Characterization of CD-ROMs for Emulation-Based Access

Klaus Rechert, Thomas Liebetraut, Oleg Stobbe, Isgandar Valizada
University of Freiburg
Hermann-Herder Str. 10
79104 Freiburg, Germany
{firstname.lastname}@rz.uni-freiburg.de

Tobias Steinke
German National Library
Adickesallee 1
60322 Frankfurt, Germany
t.steinke@dnb.de

ABSTRACT

Memory institutions have already collected a substantial amount of digital objects, predominantly CD-ROMs. Some of them are already inaccessible with current systems, most of them will be soon. Emulation offers a viable strategy for long-term access to these publications. However, these collections are huge and the objects are missing technical metadata to setup a suitable emulated environment. In this paper we propose a pragmatic approach to technical metadata which we use to implement a characterization tool to suggest a suitable emulated rendering environment.

General Terms

Infrastructure opportunities and challenges; Frameworks for digital preservation; Preservation strategies and workflows.

Keywords

Emulation Characterization Tools

1. INTRODUCTION

Memory institution have already accumulated a substantial amount of digital artifacts. For instance, the German National Library (DNB) has been collecting German publications on various data carriers since commercial publication began in the end of the seventies. There are all kinds of data carriers for many different computer systems in the magazines of the DNB. The estimation of the number of stored digital carriers is about 500000. It is not exact as the cataloging of these publications was not consistent in the early years.

Currently, a user in the DNB's reading room can order a data disk via catalogue, which will then be prepared for usage in a virtual drive. These CD-ROMs, however, need to run in the DNB's current desktop computer environment (currently Windows 7). In some cases the CD-ROMs still work but other CD-ROMs fail or may fail soon. Hence, the current access workflow lacks a future proof strategy. Furthermore, similar collections of digital publications with a dynamic character, for instance web archives, also need a future-proof access solution.

Emulation has been shown to be a useful and flexible approach to access legacy software collections [2, 15, 4]. However, in libraries and their classical reading-room scenarios, the development and maintenance of tailored emulation solutions is difficult. Especially for very large collections, analyzing and evaluating each object individually is not a feasible solution [13]. Based on these findings, the EMiL [1] project aims to develop an environment for library reading rooms that addresses the challenges of complex objects with a flexible emulation framework. The primary focus for the DNB is on providing access to multimedia CD-ROMs of the 1990s and 2000s, but the EMiL system might be used for other collections later on as well. Automation is essential, since the huge number of objects prevent manual handling of each object, in particular in view of the determination of technical metadata required for emulation.

The default scenario for the DNB is rather simple: Users in the DNB's reading room may use specific workstations for research purposes. If any of the catalogue's entry is a multimedia publication originally published on a data carrier, a click on a link initiates the transfer of the data carrier's digital image from the digital preservation system to the EMiL system. Then, the user gets access to the content of the publication using a suitable emulation environment. No additional interactions or decisions by the user should be required during this process. Hence, for this rather simple workflow it is crucial that the EMiL system is able to automatically determine the "best" available emulation environment.

Therefore the development of a trustworthy characterization tool and a flexible management of emulation environments are key tasks in the project EMiL. In this paper, we present design, implementation and evaluation of a CD-ROM characterization tool as well as necessary technical metadata and technical components.

2. PROBLEM DEFINITION

As outlined in the previous section, EMiL aims at re-enacting a multitude of digital objects using emulation technology.

[1]Emulation of Multimedia-objects in Libraries, `http://www.multimedia-emulation.de/`

However, these objects cannot run without suitable technical environments. These environments typically include a (virtual) computer hardware, an operating system and sometimes additional applications, all of which form the complete emulation or rendering environment for a digital object. Due to the many ways hardware and software can be combined, not any such environment can be used to render a specific object. Instead, the EMiL software framework has to find environments suitable for a specific object based on the technical aspects of this object.

When objects like CD-ROMs have been acquired and ingested into a memory institution's catalog, typically descriptive metadata has been gathered and associated with the object. Even though these metadata entries may also describe technical aspects of the object, (re-)using this data to identify and prepare a suitable rendering environment is difficult. Listing 1 shows an example of technical system requirements of a CD-ROM published in the late 90s. These technical descriptions were originally designed to guide potential buyers, but from today's viewpoint most of the information is irrelevant (computing power and memory and disk sizes have risen by magnitudes). A much bigger problem is that there was no standard or schema in describing system requirements; hence, using this information as technical metadata in an automated way is difficult.

