment.

Computer Architectures. As the functionality of hardware is usually better documented than software the most common level of emulation is recreating the computer architecture. Two methods which are widely in use are described next.

— *Virtualization*. An approach to create a virtual environment by using either some or all of the hardware of the host system directly is called virtualization (Virtual Dos Machine under Windows, DOSEMU[2] under Linux). Using this approach, software which is potentially able to run on the host system's hardware is run in a virtual machine hosted by the current operating system environment. An example is to have a host system running Linux and using DOSEMU to create a virtualized environment running a Microsoft Windows operating system.

— *Hardware*. The most common use of emulation is the recreation of hardware components in software on a new host system. In this case, none of the underlying hardware components are used. Instead, the whole system is rebuilt in software and the original digital artifact is executed using the original software in this simulated hardware environment. Examples are emulators for proprietary hardware such as video game console systems on a PC or emulators for PC hardware on virtual machines (e.g., Dioscuri[3]).

Interface Level. Emulating a computer system on the interface level requires to recreate the original means of input/output of the system to recreate the original communication experience with the system. For two-way communication we have to consider the following.

— *Output Devices*. Using a mobile hand-held with a 3-inch screen as an output device as the original system has a different look-aspect compared to the emulation of the same system on a PC screen with 18 inches. An example is an emulator of a Nintendo DS with two screens on a PC with a normal LCD screen running a Microsoft Windows operating system. Similarly, actual actuator output in a control system is different to compare to a simulated actuator output that may be as a visual-, acoustic-, or voltage-level setting.

[2]DOSEMU—http://dosemu.org/.
[3]Dioscuri—http://dioscuri.sourceforge.net/.

— *Input Devices*. Using paddle controllers to play the game Pong is a different feel-aspect than using a keyboard or even a mouse to control the same game in an emulated environment. This applies not only to human-computer interaction, for example, in a control system, but also to differences in sensitivity, drift, or timing behavior for all kind of sensor input.

Environment. Playing a game of Space Invaders standing in front of an arcade machine in a smoke-filled bar creates a different playing experience than the same arcade machine in a clinical museum environment. To recreate the original experience the environment has to be emulated as well, as the focus on such recreations in simulator settings for, for example, safety trainings shows.

When a system component is to be replaced by a different (emulated) component, such evaluations are standard practice.

It should be noted that digital objects represent information objects, and are thus always used in a rendered form (which may be as complex as an entire business, a system control application, an interactive complex media object or video game, a rendered office document, but also a simple XML file encoded in Unicode and rendered as readable character set or interpreted as machine-readable content). Thus, evaluating any kind of digital preservation action needs to be based on the connecting point between interface and environment. In this sense, there is no formal, conceptual difference between emulation and migration from an evaluation perspective, as even migrated objects need to be evaluated within the new rendering environment.

3. RELATED WORK

The UNESCO guidelines for the preservation of digital heritage [Webb 2005] list migration [Marcum 1996] and emulation [Granger 2000; Rothenberg 1999; van der Hoeven et al. 2007] as the main strategies for digital preservation.

Emulation is widely used in information technology for various applications. In software engineering, emulation is used as a programming technique if the architecture of the system on which the software is developed is different from the architecture of the target system [Dumke 2003]. Especially for developing software for hand-held systems or systems which do not have a development environment on the system itself (e.g., video game consoles) the software is developed and its function is verified in an emulated environment on the host system. In Devanbu and Stubblebine [2000] the use of emulation for detecting and contravening software self-destruct approaches by running the software in an emulated environment in a security context is explained. Emulation is also an option to keep legacy software running in production environments after upgrading to a new system if no new version of the software is available yet. Fidge shows how emulation is used to replace embedded microcomputers [Fidge 2003]. An emulator program is interposed between the legacy software and a replacement processor, so that the old code can be reused on the new machine. Kleiner describes how emulators for arcade machines can bring video games back to life on modern systems [Kleiner 2001]. A wide variety of emulators is available today that let users reexperience classic video games on current hardware[4].

It is, however, not immediately obvious if all the significant properties of a digital object rendered in an emulated environment are sufficiently reproduced. For digital preservation purposes it is necessary to compare the effects of different preservation actions on the significant properties of digital objects. When migrating a digital object

[4]http://www.emulator-zone.com/

to a different format, characteristics of the object are usually extracted from both file representations and compared.

In Becker et al. [2009] a preservation planning workflow that allows for repeatable evaluation of preservation alternatives is described. This workflow is implemented in the preservation planning tool *Plato* [Becker et al. 2008a]. As part of the preservation planning, automatic characterization of migrated objects using tools like Droid [Brown 2008] to identify files is used. Migration results can be validated and automatically supported by the eXtensible Characterization Languages (XCL) [Becker et al. 2008b]. The original and migrated objects are hierarchically decomposed and represented in XML. These representations can be compared to measure the effects of the migration on the digital object.

In an emulated environment the digital object is unchanged. Thaller suggests to separate the information contained within a file from the rendering of the information [Thaller 2008]. 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 and feel* aspect of an object. In Guttenbrunner [2009] challenges for the evaluation of emulation and the rendering aspects of digital objects are analyzed.

Emulation as a strategy for digital preservation is seen as one of the main solutions for preserving interactive content. Projects like KEEP[5] concentrate on researching a common platform for emulators. A case study to compare different approaches to preserve video games with one of the approaches being emulation was reported in Guttenbrunner et al. [2010a] on a human-observable and thus to some extent subjective level. Emulation was successfully implemented for the preservation of interactive media on an interactive piece of art as described in Jones [2004]. In Bonardi and Barthélemy [2008] examples for the fragility of performace works based on electronics under the aspect of reperformance are provided and the question is raised, how to guarantee authenticity when preserving the electronic material. Matthews et al. list some categories of significant properties of software depending on rendering like user interaction and object performance, which have to be considered when evaluating different renderings of software [Matthews et al. 2008].

Gladney and Lorie [2005] discuss how to encode digital objects for storage over a long term. Using a specified Universal Virtual Computer (UVC) digital objects can be encoded now and decoded on a future platform running the UVC. Gladney and Lorie [2005] show how the UVC handles static data objects. They also give suggestions about preserving interactive objects using emulators running on the UVC. The output of these emulators can be compared to the original machine now, thus making sure that the emulator behaves like the original system while the original system is still available.

But how can we compare an emulator of a system with the original system to find out if the rendered result in a new rendering environment preserves all the significant properties of the digital object rendered on the original system?

Most approaches to evaluate the validity of emulators as a preservation strategy are currently based on users manually reviewing the emulation results. In the CAMiLEON project [Hedstrom et al. 2006] 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. Doyle et al. present an emulation framework for preserving 3D objects in Doyle et al. [2009]. The evaluation

[5]http://www.keep-project.eu/

of the framework was carried out by having a group of anthropometry experts as well as a larger group of general users compare the emulated environment to the original system side by side, completing predefined tasks and filling out a questionnaire.

A manual comparison of original and emulated environment is a time-consuming and very partial process. It would have to be repeated whenever new emulators or new versions of existing emulators are to be used. Therefore it is necessary to develop a methodically solid approach which allows one to determine the effects of an emulated environment on the rendering of objects and to automate the process of evaluation to some extent. In Aitken et al. [2008] the Planets Testbed that provides a controlled environment for evaluating preservation actions supporting the automated evaluation of experiments for migration actions is described.

In Guttenbrunner et al. [2010b] 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. In this article we refine the concepts shown in the case studies to different levels of extraction of information about the rendered object. We discuss methods for applying interactive communication with the object as well as a workflow for measuring rendering environments in general.

4. DESCRIBING THE DIGITAL ARTIFACT

Knowing about the behavior of the digital artifacts under evaluation is essential to determine whether the behavior is properly reproduced by the rendering environment. In this section we will explain some key characteristics of a digital object and influences on its behavior that have to be gathered to properly evaluate its rendering process.

It is also necessary to explicitly document which characteristics of a digital object are relevant for a given preservation scenario, that is, which of these constitute significant properties. The degree of preservation of these in an emulated environment ultimately forms the basis for the evaluation and comparison of the performance of different potential emulated environments as part of the preservation planning process [Becker and Rauber 2011b]. These significant properties are collected as part of the requirements analysis phase for a digital preservation endeavor and may be documented, for example, in the form of an objective tree, grouped into different categories. Several possible preservation actions, such as different emulation environments, may then be used to render an object and evaluate its performance. A range of properties usually need to be considered that can be structurally subdivided in object- and action-specific properties, that is, those pertaining to characteristics of the digital object (e.g., reacts correctly to interaction, performs calculations identically to the original, displays colors correctly) and those pertaining to the action (emulation or migration) applied (e.g., emulation environment is open source, memory consumption, required hardware platform, licenses). (See Becker and Rauber [2011a] for a detailed discussion and examples of such properties and means how to measure their performance.) In the following two subsections we want to focus on two object characteristics that are particularly relevant for dynamic objects and evaluation settings that tend to call for emulation strategies as suitable preservation actions.

