Evaluation of preservation strategies for an interactive, software-based artwork with complex behavior using the case study *Horizons* (2008) by Geert Mul.

Claudia Roeck
University of Amsterdam
Amsterdam, Netherlands
c.rock@uva.nl

Klaus Rechert
University of Freiburg
Freiburg, Germany
klaus.rechert@rz.uni-freiburg.de

Julia Noordegraaf
University of Amsterdam
Amsterdam, Netherlands
j.j.noordegraaf@uva.nl

ABSTRACT

In order to preserve software-based art the research community has primarily focused on emulation as a preservation strategy. The University of Freiburg (D) established emulation as a service for memory institutions and research data archiving - a service that is currently used for preserving software-based art. Software migration, which might provide an alternative solution, has been researched for business applications, however less for software-based art. As a very immediate strategy it does not introduce an additional layer of translation and thus does not slow down the performance. This paper investigates to what extent the existing migration options are useful for preserving software-based art and how they compare to the emulation options currently used. What long-term impact do migration and emulation have on a software-based artwork? What maintenance works do they cause? What changes do they induce in the artwork?

The impact of migration and emulation as preservation strategies for software-based art is evaluated on the basis of a case study: the software-based artwork *Horizons* (2008) by Dutch artists Geert Mul. This case study shows that it was necessary to migrate *Horizons* first before it could be virtualised with sufficient graphics rendering performance. Hence, this paper concludes, that the combination of migration and emulation can be a good solution for graphics intensive works in the mid-term. It is a step in between short-term solutions like migration and long-term solutions like a full-system emulation; the latter only being possible when the speed advantage of the new hardware is large enough.

ACM Reference Format:
Claudia Roeck, Klaus Rechert, and Julia Noordegraaf. 2018. Evaluation of preservation strategies for an interactive, software-based artwork with complex behavior using the case study *Horizons* (2008) by Geert Mul.. In *Proceedings of (iPres2018).* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The fast obsolescence of hardware and software make continuous adaptations of a software-based artwork indispensable. The question addressed in this paper is what preservation strategies are more efficient or more sustainable in the long-term than others.

Geert Mul's artwork *Horizons* (2008) was chosen as a case study for this research, as its interactivity was difficult to describe and its video and sound were generated in real time, depending on a certain hardware. Geert Mul himself was in search of a long-term preservation strategy, as he comes across obsolescence issues each time he has to reinstall one of his software-based works. *Horizons* seemed to have a setup which is not unique for this artwork and could be applied to other software-based works with a similar setup.

The comparison of two preservation versions carried out for the same artwork is chosen as a research approach in this paper. While the long-term effects of migration and emulation for the preservation of *Horizons* are discussed, their compliance with significant properties of the artwork are also investigated. As artists often do not provide any software specifications or significant properties, the identification of specifications for software-based artworks can be challenging. In order to complete or support the description of the significant properties, a method how to record computer in- and output is suggested. With this method at hand and with criteria for the sustainability of preservation strategies, preservation strategies and their combination can be evaluated. Hopefully, the long-term view on preservation strategies will facilitate future preservation actions.

2 RELATED WORK AND DEFINITIONS

The term "software" in contemporary artworks based on digital technology has been defined in various ways. This paper will either mention software as opposed to hardware. Or it will refer to software as a software artifact in contrast to its software environment such as operating system. The Pericles project mentions a "digital environment" and, due to its interrelationships, calls it a "digital ecosystem" [3]. According to Osterweil software is "non-tangible, and non-physical, but often intended to manage tangibles" [10, p. 266]. He stresses its hierarchical structure and that its components are interconnected and have different purposes. According to him, software requires modification and evolves. All these properties are essential and at the same time a challenge for its preservation.

These challenges can be tackled by means of different preservation strategies. In particular emulation was in the focus of the research about software preservation in the last few years. In 2008 Tabea Lurk and Jürgen Enge promoted the use of emulation as a preservation strategy for software-based art in the sense of encapsulation of the digital object [9]. In 2010, Dirk Suchodeletz proposed a research agenda for the future integration of emulation (in the sense of encapsulation) into preservation workflows [15]. In the same year, Klaus Rechert, Dirk Suchodeletz and Randolph Welte

iPres2018, ,
2018. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

suggested emulation as a web service [12], emulation also proved to be a practical strategy for exhibitions.

In the digital heritage community, migration is discussed rather for audio-visual components such as video tapes, film reels or audio tapes, or for text such as the migration from MS-Word-documents to pdfs, than for software-based art. The migration of software is a topic found more frequently in the business field. For instance, Wagner defines software migration as "splitting and transferring software to a new platform or technology (transformation) - with the goal to meet new requirements and to improve future maintainability. The existing functionality is to be preserved in order to prevent the loss of business knowledge." [16, p. 40] The focus of this definition lies on a smooth transition from the old to the new system and on enhancing functionality. For the preservation of software-based art, these requirements are not in the foreground, but rather the preservation of the artwork's authenticity.

The JISC project [1], carried out in 2006, defined migration of software as transferring a digital object from one software application to another. However, this definition still reminds one of file migration. If the digital object is a complex software that consists of various files and file types that depend on each other, it requires a whole software setup and not just one single application to run them. Therefore we define migration for this paper as follows:

Software migration means the transfer of a software artifact to an updated or different software environment in order to keep its functionality. This transfer can be related to new hardware, but it is not mandatory. It may include conversion (compilation) or adaption of the software to the new environment. The reprogramming of the software functionality with a different programming language is not considered migration in this paper. It is considered as a separate preservation strategy.

When comparing preservation strategies for software-based art, success criteria for long-term preservation are needed. The Library of Congress published sustainability factors for digital file formats which are, however, not completely suitable for software, as a software artifact fulfills a different function (active, computing) than a file format (passive, being processed). Patricia Falcao discussed risks that are inherent in software-based artworks [5], but she does not analyze what impact these risks have on the selection of preservation strategies. This is why this paper identifies success factors for long-term preservation and compares preservation strategies according to these criteria.

Comparing the ways in which different preservation strategies impact the functionality and appearance of software-based art requires the identification of the significant properties of the artwork. Significant properties are well known in the field of software engineering. Wilson defines significant properties in relation to digital preservation: "The characteristics of digital objects that must be preserved over time in order to ensure the continued accessibility, usability, and meaning of the objects, and their capacity to be accepted as evidence of what they purport to record. Properties are considered to exist in one of five categories: content, context, appearance, structure and behavior." [17, p. 8]. Grace further develops the model in so far, that the different needs of stakeholders are taken into account [6]. The model is applicable to software, but it is not broad enough for a software-based artwork as this can be based on specific spatial, temporal, and hardware context. Laurenson fills that gap and lists 15 areas of focus for significant properties for software-based art[1] including Wilson's and Grace's categories [7, p. 92-94]. These areas of focus served as a checklist for the definition of significant properties. As some of these areas of focus were overlapping and some were not relevant for *Horizons*, the significant properties have been summarized under "Idea of the work", "Processes implemented by the software", "Look and Feel" and "Hardware Dependencies" in the following section.

