Towards a Risk Model for Emulation-based Preservation Strategies: A Case Study from the Software-based Art Domain

Klaus Rechert
University of Freiburg
Hermann-Herder Str. 10
79104 Freiburg, Germany
klaus.rechert@rz.uni-freiburg.de

Patrícia Falcão
Time Based Media Conservation, Tate
7-14 Mandela Way
London SE1 5SR, U.K.
patricia.falcao@tate.org.uk

Tom Ensom
Department of Digital Humanities, King's College
Drury Lane
London, WC2B 5RL, U.K.
thomas.ensom@kcl.ac.uk

ABSTRACT

Virtualization and emulation provide the technical basis for a potential preservation strategy to keep performing digital objects accessible, despite obsolescence of their original technical platform. By simulating the presence of physical components, in the form of virtual hardware, it is possible to maintain the software environments needed to run original software.

In this paper we describe a conceptual model to analyse and document the hardware and software elements of software-based artworks, in order to then identify dependencies and assess risks. This information is used to select the most appropriate options when creating disk images, and to make decisions about whether an emulation or virtualization platform is more appropriate in supporting these disk images. We conclude with recommendations for ensuring the sustainability of disk images in the medium-to-long-term, and strategies to mitigate the risks related to external dependencies.

Keywords

Software-based Art; Emulation; Preservation Strategy

1. INTRODUCTION

Successful preservation of software-based artworks requires a deep understanding of both the technical aspects that underlie the performance of the software and the properties of the artwork that must be maintained to ensure its authenticity. Only by looking at those two aspects together is it possible to determine the relevant technical strategies to apply.

Software-based artworks are typically understood as artworks for which software is the primary medium. Software usually makes use of source code, as in Richard Rinehart's definition, *"... There is one aspect of digital media that separates them from traditional art media and even other electronic art media; source code."* [15] Sometimes however, software might be created at a level abstracted from the source code itself – through the use of production tools (for example, a WYSIWYG or visual editor).

Currently, Tate's collection includes only ten software-based artworks, a small number even in the Museum context. However, these artworks have been produced by markedly different artists and programmers, over a period of ten years; resulting in a wide variety both in the functions performed by the software and in the hardware and software systems used. The software is often custom-built for a specific artwork, so there are no standards as to how it is built or documented. The unique nature of the technologies used means that individual preservation plans are essential, and for some of the artworks emulation has already proven to deliver satisfactory results. However, the expected growth of the collection and the related practical and economic constraints highlight the importance of identifying common features and developing strategies that can be applied consistently, so as to make the preservation of the artworks and related artefacts sustainable.

The proposal in this paper aims at keeping the digital artefacts required to perform an artwork available and accessible by preserving the technical platform they were developed for. These platforms, partly physical (hardware) and partly non-physical (software), are superseded by new platforms every five to ten years. Once the hardware parts are out of production, the software parts also become inaccessible and old platforms disappear from the market and general use. To keep born-digital objects accessible, a promising approach is to focus on keeping the hardware platform alive by simulating the presence of the physical parts through virtual hardware. Virtualization and emulation are able to provide the technical basis for a potential preservation strategy to keep performing digital objects accessible, despite obsolescence of their original technical platform. Virtualization and emulation are proposed as generic technical preservation strategies, which can be shared among similar artefacts. The process of creating an emulated version of an artwork has the advantage of highlighting preservation risks posed by a constantly changing technical environment. It is also the moment to evaluate specific external dependencies the digital object may have and identify possible solutions or steps that can be taken to reduce the impact of the loss of these dependencies.

We describe a methodology for evaluating whether emulation or virtualization can or should be applied to the digital artefacts that make up a software-based artwork. The methodology combines an assessment of risks for preservation, a proposal for best-practice when migrating between physical and virtual environments, and consideration of how to maintain virtual machines in the long-term. The final section addresses the fact that emulators themselves become

obsolete. It discusses ways in which dependency on a particular emulation platform can be reduced and how best to migrate to a virtual hardware environment in order to facilitate long-term access to the software. The processes described in this paper have particular potential for implementation in art museums, but may also be broadly relevant to other types of software collection.

2. RELATED WORK

Emulation as a preservation strategy has been discussed for 20 years, an early example being Jeff Rothenberg's paper from 1995, "Ensuring the Longevity of Digital Documents" [17]. These early ideas were later reframed in an art context by Richard Rinehart's work for Rhizome [14]. The term emulation was also used in various other projects in art conservation, but often in a wider sense, to refer to the re-coding of a work or changing the technologies used to instantiate a work [19, 6].

Hardware virtualisation was first specifically proposed as an approach in the context of software-based art by Tabea Lurk in 2008 [9], and positioned as a potential conservation tool. The SCART project investigated the use of emulation, virtualization and re-coding for Mondophrenetic™. [1] Today, however, emulation is still neither common practice nor has it evolved from singular project-based experiments.

In the last years, further research by various institutions has resulted in progress regarding usability and scalability of emulation and virtualization for the preservation of complex digital objects. Some of the most significant of these projects were the Olive Archive [10], the bwFLA Emulation as a Service [13] and the Internet Archive's Emularity[2]. While all three approaches greatly reduced technical hurdles and seem to be ready for broader adaptation and usage within the preservation community [16], there is still a lack of generic emulation-based preservation strategies. Recent research on software-based art preservation has mostly focused on CD-ROMs [5, 2, 3].

The notion of significant properties has been examined for complex digital objects of various kinds, including software, and these studies have included discussion of an artefact's technical features and dependencies [7]. However identifying and classifying these kinds of properties can be challenging due to, *"The diffuse nature of software-based artworks and the systems of which they are made, means that obsolescence is often more difficult to monitor than in traditional time-based media works of art and the risk of failure in these works is harder to predict."* [8] Further to that, for artworks, the concept of significant property must extend beyond the properties of the digital objects themselves. It must include other elements that influence the experience and understanding of an artwork, such as the spatial parameters of display in the gallery space [8].

