Design Decisions in Emulator Construction: A Case Study on Home Computer Software Preservation

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

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

ABSTRACT

Preserving software is widely recognized as a far more complex task then preserving static data. Emulation is usually the chosen preservation action to enable the execution of programs of obsolete systems. In this work we show how software extracted from obsolete media was preserved by developing an emulator. We explain the reengineering work involved and the design decisions made as well as the options for data injection into and extraction from the emulated environment.

In previous work, data and programs stored on audio tapes were extracted and the resulting audio files were transformed into digital objects. The objects retrieved were mainly programs, requiring emulation for execution. As no emulator for the original system previously exists, 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.

1. INTRODUCTION

Preserving digital objects for a long term does not only concern preserving static data like pictures or text documents. For a wide range of digital objects not only data has to be preserved but the actual rendering process of data is significant. This is especially true, when a digital object has to be continuously rendered, as in the preservation of software. But also whole business or scientific processes need to be stored for a long term to be able to exhume them at a later time and run them in a changed environment. One of our main concerns for preserving processes is keeping them accessible and the software originally used executable.

Preserving software across rendering environments, i.e. executing the software on a platform it was not designed for, is usually solved by executing the software in an emulator emulating the hardware of the platform and running on a different host platform. While the advantage of a hardware emulator is that it can potentially run all software designed for the hardware it emulates, it is a quite complex task to build an emulator [5] and involves expert knowledge about the hardware specifications of the original system. It is also necessary to not only emulate the hardware, but also to provide methods for providing input to the emulated system, either in the way of interaction with the system by using keyboards or other input devices, but also by injecting data from files into the system. Extracting data for usage in the host environment is also an important issue not tackled by most emulators today.

As previously published in [7] we extracted data encoded in audio wave forms from cassette tapes. Almost all the data extracted was programs written in a dialect of the computer language BASIC. The programs where converted from their original binary form to source code in readable text format. As preserving the source code is only the first step of preserving the programs, research on potential rendering environments was carried out. In this paper we now demonstrate the development of an emulator for the system and show which design decisions have to be made and what problems one has to deal with even with a fairly simple computer architecture. We show what one must consider so an emulator developed can be used for digital preservation by providing functionality for injecting data into the emulated environment and extracting data for use on the host system.

This paper is structured as follows. First we provide related work relevant for this paper. In Section 3 we examine the view-path of the original system and provide information on how the different components involved interact. We present how we implemented the view-path in an emulator in Section 4. In Sections 5 and 6 we explain the reengineering work necessary for data exchange between the emulated environment and the host environment. We explain what choices we were given to solve certain problems and what design decisions were taken for implementing the functionality, keeping digital preservation in mind. Next, we show

how the image rendered by the emulator can be evaluated against the original system and other alternative rendering environments. In Section 8 we then discuss other possible preservation actions besides emulation on different levels in the view-path. Finally we show our conclusions and give an outlook to future work.

2. RELATED WORK

Preserving software for obsolete computer platforms has to be performed in two steps: transferring the programs to a non-obsolete environment and executing the programs in a different rendering environment.

In [7] we demonstrated the documentation of the output formats of an early home computer system (the Philips Videopac G7400 utilizing an extension that allows the system to execute BASIC software and store and retrieve data from and to cassette tapes). We showed that even for comparatively simple systems a lot of steps are necessary to reengineer the data formats. In a case study shown in the same paper, we transferred data from various old tapes to a non-obsolete environment using a tool we developed. The data was then migrated to non-obsolete formats using signal processing techniques to convert the analog sound signal to binary data. While static data like images can then be opened in current viewers, software in BASIC source code format converted to readable text can not be executed in a current environment without further preservation actions.

Source code is one of the significant properties of software that allow us to migrate the software for preservation purposes [12]. For interpreted program languages like BASIC (compared to program languages where source code is compiled to executable software) the source code is equal to the executable software given the availability of a suitable interpreter.

Diessen et. al. describe in [18] the view-path as „a full set of functionality for rendering the information contained in a digital object". The view-path contains the hardware and all other secondary digital objects needed to render an object and also to run a certain piece of software. As an example, to run a simple JAVA program printing 'Hello World' on screen, a JAVA virtual machine, different libraries, an operating system running the virtual machine and the hardware to execute the operating system are needed. In OAIS [9] terminology the view-path contains the Access Software used to render the digital object as part of the representation information and all secondary digital objects needed to execute the Access Software.