3 SIGNIFICANT PROPERTIES AND DEPENDENCIES OF *HORIZONS* (2008)

This section analyses the artwork *Horizons* (2008) by Geert Mul and determines its significant properties regarding soft- and hardware. These properties will serve as preservation objectives and benchmarks for the evaluation of the preservation strategies applied later. For the preservation of the work, several aspects have to be considered: the overall idea of the work, the processes that are implemented by the software including user interaction, the look and feel of the work which is a combination between software and hardware effects and the dependencies of the software on hardware. The latter has an impact on the preservation of the work. The spatial installation parameters will not be discussed here; although significant for the re-installation of the work, they are not relevant for the choice of preservation strategies.

3.1 Idea of the work

The artwork *Horizons* by the Dutch artist Geert Mul is an interactive multimedia installation that was commissioned by the Museum Boijmans Van Beuningen in Rotterdam, The Netherlands. The work consists of reproductions of landscape paintings from the Boijmans van Beuningen collection in an interactive installation. The horizons of these digitized landscape paintings are all aligned to the same height (s. fig. 1). The concept of the work refers to the project *Search for a Horizon* by the Dutch artist Ger van Elk in 1999, also at the Boijmans van Beuningen Museum, where the artist hung paintings from the Boijmans collection side by side, with their horizons aligned. In his installation *Horizons*, Geert Mul extended this idea by presenting the collection in an interactive, dynamic, playful way. He achieved this by developing a computer program that generates a video of the digitized paintings which is projected to the wall by three digital video projectors (s. fig. 2).

Visitors walking around the space cause the image to split where they stand, revealing a new painting underneath. This interaction is enabled by a tracking sensor that sends the visitor's movement data to the computer. The sound pans seamlessly between right and left speaker depending on the visitor movement and on the number of visitors. The computer, projectors, sensor and speakers are industrial, mass produced, and they are either hidden from view

[1] 15 Areas of Focus for Significant Properties of Software-Based Art: 1- Content and assets. 2- Appearance 3- Context. 4- Other Versions. 5- Formal and Structural Elements. 6- Behaviour. 7- Time (Durations of processes etc.). 8- What are the Spatial or Environmental Parameters of a Work? 9- External Links or Dependencies. 10- Function. 11- Processes. 12- Artist's Documentation of Process (status of this documentation? Relationship to the work?). 13- Rules of Engagement. 14- Visitor Experience. 15- Legal Frameworks

Figure 1: *Horizons* by Geert Mul in the Boijmans van Beuningen Museum 2008 (source: website Geert Mul)

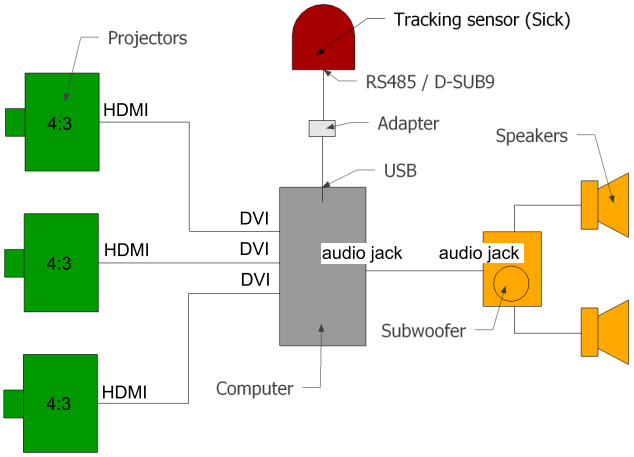

Figure 2: Equipment schematic for *Horizons* by Geert Mul (artwork creation 2008, version 2013)

or positioned in the background. Their physical appearance is not important for the work.

From this description can be concluded, that the interactivity and generative character of the video are significant for the preservation and that the equipment is replaceable, as long as its functionality is maintained. It can also be said, that the software is used as a tool to achieve the effects of the interactivity and video and sound generation, but its significance for the artwork does not extend beyond that as long as its functionality is maintained.

3.2 Processes implemented by the software

In a close collaboration with the artist, the programmer, Carlo Prelz, wrote the software from scratch. Prelz programmed the low level code in C and compiled it into native machine code. He wrote the higher level code in Ruby, which is an interpreted language. The lower level code enables the communication with the tracking sensor and the interpretation of its location data. Additionally, it handles the image composition for the video. The C code is specific for this tracking sensor brand and technology, a time of flight laser sensor. The Ruby code generates the video and the reaction to the movements of the visitor.

When a new visitor enters the space, the computer randomly chooses a new image from the database. It also randomizes the size and orientation of the image. The visitor's horizontal position defines where the new painting appears on the screen: When the visitor moves parallel to the screen, he or she drags along the painting. The movement of the painting is accompanied by vertical ripples running across the screen, similar to a stone falling in the water and causing circular ripples. The visitor's distance from the screen determines the height of the projected painting. If the visitor moves towards the screen, the painting shrinks. If the visitor stops or leaves, the ripples continue to run towards the left and right border of the image until they are completely gone. Without a visitor moving, no new ripples will be triggered and the image is still. When a new visitor enters the space and stands in front of the projection, the algorithm randomly chooses a new sound sample. The horizontal position of the visitor (movement parallel to screen) shifts the volume between left and right speakers. The vertical position of the visitor (movement away from the screen) increases the pitch.

This description of the processes is neither precise nor complete. As the generated video is the product of several programs running at the same time (reading sensor input, random image selection, creation of image columns, reaction to sensor input, etc.), the behavior of the work is difficult to predict. Without following the exact algorithm, the processes cannot be described in a precise and complete manner. As Rothenberg already indicated in 1999: "[...] we do not yet have any formal (or even informal) way of describing the full range of behaviors possible for even the simplest of digital documents, such as are produced by word processing programs. Describing the behavior of dynamic, interactive, hypermedia documents poses a far greater challenge. The only adequate specification of the behavior of a digital document is the one implicit in its interaction with its software. The only way to recreate the behavior of a digital document is to run its original software." [14, p. 22].

The behavior caused by a complex program can have infinite variations and be unpredictable. It is not possible to prove that such a program is complete and correct. As a consequence, although the software (Ruby and C-Code) is only used as a tool in this artwork and thus in itself is not conceptually relevant, the source code is regarded as a significant property of the work as the behavior it causes through its execution is too complex to describe and thus hard to fully replicate in other code.

