TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

DISSERTATION

Establishing and Verifying Authentic Performances of Digital Objects: A Framework and Process for Evaluating Digital Preservation Actions

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Doktors der technischen Wissenschaften
unter der Leitung von

ao.univ.Prof. Dr. Andreas Rauber
E188
Institut für Softwaretechnik und Interaktive Systeme

eingereicht an der Technischen Universität Wien
Fakultät für Informatik
von

Dipl.-Ing. Mark Guttenbrunner
9325367
Tongasse 6/7, 1030 Wien

Wien, am *16. Februar 2014*

Kurzfassung

Museen, Archive und Bibliotheken sehen sich immer öfter mit der Aufgabe konfrontiert, nicht nur analoge Daten auf Papier oder physische Kunstwerke zu bewahren, sondern auch kulturelle Objekte in digitaler Form in ihren Sammlungen auf viele Jahre bewahren zu müssen. Dabei geht es nicht nur darum, dass diese digitalen Daten gespeichert werden, sie müssen auch in einer Form wiedergegeben können, die möglichst dem ursprünglichen Zweck der Daten entspricht. Doch nicht nur kulturelle Objekte müssen bewahrt werden, auch Arbeitsabläufe und automatische Geschäftsprozesse müssen bewahrt werden, um sie zu späteren Zeitpunkten zum Beispiel zur Beweisführung vor Gerichten wieder aktivieren zu können.

Es gibt prinzipiell zwei verschiedene Verfahren, um digitale Daten über einen langen Zeitraum zugänglich zu halten. Entweder werden die Daten auf immer neue Formate migriert, um sie mittels aktueller Software anzeigen zu können, oder es wird die Umgebung, mit der sie angezeigt werden, als virtuelle Umgebung erhalten. Dabei werden bestimmte Schichten, die notwendig sind um ein Objekt anzuzeigen, ausgetauscht oder sogar neu eingeführt. So kann zum Beispiel die Hardware eines Systems durch einen Emulator ersetzt werden, der es erlaubt, die Software, die notwendig ist ein Objekt anzuzeigen, auf neuer Hardware auszuführen. Doch unabhängig davon, ob ein digitales Objekt auf ein neues Format migriert wird oder ob ein Emulator eingesetzt wird, um das Objekt weiterhin anzuzeigen, ist es notwendig, das Ergebnis der Aktion, die ausgeführt wird, zu prüfen. Es muss festgestellt werden, ob die Eigenschaften eines Objekts das angezeigt wird durch die Aktion verändert werden, und in welchem Ausmaß. Erst wenn der Einfluss der Aktion auf die sogenannten signifikanten Eigenschaften eines Objekts bekannt ist, kann man entscheiden, ob eine Aktion zulässig ist, oder die Darstellung eines Objekts in einer Art und Weise verändert, die nicht akzeptabel ist.

Die Forschung konzentrierte sich bisher darauf, wie Vergleiche zwischen migrierten Objekten angestellt werden können, um festzustellen, ob bei einer Migration signifikante Eigenschaften eines Objekts verändert wurden. Die Notwendigkeit, bei Emulation, bei der das Objekt nicht verändert wird, Vergleiche zwischen den dargestellten Objekten anzustellen, ist bekannt, allerdings gibt es keine Vorgehensmodelle, wie solche Vergleiche angestellt werden können.

Im Rahmen dieser Dissertation wird ein Vorgehensmodell (das Preservation Action Evaluation Framework) erstellt, das es erlaubt, die Darstellung eines Objekts in verschiedenen Umgebungen zu vergleichen. In einem ersten Schritt wird ausgeführt, dass es eigentlich keinen Unterschied macht, ob ein Objekt migriert wird, oder die Umgebung zur Darstellung eines Objekts emuliert wird. In beiden Fällen muss nicht das Objekt selbst sondern seine dargestellte Form verglichen

werden. Wir beschreiben, welche Informationen über ein Objekt und seine Umgebung gesammelt werden müssen, und an welchen Stellen in einem System ein Objekt verglichen werden kann. Anschließend zeigen wir, wie die zur Darstellung eines Objekts notwendige Umgebung neu aufgebaut werden kann, und was notwendig ist, um Vergleiche anstellen zu können. Wir beschreiben welche externen Einflüsse auf die Ausführung eines Objekts abgefangen und bei der neuerlichen Ausführung wieder zugeführt werden müssen, um eine deterministische Ausführung zu garantieren und Bedingungen zu erzeugen, die einen Vergleich erlauben. Wir beschreiben weiters, wie das Framework in einen Preservation Workflow eingebunden werden kann und in den verschiedenen Phasen des Workflows zur Anwendung kommt.

Da eine automatische Evaluierung der Darstellung digitaler Objekte nur möglich ist, wenn diese auch zu einem gewissen Grad von den ausführenden Umgebungen unterstützt ist, zeigen wir auf, welche Funktionen diese implementieren sollten, um sowohl den Datenaustausch zwischen dem Host-System und dem Gast-System, als auch die Evaluierung zu ermöglichen.

In weiterer Folge stellen wir einen obsoleten Heim-Computer aus den 80er Jahren vor. Wir zeigen wie wir ein Programm entwickelten, das es uns erlaubt, auf Audiokassetten gespeicherte Daten ohne Verwendung des Orignalsystems auszulesen, und in nicht-obsoleten Formaten zu speichern. Da einige dieser Daten allerdings Programme sind, die nicht ohne weiteres migriert werden können, entwickelten wir einen Emulator der diese Programme auf modernen Systemen lauffähig macht. Die für eine automatische Evaluierung notwendigen Funktionen wurden in den Emulator implementiert. Anschließend zeigen wir, wie anhand des Frameworks die Ausführung von zwei Objekten im Emulator evaluiert wurde, und zwar einem Geschäftsprozess und einem Videospiel. Für zwei weitere Objekte auf anderen Systemen (einem Musikklassifizierungsprozess und digitaler Kunst) zeigen wir die notwendigen Schritte einer Evaluierung im Framework.

Durch die erfolgreiche Evaluierung wurde die Validität des Frameworks und die praktische Umsetzbarkeit gezeigt. Zum Abschluss diskutieren wir noch notwendige weitere Forschungsarbeit im Bereich der Evaluierung von virtuellen Umgebungen.

Abstract

Museums, archive and libraries are often confronted with the challenge to preserve not just analogue data, but also digital objects for the long term. While one challenge is to store the data for a long term, an even more difficultly one is to keep it accessible. Digital objects have to be rendered as close as possible to how they were originally used. But not only cultural objects, also business processes and scientific data have to be kept for future generations, either for legal purposes or to reproduce scientific experiments.

Two major strategies exist to keep digital objects useable over the long term. Either a digital object is migrated from an obsolete format to a format that can be rendered using modern hardware and software, or the rendering environment of the object is preserved by creating a virtual version of the environment. Virtualizing an environment means replacing various layers in the stack of software and hardware rendering the object with different representations of this layer. For example, the hardware of a system can be replaced by a software emulator of this hardware, allowing us to render the object in a new hardware environment. Independently of the strategy used to preserve a digital object, the actual preservation action has to be evaluated. We have to determine, if the significant properties of a digital object are changed by applying the preservation action, and to what degree. Only if the influence of a preservation action on the significant properties of an object is known, a preservation planner can decide if the preservation action is valid, or if the properties are changed in a way that is not acceptable for the preservation purpose.

Previous research focussed on how to compare digital objects before and after a migration action, concentrating on the properties stored in the format of the object. Frameworks for performing an evaluation of objects rendered in a virtual environment do not exist. In this thesis we create a framework that allows the comparison of renderings of a digital object in different rendering environments, called the “Preservation Action Evaluation Framework”. First, we show that even for migrated objects a comparison has to be done on the level of the object being rendered, as not only the object properties change, but, identically to an emulation strategy, layers in the view-path used to render the object change. We describe the information that has to be collected about an object and its original environment. We show the different forms of a rendered object that exist in a virtual environment, to determine when and where a digital object’s significant properties can be compared. Next, we show the steps necessary to ensure that any differences in the rendering are caused by the rendering environment and not by changed external events influencing the rendering of the object. We also show, how the framework can be used in the different steps of a preservation workflow to evaluate the rendering of an object in the plan, preserve and re-deploy phase. As automatic evaluation has to be supported by the rendering environment to some degree, we

show guidelines to be considered when developing a virtual environment, including data exchange between a host and a guest system.

Next, we introduce an obsolete home computer used for video games and business processes. We show the implementation of a tool that allows us to migrate data stored on audio tapes to non-obsolete formats without use of the original system. As some of this data are programs, we develop an emulator that is able to execute these programs, considering the guidelines for evaluation and data-exchange. We then show how the framework is used to evaluate the rendering of different objects in the emulator in the context of a preservation workflow. We also discuss the framework's application on two more more recent objects, a scientific process and a digital artwork.

The successful evaluation of the case studies shows the validity of our framework and its implementation in a virtual environment. Finally, we discuss current and future work connected to the work shown in this thesis.

Acknowledgments

This thesis would not have seen the light of day without the support of my friends, colleagues, and family.

A special thank you goes to my adviser Andreas Rauber. His input, motivation and challenging questions continuously improved my work and helped making this thesis a reality. My colleagues I worked with on the various projects also provided valuable input and were always a source of inspiration.

Thank you also to all my friends that supported me during the long years of my studies. Without them pushing me to finish, I would probably not have had the motivation to keep going on. And thank you to my parents for everything, but especially for giving me the possibility to study and supporting me in this decision.

Part of this work was supported by the European Commission in the 6th and 7th Framework Programs, IST, through the PLANETS project, contract 033789 and the TIMBUS project, contract ICT-269940, as well as the COMET K1, FFG - Austrian Research Promotion Agency. Without the discussions with partners in and outside the projects, this work would not have been possible.

So long, and thanks for all the fish.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Preserving cultural heritage has been a major task of libraries, archives, and museums for centuries. Preservation methods for traditional materials like books and paper are well known and proven to work. The last few decades, however, saw an ever increasing amount of digital data. Digital objects in contrast to analogue materials by nature require information technology systems to execute them. But information technology systems are volatile. With the rapid change in technology, systems that are able to execute/render a digital object today will be obsolete tomorrow, in many cases leaving the digital object unusable on future systems. For static documents methods like printing the document to paper are a possible solution, but when it comes to more complex digital objects, keeping the digital object in a usable state is the only solution to preserve the object for future use. There are different reasons why a digital object might need to be preserved for future use. While the need of memory institutions to preserve cultural heritage is obvious, including complex objects like digital art or video games, it is less obvious that also processes have to be kept in a form that is executable in a future environment. These include business processes for legal purposes, or scientific processes that allow future scientists to recreate experiments and retrace and reuse previous work. But also individuals will face the problem of digital preservation, if precious emails, letters or pictures can not be opened on a future system.

Dealing with the challenge of keeping digital objects accessible in an authentic manner over a long term is the task of digital preservation. Researching a framework and process for evaluating and validating the result of a digital preservation action on the object to authentically preserve it is the topic of this thesis.

1.1 Digital Preservation

In [Rothenberg, 2000a] Rothenberg argues that meaningful digital preservation implies that what is preserved is usable. This includes that the digital object can be retrieved, accessed, deciphered, viewed, interpreted, understood, and experienced in a meaningful and valid, i.e., authentic, way. If the digital object or information entity is not usable in a meaningful and valid way, then it is not preserved at all.

Digital objects are threatened on different levels:

- the physical layer, i.e., the bitstream of the digital object,
- the logical format and environment, i.e., the conceptual object as rendered on the screen,
- the semantically conceived object and contextual knowledge, i.e., the information or knowledge transmitted by the object or the correct use of the object.

While traditional archiving deals with data being copied to new media and refreshing bits on physical media to keep the data readable, the challenge of the logical layer is still a rather new and an in its severeness underestimated problem. To actually being able to keep the object available on the conceptual layer, i.e., conceivable for a user, the object has to be interpretable on the logical layer. With hardware and software getting obsolete, we have two main choices to tackle the problem. One choice is to keep migrating the digital object to a form that is interpretable by information systems at the time of use. The other is to keep the digital object in its original logical form and try to keep the information system alive in physical or virtual form. These two strategies are called *Migration* and *Emulation*. (Note: For an analysis on their differences and identity, see Chapter 3.)

Which strategy is chosen depends on the situation and also the digital object itself. While it might be a rather simple task to convert an image from one format to another (with all the pitfalls involved like change in colors, loss of information if no lossless format is used as target format, etc.), it might not be possible for software, especially if the source code is not available. In the case of digital art or video games interaction with the object plays an important part in the conception of the object by the user. So whichever strategy is chosen, it is important that the conceptual layer of the digital object stays intact.

While certain loss might be acceptable depending on the intended audience (e.g., if only the textual information in a document is important to a user group, it might be acceptable to lose any formatting information like font and page breaks, as long as the information transmitted by the text is not compromised) in most

cases it is important that the digital object's rendering stays unchanged (though usually, some loss will be acceptable). A major task in digital preservation is thus evaluating if and how a digital object's conceptual layer changes when a preservation action, be it migration or emulation, is applied to the digital object.

Evaluation if a digital object can still be rendered successfully is usually done by checking the properties of the object stored in the bitstream. While this is an indication for a migrated object if all the properties (or the significant ones) are still in the bitstream, it does not necessarily mean that the object is rendered as intended by the interpreting software. When dealing with emulation, on the other hand, the object's bitstream stays unchanged on the logical layer, so any evaluation necessarily has to be made after the object has been rendered. Dealing with the evaluation of this rendering brings us to the problems and research questions outlined in the next section.

In [Innocenti, 2012] Innocenti argues that digital art is being rendered as a performance specific for the viewer of the artwork (especially in interactive art where the viewer influences the behavior of the artwork). Preserving the performance and thus the authenticity of the digital artwork is crucial to long term digital preservation of digital art. In [Bonardi and Barthélemy, 2008] examples for the fragility of performance works based on electronics under the aspect of re-performance are provided and the question is raised, how to guarantee authenticity when preserving the electronic material.

One of the core tasks of digital preservation is thus to make sure that a digital preservation action preserves the digital object in a meaningful and valid way, i.e., evaluating if a future rendering will preserve the significant properties of the original rendering of the digital object.

1.2 Problems and Research Questions

- **How can we determine if a digital preservation action changed significant properties of the digital object?** – Digital objects by nature need an environment to be rendered in. This is for one hardware and also usually additional software, e.g., an operating system, a viewer application. Changing any of the objects needed to render a digital object has a potential influence on the rendering of the object. To build trust that an object can be successfully redeployed and rendered in the future, every digital preservation action has to be evaluated against the original digital object.

  For migration a common (and rather simplistic) method to evaluate the significant properties of a digital object is to compare the stored properties of the object before and after migration. The migration is deemed to be successful if the significant properties are unchanged. For emulation strategies

the object stays unchanged and the environment changes, thus a comparison of stored properties is not possible. Similarly, the rendering software and hardware for static documents can change as well as the digital object (through migration). An authentic rendering of a static document has to be established and verified in this changed view-path. To tackle these challenges, we need to devise a methodology to compare digital objects based on their rendered forms instead of the form stored in a bitstream.

- **How to handle interactive and dynamic objects?** – If a static document is rendered using the same software on the same hardware, the result is usually equal for every rendering. The behavior of dynamic and complex digital objects, however, might depend on external data like user input, data from a web service or even the current time. A successful comparison of two different renderings of a digital object depends on having the object render deterministically, i.e., it has to be the same for every rendering and independent of the environment the object is rendered in. We thus need to identify approaches to establish a deterministic rendering of dynamic objects to make sure that differences in the rendering are a result of differences in the environment.

- **What is the influence of external data on the successful execution of a digital object once re-deployed in a new environment?** – External data not only influences the deterministic rendering of a digital object but can potentially also be necessary for a successful re-deployment of the object. E.g., missing external data sources like web-services might render a digital object useless. Thus, we need to establish means to capture and reapply the data for evaluation, and to emulate or simulate missing external data sources for re-deployment.

- **How can we extract significant properties from the rendered forms of a digital object?** – To extract rendered properties as opposed to extract properties from the bitstream of a digital object we have several choices of where instances of the rendering can be found in the memory of the host system, a virtual system, or even on an output device. Depending on the object one or more of those instances have to be chosen for extraction of the properties. We have to identify the levels suitable for a comparison for a digital object.

- **At what points in the rendering life cycle of a digital object do we need to extract significant properties?** – Not every state a digital object can be in is deemed significant, e.g., internal states might not be relevant for user

interactions. Thus we have to decide for a digital object at what points in the rendering life-cycle we need to compare the significant properties.

- **How does this evaluation need to be integrated into a preservation workflow?** – A preservation workflow has a wider scope than just evaluating how an object reacts to a digital preservation action at the current moment. It includes the phase of planning for the preservation of a digital object, the actual preservation and the redeployment at the time of extracting the object from the digital archive. Evaluation steps have to be repeated in the different phases of a preservation workflow to make sure that all the necessary components of a digital object are stored in the archive, and that the object is correctly rendered in the future environment once it is redeployed.

In this thesis we will address a number of research questions that are based on the above challenges, in particular:

RQ1: **How can we evaluate if a digital preservation action keeps the significant properties of a digital object intact?**

Checking if a digital preservation action can be considered to be successful contains the following questions:

a. Do we need to validate emulation actions differently than migration actions?

b. How can we compare preservation actions on a digital object based on the rendering of the object instead of on stored properties?

c. How can the evaluation of the rendering of a digital object be automated?

d. How can we combine the necessary steps to an evaluation framework and process to follow for repeatable evaluation?

RQ2: **What do we need to know about a digital object and it's environment to evaluate how a new rendering differs from the original rendering?**

To successfully rebuild the view-path of a digital object in a new environment, we need to describe both the digital object and it's dependencies as well as the original environment, raising the following questions:

a. How can we make sure that a digital object behaves deterministic in every rendering allowing us to compare the different renderings?

b. What do we need to document about the digital artifact's behavior and appearance for a successful evaluation?

c. What are the significant states of a digital artifact we have to compare?

d. What data do we need to collect about a rendering environment to allow for repeatable comparison?

e. What environment should be used as a reference for future comparisons?

f. What rendered forms of a digital object exist in a system and which of those are suitable for the comparison of renderings of a digital object?

g. How can we collect verification data to allow for a comparison to a future rendering of the digital object?

RQ3: **How can the rendering of a digital object be made deterministic over different rendering cycles and different environments?**

A complex digital object's behavior usually depends on data sources that determine how the digital object is rendered. To determine that differences in two potentially identical renderings of a digital object are caused by a faulty environment, we have to ensure that the renderings are in fact identical. This leads to the following questions:

a. How does the virtual environment influence the deterministic rendering of a digital object?

b. What external data sources influence the behavior of a digital object?

c. What locally available data influences the behavior of a digital object?

d. How can these data sources be kept identical over different rendering cycles?

e. How can we keep the data that influences the behavior of a digital object consistent over different environments?

f. How can we capture and reapply external data for the validation and verification of a digital object in its new environment?

g. How can we simulate external data sources for the re-deployment of a digital object in a future environment?

RQ4: **How can the view-path of a digital object be recreated in a new environment and a new rendering be compared to the original rendering?**

Once a digital object was successfully described and verification data has been collected, we need to re-create the view-path in a new environment and compare the new rendering to the original rendering. This raises the following questions:

a. How can the view-path of a digital object be recreated in a new rendering environment?

b. How can external data that influences the digital object's rendering be applied to the environment?

c. On what level in the new environment can we find a rendered form of the digital object corresponding to the original rendering?

d. How can we compare the extracted data from the original rendering and the new rendering?

e. What properties other than the rendered form of the digital object can we extract from the rendering environment that give evidence about the rendering process?

RQ5: **How can the evaluation framework be integrated into a preservation workflow?**

The scope of a preservation workflow is usually wider than just the comparison of two renderings of digital objects, thus the following questions arise when integrating evaluation steps into the workflow:

a. Which steps of the evaluation framework have to be performed in the different phases of the workflow?

b. How can different preservation actions be compared in a planning phase using the evaluation framework?

c. What data needs to be collected to validate the completeness of a digital object when the object is being archived?

d. What data has to be stored for evaluation between the preserve and redeploy phases of the workflow?

e. What data needs to be collected during the validation of a digital object and stored in an archive to enable the verification of an object in a future environment?

f. How can the object's proper rendering be verified in a future environment with the data collected before storage?

RQ6: **What design requirements do we have to virtual environments to allow for evaluation of renderings?**

Virtual environments do not yet support the evaluation of renderings of digital objects and digital preservation requirements in general. The following questions have to be answered to create guidelines for developing virtual environments:

a. What are the requirements on the long term stability of virtual environments?

b. How can a virtual environment support the capturing and re-applying of external data?

c. How can we export significant data about a digital object for comparison from a virtual environment?

d. What methods of data exchange between host and guest systems exist and how can it be supported by the virtual environment?

In this thesis we introduce a framework for evaluating digital preservation action results by comparing the rendering before and after applying a digital preservation action. We show the motivation for developing this framework on specific case studies with complex digital objects. We show why it is not enough to compare stored properties of the digital object, not only in emulation where the digital object stays unchanged, but also for migration actions. We further explain how to integrate the evaluation framework into a specific preservation workflow outlining the necessary steps to be taken in the different phases of the workflow. From the actions that have to be performed in the framework, we derive various requirements to virtual environments, so that automated evaluation is enabled.

On a real-world example of an obsolete system we then show how we developed tools to

1. Extract data from obsolete media without use of the original system
2. Enhance an existing emulator with capabilities to render the extracted data
3. Integrate testing capabilities into the emulator to show the validity of the framework we developed

Work presented in this thesis has involved various collaborators and project partners, with my main contributions to the field being as follows:

1. I carried out a number of case studies evaluating the rendering of complex digital objects [Guttenbrunner *et al.*, 2010a] [Guttenbrunner *et al.*, 2010b].
2. I developed a framework for evaluating rendering environments with clear steps, outlining the necessary pre-conditions for an automated evaluation [Guttenbrunner and Rauber, 2012c].
3. In [Guttenbrunner and Rauber, 2012b] I showed that independent of the preservation action taken the rendering of a digital object is what has to be evaluated, it is not sufficient to look at stored object properties.
4. Integrating the evaluation framework into a preservation workflow, I outlined the steps to be taken in the different phases of an existing preservation workflow (under review).

5. From the steps of the evaluation framework I derived requirements and design guidelines for virtual environments for digital preservation to support automated evaluation [Guttenbrunner and Rauber, 2011] [Guttenbrunner and Rauber, 2012c].

6. For a not-well preserved system I led the development of a tool to extract and migrate data from obsolete media using audio analysis [Guttenbrunner *et al.*, 2011].

7. I enhanced an existing emulator to support execution of the programs recovered from the obsolete media using the developed guidelines and outlined the design decisions that have to be taken during the implementation of an emulator for digital preservation purposes [Guttenbrunner and Rauber, 2011].

8. I integrated testing capabilities into the developed emulator to automatically evaluate various different objects for the system [Guttenbrunner and Rauber, 2012a].

1.3 Organization of this Thesis

This section outlines how this thesis is organized. It lists the different chapters and references to publications in which the main contributions to this thesis have been published in peer reviewed journals or conferences.

- In Chapter 2 we will show the view-path of a digital object and discuss the digital preservation strategies to maintain the view-path for the long term. Migration and emulation as strategies for preserving digital objects are presented along with more related work on emulation. Preservation planning for which evaluation of the digital preservation action is an important part will be discussed [Becker *et al.*, 2009]. We will present work on authenticity that will explain why it is important that the rendering of a digital object stays unchanged with regards to the significant properties identified. The significant properties of digital objects as well as the various technologies used in conjunction with emulation and digital preservation will be presented along with projects dealing with preserving complex digital objects.

- Chapter 3 argues why the evaluation of rendering is not just important for evaluating emulation strategies but also for evaluating migration strategies. Comparing the properties stored in a digital object only give a certain indication about how a digital object should be rendered. As published in [Guttenbrunner and Rauber, 2012b] we will show how the view-path of a digital

object changes not only using emulation but also using migration, thus making it necessary to compare the rendering also for migrated objects. This leads to the conclusion, that, from an evaluation and impact perspective, the commonly distinct approaches of migration and emulation are identical.

- In Chapter 4 we will describe a framework for evaluating rendering environments [Guttenbrunner and Rauber, 2012c]. We will show the pre-requisites to comparing renderings of a digital object in different environments. Based on the significant properties of a digital object we will explain where and when to extract data from the rendering for comparison. We then will show how external data can influence the rendering of a digital object, making it necessary to capture and replay that data for evaluation. We will explain how to deal with these external dependencies for the verification and discuss their replacement for usage of the digital object in the re-deployed environment.

- In Chapter 5 we then show how the data captured in Chapter 4 is used to verify the rendering of a digital object in a new environment [Guttenbrunner and Rauber, 2012c]. We present how the evaluation framework is integrated in an existing preservation workflow. We will show which of the steps in the framework have to be repeated in every phase of the framework and what data has to be stored in an archive to be able to validate the object before storage and to verify the objects rendering once it is being re-deployed in a new environment. Work in this chapter has been submitted for review to a journal.

- Chapter 6 shows the requirements of digital preservation to virtual environments and how the evaluation framework should influence the development of rendering environments for digital preservation. Long term stability requirements of virtual environments and the transfer of data between the host system and the guest system are shown, as well as techniques used to implement transfer channels. We then discuss the requirements of automated evaluation on virtual environments [Guttenbrunner and Rauber, 2011] [Guttenbrunner and Rauber, 2012c].

- In Chapter 7 we present the preservation of an obsolete home-computer-system, the Philips C7420 Home-Computer cartridge for the Philips G7400 Videopac system. We first show how data from obsolete media is extracted without use of the original system using digital archeology and audio analysis and how it is migrated to non-obsolete formats [Guttenbrunner *et al.*, 2009] [Guttenbrunner *et al.*, 2011]. Next, we show how to implement an

emulator to render software extracted from the original media. We discuss the design decisions that have to be taken to develop an emulator with digital preservation use in mind [Guttenbrunner and Rauber, 2011]. Finally, the implementation of features to allow automated evaluation will be shown [Guttenbrunner and Rauber, 2012a].

- Chapter 8 shows case studies for the evaluation of rendering using the evaluation framework. We first present the evaluation on the example of the obsolete, simple home computer system shown in Chapter 7. The features for automated evaluation implemented in the emulator are used for the evaluation of a business process example as well as of a video game. We then discuss the evaluation framework applied in the context of the preservation workflow on two more complex examples. The first example is a music classification process as an example for a scientific workflow, and the second example is a digital artwork.

- Finally, in Chapter 9 we present the conclusions of the work conducted over the course of this thesis. Ongoing as well as necessary future work on making virtual environments ready for digital preservation and the evaluation of rendering results will be discussed.

Chapter 2

Related Work

2.1 Introduction

This section presents work related to this thesis. We will start with explaining the concept of the view-path, the objects needed to render a digital object. Then this section continues with related work on digital preservation, the importance of authenticity, and emulation as a digital preservation strategy. It continues with work on preservation planning and the OAIS model before finishing with a preservation workflow used in this thesis.

2.2 View-Path

Different from analog objects like books every digital object needs an environment to render it. Rothenberg argues, that "Digital Informational Entities are Executable Programs" [Rothenberg, 2000a], i.e., that every digital object is a program that has to be interpreted by a process that knows how to perform the commands in the formal language (the format) the program is written in. This can be as simple as interpreting a string of ASCII character codes to make it human readable. Most of these interpreters are software, but on the lowest level the machine-code is interpreted by hardware (i.e., the CPU of a system). This so called view-path is thus the complete stack of objects needed to render the digital object, or as defined in [van Diessen, 2002]

> "a full set of functionality for rendering the information contained in a digital object"

Diessen et. al. identified four basic layers needed to render any digital object: the data format layer defining the structure of the bit stream (the format of the digital object), the application layer that is able to interpret the format, the operating

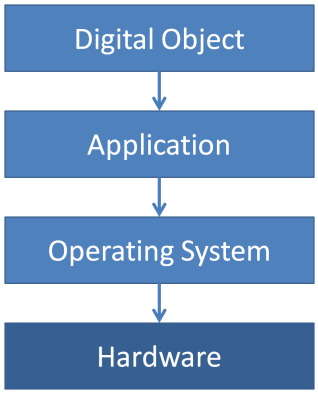


Figure 2.1: Generic view-path for rendering a digital object.

system layer providing interfaces to the hardware and file management, and the hardware layer needed to transform the digital object to a physical presentation on an output interface.

Rendering a digital object is not only rendering a presentation of the object on a screen, but any action that results in a physical manifestation of the digital object on an output interface of the technical infrastructure used to interpret the digital object, e.g., an acoustic representation through speakers, or an actor starting a water pump.

Even in the simple case of rendering a static document at least an application, the operating system, and the hardware needed to run the operating system will be present in the view-path. An example would be a PDF-A 1.1 document rendered using Adobe Acrobat Reader 10.1.3 on a Microsoft Windows 7 operating system updated to a certain date/service pack and a specific set of fonts installed and specific language settings on an x86-compatible workstation PC hardware with a specific graphics card, sound chip etc. A generalized version of this simple view-path can be seen in Figure 2.1.

Depending on the digital object some of the layers in the view-path can be missing or additional layers can be present. A computer game would usually run directly on the operating system, i.e., using operating system libraries and function calls (or on more ancient systems run directly on the hardware without supporting operating system routines), while an application written in Java will additionally have a specific Java Virtual Machine in its view-path, that acts as an abstraction layer between the application and the operating system.

At least one view-path, i.e., a technical infrastructure capable of rendering it, is associated with every digital object. Depending on the digital object more than one view-path can be valid as shown in Figure 2.2. On various hardware-

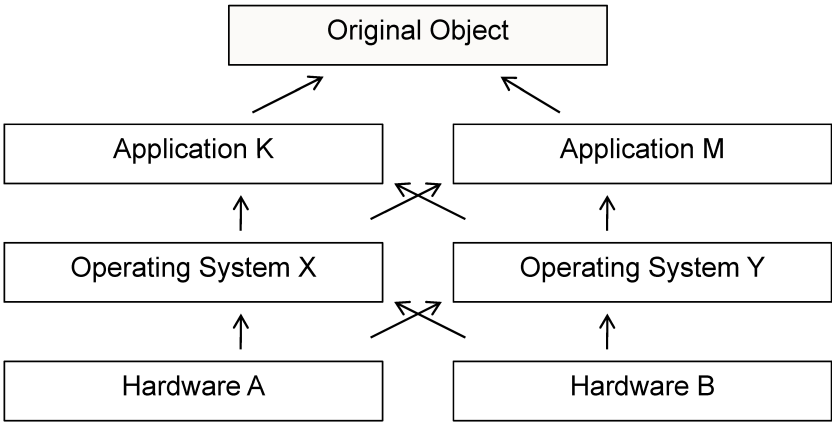


Figure 2.2: Different view-paths for displaying the same digital object.

configurations different operating systems (e.g., WinXP, Linux) can be used to run different applications (e.g., Word, Open Office) to render the same digital object (e.g., a Word-97 Document).

A change in the view-path on any of the layers will potentially also change the rendering of the digital object. E.g., using a different application to render a document on the screen will in most cases result in a different image presented to the user. But not only functional changes may impact the rendering of a digital object. On a hardware-level a slower storage can lead to buffering issues in a video. In a multi-threading environment a different load on the machine can lead to a different rendering of the digital object. Faster hardware might render an object faster and lead to changes in timing of events.

In this thesis we show how changes in the view-path affect the rendering of a digital object, and how we evaluate if the rendering of an object in a different view-path preserves the significant properties of that object.

2.3 Digital Preservation

The pace at which technology gets obsolete and is replaced by new generations of hardware and software poses a threat to the view-path of a digital object shown in Section 2.2. Whenever one of the layers in the view-path gets obsolete we need to perform some kind of action to provide access to the digital object.

Digital Preservation according to the UNESCO Guidelines for the Preservation of the Digital Heritage [Webb, 2005] is the process of preserving data of digital origin, i.e., making sure that a digital object is accessible over a long period of time. Digital preservation is not the process of digitizing non-digital data

for archival, even though once digitized data will have to be preserved digitally to keep it accessible.

2.3.1 Threats to Objects on Different Levels

In these UNESCO guidelines four different layers are listed on which a digital object is being threatened, these are:

Physical Object The physical object is the actual physical manifestation of data on a media. Media get obsolete in that reading devices used to read the media are not supported or stop working. Also, media typically has a life-span of a few years and thus can't be read properly even if reading devices are still available. Thus data has to be transferred from one media to a different media in an archival process. This process is called media refreshment.

Logical Object Data, while stored on a physical media, does not necessarily depend on a specific media but is once separated from the physical media a logical bitstream that can be interpreted by a layer in the view-path. This can for example be a program being executed directly on the physical hardware (e.g., a video game running on an old console system), a program being interpreted by a virtual machine (e.g., a binary program compiled for a JAVA virtual machine), or a data format that is interpreted by an application (e.g., a PDF document). The threat on this level is that either the physical hardware necessary to render a program, or the application used to render the data format is obsolete.

Conceptual Object The conceptual object is the object that actually has a meaning to humans in that it is rendered in a physical form recognizable by a user. This is for example a PDF document rendered on a screen and readable by a human being as opposed to the encoded form of the PDF document in the bitstream (the logical object) and the bitstream of the PDF document stored on a CD (the physical object). The threats require an interpreter or environment to turn the logical object to a conceptual object.

Essential Elements The UNESCO guidelines describe the essential elements of a digital object as a bundle of elements that embody message, purpose, or features of the object describing the context in which it was selected for preservation. This context is also called Metadata. In the example of the PDF document information about the context in which the document was created and information like author, version, preferred view-path to render the document could be recorded.

The task of digital preservation is to counter these threats by appropriate preservation actions, specifically on the logical level, making sure the conceptual object stays unchanged.

2.3.2 Countering the Threats by Preservation Actions

To counter the threats to a digital object on the different levels, we have to perform preservation actions that make sure that the conceptual object stays accessible and (using the essential elements) understandable in its context.

In the UNESCO guidelines various digital preservation strategies are listed, including

- the museum approach - preserving the original hardware and software used to render the object
- converting the digital data to a non-digital format for which we already have a comprehensive knowledge of how to preserve it
- using standard formats that are well defined, open, and likely to be supported by future rendering applications
- making software backwards compatible to interpret older versions of data
- providing viewers for each new technology generation
- migrating a digital object from an obsolete format to a non-obsolete format
- emulating the functionality of a layer in the view-path using a piece of software that performs the functionality of the original layer

Currently the most common strategies used are migration and emulation, as those are deemed to be the most long term stable solutions that can be applied in real-world environments dealing with a large number of diverse objects.

Changing the view-path of a digital object is a digital preservation action. We show how this preservation action potentially effects the rendering of a digital object and how we evaluate and validate the rendering of a digital object against the original rendering after applying the preservation action.

2.4 Migration

Marcum uses in [Marcum, 1996] the term migration for the full continuum of approaches between simply moving the bitstream of a digital object from medium to medium and the full emulation of the rendering environment of a digital object.

Digital Migration as defined by the OAIS [ISO, 2012] is the transfer of digital information within the OAIS with the intention to preserve it. This is distinguished from mere transfer in that the focus is set on preserving the full information intended for preservation, that the migrated representation replaces the original one, and that the full control over all aspects of the migration resides with the OAIS. Migration in the sense of the OAIS is not only the change of the digital object itself, but also of context information needed to understand the digital object.

However, the usual understanding of migration in digital preservation is the technique to change the digital object on the logical layer, i.e., changing the bitstream of the digital object and thus the format. One principal problem of migration is that any change in the bitstream potentially can mean loss of data. Once lost, this data can never be restored in future migration steps.

While migration can happen continuously in an archive to always keep the digital objects in a state that is accessible, it can also be used as migration on request as described in [Mellor *et al.*, 2002], i.e., migrating an object to a non-obsolete format used by the target audience once it is needed. While the format in the archive stays unchanged, the migration tools are changed to support new formats needed for dissemination. This method has also been used by Woods et. al. in [Woods and Brown, 2008] as the primary strategy to allow for browsing and searching in document collections.

Migration is one of the actions changing the view-path of a digital object. In Chapter 3 we show how the view-path of a digital object is changed by migrating the object.

2.5 Emulation / Virtual Environments

Emulation is the concept of replacing a component in a system with a different component that fulfills the same functionality. As compared to simulation the difference is shown by Rothenberg et. al. in [Rothenberg, 2000b] in a simple example - a flight simulator does not actually let you fly a plane but only simulates the process. A terminal emulation on the other hand lets you use a command line interface for input and output for the same purpose as you would use a real terminal to a computer system.

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

Rothenberg et.al. argue in [Rothenberg, 2000b] that all digital documents are software dependent and as such essentially programs intended to be interpreted in a software environment or by hardware. Saving the entire software environment used to interpret a digital object we can thus save the view-path of a digital object. However, to physically preserve the hardware used to execute the software environment is not viable, using an emulator of the hardware is a virtual way of saving the hardware, allowing an execution of the software environment and thus to render the digital object.

2.5.1 Levels of Emulation

Granger et.al. explain in [Granger, 2000] that emulation can be done on three different levels - application, operating system, and hardware platform.

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

Environments for digital objects can thus be emulated on different levels as shown in Figure 2.3. These layers can be defined as:

Application As described in Figure 2.1, an application is usually used to render a digital object (if the digital object to be preserved is not itself an application (e.g., computer games, digital art, self-running documents). By replacing the application used to render a digital object the functionality of the original application is emulated. A simple example is to replace Adobe Acrobat Reader[1] for rendering PDF documents by a different PDF-reader like Nitro PDF Reader[2]. The Nitro PDF Reader would serve as an emulator for the functionality of the Adobe Acrobat PDF Reader in the view-path of the PDF document.

[1]Adobe Acrobat PDF Reader - `http://www.adobe.com/products/reader.html`
[2]Nitro PDF Reader - `http://www.nitroreader.com/`

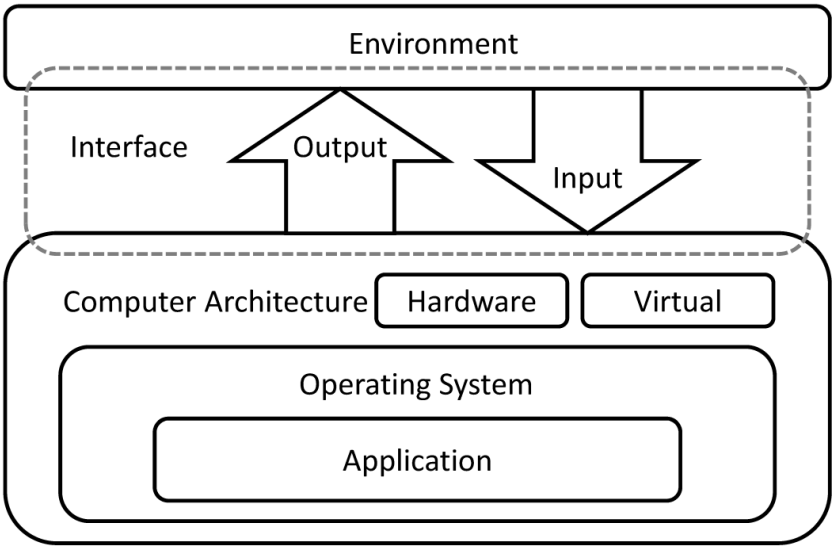


Figure 2.3: Layers of emulated environments for digital objects.

Operating System On a modern computer system usually an operating system between the application and the hardware (virtual or real) provides functionality used by the application (e.g., access to display devices, input devices). The operating system, in turn, heavily depends on the computer architecture it is running on and the application heavily depends on the operating system. By providing a middle-ware that acts like the original operating system any operating system calls in the application can be caught by the middle-ware and translated for the new environment, thus emulating the functionality of the original operating system. An example for this would be Wine[3] that allows the execution of programs running on Microsoft Windows operating systems to be executed in a Linux environment. In the view-path of a Microsoft Word document rendered by Microsoft Word 95 this would mean replacing the operating system layer Windows 95 by a Linux distribution operating system and using an additional Wine translation layer to allow for operating system functions originally performed by Microsoft Windows 95 be executed by the Linux operating system.

Computer Architecture As the functionality of hardware is usually better documented than software the most common level of emulation is recreating the computer architecture. Computer architecture can either be real hardware (an actual physically existing system) or a system only existing virtually, a virtual machine like e.g., the Java VM. While virtual machines are soft-

[3]Wine - `http://www.winehq.org/`

ware that is usually running on top of the operating system and ported into a different environment, physical hardware can be emulated using one of the following approaches:

(a) Full Hardware Emulation The most common use of emulation is the recreation of hardware components in software on a new host-system. The „Emulator" in this case is, as defined in [Slats, 2003], a program that runs on one computer virtually recreating a different computer's hardware. It provides a layer between the host system and the originally used software, replacing the functionality of the original physical hardware used to execute this software. The whole system is rebuilt in software and the original digital artifact is executed using the original software in this simulated hardware environment. Examples are emulators for proprietary hardware such as video game console systems on a PC or emulators for PC hardware on virtual machines (e.g., Dioscuri[4]).

(b) Virtualization An approach to create a virtual environment by using either some or all of the hardware of the host system directly is called virtualization. Code is directly executed on the physical CPU instead of being emulated. In reality the approach is usually a mix between hardware emulation and virtualization as all virtualization solutions emulate certain low-level instructions instead of running them directly on the real CPU. Using the virtualization approach, software which is potentially able to run on the host system's hardware is run in a virtual machine hosted by the current operating system environment. Examples for virtualization software are VirtualBox[5] and QEMU[6]. One use of virtualization is to have a host system running one operating system and create a virtualized environment running a different operating system. Another one is to run different virtualized computer systems on the same physical hardware in parallel. The long term use for digital preservation is obviously limited, as virtualization only works if the physical system uses a CPU that is compatible to the target system. As soon as the hardware becomes obsolete, computer architecture emulation has to be done using hardware emulation. If the

Interface Level Emulating a computer system on the interface level requires to recreate the original means of input/output of the system to recreate the

[4]Dioscuri - `http://dioscuri.sourceforge.net/`
[5]VirtualBox - `https://www.virtualbox.org/`
[6]QEMU - `http://wiki.qemu.org/Main_Page`

original communication experience with the system. For two-way communication we have to consider:

- Output-Devices: Using a mobile hand-held with a 3-inch screen as an output device as the original system has a different look-aspect compared to the emulation of the same system on a PC-screen with 18 inches. An example is an emulator of a Nintendo DS with two screens on a PC with a normal LCD screen running a Microsoft Windows operating system. Similarly, actual actuator output in a control system is different to compare to a simulated actuator output that may be as a visual, acoustic or voltage level setting.
- Input-Devices: Using paddle-controllers to play the game Pong is a different feel-aspect than using a keyboard or even a mouse to control the same game in an emulated environment. This applies not only to human-computer interaction, e.g. in a control system, but also to differences in sensitivity, drift or timing behavior for all kind of sensor input.
- Machine-to-Machine Communication: In distributed computing the original interfaces to a machine are recreated or emulated. This can for example be web-services that provide the expected data on a communication interface.