Different strategies for preserving digital objects exist, the major ones being migration and emulation. Migration, which involves altering the original format of the digital object ([11]), is the main strategy for preserving static content. In [14] Rothenberg explains that the emulation of the logical behavior of a computer system should be sufficient on a relatively abstract level. Lorie differentiates between the archiving data and archiving program behavior. While the first can be done without emulation, Lorie argues that it cannot be avoided for the latter [10].

Execution in an emulation environment necessitates expert knowledge about utilization of the original environment and creates issues like data exchange between the emulation environment and the host environment [13]. Although the second issue was partially solved in the emulator Dioscuri, created specifically for digital preservation [16], it is still far from being a standard in current emulators.

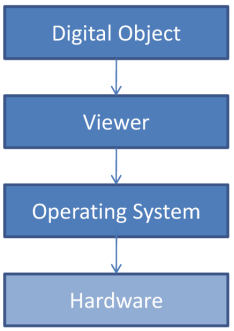

Figure 1: view-path for a generic system.

The European research project KEEP[1] performs research in legal aspects of emulation as well as develops a common platform for emulators (*Emulation Virtual Machine*) to „Keep Emulation Environments Portable". Some of the legal issues raised by KEEP also apply to the development of the emulator in this paper.

In [4] 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. Comparing renderings of the same digital objects in different environments is usually done manually by a human observer. A case study to compare different approaches to preserve video games, with one of the approaches being emulation, was reported in [6] on a human-observable and thus to some extent subjective level. In [8] we presented case studies of interactive objects comparing the rendering outcomes of different rendering environments.

In this paper we show how the concept of a view-path can be applied to an obsolete system. We explain how software for the system is preserved using emulation by implementing an appropriate emulator. Digital preservation in mind we discuss the design decisions that have to be taken and we show discuss how the emulation results can be compared.

3. 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.

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 as shown in Figure

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

Figure 2: Philips Videopac+ G7400 with plugged in Philips C7420 Home Computer cartridge.

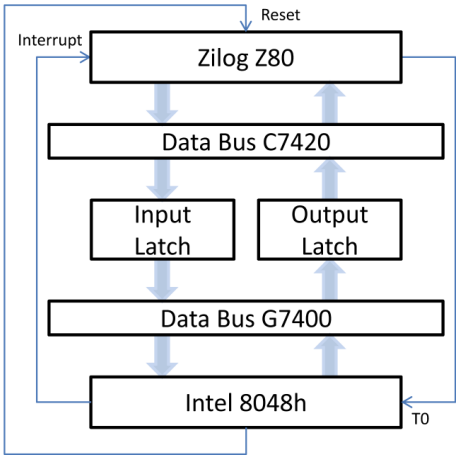

Figure 4: Communication flow between G7400 system and C7420 cartridge.

1. 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 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 2). Details about the history of the system can be found in [7]. 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[2]. Figure 3 shows a block diagram 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 4.

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 5. 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.

4. IMPLEMENTING THE VIEW-PATH IN AN EMULATOR

As we did not want to start working on the G7400 and C7420 emulator from scratch, the existing open source emulator