4.1. Determinism of the Digital Artifact

Before trying to evaluate the behavior of the digital artifact in original form and under emulation it is necessary to understand the impacts of internal and external events on the object's behavior. For evaluating the differences in behavior of an object in the original environment and an emulated environment it is necessary to ensure that the object behaves the same under the same external conditions. Otherwise it is not possible to determine if changes in behavior are an effect of the emulated environment.

4.1.1. Deterministic Behavior. Lamport and Lynch [1990] describe deterministic algorithms as "algorithms in which the actions of each process are uniquely determined by its local knowledge" [Lamport and Lynch 1990]. A deterministic algorithm is independent from external influences like user input, hardware values, random values, or other concurrent algorithms modifying the same data as the algorithm. Based on this, we can define *deterministic behavior* as the behavior of an object that is influenced only by the input that is applied to the object when it is invoked. The rendering of the object has to be independent from the factors listed earlier for the object to be considered as having deterministic behavior.

Some digital objects can be rendered with a deterministic behavior. A spreadsheet or database application will usually, under the same software and hardware conditions and after applying the same input, show the same results. Replaying a video or audio file has to produce the same results independent from the external influences listed before. A comparison of emulation effects would in this case be rather simple.

4.1.2. Nondeterministic Behavior. If the behavior of a digital object depends on user input, hardware, or random values, or has the data it uses modified by other processes, then it is not deterministic. To allow an automated comparison of two renderings of the same object it is necessary to try to create a deterministic behavior. The rendering of the digital object can be made semideterministic by identifying the external events influencing the object's behavior and equalizing the values for these events in the different environments.

Applying automated user input at the exact same moment in the rendering process can be used to eliminate differences in behavior depending on user input. Random number generators are usually initialized with values derived from hardware values like system time or the position of the raster beam on the screen. These external influences to the digital object can be made predictable by setting the system clock in the emulation environment to the same time as on the original environment during execution of the digital object and by using a pixel-exact emulation of the original system, that is, updating every pixel on the screen with the corresponding CPU cycles.

Especially video games or interactive digital art usually heavily depend on external or random elements. A recent example of a video game with unusual external events is metro-wardive[6] for the Nintendo DS/Apple iPhone which uses wireless LAN waves to create in-game objects.

4.1.3. Testing an Object for Determinism. To give hints about the deterministic or nondeterministic behavior of an object, the same series of tests with the object (user input, other external events) can be applied to the object. For a deterministic object the resulting behavior has to be reproducible. This process can be carried out automatically in a test environment.

4.2. Significant States of a Digital Artifact

When evaluating the result of emulation, two entirely different situations need to be considered, depending on the type of result to be evaluated: In the first case, only the target state of a digital object as result of a process is to be evaluated, such as running a calculation or displaying a static object. In the second, probably more common case when emulation is chosen as a preservation strategy, a sequence of

[6]metro-wardive for Nintendo DS/Apple iPhone—http://www.and-or.ch/wardive/.

states is essential, such as rendering dynamic objects, or when emulating interactive processes.

With emulation and dynamic objects one can deal with two different situations, namely a target state of the object where it is only necessary to capture the properties of the rendered object after some period of emulation and a continuously changing state of the object, which again may be conceived as a series of designated states. It is necessary to determine which states of the digital object are significant for the designated user community of the object types for which the rendering environment should be used.

4.2.1. Target State. An example for rendering an object to a target state would be loading an interactive spread sheet, applying some input events which result in changes in the state of the spread sheet. Once a defined set of actions is applied to the spread sheet a certain output is expected. This output can be considered as the target state and then be compared. One possible solution to a comparison is comparing a screenshot of the target state. Another option is to save a document after applying the input and to compare the file saved on the original system with the file saved in the emulated environment, thus evaluating the effects of handling the object in the emulated environment. This way of comparing can be used only if the target state is considered significant and not the states in between. The target state can also be the state reached after the emulated environment initially rendered the digital artifact without applying any input that influences the object's behavior.

4.2.2. Continuous Stream. If a continuous sequence of states has to be compared it can either be done by recording a stream of the output (e.g., image or sound stream) on the chosen layer (as will be discussed in Section 6.2) and comparing the resulting streams. Besides, for example, aspect ratio, brightness settings, changes in color, and audio frequency also the speed factor has to be taken into account. Images on the stream from the emulated environment can be slowed down or sped up, and not necessarily by a fixed factor over the whole stream (e.g., more objects on the screen can potentially slow down the emulation at some point). Advanced comparison mechanisms have to be used—probably with human interaction—to link certain landmark events in the stream (e.g., change in scenery in a video game from selection menu to in-game graphics).

4.2.3. Series of States. In some cases not only a target state of a digital object is sufficient, neither is perfect emulation of the entire sequence of steps required. Sometimes, only a subset of states in between is of interest for evaluation. If the emulation process in between is not of interest, single "snapshots" of these states can be taken and compared (e.g., if the way how a series of images is computed and rendered is not important but only the resulting images are of interest for the evaluation, or states reached during a simulation computation). By using state machines describing the states in between and a sliding window technology to apply a window for each of the rendered versions of the digital object, comparison can be done for each state. An example would be an image viewer application which switches between images on user input. It can take considerably longer in the emulated environment to show pictures when using special hardware effects to switch between images which are not supported by the emulation environment. If only the rendered images are of interest for the emulation, then only the starting state, the state before applying input (when the image is fully shown on the screen) and a target state have to be compared.

Table I. Example Digital Objects and Possible Hardware/Software/Output Device Combinations

Digital Object	Software	Hardware	Output Devices	Example Usage Scenarios
Defender	(no other software required)	Williams arcade machine	built-in screen and speakers	no variations possible
Super Mario Bros.	NES operating system	Nintendo NES	various TV screens	small b/w CRT-TV, color CRT-TV, LCD TV
First Finnish Underground (Interactive Art)	Microsoft Windows 95	Intel x86 PC	various output devices (e.g. monitors, video projectors)	Museum setting with a normal CRT screen, presentation in an auditorium using a projector
Website	Various different browsers and operating systems	Various hardware platforms	all kinds of display devices (e.g. monitors, mobile devices, video projectors)	Lynx text-based browser on Unix system with terminal output, Internet Explorer on Microsoft Windows 95 PC with CRT monitor
Signed PDF Document	Various PDF viewers Various operating systems	Various hardware platforms	all kinds of display devices (e.g. monitors, mobile devices, video projectors)	PDF Reader on a mobile phone, PDF Reader on a PC and connected to a projector for a presentation

5. DESCRIBING THE RENDERING ENVIRONMENT

Not only the digital object, but also the environment in which it is executed has to be documented and recreated for a faithful reproduction of the rendering process. In this section we describe the elements in the object's viewpath that potentially influence rendering results as well as methods to create uniform interaction that has to be applied to an interactive digital object by a user.

5.1. Selecting the Reference Rendering Environment

Identifying the correct rendering for a digital object means that we have to determine what the desired rendering we want to compare against actually is. A digital object can potentially be executed in a wide variety of hardware and software platform combinations, depending on various usage scenarios, and the resulting output can be played back on a potentially unlimited number of output devices. That is, most objects will have several (equally authentic) renderings usually targeted at specific designated communities (different users or usage scenarios). Some examples are shown in Table I.

In the least complex case the digital object can be rendered only on one hardware platform with predetermined software and built-in output devices. In the most complex example a digital object can be rendered by a variety of software products running on different operating systems, which in turn can be used on different hardware configurations, resulting in a variety of rather different information objects. Depending on the usage scenario the correct rendering environment of an HTML page may be anything from a Web browser via a simple HTML editor to an entire content management system. A specific XML data file may be rendered in an XML editor, being used to drive an application as input configuration, or used as basis for a visualization. The rendered output can be played back on a number of different output devices with different characteristics (e.g., CRT-TV versus LCD-TV).