3.3 Look and Feel

The look and feel of the video is equally difficult to describe. Deducing from the source code, the rate of the video generation is dependent on the image size, the CPU speed and the video card. As a consequence, it is important to determine the speed of the video patterns via visual observation. The image columns (ripples) triggered by the visitor move across half of the screen in about 15 to 30 seconds, which corresponds to a horizontal movement of about 50 to 100 pixels per second, very roughly estimated. For the artist it is

important that the movements in the video are smooth[2]. In addition to its projection speed, the video quality is also a consequence of the quality of the images in the database. The color space, deduced from these images, is RGB, 8 bit per channel, the resolution of the video frame 3072x768. The artist considers the video resolution as a significant property, although he would not exclude upscaling if the current type of projector cannot be used and purchased anymore. The sound quality is determined by the quality of the 13 wav-sound samples, which are stereo, have a bit-depth of 16 bit and a sampling frequency of 44 kHz. The sound resembles a pink noise. As a precise description of the look and feel is impossible, a video and sound recording can serve as a reference documentation.

3.4 Hardware dependencies

In 2017, Geert Mul supplied a standard desktop computer with *Horizons* installed on it to LIMA[3], a platform for preservation, distribution and research of media art, for research purposes. The Intel-based x86/64bit computer was equipped with two built-in Nvidia video cards with each two DVI connectors, and several USB sockets (s. fig. 2). As most files on this computer were created in 2013 and the components of the computer date from 2012/2013, this version will be called version 2013 and serves as a reference. This version is dependent on the Nvidia video card, as this type of video card is hard-coded in the source code of the artwork (s. section 6.1).

For the first version of *Horizons* in 2008, Geert Mul used an infrared camera as a tracking sensor. As this was expensive yet unreliable, the artist looked for a better solution. Currently, he is using a time of flight laser sensor, LMS 200 SICK, which will be called SICK sensor from here onwards. Although this is a sturdy industrial sensor with very basic maintenance requirements, it has to be assumed that this sensor will have to be replaced in the future. As there is no standard protocol for tracking sensors, the driver of the sensor, `sick.c`, was taken over from the media art lab V2, Rotterdam, adapted by the programmer and integrated in the artifact. Thus, the sensor cannot be replaced without additional programming measures.

3.5 Comments regarding the definition of significant properties

The definition of significant properties of an artwork can be contestable; often, they are not measurable or technology dependent qualities that are not applicable any more when the work is transferred to a different technology. Even if one manages to quantify a property, it is difficult to know how big the scope of acceptable deviation is - an aspect that the artist him/herself may also be unable to quantify. Besides, the definition of significant properties is partly dependent on the values of the stakeholders. For instance, a team of curators and conservators might emphasize aesthetic values above the historical values of the work, or the other way round. Grace also recognizes, that there is not one single, definitive interpretation of significance [6, p. 6]. Bettivia confirms this in her research about the significant properties of a video game where she found that different designated communities defined different significant properties for the same game [4]. Finally, the perception of what the significant properties are can shift with each new exhibition, development of technologies and of conservation theory.

The subjectivity and possible shifting of the definition of significant properties can be countered by keeping deprecated artwork instantiations, their documentation and description of significant properties. By keeping the history of the definitions of significant properties transparent, new conservators and curators can make informed decisions about what they consider significant.

Furthermore, if significant properties are either incomplete or hard to verify - as this is the case for *Horizons* - and therefore insufficient for the application of a successful long-term preservation strategy, a viable solution is to keep as much of the "original" as possible, which was aimed at for *Horizons*.

Finally, the mixed approach of technology independent description (description of function, interaction, behavior and processes) and technology dependent properties (color space, video and sound resolution, source code, hardware dependencies) comprises a wide range of properties and therefore supports long-term preservation.

4 CONSIDERATIONS FOR LONG-TERM PRESERVATION

In order to enable sustainable preservation decisions, not only the artwork's significant properties, but also criteria for a successful long-term preservation are needed. This will enable the comparison of preservation strategies. The following paragraphs establish these comparison criteria.

Adaptability to new hardware Hardware abstraction is an important means to facilitate future preservation measures. The more independent the software is from the hardware, the easier it will be in the future to transfer it to new hardware. Even if the transfer of an obsolete software to new hardware is successful, this will most likely require software changes.

Resilience to software obsolescence and adaptability to changed network protocols The resilience towards obsolescence of software applications and programming languages is therefore another important factor for the long-term preservation of software-based artworks. As any hardware change can lead to a change of software requirements and/or configuration, preservation measures that prepare the artwork for or shield it from such changes support long-term preservation. For web-based artworks, the abstraction of protocols is relevant to achieve more independence from changing Internet and data protocols.

Stabilization of software complexity and minimizing change rate Although for the preservation of artworks usually no new features are added, it might be necessary to adapt the software to new hardware. The more changes a software undergoes, the higher is the risk that bugs or deviations from the significant properties are introduced. Preservation measures should therefore try to stabilize the software artifact and reduce its change rate.

Ease of installation of the artwork / of connecting peripherals Ultimately, the goal of preservation is the presentation of the artwork. If the installation is very fragile or if it is difficult to find the right settings in order to adjust it to the space, the risk is higher that

[2]Oral communication with Geert Mul during the installation of Shan Shui (2013) in the week of 28th of October until the 4th of November 2016

[3]LIMA, **Li**ving **M**edia **A**rt, Amsterdam, Netherlands, www.li-ma.nl

it will not match the significant properties. Furthermore, if it is very complex to install, the museum might be inclined to display it less, as the installation costs are high. For these reasons, the ease of installation and the stability of the installation process can be considered success factors for long-term preservation.

Ease of maintenance and of function tests (monitoring) Article 17 of the E.C.C.O. guidelines (II) mentions that the conservator-restorer should specify the most appropriate means of continued care. Maintenance is another, more technical term for this and is essential for the continuation of technology-based art. This is also valid for software-based art, as Lehman and Ramil describe when explaining software evolution: "*In general, the change will be such as to adapt the elements of the class so that they maintain or improve their fitness to a changing environment.*" [8]. The aging of sof tware-based art is not onherent in the software, but rather in the interdependencies of software with hardware and external data sources and protocols. The external environment of the software-based artwork will continue to change, even if the software artifact itself does not. This process will eventually lead to its dysfunction, if no maintenance is undertaken. Maintenance prepares the artwork to be installed at any given time without having to start a restoration project first. Through these small maintenance activities, the artwork's functionality is automatically monitored and big surprises regarding its functionality are prevented. Ease of maintenance and of function tests can be expressed in time spent on these maintenance activities each year, taking into account the time saved for large-scale restoration work.

Scalability of preservation strategy Since tailor-made approaches to the preservation of software-based artworks are time consuming and expensive, more generic solutions such as emulation or virtualization are preferred. Specific hardware requirements might prevent this approach, but some artworks might lend themselves to a more generic approach. This would facilitate their preservation and change management. Just a few emulators would have to be maintained, instead of maintaining and migrating many different software environments. The cost of the emulator maintenance could be shared between the artworks.

5 PRESERVATION OPTIONS FOR *HORIZONS*

The following sections discuss the digital preservation strategies available for *Horizons* and give reasons why a strategy was chosen or rejected for its subsequent execution. Practical arguments of feasibility and available resources were taken into consideration.