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

When a system component is to be replaced by a different (emulated) component, evaluations if the rendering is still unchanged are standard practice.

2.5.2 Emulation Technologies Used in Digital Preservation

In this section we will present some of the concepts and technologies used and recommended for emulators in digital preservation. Included are technical recommendations as well as concepts for digital preservation emulators.

Keeping Emulators Useable

Emulators are, again, a piece of software that gets obsolete over time. To make sure, that emulators developed today can be run on future platforms, different technologies exist:

Stacked Emulation Every emulator is written for a specific host platform. Once the host platform gets obsolete, an emulator for the host platform is created, executing the emulators that used to exist on the host platform. With every generation a layer in the emulation stack is added, leading to a complex view-path. Possible errors in the emulation processes are aggregated.

Migrated Emulation Emulators existing on a host system are migrated to a new host system every time a platform gets obsolete. With emulators existing for a wide array of systems, the effort of migrating every emulator to a new system are big.

Emulation Virtual Machine In [Rothenberg, 1998] Rothenberg proposes the Emulation Virtual Machine (EVM) as a solution to stacked and migrated emulation. A wide array of emulators are developed for one virtual machine, that is different from other virtual machines in that it is long term stable, i.e., not changed over a long period of time. Once a platform gets obsolete, not all the emulators have to be ported to the new platform, only the EVM. Development of an emulation virtual machine started in the European project KEEP (Keeping Emulation Environments Portable) as the KEEP Virtual Machine (KVM) [Bergmeyer, 2011].

Modular Emulation

Many components in a system are either off the shelf components or used in different systems of the same time period (e.g., the MOS Technology 6502 microprocessor[7] used in different home-computer systems from the 1980's). Reusing components that are proven to work is a common strategy in software development. The same is true for developing an emulator. Reusing emulation code for specific components of a system for the emulation of more than just one system is thus self-evident.

Modular emulation as a concept for emulators is discussed as a conceptual model by Van der Hoeven et. al. in [van der Hoeven and van Wijngaarden, 2005]. Instead of developing all the components of an emulator for every single machine configuration, components that are already proven to work in an emulator are used to create new system configurations by combing the different components to a new system. Each component is emulated by an encapsulated piece of code emulating the functional behavior of one hardware component. By interconnecting the different components, an emulation process for the full system is created.

[7]MOS Technology 6502 – `http://en.wikipedia.org/wiki/MOS_Technology_6502`

The modular emulator is also suggested to run on a virtual machine to make it hardware independent as shown in Section 2.5.2.

The emulator Dioscuri[8], an emulator created specifically for digital preservation purposes and based on the modular emulation approach is described in [van der Hoeven *et al.*, 2007]. It is written in Java and runs on the Java Virtual Machine.

Other emulators following the modular concept are the Multiple Arcade Machine Emulator (MAME)[9] that uses configuration files to emulate arcade machine systems, as well as its sister project MESS (Multi Emulator Super System)[10], that uses the same code base to emulate a wide array of video game systems, computer, and calculators.

Universal Virtual Computer - UVC

An approach for a Universal Virtual Machine is the *Universal Virtual Computer (UVC)* as developed by IBM ([van der Hoeven *et al.*, 2005]). The concept is to design a virtual machine that while being simple enough to be easily implemented on a future system is still sufficient for the rendering of digital data that has been preserved. On the time of archiving a program is written for the UVC that is capable of rendering the digital data in the UVC. This program is stored in the archive along with the digital data. Once the data is extracted from the archive, an implementation of the UVC is done on a then current system. The program written on archival time is executed on the UVC implementation, rendering the digital data. The advantage of this approach is that the data stays unchanged, and a rendering engine is created at a time when the original can still be rendered in the original environment, thus making sure that the rendering is true to the original. Also, only one virtual machine has to be implemented on each system generation allowing the execution of all UVC programs. The disadvantage is that for every format stored in the digital archive a rendering application has to be created. A proof of concept for this approach has been done on the archiving of JPG-images.

KEEP Emulation Framework

The Emulation Framework (EF) [Lohman *et al.*, 2011] developed in the European project KEEP provides an easy way for users to invoke emulation environments for users based on the object's needs. It automates the identification of an object and selects the needed emulation environment, operating system, application, and configuration (the complete view-path necessary to render the digital object). In

[8] Dioscuri – `http://dioscuri.sourceforge.net/`
[9] Multiple Arcade Machine Emulator (MAME) – `http://mamedev.org/`
[10] Multi Emulator Super System (MESS) – `http://www.mess.org/`

the EF an Emulator Archive holds the available emulators. Those are connected to a set of disk images containing the view-path. Through an abstract model the hardware configuration necessary for the digital object is configured, and the digital object is provided in a disk image, and thus injected into the emulation environment. The EF aims to provide an environment to access digital objects in emulation environments to be as easily accessible as clicking on the digital object.

Remote Emulation

While the aforementioned Emulation Framework invokes emulators locally on the user's system, remote access to emulation tries to tackle the complexity of emulation environments and their setup by users a different way. By having the emulators run on a different system and providing access through a web interface, the user does not need to install and maintain different emulation environments on his system. The remote access to emulation as described in [Rechert *et al.*, 2010] and [von Suchodoletz *et al.*, 2011] provides an interface that allows users to select an environment, upload a digital object and interact with the object through a web interface. Besides using the emulation environment to render digital objects, migration through emulation is also made possible, as the interaction with the environment can be scripted on the server side, invoking the necessary applications and converting the file to a non-obsolete format. The resulting file is then offered for download to the user.

2.5.3 Emulation in the Context of this Thesis

In this thesis we show how emulation is comparable as a digital preservation action to migration from an evaluation view-point in Chapter 3. We explain how the rendering of a digital object is used for comparison of the significant properties in different environments, and what functionality rendering environments, specifically emulators, have to have implemented to support evaluation of renderings.

2.6 OAIS Reference Model

One of the first ISO standards in digital preservation and the most common framework used for archives is the reference model for an Open Archival Information System (OAIS) [ISO, 2012]. The description of the OAIS states that "An OAIS is an Archive, consisting of an organization, which may be part of a larger organization, of people and systems that has accepted the responsibility to preserve information and make it available for a Designated Community.".

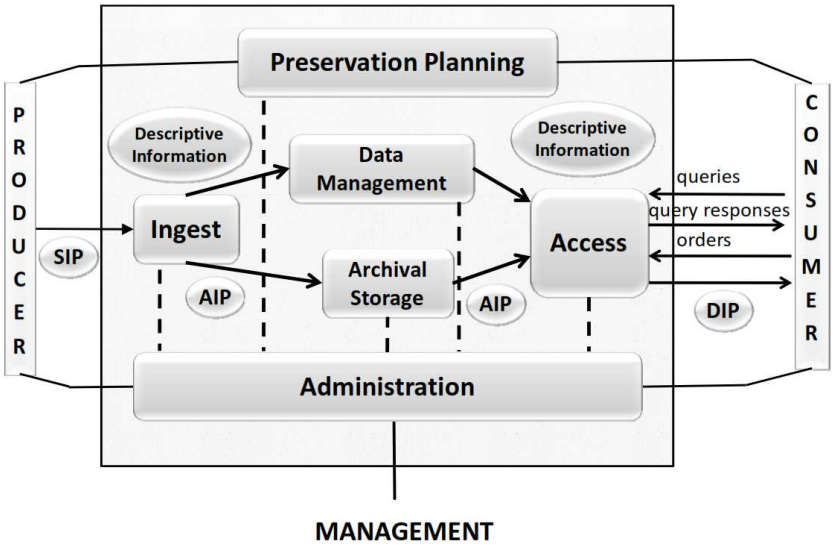


Figure 2.4: Functional entities of an OAIS archive [ISO, 2012].

The OAIS is intended to establish minimum requirements for an OAIS Archive and provide a set of archival concepts, thus establishing a common framework from which to view archival challenges related to digital information. The OAIS defines both the core components of an archive for digital preservation on a high level as well as defines a standard terminology.

It also defines the different packages containing a digital object that exist in an archive during the ingest – archive – access life-cycle of an object:

Submission Information Package (SIP) The digital object along with descriptive meta-data and all accompanying elements as specified in an interface to the archive and provided by the producer.

Archival Information Package (AIP) The digital object along with descriptive and additional information as permanently stored in the archive.

Dissemination Information Package (DIP) The form in which the digital object is provided on access to the consumer.

The different functional entities of the OAIS that manage the different forms of the digital object in the archive can be seen in Figure 2.4. The functions of the main entities are as follows:

Ingest provides services and functionalities to accept a submission from the Producer and convert the submission package to an archival package as stored in the archive. Additionally quality control of the submitted package is performed.

Archival Storage takes care of managing the archival packages in the archive, including media refreshment, error checking and providing the archival package to the Access entity.

Data Management provides services and functions for accessing the descriptive information of digital objects stored in the archive.

Administration are the services for the overall operation of the archive.

Preservation Planning is responsible for making sure that the digital information remains accessible to the designated community by monitoring the environment and providing recommendations and preservation plans.

Access provides services for accessing both meta data about the the archived digital object as well as a representation of the object itself in a form ready for dissemination to the user.

Making sure that information is preserved for a designated community is the main goal of an OAIS archive. It does so by providing functionalities that evaluate digital preservation actions to determine if the information remains accessible for the designated community, i.e., usable as described in Section 1.1.

In this thesis we will explain what data needs to be encapsulated with a digital object for storage in an archive as an AIP to allow for later verification of the object's re-deployment in a new environment.

2.7 Preservation Planning

For digital preservation purposes it is necessary to compare the effects of different preservation actions on the significant properties of digital objects. In [Becker *et al.*, 2009] we show a preservation planning workflow that allows for repeatable evaluation of preservation alternatives. This workflow is also implemented in the preservation planning tool *Plato* [Becker *et al.*, 2008a].

The workflow as shown in Figure 2.5 consists of four different phases:

Define Requirements In a first step the basis for the preservation are defined. Along with the intended audience (designated community) of the preserved objects meaningful sample objects are chosen. The requirements for the preserved digital objects (the significant properties) are defined in a tree-structure, breaking down the different requirements into categories and sub-categories down to the leaves of the tree.

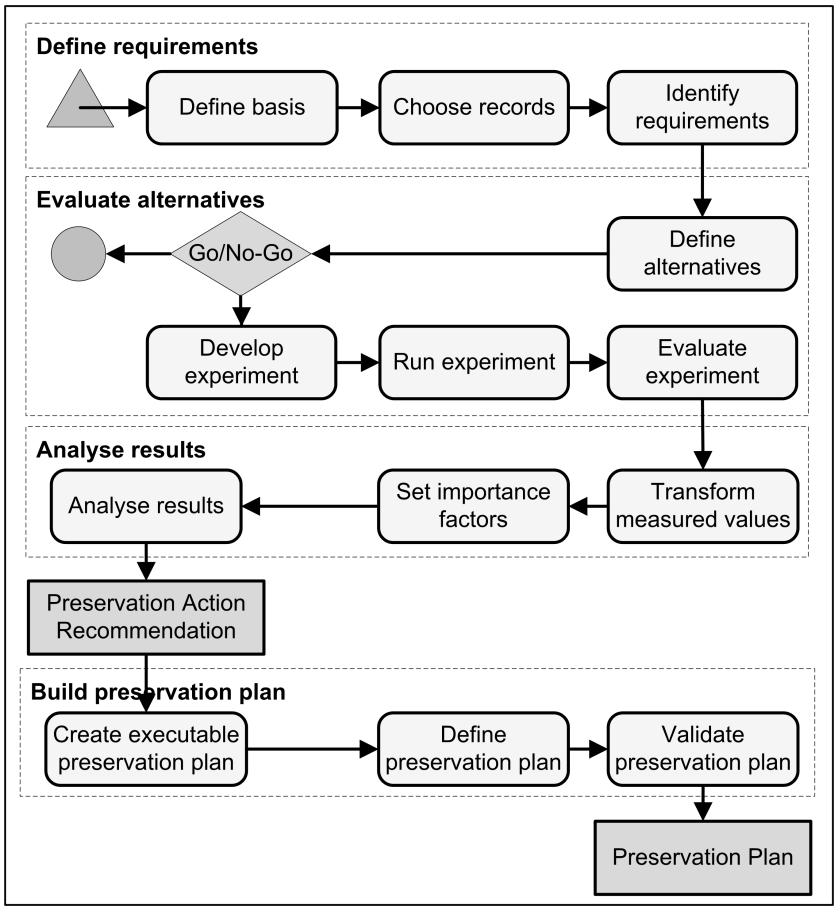


Figure 2.5: Preservation planning workflow [Becker *et al.*, 2009].

Evaluate Alternatives Next, a set of different alternatives for preservation actions are selected. Based on the defined basis a Go/No-Go decision for each of the alternatives is taken (e.g., No-Go based on not meeting basic requirements defined in the basis). The different preservation action alternatives are then run on the selected sample objects and evaluated.

Analyze Results The results of the preservation actions are then transformed to a uniform scale by extracting the significant properties and deciding how well they were preserved by the alternatives. By setting importance factors for the different branches in the tree, a utility analyses is performed resulting in a ranking of the different alternatives. The result of the analysis is a recommendation for one of the preservation action alternatives.

Build Preservation Plan Based on the recommended alternative, an executable preservation plan is created, defined and validated, resulting in a preservation plan.

Figure 2.6 shows how the preservation planning is integrated into the OAIS. By using the preservation planning approach shown in this section we can evaluate how well significant properties are preserved with different preservation actions and how well other constraints set by the preservation planner are met. In this thesis we will show a framework and process that help in evaluating the effects of different preservation actions (migration and emulation) on the rendering of a digital object.

2.8 Evaluation of Digital Preservation Actions

When performing a digital preservation action (or planning for digital preservation actions on digital objects), it is necessary to evaluate, if the digital object is properly represented in its new form, be it in an emulated environment or in a migrated format. When migrating a digital object to a different format, characteristics of the object are usually extracted from both file representations and compared. But it should be noted that digital objects represent information objects, and are thus always used in a rendered form (which may be as complex as an entire business process, a system control application, an interactive complex media object or video game, a rendered office document, but also a simple XML file encoded in Unicode and rendered as readable character set or interpreted as machine-readable content). Thus, evaluating any kind of digital preservation action needs to be based on the connecting point between interface and environment. In this sense, there is no formal, conceptual difference between emulation

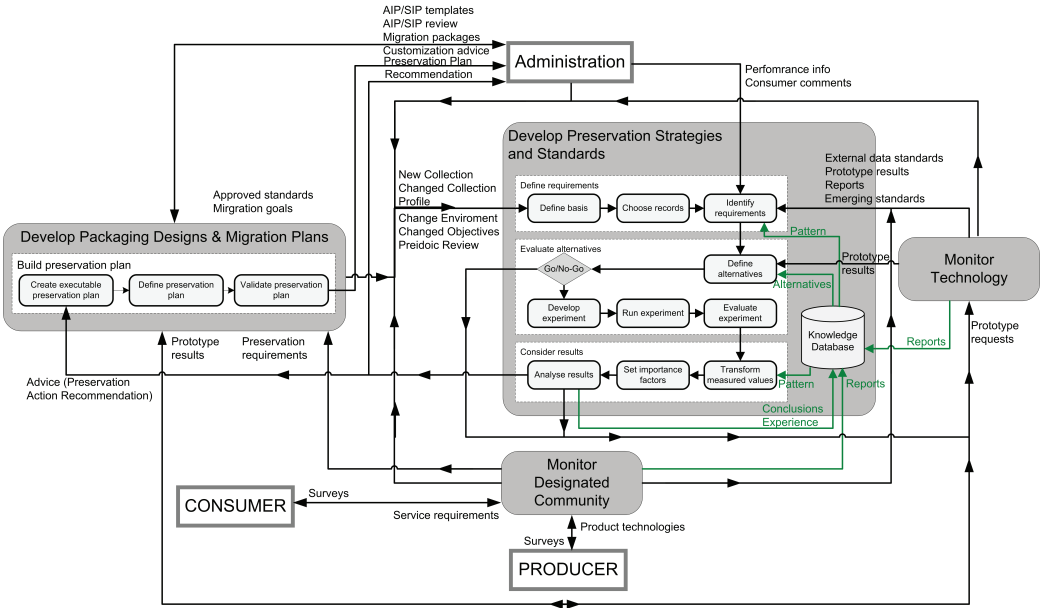


Figure 2.6: Preservation planning integrated into the OAIS. [Becker *et al.*, 2009].

and migration from an evaluation perspective, as even migrated objects need to be evaluated within the new rendering environment as shown in Chapter 3.

2.8.1 Significant Properties

A common way to compare a digital object before and after applying a digital preservation action is to compare if the significant properties of the digital object are still intact. Properties of the digital object that are significant to a designated community and a specific use case are usually defined during the preservation planning process.

To evaluate a digital preservation alternative it is thus necessary to know the properties of an object that are significant and that have to be preserved. These properties can be technical as well as social properties. Depending on the type of object and the designated use the weighting of the importance of meeting specific requirements can be different. A common way of structuring the significant properties of an object are the categories content, context, structure, appearance, and behavior or functionality [Rothenberg and Bikson, 1999].

Evaluating the preservation of significant properties after applying a digital preservation action using standardized, repeatable and objective methods was developed in a series of projects and tools. In [Rauch and Rauber, 2004] Rauch et. al. develop a preservation solution evaluation metric based on Utility Analysis. This approach is combined with the Dutch Digital Preservation Testbed ([Slats

and Verdegem, 2004], [Hofman *et al.*, November 2004]) to the DELOS Testbed for choosing a digital preservation strategy in [Strodl *et al.*, 2006]. In [Strodl *et al.*, 2007] Strodl et. al. present the PLANETS[11] Preservation Planning approach, which is based on the DELOS Testbed. The Planets Testbed shown in [Aitken *et al.*, 2010] is derived from the same DELOS Testbed and uses a similar methodology to perform mass evaluations of digital objects as the Plato preservation planning tool shown in Section 2.7. In all these approaches the significant properties of a digital object are extracted before and after applying the preservation action. A comparison of the extracted properties is performed to evaluate how well a digital preservation action preserved the significant properties. A summary of tools to extract these properties for different file formats is shown in Section 2.8.2.

The significant properties of static documents usually differ from those of dynamic and interactive content. While the appearance of the first makes it often possible to migrate the contents to other formats, the task is more complex for interactive content. Potential loss has to be investigated very closely, as for example loss of interaction can render a digital art object completely useless. Visual and audible properties as well as interaction with the object have to be preserved. Even with the same significant properties for different types of complex content the weighting of importance of these properties for preservation can be different depending on the type and the designated user community.

Significant properties of dynamic digital objects include visual and audible properties. All kinds of interactive input possibilities have to be considered. In case of application software and dynamic documents these are e.g., form fields, icons, menus and mouse and keyboard for input. For video games and digital art this can be menus, icons on the user interface, the response and support of hardware like gaming hardware, video cameras, sensors, motion detectors and mouse and keyboard again. Functionality is an important part of software preservation. In case of processes the input data has to be processed and the output data has to be rendered correctly. For video games and digital art the playing experience, response to input and audible/visual characteristics are important.

Comparing renderings of the same complex digital object in different environments is usually done manually by a human observer. A case study to compare different approaches to bpreserve video games, with one of the approaches being emulation, was reported in [Guttenbrunner *et al.*, 2010a] on a human-observable and thus to some extent subjective level.

Availability of source code is one of the significant properties of software that allow us to migrate the software for preservation purposes [Matthews *et al.*, 2008]. For interpreted program languages like BASIC (compared to program languages where source code is compiled to executable software) the source code is equal to

[11] http://www.planets-project.eu

the executable software given the availability of a suitable interpreter.

Becker et. al. present in [Becker *et al.*, 2007] case studies on sample objects of interactive multimedia art from the collection of the Ars Electronica[12]. Interactivity is designed as one of the significant properties of the art works, thus requiring a comparison of successful preservation on a rendering level. In [Guttenbrunner *et al.*, 2010a] we presented a case study on preserving console video games to evaluate existing emulators for their suitability as digital preservation alternatives. This case studies also identifies significant properties of complex content, including the behavior while being rendered and the interactive aspects of the games. It also compares different alternatives for preserving interactive content such as migration and emulation strategies.

The INSPECT project [Grace *et al.*, 2009] used the Function-Behavior-Structure (FBS) framework designed to help create and re-engineering systems. It considers the purpose of the object in the context of how it is used by the stakeholders. Focusing on the properties that are essential (significant) to the stakeholders it allows thus to concentrate on alternatives that preserve these properties best, similar to the preservation planning workflow shown earlier and followed in this thesis.

2.8.2 Identification, Validation, and Characterization of Digital Objects

Part of preservation planning is to automatically characterize migrated objects. Various tools to identify file formats and extract properties exist.

The standard UNIX tool *file* uses a local database to identify files. It is primarily used in production environments to identify files, not necessarily for digital preservation purposes.

An approach for digital preservation purposes is taken by PRONOM [Pettitt, 2003], maintained by the National Archives of the UK. PRONOM is a database of technical information about file formats, but also a platform that encompasses tools and services to support digital preservation functions. One of the tools using PRONOM as its database is Droid [Brown, 2008]. Droid is able to batch-identify files using internal signatures generated from information recorded in the PRONOM technical registry.

The JSTOR/Harvard Object Validation Environment (JHOVE) offers three different functions for analyzing files. Besides the identification of an object's format, it is also able to validate if the object conforms to the format's technical norms. As a third function it can also extract the technical properties of the object that are examined during validation either in plain text or in an XML format [Donnelly, 2006].

[12] http://www.aec.at

Besides the aforementioned registries, other sources exist, such as the Digital Formats Website[13], that holds valuable sources of information for digital preservation. Also the Global Digital Format Registry (GDFR)[14] and the Unified Digital Format Registry (UDFR)[15] can be used for identifying formats and characterizing files.

A wrapper that uses some of the aforementioned tools and more is the File Information Tool Set (FITS)[16] developed by the Harvard University Library. It identifies, validates and extracts technical meta data for a variety of file formats using the various tools and provides the output in a standardized format.

To aggregate and analyze characteristics extracted by FITS, the tool C3PO[17] was developed [Petrov and Becker, 2012]. It creates a profile of the characteristics of the content of a whole collection and allows for a selection of datasets for the planning purposes.

In [Huber-Mörk *et al.*, 2012] a case study is shown that allows quality assurance in scanned images. Image comparison by creating a finger print of each page allows the authors to detect duplicate images. A comparison of sound waves for the sake of quality assurance is described in [Jurik and Nielsen, 2012]. Audio files migrated from WAV to MP3 are compared to assure that the content is unchanged.

2.8.3 Characterization Languages

The significant properties of a digital object that are extracted either from its bitstream or a rendered form have to be stored in a characterization language, to allow for automated processing and comparison of different versions or renderings of an object.

A chracterization language designed for automatically comparing migration results is the eXtensible Characterisation Languages (XCL) [Becker *et al.*, 2008b]. The original and migrated objects are hierarchically decomposed and represented in XML. The XCL consists of three components: an extensible characterization definition language (XCDL) that is an abstract way to express the information contained in a digital object, a component that extracts the data in XCDL as defined in an extensible characterization extraction language (XCEL), and a comparator that is able to compare different XCDL extractions for equality. By comparing the XCDL extractions we can measure the effects of migration on a digital object on properties extracted from the object.

[13]Digital Formats Website – `http://www.digitalpreservation.gov/formats/index.shtml`
[14]Global Digital Format Registry – `http://www.gdfr.info/`
[15]Unified Digital Format Registry – `http://www.udfr.org/`
[16]FITS – `http://code.google.com/p/fits/`
[17]C3PO – `http://ifs.tuwien.ac.at/imp/c3po`

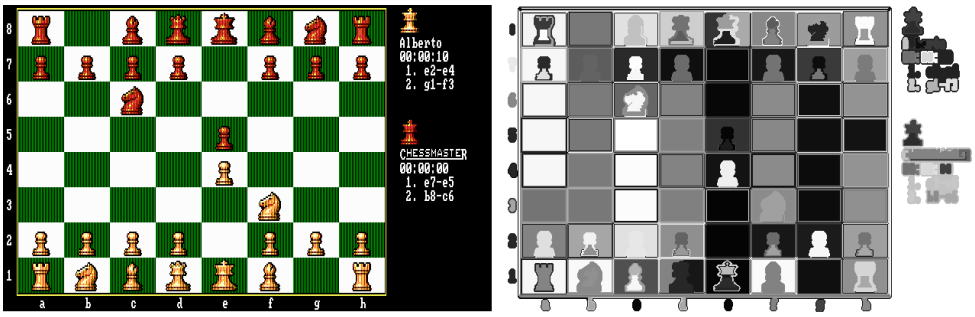


Figure 2.7: Screenshot of "Chessmaster 2100" running under DOS on the left and the segmented screenshot showing significant areas on the right.

For emulation environments however, the digital object stays unchanged, so properties extracted from the bitstream will be unchanged as well. Thus the comparison has to be done on the rendered object.

Thaller suggests in [Thaller, 2008] to separate the information contained within a file from the rendering of the information. Information stored in the file can, for example, be the coordinates of text which leads to the rendering displaying the text on a specific point on the screen. This is described as the *look & feel* aspect of an object.

As shown in [Guttenbrunner *et al.*, 2010b] XCL was extended to be able to describe significant points in the rendering of a digital object. The central analysis task of the method is to identify specific regions in the rendered digital object, which can - in a final step - be compared with the rendering of the same object, using another rendering environment. Those specific regions reflect characteristic layout properties: regions with a high frequency of pixels that could refer to a significant area. To identify and isolate such regions of interest in the prepared, cut to size and binarized image, the image is segmented by the Efficient Graph-Based Image Segmentation Algorithm as presented in [Felzenszwalb and Huttenlocher, 2004]. To facilitate comparison between two images, the upper leftmost pixel and the bottom rightmost pixel of a layout-region in relation to the pixel dimensions of the image by dividing both pixel values by the width, respectively the height, of the processed image are calculated. These relative values are embedded into XCDL files, which are connected to the screenshots, to enable a comparison of the objects through the aforementioned XCL Comparator. An application that accomplishes the described tasks was created as the XCL Layout Processor.

In [Guttenbrunner *et al.*, 2010b] we performed case studies on various interactive objects to show the validity of the approach. Figures 2.7 and 2.8 show two different images segmented to significant regions. Figure 2.9 shows a code snippet that describes the coordinates of a significant region in an image.

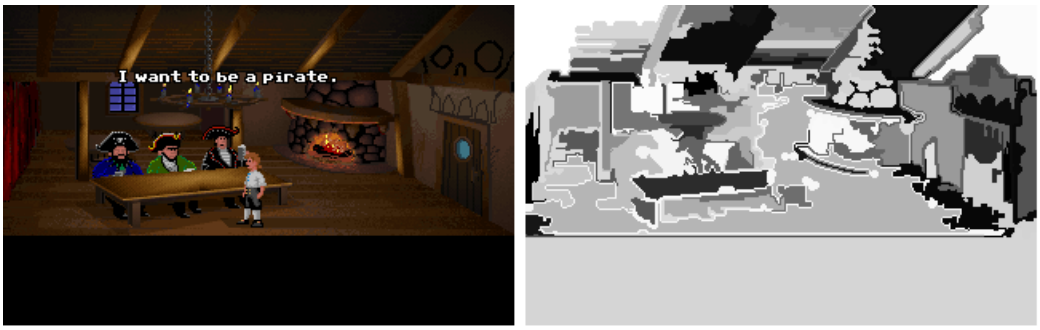


Figure 2.8: Screenshot of original DOS-Version of "The Secret of Monkey Island" (left). Significant areas in the same screenshot as a result of binarization and segmentation are shown on the right.

```
<property id="p15381" source="raw" cat="descr">
   <name id="id9998">significantCoordinates</name>
   <valueSet id="i_i1_i2xx_s1_1">
      <labValue>
         <val>0.118727 0.113586 0.232558 0.335189</val>
         <type>rational</type>
      </labValue>
   </valueSet>
</property>
```

Figure 2.9: Code snippet of XCDL enhancement for significant coordinates of identified areas.

2.9 TIMBUS Preservation Workflow

An object goes through different moments in its digital preservation life cycle. While being deployed in its original environment, planning for the preservation of the object is performed. The digital object is then stored in a long term archive. At some point in the future the digital object is taken out of the archive and redeployed in a future environment.

These significant points in time were developed into a workflow for the preservation of business processes in the TIMBUS project [Strodl *et al.*, 2012]. While this workflow as shown in Figure 2.10 contains stages specific for business processes, the actions that have to be taken are similar for all digital objects and relevant for any evaluation of renderings of a digital object.

Plan In the plan phase the context of a digital object has to be captured along with all legal implications of rendering the object at a later point in time, the necessary documentation and all other relevant data. The risk of not having the object available has to be assessed and managed. The captured context contains amongst other meta-data the view-path, as it is essential to know the requirements of rendering the digital object in a new environment. To accomplish the capturing the digital object has to be analyzed along with

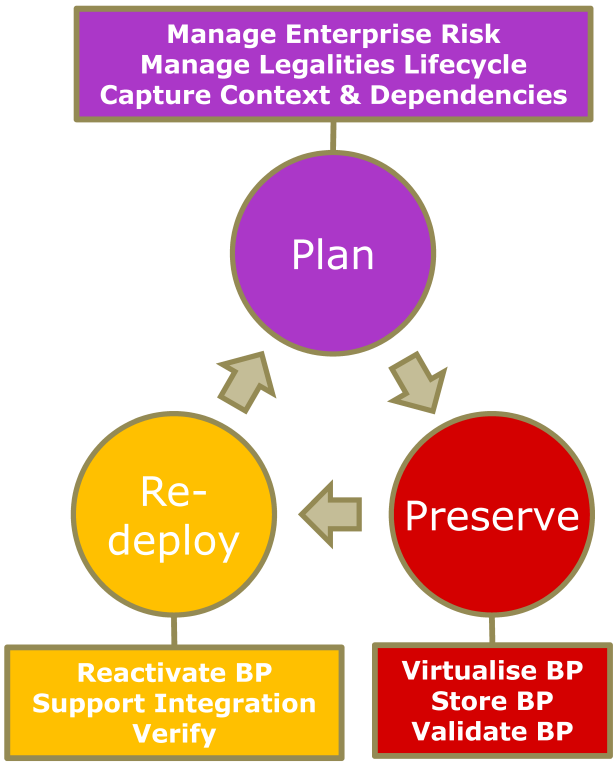


Figure 2.10: Process for Digital Preservation of Business Processes (BP) in TIM-BUS.

all its external dependencies influencing the rendering of the digital object. Knowing these dependencies is necessary to create a deterministic rendering of the digital object, i.e., to make sure that the digital object can be rendered identical under the same conditions and allowing us to do a comparison between the different renderings. To then assess the quality of a preservation action, the preservation action is carried out on the object and the result is compared to the original rendering, and a decision for one of the preservation candidates is taken. This first step of evaluation of the preservation action is similar to the actual validation and verification in the next two phases described below.

Preserve In the actual preserve phase, the digital object is first virtualized. In this step an additional layer is introduced in the view-path (e.g., a virtual machine or an emulator), allowing us to capture any communication between the object's environment and the outside world. This is for one incoming data that influences the rendering, e.g., data from a web service, user input, data from sensors or through a network protocol. But it is also data produced by the rendering in any form (e.g., on the screen as information for a user, data sent to actors or the network). We can thus validate the digital object at the time of preservation to make sure that all the necessary dependencies have been captured and will be stored along with the digital object. We also capture the data used for the validation and the output produced by the rendering process to compare this data in a later verification phase once the object is re-deployed. Data being captured includes the context as well as log files, any test-data and properties of the rendering process.

Re-deploy At a later point in time the digital object is re-deployed in a new environment, e.g., an emulator existing at the time of re-deployment. The digital object has to be integrated in a system where all the external dependencies necessary to render the digital object are provided in some form. An example for this would be a web service that existed when the object was originally in use, but might not exist anymore once it is re-deployed. Once the object has been re-deployed it is necessary to verify that the functionality is still intact. Thus we provide the data that has been captured during the verification phase to the new environment and capture the output of the digital object. If this is done for various use cases, and the data captured matches, we have strong evidence that the rendering of the digital object is unchanged compared to the original rendering at the time of preservation. Other data that will be provided to the new environment to ensure a deterministic rendering includes user input. To verify that the results of the rendering process are unchanged to the original rendering, log files and

properties captured during the rendering in the new environment are compared to the results stored in the preserve phase.

In this thesis we show how the framework for evaluating the rendering of digital objects can be used in the TIMBUS preservation workflow to verify and validate rendering results of preserved digital objects.

2.10 Projects on Preserving Complex Objects, Multimedia and Interactive Content

A number of projects that have been dedicated to preserving complex digital objects like video games, art, or processes have been started in the last years.

The first European research project dedicated specifically to emulation was the Keeping Emulation Environments Portable (KEEP) [Bergmeyer, 2011] project. Besides the aforementioned KEEP Virtual Machine and the KEEP Emulation Framework, the project also did work on the KEEP Transfer Tool Framework to transfer complex digital objects from their original media, as well as on studies researching the legal situation of emulation in Europe. Some of the legal issues raised by KEEP also apply to the development of the emulator shown in this thesis.

The Preservation of Complex Objects Symposia (POCOS)[18] project delivered different symposia in the UK on issues of complex digital objects connected to Visualizations and Simulations (e.g., visualizing 3D models of cultural heritage, virtual museums), Software Art, and Gaming Environments and Virtual Worlds. Besides the symposia, POCOS also published three books on the three different topics covered by the symposia.

The Preserving Virtual Worlds project [McDonough *et al.*, 2010] was a project on the preservation of video games and virtual worlds funded by the Library of Congress. The goals of the project were to develop basic standards for virtual world metadata and content representation, as well as investigating preservation issues by carrying out various case studies on early video games and interactive multi-player game environments. A second installment of the project, Preserving Virtual Worlds 2 (PVW2), is currently underway, focusing on significant properties for a variety of games and providing a set of best practices for preserving materials through virtualization technologies and migration.

An approach to preserve complex multimedia art was undertaken by the Guggenheim museum with the Variable Media Initiative. The outcome was the *Variable Media Questionnaire*, a questionnaire for artists and collectors of digital art

[18] POCOS – `http://www.pocos.org/`

which included descriptive elements needed for recreating the artwork. The research concluded in the *Variable Media Network*[19]. The variable media paradigm includes the artists on the decision on a preservation strategy, with the available options being storage, emulation, migration and reinterpretation.

A practical experiment on digital art preservation using emulation is presented in [Jones, 2004]. The artwork "The Erl King" (1982-85) by Grahame Weinbren and Roberta Friedman consisted of obsolete and generic hardware and software. Different preservation actions were performed during the case study. An interactive website as a re-interpretation of the original artwork was set up for visitors of the museum website. For a setting inside the museum emulation was used as a strategy. The original software for the artwork was written by the artist, so it was a very high priority to also preserve the original code.

The Ars Electronica[20] is a platform for digital art and media culture. Besides a yearly festival on art, technology, and society, it maintains a permanent media center and museum for interactive digital art. Founded in 1979, it maintains a wide array of digital artworks, many running on obsolete or generic hardware and software. In [Becker *et al.*, 2007] a case study on preservation planning on some of the artworks in the collection is shown.

Recently, the Museum of Modern Art (MoMa) started acquiring video games as artworks for their collection. The MoMa is acquiring not only the game itself but also as far as possible accompanying material like the source code or design documents. For games that are no longer running on original hardware, putting an emulated version of the game on display for visitors to interact with is planned[21].

The European project *Timeless Business Processes* (TIMBUS)[22] deals with the preservation of resilient business processes. Besides making sure that the execution context of a process processing data stays accessible, TIMBUS also concentrates on third party dependencies usually essential to render a process correctly. TIMBUS also aligns preservation actions with enterprise risk management (ERM) and business continuity management (BCM).

A project dealing with the challenges to preserve scientific experiments in data-intensive science is Workflow 4Ever (Wf4Ever)[23]. It defines models to describe scientific experiments by means of workflow-centric research objects and collects best practices for their creation and management, as well as analyzes and manages the decay in scientific workflows.

---

[19] Variable Media Network – `http://variablemedia.net/`

[20] Ars Electronica – `http://www.aec.at/`

[21] MoMa acquires video games – `http://www.moma.org/explore/inside_out/2012/11/29/video-games-14-in-the-collection-for-starters`

[22] TIMBUS – `http://timbusproject.net/`

[23] Wf4Ever – `http://www.wf4ever-project.org/`

Data and Software Preservation for Open Science (DASPOS)[24] explores the technical problems connected to the preservation of the massive data sets of High Energy Physics (HEP) experiment data. While the project concentrates on HEP experiments, the goal is to create a template for preservation useful for different disciplines.

A multitude of other projects exist, a survey of European initiatives is shown in [Strodl *et al.*, 2011].

2.11 Summary

In this chapter we described some of the foundations for work carried out in this thesis. Basic concepts for the view-path of a digital object, as well as different strategies of how to preserve a digital object for the long term were presented, with a special focus on emulation. We showed different concepts used in emulation as a digital preservation strategy. Keeping the rendering of a digital object authentic after applying a preservation action is crucial for the object to stay usable for the designated community. Preservation planning as part of the OAIS ensures that the best preservation action in a certain context is selected. Evaluating preservation actions by characterizing the object is part of preservation planning.

In this work we will show how the rendering of complex digital objects can be used for evaluation of preservation actions both in preservation planning but also in later phases in the preservation workflow presented in this chapter. To put our work in the context of the research field, we also introduced some projects doing work on the preservation of complex digital objects.

[24] DASPOS – `https://daspos.crc.nd.edu/`

Chapter 3

Comparing Renderings of Migrated and Emulated Digital Objects

3.1 Introduction

Emulation and migration as the main strategies in digital preservation are usually treated as entirely different strategies. While for evaluating the success of a migration action the object properties are compared to check if significant properties of the object change, the rendering environments of the digital object (i.e., the environment the object was originally rendered in and the environment it will be rendered in after the migration) are quite frequently not taken into account. With emulation, on the other hand, the digital object does not change, so only the rendering of the digital object in the original environment and the rendering in the emulated environment are compared to see if the rendering is identical.

What is usually not considered in the evaluation of a migration action is that every extraction of significant properties of an object is already a form of rendering i.e. an interpretation of this object. Even though not necessarily directly visible to the user, the object is rendered by the routines used to extract the properties. The problem with this approach is that the program ”rendering” the object is neither necessarily the program originally used to render it nor the one that will be used to render it and thus the results are not necessarily authentic to the original rendering once the object is rendered in a different environment. (Note that ”rendering” in this context is not restricted to the visual display of an object. It refers to all kind of interpretations of an object and the resulting effect on an environment, be it visual, acoustic or effects on a system state, files stored on media, or communication/voltage levels on I/O ports, etc.)

In this chapter we will show how migration and emulation strategies affect the view-path of a digital object on different levels. By applying a digital preservation

action the view-path is changed, both for emulation and migration actions. The rendering using the new view-path has to be compared to the object rendered in the original view-path to evaluate changes of significant properties of the object when rendered in the new view-path. While this is common knowledge when evaluating emulators this chapter will emphasize the similarities in rendering of a migrated object and the changes in the view-path.

We will show in this chapter that both the evaluation of a migration action and an emulation action should thus always include the combination between object and rendering environment using an evaluation strategy as shown in the framework in Chapters 4 and 5. Research shown in this chapter has been published in [Guttenbrunner and Rauber, 2012b].

3.2 Changing the View-Path Using Emulation

Using emulation as a digital preservation strategy, we keep the object unchanged. This means we can replace three of the layers in the original view-path shown in Figure 2.1. Based on the different levels of emulation, we have the following possible resulting view-paths:

Application The most simple level on which we can change the rendering is by replacing the application as seen in Figure 3.1b. An example for this would be to use FreePDF instead of Adobe Acrobat Reader (or using a different version of Adobe Acrobat Reader) to render a PDF document. While not a long term strategy (as the hardware and operating system stay the same), this could be the case if an application gets obsolete and needs to be replaced by a different one.

Operating System The operating system is usually very closely tied to the underlying hardware. If we keep the original application in place in the view-path and use a translation layer emulating the originally used operating system, we get an extra layer in the view-path as shown in Figure 3.1c. As with the strategy of emulating on the application level, this is not a long term strategy, as the application would still need to be run on the same hardware. This strategy would be used if the operating system gets obsolete and needs to be replaced by a different one, still running on a compatible physical hardware.

Hardware Finally, we can replace the actual physical hardware by an emulator replacing its functionality. The original application and the operating system used to run the application stay unchanged. Introducing an emulator for the hardware means that usually two additional layers are introduced into

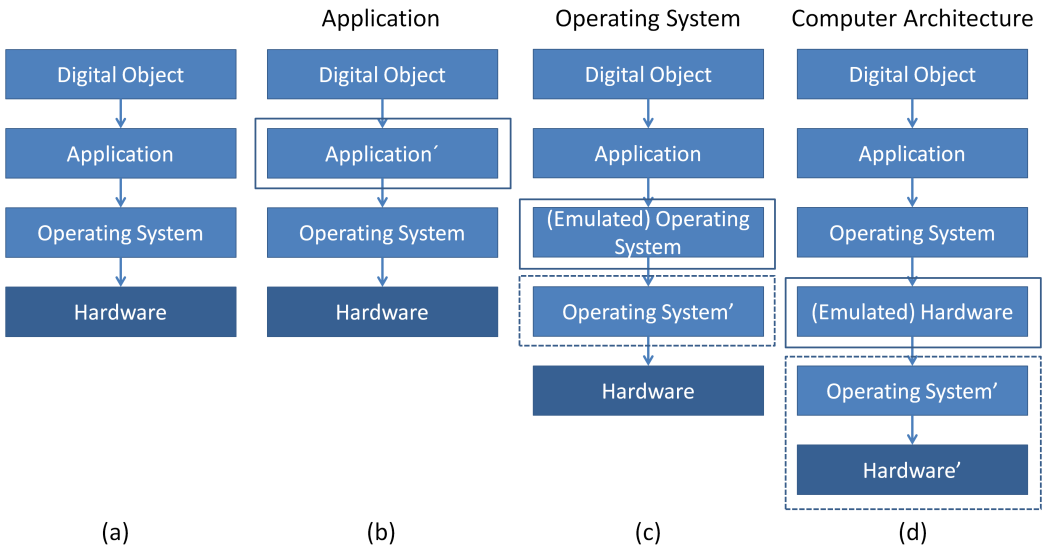


Figure 3.1: Changing the view-path by emulating the application. Full boxes show the emulated layer that changes, dashed boxes additional layer(s) that need to be introduced. Shown are the original view-path (a) and view-paths emulating the application (b), the operating system (c), and the computer architecture (d).

the view-path as shown in Figure 3.1d: the emulator of the original hardware and the operating system on which the emulator runs. If the emulator is based on a virtual machine (e.g., Java VM), we get another additional layer. As in this approach the physical hardware is replaced by software that can be ported to new machines (the emulator), this is obviously also the most promising approach for a digital preservation strategy.