Determining which of those combinations will be used as a *reference environment* is a core step in describing the digital artifact. It may even include the definition of several reference environments satisfying different stakeholder needs, which may be served by a single or multiple rendering environments. In an ideal case the rendering on these reference environments can be used as a ground truth to compare alternative rendering environments. When defining the reference environment, the following factors should be taken into account.

— The hardware and software configuration typically used to render the digital object (e.g., configuration of a standardized office PC used to create the object) must be taken into account. A repository of highly standardized typical system configurations may be helpful to minimize the effort in maintaining the rendering environments. Yet, in some cases, the very specific configuration encountered in a setting will need to be defined as reference environment.
— Another factor is the play-back devices typically used to observe the digital object (e.g., a CRT TV used to render a video game from the 1980's instead of a modern LCD screen with completely different display characteristics, or synthesized voice output on contemporary mobile phone speaker system versus studio-quality speakers in a museum setting).
— The usage scenario, that is, what purpose(s) the object is being preserved for must be considered. As different scenarios (e.g., simple replay, providing different views on data, or providing evidence in legal investigations) may require different renderings of an object, potentially multiple reference environments need to be defined.

Observing the digital object in its original environment is only possible if the reference environment can be obtained in working order using the original hardware, software, and play-back devices. This is when an original rendering environment will still be available, and when—as part of the documentation of the object and its rendering environment—evidence for measuring the significant properties of an object's rendering in a potentially different environment has to be collected. In cases where digital objects are unearthed at a later stage (e.g., through digital archeology on obsolete media formats) the ground truth has either to be set empirically by comparing either different rendering environments for a system (e.g., if five out of six emulators render an image in white/blue and the rendered image are clouds on a sky, then it is very likely that the colors are rendered incorrectly on one of the emulators) or by using other sources like software design documents such as use case scenarios, requirements specification, and software testing as well as video documentation.

5.2. Describing the Viewpath of a Digital Artifact

The resulting observable form of a digital object depends on the hardware and software used to render it. Also the configuration of both hardware and software plays a crucial role. To compare the results of rendering objects in their original environment and in emulated environments it is necessary to collect and document as much information about the original environment as possible.

In van Diessen [2002] the viewpath is described as "a full set of functionality for rendering the information contained in a digital object". The viewpath contains the hardware and all the secondary digital objects needed to render an object along with their configuration. It is possible to use different viewpaths to display the same object as shown in Figure 2. On various hardware configurations different operating systems (e.g., WinXP, Linux) can be used to run different applications (e.g., Word, Open Office) to render the same object (e.g., Word-97 document).

It is practicable to define one viewpath out of all the possible ones to compare the effects of emulation, even though different viewpaths would be possible. This is the only way to make sure that the difference in rendering is a result of the emulation and not a difference due to a different viewpath. The same Microsoft Word document can be rendered on the same hardware and the same operating system with an identical

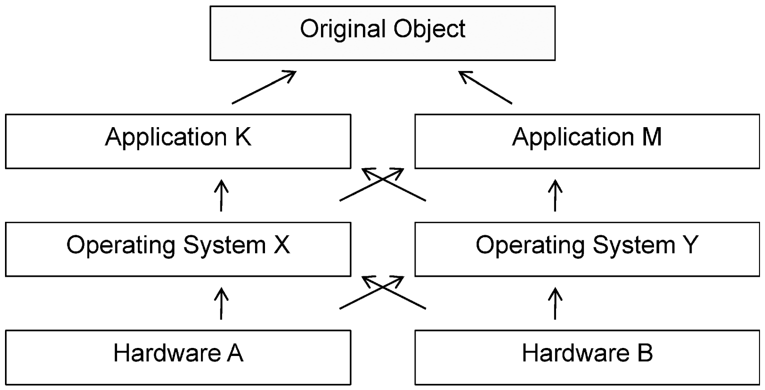

Fig. 2. Different viewpaths for displaying the same object.

set of fonts installed using different versions of Microsoft Word or using Open Office which all can give different rendered results for the same document.

The information that has to be collected can be split into hardware configuration and all the necessary software to render the digital object with their respective configuration. The following sections provide an overview of the various elements that need to be documented and recreated or standardized in an emulation setting in order to facilitate proper evaluation. The focus of this description is not to provide an exhaustive list, but to illustrate some code aspects as background for the evaluation framework presented in Section 7.1.

5.2.1. Hardware Configuration. Besides the obvious CPU type and configuration as well as memory size, configuration, and speed, other hardware components influence the rendering of an object as well and have to be considered, such as the following.

— *Graphics card, physics card*. Especially for 3D rendering the used graphics and physics cards are having a major influence on the resulting rendered images. Also to be documented are the settings of device drivers.
— *Sound card*. The sound processor that is used for creating the audible output has an influence on the characteristics of the output audio signal.
— *Input Devices*. Special input devices may be necessary not only to recreate the original look and feel, they also may be capable of recording all input activity to provide identical input sequences in the emulated environment.
— *Output Devices*. The output device plays an important role on the resulting image/sound just as much as the rendering does. Aspect ratio and display settings of an output device have to be considered if the resulting output is compared not after computing as a digital image, but after creating an actual analog or digital output.

Of course this list cannot be complete and depends on the hardware of the system that has to be documented and emulated. For manufacturing or control application, this can include sensors and actuators, on older (home-) computers or video game consoles it also includes additional processing units (e.g., Sega 32X) or memory expansion cards (e.g., Commodore Amiga, Atari 800).

On some platforms tools can assist in the description of hardware properties by determining hardware and software settings. A platform-independent tool that supports the process of creating a list of properties is the "System Information Gatherer" (Sigar)[7].

5.2.2. Operating System and Configuration. The operating system type and the version including all system updates have to be documented. Usually, operating system settings will have an effect on rendering as well. The screen resolution and color depth have the biggest influence on rendering. Other factors influencing the rendering are the installed fonts, appearance settings, color schemes, and certain installed utilities or applications.

5.2.3. Secondary Digital Objects. Other digital objects besides the operating system are needed to display the digital artifact itself or for running the viewer or editor application displaying the object. Some examples of these are as follows.

— Virtual Machine (e.g., Java Virtual Machine, .net)
— Database software (e.g., MySQL)
— Libraries (e.g., DirectX, Allegro[8])
— Software device drivers (e.g., ODBC drivers)
— Viewing or editing application (e.g., OpenOffice, PDF-Viewer)
— Fonts (for documents where fonts are not stored inside the file)
— Codecs (for decoding, e.g., audio or video data streams)

For all these secondary digital objects, not only the version, but also configuration options have to be documented to allow the complete recreation in an emulated environment.

5.2.4. Digital Artifact to be Rendered. The actual digital artifact that has to be preserved is also part of the viewpath. If this artifact is software, then the options changed from a default setting of a specific version have to be documented. Not only options that influence the rendering of the object (e.g., window size, font size, colors), but also options influencing the behavior have to be considered (e.g., notifications).

5.2.5. Additional Digital Objects Not in the Viewpath. Besides the mentioned elements in the viewpath of an object it is important to be aware of any other digital objects influencing the behavior of a computer system. Processes running in the background (e.g., virus scan software, remote desktop software) can significantly affect the performance of a system. The influence of semirandom elements on the execution of the digital artifact or on an application rendering the digital artifact has to be reduced to an absolute minimum to decrease the complexity of the system under evaluation.

5.3. Collecting Test Data

To be able to verify the performance of an object in an emulated environment a set of input-output data relationships has to be collected. This can range from static I/O data relationships via system reaction documentation for specific types of input to complex documentation of system behaviors. A potential source for this kind of information may be system design documents such as use case scenarios, requirement specifications, or system testing documentation. It may also include the capturing of system behavior in other documentary form such as video documentation.

[7]Sigar—http://www.hyperic.com/products/sigar.html.
[8]Allegro game programming library—http://alleg.sourceforge.net/.

5.4. Identifying Forms of Interaction with the Environment

As emulation is especially important for the preservation of interactive digital artifacts, it is important to evaluate whether the interactive parts of an object can be accessed in an emulated environment. Depending on the type of digital artifact, it is more or less crucial to apply the same input to get the same results. Input in this context means any interaction with the environment, for example, clicking a hyperlink on an HTML page, pressing a button in an application, and using the mouse to control a character in a video game, but also system interactions such as polling for triggers in a Web service setting or reacting to data feeds. While a few pixels and a few seconds differing from the interaction in the original system/environment may be irrelevant when clicking on a button in a Word document viewing setting, it will make a huge difference in a control system reacting on sensor input or a computer game. But also for automatically measuring the efficiency and effects of the emulation on the rendered object it is essential to determine whether a time difference in events occurs due to the emulation and not due to delayed interaction.