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

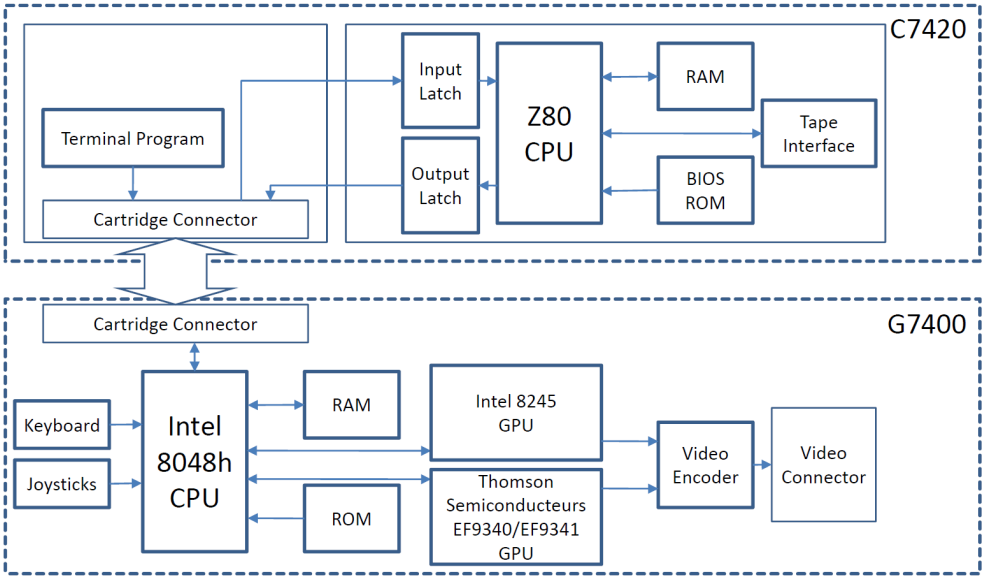

Figure 3: Block diagram of C7420 Home Computer cartridge and Philips Videopac+ G7400 system. Connection between cartridge and system is done using the cartridge connector. CPU - Central Processing Unit, GPU - Graphics Processing Unit, RAM - Random Access Memory, ROM - Read Only Memory.

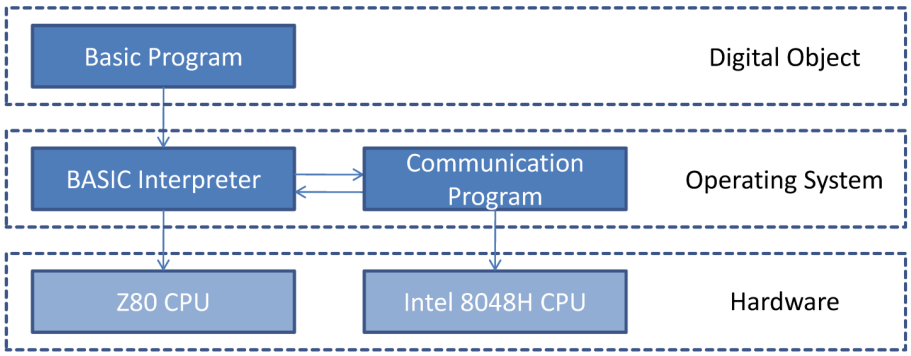

Figure 5: view-path for program execution on G7400+C7420.

O2EM[3] was used as a starting point. O2EM initially was written in 1997 as an emulator for the video game system Magnavox Odyssey2, 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 without changes.

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 [4] 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 by van der Hoeven et. al. in [17]. 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.

[3]O2EM - Sourceforge: `http://o2em.sourceforge.net/`
[4]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 to check 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 6.

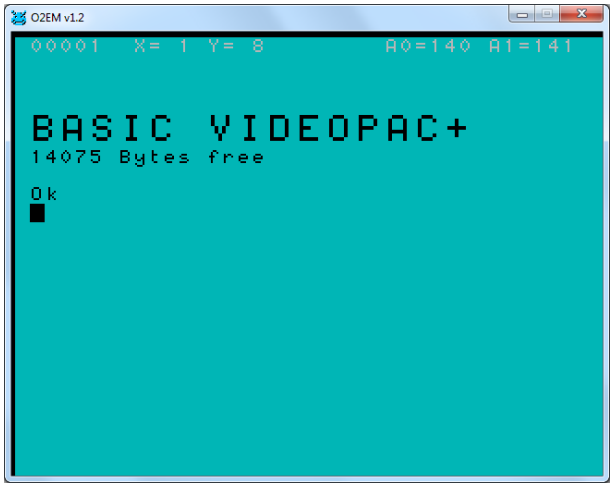

Figure 6: Start screen of C7420 Home Computer cartridge on O2EM emulator.

5. DATA INJECTION

After establishing the emulation of C7420 Home Computer cartridge, the next step is to enter data into to the emulated environment. Three options for data input are available on the original system. Below we describe these three options and the challenges they present for emulation.

5.1 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 a modern keyboard has its own key for 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:

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.

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 compares 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.

5.2 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, no additional actions had to be performed.

5.3 Files

Besides data injection through control devices, the C7420 supports the loading of files from an audio signal connected through a microphone jack. In this section we will show different possibilities of loading a file into memory.