3.3 Changing the View-Path using Migration

If we use migration as a digital preservation strategy, the digital object is changed. Based on the view-path shown in Figure 2.1 we get the following new possible view-path options.

Digital Object If only the digital object is changed, but the same application is used to render the digital object as shown in Figure 3.2b, the remainder of the view-path remains unchanged. An example would be to convert an image in format BMP to PNG and use the same tool to view the image. The use as a strategy is limited as the motivation for a digital preservation is usually that the application used to render the digital object would get

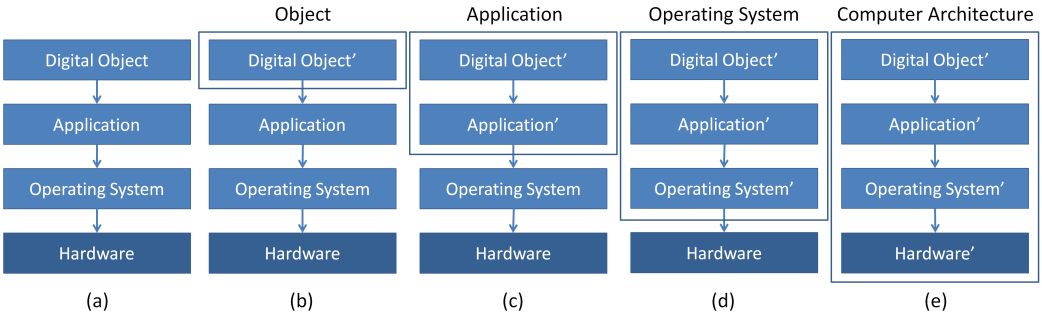


Figure 3.2: Changing the view-path by migrating the digital object. Boxes around the layers highlight the layers that change. Shown are the original view-path (a) and the changed view-path when the object is migrated (b), when a different application is used to render it (c), when a different operating system has to be used for the application (d), and when the computer architecture changes (e).

obsolete, and not just the format of the digital object. Further, note that while the application may formally remain unchanged (e.g., the same image viewer application), a different functional component will be used within the same application, thus actually constituting a different rendering engine.

Application In addition to changing the digital object we usually also need to change the application that is used to render the object. An example would be to migrate all documents from Microsoft Word DOC-format to PDF-A and using Adobe Acrobat Reader instead of Microsoft Word to render the objects. On this level the operating system and the hardware would still be unchanged as shown in Figure 3.2c. (Note that, strictly speaking, also a new version of an application basically constitutes a new application that needs to be verified as such.)

Operating System The operating system has to be replaced by a different one once it gets obsolete as shown in Figure 3.2d. Using a new operating system means usually to replace the applications (at least with new versions) as well. While it might still be possible to run it on the same hardware, this already changes 3 layers in the view-path. An example would be to migrate the object from Microsoft Word 95 format to Microsoft Word 2007 format, and at the same time change the application Microsoft Word 95 to Microsoft Word 2007 and the operating system from Microsoft Windows 95 to Microsoft Windows 7.

Hardware As hardware gets obsolete as well, we finally will have to replace the

actual physical hardware by a different one. The upgrade in hardware usually involves an upgrade in the operating system with all the before mentioned steps. As can be seen in Figure 3.2(e) this involves potentially to exchange all the layers in the view-path. Note, that we may find a situation where we replace (part of) the underlying hardware, but are still able to use, at least nominally, the same operating system. Yet, this usually needs to be treated like a complete replacement of also the operating system, and potentially any applications building on top of it as at least the operating system will usually not be identical: different drivers may be required to access different hardware components, API's behave differently, etc.

Other, more limited options, such as migrating only a specific layer, keeping those on top of it unchanged are, of course, possible as well. For example, we may replace (part of) the hardware configuration of the system, running the same operating system on top of it (albeit likely with different libraries being used, thus resulting effectively in a different operating system). Similarly, we might upgrade an operating system e.g. to a new version, or only partially by applying new security patches, keeping the viewer application identical. In any case, we are changing the view-path, thus requiring evaluation of the rendering performance of an object in the new setting.

3.4 Generalized View on the Performance of Digital Objects

Emulation strategies usually replace one layer in the view-path or add an additional layer as an interface between the emulated view-path and the new underlying layers. This adds complexity to the rendering of a digital object, with side-effects of the introduction of the new layers having to be considered. For evaluating the rendering we need to do a comparison between the rendering in the original view-path and the new view-path to judge where the rendering differs and if the digital preservation action is useful for the given setting. Any change in the view-path due to a layer getting obsolete has to lead to a re-evaluation of the rendering.

For migration the view-path of the digital object stays only the same if the same application on the same system is used. In every other case the view-path changes and every lower layer frequently leads to one or more layers on top of it being changed as well. As different code segments in the same application will be used to render an object (e.g. a PNG library instead of a JPG library) this should actually be viewed as using a different application to render the object in the new format, even if the two rendering engines are "wrapped" as one single, unchanged

application. The potential side-effect is thus even bigger than when emulating the original rendering environment of a digital object. To judge if there are side-effects due to the change in the rendering environment we have to compare the new rendering to the original rendering, just as we would do in an emulation setting. As with an emulation action, any change in the new view-path has to be re-evaluated. We need to keep in mind that it is not enough to compare the migrated object in the current environment and assume that the rendering will be the same in a future environment. If the digital object is migrated while preserving it for the long-term it has to be validated against the original rendering of the object to be able to make a statement about the authenticity of the rendering.

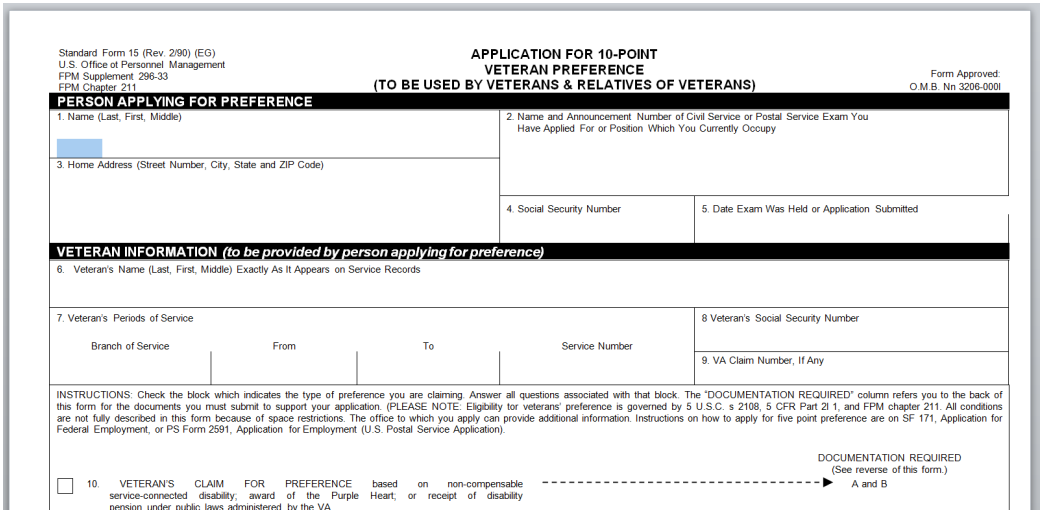
As an example for the changes in object format and its influence on the rendering a document in Microsoft Word for Windows 97-2003 (doc) format is rendered using both Microsoft Office 2007 (Figure 3.3) and OpenOffice 3.4 (Figure 3.5), both running on a PC using Windows 7 operating system. While small differences in the rendering are visible (e.g. horizontal spacing, some changed word wrapping in the “Instructions” part of the document), the properties of the document considered significant for most settings are unchanged. Next, the document has been migrated to Microsoft Word for Windows 2007 (docx) format. The document is rendered exactly identical in Microsoft Word 2007 (Figure 3.4), so the significant properties stored in the file (layout information, content, metadata, etc.) are still present. However, in OpenOffice 3.4 the rendering looks completely different to a state that is unusable for most applications (Figure 3.6).

Additionally to the observations made in the experiments in this chapter, it should be noted, that the same file is identified differently using characterization tools, thus different characteristics might be extracted, leading to differences in the extracted significant properties on migrated files. In [Tarrant and Carr, 2012] Tarrant et. al. show how PDF files are characterized and identified differently over the course of a view month, using different characterization tools shown in Section 2.8.2.

Thus, any object with preservation actions applied, whether emulation or migration, must be evaluated in the context of its rendering environment, as changes to significant properties may occur that are not properly detectable from an analysis of the static object properties only. From this perspective, migration and emulation behave virtually identically.

3.5 Summary

In this chapter we showed how the digital preservation strategies migration and emulation on different levels affect the view-path of a digital object. We then compared the effects of the two strategies and showed that the change in view-path



Standard Form 15 (Rev. 2/90) (EG)
U.S. Office ot Personnel Management
FPM Supplement 296-33
FPM Chapter 211

APPLICATION FOR 10-POINT VETERAN PREFERENCE (TO BE USED BY VETERANS & RELATIVES OF VETERANS)

Form Approved:
O.M.B. Nn 3206-000I

PERSON APPLYING FOR PREFERENCE

1. Name (Last, First, Middle)
2. Name and Announcement Number of Civil Service or Postal Service Exam You Have Applied For or Position Which You Currently Occupy
3. Home Address (Street Number, City, State and ZIP Code)
4. Social Security Number
5. Date Exam Was Held or Application Submitted

VETERAN INFORMATION *(to be provided by person applying for preference)*

6. Veteran's Name (Last, First, Middle) Exactly As It Appears on Service Records
7. Veteran's Periods of Service
   Branch of Service | From | To | Service Number
8. Veteran's Social Security Number
9. VA Claim Number, If Any

INSTRUCTIONS: Check the block which indicates the type of preference you are claiming. Answer all questions associated with that block. The "DOCUMENTATION REQUIRED" column refers you to the back of this form for the documents you must submit to support your application. (PLEASE NOTE: Eligibility tor veterans' preference is governed by 5 U.S.C. s 2108, 5 CFR Part 2l 1, and FPM chapter 211. All conditions are not fully described in this form because of space restrictions. The office to which you apply can provide additional information. Instructions on how to apply for five point preference are on SF 171, Application for Federal Employment, or PS Form 2591, Application for Employment (U.S. Postal Service Application).

DOCUMENTATION REQUIRED (See reverse of this form.)

☐ 10. VETERAN'S CLAIM FOR PREFERENCE based on non-compensable service-connected disability; award of the Purple Heart; or receipt of disability pension under public laws administered by the VA ----------▶ A and B

Figure 3.3: Sample layout region of a document in MS Word for Windows 97-2003 format rendered in MS Office 2007.

Standard Form 15 (Rev. 2/90) (EG)
U.S. Office ot Personnel Management
FPM Supplement 296-33
FPM Chapter 211

APPLICATION FOR 10-POINT VETERAN PREFERENCE (TO BE USED BY VETERANS & RELATIVES OF VETERANS)

Form Approved:
O.M.B. Nn 3206-000I

PERSON APPLYING FOR PREFERENCE

1. Name (Last, First, Middle)
2. Name and Announcement Number of Civil Service or Postal Service Exam You Have Applied For or Position Which You Currently Occupy
3. Home Address (Street Number, City, State and ZIP Code)
4. Social Security Number
5. Date Exam Was Held or Application Submitted

VETERAN INFORMATION *(to be provided by person applying for preference)*

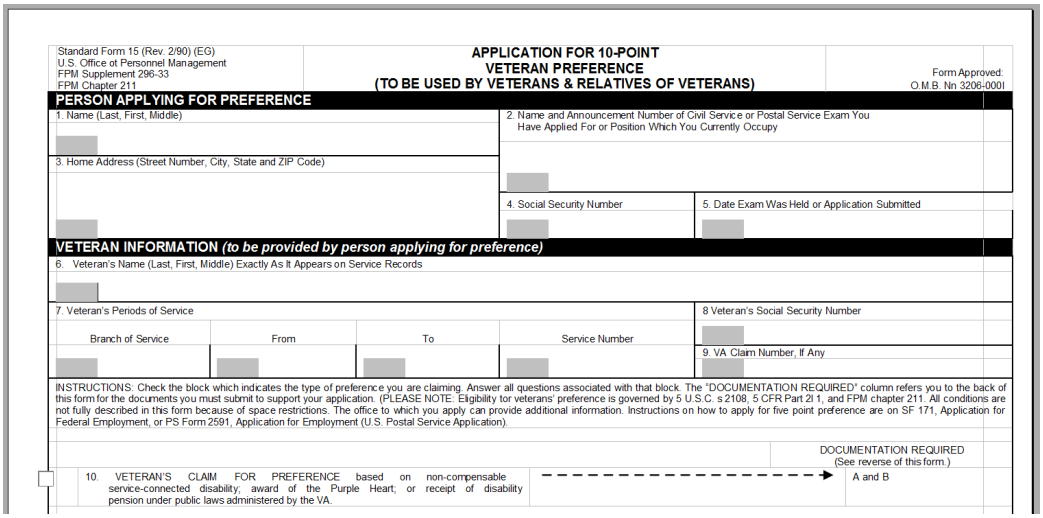
6. Veteran's Name (Last, First, Middle) Exactly As It Appears on Service Records
7. Veteran's Periods of Service
   Branch of Service | From | To | Service Number
8. Veteran's Social Security Number
9. VA Claim Number, If Any

INSTRUCTIONS: Check the block which indicates the type of preference you are claiming. Answer all questions associated with that block. The "DOCUMENTATION REQUIRED" column refers you to the back of this form for the documents you must submit to support your application. (PLEASE NOTE: Eligibility tor veterans' preference is governed by 5 U.S.C. s 2108, 5 CFR Part 2l 1, and FPM chapter 211. All conditions are not fully described in this form because of space restrictions. The office to which you apply can provide additional information. Instructions on how to apply for five point preference are on SF 171, Application for Federal Employment, or PS Form 2591, Application for Employment (U.S. Postal Service Application).

DOCUMENTATION REQUIRED (See reverse of this form.)

☐ 10. VETERAN'S CLAIM FOR PREFERENCE based on non-compensable service-connected disability; award of the Purple Heart; or receipt of disability pension under public laws administered by the VA. ----------▶ A and B

Figure 3.4: Sample layout region of a document in MS Word for Windows 97-2003 format rendered in OpenOffice 3.4.



Standard Form 15 (Rev. 2/90) (EG)
U.S. Office ot Personnel Management
FPM Supplement 296-33
FPM Chapter 211

APPLICATION FOR 10-POINT VETERAN PREFERENCE (TO BE USED BY VETERANS & RELATIVES OF VETERANS)

Form Approved: O.M.B. Nn 3206-000I

PERSON APPLYING FOR PREFERENCE

1. Name (Last, First, Middle)
2. Name and Announcement Number of Civil Service or Postal Service Exam You Have Applied For or Position Which You Currently Occupy
3. Home Address (Street Number, City, State and ZIP Code)
4. Social Security Number
5. Date Exam Was Held or Application Submitted

VETERAN INFORMATION *(to be provided by person applying for preference)*

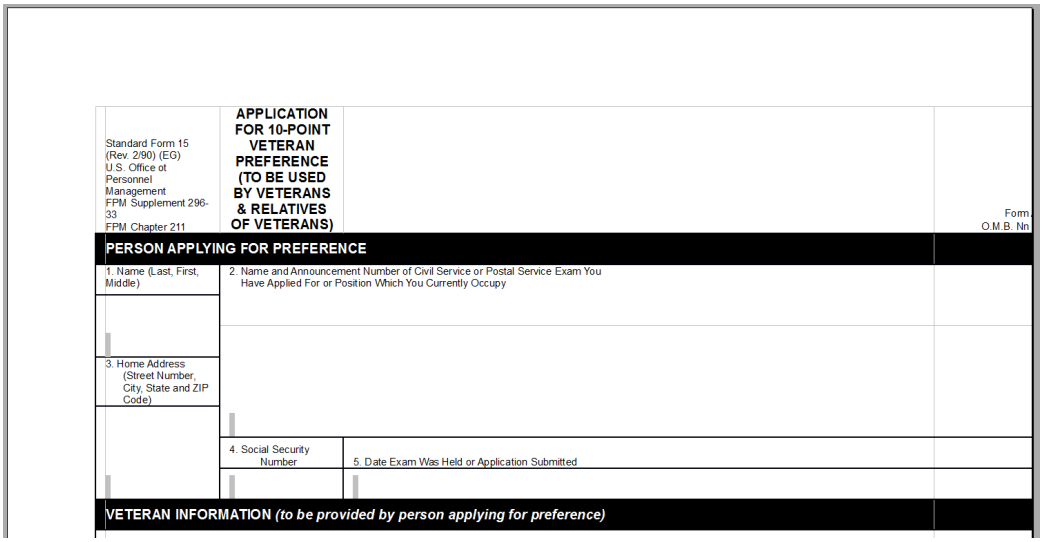
6. Veteran's Name (Last, First, Middle) Exactly As It Appears on Service Records
7. Veteran's Periods of Service
   Branch of Service | From | To | Service Number
8 Veteran's Social Security Number
9. VA Claim Number, If Any

INSTRUCTIONS: Check the block which indicates the type of preference you are claiming. Answer all questions associated with that block. The "DOCUMENTATION REQUIRED" column refers you to the back of this form for the documents you must submit to support your application. (PLEASE NOTE: Eligibility tor veterans' preference is governed by 5 U.S.C. s 2108, 5 CFR Part 21 1, and FPM chapter 211. All conditions are not fully described in this form because of space restrictions. The office to which you apply can provide additional information. Instructions on how to apply for five point preference are on SF 171, Application for Federal Employment, or PS Form 2591, Application for Employment (U.S. Postal Service Application).

DOCUMENTATION REQUIRED (See reverse of this form.)

10. VETERAN'S CLAIM FOR PREFERENCE based on non-compensable service-connected disability; award of the Purple Heart; or receipt of disability pension under public laws administered by the VA. → A and B

Figure 3.5: Sample layout region of a document in MS Word for Windows 97-2003 format migrated to MS Word for Windows 2007 (docx) format rendered in MS Office 2007.



Standard Form 15
(Rev. 2/90) (EG)
U.S. Office ot Personnel Management
FPM Supplement 296-33
FPM Chapter 211

APPLICATION FOR 10-POINT VETERAN PREFERENCE (TO BE USED BY VETERANS & RELATIVES OF VETERANS)

Form
O.M.B. Nn

PERSON APPLYING FOR PREFERENCE

1. Name (Last, First, Middle)
2. Name and Announcement Number of Civil Service or Postal Service Exam You Have Applied For or Position Which You Currently Occupy
3. Home Address (Street Number, City, State and ZIP Code)
4. Social Security Number
5. Date Exam Was Held or Application Submitted

VETERAN INFORMATION *(to be provided by person applying for preference)*

Figure 3.6: Sample layout region of a document in MS Word for Windows 97-2003 format migrated to MS Word for Windows 2007 (docx) format rendered in OpenOffice 3.4.

makes it necessary to take the rendering environment into account when evaluating any digital preservation action, be it a migration or an emulation action.

Based on these observations we showed that preservation planning strategies have to be adapted when using migration as a digital preservation strategy. To ensure an authentic rendering it is necessary to take the target rendering environment after migration into account. Any change in the target rendering environment (i.e., a layer in the view-path becoming obsolete) has to lead to a new preservation planning iteration evaluating the changed environment, even if the format of the digital object stays unchanged. This makes the evaluation process for migration strategies essentially identical to the evaluation of emulation strategies.

In the next chapter we will show a framework to evaluate the rendering of digital objects.

Chapter 4

Describing a Digital Artifact and its Environment

4.1 Introduction

In Chapter 3 we showed that applying any digital preservation action, be it migration or emulation, changes the view-path of the digital object on at least one level. This means that the renderings of the different view-paths have to be compared for an evaluation if the conceptual layer of the digital object is preserved accurately.

Knowing about the behavior of the digital artifacts under evaluation is essential to determine whether the behavior is properly reproduced by the rendering environment. In this chapter we will explain some key characteristics of a digital object and its view-path, as well as influences on the objects behavior that has to be gathered to properly evaluate its rendering process. The decisions on what level and at which moments in time comparisons have to be made are shown and used as a basis for a systematic evaluation of rendered digital objects. Parts of this chapter have been published in [Guttenbrunner and Rauber, 2012c].

4.2 Describing the Digital Artifact

It is necessary to explicitly document, which characteristics of a digital object are relevant for a given preservation scenario, i.e. which of these constitute significant properties. The degree of preservation of these in a different view-path ultimately forms the basis for the evaluation and comparison of the performance of different potential environments as part of the preservation planning process [Becker and Rauber, 2011b]. These significant properties are collected as part of the requirements analysis phase for a digital preservation endeavor and may be documented e.g. in the form of an objective tree, grouped into different categories. Several

possible preservation actions, such as different emulation environments, may then be used to render an object and evaluate its performance. A range of properties usually need to be considered that can be structurally subdivided in Object and Action specific properties, i.e. those pertaining to characteristics of the digital object (e.g. reacts correctly to interaction, performs calculations identically to the original, displays colors correctly) and those pertaining to the action (emulation or migration) applied (e.g. emulation environment is open source, memory consumption, required hardware platform, licenses). (See [Becker and Rauber, 2011a] for a detailed discussion and examples of such properties and means how to measure their performance.)

In the following two subsections we want to focus on two object characteristics that are particularly relevant for dynamic objects and evaluation settings that usually tend to call for emulation strategies as suitable preservation actions, but are just as valid for migration strategies.

4.2.1 Determinism of the Digital Artifact

Before trying to evaluate the behavior of the digital artifact in original form and in a different view-path it is necessary to understand the impacts of internal and external events on the object's behavior. For evaluating the differences in behavior of an object in the original environment and the new environment it is necessary to ensure that the object behaves the same under the same external conditions. Otherwise it is not possible to determine if changes in behavior are an effect of changes in the view-path or due to seemingly random behavior, i.e. changes in data influencing the object's behavior.

Deterministic Behavior

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

Some digital objects can be rendered with a deterministic behavior. A spreadsheet or database application will usually under the same software- and hardware-conditions and after applying the same input show the same results. Replaying a video or audio file has to produce the same results independent from the external

influences listed above. A comparison of preservation action effects would in this case be rather simple.

Non-Deterministic Behavior

If we try to preserve a single object for the long term, the object is usually self contained, i.e. all the data specific for the object (and not created by interpreting the object for the rendering) can be extracted from the object's bitstream. On the other hand, if the behavior of a digital object depends on user input, hardware or random values, or has the data it uses modified by other processes, then it is not deterministic. If the object's rendering is not deterministic, i.e. relies on external data that influences how the object is rendered, we have to make sure that this data is provided to the rendering engine identically for each rendering. This ensures that differences in the rendering are triggered by differences in the rendering environment, and not by changes in the external data.

To allow an automated comparison of two renderings of the same object it is necessary to try to create a deterministic behavior. The rendering of the digital object can be made semi-deterministic by identifying the external events influencing the object's behavior and equalizing the values for these events in the different environments. This external data can come from different sources. It can be divided into two different categories depending on where it originates.

- *Locally controlled data:* Some data can be locally controlled as it is external data coming from the host system (e.g., real time clock, files stored on the host system's file system, random number generator). If a digital object is separated from its original environment, e.g., by virtualizing its environment, this data is under control of the new rendering environment and can thus be provided identically for every rendering cycle of a digital object, to ensure deterministic rendering.

- *Not locally controlled data:* Some data is provided to a digital object through interfaces from other systems that are not local to the digital object's host system (e.g., data retrieved from web services, user input, files stored on the network). This kind of data may change with every rendering cycle of the digital object and is thus much more difficult to keep identical for deterministic execution. Additionally, as the service is located remotely, it can provide data in the form of a black box, i.e., the exact functionality of the services may not be known. The list of possible interfaces on which data is fed into the system is practically endless and obviously depends of the technical characteristics and interfaces of the hardware in the view-path. Some examples include human input through keyboard, mouse, joysticks,

> microphones, but also external data supplied using serial or parallel interfaces, network interfaces, any kind of USB devices, storage media, MIDI interfaces, etc.

For example, applying automated user input at the exact same moment in the rendering process can be used to eliminate differences in behavior depending on user input. Random number generators are usually initialized with values derived from hardware values like system time or the position of the raster beam on the screen. These external influences to the digital object can be made predictable by setting the system clock in the emulation environment to the same time as on the original environment during execution of the digital object and by using a pixel-exact emulation of the original system, i.e. updating every pixel on the screen with the corresponding CPU cycles. Especially video games or interactive digital art usually heavily depend on external or random elements. An example of a video game with unusual external events is metro-wardive[1] for the Nintendo DS/Apple iPhone which uses wireless-LAN-waves to create in-game objects.

When a digital object is described for preservation, both the internal (locally controlled data) and the external (not-locally controlled data) dependencies have to be documented and captured for a comparison of the rendering process.

Ideally, a digital object is still deployed in its original environment when it is prepared for storage in an archive in a virtualized form, to make it independent of its original environment. Thus, as a first step, both locally controlled and not locally controlled external data are captured when validating the proper execution of a process (or other digital object) after being virtualized to make sure that it behaves as expected.

In an ideal case the data can be captured using the rendering environment, as the capturing can then be directly synchronized with the execution of the process, e.g., triggered by a specific number of processor cycles having been executed in the virtual environment, or a specific amount of execution time passed in the virtual environment. If the rendering environment does not support capturing of external events, it can also be captured by processes external to the virtual environment, e.g., a listener that captures data from and to a web service [Miksa *et al.*, 2013].

As one example for external events influencing interactive objects we show how to address human interaction with the environment. The different methods to capture (and later recreate) these events are listed here. In principle human interaction is similar to any other event that influences the rendering of a digital object. The methods listed here can thus be seen as a template for any kind of event, with the same possibilities and problems discussed.

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

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

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

Use of Macros Recording a set of user actions along with the delay between them is one way of automating input to digital artifacts. One of the downsides is the difference of recording on one system and replaying on the other system, which is a change in conditions under which the system is running. By running a macro in the new environment a change in speed will not affect the execution of the macro, as the software replaying the macro also runs at a changed execution rate.

The use of macros is only possible on operating systems where processes can run parallel to the execution of the view-path of the object. It would not be possible for the preservation of games on video game consoles or for applications on most home computers or personal computers running simple disc operating systems. In some cases the digital object provides support for applying interactive options automatically either through a scripting language or by recording macros. One example would be Microsoft Excel which allows both the recording of macros and offers a scripting language to automate input. Another example is the game Quake[2], which allows the recording of user actions and is able to replay them in the game engine. Note, that the granularity of timing is different for the two examples presented here: while differences in a few frames might not make a difference for the automated input in Excel, it might make a difference for the player's

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

survival in Quake. Like in real-time systems, even a tick-based granularity can be necessary.

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

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

Recording and Applying Input on a Hardware Level The method of recording and applying input with the least side-effects on the environment is at the hardware level: recording the input directly from the input devices and applying this as an input-macro to the new environment. This kind of input macros contain the event (e.g., key X pressed, joystick left) as well as a time relative to elapsed process execution time between these events. One advantage in the use of macros recorded from the input device and applied to the new environment is that they can be used on systems that do not allow parallel execution of processes (old home-computers, video game consoles, embedded systems). It is also possible for the new environment to adjust the timing of these input events depending on the speed difference between original environment and new environment. The disadvantage is that it is

not possible to use only software components but that special hardware has to be built to intercept the communication of the input devices and the target system. A possible solution in software would be to use an emulator which is known to work nearly perfectly and use an input recording feature of the emulator to capture the input events. This feature and a standardized format are one of the requirements that have to be researched for future emulators as a digital preservation requirement. Ultimately, hard real-time system-like behavior ([Liu, 2000]) is necessary to ensure perfect process state level execution of emulations.

Use of Software Buffers A method very similar to applying keys on an actual hardware level is to use software buffers in the operating system that hold, e.g., key presses. This method can only be used, if the operating system has a specified memory region that can be used for this purpose. Similarly to the Hardware Level approach, this approach would allow a very close control of extracting/inserting data from/to the buffer supported by a virtual layer (e.g., emulator) at the exact same time the data was inserted during a previous rendering cycle. The disadvantage over control on the hardware level is that only one operating system (version) running on the hardware would be supported by this approach as opposed to all possible operating systems if the recording/applying is done on a lower level.

Testing an Object for Determinism

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

The methods described in this thesis to compare a digital object in different rendering environments can also be used to test an object for determinism. If the behavior of the object in the same view-path is different for different rendering cycles, then external events influence the rendering. By capturing the different types of external events possible and reapplying those to the rendering it is thus possible to reduce the effects of external events until all are under control of the execution mechanism. For example, a digital artwork can be rendered in its environment and re-rendered again under the same conditions. If the results of the rendering are exactly the same, then it is very likely that the object behaves deterministically. Otherwise external factors like random number generators based on the rendering time can be eliminated by setting the same time at the beginning of every rendering cycle. Likewise human input can be made consistent over the different rendering cycles.

4.2.2 Significant States of a Digital Artifact

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

With emulation and dynamic objects one can again have to deal with two different situations, namely a sequence of target states of the object where it is only necessary to capture the properties of the rendered object after some period of emulation, and a continuously changing state of the object, which again may be conceived as a series of designated states. It is necessary to determine, which states of the digital object are significant for the designated user community of the object types for which the rendering environment should be used, as usually not all data created by the rendering process is significant. Depending on the digital object's rendering process, significant points in the rendering have to be defined. Typical points are all the steps where an interaction between the system and the user or different systems takes place. E.g., when the rendering system waits for a user input, the rendered screen shown right before the user input and prompting the user for an action is significant. Also data transferred on an interface to a different system as a response to a request from that system is significant.

Target State

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

Series of States

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

Continuous Stream

If a continuous stream of states has to be compared it can either be done by recording the full output (e.g., image or sound stream) on the chosen layer (as will be discussed in Section 4.3.3) and comparing the resulting streams. Besides, e.g., aspect ratio, brightness settings, changes in color, audio frequency also the speed factor has to be taken into account. Images on the stream from the emulated environment can be slowed down or sped up, and not necessarily by a fixed factor over the whole stream (e.g., more objects on the screen can potentially slow down the emulation at some point). Advanced comparison mechanisms have to be used — probably with human interaction — to link certain landmark events in the stream (e.g., change in scenery in a video game from selection menu to in-game graphics).

4.3 Describing the Rendering Environment

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

Table 4.1: Example digital objects and possible hardware / software / output device combinations.

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

4.3.1 Selecting the Reference Rendering Environment

The view-path of a digital object contains the hardware and all the secondary digital objects needed to render an object along with their configuration. It is possible to use different view-paths to display the same object. It is practicable to define one view-path out of all the possible ones to compare the effects of emulation, even though different view-paths would be possible. For example, the same Microsoft Word-document can be rendered on the same hardware and the same operating system with an identical set of fonts installed using different versions of Microsoft Word or using Open Office which all can give different rendered results for the same document. Identifying the correct rendering for a digital object means that we have to determine what the desired rendering we want to compare against actually is. A digital object can potentially be executed in a wide variety of hardware and software platform combinations depending on various usage scenarios and the resulting output can be played back on a potentially unlimited number of output devices. That is, most objects will have several (equally authentic) renderings usually targeted at specific designated communities (different users or usage scenarios). Some examples are shown in Table 4.1.

In the least complex case the digital object can be rendered only on one hardware platform with predetermined software and built-in output-devices. In the most complex example a digital object can be rendered by a variety of software products running on different operating systems which in turn can be used on different hardware configurations resulting in a variety of rather different information

objects. Depending on the usage scenario the correct rendering environment of an HTML page may be anything from a web browser via a simple HTML-editor to an entire content management system. A specific XML-data file may be rendered in an XML-editor, being used to drive an application as input configuration, or used as basis for a visualization. The rendered output can be played back on a number of different output devices with different characteristics (e.g. CRT-TV vs. LCD-TV).

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

- hardware and software configuration typically used to render the digital object (e.g. configuration of a standardized office PC used to create the object). A repository of highly standardized typical system configurations may be helpful to minimize the effort in maintaining the rendering environments. Yet, in some cases, the very specific configuration encountered in a setting will need to be defined as reference environment.

- play-back devices typically used to observe the digital object (e.g. a CRT TV used to render a video game from the 1980's instead of a modern LCD screen with completely different display characteristics, synthesized voice output on contemporary mobile phone speaker system vs. studio quality speakers in a museum setting)

- usage scenario, i.e. what purpose(s) the object is being preserved for. As different scenarios (e.g. simple replay, providing different views on data, or providing evidence in legal investigations) may require different renderings of an object, potentially multiple reference environments need to be defined.

Observing the digital object in its original environment is only possible if the reference environment can be obtained in working order using the original hardware, software and play-back devices. This is when an original rendering environment will still be available, and when — as part of the documentation of the object and its rendering environment — evidence for measuring the significant properties of an object's rendering in a potentially different environment has to be collected. In cases where digital objects are unearthed at a later stage (e.g. through digital archeology on obsolete media formats) the ground truth has either to be set empirically by comparing either different rendering environments for a system (e.g. if

five out of six emulators render an image in white/blue and the rendered image are clouds on a sky, then it is very likely that the colors are rendered incorrectly on one of the emulators) or by using other sources like software design documents such as use case scenarios, requirements specification, and software testing as well as video documentation.

4.3.2 Describing the View-path of a Digital Artifact

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

The information that has to be collected can be split into hardware configuration and all the necessary software to render the digital object with their respective configuration. The following sections provide an overview of the various elements that need to be documented and recreated or standardized in an emulation setting in order to facilitate proper evaluation. The focus of this description is not to provide an exhaustive list, but to illustrate some core aspects as background for the evaluation framework presented in Chapters 4 and 5.

Hardware Configuration

Besides the obvious CPU type and configuration as well as memory size, configuration and speed, other hardware components influence the rendering of an object as well and have to be considered, e.g.:

Graphics-card, physics-card Especially for 3D-rendering the used graphics- and physics cards are having a major influence on the resulting rendered images. Also to be documented are the settings of device drivers.

Sound-card The sound-processor that is used for creating the audible output has an influence on the characteristics of the output audio signal.

Input-Devices Special input devices may be necessary not only to recreate the original look & feel, they also may be capable of recording all input activity to provide identical input sequences in the emulated environment.

Output-Devices The output device plays an important role on the resulting image/sound just as much as the rendering does. Aspect ratio and display settings of an output device have to be considered if the resulting output is compared not after computing as a digital image, but after creating an actual analogue or digital output.

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

On some platforms tools can assist in the description of hardware properties by determining hardware- and software settings.

Operating System and Configuration

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

Secondary Digital Objects

Other digital objects besides the operating system are needed to display the digital artifact itself or for running the viewer- or editor application displaying the object. Some examples of those are:

- Virtual Machine (e.g., Java Virtual Machine, .net)
- Database software (e.g., MySQL)
- Libraries (e.g., DirectX, Allegro[3])
- Software device drivers (e.g., ODBC drivers)
- Viewing- or editing-application (e.g., OpenOffice, PDF-Viewer)
- Fonts (for documents where fonts are not stored inside the file)
- Codecs (for decoding e.g., audio or video data streams)

For all these secondary digital objects, the version, but also configuration options have to be documented to allow the complete recreation in an emulated environment. Depending on the operating system, information about the different versions and libraries used on a system can potentially be extracted automatically using data created by package managers. A format for describing this information for package-based open source software is CUDF (Common Upgradeability Description Format)[4].

[3] Allegro game programming library — http://alleg.sourceforge.net/
[4] CUDF – `http://www.mancoosi.org/cudf/`

Digital Artifact to be Rendered

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

Additional Digital Objects not in the View-path

Besides the mentioned elements in the view-path of an object it is important to be aware of any other digital objects influencing the behavior of a computer system. Processes running in the background (e.g., virus scan software, remote desktop software) can significantly affect the performance of a system. The influence of semi-random elements on the execution of the digital artifact or on an application rendering the digital artifact has to be reduced to an absolute minimum to decrease the complexity of the system under evaluation.

4.3.3 Identifying Levels to Extract A Rendered Form

To extract a rendered object from its original environment it is essential to understand the different incarnations of a digital object when it is being rendered. These forms of a digital object on different levels are shown on the left in Figure 4.1. A rendered representation of the digital object has to be extracted on (a) suitable level(s). The significant properties of the object can then be evaluated.

Descriptive Form

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

Rendered Form in Memory

The next level where an incarnation of the digital object exists is as a rendered form after being processed by the viewer-application (or after being executed).

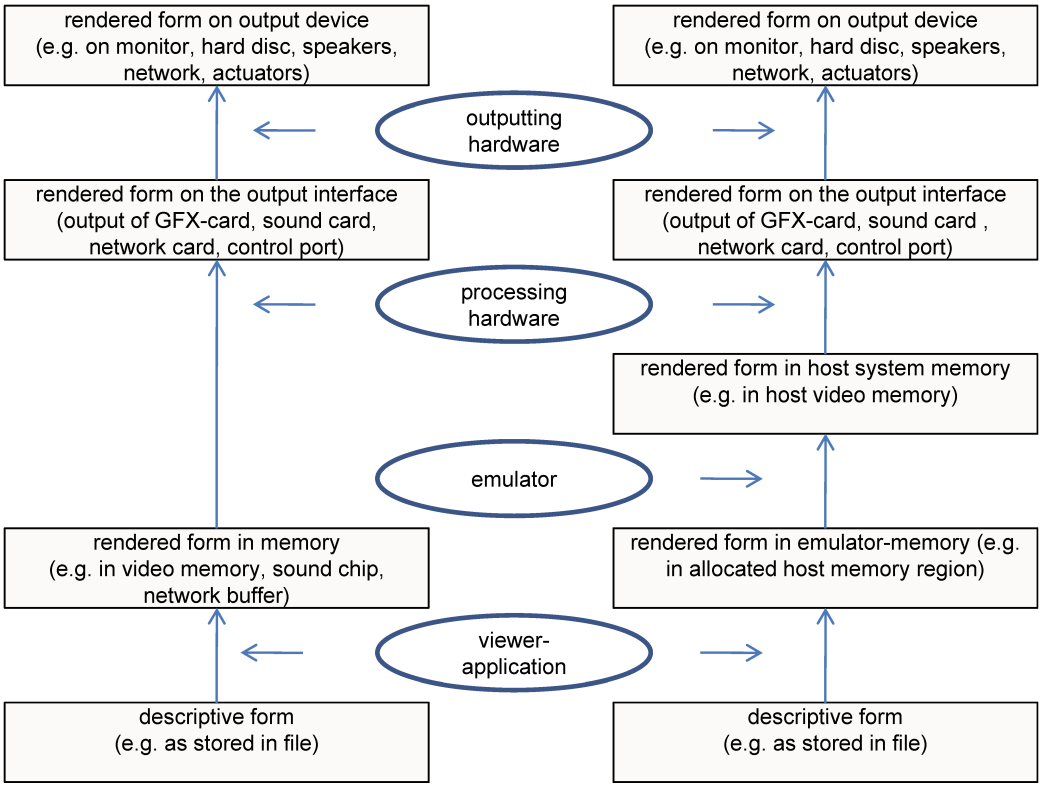


Figure 4.1: Different forms of a digital object in a system's memory. On the left the layers in an original system are shown, on the right the layers in a system hosting a virtualized view-path are shown.

On this level, e.g., the registers of the video hardware are filled or the memory area for video output is filled with the data to be displayed on screen. The same applies to other forms of object output, i.e., audio, port communication, or file system interaction. Applications recording this data on the system can be used to extract this data. An example would be a clipboard in an operating system. The output of a digital object could be copied into the clipboard and extracted through a defined interface.

Rendered Form on the Output Interface

The data in, e.g., the video memory is processed by the hardware converting it into a suitable signal for the displaying unit. Similarly, for networked systems, data is rendered on a network interface, or as instruction sets driving a storage unit. The data can then be captured on the output interface. While this is comparatively easy for one-way unencrypted communication, it gets more difficult for protocols that are not open and that involve special replies from the connected device (e.g., HDMI protocol).

Rendered Form on Output Device

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

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

4.4 Collecting Verification Data

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

the capturing of system behavior in other documentary form such as video documentation.

In Section 4.2.1 we described how external data influencing the rendering can be captured when executing a process and re-applied to a re-run of the rendering process to ensure deterministic execution using either capture- and replay functionality of the rendering environment or external tools. For a comparison of the rendering processes we also need to extract data on significant properties created during the rendering process that allows us to evaluate to which degree the rendering process of a re-deployed object meets the requirements of its designated community. Similarly to the external data that is captured during the execution of the rendering process, data created by the digital object and provided to the interfaces of the host system (e.g., the screen for rendering, audio signals, network data) can be captured for a comparison between different renderings. Also, data created for processing not available on the interfaces to the host system but only internally in the virtualized system can be captured from memory regions of the virtualized system.

In Figure 4.1 we described the levels on which data can be extracted in a system. On a first level the descriptive form of the digital object, i.e., the bitstream as stored in a file is shown. As no rendering takes place on this level, it is not usable for comparison for a successful rendering. Next, the data is rendered by some kind of interpreting hard- or software. Extracting data on this level can be done by using the interpreting application to export the rendering result, e.g., processed data. Once the rendering environment is virtualized, data about the rendering process is available on the virtual environment layer. Data on this level can be compared between different renderings in virtual environments, but not against the original system. Initiating data extraction on this level is only possible if the virtual environment supports extraction, e.g., by providing screenshots or log files about the rendering process. Both the original system and the virtual system provide the data rendered on some kind of output interface. Depending on the digital object and the data that has to be extracted, data can for example be captured using a listener watching specific network ports or hardware interfaces. Finally, data is eventually rendered on an output device, e.g., a display device, or actuators set by the rendering process. Characteristics of the output device influence the rendering, so a comparison on this level is done by capturing the rendered data after being rendered by the output device.

Depending on the significant properties of the digital object a suitable level for comparing the rendered data has to be chosen. For example, in most cases the rendered image as created on the host system can be compared to a screenshot, i.e., a video memory dump not considering screen characteristics, of the original system's rendering. In other situations the characteristics of the original output device can be a relevant factor that has to be taken into account [Phillips, 2010]. If

no output device influences the rendering, e.g., the process puts out a data stream that is not processed by an output interface but used as an input for a different process, the curator can decide to use a listener that captures the data stream for comparison. To verify if an application renders the provided data as expected in a new environment, an export from the application itself can be enough to compare the rendering of the data. The curator has to decide what the significant properties of a certain digital object are and depending on those select the suitable level for comparison.

The actual capturing of the data is done analogous to how data supplied to the virtual environment is captured - either directly if supported by the virtual environment (e.g., by using an option to create screen captures at certain points in the rendering process), or by using external tools that capture the results of the rendering on the host system, e.g., by capturing the network traffic created by the virtual environment, or capturing the rendered screen output of the virtual environment at certain intervals.

A curator thus needs to identify the significant states of the rendering process, i.e., if only the final state is of relevance (e.g., for a process that calculates data depending on its initial input and finishes once the data is calculated), if intermediary steps are relevant (e.g., for an interactive application), or if the entire rendering stream is relevant (e.g., for a rendered video stream).