5.1 Reprogramming

One preservation option is, to reprogram a software-based artwork in a different programming language and for a current software environment. The reprogramming can either be based on formal software specifications that again are based on the significant properties of the artwork or on formal specifications such as pseudo code, a generalized interpretation of the source code.

The Guggenheim museum provides an interesting example of a reprogrammed work: it reprogrammed parts of Brandon[4], a web-based artwork created in 1998 by Shu Lea Cheang, by directly translating the existing non-functional code into another language; all the Java applets were replaced with GIFs, JavaScript and new HTML [11]. While this measure (in combination with other measures) has the immediate effect of making the artwork operational again, this huge effort will have to be repeated when one of the technologies used becomes outdated.

Reprogramming based on software-specifications requires an exact description of the significant properties of the artwork. In the case of *Horizons*, where several software components interact with each other to produce the effect on the screen, the exact description of the behavior is not possible. Even though complex behavior can be caused by simple functions, these functions can be difficult to recognize. To re-engineer such behavior can therefore require a very substantial effort and the result might not be sufficient.

Reprogramming based on pseudo code would be a very interesting option, as pseudo code could be preserved as text in the long-term without problems. Rinehart who compares the source code of an artwork with a musical score suggests: "A system of formal notation for media art should be abstracted from specific environmental factors. It should be robust, generic, adaptable and portable - universal content for a universal machine." [13] While pseudo code could theoretically adopt the role of such a generic, portable code, the question has to be asked whether pseudo code and the subsequent translation to a new programming language can express all idiosyncrasies of the original language. Finally, for *Horizons*, translating several thousand lines of software into pseudo-code and then reprogramming it in a different language was considered too expensive. These reasons were the reasons why this strategy was renounced.

5.2 Migration.

Moving and integrating a software artifact into a different software environment without having to reprogram it is referred to as migration in this paper (s. section 2). Migration may be necessary due to the updating or complete change of the operating system, or due the replacement of important libraries. They are typically a consequence of the transfer to new hardware. Even though changes in the hardware or software environment may require a recompilation of the source code (if available), this preservation strategy is based on the same source code or binaries and thus, the same algorithms as used for the original work.

The migration of source code was chosen as the easiest and most straight forward migration option. It was also financially feasible and still interesting regarding the effect of the recompilation on the installed work. The operating system Debian was not replaced by a different one, as it is open source and quite common. However, it was chosen to update the operating system from Debian 7 to Debian 9 in order to use a current system and in order to test the effects this has on the significant properties of the artwork. Finally, it was a goal of this research to analyze the migration process.

5.3 Emulation.

An emulator is able to interpret the machine language commands of a specific CPU type or computer architecture. In practice emulators represent a complete computer system, typically including a sound

[4] http://brandon.guggenheim.org/

card, graphics or network adapters. Virtualization is reverted to when the performance of the emulation is not sufficient. In contrast to emulation, the guest system has direct access to the host computer's CPU and the system commands do not have to be translated form guest to host.

The emulation and virtualization were both chosen as one strategy, as they can be executed with the same emulation/virtualization software but with different settings. Although there were concerns regarding the performance and video card dependency, it was decided to test this strategy, as emulation offers the highest abstraction level of all the options mentioned and therefore interesting long-term preservation perspectives.

6 CARRIED OUT PRESERVATION STRATEGIES FOR THE SOFTWARE

6.1 Analysis of dependencies (Version 2013)

As there was no documentation of the software, some detective work had to be undertaken in order to be able to isolate the artifact from its software environment and in order to determine and document its hard- and software dependencies. As a preliminary measure, disk images were taken from the two internal hard drives of the artist's computer.

The disk image enabled the access to the source code and the identification of the folder structure of the project. From this analysis and interviews with the programmer Carlo Prelz, it appeared that the same software was used for several artworks and projects. The programmer made many changes to the software, which resulted in unused source code files located in the same project folder as the *Horizons* project.

Another resource of analysis were error messages on the artist's computer. When the sensor was not connected, the program started and an error message pointing at a central Ruby program popped up. Starting from that program, and following the calls it made for other programs and libraries it was possible to understand the start procedure of *Horizons*. The same method of source code analysis was used to understand which files were part of the *Horizons* software and which libraries it depended on.

The source code analysis showed, that C-programs were wrapped and executed through Ruby. To integrate C-code into Ruby the programmer compiled the C-code as a library that can be used by Ruby. For *Horizons*, this Ruby library is called `boij.so` (s. Fig 3). It integrates amongst others a C-program `sick.c`, which constitutes a manually written driver for the SICK tracking sensor. Furthermore, the programmer added custom-built and non-standard libraries to the standard C- and Ruby libraries. To know their location and the location and version of the compiler and interpreter used is indispensable for a migration.

A further result of source code analysis was, that intermediate hardware abstraction layers such as SDL and OpenGL were used in order to generalize the underlying hardware. Nevertheless, the model of the video card was hard-coded in one of the Ruby programs and in the configuration file of the display manager. The use of the hardware abstraction layers and the fact, that the SICK sensor was connected to the computer through a standard USB interface was the reason, why an emulation approach was considerable.

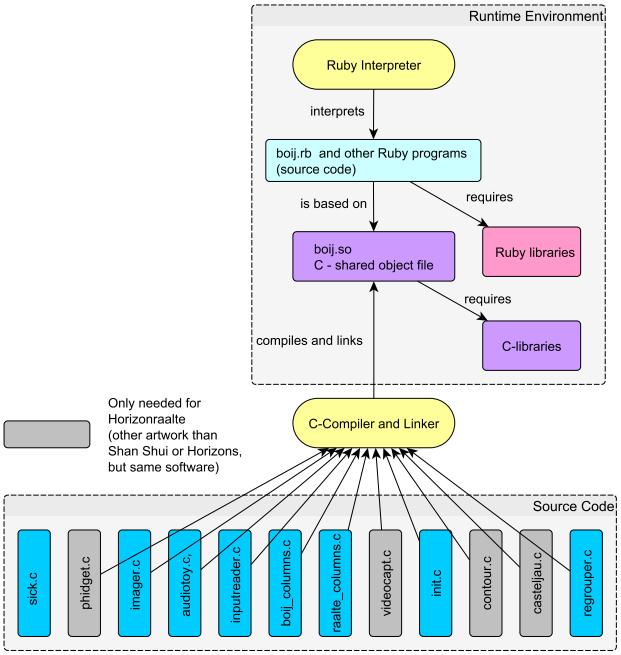

Figure 3: Simplified schema for runtime environment and source code of *Horizons*

A set of 113 images with varying aspect ratio and resolution are used for the video generation. The programmer converted these images previously from the PNG-format to a project-specific *bo2*-format with the Ruby library RMagick. An image database (SQLite) contains a list of ID's of the images used for *Horizons* and the position of their horizons. The sound generation of *Horizons* is based on 13 sound files in WAV-format. The knowledge of these file locations and relations is relevant for a migration of *Horizons*.