With software-based artworks now making their way into the collections of art museums typically associated with more traditional media, there is a pressing need to address the challenges of preserving these works and to develop the associated skills among practitioners. Best practices however, are not yet available, or indeed, arrived at with any real consensus. It is hoped that this paper will be a useful contribution to this developing area and help provoke a movement toward agreeing best practices among the community.

[1] Mondophrenetic™ a work made by Herman Asselberghs, Els Opsomer and Rony Vissers in 2001, https://www.scart.be/?q=en/content/case-study-report-mondophrenetic-2000-herman-asselberghs-els-opsomer-rony-vissers-0

[2] http://digitize.archiveteam.org/index.php/Internet_Archive_Emulation (online, 4/22/16)

3. TECHNICAL CHARACTERIZATION

At first glance a digital artefact consists of a set of byte streams, e.g. binary files. Keeping these accessible in the long-term (i.e. being able to retrieve exactly the same byte stream as originally stored) poses some risks, but from today's perspective is a manageable task supported by established procedures and tools. In contrast, keeping a digital artefact's experienceable features accessible is a much harder task, since the artefact needs to be rendered, or performed (and possibly requires interaction with viewers/visitors) and depends on a suitable technical environment to do so. To ensure the long-term availability of a computer platform's hardware (e.g. to render a (preserved) digital artefact) emulation and virtualization can be considered as potential access and preservation strategies. In order to prepare an emulation-based preservation plan, a detailed technical analysis of the digital artefact and its technical environment is required, to uncover explicit but also implicit dependencies as well as determine present and future risk-factors.

3.1 Software Layer

Many digital artefacts are not self-contained. They do not only require hardware, but also additional software to be rendered. Therefore, one part of an artefact's technical environment represents a software runtime – *the software (rendering) environment.* A minimal software environment is typically an installed and configured operating system, but in most real-world scenarios a more complex software environment with additional software installed and configured is required. When acquiring an artefact, its software environment needs to be assessed.

3.1.1 Interfaces between Hardware and Software

Operating systems (OS) play an important role in software environments, as they typically provide a hardware abstraction layer, for instance, any application is able to connect to the internet without knowing technical details about the hardware used. This abstraction is usually achieved through *technical interfaces* – the so called hardware abstraction layer.

The technical interfaces between an operating system and hardware have two sides: the top-side (OS-side) unifies usage of different hardware components (e.g. sound card, network card, graphic cards etc.) and the bottom part (hardware-side) operates the hardware in (vendor-) specific ways. The connection between top- and bottom interfaces are implemented as hardware drivers (sometimes as BIOS or ROMs extension[3]).

Through the use of OS hardware abstraction, software dependencies on physical hardware components are usually unnecessary. Software artefacts then pose only abstract hardware dependencies (e.g. the minimal screen resolution, sound and network support etc.). This approach has greatly simplified software development and improved the compatibil-

[3] Apple Macintosh Computers are a good example for relying on ROMs

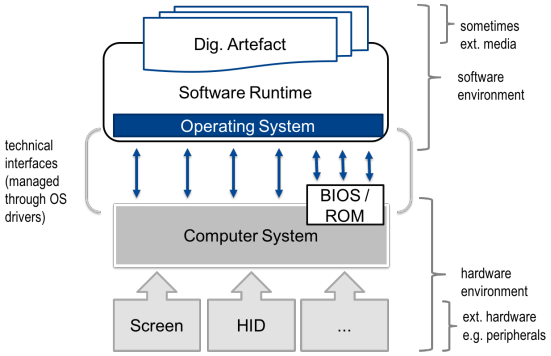

Figure 1: Rendering environment of a dig. artefact.

ity of software with a wide spectrum of different computer setups, since most artefacts and software dependencies typically use operating systems to interact with hardware.

Some digital artefacts, however, have no software dependencies and are able to interact with hardware components directly. However, these cases are rather rare (at least for typical computer setups) and usually associated with specific hardware – for example, game consoles, arcade machines, robots or similar special purpose machinery. There are also some rare cases where an operating system is used but the artefact also relies on direct access to a specific hardware resource.

3.2 Hardware Layer

The hardware layer connects the digital artefact (and its software runtime) with the physical world. Hardware components, as considered in this paper, can be classified into two classes: a 'machine' with built-in hardware (e.g. a computer, phone, tablet etc.), and external hardware components connected to this machine. For the purpose of this paper the hardware's properties are only relevant in so far as they influences the options for emulation or virtualization. The focus of this section then, is on the hardware characteristics to document when considering its use for emulation or virtualization purposes.

When an artwork is acquired it is important to analyze and describe the hardware used by a digital artefact, as this will help to define the technical environment required for that digital artefact to be rendered. The level of detail needed to describe the hardware depends on the characteristics of the software environment where the digital artefact is run.

3.2.1 Virtual Hardware

Any computational (binary) operation can be implemented in hardware (i.e. hard-wiring operations for a specific purpose as a machine or machine component) or in software (i.e. translating a set of complex logic operations by compiling source code into instructions for a generic machine – e.g. any generic CPU). Both implementation options are in theory equivalent, however operations implemented in hardware are usually by magnitudes faster compared to a pure software implementation. This equivalence allows, however, the replication of any outdated hardware component in software and the exposing of its functionality using contemporary hardware. Hence, the *Computer System* block, depicted in Fig. 1 can be implemented either as physical or virtual hardware.

There are currently two generic technical options to replace outdated hardware: virtualization and emulation. These two technologies are not mutually exclusive, and in fact share many concepts.

Virtualization. Virtualisation is a concept and a technical tool to abstract (*virtualize*) hardware resources, such that a so called virtual machine (VM) (also called guest) is not interacting with the physical hardware directly. Instead a hypervisor (also called host) provides a virtual hardware interface for guests, usually providing access to a unified set of (mostly emulated) hardware, regardless of the machine's hardware configuration the host is running on. A virtual machine is still able to utilize performance advantages of real hardware, in particular (but not restricted to) using the host's CPU. The hypervisor is in charge of enforcing rules as to how a virtual machine is able to use the host's CPU (i.e. restricting access to specific, sensitive instructions – for instance, preventing a VM accessing the host's memory), such that a VM is unable to takeover the physical machine or interfere with other VMs running on the same host. Modern computer architectures have built-in hardware features (e.g. dedicated CPU and memory-management features) to support virtualization, implementing parts of the hypervisor in hardware and thus reducing the virtualization overhead as well as the complexity of the host system (software hypervisor).