The captured data then needs to be encapsulated along with the actual digital object in a package for storage in the archive. The information necessary to separate the digital object from its original environment, i.e. both the digital object itself (e.g., a workflow-definition for a workflow engine) as well as all the elements in the view-path (for virtualization) or information about how the object has to be interpreted (for a migrated form) need to be included as well.

4.5 Summary

In this chapter we presented a framework for capturing external events influencing the rendering of a digital object and how and where to extract data from the rendering environment. We discussed the information that has to be documented both for the digital object and its properties as well as the view-path used to render the object and to make the rendering deterministic. User input as one type of external events was explained and methods described to apply it to the rendering environment to eliminate side-effects occurring due to changes in external data. We showed the different levels on which rendered forms of the digital object are available on a system as well as the stages at which data has to be captured for evaluation.The concept of a reference environment acting as a ground truth was presented.

The major steps in the framework are to

- assess the determinism of a digital object,
- determine significant states, either the target state, a sequence of states or a full stream,
- select a reference rendering environment,
- determine the levels on which to compare the renderings,
- collect verification data.

Following the framework shown in this chapter it is possible to document the rendering process of a digital object and collect verification-data that can be used to verify the rendering of the object in a new environment. In Chapter 5 we will describe how a new view-path can be evaluated using the data described in this chapter. We will show a set of steps to follow for the evaluation as well as how to apply the evaluation in a preservation workflow.

Chapter 5

Evaluating a Digital Object's Rendering in a Changed Environment

5.1 Introduction

In Chapter 4 we showed what we need to document about a digital object and its view-path to collect verification data necessary for a comparison of its rendering in a new environment. In this chapter we will show how this data is used in a comparison workflow to compare two renderings of the same object in two different view-paths. We will also show how the comparison can be used in the different stages of a preservation workflow. The "Preservation Action Evaluation Framework" formed by the framework shown in Chapter 4 and the steps for evaluation shown in this chapter have been published in [Guttenbrunner and Rauber, 2012c].

5.2 Recreating the Rendering Environment

Once the digital object, the influences on its behavior and the environment in which it is originally rendered are documented, it is necessary to faithfully recreate these conditions in a rendered environment. In this section we first describe how the original view-path has to be restored. After recreating the original environment in a rendering environment, we again discuss the rendered form of a digital object now in a new environment and for comparison.

5.2.1 Recreating the View-Path

Based on the documentation of the original rendering environment it is essential to recreate a new view-path which can replicate the rendering in the original environment.

The new view-path is built depending on the digital object and the available preservation actions. In a migration setting some of the layers in the view-path are replaced, while in the emulation setting a new layer is introduced in the view-path as shown in Chapter 3. As emulation is usually done on a hardware level, the emulator used to create the setting has to support emulation of the hardware-components found in the original system. If the exact same hardware configuration cannot be rebuilt in software, this has to be documented and the effects of replacing various parts with alternative parts have to be considered when comparing the results.

Recreating the view-path in the emulated environments has to be done following the documentation of the original system to ensure that potential differences in the new environment are a result of the new rendering process and not of side-effects of a difference in the setting. One possiblity is to create a complete image of the view-path on the original system, and to use this in a virtual environment (e.g., floppy disc for home-computers containing the disc operating system and the viewer-application, hard-disc-image containing a pre-installed operating system with all the necessary drivers and applications).

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

5.2.2 Reapplying External Data

After re-deploying the object in a new environment the data is then used to verify that the rendering process still behaves correctly. The new rendering engine (e.g., a virtual environment) will supply the necessary external data that has been collected during the documentation stage. If the new rendering environment does not provide the possibility to “simulate” data on all its interfaces to the outside world, similarly to the capturing of the data external tools that e.g., capture network traffic and respond instead of a “real” web service with the captured data are an option for replacing this functionality.

The data captured from the original environment is used to verify the proper execution in the new environment, to make sure that no side-effects through the use of a different rendering environment exist. Once the digital object is verified

to render correctly with the captured data, it may have to be integrated in a distributed system again, to fulfill its original purpose. This means that the object along with all the external dependencies has to behave as it did in the original system. It is very likely that not all external dependencies (not-locally controlled data) will be available to the runtime environment in the future. In this case these services will have to be emulated (i.e., replaced by a service that provides the same functionality as the original service) or simulated (i.e., replaced by a service that reacts with valid data without replicating the original service's functionality).

An example for not locally controlled data would be a service that provides the current temperature on top of the Eiffel tower used in a piece of electronic art. For verification purposes we can capture requests to the service and responses, ensuring a deterministic execution as compared to the rendering in the original environment. However, once the digital object is deployed in its future environment we would expect to get proper answers from the service, and not pre-recorded data. To achieve this, the once existing service has to be replaced by a new service. If the service is emulated, it will be replaced by a service that also provides the current temperature. If the service is simulated, a valid value for a temperature is returned, but not necessarily the one that is actually the current temperature, as it could for example be derived from trained data (e.g., returning random higher values during day time and lower values during night time). Similarly, a service where the internal functionality is not exactly known would have to be replaced by a service that appears to behave as the original service did in that valid (but not necessarily correct) data is supplied by the simulated service.

The functionalities described in this section allow us to deal with all the external dependencies that a digital object can have. While the rendering of a digital object in its original environment is still available, a virtualized environment can be set up. The significant data influencing the rendering of the object in the new environment can be captured and stored in an archive along with the digital object. Once the digital object is re-deployed in a future environment, the captured external data influencing the rendering can be reapplied to the new environment. Using the same data we expect the object to behave deterministic and ideally provide the same data as in the original rendering.

5.2.3 Comparing Objects

Similarly to the external data that is captured during the execution of the rendering process, data created by the digital object and provided to the interfaces of the host system (e.g., the screen for rendering, audio signals, network data) can be captured for a comparison between different renderings. Also, data created for processing not available on the interfaces to the host system but only internally in a virtualized system can be captured from memory regions of the virtualized

system.

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

5.2.4 Identifying Levels of Comparing Rendered Forms

In Section 4.3.3 we discussed the different incarnations of a digital object in its original environment. For a comparison between the original environment and a new environment we have to identify the same incarnations in the new environment. Figure 4.1 shows the levels in an original environment on the left and the corresponding levels in a new environment on the right.

Descriptive Form

The descriptive form of a digital object is identical in the original environment and the new environment for emulation actions, but different for migration actions. As shown in Chapter 3, a comparison has to be done on a higher level as every migration results in a change in the view-path and thus potentially in the rendering of the object.

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

Rendered Form in Memory

If the emulation of the original system is processing the supplied information properly, then the rendered form in the memory area used by the new environment to store the rendered information has to be identical to the rendering in the same memory areas of the original environment. This is the first layer on which a comparison of rendering effects is possible. Screenshot or video memory dump applications on the original system and in the view-path will produce the same output after executing the same actions on both systems. The same applies to other forms of object output, i.e., audio, port communication, or file system interaction.

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

Rendered Form in Host System Memory

Using emulation as a preservation action, an additional virtual layer (the emulator) is introduced in the view path. This layer has to convert the rendered form from the virtual environment to a rendered form in the host system environment. In this step the layer has to, e.g., render the video output of the virtual system for display on the host system. To do so, for example the resolution of the image may have to be altered to fit either a resolution which can be displayed on the host system or by adjusting the image to the size of a window on the host system. In this step the resulting image will already be radically different than what it was on the original system, even if it looks similar for a human observer.

→ No direct comparison is possible at this layer as there is no similar information in the original view-path.

Rendered Form on the Output Interface

This signal exists on both the original system as well as in a new view-path. The resolution and type of the displaying unit are potentially different, but using capture devices recording the input directly from this source and transforming them to the same signals would be possible. An example is a video capture device that can take different kinds of inputs and displays them the same way on the host system running the capturing software.

In the case of encrypted communication (e.g., HDMI) the data would have to be captured in its encrypted form and provided by the new view-path similarly to what the original system provided under the same conditions when the digital object was originally rendered.

→ By recording the signal of the rendering in the new view-path and of the rendering in the original system the results can then be compared. The effects of the rendering device on the signal have to be taken into account when comparing the signals (e.g., reduction of frame rate, delay in processing, enhancing of the signal properties)

Rendered Form on Output Device

Capturing the signal directly as processed by an output device (e.g., using microphones to record processed sound or a video camera to capture a video stream from a monitor) the resulting representation in the recording device converts the output to the same resolution.

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

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

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

5.2.5 Extracting Properties from the Rendering Environment

Not only the resulting rendered form of the object can be compared, certain characteristics of the environment also can be measured. Ideally, these characteristics would have to be provided by the tool doing the rendering, which can be any or every element in the view-path. While none of the available rendering environments supports it at the moment, it would certainly make an automation of the process of comparing emulation environments and the original environments easier (or enable it at all). First the characteristics and properties of the original system and the digital object have to be described or measured. Then these same properties are extracted from the new environment. Ideally the characteristics should be described in a format usable for tools, which can compare these properties in an automated way. Using this approach the process of automating preservation planning for using renderings to compare objects can be similar to the one of automating preservation planning comparing properties of migrated objects currently.

Not only the system properties, but also screen shots or even a log of events happening on the system can be provided by the rendering environment. These properties can be extracted continuously over the emulation process for either a specified time or until an event occurs. Rendering properties are usually not single-dimension like properties of most migrated objects. An additional time dimension has to be considered. The frame rate of a rendered object can change over time, as more objects can result in fewer frames per second on the output device. Properties also can be extracted at one point in time after applying all the input events or after a certain amount of seconds, frames or CPU cycles. Table 5.1 shows possible characteristics that can be extracted, their usage and metrics. The characteristics are divided in the possible categories including the metrics used to measure and compare between the different environments. Obviously, the examples in the categories depend on the used environment, e.g., a frame rate only makes sense if the view-path contains a display device that is refreshed periodically (e.g., on a system acting as a web-service reacting to data input over a network interface and reacting with output on the network interface a frame rate will not give any information about the rendered object)

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

Analyzing the extracted events and their occurrence at specific times in the rendering process lets us define meaningful key characteristics of the rendering process:

Deterministic Rendering The most important characteristic of a rendering environment is that the rendering process must be deterministic. This means that

Table 5.1: Characteristics that can be extracted from rendering environments.

Category	**Example Characteristics**	**Metric**	**Usage**	**Extractable from levels in the view-path**
Input Events	Mouse clicks, mouse moves, key presses, requests from external services	Number of events during a time period	Check for deterministic behaviour, if the same number of events where triggered	every level in view-path
Input Data received	Mouse interface, network interface	Number of bytes during a time period	Check for deterministic behaviour, if the same amount of data was transferred from an I/O device	every level in view-path
Output Events	Files on storage unit accessed, Actors activated, Network addresses access	Number of events during a time period	Check for deterministic behaviour, if the same number of events where triggered	every level in view-path
Output Data sent	Data storage, network interface	Number of bytes during a time period	Check for deterministic behaviour, if the same amount of data was transferred to an I/O device	every level in view-path
Timing characteristics	Frame rate / CPU cycles	Frames per second / Cycles per second	Speed comparisons between the original and the new rendering environment	Every level that renders an object on screen (for frame rate) / hardware (for CPU cycles)

the virtual environment has to perform the same rendering process under the same inputs. This is of crucial importance to the evaluation, as only a deterministic process lets us compare different renderings of the same object and the results of it. Events occurring at the same point in the rendering process during different renderings of the same digital object let us determine if the rendering process is executed deterministically.

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

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

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

Frames per Second Determining the frames per second we can see if the

rendering process is executed slower than it would have on the original system. If the virtual environment is in fact not fast enough, we can see from the event-log which of the drawn frames took too long to calculate and what external events happened during the slow frames.

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

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

5.3 Steps for the Evaluation of Rendering Effects

To evaluate the degree to which the change of an original environment into a new view-path preserves the original characteristics of a digital object in comparison to documentation of its behavior, the concepts presented in this thesis lead to the following evaluation steps:

1. Describe the original environment
   The original system's hardware and software components have to be documented along with all their settings to allow the recreation in a changed view-path as described in Section 5.2.1. To reduce the complexity of the system it is recommended to eliminate all unnecessary secondary digital objects (software and hardware) in the view path (e.g., no virus scan software or use of standardized minimum OS configurations).

2. Determine external events that influence the object's behavior
   Only for objects with deterministic behavior it is possible to ensure that differences in rendering compared to the original environment are results of the changed view-path. To ensure deterministic behavior of the object it is necessary to investigate what external events influence its behavior and simulate those in the changed environment (e.g., set a random number gen-

erator to the same seed to produce the same sequence of random numbers).

3. Decide on what level to compare the digital object
   As the digital object is available in various rendered forms as shown in Sections 4.3.3 and 5.2.4, it is necessary to select the ones that are most suitable for the digital objects that have to be preserved and their desired form of representation. Depending on the original system only some of the various techniques may be technically possible (e.g., on early home computer hardware no operating system allows the execution of multiple processes, so no screenshot applications can be installed).

4. Recreate the view-path
   Next, a new view-path has to be configured to match the configuration of the original environment. The view-path of the digital artifact has to be recreated, ideally by using e.g., a hard-disc image configured the same way as on the original system. We recommend the use of a view-path as close to the original as possible to reduce the side-effects of different renderings by using different secondary objects.

5. Apply standardized input to both environments
   Depending on the digital object the most suitable way to apply automated input has to be selected. Then, the external events to the original object have to be recorded and applied to the new environment. The closer the input is done to the hardware level (as described in Section 4.2.1) the fewer side-effects on the rendering of the object it will have.

6. Extract significant properties
   Next, the significant properties of the rendered object have to be extracted both from the original as well as from the new environment as described in Sections 4.4 and 5.2.5. Depending on the digital object and if it reaches a target state this has to be done once or in a continuous way. Also, depending on the deterministic or non-deterministic nature of the digital object, only certain properties make sense to be extracted.

7. Compare the significant properties
   Finally, the significant properties that have been extracted automatically as well as those that were not measured automatically but evaluated manually

have to be compared, evaluated and documented. This may serve as input to a preservation planning process, or serve as evidence for the authenticity and faithfulness provided by the new view-path.

5.4 Preservation Workflow

The previously described framework and steps to compare two renderings of a digital object using different view-paths can be used in a preservation workflow like the one shown in Section 2.9. In this section we describe the different steps previously shown mapped to the stages in the workflow. We show what data needs to be captured and is stored along with the object in a digital archive to enable a curator to verify the rendering of a digital object once the object is retrieved from the archive and redeployed in the new environment.

5.4.1 Lifecycle of a Digital Object in a Preservation Workflow

The digital object at one point in its life-cycle is separated from its original environment and prepared for storage in an archive. At this point a comparison to the digital object still deployed in its original environment is performed to validate that all necessary external dependencies have been captured and the object's rendering in the new (virtual) environment is unchanged. The output in the defined steps of this rendering process for validation has to be recorded and stored in the archive along with the digital object. Once the digital object is extracted from the archive and re-deployed in a future environment, the external data influencing the rendering as shown in Section 4.2.1 is applied to the new environment. The output of the new rendering process is then captured at the same defined steps and compared to the output of the rendering before storage in the archive. Ensuring the deterministic execution of the digital object using the same locally controlled and not-locally controlled data as described in Section 4.2.1 in both executions we expect the same results of the rendering, thus allowing us to evaluate if the rendering process is executed correctly and get indications on how well the significant properties of the object are preserved.

The terms validation and verification are used in this context to describe two different evaluation steps of the rendering process. In the validation step we check that all the data that is needed to re-execute a process is actually captured after the process is virtualized, to make sure that all external dependencies are documented and stored along with the process. In the verification step we use the external data captured to verify, that the re-deployment of the digital object in the selected environment was successful, by making sure that the result of the rendering process is identical to results that have been recorded in the validation step. The data

that is captured during the validation step and that is used for evaluation in the verification step is referred to as "verification-data" in this thesis.

As described in Section 4.2.1, both locally controlled and not locally controlled external data are captured when validating the proper execution of a process (or other digital object) after being virtualized to make sure that it behaves as expected.

The captured data then needs to be stored in the archive along with the actual digital object. The information necessary to separate the digital object from its original environment, i.e. both the digital object itself (e.g., a workflow-definition for a workflow engine) as well as all the elements in the view-path (for virtualization) or information about how the object has to be interpreted (for a migrated form) need to be stored as well.

The verification-data stored with the digital object in the archive is used to verify the proper execution in the new environment as shown in Section 5.2.2, to make sure that no side-effects through the use of a different rendering environment exist.

5.4.2 Preservation Workflow Phases

In the previous sections we showed the different moments in the life-cycle of a digital object. These phases (plan, preserve, re-deploy) were developed into a workflow for the preservation of business processes [Strodl *et al.*, 2012]. While this process contains stages specific for business processes, the actions that have to be taken are similar for all digital objects and relevant for any evaluation of renderings of a digital object. Following the steps in the process as described below the rendering of any digital object can be captured and validated to be complete and an accurate representation of the digital object in its current rendering.

The steps for evaluating a rendering shown in Section 5.3 as numbered in Figure 5.1 are mapped to the different phases in the workflow as shown below.

Plan In the plan phase the context of a digital object has to be captured. This comprises both the technical (hardware, software), but specifically also the non-technical, such as an artist's intentions, the goals and motivation behind a scientific experiment, the setting in which the artwork or experiment was being performed, along with all legal implications of rendering the object at a later point in time, the necessary documentation and all other relevant data. The risk of not having the object available has to be assessed and managed. To accomplish the capturing, the digital object has to be analyzed along with all its external dependencies influencing the rendering of the digital object. Knowing these dependencies is necessary to create a deterministic rendering of the digital object, i.e. to make sure that the digital

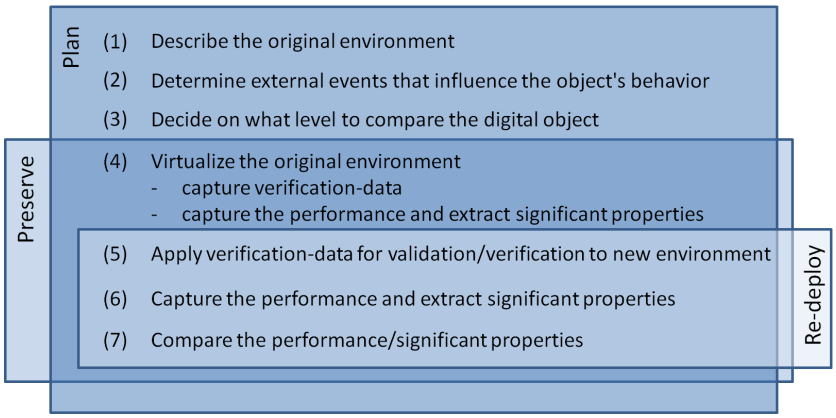


Figure 5.1: Different evaluation steps for evaluating dynamic renderings and their mapping to the preservation process phases.

object can be rendered identically under the same conditions, allowing us to do a comparison between the different renderings. To then verify the suitability of the preservation action for the digital object, the preservation action is carried out on the object and the result is compared to the original rendering, and a decision for one of the preservation candidates is taken. This first phase of an evaluation of the preservation action is similar to the actual validation and verification in the next two phases described below.

During the plan phase all of the steps in the evaluation workflow have to be performed. The original environment is described (1), not only the hardware and software components, but also external dependencies have to be documented along with the digital object itself. Part of the object properties are defined as significant properties, i.e., the properties we consider important to be present in a future rendering of the digital object. Thus we need to make sure these properties stay unchanged by the preservation action. All the described information on the object is mapped in the context model in [Mayer *et al.*, 2012b]. The technical dependencies of an object are ideally extracted automatically from the system.

Next, the external events influencing the rendering of the digital object are determined (2) by capturing external dependencies either directly from the original system (on already existing interfaces) or by abstracting the original environment in a virtual layer able to record interactions between the hosted system and the infrastructure it is embedded in. Any data exchanged on the interfaces between the abstracted and the host system is recorded as described in Section 4.2.1 either by a virtual environment or by external tools. Not only the input data is recorded, but also data rendered by the

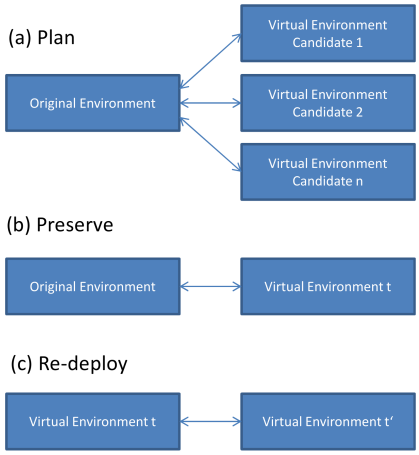


Figure 5.2: Environments used to extract data that is later used for the comparisons in different steps in the preservation process. (a) comparison of data between the original environment and different candidates for the preservation action, (b) comparison of data between the original environment and the virtualized environment for verification, (c) comparison of data between the virtual environment before storage at time t and the future virtual environment at time t'.

system. Depending on the relevant characteristics of the rendered data a suitable level according to Figure 4.1 is chosen for the extraction (3), either recorded by the virtual environment itself, or by external tools capturing the output of the virtual environment. Next, the original system is virtualized in all preservation planning candidates (4). The recorded input data is then applied to all candidates, ensuring a deterministic behavior of the digital object across the potential future environments (5). Data from all candidates is extracted (6) and compared to the extracted significant properties of the rendering in the original environment (7). Finally a decision is taken for one of the options for which the preservation planning was performed. Figure 5.2a illustrates that the original environment is compared with the different possible virtual environments at the time of planning.

During the initial planning phase, i.e., the first preservation planning performed for the object while the original environment is still available, the data captured is the "ground truth" for subsequent re-planning steps once the selected re-deploy environment gets obsolete. Subsequent re-planning is performed against the data captured as ground truth to not introduce errors in the data resulting in slight deviations of behavior of target environments from the original environment.

Preserve In the actual preserve phase, the digital object is first virtualized. In this step an additional layer is introduced in the view-path (e.g., a virtual machine or an emulator), allowing us to capture any communication between the object's environment and the outside world. This is for one incoming data that influences the rendering, e.g., data from a web service, user input, data from sensors or through a network protocol. But it is also data produced by the rendering in any form (e.g., on the screen as information for a user, data sent to actors or the network). We can thus validate the digital object at the time of preservation to make sure that all the necessary dependencies have been captured and will be stored along with the digital object. We also capture the data used for the validation and the output produced by the rendering process to compare this data in a later verification phase once the object is re-deployed. Data being captured includes the context as well as log files, any verification-data and properties of the rendering process.

In the preserve phase only the virtual environment chosen as suitable preservation action during the planning phase is considered. The steps for describing the original system have been performed and necessary decisions have been taken during the planning stage (1,2,3). In the preserve phase a more extensive capturing of verification-data is performed before storing the digital object in the archive, to validate that the complete context of the digital object necessary for evaluation has been captured. This includes capturing verification-data for all the external dependencies defined during the planning phase necessary both for verifying the digital object after being redeployed but also all dependencies necessary for actually using the object successfully in its future environment. While the data captured during the planning phase is partially reused, usually more extensive data is captured during the preserve phase. For example, additional use-cases that are more interactive and capture a broader spectrum of possible uses of the digital object are captured, more fore-ground data for verification is collected. Data used as input to the digital object in its original environment is recorded and reapplied to the object in the virtual environment (4,5), to ensure a deterministic execution and thus validate that all necessary data has been captured. Data is extracted in significant points of the rendering process (6) as defined in the planning stage to compare between the original and virtual environment as shown in Figure 5.2b (7) and thus evaluate that the environment is a valid environment for the digital object. Both data used for validating the execution as well as results of the rendering process are stored in the archive along with the digital object and all its dependencies and meta-data for later verification once the object is extracted from the archive.

Re-deploy At a later point in time the digital object is re-deployed in the new environ-

ment selected during the planning step. The digital object has to be integrated in a system where all the external dependencies necessary to render the digital object are provided in some form. An example for this would be a web service that existed when the object was originally in use, but might not exist anymore once it is re-deployed. Once the object has been re-deployed it is necessary to verify that the functionality is still intact. Thus we provide the verification-data that has been captured during the validation in the preserve phase to the new environment and capture the output of the digital object. If this is done for various use cases, and the data captured matches, we have strong evidence that the rendering of the digital object is unchanged compared to the original rendering at the time of preservation. Other data that will be provided to the new environment to ensure a deterministic rendering includes user input. To verify that the results of the rendering process are unchanged to the original rendering, log files and properties captured during the rendering in the new environment are compared to the results stored in the preserve phase.

For the re-deploy phase only the comparison steps of the evaluation workflow have to be performed. The digital object is re-deployed in the new environment. Then the data that has been captured during the execution before storage has to be applied to the new environment (5) and the significant data is extracted on the significant points (6) defined in the preservation planning phase. This data is then compared to the results captured in the original virtual environment during validation in the preserve phase as shown in Figure 5.2c (7). If the results are identical, the rendering in the new (future) environment has been verified against the original rendering by evaluating against a virtual environment that has been evaluated against the original.

It should be noted, though, that if the digital object is re-deployed in the actual environment after evaluation, only external services that are still available or can be replaced by other identical services guarantee proper functionality of the digital object beyond evaluation. If services are simulated, e.g., as in the Eiffel tower example in Section 4.2.1, the correct functionality of the digital object can only be guaranteed for values that are known to the simulation, any requests outside the simulation boundaries of the service will result in unpredictable rendering results. Within these limits, the object then is also open for new (unpredictable) user input that is different from the user input captured during the preserve-phase.

Using the evaluation workflow in the different preservation phases we thus can evaluate the rendering of a process in a new environment in all phases of

the preservation life-cycle of a digital object. By keeping the external data influencing the process renderings identical in the different environments we ensure a deterministic rendering of the object. Comparing the rendered data on the same level in both environments lets us compare the renderings and evaluate potential differences.

5.5 Summary

In this chapter we showed how the information described and captured as shown in Chapter 4 can be used to evaluate the rendering of the same digital object in a new rendering environment (i.e., in a different view-path). We first explained how the view-path is recreated and the captured external data influencing the rendering is reapplied to the new rendering. We showed how the results are compared and how the different levels shown in the previous chapter are used for this comparison.

The description of the digital artifact, its external dependencies and its view-path as shown in Chapter 4 along with the steps for evaluating the rendering of a digital object form the "Preservation Action Evaluation Framework" that allows us to compare renderings of a digital object using two different view-paths. The framework supports institutions in taking an informed decision on a rendering environment in a preservation planning process. As one example a video game museum can use the framework to select the best emulator for certain digital objects in its collection. In an industry setting the execution of (the software components of) a business process can be evaluated to ensure that the rendering is authentic with respect to the properties identified.

We then showed how the evaluation framework can be applied in a preservation workflow to make sure that the rendering of an object that is archived at one point in its life-cycle and redeployed at a later stage can be successfully verified in the new environment.

With the concepts provided in this and the previous chapter it would also be possible to automate the process of evaluating emulators to some extent. However, virtual environments today lack some of the features necessary for supporting automated evaluation, like the possibility to supply files for automated input or the extraction of significant properties of the environment and the digital object. In Chapter 6 we will show design requirements for virtual environments to allow standardization and to a certain extend automation of evaluation of renderings.

The following chapters will then show how the preservation workflow and the evaluation steps can be applied to real world examples to allow for proper evaluation of digital objects' renderings for archival storage and re-deployment for future use.

Chapter 6

Design Guidelines for Virtual Environments for Digital Preservation

6.1 Introduction

Based on the previous chapters showing a framework for evaluating a rendering environment and how to integrate it in a preservation workflow, this chapter will describe what functionality has to be provided by a rendering environments used for digital preservation to support an evaluation of the rendering of digital objects, both locally deterministic and rendering influenced by external events. We first show the basic requirements on rendering environments to be used in digital preservation. Then we show what the requirements to evaluating these environments are. Finally, we show functionality that is crucial for the usability of rendering environments for digital preservation applications.

6.2 Long Term Stability of Virtual Environments

Depending on the intended use of a rendering environment, the requirements during development are quite different. While a compromise between speed and correctness of emulation (e.g. timing between CPU and output components) usually can be taken into consideration for office applications, a much closer alignment between the internal components is necessary for video game emulators, with the most demanding requirement on correctness being real-time applications. Rendering environments thus have to be designed with the intended application in mind.

The emulator *Dioscuri* shown in Chapter 2 was the first emulator specifically

designed with digital preservation purposes in mind. The requirements that had to be met during its development as defined in [van der Hoeven *et al.*, 2007] were durability and flexibility. We will briefly revisit these below before expanding this set of requirements based on the lessons learned from the design of the evaluation framework.

6.2.1 Durable Virtual Environments

Virtual environments used in digital preservation have to be durable as they have to be sustainable for the long term. Developing the virtual environment as platform independent as possible allows for a use of the same virtual environment on different hardware and software platforms. Key requirements to developing virtual environments thus are:

Open Source Keeping a virtual environments source code accessible allows the archivist to have the emulator ported to new environments once the system the emulator is running on (either hardware or operating system) gets obsolete. Using closed source virtual environments poses the risk of unavailability once changes in the hardware / software environment of an institution become necessary.

Platform Independent Code If platform independent code is used for the development of a virtual environment, it is much more likely that the environment can be ported to a new underlying hardware or operating system once the original environment used gets obsolete. Examples for non-platform independent code include the use of machine language bound to a specific processing architecture, use of operating system dependent libraries and proprietary programming languages used for just once system.

Use of Virtual Machines as Intermediary Layers An additional virtual layer as described in Section 2.5.2 would allow virtual machines to be usable over a longer period of time, as only the virtual layer would have to be ported to a new platform, the virtual environment itself would not have to be adapted.

Most current emulators (e.g., video game system or mobile platform emulators) are generally developed for speed and immediate usage without keeping long term stability in mind. Dioscuri fulfills all the listed requirements by being open source, being developed in Java and using the Java Virtual Machine as an intermediary layer between the operating system and the emulator.

6.2.2 Flexible Virtual Environments

Being able to use one virtual environment for different guest systems prevents developing a new virtual environment whenever a single component in a system changes. The modular emulation concept shown in Section 2.5.2 allows the use of one virtual environment for different guest systems by exchanging different modules for the different components in a system through configuration. Ideally new components can be integrated using a plug-in system with specified interfaces.

The re-use of modules for components that already are in use in other projects (e.g., the same processors had been used in many different home computer system in the 1980s) also allows for reduced development time and is less likely to introduce emulation errors for these components.

Dioscuri was developed with this flexibility in mind to ensure several different computer environments can be emulated with minimal changes by using the modular concept.

6.3 Requirements for Evaluation

To support the workflow presented in Chapter 5 it is necessary to create requirements for virtual environments that are to be used for digital preservation purposes, as these requirements differ from those used for creating emulators today and the features they offer. The requirements which are necessary to support the evaluation of emulators are:

6.3.1 Recording and Replaying External Events

As previously shown the rendering of a digital object in a virtual environment has to be made deterministic. An object's rendering is non-deterministic if it depends on values that are external to the digital object itself, e.g., hardware events, user input. Thus, for evaluating a virtual environment it is necessary to apply the same values to the digital object for every repeated rendering cycle.

The values locally known and not locally known are shown in Section 4.2.1. The virtual environment has to be enabled to capture and re-apply all data available on external (i.e., not locally known) and internal hardware (i.e., locally known) interfaces, to enable a deterministic rendering. The following steps are thus necessary when developing the rendering environment:

- Identify all locally known and not-locally know interfaces for the system (e.g., by studying the hardware schematics of the system)

- Provide functionality to capture and store the data on these interfaces as well as the moment it was applied to the execution of the digital object
- Provide functionality to re-apply the data at the same time (either absolute, or relative to a variety of timing concepts such as absolute/elapsed time, processor cycles, or frames) during execution to a subsequent rendering of the digital object

Recording data and re-applying the same data to a later execution of the same digital object using the provided functionality thus allows for deterministic rendering of the digital object.

The automated replay of external events not only eases automated testing but also makes it possible to automate actions for users of objects in emulated environments (e.g., booting a system and starting an application automatically without special knowledge of the handling of the environment on the user-side).

6.3.2 Extraction of Significant Properties

Once the rendering of a digital object has been made deterministic, significant properties of the rendering process have to be extracted to compare different renderings of the same digital object. These can be categorizied into two different kinds of data, the rendered form of the digital object itself, and a log of the rendering process.

Rendered Forms

As shown in Figure 4.1 a digital object's rendering is available in different forms on the system rendering the object. Some of these forms exist inside the virtual environment in different memory regions of the virtual system:

- Random access memory (RAM) of the system
- Caches (e.g., processor cache, RAM cache)
- Hardware registers of output devices (e.g., video card memory, network buffer)

The available memory regions depend on the system being rebuilt in a virtual environment and can be found using the system's schematics and data sheets of components (also needed for developing the virtual environment).

The rendering environment also prepares renderings for the host system by providing data on the interface between the host system and the virtual system,

e.g., a rendered form of the virtual system's screen or the data passed along on the network interface. The rendering environment thus also has to be enabled to extract data of the various rendered forms of a digital object at specified times in the rendering process (similarly to how the data influencing the rendering has to be applied at certain times during the rendering).

The provided functionality is then used to compare the extracted data over different execution cycles of the same digital object. This comparison allows to observe the following characteristics:

- Extracting data from the memory regions of the virtual system, and comparing this data at the same points in time in a deterministic rendering, we can make sure, that the rendering process of the digital object in the virtual system is correct.
- Rendered data captured on the interface between host system and virtual system allows a comparison, if the virtual system correctly transforms the data calculated in the virtual system for use on the host system.

Logging of the Rendering Process

Some of the significant properties of the rendering that have to be captured are not properties of the object, but of the rendering process. These events occuring during the rendering process depend on the system being executed in the virtual environment. A list of categories and events are listed in Table 5.1.

The virtual environment has to provide functionality to log those events for a rendering of a digital object. A comparison of the timing of these characteristic figures allows then conclusions about the timing of the virtual system compared to the real system or a different virtual system. Virtual environments executing the process either too fast or too slow can thus be detected. Additionally, an analysis of accessed system resources can be used to detect if external resources are used by the rendering, making it not deterministic.

The actual events that have to be logged depend on the hardware properties of the system being virtualized. Besides the events, the following parameters should also be logged:

Executed CPU Cycles A system running in a virtual rendering environment usually runs at a different clock speed than the host system. Therefore, the number of executed cycles of the virtual machine's central processing unit is the main indicator of timing of when an event appears. This adds value for automated testing, as during an unsupervised test the emulator can be run without any speed limits, thus reducing the time needed for testing.

Elapsed Time As an additional time measurement the actual elapsed time on the host system since the rendering process started should be recorded in the log-file. This measurement gives us an indication of how the virtualized system's speed is perceived by a user of the virtual environment and may be used to normalize for timed events driven by a clock-based system rather than a timing based on CPU instructions executed.

Drawn Frame As an additional timing measurement for every event it should be recorded in which "frame" (unique consecutive image produced by the video hardware of the system) the event was registered, as the connection to a video refresh rate is an established timing instrument in the development of dynamic computer software. Similar time-stamps my be required for other output devices/ports that operate with an independent, internal timing or processor.

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

Additional Information Additional data for the recorded event can be included depending on the event logged, e.g. a key that has been pressed, a file that has been accessed etc.

The format the events are logged in can be either a structured XML file or a plain text file.

6.3.3 Timing Requirements on the Virtual System

Timing is a critical issue when executing systems in a virtual environment (especially for real-time systems). For evaluation, however, the issue is less critical. Even when dealing with real-time systems, the system is completely under control of the host system. The virtual environment software thus can capture and reapply external data needed to make a process deterministic at exactly the same moment in the rendering process. The time granularity of capturing and reapplying events has thus to take into account the guest system (or possible guest systems on the platform the virtual environment covers).

The evaluation process of a virtual environment can be completely decoupled from the absolute time the original rendering process was executed in. If the host system hardware is powerful enough, even an evaluation of a rendering in a much shorter time than the original rendering process would take in real time is possible. The events have to be applied only in the correct relative timing of the rendering process.

6.4 Data Exchange between Guest and Host System

One requirement that is not directly connected to the long term stability or testing of virtual environments but crucial for the usability of emulators for digital preservation is the exchange of data between the guest system and the host system. Virtual environments are enabled to use a data source to boot from, this is either a file that is loaded into memory on start of the rendering environment (emulating physical read-only-memory), or some kind of file-system based structure. But, currently, most emulators do not offer a possibility to copy content from the emulated environment to the host-environment during runtime. Phelps et al. see this as a major drawback for using emulators for digital preservation ([Phelps and Watry, 2005]).

Usually it is necessary to be able to access documents and reuse the contents of those documents in the host environment. The option of taking a capture of what is displayed on the screen as an image is usually not sufficient, as text or other objects that exist on the guest system would have to be transferred to the host system to re-use them properly.

Two different strategies of exchanging data between the guest system in a virtual environment and the host system running the virtual environment exist. The guest system can either be aware of running in a virtual environment, or be unaware of that fact.

6.4.1 Virtual Environment Unaware Guest System

In case of the guest system not being aware of running in a virtual environment, the virtual environment has to provide an option to capture data from a running system on different levels:

Hardware Data can be directly captured on a level of emulated hardware, i.e., ports of the system that store data, like text-based video modes of display hardware or memory dumps taken from the emulated memory. Additionally, data from the host system can also be provided on (emulated) hardware ports of a guest-system by injecting data directly into the keyboard buffer of the emulated system. While only primitive data types like text can be transferred using this method, the virtual environment does not have to be aware of the operating system being executed on the emulated hardware. The different possibilities of data that can be extracted depend on the hardware being emulated.

Operating System Another method of transferring data to the host system is extracting the data directly from the operating system of the guest system.

Knowledge of the guest system being executed as well as the inner structure of the guest operating system is necessary to extract data. One example would be to access the clipboard of the operating system and extract data that has been copied to the clipboard on the guest system. The format used to store the data in the clipboard as well as the location of the clipboard in the guest operating system have to be known by the virtual environment to extract data.

Data Carriers One method currently used is the transfer of data using data carriers that can be mounted in the virtual environment (e.g., disk drives or hard disks). In Section 2.5.2 remote access to emulation is described, where the object that will be used in the emulated environment is injected into a floppy disk image and mounted in the virtual environment. If the object is migrated in the virtual environment, the resulting migrated object is saved to the floppy disk and later extracted from the disk image on the host system. While the virtual environment not necessarily has to be aware of the guest operating system being executed, the data carrier formats have to be accessible by the guest operating system (e.g., if a Windows system is executed in the virtual environment, harddisks formated using a LINUX file system format will not be accessible, even if the carriers can be mounted in the virtual environment on a hardware level). When using data carriers the virtual environment not necessarily has to be able to inject/extract the data, as external tools can be used to prepare the disk image, and also to extract data stored on the data carrier by the guest system.

Remote Data Data can also be provided on other external interfaces than data carriers. If the guest system is running a web service that is providing data, the host system can connect to the guest system over the network interface and inject and extract data from the guest system. The guest system is unaware in this case if the data it provides is sent to a remote system on the network or a host system executing a virtual version of the guest system.

6.4.2 Virtual Environment Aware Guest System

Data can also be exchanged by the guest system and the host system if the guest system is aware that it is running in a virtual environment, and data used in the guest system has to be provided to a host system. These can either be done by installing tools in the guest system that communicate with a host system or by using an operating system that can be made aware of the fact that it is running in a virtual environment. In both cases the view-path of an object changes more than in a virtual environment unaware system, as by additionally introducing new

software or changing settings in the operating systems, the view-path is altered more severe than by only introducing an additional virtual layer.

Additional Tools

Tools being installed in a virtual environment allow the host operating system to access the clipboard of the guest operating system. These tools have to be provided by the virtual environment manufacturer for all operating systems running as guest operating systems, and have to be installed inside the guest system. One example for the virtualization software VMWare[1] are the "VM Ware Tools". For the open source Oracle VM VirtualBox[2] these tools are called "Guest Additions". Each of them provides these (or some of these) functionalities:

- File system folder sharing between host- and guest system
- Data exchange between host- and guest system
- Share the clipboard between host- and guest system
- New drivers to improve the performance of the guest system
- Access to statistics about the guest environment
- Time synchronization between host- and guest system
- Soft power operations to pause the guest operating system

To accomplish the data exchange, a communication channel between the tools installed in the guest system and virtual environment being executed on the host is established. This channel can be used by tools on both sides to exchange data.

Virtualization Aware Operating System

If an operating system is installed that is explicitely aware of being run in a virtual environment, this operating system has to explicitely be compiled to include the necessary components. The operating system makes direct system calls to the hypervisor (i.e., the virtual layer between the hardware and the guest system) and uses virtual registers provided by the virtual machine but not available on the physical hardware (i.e., processor registers). This technology called "Para-Virtualization" [Whitaker *et al.*, 2002] is mainly used to improve the performance of the guest system(s) on a physical hardware.

[1] VMWare – `http://www.vmware.com/`

[2] Oracle VM VirtualBox – `http://www.oracle.com/us/technologies/virtualization/virtualbox/overview/index.html`

As the guest system is aware of the virtualization, the necessary tools for exchanging data with the host system (and other functionalities listed in Section 6.4.2) are part of the operating system and do not have to be explicitely installed in the guest system.

6.5 Summary

By following the design guidelines for emulators for digital preservation purposes as proposed in this chapter standardization and to a certain extend automation of evaluation of emulators will be possible. The functionality for capturing and re-applying of data as well as extraction of properties has to be included in the virtual environments to allow for automated evaluation. The guidelines for long term stability should be followed to enable a dureable and flexible use of emulators in digital preservation. Extraction of data as described in Section 6.4 should be implemented to ease the use of virtual environments in digital preservation application, to raise acceptance of emulation as a viable alternative to migration.

No standard for data provided to virtual environments exists so far. It is necessary to develop a standard for handling of user input to use the same input definitions in the same format for all emulators that have to be evaluated. The same is true for extracted significant properties. As neither the set of properties nor a format to extract it to is specified yet, it is necessary to carry out future research in this field. Especially as this is a feature not yet supported by emulators a standard may be devised, that can be implemented by emulator-developers to get the properties in a unified format. One example of a characterization language that can be used is XC*L [Becker *et al.*, 2008b]. It is also necessary to develop a measurement framework that supports the extraction of properties from rendering environments and the automated comparison of rendering results.

The next chapter will show a case study on an obsolete system. An emulator for the system developed with the requirements presented in this chapter in mind will be presented and evaluated using the evaluation framework.

Chapter 7

Preserving an Obsolete System: The C7420

7.1 Introduction

In this chapter we will show how digital objects for an obsolete home computer system can be preserved on the different threat levels. After introducing the system we first show how data from original media is re-engineered and extracted (preservation on the physical layer). We present a tool that is not only able to extract the data but also to convert simple objects (e.g., images) to non-obsolete formats using migration (logical layer). Next, we show the development of an emulator that is used to preserve the more complex objects on this system (e.g., software) on the logical layer. Research shown in this chapter has been published in [Guttenbrunner *et al.*, 2011], [Guttenbrunner and Rauber, 2011], and [Guttenbrunner and Rauber, 2012a]

7.2 The C7420 Home Computer Module for the Philips Videopac+ G7400

For the case study we decided to use the Philips G7400[1]. Originally designed as a video game system with a keyboard, it can be extended to become a home computer with a Microsoft BASIC operating system using the C7420 expansion cartridge. This cartridge features three connector cables for data input, data output and a remote controlling signal used to start and stop the audio tape, if supported by the tape player.