The configuration file for the SICK sensor was in binary format and was overwritten each time, the SICK sensor was re-connected to the computer. It contains site specific sensor and software parameters. Thus, the parameters of past exhibitions were neither saved nor were they human readable. From the preservation point of view, this is a lack of the software which could be amended when migrating it (creating a new preservation feature).

Although the programmer installed Debian 7 as on operating system, he integrated an old Linux kernel (2.6) in order to be able to use the LILO boot procedure which he preferred to GRUB, the new standard boot loader for Linux kernels. Based on that boot procedure *Horizons* started up as soon as the computer was switched on and shut down in a controlled way, when the computer was switched off. This boot procedure has to be known when migrating the work.

6.2 Migration version 2017

The migration of *Horizons* not only comprised the update of the operating system, but also the implementation of several other measures supporting the preservation of *Horizons*. In particular, it

was a goal to make the work independent from a specific video card by detecting the video card automatically, so that it could be presented on any computer (with x86/64bit architecture and a capable video card).

The new base-system was built from scratch based on the knowledge gained from the runtime analysis described above and in fig. 3. The most current operating system (Debian 9 / Linux kernel 4.9) was installed in order to have a stable version that is able to deal with new computer hardware and to avoid having to update soon after the migration (OS of the 2013 version: Debian 7, kernel 2.6). This operating system update also included a new version of the Ruby interpreter and the C-compiler.

As a next step, the source code, the images, the image database and the sound were copied over. Not all data in the artist's data folder were transfered, as some directories contained other projects that were not directly related to *Horizons*. The separation and deletion of unrelated data is an ongoing task that has not been finished. Cleaning up decreases maintenance and increases the understanding of future conservators. However, throwing out seemingly unrelated data does entail a risk of breaking the code and of deleting the project history. This is why it is important to use a version control system for the source code, a solution that can support the principle of reversibility that is central to arts conservation.

After the transfer of the source code its C-library was recompiled, to match updated ABIs[5] of the new system. The Ruby programs did not have to be recompiled, as they are interpreted on the go.

For this migration, the tracking sensor was not replaced by a new model. Due to the analysis of the software in fig. 3 it is clear, though, that the `sick.c` file (driver of tracking sensor) would have to be adapted and recompiled should the sensor be replaced.

The procedure to boot the *Horizons* version 2013 was based on the LILO boot loader. As the development of the LILO boot loader ended in 2015, the automated startup procedure of *Horizons* had to be adapted to use the GRUB bootloader. The change of the automated startup procedure also included the configuration of a new display manager (NODM) to invoke the graphics window for the video.

This step-by-step reconstruction of the new system indicates that, to a certain extent, it was possible to separate the software artifact from its software environment. This knowledge is useful for future migrations and the technical understanding of the work as it will be possible to establish whether the replacement of certain software components (libraries, operating systems) is possible.

6.3 Emulation / Virtualization version 2017

Emulation (virtualization) was chosen as a second preservation strategy as it makes *Horizons* more independent from hardware than migration. The original goal was, therefore, not to change the software and its environment by running a disk image of the internal hard drive of the 2013 version in the emulator.

Graphic rendering, as used in *Horizons*, is computational intensive and was therefore offloaded to specialized hardware, a graphics processing unit (GPU) usually part of high-end video cards, to increase the rendering performance. In theory, GPUs can be emulated, i.e. the functionality of GPUs can be implemented in software which is then executed on a generic CPU. GPUs provide highly specialized functionality, optimized for specific tasks, such that a generic GPU emulation – even for older GPU models – provide a poor performance in practice. We tried to run the emulated setup using software rendering only, but the graphics rendering was not sufficient. In order to provide GPU functionality for the emulated / virtualized environment a different approach is necessary.

6.3.1 Emulation options for the graphics rendering. Qemu was chosen to emulate (virtualize) *Horizons*. Qemu and similar emulation and virtualiziation system are able to use the host's CPU to execute binary code directly (CPU virtualization) for performance reasons. For preservation purposes, virtualization is a technology to bridge the gap until either CPUs are fast enough to emulate even contemporary software or the current architecture changes significantly, such direct execution becomes impossible.

A similar approach for GPUs is required. One potential option is a GPU pass-trough, as used by public Cloud providers today[6]. With GPU pass-through, the original vendor driver is installed in the guest system, which however, ties the emulated instance again to a specific hardware setup. An alternative option, is to install a generic *virtual GPU* within the guest system and delegate the adaption to the host's GPU hardware to the emulator implementation. The Virgil3D GPU project[7] added such functionality to the Qemu emulator recently. When using virtual hardware, a new guest driver has to be implemented (in contrast to emulated hardware for which the original vendor drivers could be used). On Linux systems, this driver is available since kernel version 4.4.[8] As the Linux kernel of the artist's version was 2.6, the kernel had to be updated. Instead of only updating the kernel, we decided to use the migrated version 2017, as it already contained an updated kernel (Debian 9.1 including a Linux kernel 4.9). This allowed a direct comparison of the same software environment running on real hardware with an emulated instance. Thus, not the 2013 version of *Horizons*, but the migrated 2017 version of *Horizons* was emulated (virtualized). In other terms, the emulation/virtualization contains the migrated version in the guest system.

Fig. 4 illustrates the emulated system setup of *Horizons*. The footnote[9] below gives the qemu command that is necessary to start the virtual machine. An overview of options for the handling of GPUs in emulation/virtualisation environments is given in fig. 5.

6.3.2 Emulation options for the tracking sensor. The connection of the SICK sensor through a USB connector to the emulating laptop did not raise any concerns. The USB-input of the sensor was passed through to the guest system in the virtual machine without problems. With regard to the sensor usage, emulation /

[5] Application Binary Interface

[6] E.g. Announcing GPUs for Google Cloud Platform, https://cloudplatform.googleblog.com/2016/11/announcing-GPUs-for-Google-Cloud-Platform.html

[7] https://virgil3d.github.io/

[8] A driver for Windows guest is currently under development: https://gist.github.com/Keenuts/

[9] qemu command line for Horizons with accelerated graphics rendering: QEMU_AUDIO_DRV=pa qemu-system-x86_64 -enable-kvm -cpu host -m 4096 -vga virtio -display gtk,gl=on -machine smm=off -usbdevice tablet -netdev user,id=net0,hostfwd=tcp::22222-:22 -device e1000,netdev=net0 -soundhw hda -drive if=virtio,file=$dir/shan-shui.qcow2,cache=none -k en-us -usb "$@"