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 [7] 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. 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 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.

6. DATA EXTRACTION

While data injection is an important issue to execute and interact with software in the emulated environment, for some digital preservation applications it is 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. The methods for data extraction we implemented in the emulator are listed below.

6.1 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 As with 'CLOAD' a function to directly write a BASIC program to disk is provided. As the format of storing BASIC programs in the memory 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.

6.2 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:

Extraction from C7420 screen buffer 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.

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.

We decided to go with the more generic version and extract the data directly from the video memory of the emulator. Depending on the operating system different routines for copying data to the clipboard has to be implemented. The data that is extracted is in ASCII, 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. As not all the characters have the same code representation as in a current ASCII format table, a conversion for certain characters is performed while copying the data.

6.3 Screenshots

Screenshots of the emulated environment can be used e.g. to compare emulation results with the original environment. Extracting data in the form of screenshots can be done using one of three different methods on different levels of the emulation:

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 [7]. Using this method it is possible to compare the principal rendering inside the emulation environment. It can not be checked if the emulator renders the image correctly on the host system.

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.

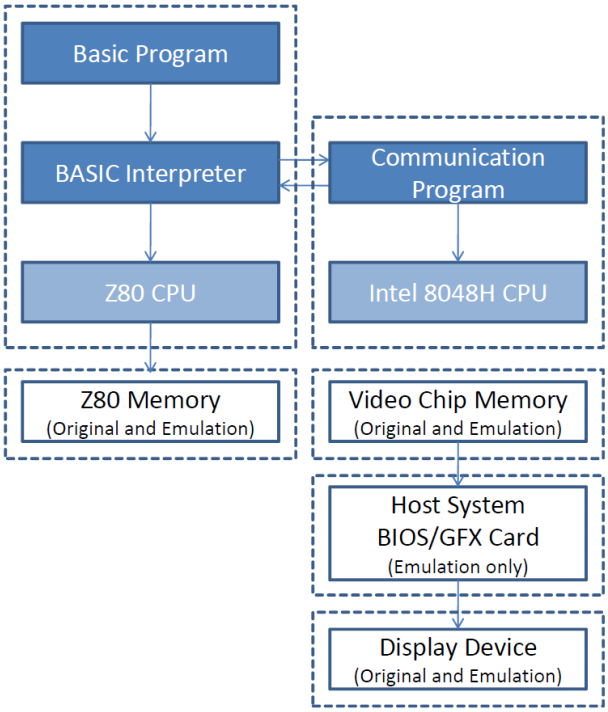

Figure 7: Different renderings in the view path of the C7420 Home Computer cartridge.

The resulting screenshots can be used e.g. to compare rendering results of different rendering environments for preservation planning purposes as described in [8].

7. EVALUATING RENDERING RESULTS

To select the best preservation solution for a certain scenario, it is necessary to compare all available preservation actions. In [2] Becker et.al. describe a preservation planning approach based on comparing significant properties of digital objects before and after applying a preservation action. While on migrated data the digital object before and after migration can be compared, the task is different when dealing with emulation. Instead of comparing the digital object, renderings of the digital object in different rendering environments are compared.

Results of rendering can be compared on different levels. Figure 7 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.

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 [8] we demonstrate how the rendering results of different rendering environments can be compared by using the characterization language XCL as described in [3] 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 supports 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 is currently not supported.

8. OTHER PRESERVATION ACTIONS

Executing programs using emulation on a hardware level is only one of the different alternatives that can be used for preserving software. Figure 8 shows the different levels in the execution view-path of the C7420 and also lists preservation action strategies for each of the levels.

8.1 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

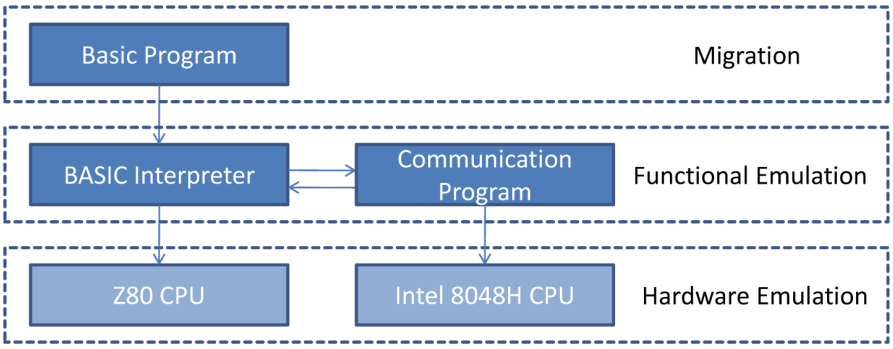

Figure 8: Preservation actions for different layers of view-path.

to implement an emulator is quite high, even though this method is probably the most accurate one.