[1]Philips G7400 on Wikipedia – `http://en.wikipedia.org/wiki/Philips_Videopac_\%2B_G7400`

The system was chosen as it is already very hard to find specimens in working condition, so there is an imminent threat of losing the data saved with this system permanently. For the purpose of this case study we also chose a system that had physical media that could be read by current hardware, in this case off-the-shelf audio recorders, as standard compact cassettes could be used for storage purposes.

7.2.1 The Philips Videopac+ G7400 Video Game Console System

In 1968 Ralph Baer created the prototype for the first home video game called Brown Box [Baer, 2005]. The American company Magnavox released the system to the public in 1972 as the "Magnavox Odyssey". The system used cartridges that did not store any information but interconnected different electronic parts to create the desired built-in games. Only black/white output was created and by applying different overlays for every game on the TV the impression of color was created.

In 1978 the successor to the Magnavox Odyssey, the Magnavox Odyssey^2, was sold in America [Herman, 2001]. In Europe the system was sold by Philips under the name "Videopac G7000" [Forster, 2009]. This system used an Intel 8048H CPU and the custom "VDC" (video display chip) Intel 8244 to display various different on-screen objects.

Magnavox also started to develop a successor to the Odyssey^2, the Odyssey^3. The system was equipped with a more powerful graphics chip then its predecessor but was made backwards compatible to the Odyssey^2. It was never sold to the public, even though some prototypes[2] were found by video game collectors on yard sales in the area of Magnavox' head quarters in Knoxville, TN, USA. In Europe the Videopac G7000 system was more successful than the Odyssey^2 in the US, so Philips released the Odyssey^3 under the "Videopac+" brand as the "Philips G7400" (shown in Figure 7.1) in Europe in 1983. The system was able to use all the cartridges for the original system, but also some additional cartridges only playable on the G7400 were released. As home computers got more popular during that time and the Philips Videopac systems were equipped with a keyboard all along, an additional cartridge that converted the system to a fully fledged home computer was released.

7.2.2 The Philips C7420 Home Computer Module

In 1983, shortly after the release of the Philips G7400 game system, Philips released the Home Computer cartridge as an add-on to convert their console system

[2] Odyssey^3 Prototypes – `http://www.dieterkoenig.at/ccc/po/s_po_o3.htm`

Figure 7.1: Philips Videopac+ G7400 game console system.

to a full fledged home computer. As the built-in 8048 processor was not powerful enough for this task, the system itself was used for input and output only and the computing was done mainly by a Zilog Z80 micro processor running at 3.754 Mhz and stored in an extra case connected to the main system (shown in Figure 7.2). The home computer module had 18 Kbytes ROM inside the cartridge. Microsoft Basic was used as a programming language for the home computer add-on and used up 8 Kbytes of these. 16 Kbyte RAM were also integrated in the module of which 14 Kbyte could be used for user programs.

To save and load programs to external storage, a microphone and a headphone connector were included which allowed the storage of data and programs utilizing home audio equipment and standard audio-tapes.

Only very few programs for the system were released commercially in printed form. Besides the manuals included with the cartridge, a book teaching how to program the system and including some example programs was released in France in 1984 [Bardon and de Merly, 1984] and a second book containing generic BASIC programs adapted to the system was released in Italy [Deconchat and Grandis, 1985].

On this system BASIC was used as the main programming language. Source code is a significant property of software and can be necessary to interpret the data stored by applications and is also necessary if software is migrated for preservation purposes [Matthews *et al.*, 2008]. As the system is used as a video game console as well, some of the programs are video games. This provides us with a situation where migration would be a possible solution to preserve some video games for the system [Guttenbrunner *et al.*, 2010a].

Figure 7.2: Philips Videopac+ C7420 Home computer cartridge: Cartridge that plugs into the system in front, connected to the main case that holds the additional CPU and memory in the back. The connectors for loading/saving data to an audio system (red, white and black cables for microphone, headphones and remote control) are attached to the main case.

7.3 Extracting Data From Obsolete Media and Migrating It to Non-Obsolete Formats

Audio tapes are magnetic tapes and are subject to various threats on the physical level as described in [Bhushan, 2000]. By converting the analog waveforms to digital waveforms and storing them as digital audio-files on current systems we can avert the immediate threat on the physical layer.

To prevent loss of data on a logical level it is necessary to re-engineer the encoding of digital bits in the analog audio signal. In [Ross and Gow, 1999] an experiment with a Sinclair Spectrum is described, where audio data was migrated to a corresponding binary stream, which could then be interpreted using an emulator of the real system.

However, to separate the digital objects from their original environment the bitstreams have to be interpreted in such a way as to extract the conceptual object from the logical bitstream. By extracting the content and saving it to a format which is not obsolete at the time of migration we can transform the data to a format that is accessible without the original hardware. No expert is needed to operate the original system as it is necessary with emulation as a preservation

Figure 7.3: Waveform of “Hello World” BASIC program (1: initial 6 kHz lead-in tone; 2: 256 x 0xFF as start of file-signature; 3: file header; 4: 128 x 0xFF as header/data separator; 5: data block)

strategy.

The essential elements or meta data of the digital object can then be added on ingest in an archival system.

7.3.1 Re-engineering the Waveform

Data on the system can be stored in various formats. The BASIC programming language variant that comes with the system supports saving program listings, screenshots, and storing and retrieving self-defined data (text strings and number arrays) using various forms of the “CSAVE” instruction.

In order to start re-engineering the storage encoding, the original machine’s output was connected to the input of a PC’s sound card. We started by writing some test programs on the original machine and recording the resulting audio files using Audacity[3]. One resulting waveform can be seen in Figure 7.3. By recording different test programs we were able to find out that there is always a 2.77 second lead-in frequency of a 6 kHz sine wave. The data block is stored in a 4.8 kHz sine wave encoding bit set (’1’) as a tone and bit cleared (’0’) as silence. Every byte is encoded as one start bit (tone), followed 8 data bits (storing least significant bits first) and 2.5 stop bits (silence) (Figure 7.4). The data is stored at a rate of 1200 bits per second. Every file consists of data-bytes in a structure of a file header and a data block as described in Table 7.1.

During our online-research we also found an active community[4] that is still using and also programming this system. One of its members, René van den Enden[5], had written small programs that allowed BASIC programs to be transferred between the original system and a PC. On request he provided us a copy of the

[3]Audacity – `http://audacity.sourceforge.net/`
[4]Videopac / Odyssey^2 Community Forum – `http://videopac.nl/forum/`
[5]Rene’s VIDEOPAC page – `http://home.kpn.nl/~rene_g7400/`

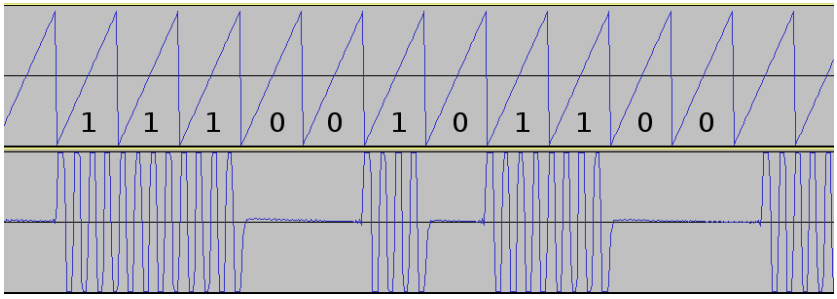


Figure 7.4: Representation of one byte in the waveform (1 start bit (1), 8 data bits (least significant first: 11010011b = D3h), 2.5 stop bits (0))

Table 7.1: File structure of the bitstream on the C7420 system.

No. of Bytes	bf Code	bf Contained Information
256	0xFF	<start of file>-signature
32		file-header
128	0xFF	separate header / data
<variable size>		data-block
10	0x00	<end of file>-signature

source code of his programs which confirmed part of our research regarding the format and provided more details we had not figured out at this point of our investigation.

7.3.2 Re-engineering File Formats

To understand the logical format of the data stored in the waveforms it was necessary to reengineer the various formats that are possible to store using the C7420. From the original user manual it became apparent that the C7420 is able to store the five different kinds of data shown in Table 7.2.

By writing small test programs storing the different formats and analyzing the resulting waveforms we were able to reengineer the bitstream of the different formats. The discovered logical formats of the bitstreams of the different file types

Table 7.2: Logical bitstream formats and corresponding command to save data on the C7420.

Logical bitstream format	**Command**
BASIC Program	CSAVE
Screenshot	CSAVES
Array	CSAVE*
String	CSAVEX
Memory Dump	CSAVEM

are described in detail in Appendix A.

7.3.3 Converting Waveform to Bitstream

In order to write a tool that is able to convert the waveform into usable data we had to develop a method of interpreting the waveform programmatically and detecting the various stages in the signal.

In our tests the signal was sampled as a 48 kHz, 16 bit, mono signal. As the C7420 outputs the signal at a rate of 1200 bits per second, we can calculate the number of samples per encoded bit (spb) using Equation 7.1.

$$spb = \frac{f}{bps} \tag{7.1}$$

where,
spb = samples per bit in the digitized audio stream
f = sample frequency of the waveform
bps = bits per second as output from the C7420

The signal output by the C7420 is a sine wave with a frequency of 4.8 kHz, so every bit is represented by 4 sine periods.

We implemented two different methods of interpreting the signal. Method 1 was taken from the sample programs we got from René van den Enden. For each sample we need to decide if it marks silence or signal. The algorithm scans the sample stream of the digitized waveform until an absolute value greater than half the maximum amplitude of the signal is found. High amplitude is interpreted as a signal and such as the start of a coded bit “1”. More samples are subsequently read and counted either as “signal” or “no signal”. If more than a pre-defined and adjustable threshold of “no signal” samples are found, it is assumed that the end of a coded “1” has been reached and a coded “0” starts. For a coded “1” bit to be properly recognized, half the number of samples over the duration of 4 sine waves has to be interpreted as “signal”. Figure 7.5 shows a sample waveform and the values counted as “signal” (marked on the horizontal axis as “1”) and “no signal” (marked as “0”).

While we were able to read the original signal output by the console system without errors using this method, we encountered the following problems when we tried to interpret the signal stored on audio tapes:

- Missing parts of a coded bit: As a certain threshold of “no signal” was defined as the beginning of a coded “0”, errors were encountered while interpreting the signal if a small part of the bit had been lost due to data loss on the audio tape.

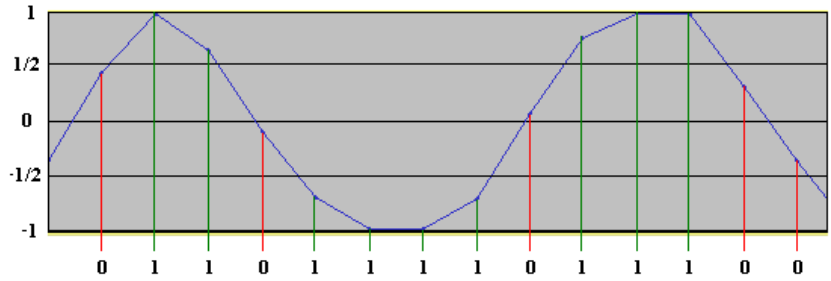


Figure 7.5: Interpretation of the wave signal using method 1. Vertical axis shows the strength of the amplitude, horizontal axes the parts of the sine wave interpreted as “signal” (1) or “no signal” (0).

- Noise in the signal: Most parts of the tapes contained noise which was incorrectly interpreted as signal.
- Differences in amplitude due to independent recordings on the same tape: While we were able to adjust the level of the input signal using the software for recording the signal from the audio source, changes in the signal over various parts of one tape made parts of the tape unreadable.

To reduce the sensitivity of the algorithm that converts the waveform into a bit stream we implemented a second method. For Method 2 we not only looked at single samples in the waveform but also calculated the sum of piecewise linear approximations of the amplitude, thus calculating the arc length of the sine wave for silence and signal first. The arc length of a curve for a bit that represents “1” is longer than the arc length of a curve for a bit that represents “0” due to the higher amplitude. To decide if a bit is set or cleared, a cut-off value between signal and silence wave arc length is used.

For every sample in the signal, the samples before it are used to calculate the arc length of the sine wave up to the sample. If the arc length is above the cut-off value then the sample is recognized as “1” otherwise it is recognized as “0”.

The algorithm is also able to adjust itself to changes in volume or noise, as the threshold which decides if a bit is set or cleared is constantly adjusted for every file in an input stream in parts of the signal which are known to be signal or silence. This way we are able to compensate for noise in the signal as well as for changes in volume. Missing parts of a signal bit have less influence in the recognition as not only the missing part, but also all parts before it are used to decide the state of the bit.

7.3.4 Migration Tool

Using the algorithms for converting the digitized waveform to a binary stream native to the system, together with the information we gathered about file formats, we developed a tool that is able to read the data contained in the waveform. Both described methods of interpreting the waveform were implemented.

The tool is written in JAVA. By using a virtual machine as a platform, the tool is independent from actual hardware for better long term stability. The tool and demo files can be found on the project homepage[6].

The following functions were implemented in the migration tool:

- Opening an audio stream and loading the contained files (either from an audio file (WAV or FLAC) or directly from an audio-in device)
- Opening files in the C7420-native file format (binary streams converted from WAV-file)
- Saving the opened audio stream as a C7420-native file format (binary stream)
- Saving data in a non-obsolete format (screenshots as PNG, basic-programs and arrays as text files, binary data as binary)
- Saving data as an audio stream (either to an audio file (WAV or FLAC) or directly onto the standard audio-out device)
- Opening and saving compressed Zip-archives containing a collection of migrated files
- Creating new files of the different formats in the application (including syntax highlighting for BASIC-programs)

All the data formats used by the C7420 as described in Section 7.3.2 are supported by the migration tool.

Every file is opened in a new tab inside the application in an editor that is linked to the file type. The information associated with the file and stored in the file header (native file name, address in memory to load to) can be edited as well. A screenshot of the migration tool can be seen in Figure 7.6.

[6]Home Computer Audio Migration – `http://www.ifs.tuwien.ac.at/dp/hc_audio_migration/`

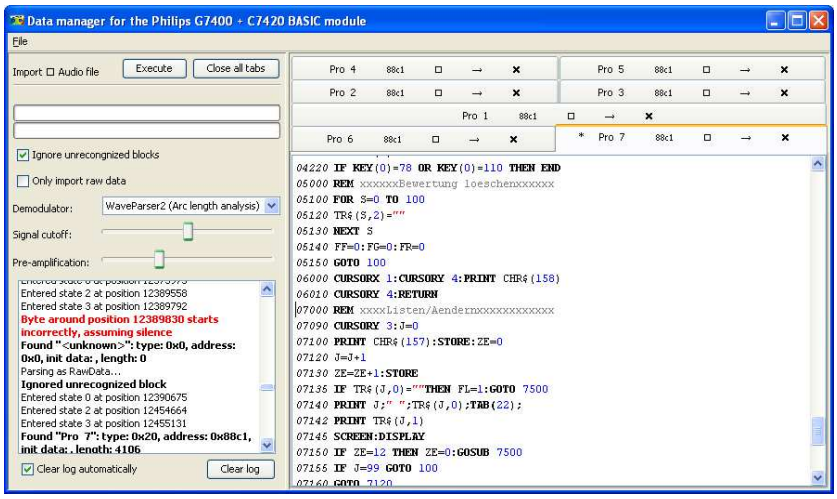


Figure 7.6: Screenshot of the migration tool GUI with 7 BASIC programs imported from a WAV-file recorded from an original tape. The import-log on the lower left shows events and errors during the import. Various import settings can be configured on the upper left and the imported programs are shown in tabs on the right.

7.3.5 Evaluating the Migration Tool

To evaluate the usability of the migration tool, we recorded different programs and other data as output from the original system. The data was recorded as a waveform using Audacity and then converted to user readable data in the migration tool using both implemented methods for converting the waveform. Then the data was loaded back into the original system, both from the recorded audio stream and from a stream reencoded using the migration tool.

The migration tool was able to restore all the data in the waveform as output from the original machine with both methods of converting the signal. The original stream outputted by the machine and the re-encoded stream from the migration tool, both gave the same results when the data was loaded back to the original machine. For a clean signal that was not distorted due to age, the migration tool perfectly read and wrote the data from and to the original machine.

Additionally, we acquired three audio-tapes created with the original system approximately 20 years ago from a private archive. Two of the tapes were standard Philips FE*I 60 normal position audio tapes as used for recording music while one was a C-10 computer cassette tape from manufacturer a11 specially manufactured for recording data (Figure 7.7). The source who recorded the tapes and the contents was not known before we started the experiments.

Figure 7.7: Tapes used for evaluation of migration tool, left upper corner C10 computer cassette, left lower and right Philips FE-I 60 normal position audio tapes.

We used a standard HIFI-system as an audio player and the software Audacity to record the audio streams as 44 KHz, 16 bit mono digital signal. The audio streams were saved as uncompressed WAV-files [Petermichl, 2009] containing the pulse code modulated (PCM) [Cattermole, 1969] raw audio data as bit stream. Two of the tapes had data recorded on both sides of the tape; one had data only on side A. Five WAV-files were obtained, one per side and per tape.

Each file was then loaded using the migration tool. The resulting migrated files were stored in a Zip-archive. For comparison the files were also loaded onto the original system.

A visual check for the characteristic waveform was done using Audacity to see how many files we expected the migration tool and the original system to find. A comparison between expected and loaded files can be found in Table 7.3 (first column for each method shows recognized files, second shows unrecognized files and third shows false positives).

Some files on the C10 tape were recognized by the original system but could not be loaded due to a “Bad label” error (with the suggestion to reposition the tape); while on the other 2 tapes no files were recognized at all. No files were correctly recognized using method 1. All but one file were recognized by method 2. Ten additional files recognized using method 2 were false positives that were easily detectable in the user interface and recognition could even be suppressed

Table 7.3: Comparison of expected (visual analysis of waveform) and loaded files (using different methods) on evaluated tapes containing C7420 data.

tape-side	visual	C7420		method 1		method 2		
C10-A	8	**5**	3	**0**	8	**7**	1	5
C10-B	2	**1**	1	**0**	2	**2**	0	0
Philips-1-A	6	**0**	6	**0**	6	**6**	0	3
Philips-2-A	6	**0**	6	**0**	6	**6**	0	2
Philips-2-B	1	**0**	1	**0**	1	**1**	0	0
Total	**23**	6	17	0	0	22	1	10

Table 7.4: Data with and without errors as recognized using the migration tool.

tape-side	loaded	not recognized or wrong file format	with errors	no errors
C10-A	7	0	4	3
C10-B	2	0	2	0
Philips-1-A	6	1	5	0
Philips-2-A	6	1	5	0
Philips-2-B	1	1	0	0
Total	22	3	16	3

by checking a check-box in the migration tool, allowing the user to ignore unrecognized blocks of data (usually noise on the tape).

The files that were recognized contained BASIC-programs. To check the files for validity we loaded them on the original system from tape and also loaded them on the original system as output from the migration tool.

From the 23 files on the three tapes no file was readable and usable on the C7420. All the 6 files that were recognized on the tapes were loaded with a “Bad file” error message and were not usable due to missing lines and incorrectly interpreted bytes. Thus the original system could not be used to load the data from the original tapes.

The results of recognized data in the loaded files using the migration tool can be seen in Table 7.4.

3 of the 22 files loaded could be recognized without errors. 16 files were loaded with various warnings in the migration tool, indicating that some bytes could not be recognized or were misidentified (e.g. wrong checksum, missing bits in bytes). Three files were not recognized in the correct format and shown as binary stream only.

Without manual pre-processing of the waveform or manual post-processing of the binary stream we were able to load 19 files opposed to just 6 files loaded by the original system.

The files loaded with errors were in various states of completeness. Some files were missing various lines at the end of the file. Other program lines were erroneous due to incorrectly identified bytes (an example can be seen in Figure 7.8). As the original data stored on the tapes was not available for comparison,

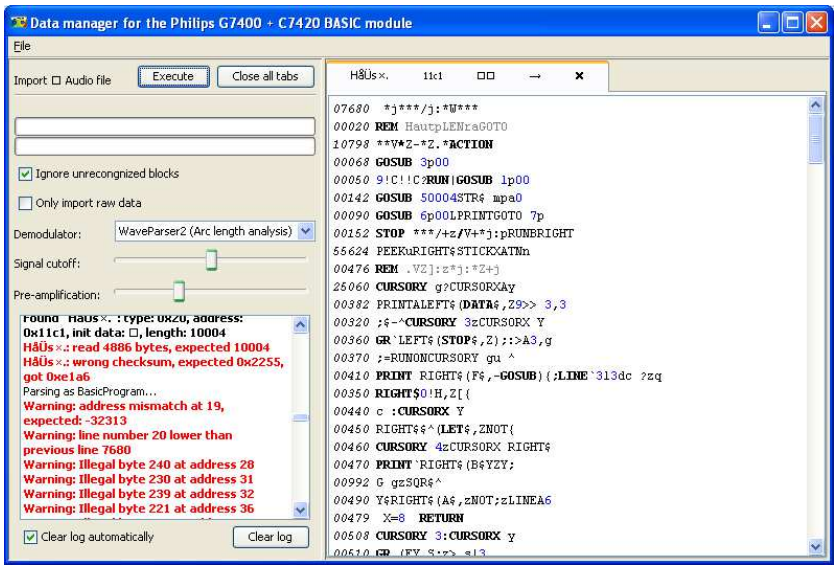


Figure 7.8: Screenshot of a BASIC-program imported with errors from a WAV-file. In the program listing on the right side incorrect arguments for commands and line numbers out of order can be found. The log on the left side shows error events that occurred during the import.

it is not possible to quantify the errors. But in general it seems that only single bytes were lost. As the data on the tapes consists of BASIC programs it should be possible to correct the errors by re-engineering the recovered program sources and thus reconstruct most of the data on the tapes.

7.3.6 Observations on the Migration Tasks

During the reengineering of the system and the creation of the migration tool different observations about these tasks and the applicability to other media types were made as follows.

Reengineering of the System

While digital archeology and reengineering systems is seen as a rather complex task, this case study shows that the reengineering of the format is easier while having access to the original system, as this way test data can be produced. Interpreting the number format without seeing the effects of the changed numbers on the original machine would have been a rather difficult task. It should also be noted that non-commercial “retro gaming” communities still working with the

system can be an excellent source not only for emulation, but also for data archeology on home computer systems.

The results of re-engineering the logical data can be used for other media as well. Re-engineering file formats can either be done using original systems or emulators, if available. Expert knowledge in handling the system has to be at hand to complete these tasks.

Evaluated Tapes

Examination of the data on tapes from a private archive showed that the data was no longer readable on the original machine. Using the migration tool we were able to retrieve most of the data with small errors. The evaluation also showed that it is necessary to act now and migrate data that was stored on magnetic tapes 20 years ago, as the lifetime of magnetic tapes is expected to be a maximum of 20 years (vanBogart:LifeExpectancy). Most of the data retrieved in the experiment could not be extracted without errors.

Improvement of Migration Results

As shown in the evaluation not all of the programs stored on the tapes were read without errors. A corrupted byte does not just change one letter in the command as every BASIC command is encoded in one byte, but results in a completely different command. Automatic correction of the files would thus be only possible by checking the BASIC programs for certain rules like commands which allow or enforce a certain number or types of arguments and point out inconsistencies to an expert doing the migration. He or she can then correct the results manually. Possible automatic support could also be offered by showing commands with e.g. a one bit difference in the encoded byte.

Media Refresh

Using the developed migration tool it is possible to refresh the media (audio cassettes) by reading and decoding the content, recoding it into a waveform and recording it to the tape again without using the original system.

Interpreting Results For Other Media Types

As audio tapes can be read using standard non-proprietary audio equipment, access to the physical layer of data is not in immediate danger. Other magnetic media like floppy discs cannot be read as easily. Even with floppy drives using the same media size ($8''$, $5\frac{1}{4}''$, $3\frac{1}{2}''$) access to data written on non-compatible computer systems is not possible.

7.3.7 Information Lost Due to Migration

Not only the information stored in the files which have been migrated has to be considered, but also how this information is rendered on the screen, e.g. for image formats. Thus it is necessary to characterize the potential objects that have to be migrated and look at their significant properties.

While most of the information that can be stored in files on the Philips Videopac+ G7400 can be migrated to non-obsolete formats used today, certain restrictions apply:

Screenshots The G7400 is able to render blinking information on screen. By choosing PNG as a non-obsolete (static) format, this dynamic information of the data is lost. Additionally it is possible to define custom characters using the BASIC language. As these are not stored in the waveform with the screenshot data, a complete program with the definition of the custom characters would have to be stored and preserved to keep the information available. To correctly render the characters again on a new system either the program containing the character definitions has to be analyzed and incorporate that information as well when migrating to a new format or the program has to be executed in an emulated environment to recreate the original rendering.

String Arrays As a string array contains only the addresses of the strings stored in it and the strings themselves are each stored in separate files, the interrelation between these files is lost without the logic of the program that establishes the link between them.

Program Results The data extracted from the tapes during evaluation was programs stored on the tapes. While the source code (i.e., the text of the BASIC program files) was successfully converted to a text file viewable on a non-obsolete system today, the programs can not actually be executed without the proper rendering environment. While the functionality of the programs can be reengineered from the source code, the rendering results and thus the conceptual layer the programs were creating on screen is lost using migration.

For the listed data formats no adequate non-obsolete formats are available to store the information necessary for the rendering process. To properly recreate the logical layer a rendering environment able to interpret the logical formats had to be created. With the possibility to save the data encoded in the waveform as a system native binary stream, files can be stored for usage in an emulator. As there were no emulators for the C7420 available, we show how an existing emulator for

the Philips Videopac+ G7400 was extended to include emulation for the C7420 home computer cartridge in Section 7.4.

7.4 Emulating the C7420 Rendering Environment

In this section we show how software extracted from obsolete media was preserved on a logical level by developing an emulator. We explain the reengineering work involved and the design decisions made in accordance to the guidelines shown in Section 6.2 as well as the options for data injection into and extraction from the emulated environment according to Section 6.4 to use the emulator in digital preservation applications. We also show how we implemented automated capturing of events and extraction of significant properties for automated evaluation as shown in Section 6.3.

In Section 7.3 we demonstrated how data and programs stored on audio tapes were extracted and the resulting audio files were transformed into digital objects using bitstream preservation and migration. The objects retrieved were mainly programs, requiring a rendering environment to execute these programs. This can either be done by migrating the programs to a current system, or by using a virtual environment for execution. As no emulator for the original system previously existed, we here show how we implemented one. We first describe the system in more detail and explain the reengineering of the view-path for the execution of programs on the original system. We show how an existing emulator for a video game system was expanded by emulation capabilities for the view-path of the home computer and how the different options for data exchange with the host environment were implemented on different levels in the view-path. We explain how differences in input and output formats and methods influence the development of an emulator and that, depending on the original system, the transfer of data between the emulated environment and the host environment enforces implicit migration of the data to become usable.

7.4.1 Program Execution on the Original System

For identifying the elements needed for the execution of software on the original system, we first have to determine the view-path of the software.

As shown in Section 2.2, in the most simple case the view-path of a digital object contains the digital object, the viewer used to render the object, the operating system to execute the viewer and the hardware to run the operating system. Depending on the digital object and the system used, some elements in the view-path can be missing. E.g. if the digital object is software, then usually the software is

Figure 7.9: Philips Videopac+ G7400 with plugged in Philips C7420 Home Computer cartridge.

running directly “on top” of the operating system. In the case of early computers, the software runs directly on the hardware without the use of an operating system.

To determine the view path on the original system, information about the hardware and the software running (e.g. BIOS) has to be collected. This information can be collected using different sources like the original circuit diagrams of the system and the cartridge, disassembled code of the Z80 BIOS and the terminal software, and last but not least valuable information found out by other members of a community still working actively with the original system (expert knowledge).

The original system used to execute the digital objects is a Philips Videopac+ G7400 video game system, which is expanded to a home computer using the Philips C7420 Home Computer cartridge (Figure 7.9). Using the C7420 cartridge, the video game system was extended by an extra processor (Zilog Z80), more memory (RAM) and an extra operating system (ROM) implementing the programming language Microsoft BASIC-80[7]. Figure 7.10 shows a block dia-

[7]Microsoft BASIC - Wikipedia: `http://en.wikipedia.org/wiki/Microsoft_BASIC`

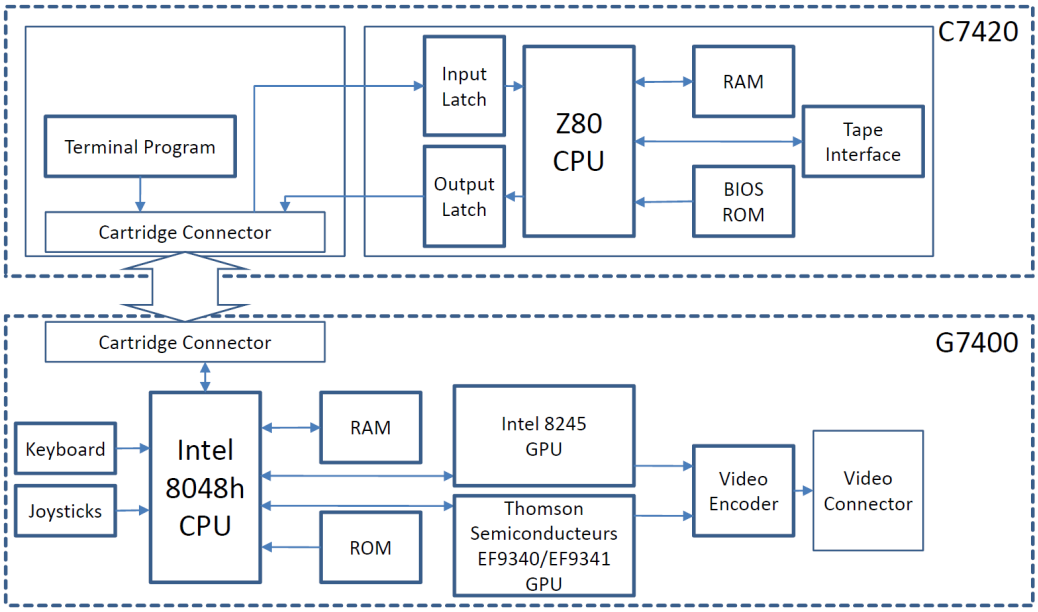


Figure 7.10: Block diagram of C7420 Home Computer cartridge and Philips Videopac+ G7400 system. Connection between cartridge and system is done using the cartridge connector. CPU - Central Processing Unit, GPU - Graphics Processing Unit, RAM - Random Access Memory, ROM - Read Only Memory.

gram of important parts of both the C7420 cartridge and the G7400 System.

The communication of the C7420 cartridge with the G7400 main system is done using a program running on the Intel 8048h processor inside the G7400 that serves as a terminal program by checking the system hardware for input (keyboard and joysticks) and also issues the commands for output sent from the C7420 cartridge to the relevant registers of the Intel 8245 VDC (Video Display Control) chip and the Thomson Semiconducteurs EF9340/EF9341 chip pair inside the G7400. These 3 chips produce all the visible and audible output of the system. Communication between the software running on the Z80 processor and the software running on the 8048h processor is managed by using two 8-bit registers that serve as a read and write latch. The Z80 processor writes information to the latch and then sets an input line on the 8048h processor. By checking the input line, the 8048h knows if information is available and proceeds reading the latch. For the other direction the 8048h writes to a different latch and sets a line that is connected to the Interrupt line of the Z80 processor, thus triggering an interrupt service routine on the Z80 that then can read the latch. Additionally the 8048h can send a RESET signal to the Z80 to reset the processor. The communication flow can be seen in Figure 7.11.

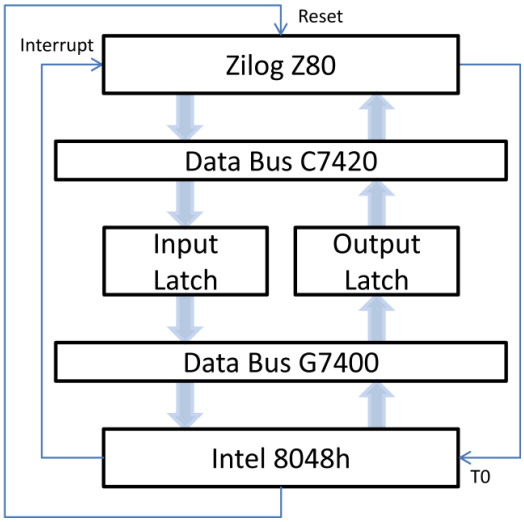


Figure 7.11: Communication flow between G7400 system and C7420 cartridge.

The BIOS, which is run on the Z80 processor, executes BASIC commands either entered by the user or stored as a program with line numbers. Results of operations are sent to the relevant registers on the G7400 using the described flow of communication. Commands accepting input are receiving the relevant input data from the G7400. Additionally to the data exchange with the G7400, the C7420 can store and retrieve data from an audio source connected directly to the cartridge using microphone / headphone plugs.

The resulting view-path for the G7400 system with C7420 cartridge can be seen in Figure 7.12. The digital object, in this case a BASIC program, is executed by the BASIC interpreter of the operating system. The BASIC interpreter is run on the Z80 CPU. Additionally, in this case a second branch of the view-path exists, which handles the input and output. In parallel to the operating system running on the Z80 processor, a terminal program for communication with the Z80 is run on the 8048h CPU, communicating input and output data between the G7400 system and the C7420 cartridge.

7.4.2 Implementing the view-path in an Emulator

As an emulator for the Philips G7400 had been in development for several years aleady and was known to work well, we decided not to start implementing a new emulator and add the C7420 emulation from scratch. The existing open source emulator O2EM[8] was used as a starting point for implementing the C7420 emu-

[8]O2EM - Sourceforge: `http://o2em.sourceforge.net/`

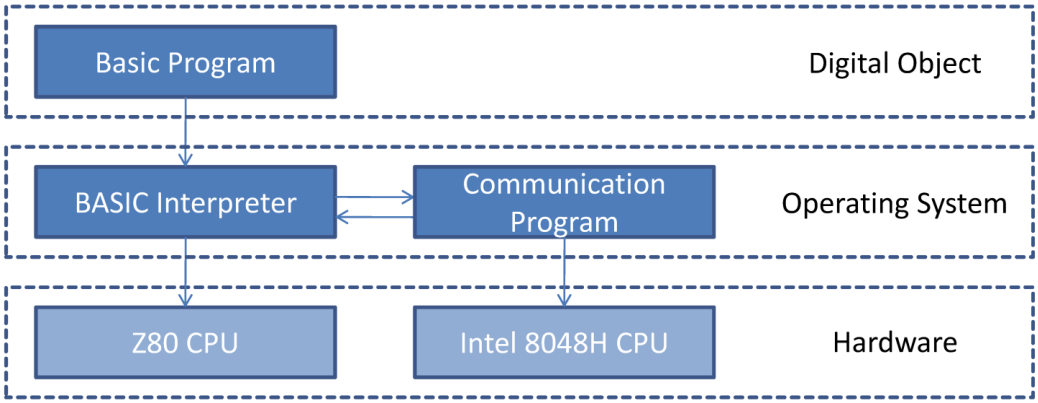


Figure 7.12: view-path for program execution on G7400+C7420.

lation. O2EM initially was written in 1997 as an emulator for the video game system Magnavox Odyssey[2], which is the American version of the Philips Videopac G7000. It was later modified for supporting the different screen timing of the European system as well as the additional functionality of the successor of the Philips G7000, the G7400. The emulator is written in the programming language C, and is thus portable to different systems with minimal changes, thus satisfying some (but not all) the durability guidelines shown in Section 6.2.1.

To integrate C7420 emulation into O2EM we first have to integrate emulation for the Z80 processor that would run side by side to the original 8048h emulation. An existing emulator of the Zilog Z80 [9] programmed by Marat Fayzullin is used. Using a separate module for emulating the Z80 processor component also follows the principle of modular emulation as described in Section 6.2.2. By using a Z80 processor emulation that is already proven to work in other emulators we can make sure, that the development effort on our side is reduced, minimizing also the risk of introducing erroneous emulation behavior by relying on existing, tested modules. Integration of the processor emulation consists basically of the following steps:

Z80 Memory Access and Interrupt After defining the 64 KByte memory of the C7420 as an array, the BIOS for the C7420 is loaded into the first 8 KBytes of the memory. Function prototypes provided by the Z80 emulator to access the memory are filled with code to access the memory (fetching instructions from the memory and reading and writing data). The prototype function checking for interrupts has to be adapted to signal an interrupt to the Z80 if the 8048h emulation sets the corresponding variable.

[9]Marat Fayzullin Emulation Resources – `http://fms.komkon.org/EMUL8/`

Z80 Input and Output Functions The Z80 processor has instructions for writing to output ports and also reading from them. These ports are used to access the latches for communication of the Z80 processor with the 8048h processor. The prototype functions are implemented to read from the latch defined at port 0xC0 and write to the latch defined at port 0xE0, as well as setting the T0 line of the 8048h.

I8048h Instructions, Input and Output Functions The 8048h instructions used to check the T0 line were previously only implemented to support a different kind of expansion for the G7400 system. These instructions have to be adapted in order to read the line that is set by the Z80 processor and reset it (to tell the Z80 processor that the 8048h recognized a written byte). Reading and writing to external memory also has to be adapted to read from the latch-register defined as external memory on address 0xE0 and write to the latch register defined as external memory on address 0xC0. Additionally, the write-function to the output ports of the 8048h has to be adapted, as pulling the lower two bits of Port 1 to low is supposed to reset the Z80 and pulling just Bit 1 of Port 1 to low signals an interrupt on the Z80.

Execution of Z80 cycles Finally, the emulation main loop has to be extended to include the execution of Z80 instructions. The 8048h processor is running at a clock rate of 0.394 MHz internally, while the Z80 processor is running at a 3.547 MHz clock rate, which makes it roughly execute 10 clock cycles for every 8048h clock cycle. Completely accurate cycle exact timing was not a necessity, as the communication between Z80 and 8048h is based on a handshake protocol, so one waits until the other provides the necessary data. The main execution loop sets the counter of cycles to execute to 10 and invokes the Z80 emulation.

To actually synchronize the emulation of the 8048h and the Z80 and implement the aforementioned steps, debug output of instructions of both processors is enabled and the log analyzed to find out exactly, which processor is doing what at a given point in time. By debugging through the assembler instructions of both processors, the handshaking can be established and the emulator starts up with the start screen of the C7420 Home Computer cartridge as shown in Figure 7.13.

7.4.3 Data Injection

After establishing the emulation of the C7420 Home Computer cartridge, the next step is to enter data into to the emulated environment emulating the user interfaces to the system. Three options for data input are available on the original

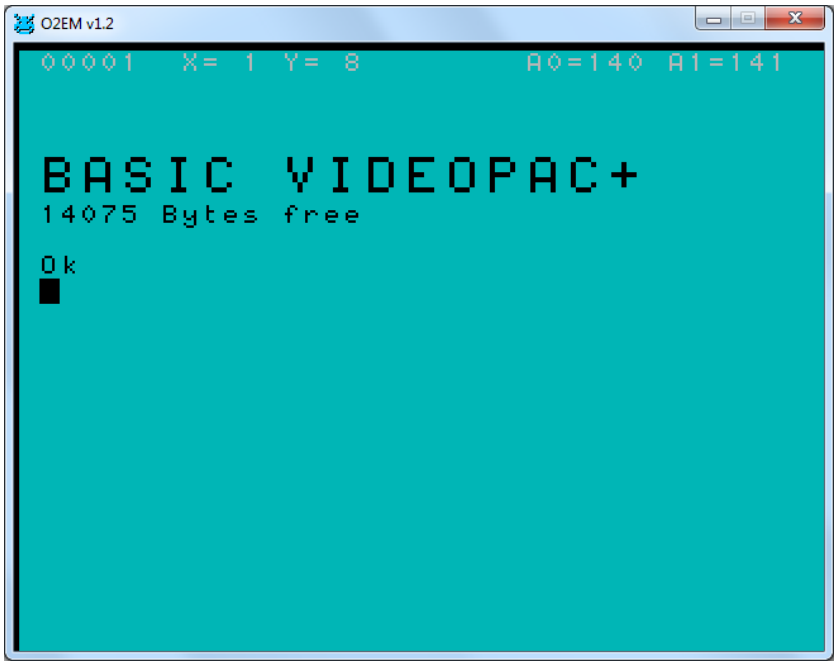


Figure 7.13: Start screen of C7420 Home Computer cartridge on O2EM emulator.

system. Below we describe these three options and the challenges they present for emulation.

Keyboard

An obvious method of data entry to the emulated environment is a key press. The previous implementation of the keyboard routine mapped every key on the original G7000 system keyboard to a key on a standard PC keyboard. This was sufficient for the currently emulated programs as the extra keys of the G7400 keyboard were not used in any of the supported programs.

In a first step we correct the keyboard routine to support the extra two rows of keys on the G7400's keyboard. This provides us with the possibility of mapping every key on the G7400 keyboard to a key on a modern keyboard. Unfortunately, the differences between current keyboards and the original G7400 keyboard are quite significant. As an example, a special key providing opening and closing brackets ("[" and "]") exists which is not directly to be found on a modern keyboard but only reached through key combinations. Additionally, various key combinations create different effects, for example the number sign ("#") is printed on the G7400 keyboard as a combination of the SHIFT key and the number "0", whereas most modern keyboards have a key assigned to it.

The BIOS of the G7400 checks the keys by going through every line of keys on the keyboard and reporting which key is pressed. Combinations of keys (e.g. SHIFT and a number) are recognized in the terminal software of the C7420 running on the 8048h processor. This software converts the pressed key to an ASCII encoded character depending on the combination of keys pressed and sends the ASCII code to the Z80 BIOS routine.

To improve the keyboard routine, we identify the following levels where it can be intercepted, the first two being an operating system level and the latter one being on the hardware level:

Z80 BIOS Directly inserting key-presses into the keyboard routine of the Z80. The Z80 reads the keys received from the terminal program running on the 8048h and writes them in a keyboard buffer. Keys read in ASCII-format from the host-keyboard can be directly written into the keyboard buffer (with the exception of characters that have a different code on the C7420 system). This would be a special routine only working for the C7420 BIOS, as it uses specifics otherwise not found on the system. It also would not be compatible with the current keyboard routine.

Communication interface Alternatively, keys can be written to the memory of the 8048h. As the keyboard routine in the terminal software already converts the key presses to ASCII, keys could be written as received from the keyboard functions. This method like the previous one would be a special implementation for the C7420. The existing hardware emulation would have to be disabled to not interfere with the other routine.

Hardware level Adapting the keyboard routine on the hardware emulation level offers the most compatibility not only for the C7420 Home Computer cartridge but for all other software developed for the G7400 system as well. Instead of the current implementation to have a one-to-one relationship between a key on the host keyboard and a key on the emulated hardware, with the flaws described above, a new routine could do a mapping of the actually entered character on the host system and set the appropriate keys in the emulated environment to simulate key-presses corresponding to the entered character.