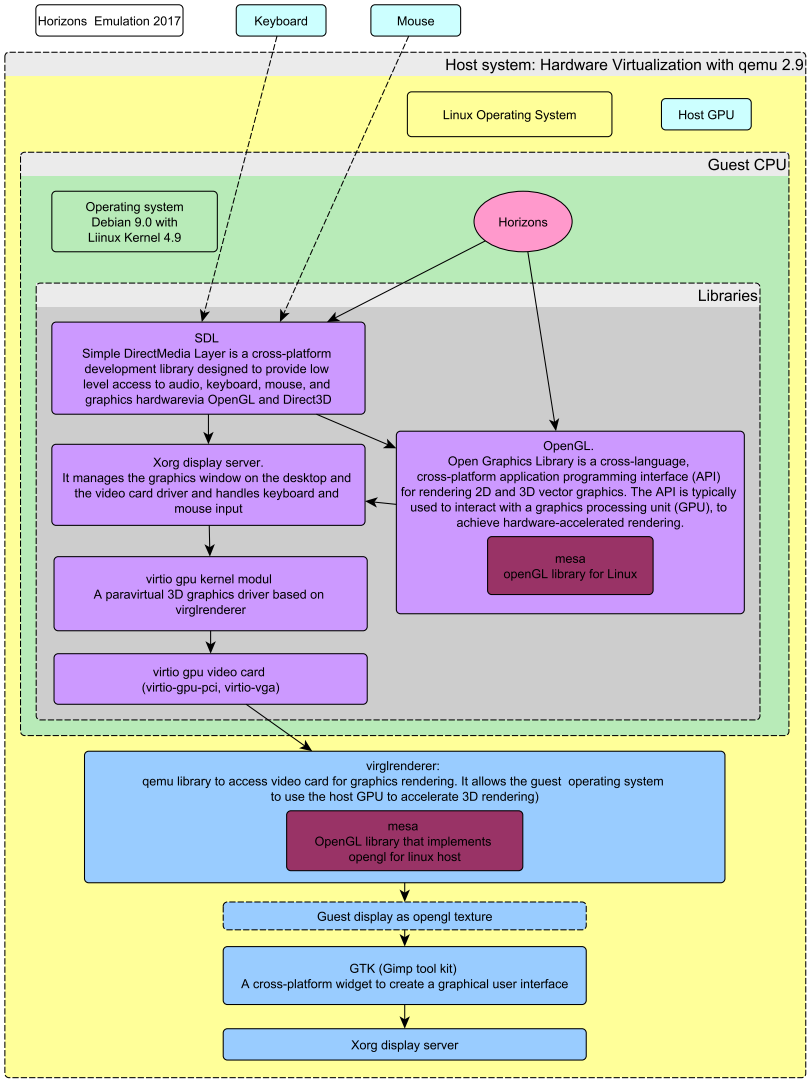

Figure 4: Virtualization of *Horizons* by migrating it from Debian 7 to Debian 9 with qemu

virtualization is a valid option. However, this does not solve the problem of the SICK sensor being replaced by a different sensor type. The SICK sensor is interwoven with the software artifact that is able to read the specific SICK sensor protocol (a manually written driver). If the sensor is replaced by a sensor with a different protocol, which is likely in the future, the software will not be able to parse it. Ideally – for preservation purposes – changes of the artifact or software setup should be avoided when replacing the sensor. This could be achieved with an additional software interface that translates the data of different sensor technologies and brands to the current SICK data protocol. While such an approach requires additional effort, adaptations of the software for the sake of preservation could be clearly held apart. Fig. 6 depicts these options for the integration of the tracking sensor in an emulated/virtualized system.

7 SENSOR AND SCREEN RECORDINGS AS A METHOD TO DOCUMENT AND COMPARE ELUSIVE SIGNIFICANT PROPERTIES

7.1 Sensor recording: stabilizing the input

Certain significant properties are difficult to quantify. For *Horizons*, the behavior of the generated video is difficult to grasp due to the indefinite possibilities of visitor behavior, due to the random processes built-in the software, and due to the complex interplay of the programs/processes. In addition, it is difficult to quantify the smooth rendering of the video, a property that is important to the artist. When comparing preservation versions or artwork instantiations, these elusive properties are important for their evaluation.