Hardware components available to guest VMs are either fully emulated or, to improve performance by eliminating most of the emulation overhead, paravirtualized [18]. Paravirtualized hardware offers (almost) direct and efficient access to the host's hardware, typically (but not restricted to) network cards and storage controllers (e.g. disk access). When using paravirtualized hardware, a driver specific to the virtualization system needs to be installed within the guest system. In contrast, when using fully emulated hardware components, the guest system is able to use the same driver code as if it were using a physical hardware component.

Emulation. An emulator (usually) implements a specific outdated computer system, primarily a CPU architecture, interpreting the outdated machine's instructions and translating these to equivalent instructions of the current host system. This however, is not sufficient to run or execute even the simpler applications. There is additional hardware emulation required to attach storage devices (e.g. a bus to a floppy or hard-drive controller), a memory management unit (MMU), video and audio output, network interfaces, and interfaces for interaction with users. In contrast to virtualization, emulation implements a complete computer system solely in software. Therefore, an emulated system is independent of the current computer architecture, e.g. we are able to run a Motorola 68k emulator (e.g. to boot MacOS System 7) on a current Intel-based Windows PC.

3.2.2 Virtualization or Emulation?

The main difference between emulation and virtualization is the reliance on contemporary hardware (or the lack thereof). Virtualization relies on and utilizes real hardware for performance reasons as it offers typically (almost) native execution speed, but has restricted platform support. By

definition, virtualizers (such as VirtualBox and VMWare), are only able to run Intel-based x86 VMs, whereas emulators cover almost any technical platform, in particular obsolete ones. Furthermore, the close ties to today's computer platforms restrict a virtualized machine's longevity, particularly if the virtual machine relies on contemporary (paravirtualized) hardware components. To support paravirtualized hardware, the VM (and the virtualization technology) not only rely on VM-specific drivers installed in guest systems, but these drivers also expect appropriate support from the host OS, typically as a host-OS kernel extension to support interaction with the host's hardware directly. Any major OS-kernel upgrade (e.g. changes to internal kernel interfaces) requires an upgrade of the virtualizer too, and therefore the longevity of the virtual machine guest depends also on vendor or community supporting current operating systems.

As all hardware components (and respectively their low-level interfaces to be used by drivers) of a computer system have to be re-implemented in software, and the availability of drivers for old operating systems is crucial, only a small set of emulated hardware components are provided as virtual or emulated hardware. Typically, emulator developers focused on hardware in widespread use and with good driver support, e.g. Soundblaster 16/32 for soundcards and Intel's E10/100/1000 for network cards. In practice there is a significant overlap between virtualizer and emulators, with both supporting a similar set of emulated hardware components, a useful property for a mixed preservation strategy. Hence, for digital preservation and related tasks, one should avoid extensions or drivers specific of a certain virtualizer or emulator. If possible, drivers originally published by hardware vendors should be used, since using the original driver also verifies (at least partly) correctness and completeness in the emulated hardware. Furthermore, by using emulated standard hardware and their drivers, both the effort and risk of migrating a virtualized system to an emulated one is reduced. Contemporary emulators require a specific contemporary host system, which is to say that emulators are normal software components with specific requirements regarding their software and (abstract) hardware environment. However, the guest systems running on emulated hardware are usually not specifically configured for a certain emulator (e.g. using original hardware drivers). Hence, migrating an emulated guest to a new emulator of the same hardware architecture, will require little or ideally no adaptation of the system.

To summarize, using virtualization technology can be a useful addition to a general emulation strategy, in particular if performance matters. Even though a virtualization solution can not be considered a long-term solution, if carefully configured (e.g. avoiding paravirtualized drivers) the effort required to migrate a system to a new virtualizer or emulator is lowered. Furthermore, having two similar systems at hand (e.g. VirtualBox for virtualization and QEMU for emulation) offers the option to pursue a two-track strategy, and in particular allows to practice system migrations between two virtual hardware stacks.

3.3 Conceptual Layers

Based on the aforementioned structural view and discussion a (minimal) technical description of a digital artefact can be divided into three conceptual layers:

1. *Artefact Description & Configuration* This layer conceptually captures the technical description of the object, in particular the technical properties of the artefact itself and its specific (artefact-specific) configurations and settings.

2. *Software Environment & Configuration* This layer conceptually captures all installed software components and applications, including the operating system (if present). Furthermore, it may capture the configuration and settings of individual software components and the operating system.

3. *Hardware Environment* This layer conceptually captures the hardware components used by both upper layers.

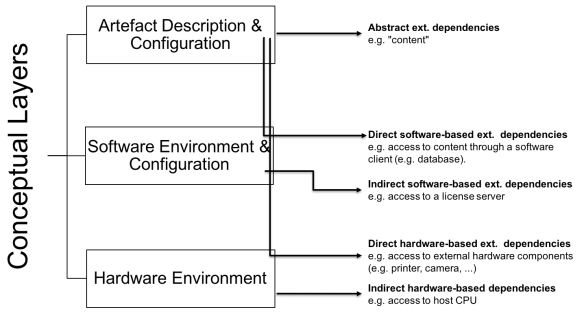

Figure 2: Characterization of external dependencies derived from conceptual layers.

3.4 Characterization of External Dependencies

A digital artefact requires more than an isolated technical environment (consisting of data and/or extracted disk images) and a computer system or installation to be rendered. For this reason, external dependencies need to be determined, characterized and documented.

The three logical layers together describe the technical environment and characteristics of a digital artefact. In each layer, individual components may depend on functionality or data not available within the local setup (*direct* external dependencies). Additionally, there may be *indirect* external dependencies. For instance, a direct software dependency originating from the artefact layer may itself have external dependencies.