The levels correspond to the description shown in Section 6.4. As the C7420 environment is a single process system, no additional tools can be installed in the environment, thus the data exchange has to be made with the guest system being unaware of the virtual environment it is executed in. Using the operating system level option, the emulator would have to be aware of the operating system being executed in the environment. Using the hardware level, no knowledge of the executed program is necessary for the keyboard routine.

We decided to extend the keyboard routine on the hardware level to reach the best compatibility for all programs running on the hardware. In a first step we create a mapping for all useful key-presses on the G7400 (e.g. combinations like “CONTROL”, “SHIFT” and a character don’t have any effect on the C7420, and even thought they could be theoretically read by replacing the G7400 BIOS routines by a self-written routine, the ergonomics of the membrane keyboard make it hard to press two keys at the same time). Next we replace the routine that reads the state of the mapped keys by a routine that first reads the ASCII Code of the entered character (considering modifier keys like Shift or Control), and sets the corresponding keys on the G7400 emulation using a “best guess” strategy to decide what the user actually wanted (e.g. entering “=” sign on the host keyboard (using a combination of different keys on the host keyboard) is mapped to pressing the “=” key on the G7400 keyboard. Likewise entering “;” on the host keyboard emulates a key press of the Clear key and the Shift key on the G7400 keyboard, which - in the original system - produced the semi-colon. Some of the keys had to be emulated by non-obvious combinations, for example one key for creating a character consisting of two dots, not available in ASCII or an modern keyboard, was simulated by entering “$”.

To test the validity of the keyboard routine, we wrote an assembler routine that reads out the pressed key and compared the results of the program on the real hardware and the emulator. Entering key-presses to the emulated C7420 environment also now creates the expected results. We also checked some samples of other software running on the emulator to make sure that the new keyboard routine did not break other software for the system.

Joysticks

The original system has two joysticks that are emulated by O2EM either using actual joysticks connected to the host environment or keyboard emulation for the joysticks. The polled data is provided to the emulated environment as soon as the BIOS of the G7400 tries to read the hardware ports. It is then handed over to the BIOS running on C7420 and can be read using the correspondent BASIC commands (e.g. STICK(0)). As the joysticks were already properly emulated by the original emulator on a hardware level, no additional actions had to be performed.

Files

Besides data injection through control devices, the C7420 supports the loading of files from an audio signal connected through a microphone jack during runtime. In this section we will show different possibilities of loading a file into memory, both on a hardware level and on an operating system level.

Hardware Emulation On a hardware emulation level, the component for reading data from the audio source, converting it to a digital signal and providing it on the input port of the Z80 is the most complex one. Basically, when the user tries to load a file using the 'CLOAD' command, the bits provided in the audio stream are decoded, assembled to a byte and written to the appropriate memory location. By reengineering the original BIOS routine of the 'CLOAD' command and based on the format as described in Appendix A we were able to create a routine that emulates that behavior of the original tape interface and provides the correct data in the correct timing to the CPU. The original tape was simulated by providing a directory in which the different files are stored. Using 'CLOAD' without a filename loads the file first written into the directory, subsequent calls of 'CLOAD' load the next file respectively. Using 'CLOAD' with a filename loads the file with the specified filename. 'CLOAD' supports loading of every file type supported by the C7420, i.e. BASIC programs, screenshots, data, and memory dumps.

Direct Writing to Memory An alternative to the aforementioned method of hardware emulation is to load a file into memory and directly write the loaded bytes into the correct memory locations on an operating system level. For this purpose the behavior of the original 'CLOAD' has to be reengineered even more to find out what all memory positions are affected (e.g. counter for free memory). Using this method we implement a special key that presents the user with a file-browser-dialog to select a file. Only BASIC programs can be stored using the direct memory method.

Both of the aforementioned methods result in the same memory structure when loading a file, with writing directly into memory being much faster (as the file is instantly loaded) whereas the hardware emulation preserves the original timing and thus needs a few minutes for programs with more than 100 lines. Using the hardware emulation it is possible to have programs load and save data from within the guest system using the original BIOS functions.

The data loaded from the tape interface is basically in the exact same format as written into memory (with the addition of leading and trailing bytes and some start- and stop-bits to separate bytes). To provide better support for using the emulator as a cross-programming-tool, we also implement implicit migration of BASIC files in text format. Loading a text file containing human readable BASIC source code is automatically detected and migrated back to the original binary format with encoded line numbers and encoded BASIC commands, so it can be used again in the original environment, the C7420.

7.4.4 Data Extraction for Application Use

While data injection is an important issue to execute and interact with software in the emulated environment, for most digital preservation applications it is also necessary to extract data from the emulated environment. Especially if emulation is used to access data stored in its original format and the data has to be used in the host environment, methods of copying data to one's current environment have to be provided as shown in Section 6.4. The methods for data extraction we implemented in the emulator are listed below.

Files

Using an emulator to modify data stored in an obsolete format makes it necessary to be able to save previously loaded files again. Again, two different methods are implemented:

Hardware Emulation The BASIC command "CSAVE" for saving data is implemented analogue to the command for loading files. We again have to reengineer the format by examining the code of the BIOS written in Z80 machine language to observe, what data is written to the output interface. The data stored by the BIOS is written to an array and saved under the filename given with the command. "CSAVE" works for all possible variations, saving programs, data, screenshots and memory dumps.

Direct Read From Memory on Operating System Level As with "CLOAD" a function to directly write a BASIC program from the operating system to disk is provided. As the format of storing BASIC programs in the operating system memory area of the C7420 was analyzed for creating the other file functions, it was also possible to create a function to provide a dialog to the user to ask for a filename and directly dump the memory in the correct format to a file.

As with "CLOAD" the resulting file is the same in both cases, with the hardware emulation being compatible to all formats and the direct read from memory version being easier to use without expert knowledge and being considerably faster. The choice of type of BASIC file (either in text format for easy readability or in binary format as originally created by the system) can be specified as a command line option for the emulator.

Clipboard

One feature hardly present in emulators today but crucial for their use for digital preservation purposes is the possibility to extract rendered text in machine-

readable form as separated characters from the emulated environment for use in the host environment. As the original environment in the C7420 does not support marking regions of text on the screen, and putting it in an internal clipboard, we decided to implement a function that copies the whole screen content as characters into the clipboard of the host system, so the text can be pasted into any application. Two different hook points for extracting data from the C7420 are possible:

Operating System Level: Extraction from C7420 screen buffer The operating system of the C7420 Home Computer cartridge holds an internal representation of the screen buffer for manipulation through the Z80 in the Z80 memory area (RAM). Extracting the characters from there would be possible by reengineering the memory location the screen data is saved at, as well as the format it is saved in. This would be the preferred option if the data was not rendered in the hardware chip as text on the screen.

Hardware Level: Extraction from emulator screen buffer The G7400 uses a teletext type of display chip for rendering graphics of the C7420. Thus a representation of the screen data (the characters) has to be held in the video screen buffer for rendering the image. By extracting data from the video screen buffer we not only create the possibility of copying data from the C7420 cartridge but also from all other software for the G7400 using the video chip to render data.

We decided to go with the more generic version and extract the data directly from the video memory of the emulator. Depending on the host operating system different routines for copying data to the clipboard have to be implemented. The data that is extracted is in ASCII format, so we can directly use it for copying it to the clipboard. The video chip is able to apply certain special effects on the characters (e.g. double size, blinking characters, underlined characters). As we need to get a text representation of the data for later usage in other applications we decided to ignore the format and just copy the actual characters to the clipboard. Not all the characters have the same code representation as in current ASCII format table, so a conversion for certain characters is performed while copying the data. The list of characters and the representation in ASCII format and C7420 internal format can be found in Appendix B.

Screenshots

Screenshots of the emulated environment can be used to include renderings in the virtual environment as images in documents on the host system. Different built-in features and external tools allow for extraction of rendered screens of the virtual environment:

In the Emulated Environment Using the screenshot feature of the C7420 (the 'CSAVES' BASIC command) the screenshot can be saved to a file and converted to a non-obsolete format using the tool we developed in [Guttenbrunner *et al.*, 2011].

Inside the Emulator The emulator O2EM has a built-in feature that allows saving screenshots of the rendered environment. Using this feature it is possible to manually save screenshots at certain points in the emulation.

From the Host Environment Using a screenshot tool inside the host environment automatic screenshots at different time points can be taken as well as a video of the emulation.

All of the described data extracted from the virtual environment can also be used to evaluate emulation accuracy, e.g., to compare emulation results with the original environment. A more detailed description of extracting data for evaluation purposes will be shown in Section 7.5.

7.5 Implementing Evaluation Functionality

As shown in Section 6, to automatically extract data of a continuous rendering process, the virtual environment has to support the extraction of properties of these process. To implement the extraction of data in O2EM, we have to take a look at the different levels inside emulated system on which the rendered forms of a digital object exist. Figure 7.14 shows the different levels on which an image is rendered inside the view-path of the C7420 Home Computer cartridge in conjunction with the G7400 system.

In detail the levels on which we can compare the rendering results are:

Z80 Memory The BIOS running on the Z80 has an internal representation of the screen memory that can be extracted using the screenshot feature 'CSAVES'. Doing this on the original system and on the emulated system, we receive two files which can directly be compared. If the files are identical, then the emulation of the Z80 CPU is correct (for the rendering of the test digital object). Yet, we cannot ascertain, that the actual rendering as provided by the emulator matches the rendering of the original system.

Video Chip Memory Another representation of the rendered object exists in the Memory of the video chip. This memory region is emulated in the emulator and can be read out. Unfortunately it cannot be read on the original system without directly reading the signals from the hardware and decoding them accordingly.

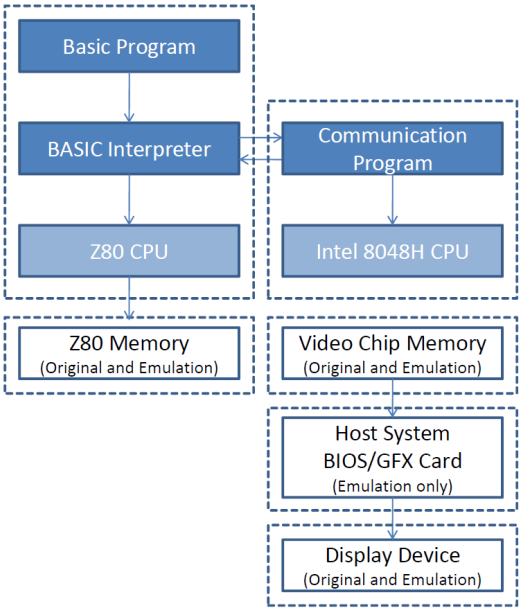


Figure 7.14: Different renderings in the view path of the C7420 Home Computer cartridge.

Host System BIOS The emulator renders the image stored in the video chip registers. The image is rendered and saved either in the Host system representation of the screen content or directly in the video card memory. Obviously this representation of the rendering exists only in the emulated rendering environment. Using this representation (basically creating a screenshot of the emulator's output) we can compare different rendering environments running on a host system (e.g. emulator of architecture level, high level emulator). In [Guttenbrunner *et al.*, 2010b] we demonstrate how the rendering results of different rendering environments can be compared by using the characterization language XCL as described in [Becker *et al.*, 2008c] for objectively comparing the significant properties of two screenshots.

Display Device Finally, a comparison on the level of the display device (comparing the output of the original system on a display device with the output of the emulator on a different or even the same output device) can be performed. This comparison is usually done manually and subjectively by the human preservation planner.

Not only the level of extraction of an image for comparison is relevant, also the time line is important. Usually, especially with interactive and dynamic software, we are not only interested in a screenshot at a certain point in time, but either

a series of screenshots or a continuous extraction of a video stream, which also allows the comparison of factors like timeliness and synchronicity, e.g. with sound output, compared to the original.

While the emulator supported already the extraction of screenshots (activated by pressing a key), a continuous extraction of images or extraction of images after a certain amount of elapsed time or executed machine cycles was not supported.

In Section 7.4 we showed how we implemented features in the emulator to make it usable for digital preservation purposes from a user's point of view (e.g. data exchange between the emulated and the host system). To actually be able to use an emulator in a digital archive, however, we need the possibility to evaluate the rendering process of digital objects more objectively and in an automated way. Based on the framework shown in Chapters 4 and 5, as well as the design guidelines derived from it shown in Chapter 6 we decided to implement the following features to aid automated evaluation:

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

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

Memory Dumps We also implemented a feature to trigger memory dumps of the different memory regions in the system, including the hardware registers of the Intel 8245 GPU shown in Figure 7.10. This allows us to not only rely on screenshots of the emulator or files saved in the home-computer-mode as a way to extract data from the rendering process. This corresponds to the level of the rendered form in memory in Figure 4.1.

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

7.5.1 Recording of Events

As described in Section 6.3.2, information about events occurring on the virtualized system have to be logged to draw conclusions on the rendering process of

the rendered digital object. The timing parameters as well as information about the events have been implemented. When starting the emulator the event-log file that should be created can be specified as an extra parameter. To easily import the resulting file in spreadsheet applications for further processing and analysis, as well as for processing speed reasons we decided to use a comma separated value (CSV) format escaping commas that form part of the input in the log.

We included the following different types of events for the system emulated in the emulator based on its hardware properties:

Operating the Environment

To be able to evaluate the rendering process reliably, we have to make sure that the rendering is always exactly the same under the same conditions applied to the rendering environment, i.e. the emulator is deterministic in its behavior as described in Section 4.2.1. For any object rendered in the environment relying on external input to the rendering environment (e.g. user input, network activity, access to files on the host system) the type of input as well as the actual input data have to be stored to be able to provide the same data on a re-run for evaluation purposes.

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

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

Extraction of Data

As a basis for comparing the results of the emulation process, it is necessary to extract not only the occurrence of an event but the actual data that is read or written as a result of the rendering. In Figure 4.1 we showed different levels on which a rendered object exists during the rendering process. From inside the emulator we have access to two different forms of rendered information: the form in the (emulated) memory of the system (e.g. hardware registers of the multimedia

processor, usually triggering an output on the original system) as well as the form that is already translated to the host system (e.g. a rendered screen based on hardware registers of the emulated system's video hardware).

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

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

Internal Events

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

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

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

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

The list of implemented events as well as the data recorded can be found in Appendix C along with an example log file.

7.5.2 Automated Execution

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

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

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

In detail the following actions where implemented:

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

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

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

By first recording external events and later applying the event-log as a command file for a new version of the emulator (or even a different emulator) it is possible to extract key characteristics of the rendering process as shown in Sec-

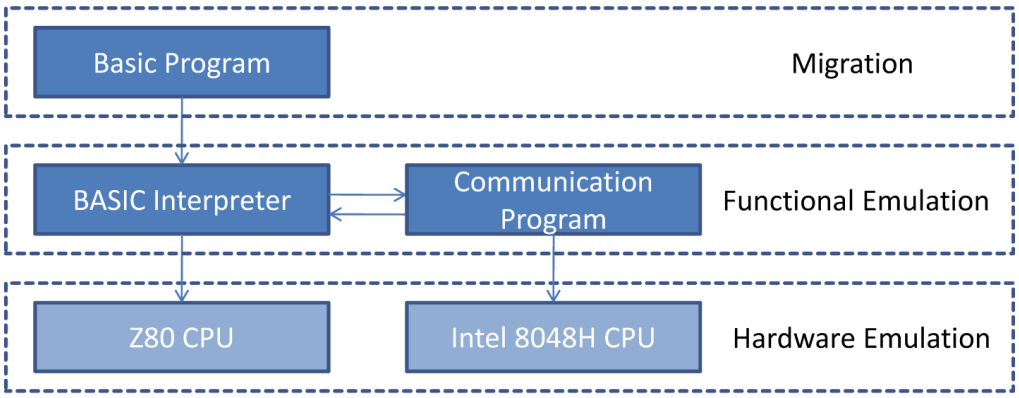


Figure 7.15: Preservation actions for different layers of view-path.

tion 5.2.5 and rendered forms of the digital object. This validation data can be used to evaluate the performance and accuracy of the emulator. If the resulting data extracted at significant points in the rendering process is identical, we have a strong indication that the rendering process is unchanged.

7.6 Discussion of Alternative Preservation Actions for the Philips Videopac System

In the previous sections we showed a tool to migrate static objects originally rendered on the Philips Videopac C7420 system. We also showed how dynamic objects (i.e., software) can be rendered using emulation of the original hardware, using the initial view-path used on the original system to render the objects. But executing programs using emulation on a hardware level is only one of the different alternatives that can be used for preserving software. Figure 7.15 shows the different levels in the execution view-path of the C7420 and also lists preservation action strategies for each of the levels.

7.6.1 Hardware Level

On the hardware level the emulator that was implemented can be used to preserve the system's behavior and thus create a rendering environment where the original operating system software (BIOS) can be used to execute the programs. As shown before, the reengineering effort necessary to implement an emulator is quite high, even though this method is the most accurate one, as every technical aspect of the system is replicated in the emulator.

7.6.2 Functional Level

Creating an emulator for the BASIC-programs not on a hardware level but on a functional level would require to implement an interpreter for the BASIC-code, that emulates the functions of the original BASIC-commands. Instead of executing the underlying Z80 machine language code in the BIOS if, e.g., a "PRINT" command is executed, the interpreter would emulate the behavior of the command, i.e. printing characters on the screen. Data extraction and injection is obviously much less complex, as the rendering environment can be directly manipulated and the behavior of each command is under control of the rendering environment. As the functional level is a higher level than the hardware level, only the functionality of the C7420 basic interpreter would be emulated, while the emulation on the hardware level allows other software for the Philips Videopac game system to be rendered as well.

7.6.3 Source Code Migration

A completely different strategy than emulating the system on a hardware level or emulating the commands on a functional level is the migration of the BASIC-programs to a non-obsolete programming language. Running a parser over the programs and migrating every command to a representation in a non-obsolete programming language allows us to create stand-alone versions of the programs that can be run without the need of an emulator program. While some of the commands would be quite easy to migrate (e.g. mathematical operations), others would involve more complex implementations (e.g., setting a different screen mode, displaying characters on the screen). Another obstacle to overcome in the special case of the C7420 is the flow of program execution, if the target language is a structured programming language instead of an unstructured one that is line-based like the used Microsoft BASIC-80 language. Jumps in the program between line numbers (and even to calculated line numbers stored in variables) have to be converted to different types of control flow statements (e.g. loops or choices). The principal possibility of this conversion has already been shown in [Ashcroft and Manna, 1979].

7.7 Summary

In this chapter we showed how we created tools for the preservation of an obsolete system on the physical and logical layer. We first presented the reengineering of the physical layer, and created a tool that allowed us to extract the bitstream of the different file formats of the system without using the original system but off-the-

shelf audio equipment. Next, the bitstream was migrated to non-obsolete formats, allowing us to render the logical layer on today's infrastructure. Finally, for all the data formats that could not be properly migrated, we created an emulator that allowed us to render the bitstream extracted from the physical media in a virtual environment, thus recreating the logical layer even for software for the original system.

One important lesson learned while implementing the emulator was that the input and output routines will most likely have to be adapted at the time of dissemination of archived data. A change in layout of keyboards used between archiving the emulator and the data to be rendered will already enforce a change in the keyboard routines of the emulator. If the method of entering data changes from keyboard to something else (which is not an unlikely scenario given a time frame of 50 to 100 years) the mapping of data input has to be completely adapted. Similarly, the data extraction from the emulated environment in the shown example already enforced a change in certain character codes. Given a longer time frame between archival and reuse of the archived emulator, these kind of adaptions are even more likely to be necessary, even if the environment for the emulator (e.g. an emulation virtual machine as described in Section 2.5.2 keeps the emulator executable.

Following the guidelines shown in Chapter 6 we also implemented functionality to support the evaluation of the emulator. We first introduced the event-log of the rendering process with different properties that allow us to re-run a rendering in the same environment and potentially also in different ones. We showed the different kinds of events that have to be recorded and that have been implemented depending on the original system. The different types of external data that can influence the rendering process have been explained as well as the different types of data that can be exported from the rendering environment for a comparison of different rendering processes. We then explained how the event-log can be used to automate the process of applying the same input data to the emulator to ensure a deterministic rendering of the digital object.

In the next chapter we will show how this functionality was used to evaluate the rendering of digital objects in O2EM. Finally, other preservation action alternatives for preserving software for the system were briefly discussed.

Chapter 8

Evaluation Case Studies

8.1 Introduction

In the Chapters 4 and 5 we showed the Preservation Action Evaluation Framework and how it can be integrated in a preservation workflow. Support for the framework was implemented in an emulator in Chapter 7, following the guidelines outlined in Chapter 6. We first show an evaluation of the implemented emulator using the implemented functionality supporting automated evaluation. Then, we apply the framework to more complex examples, namely a music classification process as an example for a scientific workflow, and a digital artwork rendered originally in a now obsolete hardware and software environment.

Some of the research shown in this chapter has been published in [Guttenbrunner and Rauber, 2012a].

8.2 Evaluation of O2EM-Emulator

In this section we describe two experiments we performed on different digital objects suitable for the emulator we adapted in Section 7.4. We describe the steps undertaken and the results of the rendering processes as well as the analysis of the resulting event-log files.

8.2.1 Business Process Example: Cassa

As a first example we chose an application that runs in the home-computer mode of the system. We chose an application that allowed us to save data to the external tape drive and reload the data and render it during later use.

The program *Cassa* for the Philips Videopac C7420 home-computer is an early program written in the programming language Microsoft BASIC-80. It was pub-

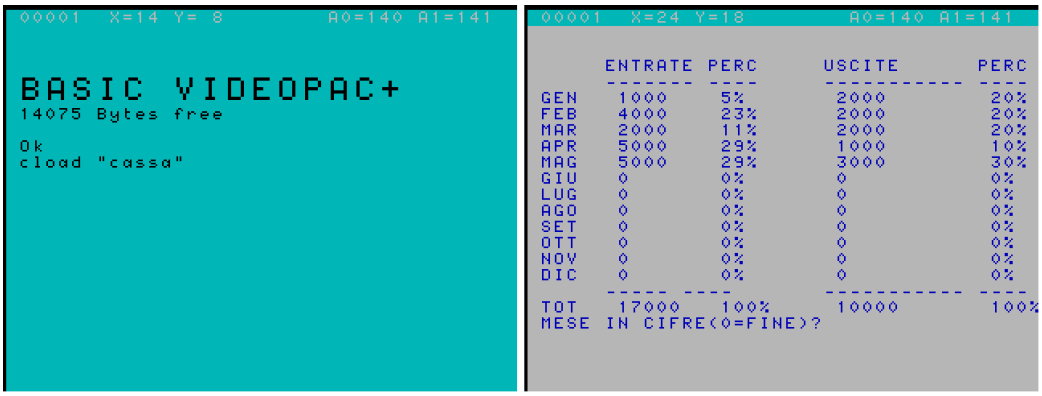


Figure 8.1: Screenshots of the program Cassa on the Philips Videopac C7420 home-computer. Interactive loading of the program on the left, final rendered data on the right.

lished in Italy only as a manual supplement to the home-computer add-on of the system. The program is a typical representation of what home computers were used for at that time besides playing games. One could enter figures of monthly income and spendings, save the data to a tape using an attached tape recorder and display the saved data in various renderings at a later point in time (Figure 8.1).

We started the computer in home-computer mode, loaded the program, entered various fictitious data and saved the data in the program. For the actual evaluation we recorded the following process in the event-log: starting up the emulator in home-computer mode, loading the program (of which we took a screenshot as seen on the left in Figure 8.1), loading the data into the program and displaying the data as also shown on the right in Figure 8.1. So not only the recorded user input but actual data loaded from an external drive influenced the rendering (i.e. what was shown on the screen).

Plan As a first step in the plan phase non-technical aspects like the context in which the system and the program were used in, as well as technical aspects like the environment and all dependencies of the digital object have to be captured. In a museum setting, the historical context of the object would need to be captured to demonstrate one of the earliest examples of how home computers were once used to manage household income and spending. The program Cassa shows how typical data input and processing, as well as typical workflows of users working with a home computer system at the time the system was used in the early eighties. The original setting including the data carriers used to store the data (in this case standard audio tapes) and other properties of the system (e.g., utilization of the home

TV system to use the computer instead of the dedicated display units used today) are documented using pictures of typical set-ups of the system in a family's living room. All the non-technical information about the use of the system and the program itself explain the context the evaluated process was running in at a later point in time.

Once the system itself is documented, the process running on the system has to be documented as well and evaluated against a virtual representation of the process. As only one virtual environment (and emulator) is available for the system, it was evaluated if this would work as a proper preservation action. For our case study we picked one typical use case - loading data from the tape and displaying it as a list.

The second step is to determine the events influencing the rendering of the process. There are a series of external events and external data being supplied that need to be captured during this use case: First the environment is invoked (either by starting the original system or starting an emulator). Then the program is loaded from tape and started by the user. What follows is a sequence of user actions (e.g. selections, confirmations) and system actions (e.g., loading data, rendering data) described in the workflow in Figure 8.2.1. The program first shows an intro screen and then a menu with different options to the user. The user selects to load the data from tape. Afterwards, the system returns to the menu and the user selects to display the data on screen.

If we run the emulator with the workflow described in 8.2.1, an event-log is created where we can see the external files that have been loaded. These include not only the application Cassa itself, but also the file used for storing user entered data. This enables us to identify which resources have been accessed and keep (or simulate) the necessary data for a later verification of the rendering of the preserved application.

Next we have to decide what the significant points are on which we want to compare the rendering and on what level of output we want to compare as step three. To successfully repeat the process in a changed rendering environment, all the actions immediately happening before a user interaction have to show the same result as in the original environment. E.g., if the system waits for the user to press the <RET>-key, then the screen prompting the user to do so has to be correctly rendered, so the user knows what to do. Also the final result of the process has to be the same as on the original environment. If the data displayed is not correctly rendered, then the rendering of the process is of no use to the user. Intermediary steps on the other hand might not always be relevant for the rendering of the process. E.g., in

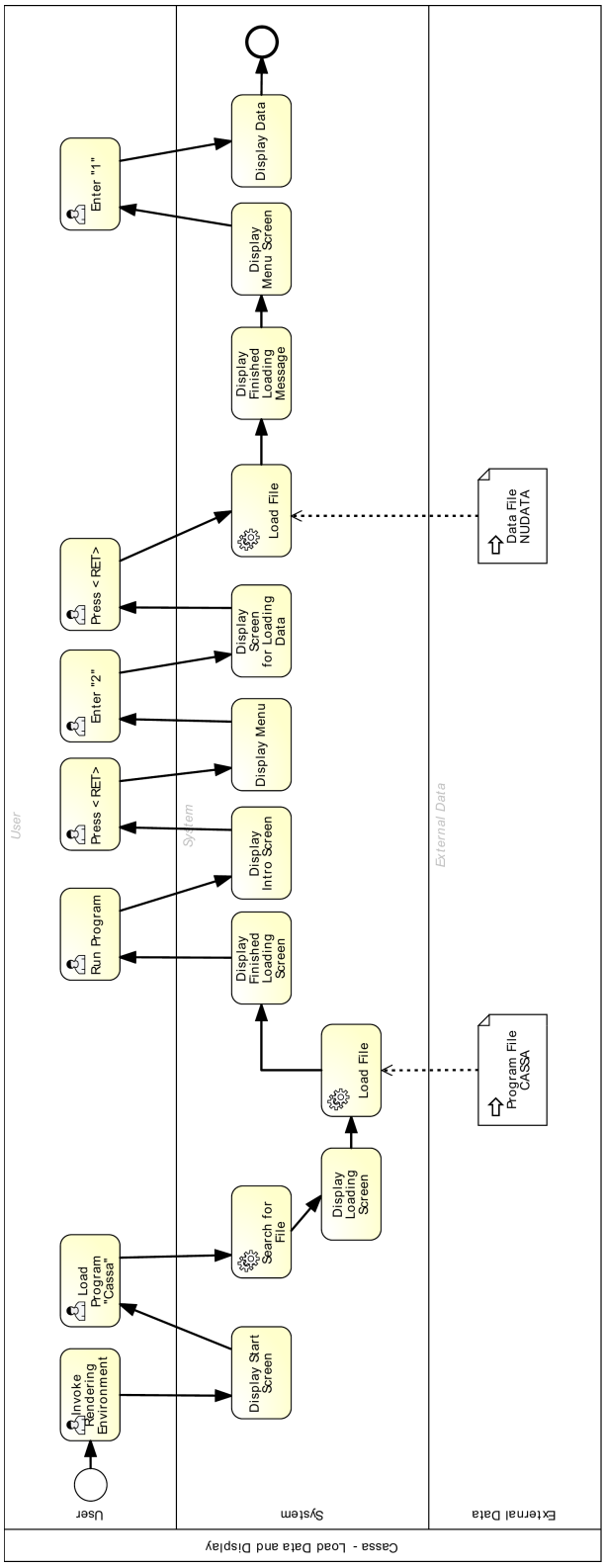


Figure 8.2: Workflow of the use-case of displaying data in the cassa application.

the Cassa-Example it will in most cases not be considered significant if the intermediary step “Display Loading Screen”, that is followed by another system action, and does not require user interaction or even convey valuable information to a user, is rendered correctly. Rendered correctly in this case means that the screen is rendered exactly as on the original system, for exactly the same time, or even rendered at all. The comparison was done on the level of it being rendered on a physical display unit (a TV set the original system was connected to vs. a computer monitor used to display the output of the virtual environment).

As a step four we virtualized the process to capture the external events and the performance of the process. The data both from the tape containing the program as well as the tape containing the data was transferred to the host system of the emulator we were going to use as shown in Section 7.3. Next, in step five, the virtual environment was started, repeating the process side-by-side to the original system.

As the original system used was a quite old and closed system not allowing for any automated extraction of data we had to compare the results manually. Every output of the system deemed significant as defined above was compared to the same output on the original system after applying the same input to both the original and the virtual system in step six and seven of the evaluation framework. Verifying thus that both intermediary steps as well as the end-result of loading the data in the virtual environment was identical, the decision was taken that the emulator is a valid virtual environment representing the output of the original system in a way that was usable for a future user. Once this decision was taken and the emulator was approved as a valid reference environment, future preservation planning processes will be based on this decision and comparison of preservation actions can be done to the emulator, including automatic extraction of data from the virtualized system.

To test the emulator in home-computer mode for determinism, we not only recorded screenshots (as due to the missing random element in the application those would most probably be similar), but also save the memory content of all different memory regions (RAM internal to the 8048h processor, RAM external to the 8048h processor, RAM connected to the Z80 processor and registers of the video processor) along with an image of the displayed screen. As not the whole stream of displayed data was deemed significant, but only the states in the workflow shown in Figure 8.2.1 we only recorded a series of states as described in Section 4.2.2. During execution of the program in O2EM screenshots have been saved on different stages in the workflow along with the user input and the data loaded. Screenshots were

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

Table 8.1: Characteristics for testing the application Cassa with original (=limited) and unlimited speed.

taken on every transition from data displayed (System) to an action required by the User in the workflow shown in Figure 8.2.1. Making sure the user is presented with the information allowing him to make the appropriate choice in the workflow. In the final state "Display Data" not only the screenshot but also the described memory dumps where saved.

To evaluate if the emulator executed this process deterministically, the process was re-run, and the same data was captured as in the initial run. As the data was equal to the data saved in the initial run of the process, using the same input data provided automatically to the emulator, it was asserted that the emulator rendered the object deterministically.

We ran our test under two different settings in the emulator, first with speed limited as a user would usually experience it, and a second time without speed limit, simulating a verification where the test should be performed as fast as possible. We compared all the exported data files (screenshot and memory) with the result, that in all cases the files where exactly the same. Verifying thus that both intermediary steps as well as the end-result of loading the data in the virtual environment was similar, the decision was taken that the emulator is a valid virtual environment representing the output of the original system in a way that was usable for a future user.

As for the timing of the different runs as shown in Table 8.1, we can see that on our system the unlimited test executed the exact same test in only 7.7% of the time needed for a correctly timed emulation while creating the same results.

Preserve At the actual stage of preserving the system, the current status of the tapes were transferred again and the evaluation as explained in the plan phase was repeated. In Section 7.5 we showed how we enhanced the emulator with abilities to record both external data of any kind possible for the system (user input, data loaded from a tape drive), as well as with the ability to extract data in various forms, e.g., screenshots and memory dumps. The virtual environment was set up to record all external data, both screenshots and externally accessed data. Every output of the system was recorded both in

screenshot and memory dump form, and again compared to the same output on the original system. The result of the evaluation was a set of digital objects (the program "Cassa", the data file loaded by the program) and the files created by the virtual environment (logfile of all events during the process including all external data, e.g., user input, applied to the virtual environment, but also the resulting memory dumps and screenshots). For example, for the process shown in Figure 8.2.1 and used in the plan phase to evaluate the preservation action during preservation planning, the following technical data for validation purposes was stored in a package for submission to an archive:

- the necessary external data (the program as well as the data stored to tape),
- the logfile containing both all the external events (i.e., user input and external data applied to the process) as well as internal events as shown in Section 7.5,
- nine screenshots at the defined significant points in the process,
- the state of the system after the process finished running (i.e., in the state "Display Data" consisting of a memory dump of the internal memory of the system as well as the display memory).

The entire package of the technical data for validation has a size of about 15 Mega-Bytes, with the major part being the screenshots as well as the logfile and only some Kilo-Bytes being the program itself and the data stored by the process.

For validation purposes the virtual environment was setup again with both the digital objects and the recorded rendering results and external data. In the second step of the validation the emulator was setup to automatically process the recorded external data instead of reacting to real external events. It again captured the rendering results in memory and on the screen. This step was performed to make sure that the rendering was deterministic, i.e., all the data necessary for the rendering to be done identically in every process was locally known and thus the data can be stored for recreating the process at a later point in time.

Usually during the preserve phase a series of use-cases similar to the one described in the plan-phase are carried out and all external events and rendering results are being recorded and stored along with the digital object (the actual program) in the archive. Using a wider array of use-cases the validity of the virtualized process compared to the original process running on the actual hardware is ensured.

Re-deploy At a later point in time the program along with all the stored data will be extracted from the archive. The virtual environment valid at the time of re-deployment is setup and a test is performed if the results of the rendering in the new virtual environment are identical to the results of the renderings stored in the archive. For the program *Cassa* this means that either a different emulator for the original system or a different version of the emulator used when preserving the process will be available. This emulator will allow us to use the logfile created with all events as input and thus also will create the screenshots and memory dumps as the current version of the emulator does. In this verification step we make sure at the time of re-deployment, that a future emulator will produce the same results as the current version of the emulator - the results that have been compared to the original system at a time when the original system was still available.

As shown in the plan phase, this step can be automated, so even a large number of stored processes and use-cases can be checked without manual effort and even in a much shorter time than real-time emulating the system. We thus can verify all the use-cases by applying the external data and comparing the significant outputs of the processes, if necessary even for different virtual environments available at the time of re-deployment.

Depending on the reason for re-deployment, the actual setting of the emulator can vary. If new data-files for the process have been discovered and have to be rendered using the original program, the verification allows us to make sure, that the original data used for validation produces the same results as in the original virtual environment. This is a strong indication that even different external data unknown at the time of preservation (i.e., different data revived from tapes) will render correctly in the re-deployed virtual environment.

Only one virtual environment for the system was available at the time of carrying out the case study. Thus, to simulate a re-deploy phase for the case study, the package used for submission to the archive was taken as it would be received from an archive at a later point in time. The virtual environment was installed on a system with different hardware then the original system used to capture the data. By executing the captured log on this system, screenshots and memory dumps are created at the same points in the execution process as during the capturing. The resulting data was then compared to the data captured for comparison. As the data was identical, the virtual environment was executed in the “new” environment identically to the original environment. The view-path for the program “Cassa” was thus successfully restored on a new environment.

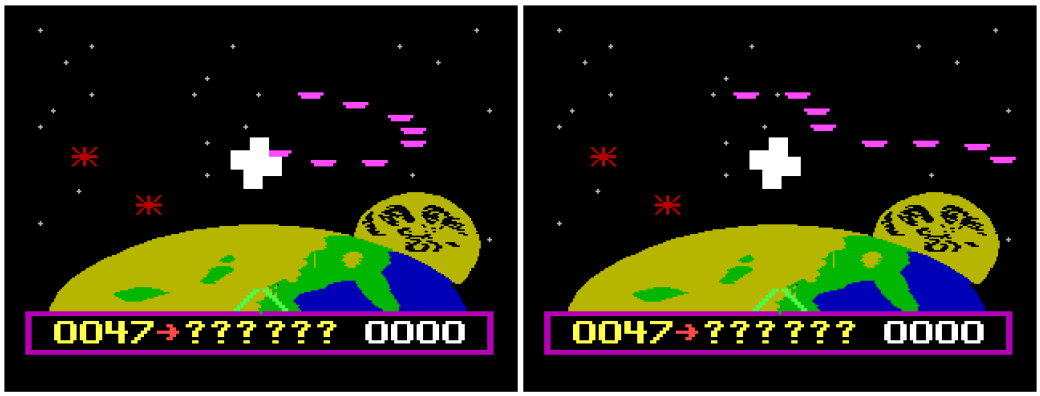


Figure 8.3: Non-deterministic rendering of Terrahawks - result of initial recording (left) and re-run (right).

To summarize, the same external data has been provided to different consecutive renderings of the digital object to make the process deterministic, thus ensuring that differences in the rendering would originate in differences in the rendering environment and not in differences of the behavior of the object. The results were compared to the original rendering as described in Section 5.4.2, thus verifying that the virtualized process is an accurate representation of the original process. Next, the resulting input-files, log-files and rendering result exports were stored along with the necessary digital objects (the program CASSA as well as the data file) during the preserve phase. This lets us at a later point in time (the redeployment phase described in Section 5.4.2) validate that the redeployed object's behavior is similar to the one stored in the preserve phase. The new environment in which the object will be redeployed will be any environment in which a viewpath able to render the digital objects Cassa with its external dependencies (the data file) can be restored.

8.2.2 Video Game: Terrahawks

During the case study for "Cassa" we determined, that the emulator was able to deterministically render a business process example. We chose a video game as a second example because those are usually the most timing sensitive objects on the chosen hardware. We chose a video game running in the standard mode of the emulator emulating a Philips Videopac G7000 running in European timing mode (PAL video standard). We chose the game *Terrahawks* that creates a random impression using external events to change the game play on every execution, to see if repeated execution of the game will produce the same rendering results,

i.e. if the rendering process can be made deterministic for more timing sensitive objects than business processes.

The process of planning for preservation is identical as for the example “Cassa”. Technical aspects of the system and non-technical aspects about the setting of a game played in the early eighties would have to be recorded for a museum setting.

The use-case to be documented for Terrahawks was a typical game play workflow. External events influencing the rendering of Terrahawks are input events using joystick and keyboard. As first step the emulator was started and the game started by pressing the key “0”. One game was played using joystick input. Once the player lost his life, keys were pressed to enter the players name for the high score. A screenshot was taken after the game resulted in the player losing his life and entering his name (at which point the game just restarts, showing the new highest score on the bottom). After entering the name, the joystick was moved some more in the newly started game. A second screenshot was taken and finally the emulation environment was exited by pressing “<ESC>”. The significant events in the use-case were recorded in an event log, excerpts of the log are shown in Section C.3.

In a second step the emulator was restarted with the event-log file given as a command-file to see if the emulator executes the rendering deterministically and the same resulting screenshots are created if the same input is applied. The previously recorded input was applied automatically during the rendering process. However the resulting screenshot taken at the same point in the rendering process as the original screenshot differed from the initial run of the emulator as shown in Figure 8.3.

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

As timing in video games (especially action games) is crucial for the game experience, we used the rendering log to compare the timing of the real hardware (known due to the original system’s schematics) to the values measured in the log as described in Section 5.2.5. The measured values as well as the expected values calculated from the original system’s specification can be seen in Table 8.2.

Based on these results it can be seen that due to the evaluation log we detected another error in the emulator. Even though the emulator was executed with the timing set to European TV-standard PAL timing (50 frames per second), the emulator was still rendering 60 frames per second as in the North American TV

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

Table 8.2: Calculated versus measured key characteristics taken from the event-log of running Terrahawks in O2EM.

standard NTSC. The time taken for each frame was consistently 1/60 of a second, which is correct based on NTSC timing. The emulator was running fast enough to render every frame in less time than the original system would have needed, keeping the subjective feeling of speed for the user steady. Furthermore, it can be seen in Table 8.2 that the timing inside the emulator is not cycle-correct, thus timing-sensitive applications would not run correctly.

The findings about the incorrect timing were used to fix the frame rendering error in O2EM and helped improving the timing in the emulator to the actual calculated values, thus helping us to create a better rendering environment.

The further steps for Plan, Preserve and Re-Deploy phase are identical to the business process example shown in Section 8.2.1, as the rendering environment used is the same one.

8.3 Re-running Scientific Experiments: Music Analysis Workflow

While the case study for “Cassa” was carried out on a quite simple system with very limited external dependencies, the second example we discuss is a scientific workflow described in detail in [Mayer and Rauber, 2012]. Today in most cases papers are published presenting results of experiments and explaining the process how data for the experiment was created and processed. To actually repeat a scientific experiments though, the same process would have to be carried out to achieve the same results and ideally find methods to improve the results of the original paper by doing additional research. Re-running a scientific experiment requires the original view-path of the process creating or processing the data. If an experiment has to be repeated once the original systems are not available anymore, the view-path has to be recreated in a new environment and validated against the original behavior of the process. Using the methodology shown in this thesis we

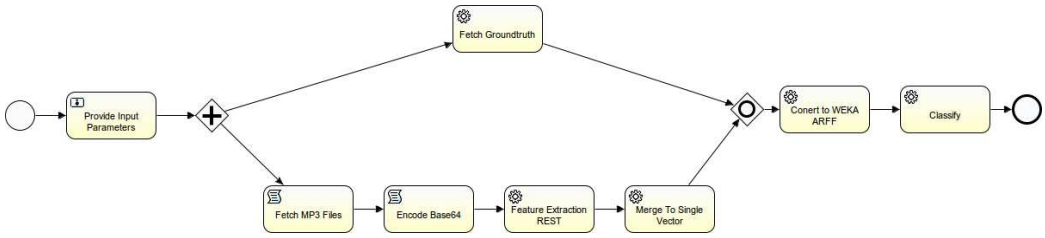


Figure 8.4: Musical genre classification workflow [Mayer *et al.*, 2012a]

thus show how a scientific experiment and its results can be preserved for the long term for validation purposes.

Plan In a first step the digital object and its context have to be described. The workflow is a musical genre classification process from the music information retrieval research community that can be seen in Figure 8.4. It accepts a URL to a list of MP3 files as an input. The html-document is retrieved and the MP3 files are extracted from the document. Every MP3 file is then fetched from its URL, encoded and sent to a feature extraction web service. At the same time the ground truth file is fetched from a different web resource. The extracted features are merged to a single vector and combined with the ground truth and converted into a WEKA[1] ARFF (Attribute-Relation File Format) format file. Finally the classification is performed using the machine learning tool WEKA and a classification report as well as the accuracy measure are provided as output of the process. In [Mayer and Rauber, 2012] the context model for the process is shown in detail.