It is virtually impossible to manually apply the same interaction twice to a system, even if it is done shortly after the original interaction and side by side with the original system. If interaction has to be applied years after applying it on the original system (e.g., to test the validity of newly developed emulators), it will be even harder. Some methods of applying automated interaction to a digital artifact are described next.

5.4.1. Use of Macros. Recording a set of user actions along with the delay between them is one way of automating input to digital artifacts. One of the downsides is the difference of recording on one system and replaying on the other system, which is a change in conditions under which the system is running. By running a macro in the emulated environment a change in speed will not affect the execution of the macro, as the software replaying the macro also runs at a changed execution rate. The use of macros is only possible on operating systems where processes can run parallel to the execution of the viewpath of the object. It would not be possible for the preservation of games on video game consoles or for applications on most home computers or personal computers running simple disc operating systems. In some cases the digital object provides support for applying interactive options automatically either through a scripting language or by recording macros. One example would be Microsoft Excel which allows both the recording of macros and offers a scripting language to automate input. Another example is the game Quake[9], which allows the recording of user actions and is able to replay them in the game engine. Note that the granularity of timing is different for the two examples presented here: while differences in a few frames might not make a difference for the automated input in Excel, it might make a difference for the player's survival in Quake. On real-time systems even a tick-based granularity can be necessary.

5.4.2. Remote Access. By using remote access to the original system as well as the emulated system (e.g., at the time of evaluation during a preservation planning process) it would be possible to simulate the same input to both systems with the same amount of interference in both systems. The system from which the remote access is established will replay the same interaction macro to both destination systems. The downside of this method is a change in execution speed. If the emulated target system runs at a different speed than the original system, the input will come either too early or too late.

[9]Quake by id Software—http://www.idsoftware.com/games/quake/quake/.

5.4.3. Controller Applications. One problem which arises by using macros is a potential random (nondeterministic) behavior as described in Section 2. While this is usually not an issue with interactive software or interactive documents, it plays a major role for most games and for interactive art. By using controller applications that react to the digital object's behavior (e.g., by analyzing the rendered form of the digital object in the environment) it will to some extent be possible to apply automated input. A controller application runs either in the same environment or through remote access. It will handle the input/output communication with the digital artifact. This application can be used on the original system as well as within the emulation environment. The advantage is that, contrary to macro-based input, the output of the digital artifact can be taken into account when having a controller application. Still, the same limitations as with macros apply. It is necessary to execute the application parallel to the viewpath of the object that has to be preserved. Another disadvantage of a controller application is that it has to be customized for each type of digital object which has to be preserved and as such requires a huge effort compared to the benefits.

5.4.4. Recording and Applying Input on a Hardware Level. The method of recording and applying input with the least side-effects on the environment is at the hardware level: recording the input directly from the input devices and applying this as an input macro to the emulator. This kind of input macros contain the event (e.g., key X pressed, joystick left) as well as a time relative to elapsed process execution time between these events. One advantage in the use of macros recorded from the input device and applied to the emulator is that they can be used on systems that do not allow parallel execution of processes (old home computers, video game consoles, embedded systems). It is also possible for the emulator to adjust the timing of these input events depending on the speed difference between original environment and emulated environment. The disadvantage is that it is not possible to use only software components but that special hardware has to be built to intercept the communication of the input devices and the target system. A possible solution in software would be to use an emulator which is known to work nearly perfectly and use an input recording feature of the emulator to capture the input events. Right now emulators are usually not able to record or replay input. This feature and a standardized format are one of the requirements that have to be researched for future emulators as a digital preservation requirement. Ultimately, hard real-time system-like behavior [Liu 2000] is necessary to ensure perfect process-state-level execution of emulations.

6. RECREATING THE RENDERING ENVIRONMENT

Once the digital object, the influences on its behavior, and the environment in which it is originally rendered are documented, it is necessary to faithfully recreate these conditions in a rendered environment. In this section we first describe how the original viewpath has to be restored. After recreating the original environment in a rendering environment, it is necessary to decide where to extract the rendered form of a digital object from for comparison. A digital object is rendered to different incarnations inside the emulation environment, for example, in the main memory, or on the output device. Next we take a look at how to extract information about the rendering process from the rendering environment.

6.1. Recreating the Viewpath

Based on the documentation of the original rendering environment it is essential to recreate an emulated environment which can replicate this original environment.

As emulation is usually done on a hardware level, the emulator used to create the setting has to support emulation of the hardware components found in the original

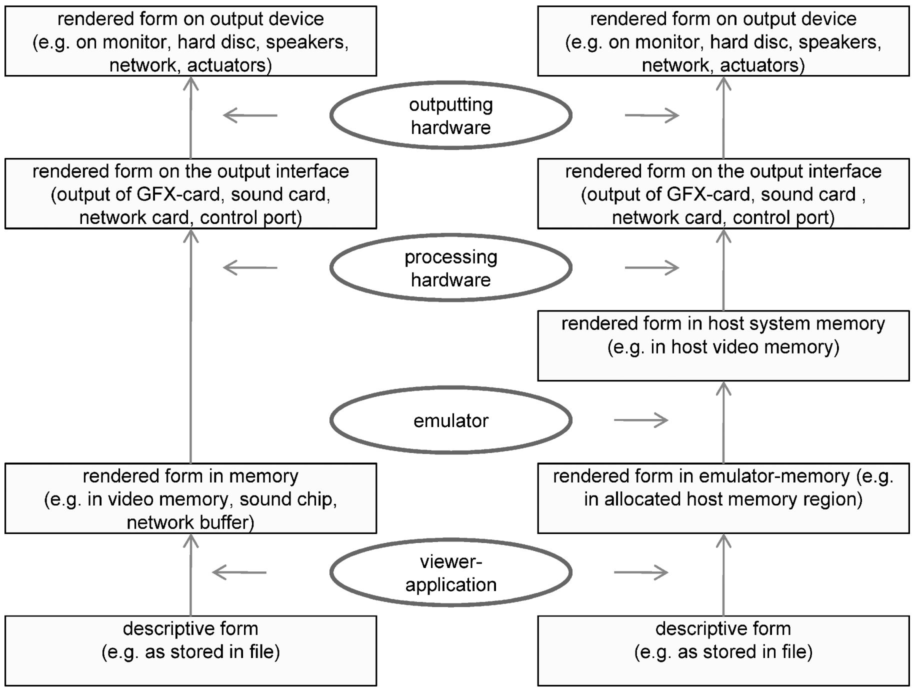
rendered form on output device (e.g. on monitor, hard disc, speakers, network, actuators)
rendered form on output device (e.g. on monitor, hard disc, speakers, network, actuators)
outputting hardware
rendered form on the output interface (output of GFX-card, sound card, network card, control port)
rendered form on the output interface (output of GFX-card, sound card , network card, control port)
processing hardware
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)

Fig. 3. Different forms of a digital object in a system's memory. On the left the layers in an original system are shown, on the right the layers in the system hosting the emulator are shown.

system. If the exact same hardware configuration cannot be rebuilt in software, this has to be documented and the effects of replacing various parts with alternative parts have to be considered when comparing the results.

Recreating the viewpath in the emulated environments has to be done following the documentation of the original system to ensure that potential differences in the emulated environment are a result of the emulation process and not of side-effects of a difference in the setting. If it is possible to create a complete image of the viewpath on the original system, we advise to use this in the emulated environment (e.g., floppy disc for home computers containing the disc operating system and the viewer application, hard disc image containing a preinstalled operating system with all the necessary drivers and applications).

An example for the use of emulators that uses preconfigured images to create emulation environments for displaying digital objects is GRATE (Global Remote Access to Emulation Services) presented in von Suchodoletz and van der Hoeven [2008]. It is a framework which allows remote access to emulators running hard disc images and injection of a digital object into the emulation environment.

6.2. Identifying Levels of Comparing Rendered Forms of a Digital Object

To compare the effects of rendering an object in an emulated environment to the rendering of the same object in its original environment, it is essential to understand the different incarnations of a digital object when it is being rendered. These forms of a digital object on different levels are shown in Figure 3. A rendered representation of the digital object has to be extracted on (a) suitable level(s). The significant properties of the object can then be evaluated.