From the three conceptual layers we can derive the following five types of external dependency:

1. *Abstract external dependency:* Abstract dependencies are posed directly by the digital artefact. These dependencies are abstract in that they do not rely explicitly on a specific software or hardware component. For instance, an artefact's performance might depend on access to some kind of data source stored at an external site, but does not rely on specific software to retrieve or modify the data (e.g. the data is directly accessible through the local file system).

2. *Direct software-based external dependency:* To access external data and/or functionality the artefact requires additional software. For instance, a specific client software is required to connect to a remote database and retrieve data.

3. *Indirect software-based external dependency:* This type of external dependency is not posed by the artefact directly but by another of the artefact's software dependencies. It is therefore called an indirect software dependency. For instance, database client software might require access to a license server to function.

4. *Direct hardware-based external dependency:* The digital artefact requires access to external hardware, such as direct access to a specific printer.

5. *Indirect hardware-based external dependency:* The software environment of the digital artefact requires access to specific hardware, e.g. a software component requires access to a hardware license dongle to function.

3.4.1 Peripherals

An important subset of external dependencies are external (connected) hardware components, which can be seen as direct or indirect hardware-based dependencies. A general characterization of external hardware is beyond the scope of this paper. Instead, this section will focus on the characterization of the communication between a virtualized or emulated machine and external hardware, as well as data protocols used, (i.e. how information is exchanged between software and external hardware). This is the essential information needed when considering the use of emulators to run an artefact.

To (re-)connect external hardware components a physical machine is required. In the case of an emulated or virtualized computer system, the host system needs to be able to connect and interact with external hardware components such as human interface devices (HID) (e.g. mouse and keyboard), printers or other peripherals, e.g. by using a suitable connector or a passive (if electrically compatible) or active (e.g. analogue-to-digital converter) adapter, to provide a compatible connection.

Second, the host operating system needs to provide a software interface for applications to communicate with external hardware. For example, a COM port, the Windows software-side representation of a serial connection used for generic peripheral devices such as printers. If external hardware are accessible through the host OS's interfaces, an emulator (acting as a normal software application) is then able to use this external hardware.

Finally, the emulator needs to provide a virtual hardware interface connected to the host's software interface, visible to and usable by the guest OS. Through all these layers integrity of the data protocols, used to communicate between software within the emulated environment and external hardware, needs to be maintained.

Similarly to the previously discussed built-in hardware components, judging the relevant and controllable risks posed by an external component on the complete installation should be focused on its technical interfaces. Fortunately, the number and type of external technical interfaces is low. Their types are standardized and mostly use general purpose technologies (such as USB, serial, parallel etc.). Some older external components and interfaces aren't supported by emulators anymore, mostly for practical reasons, such as host systems providing no appropriate connectors (e.g. a mobile phone or tablet being used as an emulator host has limited connector options). In these cases, usually an emulated or simulated option is provided (e.g. a GUI console gamepad emulation on a touchscreen device).

4. AN EMULATION-BASED PRESERVATION STRATEGY

In the previous section three conceptual layers describing the technical characteristics of a digital artefact were identified. The structural separation of hardware, software environment and the digital artefact facilitates the evaluation of preservation risk factors and strategies without considering the specificities of all layers.

By choosing a virtualization or emulation strategy the focus of preservation moves from physical objects (*Hardware Environment* layer) to disk images (*Software Environment* layer). Hardware will inevitably suffer from technical and physical decay, and for cost reasons can only be preserved in individual cases (even then eventually failing). The same applies to emulators. In contrast, preserved software environments don't change in their hardware requirements over time and can be considered as constant and stable in their requirements. The goal of this strategy can be described as having software environments (disk images) to *run anywhere* and *run forever*. *Run anywhere* translates into making a complex software installation portable. Most relevant is to create the most generic software environment possible, with regards to hardware dependencies, such that a digital artefact is able to perform outside of its original environment. *Run Forever* can only be achieved if the disk images are maintained over time. For that both the technical interfaces and external dependencies of disk images must be regularly monitored. This can be done either by referring to technical documentation, if available, or by performing periodical tests to assess the functionality of the dependencies. If an interface's functionality breaks or an external dependency becomes unavailable, the disk image (respectively its software environment) or the artefact itself must be adapted so they can perform again in the changed technical environment.

For that to be possible, steps must be taken when creating the disk images to facilitate their longevity. Alongside this there must be careful planning for their obsolescence, or the obsolescence of their technical environment, in particular the emulation or virtualization platform. The first step for a successful emulation strategy is knowing what information and resources are available to support this process. Having the artefact's source code available (enabling the recreation of the artefact on a current machine) allows for a higher technical abstraction level and may also open the door to alternative strategies (e.g. a traditional migration approach). If source code is not available, creating an emulated version of the work may be the only suitable choice. How this emulated version is then created will depend on whether the software can be re-installed and if all the required software dependencies are available and operational. Hence, for an emulation-based preservation strategy the artefact's software environment needs to be determined, in particular its software environment's composition and the technical interfaces to the hardware layer (cf. Section 4.1).

Having a (detailed) software environment description allows to focus preservation planning and preservation action activities on monitoring and managing the technical links between software and hardware environments. While these

links are stable in the short term, emulators are also subject to the software life-cycle, and will become technically obsolete at some point. Then, if new emulation software is required, a new (emulated) hardware environment needs to be determined which meets the technical requirements of the software runtime. If the technical interfaces between hardware and software environment are well documented – as a vital part of a software environment description, all affected software environments can be determined, and the search for new emulators can be guided effectively. If no perfect match is found, the required adaptations of the affected software environments can be predicted in an automated way using this same documentation (cf. Section 4.2).

For the remainder of this section we assume that the artefact itself is either already available as a file or bitstream or the artefact is part of the disk image. Furthermore, we assume that the artefact's significant properties have been assessed and the artwork is available for verification purposes.