Listing 1: System requirements as posted on the CD-ROM box or booklet.

```
Windows 3.1x, 95, NT 4.0
IBM Compatible PC
80486 Processor or higher
Minimum: 8 MB Memory (RAM)
10 MB free hard disc space
Minimum DIN/ISO 9660 CD-ROM drive
MSCDEX 2.21
Mouse
```

The KEEP Emulation Framework [11] provides file characterization in order to determine a ViewPath [14]. Based on a file format, a rendering application is chosen; using the application, an operating system and finally an emulator is determined [1]. The information required is stored in the framework's database. This file-based approach can be used very efficiently to cope with certain types of digital objects (digital pictures, text documents, etc.). File types can be automatically characterized which allows to select the corresponding viewer software.

This single-file characterization approach falls short on more complex objects. Object types that consist of a collection of different files and formats that are wrapped into a single container format (e.g. ZIP or ISO9660 images) cannot be classified with a single file format. Today's file characterization tools may recognize the container format, which is, however, of limited use for a useable access strategy. Even if the container's content is analyzed file by file, this approach can only provide a file-by-file re-enactment and loses the collection-type character of the object. This is true especially for interactive CD-ROM publications (e.g. multimedia productions, interactive educational content, encyclopedias etc.) where just providing access to individual image and text files is not sufficient if the original digital object provides a rich application that guides the user through the available material.

Even if an object's "viewer" software or its ViewPath is known, this information is neither unambiguous nor sufficient. Usually, executing a specific application is not possible without prior installation, neither is creating such an environment in an automated way. Similarly, an emulator system required to run such software needs to be set up and configured properly. Furthermore, there exists an (almost) unlimited number of potential software environments satisfying an object's technical requirements. Hence, in a practical scenario an image archive with associated metadata is required which provides means to search for a matching environment.

In earlier work we have presented workflows for manual ingest of CD-ROMs, i.e. the user individually chooses from a list of available rendering environments [9]. If no suitable environment is available, a new environment has to be built, for instance, by installing additional software. The result of the CD-ROM ingest workflow is a semantic link between a digital object and the technical description of an emulated system environment able to render a certain object. This approach is viable and useful for instance in the digital art domain since the rendering quality can be evaluated. However, good knowledge of each CD-ROM is required to be able to choose or create an appropriate rendering environment. Traditional memory institutions, however, hold large collections of different digital objects. A manual (re-)ingest of these objects is labor-intensive, as this workflow requires an in-depth analysis of each object to determine its rendering environment. For a majority of these objects (e.g. for supplementary CDs) the total costs would probably not be justifiable.

Our goal is to support and automate this process as much as possible. From a set of available rendering environments, only a few are suitable to render a given object. This matching process requires a characterization tool as a prerequisite. Furthermore, with respect to the memory institution's large collections, gathering information on required environments covering (most of) the collection's objects is necessary to build a comprehensive image archive to be chosen from.

3. TECHNICAL METADATA

The problem definition and requirements show that technical metadata describing the capabilities of the rendering environment is necessary. Several models for describing a computing environment have already been developed. The PREMIS data dictionary [12] provides a semantic *environment* entity to describe rendering environments, which has been recently reworked and extended to improve expressiveness, in particular in emulation use-cases [7, 6]. Similarly, The Trustworthy Online Technical Environment Database – TOTEM [8] provides a comprehensive data model to describe environments in great detail. Furthermore, tools have been proposed to determine environment information, for instance capturing a digital object's runtime dependencies [5].

The major trade-off, from a practical perspective, is choosing between the level of detail and the ability to re-use environments with new emulators, different objects or usage contexts. A small number of generic environments to render a large number of digital objects is preferable, as it reduces the burden of preservation planning. With detailed and very specific environment descriptions, re-use of environments becomes less likely. Also the complexity of associated tools matters. For instance, with more details, it becomes more difficult to design a matching algorithm that links rather generic emulator software with specific hardware requirements.

For this reason, we pursue a constructive approach, with less focus on formal modeling yet. The primary purpose of our technical metadata is to describe environments to render digital artifacts but also to describe environments such that preservation planning for emulation-based preservation strategies become possible. In order to integrate emulation related metadata seamlessly into a memory institution's existing preservation systems, a formal model and possibly encoding in a standard metadata language (e.g. PREMIS) may be required. This, however is left for future work.