The process has been modeled to run using the Taverna[2] workflow engine running on a current Linux system. The workflow definition can be seen in Figure 8.5. The system has then been virtualized to be run in a Virtual Machine like VirtualBox.

To evaluate the process we follow the steps as defined in the Plan-phase in Section 5.4.2. After describing the process as a first step, the external dependencies are determined using both the context model created for the process, as well as by measuring the outputs on the interface between the virtual machine and the host system in step two. The process is part of a distributed system. In the workflow shown in Figure 8.4 we can identify

[1]WEKA - `http://www.cs.waikato.ac.nz/ml/weka/`
[2]Taverna Workflow Management System - `http://www.taverna.org.uk/`

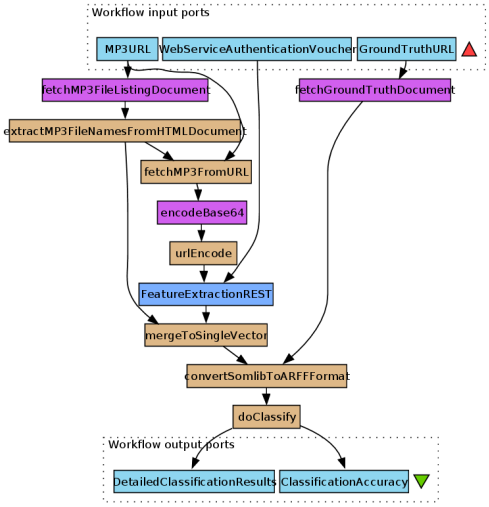


Figure 8.5: Musical genre classification, including fetching of data, modeled in the Taverna workflow engine [Mayer and Rauber, 2012]

different steps where data is sent to external resources and responses are retrieved: Fetching the MP3 file list (fetchMP3FileListingDocument), fetching each MP3 file on the list (fetchMP3FromURL), sending the encoded MP3 file to the web service and retrieving the features (featureExtractionREST), fetching the ground truth (fetchGroundTruthDocument). Similarly to the Cassa case study we have to make sure that the necessary input does not change for evaluation. The list of MP3 files in a preserve phase as well as the ground truth fetched in this phase have to match the list and the ground truth in a later re-deploy phase to be able to compare the results. The feature extraction web service also has to be still available and provide the same features for the same input. All the data transported over the network, i.e., transferred between external resources and the process, has to be captured. Outgoing traffic is data rendered by the process, incoming traffic is external data as response to requests from the process. By storing the data exchanged between the process and the external resources we can make sure that the data stays constant. In step three the decision is taken to compare the process execution by comparing the resulting network stream created by the process. The process executes deterministic, i.e. renders the same results that can be captured on the output ports of the process (DetailedClassificationResults, ClassificationAccuracy) provided that the new rendering environment behaves similar to the original one.

To actually capture the data exchanged between the process and external resources we had different options:

Capturing by the rendering environment The rendering environment we use for verifying the process after virtualization can capture the data on the interface between the virtual and the host system. VirtualBox is an open source software that has various hook points where code could be integrated to capture data. A future emulator executing the virtualized system would have to provide functionality then to apply the captured data to the re-deployed process similar to the emulator used for the Cassa case study.

Use a listener Using a listener to capture the requests to the webservice and the response of the webservice and simulating the external resources for the evaluation is another option for providing identical results on every rerun of the process with the same parameters.

Use workflow engine specific functionality Taverna as a workflow engine is able to record the data that is transmitted to and from the different steps. This data can also be used for a rerun to create deterministic rendering of the process.

Contrary to the "Cassa" example we used a listener to capture the traffic sent to the external resources and simulate the resources for a later re-run of the process. This decision was taken as the process accepts external inputs from network resources and also provides its results as outputs of the process on the network. Implementing external listeners that do not enforce a change in the virtual environment (i.e., implementing capturing / replaying external data on a hardware emulation level inside the virtual machine) is considerable less implementation effort and also makes the chosen solution independent from the used virtual environment. The listener used is described in [Miksa *et al.*, 2013]. The verification-data traffic (to and from the web-service as network XML-stream) as well as the output of the process (also an XML-stream as transmitted as a result to a request on the network) are captured from the rendered form in memory according to step four in the framework. Only the target state of the process is relevant, as the resulting transferred XML-stream of the process has to be identical between different renderings using the same input data. Re-running the process with the listener simulating the data provided to the process (step five) and again capturing the process output (step six) and comparing it to the resulting data captured during the original run (i.e., checking if the resulting XML-streams are identical given the same input conditions) (step seven) we thus

ensure that the necessary external dependencies have been captured and using the listeners is a viable solution to make the process deterministic and the virtual machine renders the process as intended.

Preserve Similar to the plan-phase the data exchanged between the Music Analysis Workflow process and the different external dependencies is captured when the process is preserved as described in step four during a real run of the process using the listeners capturing the network data. For validation purposes the process is then re-run with the captured validation data provided by the listener program instead of the real web-services (step five). The output of the process (the XML-stream containing the classification and accuracy) is captured (step six) and compared if it is identical to the file captured in step four as a seventh and final step. This ensures that the rendering is identical and all the data necessary for making the process rendering deterministic was successfully captured if the resulting data captured during both runs matches. The digital objects (process definition, verification-data captured by the listener as well as resulting data of the process) can be stored in an archive for later re-deployment.

Re-deploy When redeploying the process at a future point in time, the verification of the then used virtual environment is done by applying the captured data exchanged between the process and outside dependencies (step five) and recapturing the rendering results of the process (step six). Instead of the web-services used by the process, the data captured by the listener is used to verify that the process executes identically with the same results in the new environment. Re-deploying the listener is obviously a challenge in itself as most likely it either also has to be run in a virtual environment as the original platform no longer exists. Alternatively, the listener can be reimplemented in the new environment to provide the verification-data that was stored in the archive with the process. By comparing the results on the output ports of the Music Analysis Workflow to the expected results as stored in the archive along with the process (step seven) we can thus verify that the process is running correctly in the new environment for the verification-data.

In case the process is to be re-deployed and supposed to run on data that has not been recorded for evaluation, the web-services originally invoked can be replaced either by the virtual environment supplying recorded data or using the listener that just replies to known requests with recorded responses. As it is very likely that the necessary web-services are not going to exist anymore at this point in time, those either have to be virtualized and re-deployed as well or replaced by mock-up services providing valid data for the process.

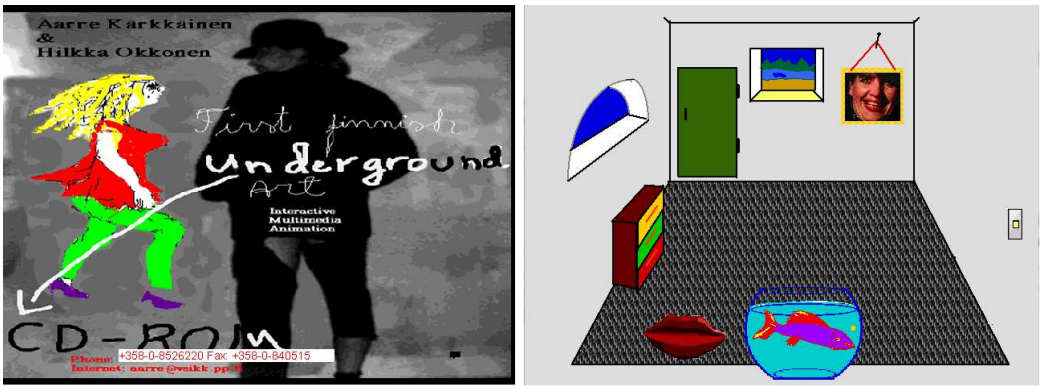


Figure 8.6: First Finnish Underground digital artwork (1995). Title screen (left) and first interactive screen (right) are shown.

8.4 Digital Art Example: First Finnish Underground (Kärkkäinen/Okkonen)

As a third example we discuss the preservation of a piece of interactive art from 1995 from the Ars Electronica[3] electronic art collection. The artwork is a Macromedia Director interactive animation. The title screen and the entrance screen of the artwork are shown in Figure 8.6. The consumer interacts with the artwork by clicking on certain regions of the screen and getting different animations as a result along with audio clips and movies. Some interactive parts require the consumer to drag and drop items on the screen instead of simple clicking.

Plan Describing the digital object along with its dependencies is again the first step in the planning phase. The artwork's original environment was an Intel x86 PC, and the artwork is available as a Windows 95 executable along with additional movie files that needs specific PC-hardware (e.g., a sound card) in its view-path to work as intended. It reacts to user interaction as part of the artwork and experience. External dependencies are thus the user interaction as well as the reaction of the digital object to the specific interaction (step 2). Significant states for the artwork are basically a continuous stream. To keep the consumer experience intact and the "performance" of the artwork as it was originally planned, every single picture along with the accompanying acoustics presented to the consumer has to be present and the reaction time of the artwork has to be as the original. Movies and animations played by the artwork have to be preserved as originally intended.

[3] Ars Electronica - `http://www.aec.at`

Thus a continuous comparison of renderings on the screen as well as the sound effects being played has to be done. A possible way for a comparison is to record the performance in response to the user input as a movie, along with the consumer input and record another movie again in the virtualized environment. If the movies are identical with the same consumer input applied, the performance is reacting similarly to the consumer in both environments and thus is rendered correctly. No other external data is created, so the comparison on a visual and acoustic level is sufficient (step 3).

To evaluate if the artwork behaves deterministic we have to record user input to the artwork as well as the result of the rendering in response to the input (step 4). For proper validation we need to perform the same rendering on both the original artwork (if still possible) and a version that is already separated from its original environment (e.g., in one or more virtual environments emulating the original system on a hardware level) (step 5) and re-capture the renderings in the new environment (step 6). Comparing the significant properties of extracted data from the renderings we can then decide if a virtual environment preserves the original artworks properties sufficiently (step 7). Possible virtual environments in this case could be either a virtual machine running Windows 95 on an Intel x86 PC hardware or an emulator that completely emulates the hardware of the original system. Operating system emulation, e.g., Wine on a Linux system, would also be a possible candidate.

Ideally both user input and artwork output can be recorded by the virtual environment used to render the artwork. If no direct recording of the interaction and the artwork's response is possible (i.e., no support by the rendering environments), external tools that record and replay user input on a Windows operating system and tools that capture screen output can be utilized. By using external tools for capturing and replaying consumer input and for recording the performance the problem exists that the artwork is executed in a multi-threaded environment. Thus probably no exactly deterministic results can be achieved (depending on the sensitivity of the artwork to consumer input).

In our experiment we decided to virtualize the artwork's original environment with a use of a virtual machine running Windows 95 on an Intel x86 PC hardware. We implemented the tool "VirtualBox Record&Playback"[4] shown in Figure 8.7 that runs on the host system and allows us to take control of the virtual system. Thus we were able to control the environment

[4]VirtualBox Record&Playback – `http://sourceforge.net/projects/vboxrecplay/`

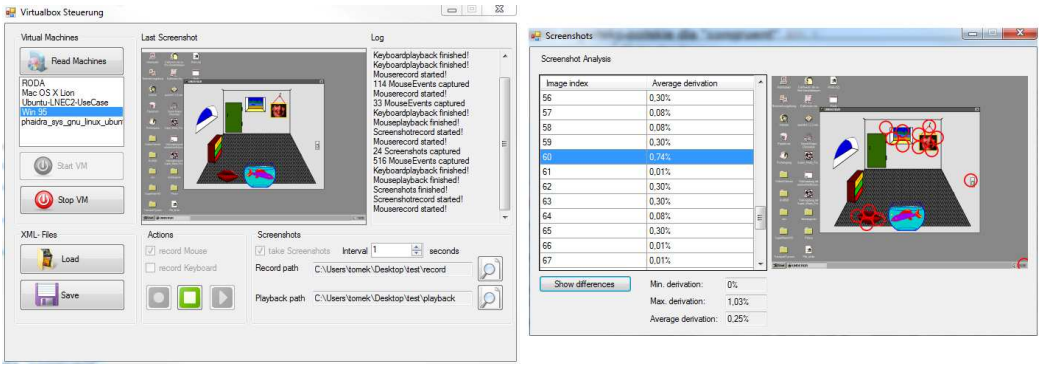


Figure 8.7: Tool for controlling of virtual environment execution. Main window is shown on the left. It allows to select a virtual machine which will be controlled, specify the time interval between screenshots, and the kind of events to be captured (mouse and/or keyboard events). The screenshot comparison window is presented on the right. It depicts differences detected between corresponding screenshots (marked with red circles).

externally by recording and replaying the user input, as well as capturing results of the renderings on the screen. Unfortunately, the used virtual environment was not capable of recording the rendering effects as a continuous stream (movie) and therefore a series of state comparisons have been used instead. Our tool is able to take screenshots of the rendered environment at a given time interval while the original interaction is recorded and also while it is replayed to the virtualized system. The tool is also capable of comparing and detecting differences (if any exist) between corresponding screenshots (as shown on the right in Figure 8.7). It utilizes the VirtualBox API to perform its tasks.

During the course of the experiment we recorded and replayed several combinations of interaction with the artwork. Although the experiment was conducted in a multi-threaded environment, for the most part of the artwork's performance this had no critical impact on the correctness of recording and replaying of mouse events. The tool was able to correctly mimic the recorded scenario and to collect data for screenshot analysis. However, the screenshot analysis revealed that some parts of the artwork did not react as expected to the applied user input but seemingly random elements appeared on the screen resulting in differences in the captured screenshots. The source of the randomness can either be differences in the input that was due to it being an external program not exactly at the exact same moment during the rendering as during the recording phase. Other exter-

nal factors that could have been used to create randomness were also not recorded/replayed to the virtual system (e.g., real time clock of the host system, hardware values like position of the electronic beam on the screen).

Thus, we come to the conclusion that in this case controlling interactions external to the virtual system is not enough to create a deterministic rendering of the artwork. Capturing and replaying capabilities for all external influences to the virtual machine would have to be implemented to successfully compare renderings between the artworks rendering in the virtual system and in its original environment. Thus, we can only hypothetically describe what would happen in the Preservation and Redeployment phases of the preservation workflow.

Preserve In the preserve phase various typical user-interactions to the artwork along with data extracted from renderings of the artwork in response to the user interactions would be recorded (step 4). Capturing consumer input and artwork output is done as described in the plan phase, ideally by the virtual environment. The replay step will show, if the artwork really behaves deterministic when using the same input (step 5) and recording the output of the artwork again (step 6), comparing it to the previously recorded rendering (step 7). Once a deterministic rendering of the artwork with all necessary auxiliary data is achieved, the data applied to the artwork as well as the response (e.g. in the form of a continuous video recording of the artwork) are stored along with the artwork for a later verification once the artwork is taken out of the archive.

Re-deploy If the artwork needs to be re-deployed in a future environment (e.g., for an exhibition), it will be executed in a virtual environment recreating the original view-path. Using the different consumer-inputs recorded and stored along with the artwork in the preserve phase the artwork can then be rendered in the new environment (step 5). The rendering of the artwork is again recorded as a continuous stream (step 6) and compared to the data of the original rendering that has been stored in the archive as well (step 7). Using this method we thus can verify that the artwork still behaves as it did in its original environment. Once the artwork is re-deployed in an exhibition setting with random user interaction that is different from the one recorded in the preserve phase we can thus be confident that the artwork will react to the user as the original artwork would have, thus creating a similar experience to the user (on a technical level).

The comparison described in this chapter was based on screen capturing on the system hosting the artwork. Depending on the artists' intention of the performance, the screen output might also have to be compared on the level

taking the output device properties into account. When the artwork was created, CRT-output devices with output characteristics different to TFT-displays usually used today have been in use, or even devices that might be in use once the artwork is redeployed in a future environment for an exhibition. Using a different display device might considerable change the performance of an artwork experienced by a consumer. While this change might be acceptable (e.g., if a performance closer to the original one is no longer technically possible), it has to be a conscious decision to accept a loss of certain significant properties compared to not performing the artwork at all.

8.5 Summary

In this chapter we presented how the framework presented in Chapter 4 and the preservation workflow shown in Chapter 5 can be applied to evaluate the rendering process of different types of digital objects.

We first evaluated two different digital objects in the emulator O2EM and explained how the event-logs helped us to identify flaws in the rendering process. We rendered different objects in the emulator and analyzed the event-log files, which led us to the following conclusions:

Deterministic Emulation Automatically evaluating emulators by comparing the rendering results at different points in the rendering requires that the rendering environment behaves the same provided with the same external data. In the case of the game 'Terrahawks' evaluated in Section 8.2.2 the emulation was initially not deterministic, leading to different results of the rendering process, even though the obvious external data (user input) was kept constant. Only by making the rendering process deterministic, we could successfully compare the renderings in consecutive executions of the emulator. This would also be the basis for later comparison of the rendering to later emulator versions or even other emulators.

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

Key Characteristics Using the key characteristics about the rendering process which we extracted from the event-log we were able to draw conclusions on

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

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

Overall, we successfully implemented some of the concepts described in Chapter 4, 5 and 6 in the existing emulator O2EM. This not only allowed improving the emulator for more accuracy, but also gave us a better understanding of the evaluation of rendering environments in general. We showed that it is possible to automate the process of evaluating interactive objects beyond the manual testing of emulators with human interaction.

We then showed two objects representing more complex systems, a music classification workflow, and a digital artwork. We showed on what level renderings of the digital objects have to be extracted and what kind of data should be captured. We also discussed the emulation of external data that influences the rendering of the two presented digital objects, web-services in case of the music classification workflow, and user input in case of the interactive digital artwork. The discussed methods for extracting data from the virtual rendering environments require the implementation of functionality as described in Chapter 6.

Chapter 9

Conclusions and Outlook

9.1 Contributions

9.1.1 Challenges

Every digital object needs an environment to be rendered in. The stack of objects needed to render a digital object, i.e., the view-path is described in Section 2.2. Technical obsolescence is a threat to the view-path of a digital object, as the secondary objects needed render a digital object get obsolete.

Digital preservation is the process of making sure that a digital object is accessible over a long period of time. A digital object's view-path is threatened on various levels, besides the physical object or storage media, the logical object (the format) of the digital object, and the conceptual object, i.e., the object that is rendered in a physical form recognizable by a user. To keep an object accessible, we have to make sure to retain a view-path that allows us to render the object so that all properties significant to the designated audience stay intact. The main strategies to preserve a digital object are commonly considered being emulation and migration. Migration changes the digital object in it's logical format and thus the view-path used to render the new format. With emulation the logical format of the digital object stays unchanged, and only layers in the view-path are replaced.

Preservation planning is the process of making sure that a preservation action changes the view-path in a way that keeps the significant properties of an object intact. Traditionally the digital object is characterized by extracting properties stored in the bitstream of the object and compare if the action changed those when using migration as a strategy. Various characterization tools and languages allowing a comparison of extracted properties exist. But not only the stored properties have to be compared, to make sure that the conceptual object stays intact, an authentic rendering of the object is crucial. Preserving complex and interactive digital objects is the subject of various ongoing research projects, so a methodol-

ogy to evaluate authentic rendering is required.

9.1.2 Comparison of Rendering in Migration and Emulation

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

Migration focuses on transforming the object from one file format or encoding to another. Consequently, evaluation of migration solutions predominantly focuses on the characteristics of the source and target object type, and in how far these significant properties can be preserved. Yet, this approach to evaluation is falling short of several of the key requirements of evaluating the authentic preservation of digital objects. It basically views the digital object solely in its encoded form, rather than as intellectual object that only becomes such via interpretation via some form of rendering process, e.g., opening in a specific viewer software or running in a certain environment. However, as we have shown in Chapter 3 this view seems too limited.

In reality, when preservation actions are being evaluated, characterization tools that analyze object structure and content are usually combined with a visual evaluation when both source and target objects are opened in viewer software and verified visually. This corresponds to the evaluation approach presented in this thesis, focusing on a single target state. In fact, every intellectual object in digital form needs to be evaluated with its entire view path, even when this may consist of any of presumably interchangeable standard viewers such as Adobe Acrobat for PDF or any of the myriad of image viewers.

In Chapter 3 we showed how the generic view-path for the rendering of a digital object is composed. We took a look at the different levels on which emulation can be performed and how the digital preservation strategies migration and emulation on the different levels affect the view-path of a digital object. We then compared the effects of the two strategies and showed that the change in view-path makes it necessary to take the rendering environment into account when evaluating any digital preservation action, be it a migration or an emulation action.

We thus argue that the approach to evaluate renderings of a digital object presented in this thesis is not only recommended for interactive content and in combination with emulation and viewer approaches to preservation, but basically applies to the evaluation of any preservation action taken.

9.1.3 Preservation Action Evaluation Framework

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

In Chapter 4 we showed the information we have to collect about a digital artifact, its determinism and significant states, as well as the view-path originally used to render the digital object. We showed the verification data that has to be collected for the object to verify a rendering in a different view-path.

We showed various methods on how to capture user input on the original system and apply it to the emulated environment in Chapter 4. Depending on the object and the technical possibilities it has to be decided which of the various rendered manifestations of a digital object on the original system and in the emulated environment have to be compared. The selection depends also on the significant properties of the object that has to be evaluated.

In Chapter 5 we then showed how the validation data is reapplied to a new rendering environment. The rendered forms of a digital object existing in the view-path are compared to the rendered forms in the original view-path and characteristics of the rendering process are extracted. A list of steps for the evaluation of rendering effects are shown, that form together with the descriptions shown in both chapters the *Preservation Action Evaluation Framework*.

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

The Preservation Action Evaluation Framework describes the evaluation of rendering effects in emulated environments compared to the original environment. There will be cases where the original environment is no longer available or cannot be accessed with reasonable effort (e.g., data archeology). The concepts shown not only allow for a comparison between the original environment and a changed view-path, but also for a comparison between different new view-paths. Usually, it is possible to get an idea about the rendering in the original environment by evaluating various emulated environments, even if the original appearance is no longer known. For example, if certain elements in a screenshot in one emulator

are visible but missing in a screenshot from a different emulator, it is obvious that one of the emulators lacks the ability to render these objects.

9.1.4 Preserving Processes in a Preservation Workflow

In Chapter 5 we also showed in which phases in a preservation workflow an evaluation of the rendering has to be performed according to the validation workflow we previously advised for evaluating the renderings of digital objects.

We showed how the steps for evaluating a rendering can be applied in the different phases of a preservation workflow. In the planning phase we describe the object and environment as well as all of its constraints, including external data sources. Deciding on where to extract data from the system and in what intervals helps us to compare the renderings of a digital object in different environments and create a preservation plan for the digital object. Once we are ready to preserve the object, we have already all necessary information about it and need to collect validation-data as well as extracted significant renderings, which can be used for a comparison once the object is taken out from the archive and re-deployed in its new environment. Following the preservation workflow using the Preservation Action Evaluation Framework allows us to validate at archival time if all necessary components of a digital object have been captured. It ensures that we can successfully verify the digital object's rendering in a future environment that is currently unknown.

9.1.5 Design Requirements for Virtual Environments

Virtual environments used for digital preservation purposes have to fulfill certain requirements. In Chapter 6 we showed what these requirements are on long term stability considering both the durability and the flexibility of virtual environments.

Most importantly, the execution of all parts of the virtual environment has to be made deterministic, i.e. independent from events on a host system and with the ability to provide external data to the same rendering process over and over creating the same rendering as a result. Information about the rendering process has to be provided by the virtual environment. This includes not only the rendered form of the digital object, i.e., data calculated in memory regions of the virtual system and data transformed for interfaces to the host system (e.g., a rendered image for display on the screen), but also information about the rendering process itself. The support for extracting characteristics about the rendering process like timing information and the occurrence of events have to be provided by the virtual environment.

Not only functionality for evaluation has to be considered when developing a virtual environment for digital preservation purposes. Features like the possibility

to transfer data between the host system and the virtual system are essential for a successful use of emulation as a digital preservation access strategy. We showed different strategies to exchange data with virtual environments aware of the guest system running and guest systems being aware of the fact that they are being executed in a virtual environment.

9.1.6 Preserving Digital Objects For An Obsolete System – The C7420

Based on the previous chapters we showed in Chapter 7 how the threats to the data stored for an obsolete system were tackled.

Data stored on audio tapes was extracted as shown in Section 7.3. By using digital archeology to re-engineer the format of the encoded data on the physical storage media we countered the threat of obsolescence on the physical level. Using off-the-shelf audio hardware and decoding the audio waves we were then able to transfer data without using the original system and even improved the quality of the data being read from the tapes. More data was successfully retrieved by our approach than could be read using the original system.

To counter obsolescence on the logical level, the data was decoded and converted to non-obsolete formats where possible. Images and textual data stored on the tapes were migrated to non-obsolete image formats and text stored in formats readable on current systems. Programs stored on the tapes in BASIC programming language were also migrated to a readable text format.

The case study on migration proved that it is possible to extract proprietary data from an analog audio signal stored by a system without previous knowledge of the format it is stored in. By having access to the original system to write test programs we were able to reengineer the audio waveform as well as all data formats and write a tool to migrate the data to non-obsolete formats. Archives or libraries that have or may receive audio tapes containing data for the Philips G7400 can use the created tool to migrate digital data without access to the original system or knowledge of how to handle the system.

If reengineering actions are undertaken today, while the original systems still work, it is possible to develop tools for the migration of digital objects now. Once the original systems do not work anymore, it will not be possible to run code on the original system, thus having to reengineer the system on a circuit-diagram level and disassembling the BIOS source code, which will make the task more difficult and time consuming.

As the programs received from the tapes were readable in their source code form, but not renderable to re-create the logical layer of an interactive executable program, an existing emulator for the system was extended to allow for execution

of programs for the home computer add-on of the system in Section 7.4. We presented the reengineering work involved in enabling emulation of the system itself as well as reengineering necessary for emulating save and load functions. The emulation was implemented keeping digital preservation applications in mind, so data injection and extraction with ease of use for users without expert knowledge of the system was implemented. We described what challenges arose while implementing the emulation and what design decisions were taken and why. We also explained how the guidelines shown in Chapter 6 were considered when implementing certain features like extracting data from the emulation environment.

The work performed for this emulator shows how complex the task to develop an emulator is and what steps are involved especially for a system without proper and open documentation. It further shows that the actual implementation of the emulation of the C7420 Home Computer cartridge was in this special case a comparatively less complex task, as a well documented and already emulated Z80 processor was used as the central processing unit of the C7420. The more time intensive task was the reengineering of the components used for data injection and data extraction, on one hand the emulation of the C7420 tape interface, and on the other hand the proper emulation of keyboard input and data extraction to the clipboard.

To support evaluation of the emulator we also implemented functionality described in Section 6.3.

9.1.7 Evaluated Case Studies

The case studies shown in Table 9.1 showed the different external data that has to be captured for evaluating the rendering of processes, both interactive between a user and a system as well as a system that is retrieving data from various external sources. While in the first two case studies shown in Chapter 8 the layer to capture and replay the data was implemented in the emulator of the system, we showed what would be necessary to implement it for the more complex science process and virtual artwork on more recent systems. Virtualizing the system helps us in capturing the needed data. By inserting an abstraction layer between the now virtual system and a host system, we can implement listeners either on the host system or in the abstraction layer (the virtual machine), to capture the data on the interface between the systems. For a complete monitoring of a virtual system we have to implement capturing and replay for all the interfaces between host and target system. All the data fed into the system from external sources needs to be captured, so by reapplying it at a later point in time the rendering can be made deterministic. Similarly all significant data created by the system and provided to the host rendering environment on any interface (e.g., screen, network) has to be captured to be used as ground truth when re-deploying the object in a new

Digital Object	Virtualization	Capturing Level	Comparison Level	Successful Comparison
Historic Business Process: Cassa	Emulator O2EM	Hardware emulation level	Screenshots taken from emulation environment	Yes (all seemingly random events captured and re-played in Emulator)
Video Game: Terrahawks	Emulator O2EM	Hardware emulation level	Screenshots taken from emulation environment	Yes (all seemingly random events captured and re-played in Emulator)
Scientific Experiment: Music Analysis Workflow	Virtual Machine	Final result files	Captured network traffic	Yes (no random events influencing rendering besides network data)
Digital Art: First Finnish Underground	Virtual Machine	External tool capturing user input	Screenshots taken from virtual machine	No (random events influencing rendering besides user input)

Table 9.1: Case studies carried out with external events captured and re-applied on different levels.

environment at a future point in time.

9.2 Achievements

In the introduction chapter in this thesis we discussed some of the research questions that arise from the need of evaluating a virtual environment. We will revisit these questions below to see how they have been addressed.

RQ1: **How can we evaluate if a digital preservation action keeps the significant properties of a digital object intact?**

We showed that the effects of a digital preservation action have to be evaluated considering the rendering environment. We then explained the prerequisites and what we need to document about the view-path and the digital object to recreate a similar view-path in a different environment. We also showed the levels on which rendered forms of a digital object are found on a system hosting a virtual environment and how we can extract the significant properties of the rendering process and the object. We showed the applicability of the approach in a case study for an obsolete system. We developed a framework to evaluate renderings of a digital object. By following the steps defined in the framework, and taking the decisions outlined in the framework depending on the digital object's nature, the rendering of a digital object can be evaluated against a defined ground truth or other rendering environments.

RQ2: **What do we need to know about a digital object and it's environment to evaluate how a new rendering differs from the original rendering?**

We showed that we need to ensure a deterministic behavior of a digital object to compare different renderings. We explained that the significant states of a digital object are either a target state, a series of states or a continuous stream, depending on the digital object. We showed how we have to describe the rendering environment to allow a recreation of the view-path in a new rendering environment and on what levels rendered forms of a digital object can be extracted.

RQ3: **How can the rendering of a digital object be made deterministic over different rendering cycles and different environments?**

We document the different kinds of data influencing the rendering of a digital object. Some of those are locally known and are either set through the rendering environment once the virtual environment is invoked or stored along with the digital object's view-path. Other events are external to the virtual environment and have to be applied consistently over different rendering cycles.

RQ4: **How can the view-path of a digital object be recreated in a new environment and a new rendering be compared to the original rendering?**

We explained how to recreate the view-path of a digital object in a new environment. We showed the functionality necessary to support the application of external data and created a deterministic rendering using this functionality in the emulator developed for the case study.

RQ5: **How can the evaluation framework be integrated into a preservation workflow?**

We outlined how the framework can be integrated into the wider scope of a preservation workflow. Explaining the data that has to be captured and archived along with the digital object for a successful verification once the object is redeployed we showed which steps of the evaluation framework have to be applied in which phases of the workflow.

RQ6: **What design requirements do we have to virtual environments to allow for evaluation of renderings?**

We outlined the requirements to virtual environments in terms of long term stability and the necessary functionality for supporting and automating evaluation of renderings of a digital object. We also outlined the methods for data exchange between a host and the guest system for enhancing the usability of virtual environments for digital preservation purposes.

9.3 Ongoing and Necessary Future Work

The work presented in this thesis shows a framework for validating rendering results in virtual environments and how to integrate it in the different phases of a preservation workflow. We showed on a rather simple system how to apply the concepts presented in this work. However, future work on the subject is necessary as outlined in the sections below.

9.3.1 Characterization of Environments

While the characterization of the bitstream of digital objects is well underway and already implemented for a wide array of static objects like images, the necessary work for extracting the properties of rendering environments and the dependencies of digital objects has just began. In [Mayer *et al.*, 2012b] a context model that is currently developed in the European research project TIMBUS is shown. The context model describes hardware and software dependencies as well as external dependencies of a digital object. Technical meta-data is also available in the Trustworthy Online Technical Environment Metadata (TOTEM) Registry[1]. The TOTEM registry makes complex hardware and software relationships for digital objects available for the digital preservation community [Anderson *et al.*, 2010].

Describing the context is an important pre-requisite of capturing all the dependencies of a digital object. It allows a reconstruction of the object's view-path and also for validating an object's completeness for preservation.

9.3.2 Ease of Access to Emulation

In Section 2.5.2 some technologies to ease the access to emulation both locally and via remote access have been shown. To make emulation available for a broader audience, these techniques have to be extended to have digital objects accessible in their original environment as easily as a migrated object on the user's desktop.

The ongoing bwFla project (Baden-Wuerttemberg Functional Longterm Archiving and Access)[2] aims to provide easy access to digital objects' original environments using remote access to emulation as described in [von Suchodoletz *et al.*, 2011]. As setting up the object's environment including all hardware and software dependencies is a complex task, it is necessary to support the user in identifying and setting up all necessary secondary digital objects needed in the view-path. Rechert et. al. show a workflow for ingesting environments for complex digital objects [Rechert *et al.*, 2012]. The view-path of a digital object is constructed by

[1] TOTEM – `http://www.keep-totem.co.uk/`
[2] bwFLA – `http://bw-fla.uni-freiburg.de/`

recording the user input during installation of software components. By preserving the setup-process now when expert knowledge is still available, unattended reconstruction of the same view-path is possible at a later point in time.

Only once invoking an emulation environment with the necessary view-path for a digital object is available for users without expert knowledge of the original environment, emulation will be a viable solution for the long term.

9.3.3 Strengthen Emulation as a Digital Preservation Strategy

Emulation still is not considered as a strategy as important to digital preservation as migration. Common arguments against emulation are its complexity both on a system side for setup and necessary expert knowledge for users. Traditionally static documents have been in the main focus of digital preservation. With the focus shifting to more complex, interactive objects and processes, migration as a strategy is not always an option that can be followed. Also, as shown in Chapter 3, every migration changes the view-path of the digital object in a way similar to how emulation changes the view-path. Evaluation of the strategies should be done similarly.

On the 9th International Conference on Preservation of Digital Objects (iPres) 2012 the workshop “Towards Practical Emulation Tools and Strategies - State of the Art Research Meets Real-World Requirements” on the topic of emulation was conducted [von Suchodoletz *et al.*, 2013]. More than 50 participants showed that the interest in emulation is growing in the digital preservation community as the objects that need to be preserved are getting more complex. However, the tools and strategies existing today are mere prototypes and hardly any tools usable for productive deployment exist.

The number of projects on the subject of emulation and preservation of complex objects shown in Section 2.10 shows a growing research interest into solutions for preserving complex objects. However, development of tools still has to be picked up by major commercial players in the domain. With emulation growing to be a strategy complementing migration for objects that cannot be migrated, this is likely to change. Considering the guidelines for developing functionality for virtual environments shown in this thesis, e.g., for easier exchange of data between emulators and their host systems, will also lower the barrier for use of emulators in digital preservation applications.

9.3.4 Connect Virtual Environment Authors and Digital Preservation Stakeholders

The design guidelines shown in Chapter 6 in this thesis allow for an automated and systematic evaluation of renderings of digital objects. However, the necessary functionality to ensure deterministic rendering of an object in the virtual environment is not available in virtual environments yet. Emulators and virtual environments like VirtualBox[3] are not developed with digital preservation requirements in mind, as the developers of these environments are usually not aware of digital preservation.

A first step into raising awareness of digital preservation was made in the KEEP workshop "Joining Forces. International expert workshop about digital preservation" [Lange, 2012]. Emulator authors were brought together with practitioners in digital preservation to discuss the requirements of digital preservation and the potential benefits for emulation authors.

9.3.5 Standardization

A pre-requisite for having wide support for automation of renderings in rendering environments is the existence of standards. Formats for input and output data have to be defined that allow us to do comparisons over various different environments. Virtual Environments have to be enabled to export rendered data in this defined format then and provide a log about what data was created at what time, using what input to the process.

Provided virtual environments are adapted to be preservation aware, successful automated evaluation over a variety of rendering environments is possible and will aid in allowing the use of emulation as a major quality assured digital preservation strategy.

[3] VirtualBox – `http://www.virtualbox.org`

Bibliography

[Aitken *et al.*, 2010] Brian Aitken, Seamus Ross, Andrew Lindley, Edith Michaeler, Andrew Jackson, and Maurice Dobbelsteen. The planets testbed. In Mounia Lalmas, Joemon Jose, Andreas Rauber, Fabrizio Sebastiani, and Ingo Frommholz, editors, *Research and Advanced Technology for Digital Libraries*, volume 6273 of *Lecture Notes in Computer Science*, pages 401–404. Springer Berlin Heidelberg, 2010.

[Anderson *et al.*, 2010] David Anderson, Janet Delve, and Dan Pinchbeck. Toward a workable emulation-based preservation strategy: Rationale and technical metadata. *New Review of Information Networking*, 15(2):110–131, 2010.

[Ashcroft and Manna, 1979] Edward Ashcroft and Zohar Manna. The translation of 'go to' programs to 'while' programs. In *Classics in software engineering*, pages 49–61. Yourdon Press, Upper Saddle River, NJ, USA, 1979.

[Baer, 2005] Ralph H. Baer. *Videogames: in the beginning*. Rolenta Press, 2005.

[Bardon and de Merly, 1984] Christophe Bardon and Benoit de Merly. *Jeux Sur Philips C7420 Videopac+*. Edimicro, Paris, France, 1984.

[Becker and Rauber, 2011a] Christoph Becker and Andreas Rauber. Decision criteria in digital preservation: What to measure and how. *Journal of the American Society for Information Science and Technology (JASIST)*, 62(6):1009–1028, June 2011.

[Becker and Rauber, 2011b] Christoph Becker and Andreas Rauber. Preservation decisions: Terms and Conditions Apply. Challenges, Misperceptions and Lessons Learned in Preservation Planning. In *Proceedings of the ACM/IEEE Joint Conference on Digital Libraries (JCDL 2011)*, pages 67–76, Ottawa, ON, Canada, June 2011.

[Becker *et al.*, 2007] Christoph Becker, Guenther Kolar, Josef Kueng, and Andreas Rauber. Preserving interactive multimedia art: A case study in preservation planning. In *Asian Digital Libraries. Looking Back 10 Years and Forging*

New Frontiers. Proceedings of the Tenth Conference on Asian Digital Libraries (ICADL'07), volume 4822/2007 of *Lecture Notes in Computer Science*, pages 257–266, Hanoi, Vietnam, December 10-13 2007. Springer Berlin / Heidelberg.

[Becker *et al.*, 2008a] Christoph Becker, Hannes Kulovits, Andreas Rauber, and Hans Hofman. Plato: a service-oriented decision support system for preservation planning. In *Proceedings of the ACM/IEEE Joint Conference on Digital Libraries (JCDL'08)*, Pittsburgh, Pennsylvania, USA, June 2008. ACM.

[Becker *et al.*, 2008b] Christoph Becker, Andreas Rauber, Volker Heydegger, Jan Schnasse, and Manfred Thaller. A generic XML language for characterising objects to support digital preservation. In *Proceedings of the 23rd Annual ACM Symposium on Applied Computing (SAC'08)*, volume 1, pages 402–406, Fortaleza, Brazil, March 16-20 2008. ACM.

[Becker *et al.*, 2008c] Christoph Becker, Andreas Rauber, Volker Heydegger, Jan Schnasse, and Manfred Thaller. Systematic characterisation of objects in digital preservation: The extensible characterisation languages. *Journal of Universal Computer Science*, 14(18):2936–2952, 2008. `http://www.jucs.org/jucs_14_18/systematic_characterisation_of_objects`.

[Becker *et al.*, 2009] Christoph Becker, Hannes Kulovits, Mark Guttenbrunner, Stephan Strodl, Andreas Rauber, and Hans Hofman. Systematic planning for digital preservation: Evaluating potential strategies and building preservation plans. *International Journal on Digital Libraries*, 10(4):133–157, 2009.

[Bergmeyer, 2011] Winfried Bergmeyer. The KEEP emulation framework. In *Proceedings of the 1st International Workshop on Semantic Digital Archives*, pages 8–22, Berlin, Germany, September 29 2011.

[Bhushan, 2000] Bharat Bhushan. *Mechanics and Reliability of Flexible Magnetic Media*. Springer, 2000.

[Bonardi and Barthélemy, 2008] Alain Bonardi and Jérome Barthélemy. The preservation, emulation, migration, and virtualization of live electronics for performing arts: An overview of musical and technical issues. *Journal on Computing and Cultural Heritage*, 1(1):1–16, 2008.

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

[Cattermole, 1969] Kenneth W. Cattermole. *Principles of pulse code modulation*. Iliffe, 1969.

[Deconchat and Grandis, 1985] Jacques Deconchat and Valentino C. Grandis. *102 Programmi per Philips C7420 Videopac+*. Editoriale per le scienze informatiche, Milan, Italy, 1985.

[Donnelly, 2006] M Donnelly. JSTOR/Harvard object validation environment (JHOVE). *Digital Curation Centre Case Studies and Interviews*, 2006.

[Felzenszwalb and Huttenlocher, 2004] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59:2004, 2004.

[Forster, 2009] Winnie Forster. *The Encyclopedia of Game Machines: Consoles, Handhelds and Home Computers 1972-2009*. GAMEplan, 2009.

[Grace *et al.*, 2009] Stephen Grace, Gareth Knight, and Lynne Montague. Investigating the significant properties of electronic content over time (inspect): Final report, 2009. `http://www.significantproperties.org.uk/inspect-finalreport.pdf`.

[Granger, 2000] Stewart Granger. Emulation as a digital preservation strategy. *D-Lib Magazine*, Vol. 6 (10), 2000. `http://www.dlib.org/dlib/october00/granger/10granger.html`.

[Guttenbrunner and Rauber, 2011] Mark Guttenbrunner and Andreas Rauber. Design decisions in emulator construction: A case study on home computer software preservation. In *Proceedings of the 8th International Conference on Preservation of Digital Objects (iPRES 2011)*, pages 171–180, Singapore, November 2011.

[Guttenbrunner and Rauber, 2012a] Mark Guttenbrunner and Andreas Rauber. Evaluating an emulation environment: Automation and significant key characteristics. In *Proceedings of the 9th International Conference on Digital Preservation (iPRES 2012)*, pages 201–208, Toronto, Canada, October 1-5 2012.

[Guttenbrunner and Rauber, 2012b] Mark Guttenbrunner and Andreas Rauber. Evaluating emulation and migration: Birds of a feather? In *Proceedings of the 14th Conference on Asian Digital Libraries (ICADL'12)*, pages 158–167, Taipei, Taiwan, November 12-15 2012.

[Guttenbrunner and Rauber, 2012c] Mark Guttenbrunner and Andreas Rauber. A measurement framework for evaluating emulators for digital preservation.

ACM Transactions on Information Systems (TOIS), 30(2):14:1–14:28, May 2012.

[Guttenbrunner *et al.*, 2009] Mark Guttenbrunner, Mihai Ghete, Annu John, Chrisanth Lederer, and Andreas Rauber. Digital archeology: Recovering digital objects from audio waveforms. In *Proceedings of the Sixth international Conference on Preservation of Digital Objects (iPRES 2009)*, pages 90–97, San Francisco, USA, October 2009.

[Guttenbrunner *et al.*, 2010a] Mark Guttenbrunner, Christoph Becker, and Andreas Rauber. Keeping the game alive: Evaluating strategies for the preservation of console video games. *International Journal of Digital Curation (IJDC)*, 5(1):64–90, 2010.