4.1 Acquiring Software Environments

The task of determining an artefact's software environment, starts with the analysis of the artefact with the goal to produce an accessible, performing setup without relying on the availability of physical hardware components and to gather enough information to support future preservation tasks, i.e. ensuring the artefact's longevity. Depending on the artefact's composition, e.g. individual technical components present – a binary object, its configuration, possibly the object's source code, any kind of documentation and a reference installation in form of a computer system – different options are available to pursue both goals. Furthermore, the levels of documentation depend on a series of factors, primarily if the artist and/or the artist's programmer has supplied any technical details about the software, and whether they are available to answer any questions about the work's technical makeup, runtime process and system configuration. In a first step, all components of an artefact are assessed regarding information about the artefact's technical dependencies, i.e. software, hardware and external dependencies and to support the selection of virtual hardware.

4.1.1 Selecting Virtual Hardware

Emulators and virtualizer provide only a limited selection of supported hardware components. Their capabilities are best described by a list of supported computer systems (e.g. x86 ISA PC or Apple Macintosh Performa) and a list of supported operating systems. Therefore, the most important information to be documented in the original computer system to be preserved is therefore the general hardware architecture (e.g. CPU type, bus architecture or ROM type/version), in order to help choosing an appropriate emulator or virtualizer. The choice of an emulator or virtualizer also requires information about the software environment (e.g. the emulator must support Windows 3.11), as even if an emulator supports a particular computer system, it may not support – or support only partially – an apparently compatible operating system. Detailed hardware information will only rule out incompatible emulated computer platforms. The final decision on a suitable emulator/virtualizer requires that support for both the computer system and the associated software environment (primarily operating system support) are assessed.

A further, more detailed comparison between the identified hardware components and the features of a specific emulator is useful to estimate up-front the work required to migrate the machine's software environment (disk image) to a new hardware environment. A detailed list of hardware components installed provides insights on how the operating system was configured. For example, by comparing a detailed list of hardware components with a list of the hardware supported by an emulator it is possible to predict if a software environment (in form of the computer's disk image) will boot directly using emulated software. If the system is able to boot to a certain point (e.g. a simple 'safe' environment with all non-essential hardware features disabled), built-in operating system tools can be used to adapt the system to the new, emulated hardware environment. These adaptations could involve identifying available hardware or suggesting drivers.

The importance of specific hardware components can only be assessed using information about the artefact and/or its software environment. For instance, if the hardware setup involves a high-end 3D graphics card, its importance to the whole installation can be judged on how the graphics card is used: is the performance of the card a key factor or does the software depend directly (i.e. direct hardware access) or indirectly (i.e. through an abstraction layer such as DirectX or OpenGL) on specific hardware features.

The necessity of incorporating software environment information (to assess the role of hardware components) as part of a digital artefact's technical environment highlights the importance of the technical interfaces between software and hardware environment. These pose high risks to the artefact's functionality.

4.1.2 Workflows

To provide a runtime environment for a digital artefact, any emulation-based preservation strategy is likely to result at some point in managing a set of software environments or virtual disk images containing instances thereof. Disk images may contain the digital artefact (or parts of it), or might be prepared separately from the digital artefact, which is held on separate virtual media (or similar) and to be used with this disk image. Three different workflows/strategies can be applied to the task of image acquisition, depending on practical options and requirements, as well as on information available about the artefact and environment. Fig. 3 illustrates the software environment acquisition process.

Generalization: If the original computer system is available, images of physical media (e.g. a hard disk) can be made. To make these useable in an emulation environment and to facilitate the long-term management of disk images, as a first step it is necessary to *generalise* the technical interfaces between hardware and the lowest software layer (typically the OS, respectively OS hardware drivers and configuration). In case of disk images originating from physical media, generalisation is part of *migrating* a software environment from physical hardware to virtual/emulated hardware. Generalising means that any specific drivers (for instance, from a disk image made from a physical computer) are replaced with drivers supported by the virtualization or emulation platforms in use. Hardware dependencies should be systematically determined and all necessary adaptations kept to a minimum when migrating to virtual / emulated

hardware, e.g. to maintain the environment's authenticity.

As part of this process, the choice of emulated hardware and drivers should be consistent, so that for each combination of OS and hardware platform always the same (emulated) hardware component is used. This means that for each emulated hardware platform and all associated software environments there is only a limited number of technical interfaces to be maintained and monitored, and this consequently means that the same migration strategy can be applied to different software environments. For example, ten different physical computers may use ten different video cards, which may each use different drivers with the same functions. By generalising the disk images created from these computers the number of drivers needed for that video card can be reduced, so that instead of ten different drivers only one is needed, and later on only one may need to be replaced (if necessary at all) for migration to another emulator.

As a result the generalisation workflow produces a disk image to be used with a contemporary emulator. Additionally, external and hardware dependencies are uncovered by running the artefact and its software setup in a different virtual environment. In particular, the image's technical interfaces can be documented by running on well understood virtual hardware. To ensure that the significant properties are not affected the initial process of image generalisation should be performed during or after image acquisition, and preferably a comparison between the original and generalised systems should be made.

Rebuilding A second workflow is necessary if either no computer system was available to be imaged or – in order to reduce future preservation risks – a secondary set of software environments (disk images) are desired.

Ideally the configuration of a software environment is known, i.e. available as (machine readable) metadata, such that the software environment can be rebuilt if necessary. If the software environment is not known, an artefact's dependencies may be determined by using its original environment as a reference (e.g. the artist's original computer). This reference environment can be analyzed and the artwork can be isolated from its original environment to be rendered in another technically compatible environment. Either using documentation derived from a reference setup or systematically determined software dependencies (e.g. using tools for analyzing an artefact's technical format or its runtime behavior [4]), a suitable software rendering environment can be remodeled by re-installing and re-configuring an artefact's software environment in an emulated environment.