A complete set of metadata required to re-enact a virtual environment is called an emulation environment and can roughly be divided into two parts, a hardware environment and a software environment. The former describes the technical features of an (emulated) computer system, while the latter describes the software utilizing the computer hardware, usually in form of a virtual disk image. A digital artifact typically poses (abstract) requirements on its rendering environment, e.g. a Windows operating system (Windows 95 or newer) with a set of applications installed and sound support enabled. However, specific requirements on the sound hardware are rare. Hence, an artifact is (tightly) linked to one or more software environments, but in general indifferent to the underlying hardware environment (as long as appropriate functionality is provided). The hardware environment, on the other hand, is selected based on the aggregated software environment's requirements and hardware's capabilities. To use hardware features, usually driver software needs to be installed and/or the operating system requires configuration. A software environment thus limits the choice of useable hardware configurations.

The separation of hardware and software environment descriptions allows to change the underlying (emulated) hardware without reevaluating (a huge number of) digital artifacts. This, makes it easier and more cost effective to cope with technological changes. Operating systems (in particular old ones) do not change their hardware requirements or their features over time. Hence, a particular software environment description can be considered as constant (if complete), especially regarding the hardware interfaces used. Even though the link between a specific software and hardware configuration is also stable in the short run, emulators are also prone to a software life-cycle and will be technically obsolete at some point. Then, if new emulation software is required, a new hardware environment description is created, describing individual emulated hardware components. If the connecting interfaces between hardware (hardware component description) and software (driver and operating system configuration) are made explicit in the technical metadata, all affected software environments can be determined, and the search for new emulators can be guided. If no perfect match is found, the necessary adaptions of affected software environments can be predicted in an automated way. This allows to focus preservation planning activities on monitoring the links between software and hardware environments.

Listing 2: General structure of a software environment description.
```
<swEnvironment>
 <id>..</id>
 <description>...</description>
 [...]

 <binding>
  <url>nbd://my.host?exportname=disk.img</url>
  <md5sum>...</md5sum>
  [...]
 </binding>

 <systemConfiguration />

 <softwareCollection />

</swEnvironment>
```

3.1 Software Environment

A software environment description's primary purpose is to describe a computer system's software setup. In the case of emulation workflows, these setups can be found in the form of disk images, representing a virtual hard disk to be used with an emulator. Listing 2 illustrates an `swEnvironment`'s general structure and main elements.

The first important element is a data `binding` definition, referring to the location of the disk image. Data bindings define volumes that can later be used to emulate a medium but only represent the bare data or data source. This element may contain further information, such as fixity information, format and file-system of the container.

The second main element of a software environment describes its relation to a potential hardware environment description. The `systemConfiguration` tag describes if there is an operating system installed and configuration of (required) hardware dependencies. The `osConfiguration` element describes the operating system as a `swComponent`, in particular with its rendering capabilities. For instance, a Windows 98 installation is able to render (execute) Win32 executables (`x-fmt/411`) (native format) but may also run Win16 (`x-fmt/410`) and MS-DOS executables (`x-fmt/409`) (import formats). In our implementation, we currently use PRONOM IDs (PUID) [3] to specify the supported rendering capabilities, if available. While the format specifications can be in any form, we chose PUIDs both because of its simple scheme and to be able to (re-)use its file format and software descriptions. While the PRONOM registry is far from complete, in particular with respect to software descriptions, it provides a viable (and well known) starting point, which could be quickly extended with a growing emulation community.

In order to describe a software's rendering capabilities, we chose a slightly different structure. While PRONOM uses *create* and *render* categories to associate file formats to a software description, we distinguish between *import*, *export* and *native* formats. For our use-cases it is helpful to be able to choose between applications rendering their native format and applications which are able to render a format to a certain extent. Software usually has a native data format that can be rendered without losing information. At the same time, it often allows some sort of interoperability with other software and thus provides at least partial support for other formats. For instance, OpenOffice.org natively supports OpenDocument but is also able to open Word documents, even if the rendering quality may be imperfect. This concept also provides a path to migrate independent data objects to other formats, e.g. to migrate a Word Perfect file to current Office Open XML.

As a specific software environment, in particular the embedded operating system, does not run on any hardware, the system configuration section also contains information about specific hardware configuration. An operating system's configuration and its (additionally installed) drivers define a set of "expectations" on the underlying hardware system. This `hwConfiguration` description can then be matched against hardware environment descriptions. Listing 3 illustrates a simple system configuration of a typical Windows 98 installation.