6.2.1. Descriptive Form. When using emulation as a digital preservation strategy, the original binary object is not changed like when using a migration strategy. Thus, the descriptive information used to render the object stays the same unlike when the object is migrated to a new manifestation. In the case of migration, this constitutes the first (and albeit frequently the only) level of comparison to evaluate the quality of migration actions. Yet, as the ultimate goal of digital preservation is not the preservation of the descriptive form of a digital object as a static file encoded in whichever way, thorough evaluation will usually need to consider comparisons at levels further up the viewpath hierarchy according to the given application scenario.

As the descriptive information of the object cannot be used directly to evaluate the emulation effects, an already rendered form of the object has to be used.

6.2.2. Rendered Form in Memory. The next level where an incarnation of the digital object exists is as a rendered form after being processed by the viewer application (or after being executed). On this level, for example, the registers of the video hardware are filled or the memory area for video output is filled with the data to be displayed on screen. If the emulation of the original system is processing the supplied information properly, then the rendered form in the memory area used by the emulator to store the rendered information has to be identical to the rendering in the same memory areas of the original environment. This is the first layer on which a comparison of emulation effects is possible. Screenshot or video memory dump applications on the original system and on the emulated system will produce the same output after executing the same actions on both systems. The same applies to other forms of object output, that is, audio, port communication, or file system interaction.

This rendered form of the digital object can be used to evaluate the internal processing of the object but not the ability of the emulator to translate the output of the original system to the host system's environment. It also is necessary to install applications running on the original system and the host system that can capture the rendered form of the object (e.g., as dump of specific memory regions or in the form of memory "screenshots" taken inside the emulated environment).

6.2.3. Rendered Form in Host System Memory. Further on in the rendering process the emulator has to convert the rendered form from the emulated environment to a rendered form in the host system environment. In this step the emulator has to, for example, render the video output of the emulated system for display on the host system. To do so, for example, the resolution of the image may have to be altered to fit either a resolution which can be displayed on the host system or by adjusting the image to the size of a window on the host system. In this step the resulting image will already be radically different than what it was on the original system, even if it looks similar for a human observer.

No direct comparison is possible at this layer as there is no similar information in the original system.

6.2.4. Rendered Form on the Output Interface. The data in, for example, the video memory is processed by the hardware converting it into a suitable signal for the displaying unit. This signal exists on both the original system as well as the system hosting the emulator. The resolution and type of the displaying unit are potentially different, but using capture devices recording the input directly from this source and transforming them to the same signals would be possible. An example is a video capture device that can take different kinds of inputs and display them the same way on the host system running the capturing software. Similarly, for networked systems, differences in internal protocol structures of the network interface, or instruction sets driving a storage unit need to be considered.

By recording the signal of the emulating system and of the original system the results can then be compared. The effects of the rendering device on the signal have to be considered as well (reduction of frame rate, delay in processing, enhancing of the signal properties, etc.)

6.2.5. Rendered Form on Output Device. The final level on which a rendered form of the digital object is available is the resulting output on the output device (e.g., the image displayed on a screen, sound emitted by a loudspeaker, voltage levels present at a port, or binary sequence present on a storage medium). This has to be captured by devices recording the signal after it is processed by the output device (e.g., the image on a TV-screen with a camera or speakers by using microphones). This way the resulting representation in the recording device converts the output to the same resolution.

A comparison of the rendered object is possible at this layer. The settings of the output devices have to be taken into account. For example, by comparing the resulting images from an original system connected to a TV-screen and a host system running an emulator of this system, the aspect ratio, the brightness settings, contrast settings, etc., will usually not be the same. In Phillips [2010] the author describes the influence of the output device (in this case a CRT monitor compared to a modern LCD flat panel) on the resulting image. Pixels that have sharp edges on the LCD display were supposed to be displayed as blended pixels on the original screen due to technical characteristics of the two different screen technologies. Similarly, the audio quality offered by different speaker systems, or the preciseness of a storage media writing device may be different, resulting in a different rendering of an object in its environment even if the underlying system memory representations were identified.

More systematically, each transition from one level to the next represents interfaces between systems, both on the original as well as the recreated viewpath. The properties of these systems and their effect on the viewpath thus need to be considered, both individually as well as on an integrated level up to the desired representation level.

6.3. Comparing Objects

Note that different characteristics of an object can be measured at various levels. Thus, comparison may be required at several levels to determine how faithful the rendering of an object ultimately is. We will now take a detailed look at these levels, their properties, and specifically means for comparing original and recreated renderings with varying degrees of automation. Also note that each level in the hierarchy corresponds to some kind of computation/transformation being performed on an object. Thus, identity at some level does not imply identity at the ultimate rendering at the top level of final output devices, nor does the fact that an object differs at a certain level mean that it cannot result in an identical rendering at higher levels. For example, consider an object stored for emulation. This will usually be bit-stream identical in its descriptive form, yet emulators or viewers may render the object differently on the output interface level, whereas different output devices may result in further differences on the top level. Conversely, a migrated object will be different in its descriptive form (e.g., a TIFF version of a GIF image), yet may result in an identical rendering on the output interface. In some cases it may be required to specifically produce an entirely different representation at the output interface level in order to obtain a close to identical rendering on the actual output devices. See Phillips [2010] for an example of the effort required to recreate the effects of the fluorescent behavior of analog CRT screens on modern LCD screens.

6.4. Extracting Properties from the Emulated Environment

Not only the resulting rendered form of the object can be compared, but also certain characteristics can be measured in the emulated environment. While with migration it is in principal possible to use tools to examine the resulting object, with emulation these characteristics would have to be provided by the tool doing the rendering, the emulator. While none of the available emulators supports it at the moment, it would certainly make an automation of the process of comparing emulation environments and the original environments easier. First the characteristics and properties of the original system and the digital object have to be described. Then these same properties are extracted from the emulation environment. Ideally the characteristics should be described in a format usable for tools, which can compare these properties in an automated way. Using this approach the process of automating preservation planning for emulators can be similar to the one of automating preservation planning for migration.

Not only system properties, but also screen shots or even a log of events happening on the system can be provided by emulators. These properties can be extracted continuously over the emulation process for either a specified time or until an event occurs. Emulation properties are usually not single-dimension like properties of most migrated objects. An additional time dimension has to be considered. The frame rate can change over time, as more objects can result in fewer frames per second on the output device. Properties also can be extracted at one point in time after applying all the input events or after a certain amount of seconds, frames, or CPU cycles.

Some possible characteristics to be extracted are as follows.

— Frame rate (average, max, min)
— CPU cycles per second (average, max, min)
— Number of files opened on a certain I/O device
— Number of bytes read from I/O devices
— Number of certain input/output events

Not all of the properties make sense for deterministic and nondeterministic behavior, for example, a minimum frame-rate would make sense in both cases whereas a number of files opened can differ if the behavior of the object is nondeterministic.

Usually not all the significant properties of a digital object can be measured automatically. In that case as many properties about the object's behavior in the environment as possible should be extracted from the emulation environment whereas all the properties that are either social (e.g., connected to the feel aspect) or that are too difficult to measure technically have to be evaluated manually.

By defining the properties of the digital object that has to be rendered, measuring and comparing them to the original reference system and setting importance factors (exact speed is probably an issue for a computer game but not for an interactive spreadsheet), emulation alternatives can be compared and the best alternative for a certain scenario can be chosen as described in Strodl et al. [2007].

7. FRAMEWORK FOR THE EVALUATION OF EMULATION EFFECTS

In this section we present a workflow for evaluating the (behavior of) information objects in their rendering environment(s). After presenting the workflow we show some sample scenarios along with the decisions to be taken and finally propose design requirements for developing emulators that fulfill basic needs of digital preservation purposes.

7.1. Evaluation Workflow

To evaluate the degree to which the emulation of an original environment preserves the original characteristics of a digital object in comparison to documentation of its behavior, the concepts presented in this article lead to an evaluation workflow with the following steps.

(1) Describe the original environment
The original system's hardware and software components have to be documented along with all their settings to allow the recreation in an emulated environment as described in Section 2. To reduce the complexity of the system it is recommended to eliminate all unnecessary secondary digital objects (software and hardware) in the viewpath (e.g., no virus scan software or use of standardized minimum OS configurations).

(2) Determine external events that influence the object's behavior
Only for objects with deterministic behavior is it possible to ensure that differences in rendering compared to the original environment are results of the emulated environment. To ensure deterministic behavior of the object it is necessary to investigate what external events influence its behavior and simulate those in the emulated environment (e.g., set a random number generator to the same seed to produce the same sequence of random numbers).