When re-building environments, a consistent configuration of an operating system is built. For efficiency reasons, a specific operating system on a specific hardware platform is only installed once, any more sophisticated software environments are derived from these base images. Also in this case the initial choice of the system's configuration matters, as it will affect the preservation options of all derived environments. The choices on a software environment's hardware components could be based on emulator support (do other emulators/virtualizers, in particular open source, support this hardware component?), the popularity of the device while in production and available driver support. If a popular and tested open source implementation of this hardware component is available, it seems more likely that future emulators will resort to that implementation instead of

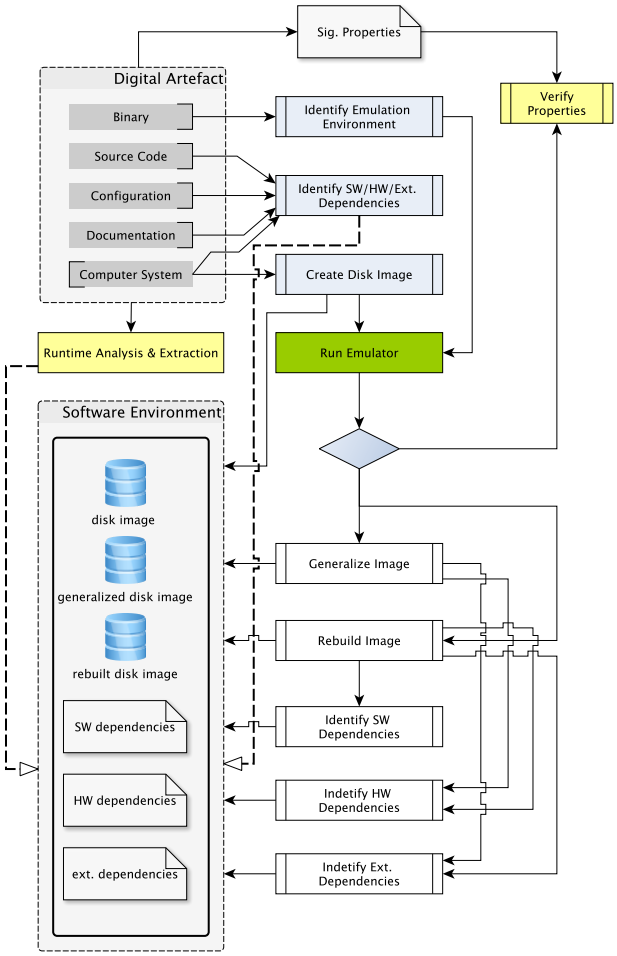

Figure 3: Acquiring software environments

implementing a historic hardware component from its specifications. The same applies for the availability of drivers: if a hardware component has been successfully used in emulators even after the component was out of production, it is highly likely that archived versions of drivers of these hardware components remain available.

The process of manually rebuilding an artefact's rendering environment can also be used to create verified and machine readable technical metadata. Machine readable installation information (e.g. what type of software is installed), and more importantly the environment's configuration, may be created in an automated way during a structured software environment rebuilding workflow [12].

Pre-built Environments In some cases a complete computer system is not available, i.e. only the (binary) artefact is available without a reference runtime or setup. In this case an alternative is the use of pre-built software environments. These environments resemble typical computer systems of a certain period, e.g. a typical MS Windows 98 installation equipped with popular software, utilities and libraries. In this case a suitable pre-built environment needs to be identified for a given digital artefact and, if necessary, adapted to the artefact's specific requirements. Similar to re-built environments, this approach results in a well documented

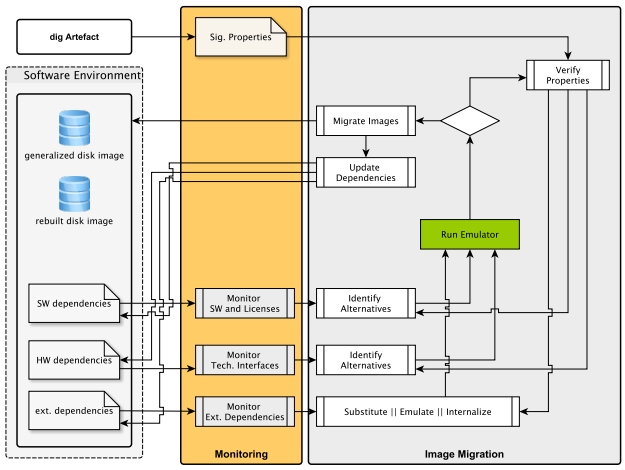

Figure 4: Maintaining software environments

and well understood (base) environment, shared among a set of similar digital artefacts.

4.2 Maintenance and long-term Preservation

The outcome of a software environment acquisition workflow is a disk image (representing an instance of a software environment). Even though the technical file format of the resulting disk image is identical in all strategies (usually a virtual hard-disk, bootable by an emulator) different workflows must be followed for maintenance and long-term preservation. The actual available preservation actions will depend on information about the content and configuration of the disk images, in particular its technical dependencies, rather than on their file format. In order to maintain a software environment's *run forever* property, its technical dependencies require monitoring, particularly:

- monitoring of software components used, including availability of licenses and external dependencies;
- monitoring of existing and emerging emulation and virtualization technologies for continued support of identified technical interfaces;
- monitoring of external dependencies.

Through monitoring technical dependencies, the identification of an imminent risk of obsolescence may indicate the need to migrate a disk image. This strategy then becomes very similar to the one applied to simpler digital objects such as spreadsheets or word processor files. In general, to implement a migration strategy, objects are monitored regarding their technical support (and therefore proximity to technical obsolescence) and migrated to a new, more sustainable or current format if required. Ideally all the significant properties of these objects' are preserved in the process. These interfaces may break if an emulator drops support for specific hardware (e.g. emulator upgrade) or if an emulator becomes unavailable. In the case of digital disk images, the significant properties to be addressed by a migration strategy usually relate to the technical interfaces identified. The functionality of technical interfaces can be documented, monitored and tested automatically, at least to certain extent. For instance, the software-side (or driver) of a technical interface to a sound card can be verified through a generic test (i.e. that sound is produced for different formats and configurations). If the test is successful then it is highly likely that any digital artefact using the same interface is also able to produce sound output. However, a general assessment of the emulator's functionality, in particular the equivalence of original and emulated CPU, is a much harder task [1]. Furthermore, computational performance features such as the rendered frame rate of graphics after processing, disk I/O performance, synchronous video and audio or even interactivity (for example, the latency between a user input event such as a mouse click and the system's visible reaction), would all need to be verified.