Listing 3: A software environment's system configuration description.

```
<systemConfiguration>
 <osConfiguration>
  <swComponent id="x-sfw/37">
   <description>Windows 98 SE</description>
   <nativeFormats>
    <fmt puid="x-fmt/411" />
   </nativeFormats>
   <importFormats>
    <fmt puid="x-fmt/410" />
    <fmt puid="x-fmt/409" />
   </importFormats>
  </swComponent>
 </osConfiguration>

 <hwConfiguration>
  <hwcomponent class="audio" device="sb16">
   <param key="irq" value="5"</param>
   <swComponent id="x-driver/99 />
  </hwcomponent>

  <hwcomponent class="storage" device="piix3"
      id="ide_1" />
  <hwcomponent class="storage" device="cdrom">
   <param key="controller" value="ide_1"</
       param>
 </hwConfiguration>
</systemConfiguration>
```

The metadata associated with a `softwareCollection` extends its rendering capabilities, i.e. the type of file that can be used within this software environment. Software metadata could be kept directly in the software environment description but it is preferable to use only references to the associated software archive or technical registry. This way, software properties, in particular, license information, where conditions may change over time, can be centrally maintained and evaluated on-the-fly.

Listing 4: Excerpt of a software collection description.

```
<softwareCollection>
 <swComponent puid="x-sfw/68">
  <description>
   WordPerfect Office V.11
  </description>
  <nativeFormats>
   <fmt puid="x-fmt/44" />
   [...]
  </nativeFormats>
  <importFormats>
   <fmt puid="fmt/125" />
   [...]
  </importFormats>
  <exportFormats>
   <fmt puid="fmt/97" />
   [...]
  </exportFormats>
 </swComponent>
 [...]
</softwareCollection>
```

3.2 Hardware Environment

The basis for running any software environment is either a physical or an emulated computer system. The hardware environment description is quite similar to the software environment, as it defines a set of available hardware interfaces that make up a computer system.

A hardware environment is typically defined through its basic architecture (e.g. x86 compatible PC) and its specific hardware features. A list of list of `hwComponents` describes available hardware components, which a software environment is able to utilize. Each `hwComponent` represents specific hardware and describes configuration options, if necessary. For instance, Listing 5 lists the audio cards supported by the QEMU i386 system emulator. The system environment's `hwConfiguration` from Listing 3 then chooses one of these cards (Sound Blaster 16) and configures it accordingly.

The source of this environment description is usually the emulator's manual, describing the emulator's capabilities. By making the list of supported hardware explicit, different emulators can be compared. For instance, Virtual Box supports only three sound cards (Sound Blaster 16, AC97 and Intel HDA), VMWare supports only Sound Blaster and Intel HDA cards. The same applies to other hardware components like network and graphics cards and storage controller. Based on this information, one can produce guidelines to guide the development of software environments, in particular operating system installation and its configuration. Hence, the software environment from Listing 3 should be compatible with QEMU, VirtualBox and VMWare with respect to audio hardware requirements.

Every hardware environment available in the EMiL framework is backed by an emulation component implementation (cf. [10]). Each component is able to parse a `systemConfiguration` element and translate its requirements into native configuration for a specific emulator software. Table 1 shows an overview of EMiL's supported emulators.

Listing 5: System description excerpt as featured by a current QEMU emulator.

```
<hwEnvironment>
 <id>...</id>
 <name>QEMU i386</name>
 <architecture>x86</architecture>
 [...]

 <hwComponents>
  <hwcomponent class="audio" device="sb16">
   <param key="irq" value="5"</param>
   <param key="irq" value="7"</param>
   [...]
  </hwComponent>
  <hwComponent class="audio" device="ac97"/>
  <hwComponent class="audio" device="es1370"/>
  <hwComponent class="audio" device="hd"/>
  [...]

  <hwComponent class="storage" device="piix3"
      id="ide_1" />
  <hwComponent class="storage" device="cdrom">
   <param key="controller" value="ide_1"</param
      >
  </hwComponent>
 </hwComponents>
</hwEnvironment>
```

4. DESIGN & IMPLEMENTATION OF A CD-ROM CHARACTERIZATION TOOL

Based on the DNB's reading room scenario, the access workflow starts by requesting an object from the library's catalog. The goal is now to determine a suitable environment for this object automatically. To this end, a software environment suitable for rendering the object at hand has to be determined. This matching process has to be based on the requirements of the object and its expectations about the environment it runs in, e.g. a certain operating system version or support for the file type of the digital object.