8.2 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 can be controlled.

8.3 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 [1].

9. CONCLUSIONS AND FUTURE WORK

In this paper we described how an emulator for an early home computer system was developed. 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 we were trying to keep special digital preservation requirements in mind when implementing certain features like extracting data from the emulation environment. We showed how different rendering environments can be compared and on what levels specifically for the machine in the case study, and how this either is already supported or would have to be implemented in the future. Finally, we discussed other options for preserving software for the home computer system evaluated like source code migration and high level emulation in the form of a BASIC interpreter.

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 what design decisions arise during the development of an emulator especially when having a long term approach in mind and not only a short term solution for executing software of a recently obsolete system.

The implementation of the emulator was considered a success as the digital objects migrated previously from audio tapes could be injected and successfully executed in the emulated environment. The case study also showed 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.

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 [15]) keeps the emulator executable.

For future work we plan to implement other strategies for preserving the C7420 software as listed in Section 8. A comparison of the different strategies on different levels of the view path will be performed to show how the quality of emulation can be objectively measured. The results of the work carried out on the fairly simple C7420 Home Computer cartridge system will then be applied to more complex systems.

10. ACKNOWLEDGMENTS

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

11. REFERENCES

[1] E. Ashcroft and Z. Manna. *The translation of 'go to' programs to 'while' programs*, pages 49–61. Yourdon Press, Upper Saddle River, NJ, USA, 1979.

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

[3] C. Becker, A. Rauber, V. Heydegger, J. Schnasse, and M. 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`.

[4] A. Bonardi and J. Barthélemy. The preservation, emulation, migration, and virtualization of live electronics for performing arts: An overview of musical and technical issues. *J. Comput. Cult. Herit.*, 1(1):1–16, 2008.

[5] S. Granger. Emulation as a digital preservation strategy. *D-Lib Magazine*, Vol. 6 (10), 2000. `http://www.dlib.org/dlib/october00/granger/10granger.html`.

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

[7] M. Guttenbrunner, M. Ghete, A. John, C. Lederer, and A. 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.

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

[9] ISO. *Space data and information transfer systems – Open archival information system – Reference model (ISO 14721:2003)*, 2003.

[10] R. Lorie. A project on preservation of digital data. *RLG DigiNews*, Vol. 5 (3), 2001. `http://www.rlg.org/preserv/diginews/diginews5-3.html#feature2`.

[11] D. B. Marcum. The preservation of digital information. *The Journal of Academic Librarianship*, 22(6):451 – 454, 1996.

[12] B. Matthews, B. McIlwrath, D. Giaretta, and E. Conway. The significant properties of software: A study. JISC Study, 2008. `http://www.jisc.ac.uk/media/documents/programmes/preservation/spsoftware_report_redacted.pdf`.

[13] T. A. Phelps and P. Watry. A no-compromises architecture for digital document preservation. In *Proceedings from 9th European Conference on Research and Advanced Technology for Digital Libraries*, pages 266–277, 2005.

[14] J. Rothenberg. *Using Emulation to Preserve Digital Documents, Tech. Rep.* Koninklijke Bibliotheek, 2000.

[15] J. Slats. Emulation: Context and current status. Tech. Rep., 2003. `http://www.digitaleduurzaamheid.nl/bibliotheek/docs/white_paper_emulatie_EN.pdf`.

[16] J. van der Hoeven, B. Lohman, and R. Verdegem. Emulation for digital preservation in practice: The results. *International Journal of Digital Curation*, Vol. 2 (2):123–132, 2007.

[17] J. van der Hoeven and H. van Wijngaarden. Modular emulation as a long-term preservation strategy for digital objects. In *5th International Web Archiving Workshop (IWAW05)*, 2005.

[18] R. 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`.