4.2.1 *Preservation Risks*

There are several levels of documentation possible for all three monitoring activities, depending on the choices made acquiring the software environment.

The highest level of risk for emulation exists when there is only limited knowledge about the software environment, its composition and configuration as well as hardware dependencies, i.e. technical interfaces. If, due to limited information about technical dependencies, an a-priori risk assessment is not possible, a future migration strategy may fall back to trial-and-error. Furthermore, there is a risk of an incomplete monitoring process, potentially missing obsolescence indicators. Similarly, there is a high risk of failing to rebuild the software environment if the software environment's composition and configuration is unknown. To reverse engineer an environment it is essential to have both a good knowledge of the computer system and installable software packages. Over time both these factors tend to decrease and as a consequence risk increases significantly over time. If there are resources available for technical analysis, documentation and collection software packages at acquisition, then long-term preservation risk can lowered more efficiently. This is particularly relevant for hardware dependencies and the configuration of the operating system, for instance a complete list of all the drivers installed and description of the hardware configuration used. With this information at hand an a priori assessment of technical migration risks becomes possible, as necessary drivers can be collected in advance and potential alternatives considered.

The lowest risk level is achieved when complete documentation is available about the enclosed software environment and its hardware dependencies, such that the environment can be rebuilt from scratch if necessary. In this case a preservation strategy does not solely depend on the acquired disk image (i.e. a single, "fixed" setup), as multiple strategies can be applied simultaneously. The same applies to disk images which are specifically built for preservation purposes. These images were already built within a virtualized / emulated environment, so reducing migration risk, as information on the images' content and configuration is – assuming a (semi-)automated documentation process – readily available and the installation procedures easily reproducible. The effort required for maintenance, however, may differ. This is due to different creation processes and creation goals.

Images based on documented system requirements and installation instructions replicate an existing system as closely as possible. Depending on the granularity of the available documentation, there may be slight variations and alterations to the original system specification, for example, to

cope with external dependencies and/or different hardware options in the virtual machine. A further migration of these images to a new platform may require an individualized strategy, as their similarity to the original system should be maintained. In contrast, images with software installations reduced to an artwork's essential software components are more resilient to technological change, as wider variations and adaptations are acceptable, as long as the artefact can be rendered, i.e. its significant properties remain intact. In both cases, re-produced images are able to share a common technological base, e.g. share a operating system installation and configuration as well as relying on the same technical interfaces. Through a shared technological base preservation risks can be shared among similar artefacts and collecting institutions.

In general, for artefacts or software environments interacting directly with hardware, there is a higher risk of an emulation strategy failing, in particular if they rely on custom built or modified hardware components. Even if they rely on widely used hardware, not every feature (and possible quirk) of real physical hardware components may be emulated accurately. Furthermore, emulators may have bugs or behave differently compared to the original systems. In contrast, artefacts relying on operating systems to interact with emulated hardware are more likely to be successfully re-enacted using emulation, as emulator implementations usually try to cover the feature-set of popular hardware (and drivers) and/or the behaviour of a popular operating system.

4.2.2 External Dependencies

The management of external dependencies requires a different set of strategies, as external dependencies are technically and conceptually more diverse than hardware found in computer systems and there is usually no drop-in replacement available. Due to the diversity of external dependencies, only abstract workflows are presented.

The first, and usually most efficient strategy is internalisation of external dependencies, such that they become either a part of the digital artefact or its software environment. In general, there are two types of dependencies which can be internalized, abstract data dependencies and functional dependencies. A simple example for a data dependency is an artefact accessing data which is externally stored. An internalisation option is to copy/mirror the data and make it locally accessible, such that it becomes a part of the artefact. In general, this option is applicable for abstract data dependencies, with data being accessed or served through standard protocols, e.g. protocols directly supported by the OS. Most of the times, modifications to the object and sometime even to the software environment can be avoided. In some cases, however, changes have to be made, e.g. to point to the new location of the data (which is in particular problematic if the digital artefact used hard-coded URIs and the artefact can not be changed).

For pure functional external dependencies, e.g. a computation is made by an external machine and the artefact requires the result to perform or a software dependency requires external functionality, such as a license server, or mixed functional and data dependencies (e.g. data is served through a specific protocol, which requires additional software support such as databases), can be internalized, if the (server) machine is available and suitable for emulation. The internalized machine can then be treated as a dependent, secondary artefact, emulated and connect to the primary artefact.

A second strategy to deal with external dependencies is making technical dependencies abstract. Through abstraction the risks of failing or obsolete components to the whole setup can be reduced. The main goal is to abstract technical requirements, such that equivalent replacements can be identified and applied. For instance, a problematic software dependency with a dependency on a license server may be exchanged with a less problematic one, e.g. a software product providing the same technical features for a given digital file format but does not rely on a external functionality. This strategy should be included in preservation planning activities, as it may not always yield into direct useable results, but prepares the ground for future preservation actions.

Finally, emulation and simulation can be pursued, if other strategies fail or are not applicable. This strategy requires broadening the scope of emulation to include interfaces and behaviour of external components. For instance, if an artefact relies on the availability of a video-portal accessed through a dedicated web-service protocol, the protocol interface may be simulated and either translated to a newer protocol version to retrieve content or the whole service is simulated using e.g. recorded or archived content. An example of emulated web services in the domain of research data management is provided by Miska et al [11]. A similar strategy can be applied to external hardware and hardware protocols.

5. CONCLUSION

The first step for the preservation of a software-based artwork is a technical analysis of the hardware and software setup required for its display. This analysis provides the basis for a description, which can be broken down into three conceptual layers: artefact descriptions and configuration; software environment and configuration; and hardware environment Assessing preservation options for each of these layers individually, provides a differentiated view on technological risk and potential mitigation options, and helps to make a changing technological environment more manageable.