Table 1: List of EMiL standard environments

Operating System	Arch	Emulator	Alt. Emulators
MS-DOS	x86	QEMU	Dosbox, VBox(VX)
MS Windows 3.11	x86	QEMU	Dosbox, VBox(VX)
MS Windows 9x	x86	QEMU	VBox(VX)
MS Windows XP	x86	QEMU	VBox(VX)
Linux i386	x86	QEMU	VBox(VX)
Apple II	MOS Tec	PCE	vmac-mini, MESS
Apple System 7	m68k	BasiliskII	MESS
Apple System 8	ppc	Sheepshaver	MESS
Apple System 9	ppc	Sheepshaver	
Amiga	m68k	x-uae	MESS
C64	MOS Tec	VICE	MESS
Atari	m68k	hatari	MESS

4.1 Building an Image Archive

The test collection for the EMiL project consisted mainly of interactive, runnable software objects rather than bare document formats (like images or Word documents). CD-ROMs were usually made for the mass market and are therefore mostly self-contained, i.e. if additional software was required, which is not already part of the CD-ROM image (Quicktime or Acrobat Reader are popular examples). Hence, the most important feature for a software environment is to run applications made for a specific operating system and computer architecture.

In order to provide suitable runtime environments, firstly, a list of necessary standard environments has to be compiled. These so-called standard environments provide a basic operating system installation and configuration, so that there is least one hardware environment (i.e. emulator) satisfying the resulting software environment's `systemConfiguration`. We have chosen executable file formats as our primary identifier for the gathering runtime requirements.

Executable file formats, however, are usually not very precise, because they are designed as a very basic interaction point between the operating system and CPU code. For instance, the portable executable (PE) file format used on Windows operating systems has been stable since Windows NT 3.1 and Windows 95 until the recent introduction of the 64-bit architecture and is still today used for non-64-bit binaries on Windows. However, a PE binary designed for Windows 95, is not guaranteed to run on Windows 8 and vice versa. To cope with this dilemma, we implemented a second classifier to the matching mechanism. Depending on the file's timestamp we can distinguish between different epochs of an operating system or software package and thus "authentic" software environments.

Using these two classifiers, the binary file format and the epoch, we analyzed all images in the sample collection and produced a set of operating system and architecture (cf. Table 1). Column 3 and 4 of the table also show emulators that provide the technical features to emulate these environments. This classification step showed that the object collection provided by the DNB can be re-enacted using comparatively few environments, in particular if compared to the large number of objects.

Once the basic set of software environments required to access the digital objects has been determined, these software environments have to be created. This process includes the installation of an operating system, configuring it accordingly and installing required drivers for e.g. network or sound support. As a hardware basis, a typical configuration of the corresponding epochs computer systems is used, respectively. The result is a freshly made software environment with its characteristics known and described with a complete set of technical metadata. These software environments, including the disk images of the installed operating system, are then ingested into the project's image archive. Environments can then be used to be matched against the analysis results of individual objects. As they were created according to the requirements gained from the previous analysis of the whole collection, we can ensure that all objects find a matching software environment or we can determine precisely which objects lack specific features not (yet) included in the image archive.

4.2 Managing Software

For some objects in the sample collection, the process described previously failed due to our focus on executable binary formats. For instance, one object contained only a single PPT file with all other media directly embedded. This object, obviously, contains no executable binary format and

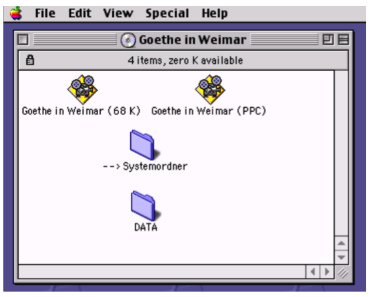

Figure 1: Content of a hybrid CD-ROM opened on an Apple Macintosh computer system.

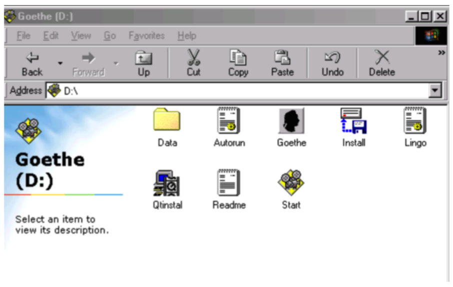

Figure 2: Content of the same CD-ROM opened on a Windows computer system.

thus does not require a specific operating system per se. However, the PPT file requires another piece of software that is shipped neither with the operating system nor the digital object, namely Microsoft PowerPoint.