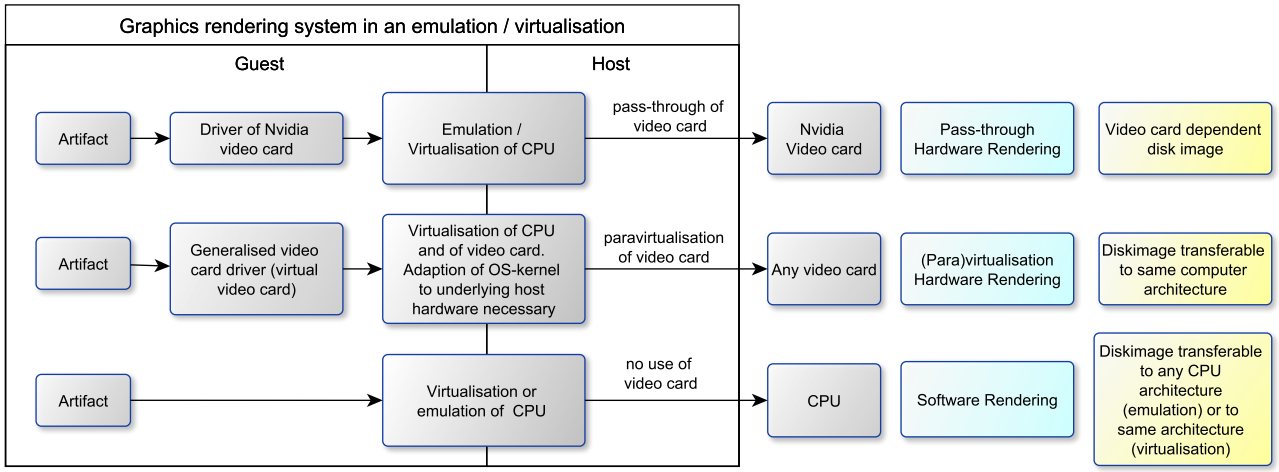

Figure 5: Emulation options for the graphics rendering of *Horizons*

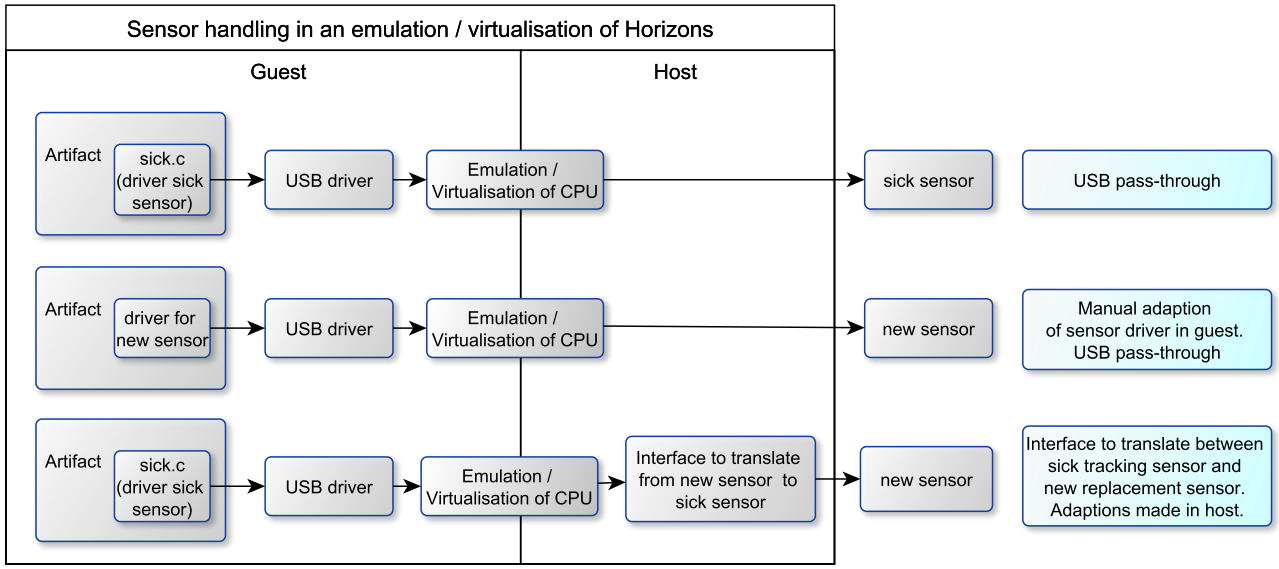

Figure 6: Emulation options for the tracking sensor of *Horizons*

Thus, as a quality control for carried out preservation measures we are suggesting to compare the video and sound output of different instantiations of *Horizons*. Yet, in order to receive comparable output, it would be necessary to carry out this comparison in the same space and based on the same person's walk across the space. This procedure is not completely reproducible. Consequently, the setup of *Horizons* was adapted, so that it became possible to record a person's walk, including the information about the space which the software considers as background. It also includes the specific program settings that were chosen in the "look-around" configuration menu of the custom-made software. All this information is saved in a file that can be replayed by the *Horizons* software instead of using the sensor input. This stabilized input facilitates the comparison of different artwork instantiations.

The sensor input and therefore the file recording is sensor-model-specific. Different sensor models or technologies can therefore only be compared with each other in this way if an interface bridges the difference between the two sensor protocols as suggested in section 6.3.2. This interface provides the flexibility to make and use recordings with a new sensor while it is still possible to use the old sick sensor recording. As long as the program that interprets the sensor input file does not change, any other software changes can be tested.

7.2 Screen and sound recording of stabilized input

The sensor recording enables the comparison of different screen and sound recordings, as it is now possible to produce them from the same sensor input. From the tests carried out it can be concluded that the screen recording in conjunction with sensor input recording works very well in order to assess the generated video patterns. It is more precise than comparing documentary video recordings, which are made in various angles and with undefined visitor movements. Still, the smoothness of the movement cannot be checked with this procedure, as the process of screen recording reduces the graphics rendering performance. The impact of the screen recording on the rendering quality was noticeable, while running the artwork without screen recording yielded a smooth rendering.

For sound recording a special audio driver had to be installed in the guest system and the emulator's sound had to be redirected to that virtual driver. For the 2013 version the audio could only be captured through the analogue computer output, which resulted in a very poor recording quality.

7.3 Compliance with significant properties of the artwork

The comparison of the videos and sound generated by different artwork instantiations was possible due to the recording of the sensor input. However, when comparing video and sound, one has to be aware, that the software randomly picks an image of varying resolution and a sound sample of varying length. This also has a certain, subordinate impact on the video and sound pattern and can be recognized by comparing several screen recordings of the same sensor input and the same hard- and software setup.

Compared to the 2013 version, the migrated version shows some deviations in the video pattern that are only noticeable in a direct comparison of screen-shots (s. Fig. 7). For instance, at the moment of the snapshot taken in Fig. 7, a fourth image is appearing in the migrated version whereas it is not visible in the 2013 version.

The video patterns of the emulated version are very similar to the ones of the migrated version. This was to be expected, as the emulated version contains the migrated version in the guest system. However, the sound playback of the emulated version was not satisfying, because it contained a disturbing crackling. This disturbance of the sound quality would not be acceptable in an official representation of the artwork. As the sound recording of the emulated version was of a good quality, it might be either a performance (sound dropouts) or a driver problem. Having tested several Qemu audio drivers and different emulated audio hardware, none of them yielded a satisfactory result. Other tests passing the sound through to an external USB audio card did not yield better results either. Hence, the work cannot be presented within an emulator at the moment. Sound performance issues are a known problem for Qemu, so is likely that it will be resolved in the future.

While the recording method served to compare the look and feel of the emulated and the migrated version, the compliance with other significant properties resulting from the idea of the work and from the processes implemented by the software had to be checked, too. With both migration and emulation, the interactive

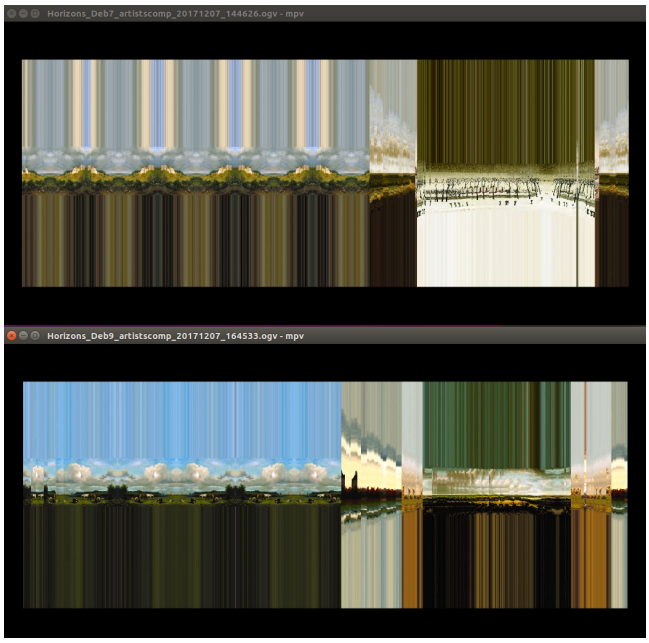

Figure 7: : Corresponding pattern (of different paintings) in screen recordings made on the artist's computer. On the top *Horizons* version 2013, below the migrated version 2017.

nature of the work is preserved, meaning, that both react to a sensor input. Furthermore, both are directly executing the original code. This does not only comply with the significant property of code execution, but is also in line with article 8 of the E.C.C.O. directive which states, that preventive conservation should prevail over "physical" work and that treatment should be limited to what is necessary[2]. Except for the sound rendering in the emulation, it can be summarized, that both the emulated and migrated version sufficiently complied with the significant properties.