[Guttenbrunner *et al.*, 2010b] Mark Guttenbrunner, J. Wieners, Andreas Rauber, and Manfred Thaller. Same same but different - comparing rendering environments for interactive digital objects. In *EuroMed*, volume 6436 of *Lecture Notes in Computer Science*, pages 140–152. Springer, 2010.

[Guttenbrunner *et al.*, 2011] Mark Guttenbrunner, Mihai Ghete, Annu John, Chrisanth Lederer, and Andread Rauber. Migrating home computer audio waveforms to digital objects: A case study on digital archaeology. *International Journal of Digital Curation (IJDC)*, 6(1):79–98, 2011.

[Herman, 2001] Leonard Herman. *PHOENIX The Fall & Rise of Videogames - Third Edition*. Rolenta Press, 2001.

[Hofman *et al.*, November 2004] H. Hofman, R. Verdegem, M. Day, A. Rauber, M. Thaller, and S. Ross. DELOS WP6 (deliverable d6.1.1) framework for testbed for digital preservation experiments. Tech. Rep., DELOS Network of Excellence, November 2004.

[Huber-Mörk *et al.*, 2012] Reinhold Huber-Mörk, Alexander Schindler, and Sven Schlarb. Duplicate detection for quality assurance of document image collections. In *Proceedings of the 9th International Conference on Digital Preservation (iPRES 2012)*, pages 187–194, Toronto, Canada, October 1-5 2012.

[IEEE, 1987] IEEE. IEEE standard 754-1985 for binary floating point arithmetic. *Reprinted in SIGPLAN*, 22(2):9–25, 1987.

[Innocenti, 2012] Perla Innocenti. Rethinking authenticity in digital art preservation. In *Proceedings of the 9th International Conference on Digital Preservation (iPRES 2012)*, pages 62–67, Toronto, Canada, October 1-5 2012.

[ISO, 2012] ISO. *Space data and information transfer systems – Open archival information system (OAIS – Reference model (ISO 14721:2012)*, 2012.

[Jones, 2004] Caitlin 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, 2004.

[Jurik and Nielsen, 2012] Bolette Ammitzbøll Jurik and Jesper Sindahl Nielsen. Audio quality assurance: An application of cross correlation. In *Proceedings of the 9th International Conference on Digital Preservation (iPRES 2012)*, pages 195–200, Toronto, Canada, October 1-5 2012.

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

[Lange, 2012] Andreas Lange. Results from the KEEP workshop joining forces. International expert workshop about digital preservation. Technical report, Computerspielemuseum, 2012. `http://bw-fla.uni-freiburg.de/wp-uploads/bw-fla.uni-freiburg.de/2012/03/strategy_paper_KEEP_expert_workshop_Joining_Forces_Berlin_final.pdf`.

[Liu, 2000] Jane W. S. W. Liu. *Real-Time Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2000.

[Lohman *et al.*, 2011] Bram Lohman, Bart Kiers, and David Michel. Emulation as a business solution: the emulation framework. In *Proceedings of the 8th International Conference on Preservation of Digital Objects (iPRES 2011)*, pages 167–170, 11 2011.

[Marcum, 1996] Deanna B. Marcum. The preservation of digital information. *The Journal of Academic Librarianship*, 22(6):451 – 454, 1996.

[Matthews *et al.*, 2008] Brian Matthews, Brian McIlwrath, David Giaretta, and Esther Conway. The significant properties of software: A study. JISC Study, 2008. `http://www.jisc.ac.uk/media/documents/programmes/preservation/spsoftware_report_redacted.pdf`.

[Mayer and Rauber, 2012] Rudolf Mayer and Andreas Rauber. Towards time-resilient mir processes. In *Proceedings of the 13th International Society for*

Music Information Retrieval Conference (ISMIR 2012) to appear, Porto, Portugal, October 8-12 2012.

[Mayer *et al.*, 2012a] Rudolf Mayer, Stefan Proell, and Andreas Rauber. On the applicability of workflow management systems for the preservation of business processes. In *Proceedings of the 9th International Conference on Digital Preservation (iPRES 2012)*, pages 109–116, Toronto, Canada, October 1-5 2012.

[Mayer *et al.*, 2012b] Rudolf Mayer, Andreas Rauber, Martin Alexander Neumann, John Thomson, and Gonçalo Antunes. Preserving scientific processes from design to publication. In *Proceedings of the 15th International Conference on Theory and Practice of Digital Libraries (TPDL 2012)*, pages 113–124, Cyprus, September 23–29 2012. Springer.

[McDonough *et al.*, 2010] Jerome P. McDonough, Robert Olendorf, Matthew Kirschenbaum, Kari Kraus, Doug Reside, Rachel Donahue, Andrew Phelps, Christopher Egert, Henry Lowood, and Susan Rojo. Preserving virtual worlds final report. Technical report, University of Illinois, University of Maryland, Rochester Institute of Technology, Stanford University, 2010.

[Mellor *et al.*, 2002] Phil Mellor, Paul Wheatley, and Derrek Sergeant. Migration on request, a practical technique for preservation. In *Proceedings of the 6th European Conference on Digital Libraries (ECDL 2002)*, pages 516–526. Springer, 2002.

[Miksa *et al.*, 2013] Tomasz Miksa, Rudolf Mayer, and Andreas Rauber. Ensuring sustainability of web services dependent processes. *International Journal on Computational Science and Engineering (IJCSE)*, (accepted for publication):1–12, 2013.

[Petermichl, 2009] Karl Petermichl. Dateiformate für Audio. In *Handbuch der Audiotechnik*, chapter 12. Springer Berlin Heidelberg, 2009.

[Petrov and Becker, 2012] Petar Petrov and Christoph Becker. Large-scale content profiling for preservation analysis. In *Proceedings of the 9th International Conference on Digital Preservation (iPRES 2012)*, Toronto, Canada, October 1-5 2012.

[Pettitt, 2003] Jo Pettitt. PRONOM - field descriptions. *The National Archives, Digital Preservation Department*, 2003.

[Phelps and Watry, 2005] Thomas A. Phelps and Paul B. Watry. A no-compromises architecture for digital document preservation. In *Proceedings*

of the 9th European Conference on Research and Advanced Technology for Digital Libraries (ECDL2005), pages 266–277, Vienna, Austria, 2005.

[Phillips, 2010] George Phillips. Simplicity betrayed. *Communications of the ACM*, 53(6):52–58, 2010.

[Rauch and Rauber, 2004] Carl Rauch and Andreas Rauber. Preserving digital media: Towards a preservation solution evaluation metric. In *Digital Libraries: International Collaboration and Cross-Fertilization*, pages 203–212, 2004.

[Rechert *et al.*, 2010] Klaus Rechert, Dirk von Suchodoletz, and Randolph Welte. Emulation based services in digital preservation. In *Proceedings of the 10th Annual Joint Conference on Digital libraries*, JCDL '10, pages 365–368, Gold Coast, Queensland, Australia, 2010.

[Rechert *et al.*, 2012] Klaus Rechert, Dirk von Suchodoletz, and Isgandar Valizada. Future-proof preservation of complex software environments. In *Proceedings of the 9th International Conference on Digital Preservation (iPRES 2012)*, pages 179–182, Toronto, Canada, October 1-5 2012.

[Ross and Gow, 1999] Seamus Ross and Ann Gow. *Digital Archaeology: Rescuing Neglected and Damaged Data Resources, a JISC/NPO Study Within the Electronic Libraries (eLib) Programme on the Preservation of Electronic Materials*. Electronic libraries programme studies. Library Information Technology Center, 1999.

[Rothenberg and Bikson, 1999] Jeff Rothenberg and Tora K. Bikson. *Carrying Authentic, Understandable and Usable Digital Records Through Time: Report to the Dutch National Archives and Ministry of the Interior*. RAND-Europe, 1999.

[Rothenberg, 1998] Jeff Rothenberg. Avoiding technological quicksand: Finding a viable technical foundation for digital preservation. Technical report, January 1998. `http://www.clir.org/pubs/reports/rothenberg/contents.html`.

[Rothenberg, 2000a] Jeff Rothenberg. *Preserving Authentic Digital Information*, pages 51–68. Council on Library and Information Resources, Washington, D.C., USA, 2000.

[Rothenberg, 2000b] Jeff Rothenberg. *Using Emulation to Preserve Digital Documents, Technical Report*. Koninklijke Bibliotheek, 2000.

[Slats and Verdegem, 2004] Jacqueline Slats and Remco Verdegem. Practical experiences of the dutch digital preservation testbed. *VINE (The journal of information and knowledge management systems*, 34(2):56–65, 2004. `http://www.digitaleduurzaamheid.nl/bibliotheek/docs/Article_in_VINE_2004.pdf`.

[Slats, 2003] Jacqueline Slats. Emulation: Context and current status. Tech. Rep., 2003. `http://www.digitaleduurzaamheid.nl/bibliotheek/docs/white_paper_emulatie_EN.pdf`.

[Strodl *et al.*, 2006] Stephan Strodl, Andreas Rauber, Carl Rauch, Hans Hofman, Franca Debole, and Giuseppe Amato. The DELOS Testbed for Choosing a Digital Preservation Strategy. In *Proceedings of the 9th International Conference on Asian Digital Libraries*, pages 323–332, 2006.

[Strodl *et al.*, 2007] Stephan Strodl, Christoph Becker, Robert Neumayer, and Andreas Rauber. How to choose a digital preservation strategy: Evaluating a preservation planning procedure. In *Proceedings of the 7th ACM IEEE Joint Conference on Digital Libraries (JCDL'07)*, pages 29–38, June 2007.

[Strodl *et al.*, 2011] Stephan Strodl, Petar Petrov, and Andreas Rauber. Research on digital preservation within projects co-funded by the european union in the ICT programme. Technical report, Vienna University of Technology, May 2011.

[Strodl *et al.*, 2012] Stephan Strodl, Daniel Draws, Goncalo Antunes, and Andreas Rauber. Business process preservation, how to capture, document & evaluate. In *Proceedings of the 9th International Conference on Preservation of Digital Objects (iPRES 2012)*, Toronto, Canada, October 2012.

[Tarrant and Carr, 2012] David Tarrant and Leslie Carr. LDS3: applying digital preservation principals to linked data systems. In *Proceedings of the 9th International Conference on Digital Preservation (iPRES 2012)*, Toronto, Canada, October 1-5 2012.

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

[van der Hoeven and van Wijngaarden, 2005] Jeffrey van der Hoeven and Hilde van Wijngaarden. Modular emulation as a long-term preservation strategy for digital objects. In *5th International Web Archiving Workshop (IWAW05)*, Vienna, Austria, November 2005.

[van der Hoeven *et al.*, 2005] Jeffrey van der Hoeven, Raymond J. van der Diessen, and Kerstin van der Meer. Development of a universal virtual computer (UVC) for long-term preservation of digital objects. *Journal of Information Science*, 31(3):196–208, 2005.

[van der Hoeven *et al.*, 2007] Jeffrey van der Hoeven, Bram Lohman, and Remco Verdegem. Emulation for digital preservation in practice: The results. *International Journal of Digital Curation*, Vol. 2 (2):123–132, 2007.

[van Diessen, 2002] Raymond J. van Diessen. Preservation requirements in a deposit system. *IBM/KB Long-Term Preservation Study Report Series Number 3 Chapter 3*, 2002. `http://www-05.ibm.com/nl/dias/resource/preservation.pdf`.

[von Suchodoletz and van der Hoeven, 2008] Dirk von Suchodoletz and Jeffrey van der Hoeven. Emulation: From digital artefact to remotely rendered environments. In *Proceedings of the Fifth International Conference on Preservation of Digital Objects (iPRES 2008)*, pages 93–97, London, UK, September 2008.

[von Suchodoletz *et al.*, 2011] Dirk von Suchodoletz, Klaus Rechert, and Isgandar Valizada. Remote emulation for migration services in a distributed preservation framework. In *Proceedings of the 8th international Conference on Preservation of Digital Objects (iPRES 2011)*, pages 158–166, Singapore, 11 2011.

[von Suchodoletz *et al.*, 2013] Dirk von Suchodoletz, Mark Guttenbrunner, and Klaus Rechert. Report on the first iPRES workshop on practical emulation tools and strategies. *D-Lib Magazine*, Vol. 19 (3/4), 2013.

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

[Whitaker *et al.*, 2002] Andrew Whitaker, Marianne Shaw, and Steven D. Gribble. Denali: Lightweight virtual machines for distributed and networked applications. In *In Proceedings of the USENIX Annual Technical Conference*, 2002.

[Woods and Brown, 2008] Kam Woods and Geoffrey Brown. Migration performance for legacy data access. *International Journal of Digital Curation*, 3(2), 2008.

Appendix A

Data Formats of C7420 Home Computer System

A.1 Introduction

To fully understand the file formats of the system presented in Chapter 7.2 we had to reengineer the data formats. This appendix shows the detailed specifications for all possible formats supported by the system as documented in the reengineering process.

A.2 File Formats

To understand the logical format of the data stored in the waveforms it was necessary to find out what kind of data the C7420 can store. Using the original user manual it became apparent that the C7420 is able to store five different kinds of logical bitstream formats with the commands shown in Table A.1.

Table A.1: Logical bitstream formats and corresponding command to save data on the C7420.

Logical bitstream format	**Command**
BASIC Program	CSAVE
Screenshot	CSAVES
Array	CSAVE*
String	CSAVEX
Memory Dump	CSAVEM

A.3 File Header and Data Block

By saving different kinds of test data we were able to first identify the format of the 32 byte file header, which is used for determining the format of the data block:

- 10 bytes 0xD3
- 1 byte determining the format of the file, usually the character after “CSAVE” (e.g. ’S’ for screenshot, 0x20 (Space) for BASIC program)
- 6 bytes for the program name
- 1 byte 0x00
- 5 bytes ASCII characters of the line number at which the execution of the program should start (for BASIC programs only)
- 3 bytes 0x00
- 2 bytes start address in memory (Least Significant Byte (LSB) first)
- 2 bytes length of data block in bytes (excluding the first leading byte 0x00, LSB first)
- 2 bytes checksum: all data bytes added up to a 16 bit value (LSB first)

The data block is separated from the file header by 128 bytes 0xFF. It starts with 0x00 and continues, depending on the specified format in the file header, with the data for each format as specified in the following sections.

A.4 Basic Program

For BASIC programs, the data block is split up into lines which contain the following information:

- 2 bytes RAM address of the next BASIC line (LSB first)
- 2 bytes line number (LSB first)
- The actual line with the BASIC commands
- 1 byte 0x00

At the end of the BASIC program data block 2 bytes 0x00 are written.

Every BASIC command is encoded as a byte code between 0x80 and 0xDF. The byte codes for all other characters in a BASIC command line (including white space) are stored exactly as they are input in the program.

Example BASIC line and encoding:

10 PRINT “HELLO”

Bytes	**Representing**
CF 88	0x88CF (address of next BASIC line in RAM)
0A 00	0x000A = 10 (line number)
94	PRINT (encoded command)
20 22	<SPACE><QUOTATION MARK>
48 45 4C 4C 4F	H E L L O
22	<QUOTATION MARK>
00	indicates end of line

A.5 Screenshot

The Philips G7400 using the C7420 Home Computer Module can display images that are built up using 8x10 pixel characters. 23 of the 24 40-column rows on the screen can be used for graphics, the uppermost row is used to display internal information such as cursor coordinates and cannot be accessed using the standard functions for loading and saving screenshots.

Users can change the representation (glyph) of the built in graphics and text mode characters using the SETEG and SETET commands. Both of these commands have two parameters: the character code of the symbol to be replaced and a string consisting of twenty hexadecimal digits describing the appearance of the symbol. Each character uses an 8x10 pixel grid and is encoded as follows:

- Two hexadecimal digits (one byte) are used for each row of the grid, starting with the topmost one.
- The n-th bit of such a byte, starting with the lowest significant bit, corresponds to the n-th pixel of the row from the right.

A screenshot data block contains character and formatting data which can be used to fill 23 40-column rows on the screen and thus is 1840 bytes long. Only the pointer to the used character and the formatting byte are stored in a screenshot file, user defined characters are lost if the program defining the characters is not stored together with the screenshot file.

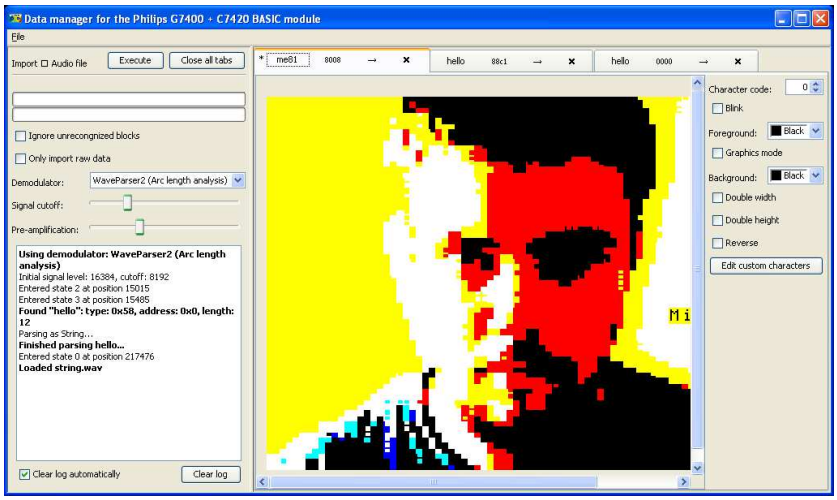


Figure A.1: Migrationtool with image loaded from wav-file.

The formatting of every 2 bytes of data for each screen position is described below, as well as how the formatting data for a byte influences the rendering of a character on the screen. An example image loaded in the migration tool shown in Chapter 7 can be seen in Figure A.1.

A.5.1 Formatting

Each of the 40x23 characters is encoded using two bytes: one byte containing the character code, followed by a byte containing formatting data. A character is displayed either in text mode or graphics mode - this is stored as part of the formatting byte associated with it and determines the used graphical representation (glyph), as well as the meaning of the remaining formatting data, as shown in Figure A.2.

A.5.2 Foreground and Background Colors

There are eight possible colors, each being a combination of red, green and blue. In the three bit representation used by the device, the first (least significant bit determines the amount of red, the second bit determines the amount of green and the third bit determines the amount of blue. The resulting colors, ordered from 0 to 7 are: black, red, green, yellow, blue, magenta, cyan and white.

In graphics mode, each character has its own foreground and background color as specified by the character's formatting byte.

Format Byte Bit Usage in Text Mode:

```
7 6 5 4 3 2 1 0 (lsb)
| | | | | | | |
| | | | | +-+-+--- foreground color
| | | | +--------- !blink
| | | +----------- double height
| | +------------- double width
| +--------------- reverse
+----------------- 0 (controls graphics mode)
```

Format Byte Bit Usage in Graphics Mode:

```
7 6 5 4 3 2 1 0 (lsb)
| | | | | | | |
| | | | | +-+-+--- foreground color
| | | | +--------- !blink
| +-+-+----------- background color
+----------------- 1 (controls graphics mode)
```

Figure A.2: Structure of byte used for formatting in Text Mode (top) and Graphics Mode (bottom).

In text mode, each character has its own foreground color encoded in the formatting byte. The background color is "inherited" from the last character previously encountered on the row that was either a graphics mode character or a text mode character with a code greater or equal to 128. In the second case, the background color is taken from the same bits as from a graphics character (bits 4,5 and 6). This method is generally used for setting the background color at the beginning of a row and can be seen in screenshots originating from the device - the first column contains characters with code 128 and a formatting byte with both hexadecimal digits set to the desired background color for each line.

A.5.3 Double Width and Height

Text mode characters can be displayed in double width or height. Since glyph sizes are fixed, two consecutive grid cells/glyphs are required to fully display one double width or double height character, and a total of four cells/glyphs is required for a double width, double height character. Setting the double width / double height attribute for a single character results in only half of it being shown (or a quarter, if both attributes are set).

The same character code / format pair must yield 2 or 4 different glyphs, depending on which part of the character needs to be drawn. The rules used to determine which part to draw are as follows:

Double width

Each occurrence of a double width character after a single width character (or after a complete double width character) uses the first glyph (left part of the character). A double width character directly following the first glyph uses the second glyph.

Double height

Each line is assigned either top glyphs or bottom glyphs. A line containing double height characters that comes after a line containing no such characters uses top glyphs, consecutive lines are assigned bottom and top glyphs in an alternating fashion.

A.5.4 Blink and Reverse

The blink attribute makes a character blink on the screen - it is shown for one second, then hidden for one second, then shown again, and so on.

The reverse attribute reverses the background color and foreground color of a character. It also reverses the blinking phase for that character.

A.6 Array

The first byte of an array encodes the number of dimensions of the array. For each dimension, two subsequent bytes encode the number of fields in the dimension (LSB first). Finally, for every entry in the array, 4 bytes are used to express the value in different formats, depending on whether the array contains strings or numbers.

A.6.1 String Array

- 1 byte length of the string in bytes
- 1 unused byte
- 2 bytes address of the string in memory

Note that the actual string data is not saved in the array but the strings have to be saved and loaded separately using the string save command “CSAVEX”.

A.6.2 Number Array

By saving number arrays on the original system, examining the resulting byte stream, changing values and re-loading the array onto the original system we were able to find out that a floating point format is used to store numbers. The encoding is similar to, but does not follow the IEEE 754 floating point standard [IEEE, 1987], as that was released 2 years later than the C7420 cartridge. With further testing, the bits for mantissa, sign and exponent were determined. 4 bytes are used to encode the number as a 32 bit floating point value (LSB first), with the following meaning of the bits:

bit 25–32	**bit 24**	**bit 1–23**
exponent (exponent bias = 129)	sign (1 = negative)	mantissa

So any number can be calculated using the following equation:

$$number = sign \times mantissa \times 2^{exponent}$$

where,
$sign = (-1)^{<bit24>}$
$mantissa = 1 + (< bit1 - 23 > /2^{23})$
$exponent =< bit25 - 32 > -129$

A.7 String

Strings are stored as a stream of bytes using the ASCII encoding (number of bytes according to the file header information).

A.8 Memory Dump

Memory dumps are stored as byte values (number of bytes according to the file header information).

Appendix B

ASCII Table for C7420 Home Computer System

B.1 Introduction

The C7420 Home Computer System uses numeric codes for characters that are different from standardized ASCII character codes used today. The following list shows the characters, their numeric codes in the C7420 system and the ASCII character codes for the characters.

The lists shown in this appendix are used to convert both data imported to the C7420 rendering environment as well as data exported from the environment (either in file format or when copying data to the clipboard). The lists where created using the C7420 character tables in Appendix D of the original system's manual.

B.2 Converting C7420 Character Set to ASCII

Figure B.1 shows what character code used in the C7420 cartridge is mapped to what character in the Windows-1252 ASCII extended character set. The list shows only characters that have different character codes in the C7420 character set than in the ASCII character set, or that are characters that exist on the C7420 and not in the ASCII character set. Every character in the C7420 set has been mapped to a character in the ASCII character set. If no corresponding printable ASCII character exists, a textual description of the character was used in the table. The character mapping was performed by selecting characters in the ASCII character set that closest resemble the original character in the C7420 character set.

The conversion shown in this section is used whenever data is migrated from the C7420 environment to the host environment (i.e., copying screen content to

the clipboard, saving file content in ASCII format).

Figure B.1: Characters mapped from the C7420 character set to ASCII character set.

Code C7420	Character C7420	ASCII Code CP-1252	Character ASCII	Code C7420	Character C7420	ASCII Code CP-1252	Character ASCII
0x00	Inverted question mark	0xBF	¿	0x16	û	0xFB	û
0x01	Â	0xC2	Â	0x17	à	0xE0	à
0x02	É	0xC9	É	0x18	÷	0xF7	÷
0x03	£	0xA3	£	0x19	è	0xE8	è
0x04	$	0x24	$	0x1A	œ	0x9C	œ
0x05	Ç	0xC7	Ç	0x1B	ê	0xEA	ê
0x06	#	0x23	#	0x1C	¼	0xBC	¼
0x07	À	0xC0	À	0x1D	½	0xBD	½
0x08	Ù	0xD9	Ù	0x1E	¾	0xBE	¾
0x09	È	0xC8	È	0x1F	Ô	0xF4	Ô
0x0A	Œ	0x8C	Œ	0x23	Ë	0xCB	Ë
0x0B	Ê	0xCA	Ê	0x24	â	0xE2	â
0x0C	left arrow	0x3C	<	0x5E	î	0xEE	î
0x0D	up arrow	0x5E	^	0x5F	horicontal line bottom	0x5F	_
0x0E	right arrow	0x3E	>	0x60	horizontal line center	0x97	—
0x0F	down arrow	0x76	v	0x7B	vertical line left	0x7B	{
0x10	°	0xBA	°	0x7C	vertical line center	0x7C	|
0x11	±	0xB1	±	0x7D	vertical line right	0x7D	}
0x12	é	0xE9	é	0x7E	horizontal line top	0x97	—
0x13	ë	0xEB	ë	0x7F	full block	0xA0	<NBSP>
0x14	ï	0xEF	ï	0xFE	..	0x85	...
0x15	ç	0xE7	ç	0xFF	^		^

B.3 Converting ASCII Character Set to C7420 Characters

To convert data from the Windows-1252 ASCII extended character set, the mapping shown in Figures B.2 and B.3 are used. Shown are again only the characters that have a different code in the two character sets or exist in one character set, but not in the other. The ASCII extended character set used consists of 8 bit, while for C7420 basically only 7 bits are used (with the two exceptions 0xFE and 0xFF shown in Figure B.1). By mapping a larger set of characters to a smaller set, some of the characters in the C7420 set are reused for similar looking characters in the ASCII set. Control characters in the ASCII character set (0x00-0x1F) have not been mapped.

The conversion shown in this section is used when data from the host environment is transferred to the C7420 environment (i.e., copying data into the keyboard buffer of the C7420, loading file content from ASCII format).

Figure B.2: Characters mapped from the ASCII character set to C7420 character set: 0x23-0xD4

ASCII Code CP-1252	Character ASCII	C7420 Code	Character C7420
0x23	#	0x06	#
0x24	$	0x04	$
0x5E	^	0xFF	^
0x60	`	0x27	'
0x7B	{	0x7B	vertical line left
0x7C	|	0x7C	vertical line center
0x7D	}	0x7D	vertical line right
0x7E	~	0x2D	-
0x80	€	0x45	E
0x82	‚	0x2X	,
0x83	ƒ	0x66	f
0x84	„	0x22	"
0x85	…	0xFE	..
0x86	†	0x2B	+
0x87	‡	0x2B	+
0x88	^	0xFF	^
0x89	‰	0x25	%
0x8A	Š	0x53	S
0x8B	‹	0x3C	<
0x8C	Œ	0x0A	Œ
0x8E	Ž	0x5A	Z
0x91	‘	0x27	'

ASCII Code CP-1252	Character ASCII	C7420 Code	Character C7420
0x92	’	0x27	'
0x93	“	0x22	"
0x94	”	0x22	"
0x95	•	0x10	°
0x96	–	0x2D	-
0x97	—	0x60	—
0x98	˜	0x2D	-
0x99	™	0x74	t
0x9A	š	0x73	s
0x9B	›	0x3E	>
0x9C	œ	0x1A	Œ
0x9E	ž	0x5A	Z
0x9F	Ÿ	0x59	Y
0xA0	NBSP	0x20	space
0xA1	¡	0x69	I
0xA2	¢	0x63	c
0xA4	£	0x03	£
0xA4	¤	0x10	°
0xA5	¥	0x59	Y
0xA6	¦	0x7C	|
0xA7	§	0x24	$
0xA8	¨	0xFE	..

ASCII Code CP-1252	Character ASCII	C7420 Code	Character C7420
0xA9	©	0x63	c
0xAA	ª	0x61	a
0xAB	«	0x3C	<
0xAC	¬	0x2D	-
0xAD	-	0x2D	-
0xAE	®	0x52	R
0xAF	¯	0x7E	horizontal line top
0xB0	°	0x10	°
0xB1	±	0x11	±
0xB2	²	0x32	2
0xB3	³	0x33	3
0xB4	´	0x27	'
0xB5	µ	0x75	u
0xB6	¶	0x50	P
0xB6	·	0x2E	.
0xB8	¸	0x2E	.
0xB9	¹	0x31	1
0xBA	º	0x10	°
0xBB	»	0x3E	>
0xBC	¼	0x1C	¼
0xBD	½	0x1D	½
0xBE	¾	0x1E	¾

ASCII Code CP-1252	Character ASCII	C7420 Code	Character C7420
0xBF	¿	0x00	Inverted questionmark
0xC0	À	0x07	À
0xC1	Á	0x07	À
0xC2	Â	0x01	Â
0xC3	Ã	0x01	Â
0xC4	Ä	0x01	Â
0xC5	Å	0x01	Â
0xC6	Æ	0x41	A
0xC7	Ç	0x05	Ç
0xC8	È	0x09	È
0xC9	É	0x09	È
0xCA	Ê	0x0B	Ê
0xCB	Ë	0x0B	Ê
0xCC	Ì	0x49	I
0xCD	Í	0x49	I
0xCE	Î	0x49	I
0xCF	Ï	0x49	I
0xD0	Ð	0x44	D
0xD1	Ñ	0x4E	N
0xD2	Ò	0x4F	O
0xD3	Ó	0x4F	O
0xD4	Ô	0x4F	O

Figure B.3: Characters mapped from the ASCII character set to C7420 character set: 0xD5-0xFF

ASCII Code CP-1252	Character ASCII	C7420 Code	Character C7420
0xD5	Õ	0x4F	O
0xD6	Ö	0x4F	O
0xD7	×	0x78	x
0xD8	Ø	0x4F	O
0xD9	Ù	0x08	Ù
0xDA	Ú	0x08	Ù
0xDB	Û	0x08	Ù
0xDC	Ü	0x08	Ù
0xDD	Ý	0x59	Y
0xDE	Þ	0x50	P
0xDF	ß	0x53	S
0xE0	à	0x17	à
0xE1	á	0x17	à
0xE2	â	0x17	à
0xE3	ã	0x17	à
0xE4	ä	0x17	à
0xE5	å	0x17	à
0xE6	æ	0x61	a
0xE7	ç	0x15	ç
0xE8	è	0x19	è
0xE9	é	0x12	é
0xEA	ê	0x1B	ê

ASCII Code CP-1252	Character ASCII	C7420 Code	Character C7420
0xEB	ë	0x13	ë
0xEC	ì	0x69	i
0xED	í	0x69	i
0xEE	î	0x5E	î
0xEF	ï	0x14	ï
0xF0	ð	0x6F	o
0xF1	ñ	0x6E	n
0xF2	ò	0x6F	o
0xF3	ó	0x6F	o
0xF4	ô	0x6F	o
0xF5	õ	0x6F	o
0xF6	ö	0x6F	o
0xF7	÷	0x18	÷
0xF8	ø	0x6F	o
0xF9	ù	0x16	û
0xFA	ú	0x16	û
0xFB	û	0x16	û
0xFC	ü	0x16	û
0xFD	ý	0x79	y
0xFE	þ	0x70	p
0xFF	ÿ	0x79	y

Appendix C

Event Log for O2EM

C.1 Introduction

To enable an automated evaluation of the emulator shown in Chapter 7, a log was implemented that contains important events in the rendering process. The saved log can also be used as a command file for the emulator once the rendering process is repeated. In this chapter we show the various events that were implemented for logging in the event log as well as snippets from an example log.

C.2 Implemented Events

Table C.1 shows the events that have been implemented in O2EM shown in Section 7.4. It shows the type of event, its usage, the code associated to the event, and any additional data recorded.

Usage can be any of the following:

Log only an Event that is only logged, and not used when replaying the log-file on a re-run of the rendering process

Log and Replay an Event that is also triggered again on a re-run of the rendering process at the same point in time

Table C.1: Events implemented in the O2EM Event Log.

code	event	usage	description	additional data
0	Start Emulation	Log only	Logs the time of the rendering process start	filename of cartridge/checksum of cartridge file, emulator name, version number and current date/time
1	Keypress	Log and replay	Logs a key event in the emulator	key code and associated character
2	Joystick	Log and replay	Logs a joystick event	joystick number (0 or 1) and the joystick value read from the hardware port
3	Screenshot	Log and replay	Logs the event of a screenshot being taken in the emulator	filename of screenshot
4	Memory Save	Log and replay	Logs the event of a memory region being saved	filename of memory dump and region being saved (VDC = video processor registers, Z80 = C7420 RAM, EXTRAM / INTRAM = RAM external / internal of Intel 8048h processor, VPP = second video processor registers)
5	C7420 File loaded	Log only	Logs a read access to an external file (home computer cartridge emulation only)	filename
6	C7420 File saved	Log only	Logs a write access to an external file (home computer cartridge emulation only)	filename
254	Vertical Blank	Log only	Logs the occurrence of a vertical blank event (frame start)	
255	Emulation Stop	Log and replay	Logs the time the rendering process was finished, also stops the rendering process on a replay	

C.3 Example Log

The following snippets are taken from a log created during a rendering of Terrahawks during the evaluation shown in Section 8.2.2. Only relevant parts of the 3863 line long log taken over a period of 83,121 seconds are shown, parts containing only a logging of vertical blank events and joystick movements were shortened/suppressed.

```
< Starting the rendering process >
0;0,000;0;0;Emulation started;D:/Data/Emu/G7000/o2em120src/bin/roms/videopac/vp_51pl
.bin|764894A1;O2EM v1.21|2012-05-02 21:13:42
5494;0,000;0;254;Vertical Blank;;
12754;0,052;1;254;Vertical Blank;;
20013;0,072;2;254;Vertical Blank;;
27271;0,092;3;254;Vertical Blank;;
34531;0,112;4;254;Vertical Blank;;
41789;0,132;5;254;Vertical Blank;;
49048;0,152;6;254;Vertical Blank;;
56307;0,172;7;254;Vertical Blank;;
63566;0,192;8;254;Vertical Blank;;
70825;0,212;9;254;Vertical Blank;;
78085;0,232;10;254;Vertical Blank;;
85344;0,252;11;254;Vertical Blank;;
...
```

```
< Starting the game by pressing the key '0' >
491847;1,372;67;254;Vertical Blank;;
499106;1,392;68;254;Vertical Blank;;
499190;1,392;68;1;Keypress;48;'0'
499197;1,392;68;1;Keypress;48;'0'
506365;1,412;69;254;Vertical Blank;;
513624;1,432;70;254;Vertical Blank;;
...
< Gameplay (vertical blank and joystick movement events) >
2524367;6,971;347;254;Vertical Blank;;
2531626;6,991;348;254;Vertical Blank;;
2535061;7,011;349;2;Joystick;0;247
2538886;7,011;349;254;Vertical Blank;;
2542266;7,031;350;2;Joystick;0;247
2546144;7,031;350;254;Vertical Blank;;
2549699;7,051;351;2;Joystick;0;247
2553403;7,051;351;254;Vertical Blank;;
2557134;7,071;352;2;Joystick;0;247
2560662;7,071;352;254;Vertical Blank;;
...
7862002;21,692;1083;2;Joystick;0;239
7866992;21,692;1083;254;Vertical Blank;;
7869164;21,712;1084;2;Joystick;0;239
7874250;21,712;1084;254;Vertical Blank;;
7881510;21,732;1085;254;Vertical Blank;;
...
< Gameplay finished, entering the name 'mark' for the high score >
8963100;25,210;1234;254;Vertical Blank;;
8970359;25,230;1235;254;Vertical Blank;;
8971205;25,230;1235;1;Keypress;109;'m'
8977619;25,250;1236;254;Vertical Blank;;
8978467;25,250;1236;1;Keypress;109;'m'
8984878;25,270;1237;254;Vertical Blank;;
8992136;25,290;1238;254;Vertical Blank;;
8993024;25,290;1238;1;Keypress;109;'m'
8999396;25,310;1239;254;Vertical Blank;;
9006654;25,330;1240;254;Vertical Blank;;
9007540;25,330;1240;1;Keypress;109;'m'
9013913;25,350;1241;254;Vertical Blank;;
9014801;25,350;1241;1;Keypress;97;'a'
9021172;25,370;1242;254;Vertical Blank;;
9022060;25,370;1242;1;Keypress;97;'a'
9028432;25,390;1243;254;Vertical Blank;;
9035690;25,410;1244;254;Vertical Blank;;
9036578;25,410;1244;1;Keypress;97;'a'
9042949;25,430;1245;254;Vertical Blank;;
9050209;25,450;1246;254;Vertical Blank;;
9051057;25,450;1246;1;Keypress;97;'a'
9057467;25,470;1247;254;Vertical Blank;;
9064726;25,490;1248;254;Vertical Blank;;
9071986;25,510;1249;254;Vertical Blank;;
9079244;25,530;1250;254;Vertical Blank;;
9086503;25,550;1251;254;Vertical Blank;;
9093763;25,570;1252;254;Vertical Blank;;
9101022;25,590;1253;254;Vertical Blank;;
9108281;25,610;1254;254;Vertical Blank;;
9115539;25,630;1255;254;Vertical Blank;;
9122799;25,650;1256;254;Vertical Blank;;
9130058;25,670;1257;254;Vertical Blank;;
9137317;25,690;1258;254;Vertical Blank;;
9144575;25,710;1259;254;Vertical Blank;;
9151835;25,730;1260;254;Vertical Blank;;
```

```
9159094;25,750;1261;254;Vertical Blank;;
9166352;25,770;1262;254;Vertical Blank;;
9173612;25,790;1263;254;Vertical Blank;;
9180871;25,810;1264;254;Vertical Blank;;
9188129;25,830;1265;254;Vertical Blank;;
9195389;25,850;1266;254;Vertical Blank;;
9202648;25,870;1267;254;Vertical Blank;;
9209906;25,890;1268;254;Vertical Blank;;
9217166;25,910;1269;254;Vertical Blank;;
9224425;25,930;1270;254;Vertical Blank;;
9231683;25,950;1271;254;Vertical Blank;;
9238943;25,970;1272;254;Vertical Blank;;
9239757;25,970;1272;1;Keypress;114;'r'
9246202;25,990;1273;254;Vertical Blank;;
9247018;25,990;1273;1;Keypress;114;'r'
9253460;26,010;1274;254;Vertical Blank;;
9260720;26,030;1275;254;Vertical Blank;;
9261609;26,030;1275;1;Keypress;114;'r'
9267979;26,050;1276;254;Vertical Blank;;
9275237;26,070;1277;254;Vertical Blank;;
9276126;26,070;1277;1;Keypress;114;'r'
9282496;26,090;1278;254;Vertical Blank;;
9289756;26,110;1279;254;Vertical Blank;;
9290645;26,110;1279;1;Keypress;107;'k'
9297014;26,130;1280;254;Vertical Blank;;
9297901;26,130;1280;1;Keypress;107;'k'
9304273;26,150;1281;254;Vertical Blank;;
9311533;26,170;1282;254;Vertical Blank;;
9312422;26,170;1282;1;Keypress;107;'k'
9318791;26,190;1283;254;Vertical Blank;;
9326050;26,210;1284;254;Vertical Blank;;
9326937;26,210;1284;1;Keypress;107;'k'
9333309;26,230;1285;254;Vertical Blank;;
9340569;26,251;1286;254;Vertical Blank;;
...
< Gameplay of another game - joystick movements and vertical blank events >
10001138;28,560;1377;254;Vertical Blank;;
10008397;28,580;1378;254;Vertical Blank;;
10010238;28,600;1379;2;Joystick;0;253
10015655;28,600;1379;254;Vertical Blank;;
10019489;28,620;1380;2;Joystick;0;253
10022915;28,620;1380;254;Vertical Blank;;
10026155;28,640;1381;2;Joystick;0;253
10030173;28,640;1381;254;Vertical Blank;;
10033416;28,660;1382;2;Joystick;0;253
...
< Taking a screenshot of the currently displayed image as file terra1_1.bmp >
18936966;53,180;2608;254;Vertical Blank;;
18944226;53,200;2609;254;Vertical Blank;;
18944226;64,731;2609;3;Screenshot saved;bin/scshot/terra1_1.bmp;
18951484;65,290;2610;254;Vertical Blank;;
18958744;65,310;2611;254;Vertical Blank;;
18966003;65,330;2612;254;Vertical Blank;;
18973262;65,350;2613;254;Vertical Blank;;
18980521;65,370;2614;254;Vertical Blank;;
18987779;65,390;2615;254;Vertical Blank;;
18995039;65,410;2616;254;Vertical Blank;;
19002297;65,430;2617;254;Vertical Blank;;
19009557;65,450;2618;254;Vertical Blank;;
19016815;65,470;2619;254;Vertical Blank;;
19024075;65,490;2620;254;Vertical Blank;;
19031334;65,510;2621;254;Vertical Blank;;
```

```
19038593;65,530;2622;254;Vertical Blank;;
19045852;65,550;2623;254;Vertical Blank;;
19053111;65,570;2624;254;Vertical Blank;;
19060370;65,590;2625;254;Vertical Blank;;
19067629;65,610;2626;254;Vertical Blank;;
19074887;65,630;2627;254;Vertical Blank;;
19082147;65,650;2628;254;Vertical Blank;;
19089405;65,670;2629;254;Vertical Blank;;
19091262;65,690;2630;2;Joystick;0;247
19096664;65,690;2630;254;Vertical Blank;;
19100267;65,710;2631;2;Joystick;0;247
19103924;65,710;2631;254;Vertical Blank;;
19107574;65,730;2632;2;Joystick;0;247
19111183;65,730;2632;254;Vertical Blank;;
19113043;65,749;2633;2;Joystick;0;247
19118442;65,750;2633;254;Vertical Blank;;
19121828;65,770;2634;2;Joystick;0;247
19125700;65,770;2634;254;Vertical Blank;;
19129280;65,790;2635;2;Joystick;0;247
19132959;65,790;2635;254;Vertical Blank;;
19134817;65,809;2636;2;Joystick;0;247
19140218;65,810;2636;254;Vertical Blank;;
19143678;65,830;2637;2;Joystick;0;247
...
< After some more action on screen and joystick movements taking a second screenshot of the
currently displayed image as file terra2_!.bmp >
19989522;68,149;2753;254;Vertical Blank;;
19996781;68,169;2754;254;Vertical Blank;;
19996781;81,086;2754;3;Screenshot saved;bin/scshot/terra2_1.bmp;
20004040;81,628;2755;254;Vertical Blank;;
20011298;81,648;2756;254;Vertical Blank;;
...
< Ending the emulation process by pressing 'ESC' to exit the emulator >
20483133;82,948;2821;254;Vertical Blank;;
20490393;82,968;2822;254;Vertical Blank;;
20491209;83,118;2822;1;Keypress;15131;'ESC'
20491219;83,118;2822;1;Keypress;15131;'ESC'
20491229;83,118;2822;1;Keypress;15131;'ESC'
20491239;83,118;2822;1;Keypress;15131;'ESC'
20491249;83,118;2822;1;Keypress;15131;'ESC'
20492159;83,121;2823;255;Emulation stopped;;
```