To add to the existing base environments and provide auxiliary software environments that provide support for further file formats, it is possible to create derived environments within the image archive. To facilitate this task, a separate software archive can be provided that contains installation media for several additional software packages, e.g. MS Office. Accompanying these installation media, there is also metadata describing the additional native, import and export file formats provided by the software. The installation process, then, is similar to the re-enactment of a digital object: a media container and file system analysis yields the required software environment that this software can be installed on. After the installation process, the modified image can be ingested back into the image archive with the updated metadata as a derived environment based on the original base environment.

Internally, only the modified data blocks of the hard disk image are stored while the original blocks just reference the original base image. Therefore, the derivative remains dependent on the availability of the original environment. Due to this link, it is also possible to precisely tell which software environments may require looking into, once a base environment changes for some reason (e.g. because new emulators require new drivers to be installed).

Using these newly created derived software environments, the object only containing a PPT file can be started by searching for the respective PRONOM IDs. Now, the derivative with MS Office will be a matching result and the object can then be rendered using PowerPoint. Having such software available individually also provides means for ensuring the license limits of certain software is not exceeded. For instance, a memory institution may have a large number of operating system licenses but only a very limited number of special-purpose software that not everyone needs. Rendering all objects that do not require that special-purpose software using software environments that do not contain them allows for a larger number of parallel users.

4.3 On-the-fly Object Characterization

Once the image archive contains software environments and corresponding metadata, we are able to match the requirements of a digital object requested by the access workflow against the software environments and find a suitable rendering environment. As the original text-based requirements cannot be used to automatically and reliably find a rendering environment, we have to provide a different matching process that relies on the object's actual contents. This matching process is outlined in Fig. 3.

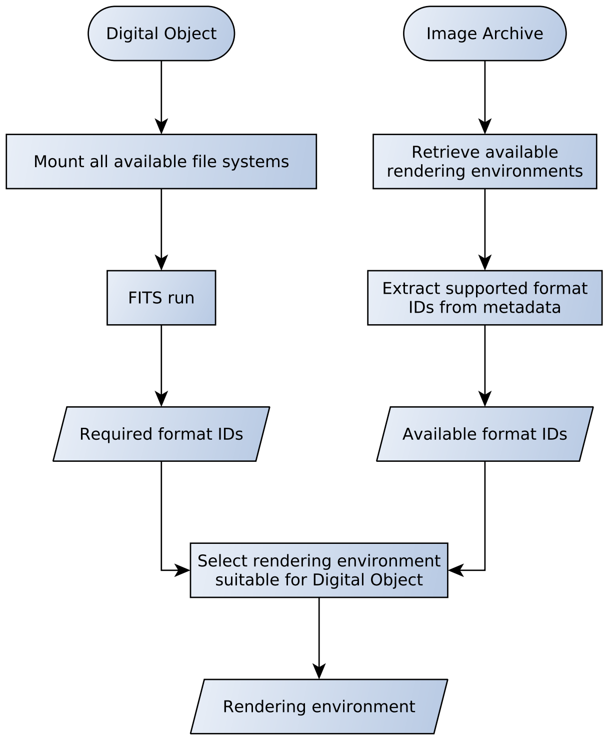
Digital Object
Image Archive
Mount all available file systems
Retrieve available rendering environments
FITS run
Extract supported format IDs from metadata
Required format IDs
Available format IDs
Select rendering environment suitable for Digital Object
Rendering environment

Figure 3: Characterization Workflow

As a first step, the digital object's container format has to be determined and its internal structure or filesystem has to be accessed. In our case, all containers were CD-ROM images, but some were, unfortunately, not standard ISO 9660 images. Many CD-ROMs of the multimedia age were produced as *hybrid* CD-ROMs containing both, an ISO 9660 file system as well as one (or more) additional file systems to overcome the restrictions of ISO 9660 and to implement system-specific features.

A common example are HFS/HFS+ hybrid CD-ROMs produced to be used with Apple computer systems and IBM-PCs. While Windows or DOS only sees the ISO 9660 filesystem, MacOS was able to access the HFS filesystem contained on the same CD-ROM to provide a different view on the contents. Fig. 1 shows the content of a HFS-hybrid CD-ROM opened on an Apple Macintosh computer system while Fig. 2 shows the content of the same CD-ROM opened on a Windows computer system.

For characterizing the container's content, both "views" of an object need to be evaluated. Unfortunately, none of the tools currently included in FITS are able to characterize the container format correctly. Running FITS on a hybrid disk image file will return a `SINGLE_RESULT` classifying the file as `application/x-iso9660-image`. To correctly identify the HFS filesystem on these images, we had to implement an additional container classification tool, matching for `Apple_partition_map` and HFS/HFS+ Master Directory Block signature to detect HFS/HFS+ filesystems on the CD-ROM images.