(3) Decide on what level to compare the digital object
As the digital object is available in various rendered forms as shown in Section 5, it is necessary to select the one that is most suitable for the digital objects that have to be preserved and their desired form of representation. Depending on the original system only some of the various techniques may be technically possible (e.g., on early home computer hardware no operating system allows the execution of multiple processes, so no screenshot applications can be installed).

(4) Recreate the environment in emulation
Next, an emulator on a host system has to be configured to match the hardware and software configuration of the original environment. The viewpath of the digital artifact has to be recreated, ideally by using, for example, a hard disc image configured the same way as on the original system. We recommend the use of the same viewpath to eliminate the side-effects of different renderings by using different secondary objects.

(5) Apply standardized input to both environments
Depending on the digital object the most suitable way to apply automated input has to be selected. Then, the input to the original object has to be recorded and applied to the emulated environment. The closer the input is done to the hardware level (e.g., as described in Section 4.4) the fewer side-effects on the rendering of the object it will have.

(6) Extract significant properties
Next, the significant properties of the rendered object have to be extracted from the emulation environment as described in Section 6. Depending on the digital object and if it reaches a target state this has to be done once or in a continuous way. Also, depending on the deterministic or nondeterministic nature of the digital object, only certain properties make sense to be extracted.

(7) Compare the significant properties
Finally, the significant properties that have been extracted automatically as well as those that were not measured automatically but evaluated, manually have to be compared, evaluated, and documented. This may serve as input to a preservation planning process, or serve as evidence for the authenticity and faithfulness provided by the emulator.

Table II. Example Categories and Digital Objects Rendered in Emulation Environments

Category: Example object	**Description**
Static Document: *Signed PDF Document*	— Static PDF document that is signed and cannot be migrated as the signature has to stay intact — Interaction not necessary once the document is displayed — View-path consists mainly of the viewer-application and the operating system
Interactive Document: *Adobe Acrobat 3D PDF*	— Document containing various 3D objects embedded using the 3D PDF Standard — Viewer-application, operating system and 3D drivers (e.g., Direct3D) necessary in view-path — Interaction with the document has to be possible (e.g., rotating a 3D object)
Standalone Application: Assumed *proprietary database application* to manage customer data (Java-application developed by external partner, no source available)	— Conversion to new format not possible as logic and data are not completely separated — Querying the data has to be possible for future inquiries — View-path includes JAVA virtual machine, operating system, application — Interaction necessary for querying data
Distributed Application: *Distributed business process* across various systems communicating over a network	— Application behavior has to be preserved to execute the business process in the new environment — View-path different for every application in the distributed environment — Interaction with at least some components of the system necessary
Scientific Application: *Chemical process simulator* used to provide proving for design of machine tools	— Exact behavior of the application and created data has to be preserved for legal reasons — Experiments taken in the simulation have to be repeatable, so decision making upon the data is provable — Exact view-path has to be stored with the application (operating system, data storage, virtual machine) — Results have to be rendered exactly as presented to the user at the time of decision making
Linked Documents: *website* harvested by web crawler	— View path consists of operating system, browser, all necessary drivers, codecs, browser-plug-ins (e.g., Shockwave Flash) — Interaction necessary to follow hyper-links
Interactive Art: *First Finnish Underground (Krkkinen/Okkonen)* (Intel x86 PC)	— Interactive artwork from 1995 (Ars Electronica collection) — View-path contains specific PC-hardware (e.g., sound card) and needs MS Windows 95 — No random element — Interaction as part of the artwork and experience
Action game with fixed events: *Super Mario Bros.* (Nintendo NES)	— Video game on an early console system — No additional view-path besides the game itself, game runs directly on the hardware — Events in the game happen always at the same moment (enemies appear on fixed positions with repeatable behaviour when the screen scrolled a certain amount)
Action game with random events: *Defender* (Williams arcade machine)	— Video game from 1980 — Runs on dedicated hardware, no additional elements in view path — Random element—even with the same input applied the game's enemies are placed differently — Without re-engineering the game it is not possible to determine the external events making the game non-deterministic
Interactive Fiction: *Zork: The Great Underground Empire—Part I* (Atari 800XL home computer)	— View-path consists of the application-files and the hardware — Interactivity is important to enter commands through a keyboard — No random elements appear — The graphical representation (original font, original colors) may be important depending on the context (story content vs. original appearance)

7.2. Sample Scenarios

By following the steps in the presented workflow it is possible to reduce the side-effects of external events on the rendering of a digital object. Some examples for various categories of digital objects are listed in Table II. For every example object some key features (e.g., viewpath, reasoning for emulation as a preservation strategy, kind of interaction necessary, random elements) are included in the table. In Table III possible decisions and reasoning for, for example, emulation level, application of input, extraction level of manifestation of the object, are shown for each category of objects.

Table III. Example Types of Digital Objects and Possible Test Environment Decisions

Category	Decisions
Static Document	— Emulation on application level possible (viewer) — Original speed is not a significant property — No interaction necessary — Extraction and comparison as a screenshot on video memory level possible — One target state significant
Interactive Document	— Emulation on application level possible (viewer) — Original speed is not a core significant property (within limits) — Interaction through macros possible on original system and in emulated environment — Extraction either on video memory level or after rendering for the host system — Series of target states significant (state before next input)
Standalone Application	— Emulation on operating system level possible — Original speed is not a core significant property — Interaction through macros possible on original system and in emulated environment — Extraction either on video memory level or after rendering for the host system — Series of target states significant (state before next input)
Distributed Application	— Emulation on application, hardware or operating system level possible (depending on the component) — Original speed might be a core significant property if communication requires exact timing — Interaction through macros possible where necessary — Extraction on rendering application interface level (e.g. network layer) — Series of target states significant (different steps in the process where the system is in a valid state)
Scientific Application	— Emulation on hardware level (to not change any of the application's parameters) — Original speed usually not a significant property — Interaction necessary to execute the simulation in the new rendering environment — Extraction either on video memory level or after rendering for the host system — Usually one target state significant (result of the simulation), however intermediary results can be of interest
Linked Documents	— Emulation on application or operating system level possible — Original speed is not a significant property — Interaction through macros possible on original system and in emulated environment — Extraction either on video memory level or after rendering for the host system — Series of target states significant (state before next input)
Interactive Art	— Emulation only on computer architecture level (hardware emulation or virtualization) — Original speed has to be preserved — Interaction through macros possible on original system and in emulated environment — Extraction either on video memory level or after rendering for the host system — Continuous stream of output significant
Action game with fixed events	— Emulation only on computer architecture level (hardware emulation) — Original speed has to be preserved — Interaction with original system possible only on a hardware level — Extraction after rendering through original system (display interface or device) — Continuous stream of output significant
Action game with random events	— Emulation only on computer architecture level (hardware emulation) — Original speed has to be preserved — Interaction with original system possible only on a hardware level — Extraction after rendering through original system (display interface or device) — Continuous stream of output significant
Interactive Fiction	— Emulation only on computer architecture level (hardware emulation) — Original speed has to be preserved — Interaction with original system possible only on a hardware level — Extraction after rendering through original system (display interface or device) — Series of target states significant (state before next input)

As first example for *static documents* a signed PDF document was selected. As the signature and the whole file have to stay unchanged, migration is not the preferred method of preservation. Emulation for this type of document could be done on application level, as a viewer for obsolete documents on a current hardware can be created. Neither speed nor interaction with the document are significant, and only the final target state of the document (the document displayed on a screen) is significant, not the loading process.

The next category are *interactive documents*. A 3D PDF document was chosen as an example for this category. As with the static signed PDF document, emulation can be performed on an application level and the original speed is not a significant property. However, the possibility of interaction with the 3D object in the document has to stay intact. Also, not only the final target state but also a series of target states after applying interaction are significant.

As example for the category *standalone application* a proprietary application with integrated database was selected. In the assumed scenario, data and application logic are not perfectly separated. As it is necessary to perform queries on data in the database in an emulated environment, interaction is significant. Also, the preservation of a series of states (before and after applying interaction to the application) is important, while original speed is not a core issue.

For the category *distributed application* we considered a distributed business process. Different applications involved in the process can be replaced either at different points in time while keeping the interface intact or at the same time by preserving all of the distributed system's components at once. Rendering of the replaced components can be done either on an application, operating system, or hardware layer, depending on the type of application. If single components of the system are preserved using different environments, it is necessary for the rendering environments to provide the interfaces of the rendered environment to the environment they are executed in (e.g. the network layer), to keep the communication between the different components of the distributed application intact (communication with the world "outside" the rendering environment). Timing will be an issue to synchronize activities between system components, unless clear linear workflows are specified without concurrency issues.