8 DISCUSSION OF PRESERVATION STRATEGIES CARRIED OUT FOR *HORIZONS*

The preservation strategies applied to our use-case deal differently with hardware and software dependencies. The main differences between an emulation and migration strategy, however, are found in future, reoccurring, preservation tasks. The migration strategy runs the risk of having to adapt the source code or replace the programming language in the future, as programming languages evolve or become obsolete. In a similar way the graphic system used (SDL/OpenGL/X11) eventually requires substitutes. These tasks are labor-intensive, difficult to automate and their complexity is difficult to predict. It has to be assumed, that all these updates and changes could have a perceivable accumulated effect on the generation of the *Horizons* video and on the complexity of the source code, as already the jump from Debian 7 to Debian 9 was visible (but not audible due to the noise quality of the sound). Hence, the complexity of the software artifact definitely increases with the migration strategy in contrast to the emulation strategy that requires fewer changes

and slows down the growth of complexity. However, the emulator itself might become obsolete and require a substitute.

At the moment, the emulated version does not provide many advantages over the migrated version. It is rather a proof of concept. Despite of the current performance restriction, the emulation provides a better starting point for future, reoccurring, preservation work as it provides a higher level of abstraction. With the rapid improvement of hardware performance, there is a good chance that a pure emulation will soon be possible and adaptations in the guest system will no longer be necessary or less invasive. The most important advantage of the emulated version is that preservation work (e.g. its transfer to new hardware) is usually possible without specific knowledge of the artwork. In contrast, a migration does require more detailed knowledge about the identity and function of the software artifact and of its software environment.

For the installation of *Horizons* the connection of peripherals pose the biggest risk. In the long-term, both preservation strategies will have to cope with new sensors and projectors. The installation of the projectors with the correct aspect ratio and resolution has not been tested with either of the preservation versions and might take some fine-tuning, especially for the emulated version, as the video signal will have to be split onto two to three projectors. The connection of the SICK sensor worked for both versions. The connection of speakers is not expected to cause any problems for the migrated version. For the emulated version, the sound playback will have to be resolved.

9 CONCLUSIONS

The behavior of an interactive, software-based artwork such as *Horizons* is too complex to describe it so precisely, that the software could be reproduced from it. In addition, as the source code consists of several thousand lines, the translation into pseudo-code and from there into another programming language would be expensive, and the result would most likely deviate from the use of the original source code. These practical reasons led to the definition of the source code as a significant property. As Pip Laurenson describes in [7, p. 93], significant properties can also include dependencies. The source code can be seen as a dependency (as it is in practice difficult to replace it), but also as a detailed notation of the processes and algorithms.

There is no clear winner emerging from the above evaluation of preservation strategies mainly due to the fact, that the carried out emulation version relies on virtualized hardware and remains computer architecture dependent, for performance reasons. The most important advantage of the emulated version – essentially a result of combining migration and emulation – is that its transfer to new hardware is possible without specific knowledge of the artwork in contrast to the migration, where this is needed. Furthermore, the change rate of the software artifact is reduced. The combination or merging of emulation and migration is therefore a strategy which can be successful in the mid-term, as long as the new hardware does not have enough speed advantage to enable a full-system-emulation. Furthermore, the emulation of the migrated version allows the identification of emulation effects and hardware dependencies and serves as a sandbox for the migration.

With the emulator, the artwork is encapsulated and thus its behavior does not need to be understood and described in every detail. In order to still be capable of comparing preservation versions, elusive significant properties can be partially captured by recording video and sound from a stabilized input.

The encapsulation has another benefit: it protects and separates the artwork from changes later introduced. The sensor interface that could translate other sensors to the current sick sensor protocol would support the replacement of the tracking sensor. This interface and its connection to the emulator is left for future research.

Thus, this research shows that the combination of migration and emulation can be a good solution for interactive, graphics intensive works in the mid-term. It steps in between short-term solutions like migration and long-term solutions like a full-system emulation.

10 ACKNOWLEDGEMENTS

This research was funded as part of NACCA (New Approaches in the Conservation of Contemporary Art, www.nacca.eu), a Marie Skłodowska-Curie Innovative Training Network of the European Union. LIMA, Amsterdam (www.li-ma.nl) provided the infrastructure and secondment placement for this research, Geert Mul made his artwork including hard- and software available, and Teal Gaure provided programming support.

REFERENCES

[1] 02/01/2008. Digital preservation strategies. (02/01/2008). http://www.paradigm.ac.uk/workbook/preservation-strategies/selecting-strategy.html
[2] 2003. E.C.C.O. Professional Guidelines (II): Code of Ethics. (2003).
[3] 2017. Acting on Change: Model-Âŋdriven Management of Evolving Digital Ecosystems: White Paper. (2017).
[4] Rhiannon Bettivia. 2016. Mapping Significance of Video Games in OAIS. In *iPRES 2016 - 13th International Conference on Digital Preservation*.
[5] Patricia Falcão. 2010. *Developing a Risk Assessment Tool for the conservation of software-based artworks.* Master's Thesis. Hochschule der Künste Bern, Bern, Switzerland.
[6] Stephen Grace. 2009. Investigating the Significant Properties of Electronic Content over Time. (2009).
[7] Pip Laurenson. 2014. Old Media, New Media? Significant Difference and the Conservation of Software-Based Art. In *New collecting*, Beryl Graham (Ed.). Ashgate Publishing Limited, Surrey (UK) / Burlington (USA), 73–96.
[8] Meir M. Lehman and Juan F. Ramil. 2002. Software Evolution and Software Evolution Processes. *Annals of Software Engineering* 14, 1 (2002), 275–309. https://doi.org/10.1023/A:1020557525901
[9] Tabea Lurk. 2008. Virtualisation as a conservation measure: Contribution to the Handling of Born Digital Media Art. (2008).
[10] Len Osterweil. 2008. What is software? *Springer Science+Business Media* (2008).
[11] Joanna Phillips, Deena Engels, Emma Dickson, and Jonathan Farbowitz. 2017. Restoring Brandon. Shu Lea Cheang's Early Web Artwork. (2017). https://www.guggenheim.org/blogs/checklist/restoring-brandon-shu-lea-cheangs-early-web-artwork
[12] Klaus Rechert, Dirk von Suchodeletz, and Randolph Welte. 2010. Emulation Based Services in Digital Preservation. In *Proceedings of the 10th annual joint conference on Digital libraries*, Jane Hunter (Ed.). ACM, New York, NY.
[13] Richard Rinehart. 2007. The Media Art Notation System: Documenting and Preserving Digital/Media Art. *Leonardo* 40, 2 (2007), 181–187.
[14] Jeff Rothenberg. 1999. *Avoiding technological quicksand: Finding a viable technical foundation for digital preservation a report to the Council on Library and Information Resources.* Council on Library and Information Resources, Washington DC.
[15] Dirk von Suchodoletz. 2010. Future Emulation and Automation Research Agenda. *Dagstuhl Seminar Proceedings. Automation in Digital Preservation* (2010).
[16] Christian Wagner. 2014. *Model-Driven Software Migration: A Methodology: Reengineering, Recovery and Modernization of Legacy Systems* (aufl. 2014 ed.). Springer Fachmedien Wiesbaden, Wiesbaden.
[17] Andrew Wilson. 2007. Significant Properties Report: InSPECT Work Package 2.2. (2007).