Once the filesystems contained in the images are identified, characterization tools can be used to determine the file formats of all files contained in the object. To automate the process of finding a suitable environment, we aggregate file format information of all files found. The resulting required format IDs can then be matched against list of import or native file formats in the software environments that are available from the image archive. In this step, certain formats may have to be prioritized over others. For example, a multimedia CD-ROM contains many picture files that belong to a multimedia application. While theses files can be viewed by any software environment that can view e.g. JPEG files, the multimedia application itself may only run on a PPC Apple MacOS.

In the final step, a very similar matching process is used to find a hardware environment that fulfills the requirements of the software environment. The resulting set of digital object, software and hardware environment is then the complete rendering environment and can be used by the EMiL emulation framework to re-enact the object.

5. RESULTS

In a first round we classified 69 CD-ROMs which were carefully selected to reflect the the diversity of the memory institution's collections. The goal was to have a wide representation of objects for different time periods and different types of objects. For instance, objects with more or less complex applications, interactive multimedia content especially with audio and video, 3D rendering, etc., but also objects made for different operating systems (Mac/Windows hybrid CDs) were chosen. The CDs were published between 1991 and 2009. For most of these CD-ROMs we had a transcript of their original system requirements. These requirements are used as ground truth for evaluating our classification. This information was not available for a few items, which meant that these CD-ROMs had to be tested manually.

Based on the analysis of executables found on each CD-ROM, we could determine from 66 CDs at least one suitable rendering environment selecting an appropriate operating system. 35 CDs were classified with multiple environments. While 11 CDs were hybrids which can either be run under a Windows or a Mac OS environment, we found 24 CDs which provided binaries for different Windows environments (e.g. legacy support for Windows 3.11 or MS-DOS). Even though final characterization results could be stored as metadata along with each object both characterization tools and available environments may be improved over time. On-demand characterization would provide the best user-experience, since newly added or improved environments can be considered. On a 4 CPU machine using 8 parallel threads, characterization of a single CD took less the 30 seconds, in most cases even less then 10 seconds. Only one object took more than 60 seconds to process, which was in fact a 4 Gb DVD.

For some CD-ROMs however, our simple file-by-file classification approach failed, e.g. because no executables were present on the CD-ROM. For instance, we have found a "Chinese Language Course", which contained more than 6000 files, however, most were encoded as HTML (fmt/96), JPEGs (fmt/41,43) and WAV (fmt/143) format. Figure 4 shows the format distribution as histogram. With no browser or other executable on the medium, a characterization solely on executable formats cannot determine a suitable environment. In this case, the aggregate file format information indicates that any environment with a Web browser installed would be suitable.

In a similar case, only PPT files were present on the CD, which would require a software environment containing an Office installation and a CD which only contained images. In the specific case of CD-ROMs, presence of an *autorun.inf* indicates the requirement for a Windows environment. Such information can be used for a secondary classification step, by defining sets of file formats typically associated with specific software setups or domain specific applications. Even if no suitable environment is found, the analysis of file format distribution can give useful hints about (additionally) required setups for a specific collection.

6. CONCLUSION & OUTLOOK

We presented a first step towards an automated reading-room access workflow for a large digital media collection. Our goal was to support users when accessing a CD-ROM from a memory institution's catalog and ideally render it instantly in a suitable emulated environment. To achieve this we have implemented a characterization tool for digital media containers accompanied with a technical framework and workflows. The characterization workflow successfully determined at least one suitable environment for 66 out of 69 objects based only on executable file formats and time signatures. Hence, for a vast majority of objects, a very simple heuristic can be applied to automate access.

Our evaluation, however, also shows the limitations of such a simple approach. For instance, if no executables are present, no classification of a basic rendering environment is possible. For more sophisticated environments, e.g. an environment for typical office workflows or specific engineering tasks, a thorough description of the available software and its supported file formats is necessary. The characterization workflow is then easily able to find an environment that can