As a *scientific application* we chose an application for simulating chemical processes. This kind of application is used in the design process of machine tools. It is important in the event of failure to prove that the design was done considering all facts and having the original data and experiments available. The application and its data have to be rendered exactly the same way as presented to the user at the time the simulation was carried out, so it is necessary to keep the viewpath of the application intact. Emulation has to be done on a hardware level to not change any parameters which might influence the results of the simulation. Interaction with the application is necessary to start the simulation process and examine the data, however, original speed is usually not a necessity.

A Website harvested by a Web crawler is an example for *linked documents*. Interaction is necessary to follow the links between the documents and the rendered pages are the significant target states, while the loading process of pages is not essential. (Note that additional requirements may arise from the actual documents being collected, which may include examples of most of the scenarios listed in this section.)

For *interactive art* we selected an artwork from the Ars Electronica[10] collection as an example. No random elements are included in the program, but the interactivity is a crucial part of the artwork, as the user has to explore the virtual room and all the pieces of the artwork connected in it. As special hardware is necessary to execute the application, so emulation is thus only possible on a computer architecture level. The original speed as well as the continuous stream of output are significant properties of this digital object to maintain the original experience of the artwork.

When it comes to preserving video games one has to differentiate between games with random elements and games which behave the same if the player also behaves the same. As an example for a game that is not really random "Super Mario Bros." for the Nintendo entertainment system has enemies that always appear on the same

[10]Ars Electronica—http://www.aec.at/.

position allowing a game with recorded inputs to replay the interaction on consecutive games and thus comparing the outcome of the emulation process. On the other hand the arcade game "Defender" uses a randomizer to create enemies on different positions in the game every time. It is a complex task to determine the source of randomization (e.g., seed value dependent on time, position of video beam) and create the same behavior. In these cases an emulation on a computer architecture level is the only possible level to preserve the game and the continuous output streams are significant.

For *interactive fiction*[11] we selected the text adventure "Zork: The Great Underground Empire—Part I". While interactivity is important to enter text commands, not the complete output stream is significant, but only the states between the interaction. Depending on the purpose of the preservation the graphical representation of the game can be significant (e.g., for museum purposes to show how early games looked like) or only the game content (e.g., for preserving the storyline of "Zork").

The proposed steps make it possible to evaluate the actual effects that the emulation has on the rendering of these objects compared to the rendering in the original environment. With automated extraction from the emulated environment it also would be possible to automate the process of evaluation of emulators to some extent.

7.3. Design Requirements for Emulation Environments for Digital Preservation

To support the presented workflow it is necessary to create design requirements for emulators that are to be used for digital preservation purposes, as these requirements differ from those used for creating emulators today and the features they offer. Some of these requirements which are necessary to support the evaluation of emulators are as follows.

- *Extraction of significant properties of the digital object, the emulation environment, and the process of emulating the system*: As neither the set of properties nor a format to extract it to is specified yet, it would be necessary to carry out more research in this field. Especially as this is a feature not yet supported by emulators a standard may be devised that can be implemented by emulator-developers to get the properties in a unified format. One example of a characterization language that can be used is XC*L [Becker et al. 2008b].
- *Recording and applying automated input on a hardware level*: For automated and repeatable testing it is necessary to apply input commands to emulation environments. Doing this on a hardware level makes it independent from operating systems. This feature not only eases automated testing but also makes it possible to automate actions for users of objects in emulated environments (e.g., booting a system and starting an application automatically without special knowledge of the handling of the environment on the user-side). As no standard exists so far it is necessary to develop a standard for handling of user input to use the same input definitions in the same format for all emulators that have to be evaluated.

Other requirements that are not directly connected to the testing of emulation environments but that are crucial for the long-term stability and usability of emulators for digital preservation include the following.

- Video game system or mobile platform emulators are generally developed for speed and immediate usage. For portability and long-term stability the use of virtual machines, platform-independent code, or modular development is necessary and recommended. A virtual machine specifically to be used for emulation is currently under development in the EC project KEEP.

[11]Wikipedia—Interactive Fiction—http://en.wikipedia.org/wiki/Interactive_fiction.

— Depending on the intended use of an emulator, the requirements on the emulator are quite different. While a compromise between speed and correctness of emulation (e.g., timing between CPU and output components) usually can be taken into consideration for office applications, a much closer alignment between the internal components is necessary for video game emulators, with the most demanding requirement on correctness being real-time applications. Emulators have to be designed with the intended application in mind.
— Most emulators do not offer a possibility to copy contents from the emulated environment to the host environment. Phelps and Watry [2005] see this as a major drawback for using emulators for digital preservation. Usually it is necessary to be able to access documents and reuse the contents of those documents in the host environment. In most emulation environments data extraction can only be done by using transfer methods like clipboard sharing over the network or by using tools to extract data from disc images. One notable exception is Dioscuri [van der Hoeven et al. 2007]. Dioscuri is an emulator specifically developed with digital preservation purposes in mind and allowing the user, amongst others, to copy data between host environment and emulated environment.

8. CONCLUSIONS AND FUTURE WORK

In this article we presented a workflow to evaluate the emulation effects on a digital object compared to the object's behavior in its original environment. In this section we present the conclusions of the work presented in this article along with how the concepts apply to other digital preservation strategies. Finally we give an outlook to necessary future work.

8.1. Evaluating Emulation

The most important task when evaluating emulators is to reduce the side-effects that occur not due to the emulation but due to external events that are different in the original and the emulated environment. To reduce the influence of these events on the rendering of a digital object, it is necessary to document the original system and its properties as precisely as possible and then recreate the original setting on the host system using an emulator. To make the behavior of the digital object as deterministic as possible it is necessary to apply input in an automated form.

We showed various methods on how to capture input on the original system and apply it to the emulated environment. Depending on the object and the technical possibilities it has to be decided which of the various rendered manifestations of a digital object on the original system and in the emulated environment have to be compared. The selection depends also on the significant properties of the object that has to be evaluated.

Using the workflow shown in this article the effects of rendering an object in an emulated environment can be evaluated. By testing how the significant properties of the object are affected in different emulation environments it is possible to choose the optimal solution for a preservation planning case by comparing them using, for example, the Planets preservation planning approach [Becker et al. 2009].

This article describes the evaluation of rendering effects in emulated environments compared to the original environment. There will be cases where the original environment is no longer available or cannot be accessed with reasonable effort (e.g., data archeology). The concepts shown here also allow for a comparison between different emulated environments. Usually it is possible to get an idea about the rendering in the original environment by evaluating various emulated environments, even if the original appearance is no longer known. For example, if the music in a piece of interactive art is present in one emulated environment and missing in another environment, then it is obvious that this emulator lacks some ability to render audible characteristics of the object.

8.2. General Applicability to Other Preservation Actions

Conventionally, emulation is seen as a distinct type of preservation action, significantly different from migration, standardization, and other approaches. It is attributed with changing the environment rather than the object, and due to its characteristics frequently recommended for (inter)active digital objects. However, at closer inspection this view seems too limited.

Migration focuses on transforming the object from one file format or encoding to another. Consequently, evaluation of migration solutions predominantly focuses on the characteristics of the source and target object type, and in how far these significant properties can be preserved. Yet, this approach to evaluation is falling short of several of the key requirements of evaluating the authentic preservation of digital objects. It basically views the digital object solely in its encoded form, rather than as intellectual object that only becomes such via interpretation via some form of rendering process, for example, opening in a specific viewer software or running in a certain environment.

In reality, when preservation actions are being evaluated, characterization tools that analyze object structure and content are usually combined with a visual evaluation when both source and target objects are opened in viewer software and verified visually. This corresponds to the evaluation approach presented in this article, focusing on a single target state. In fact, every intellectual object in digital form needs to be evaluated with its entire viewpath, even when this may consist of any of presumably interchangeable standard viewers such as Adobe Acrobat for PDF or any of the myriad of image viewers. There seems to be a tacit understanding that in the case of static and standard object formats it is only the object structure that is important to be evaluated rather than the combination of object and recommended rendering environment—although experience and best practice teach the opposite. We thus argue that the approach presented in this article is not only recommended for interactive content and in combination with emulation and viewer approaches to preservation, but basically applies to the evaluation of any preservation action taken.