The higher the degree to which an artefact can be abstracted from its technical environment, the more options for preservation actions remain. If an artefact can be re-built for different technical environments, its software, hardware and external dependencies may substituted (e.g. by changing the code or the artefact's setup). An artefact's software environment is of particular interest as it usually connects digital objects with hardware. If the software environment is known in its composition and configuration, the environment can, if necessary, be rebuilt in order to mitigate risk (e.g. substituting problematic dependencies). Furthermore, it becomes possible to consolidate disk images by, for example, building on a common base system consisting of operating system and a unified hardware configuration. Breaking down the technical characterization to a (common) set of technical interfaces shared among many artefacts of a similar type makes it possible to focus monitoring and technical migration work. If technical interfaces break, disk images may be migrated to a new (virtual) technical environment. Ideally, a migration path is only developed once and applied to all suitable artefacts.

In this paper we have presented an approach which, instead of looking at work-specific properties, focuses on the digital artefact's technical dependencies. If emulators were

able to perfectly reproduce out-dated hardware components, this technical perspective would be sufficient – at least concerning any computational aspects of the artwork. In practice however, emulators are far from perfect in this respect, such that a manual verification of an emulated result is indispensable. For this reason, the proposed migration method relies heavily on the ability to verify the performance of a digital artwork in a new technical environment. For digital artworks any kind of automated testing of technical properties has its limitations. Software-based artworks have a second, mostly conceptual layer of significant properties, which cannot be tested in an automated way and require a specialist's assessment (for example, qualities of the artefact's behavior). Still, a structured and (partly) automated verification of an emulation's (technical) performance characteristics remains one of the most important open challenges when implementing an emulation-based preservation strategy.

Furthermore, the technical approach presented requires a wider supporting framework. Primarily, a dedicated software archive is necessary (which includes management of licenses) to help ensure that a given software environment can be rebuilt. Additionally, it is useful to maintain a testbed of typical environments and common technical interfaces to be tested on newly released emulators. In contrast to testing an artwork's work-specific significant properties, these activities, and in particular the technical infrastructure, can be shared and re-used not only for sets of similar artworks but also among different institutions.

6. ACKNOWLEDGEMENTS

This project has received support from the European Union's Seventh Framework Programme for research, technological development and demonstration under grant agreement number 601138 – PERICLES.

7. REFERENCES

[1] N. Amit, D. Tsafrir, A. Schuster, A. Ayoub, and E. Shlomo. Virtual cpu validation. In *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP '15, pages 311–327, New York, NY, USA, 2015. ACM.

[2] G. Brown. Developing virtual cd-rom collections: The voyager company publications. *International Journal of Digital Curation*, 7(2):3–22, 2012.

[3] M. Casad, O. Y. Rieger, and D. Alexander. Enduring access to rich media content: Understanding use and usability requirements. *D-Lib Magazine*, 21(9/10), 2015.

[4] F. Corubolo, A. Eggers, A. Hasan, M. Hedges, S. Waddington, and J. Ludwig. A pragmatic approach to significant environment information collection to support object reuse. In *iPRES 2014 proceedings*, 2014.

[5] D. Espenschied, K. Rechert, I. Valizada, D. von Suchodoletz, and N. Russler. Large-Scale Curation and Presentation of CD-ROM Art. In *iPres 2013 10th International Conference on Preservation of Digital Objects*. Biblioteca Nacional de Portugal, 2013.

[6] C. Jones. Seeing double: Emulation in theory and practice. the erl king case study. In *Electronic Media Group, Annual Meeting of the American Institute for Conservation of Historic and Artistic Works. Variable Media Network, Solomon R. Guggenheim Museum*, pages 516–526, 2004.

[7] G. Knight and M. Pennock. Data without meaning: Establishing the significant properties of digital research. *International Journal of Digital Curation*, 4(1), 2008.

[8] P. Laurenson. *Old Media, New Media? Significant Difference and the Conservation of Software Based Art*. New Collecting: Exhibiting and Audiences after New Media Art. Ashgate, 2014.

[9] T. Lurk. Virtualisation as conservation measure. In *Archiving Conference*, volume 2008, pages 221–225. Society for Imaging Science and Technology, 2008.

[10] G. S. C. Mahadev Satyanarayanan, B. Gilbert, Y. Abe, J. Harkes, D. Ryan, E. Linke, and K. Webster. One-click time travel. Technical report, Technical report, Computer Science, Carnegie Mellon University, 2015.

[11] T. Miksa, R. Mayer, and A. Rauber. Ensuring sustainability of web services dependent processes. *Int. J. Comput. Sci. Eng.*, 10(1/2):70–81, Jan. 2015.

[12] K. Rechert, I. Valizada, and D. von Suchodoletz. Future-proof preservation of complex software environments. In *Proceedings of the 9th International Conference on Preservation of Digital Objects (iPRES2012)*, pages 179–183. University of Toronto Faculty of Information, 2012.

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

[14] R. Rinehart. The straw that broke the museum's back? collecting and preserving digital media art works for the next century. SWITCH: Online Journal of New Media. http://switch.sjsu.edu/web/v6n1/articlea.htm, 2002.

[15] R. Rinehart. *Nailing down bits: Digital art and intellectual property*. Canadian Heritage Information Network (CHIN), 2006.

[16] D. S. Rosenthal. Emulation & virtualization as preservation strategies. https://mellon.org/resources/news/articles/emulation-virtualization-preservation-strategies/, 2015.

[17] J. Rothenberg. Ensuring the longevity of digital documents. *Scientific American*, 272(1), 1995.

[18] R. Russell. Virtio: Towards a de-facto standard for virtual i/o devices. *ACM SIGOPS Operating Systems Review*, 42(5):95–103, 2008.

[19] G. Wijers. To emulate or not. *Inside Installations. Theory and Practice in the Care of Complex Artworks*, pages 81–89, 2011.