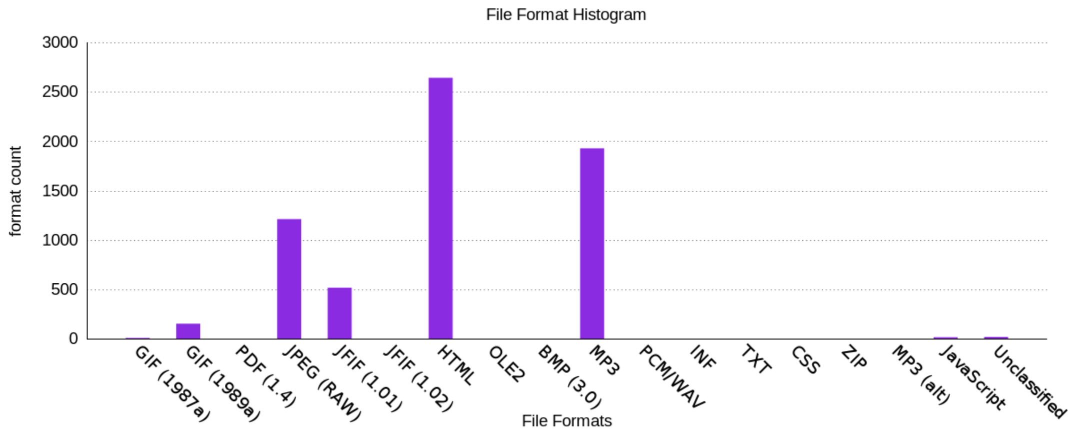

Figure 4: File format distribution of a CD-ROM containing an HTML-based language course.

render a specific file found on the CD-ROM.

For the aforementioned HTML-based language course, however, more considerations have to be taken into account. While almost any editor is able to open HTML files, the results may not be what the user expects. Similarly, all the WAV files found on the medium can be opened using a simple audio player. The object itself, however, has a characteristic "Web medium" footprint and clearly should be opened in a Web browser that allows using the language course in the guided and interactive fashion that was intended, rather than opening every file individually. Therefore, a more complex characterization approach that recognizes a "Web medium" footprint may be required.

A future option is to encode our technical metadata using the PREMIS data dictionary or similar established metadata standards to support interoperability and to simplify adaption or integration of emulation-based preservation strategies. In a similar way, technical interfaces to technical registries, such as PRONOM and TOTEM, enable rapid publication and sharing of new file format signatures, and especially, verified relations between file formats and necessary rendering software.

7. REFERENCES

[1] D. Anderson, J. Delve, and D. Pinchbeck. Document describing metadata for the specified range of digital objects. KEEP Public Deliverable D3.2a (online).

[2] T. Bähr, M. Lindlar, K. Rechert, and T. Liebetraut. Functional Access to Electronic Media Collections using Emulation-as-a-Service. In *Proceedings of the 11th International Conference on Digital Preservation (iPres14)*, page 332. State Library of Victoria, 2014.

[3] A. Brown. Pronom 4 information model. 2005.

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

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

[6] A. Dappert, S. Peyrard, C. C. Chou, and J. Delve. Describing and preserving digital object environments. *New Review of Information Networking*, 18(2):106–173, 2013.

[7] A. Dappert, S. Peyrard, J. Delve, and C. C. Chou. Describing digital object environments in premis. In *9th International Conference on Preservation of Digital Objects (iPRES2012)*, pages 69–76. University of Toronto, 2012.

[8] J. Delve and D. Anderson. *The Trustworthy Online Technical Environment Metadata Database – TOTEM*. Number 4 in Kölner Beiträge zu einer geisteswissenschaftlichen Fachinformatik. Verlag Dr. Kovač, Hamburg, 2012.

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

[10] T. Liebetraut, K. Rechert, I. Valizada, K. Meier, and D. von Suchodoloetz. Emulation-as-a-Service – The Past in the Cloud. In *7th IEEE International Conference on Cloud Computing (IEEE CLOUD)*, pages 906 – 913, 2014.

[11] D. Pinchbeck, D. Anderson, J. Delve, G. Alemu, A. Ciuffreda, and A. Lange. Emulation as a strategy for the preservation of games: the keep project. In *DiGRA 2009 – Breaking New Ground: Innovation in Games, Play, Practice and Theory*, 2009.

[12] PREMIS Editorial Committee. PREMIS data dictionary for preservation metadata, version 2.0. 2008.

[13] K. Rechert, D. von Suchodoloetz, T. Liebetraut, D. de Fries, and T. Steinke. Design and Development of an emulation-driven Access System for Reading Rooms. In *Archiving 2014*, pages 123–132. IS&T, 2014.

[14] J. van der Hoeven and D. von Suchodoletz. Emulation: From digital artefact to remotely rendered environments. *International Journal of Digital Curation*, 4(3), 2009.

[15] K. Woods and G. Brown. Assisted emulation for legacy executables. *International Journal of Digital Curation*, 5(1), 2010.