8.3. Applicability and Future Work

Following the framework shown in this article it is possible to compare different renderings of the same digital object in different environments. The framework supports institutions in taking an informed decision on a rendering environment in a preservation planning process. As one example a video game museum can use the framework to select the best emulator for certain digital objects in its collection. In an industry setting the execution of (the software components of) a business process can be evaluated to ensure that the rendering is authentic with respect to the properties identified.

With the concepts provided in this article it would also be possible to automate the process of evaluating emulators to some extent. However, emulators today lack some of the features necessary for supporting automated evaluation, like the possibility to supply files for automated input or the extraction of significant properties of the environment and the digital object. By following the design requirements for emulators for digital preservation purposes as proposed in this article, standardization and to a certain extent automation of evaluation of emulators will be possible. It is also necessary to develop a measurement framework that supports the extraction of properties from rendering environments and the automated comparison of rendering results.

REFERENCES

AITKEN, B., HELWIG, P., JACKSON, A. N., LINDLEY, A., NICCHIARELLI, E., AND ROSS, S. 2008. The planets testbed: Science for digital preservation. *Code4Lib 3*.

BECKER, C. AND RAUBER, A. 2011a. Decision criteria in digital preservation: What to measure and how. *J. Amer. Soc. Inf. Sci. Technol. 62*, 6, 1009–1028.

BECKER, C. AND RAUBER, A. 2011b. Preservation decisions: Terms and conditions apply. Challenges, misperceptions and lessons learned in preservation planning. In *Proceedings of the ACM/IEEE Joint Conference on Digital Libraries (JCDL'11)*.

BECKER, C., KULOVITS, H., RAUBER, A., AND HOFMAN, H. 2008a. Plato: A service-oriented decision support system for preservation planning. In *Proceedings of the ACM/IEEE Joint Conference on Digital Libraries (JCDL'08)*. ACM.

BECKER, C., RAUBER, A., HEYDEGGER, V., SCHNASSE, J., AND THALLER, M. 2008b. A generic XML language for characterising objects to support digital preservation. In *Proceedings of the 23rd Annual ACM Symposium on Applied Computing (SAC'08)*. Vol. 1. ACM, 402–406.

BECKER, C., KULOVITS, H., GUTTENBRUNNER, M., STRODL, S., RAUBER, A., AND HOFMAN, H. 2009. Systematic planning for digital preservation: Evaluating potential strategies and building preservation plans. *Int. J. Digital Librar. 10*, 4, 133–157.

BONARDI, A. AND BARTHÉLEMY, J. 2008. The preservation, emulation, migration, and virtualization of live electronics for performing arts: An overview of musical and technical issues. *J. Comput. Cult. Herit. 1*, 1, 1–16.

BROWN, A. 2008. Automatic format identification using PRONOM and DROID. Digital Preservation Tech. paper 1. `http://www.nationalarchives.gov.uk/aboutapps/fileformat/pdf/automatic_format_identification.pdf` (Last accessed 2/12).

DEVANBU, P. T. AND STUBBLEBINE, S. 2000. Software engineering for security: A roadmap. In *Proceedings of the Conference on The Future of Software Engineering*. ACM, New York, 227–239.

DOYLE, J., VIKTOR, H., AND PAQUET, E. 2009. Long-Term digital preservation: Preserving authenticity and usability of 3-D data. *Int. J. Digital Librar. 10*, 1, 33–47.

DUMKE, R. 2003. *Software Engineering-Eine Einführung für Informatiker und Ingenieure: Systeme, Erfahrungen, Methoden, Tools. 4*. Vieweg.

FIDGE, C. J. 2003. Verifying emulation of legacy mission computer systems. In *Proceedings of the FME'03 Conference on Formal Methods*. 187–207.

GLADNEY, H. M. AND LORIE, R. A. 2005. Trustworthy 100-year digital objects: Durable encoding for when it's too late to ask. *ACM Trans. Inf. Syst. 23*, 3, 299–324.

GRANGER, S. 2000. Emulation as a digital preservation strategy. *D-Lib Mag. 6*, 10. `http://www.dlib.org/dlib/october00/granger/10granger.html`. (Last accessed 2/12).

GUTTENBRUNNER, M. 2009. Challenges for the evaluation of emulation. In *Proceedings of the Workshop on Data Analysis (WDA'09)*. 29–35.

GUTTENBRUNNER, M., BECKER, C., AND RAUBER, A. 2010a. Keeping the game alive: Evaluating strategies for the preservation of console video games. *Int. J. Digital Cur. 5*, 1, 64–90.

GUTTENBRUNNER, M., WIENERS, J., RAUBER, A., AND THALLER, M. 2010b. Same same but different - Comparing rendering environments for interactive digital objects. In *EuroMed*, M. Ioannides, D. W. Fellner, A. Georgopoulos, and D. G. Hadjimitsis Eds., Lecture Notes in Computer Science, vol. 6436, Springer, 140–152.

HEDSTROM, M., LEE, C., OLSON, J., AND LAMPE, C. 2006. The old version flickers more: Digital preservation from the user's perspective. *Amer. Archiv. 69*, 28.

JONES, C. 2004. Seeing double: Emulation in theory and practice. The Erl King case study. In *Proceedings of the Annual Meeting of the American Institute for Conservation of Historic and Artistic Works*. Electronic Media Group.

KLEINER, K. 2001. I love space invaders. *New Sci. 172*, 2313, 46–8.

LAMPORT, L. AND LYNCH, N. 1990. *Handbook of Theoretical Computer Science* Vol. B: Formal Models and Semantics. Elsevier Science Publishers B.V., 1157–1200.

LIU, J. W. S. W. 2000. *Real-Time Systems*. Prentice Hall PTR, Upper Saddle River, NJ.

MARCUM, D. B. 1996. The preservation of digital information. *J. Acad. Librarian. 22*, 6, 451–454.

MATTHEWS, B., MCILWRATH, B., GIARETTA, D., AND CONWAY, E. 2008. The significant properties of software: A study. JISC Study. `http://www.jisc.ac.uk/media/documents/programmes/preservation/spsoftware_report_redacted.pdf`. (Last accessed 2/12).

PHELPS, T. A. AND WATRY, P. 2005. A no-compromises architecture for digital document preservation. In *Proceedings from 9th European Conference on Research and Advanced Technology for Digital Libraries (ECDL'05)*. 266–277.

PHILLIPS, G. 2010. Simplicity betrayed. *Comm. ACM 53*, 6, 52–58.

ROTHENBERG, J. 1999. *Avoiding Technological Quicksand: Finding a Viable Technical Foundation for Digital Preservation*. Council on Library and Information Resources. `http://www.clir.org/pubs/reports/rothenberg/contents.html`. (Last accessed 2/12).

SLATS, J. 2003. Emulation: Context and current status. Tech. rep. `http://www.nationaalarchief.nl/kennisbank/emulation-context-and-current-status-2003`. (Last accessed 2/12).

STRODL, S., BECKER, C., NEUMAYER, R., AND RAUBER, A. 2007. How to choose a digital preservation strategy: Evaluating a preservation planning procedure. In *Proceedings of the ACM IEEE Joint Conference on Digital Libraries (JCDL'07)*. 29–38.

THALLER, M. 2008. Interaction testing benchmark deliverable PC/2 - D6. Internal Deliverable, EU Project Planets. `http://planetarium.hki.uni-koeln.de/planets_cms/sites/default/files/PC2D15_CIM.pdf` (Last accessed 2/12).

VAN DER HOEVEN, J., LOHMAN, B., AND VERDEGEM, R. 2007. Emulation for digital preservation in practice: The results. *Int. J. Digital Cur. 2*, 2, 123–132.

VAN DIESSEN, R. J. 2002. Preservation requirements in a deposit system. IBM/KB Long-Term Preservation Study Report Series Number 3 Chapter 3. `http://www.kb.nl/hrd/dd/dd_onderzoek/reports/3-preservation.pdf` (Last accessed 2/12).

VON SUCHODOLETZ, D. AND VAN DER HOEVEN, J. 2008. Emulation: From digital artefact to remotely rendered environments. In *Proceedings of the 5th International Conference on Preservation of Digital Objects (iPRES'08)*. 93–97.

WEBB, C. 2005. *Guidelines for the Preservation of the Digital Heritage*. Information Society Division United Nations Educational, Scientific and Cultural Organization (UNESCO) – National Library of Australia. `http://unesdoc.unesco.org/images/0013/001300/130071e.pdf`. (Last accessed 2/12).

Received April 2011; revised January 2012; accepted February 2012