Management and Orchestration of Distributed Data Sources to Simplify Access to Emulation-as-a-Service

Thomas Liebetraut and Klaus Rechert
Albert-Ludwigs University Freiburg
Hermann-Herder-Str. 10
79104 Freiburg i. B., Germany
{firstname.lastname}@rz.uni-freiburg.de

ABSTRACT

Emulation-as-a-Service makes emulation widely available for non-experts and thus, emulation could prove valuable as a tool in digital preservation workflows. Providing these emulation services to access preserved and archived digital objects poses further challenges to data management. Digital artifacts are usually stored and maintained in dedicated repositories and object owners want to or are required to stay in control over their intellectual property.

In this paper we propose a distributed storage and data access model that ensures that the user stays in control over his digital objects by simultaneously providing efficient data transport and support for (space) efficient management of user modifications. Finally, a mechanism for orchestration of both storage and emulation services to re-enact a single pre-defined setup is presented.

General Terms

Infrastructure

Keywords

Emulation as a Service, Distributed Data, Framework, Cloud Computing

1. INTRODUCTION

Emulation of legacy computer systems is technically challenging and requires computing power as well as specialized knowledge about computing technology. These challenges pose a hurdle to non-technical users of emulation services that want to preserve and access digital objects like interactive art or legacy software. The goal of the Emulation-as-a-Service (EaaS) [10] framework is providing emulation services to these non-technical users like memory institutions or owners of digital object collections.

To implement the EaaS service model and make it usable for preservation purposes, a certain modularization and division of duties is required. Therefore, the EaaS framework is divided into the actual emulation service provided by the service provider, archives storing and maintaining digital objects provided by their respective owners, and modular workflows to access and interact with the digital object. While providing and maintaining emulation components requires highly specialized knowledge and will probably always be done by specialized service providers, the archive component is designed to be provided by different institutions.

Libraries and owners of collections of digital objects, may want to or are even required to stay in control over their intellectual creations, making it necessary to keep these digital objects in a separate archive. Consequently, there are potentially many decentrally organized archives that are operated independently from each other. They all may have different requirements on how to maintain and create the archived data and there may be little or no coordination between different archive providers. In some cases, users may choose not to use a public archive or storage service and create their own micro-archives that suit their specific needs. Some may only exist over the course of a single session. In such a decentralized structure, archives and emulation services may appear and disappear as well as digital objects may be relocated to other archives. But also object owners or users may decide to switch to a different EaaS provider. For this, we propose a comprehensive set of interfaces and metadata to orchestrate an EaaS service and coordinate access to multiple heterogeneous archives in a unified way.

An EaaS service provider may opt to provide various ready-made emulated computer environments, so-called base images with operating systems (OS) and drivers already installed and configured, sharing the costs of maintenance and technical expertise to create these environments. Instances of emulated environments, i.e. an installed and configured OS plus software stack on a virtual disk image, may reach up to hundreds of GB in file size. Even with currently available network bandwidth, copying a full environment to an EaaS Cloud service is inefficient and impairs the user experience. In addition, users may need to change, customize or personalize environments. Hence, user modifications need to be tracked and stored for subsequent usage. Therefore, we propose a distributed storage and data access interface that (1) ensures that the user stays in control over his digital objects, (2) provides efficient data transport even with limited bandwidth and (3) supports efficient management of user modifications.

2. RELATED WORK

The concept of legacy platform emulation is closely tied to the development of computer systems and is well established as a tool to bridge a technological gap [7]. Recently, emulation has evolved as a tool for preservation of complex digital assets [12, 16, 17]. Furthermore, emulation setups have been formalized to assess authenticity and performance [4], and specific aspects of simulation of individual technical components such as CRT screen simulation have been addressed [3, 14].

While these works have greatly promoted emulation in a scientific context as well as the professional use of emulation in digital preservation, many of these aspects have to be orchestrated and implemented individually for each purpose. For instance, emulation has been used to provide access to a large collection of legacy CD-ROMs [2, 18]. Furthermore, requirements and workflows have been developed for preparing ready-made environments to render certain digital artifacts [11]. To enable several institutions to make use of and potentially contribute to the collection, the digital objects were made available through a distributed filesystem and required a specific emulator setup on the user's site.

The KEEP project [1] addressed this problem by networked provisioning of various complex emulator setups [5, 8]. While a networked approach reduces technical and organizational hurdles on the client's side significantly, it still requires technical expertise and manual tasks carried out by the user. Furthermore, data management, especially maintenance of specific environments has not been addressed yet.

A more community-centered approach is the Olive platform [2], which is specifically designed to allow collaboration of different curators on a Cloud-based library. Olive also uses local emulation using a thin client approach to run virtual machines, but it uses its own protocol to stream data necessary to execute the virtual machine over the network. Modifications to a virtual machine, for example, newly installed software, can be transferred back to the archive, making derivatives of digital objects possible [13]. With our proposed data management approach, we split generic computer and software environments, potentially ready-made for emulation purposes, from highly specific user adaptions and user data. This way, the object owner remains in complete control of both how and by whom the objects are accessed as well as how and by whom the objects are stored and maintained but still benefits from cost-effective shared maintenance of common components.

3. REQUIREMENTS & ARCHITECTURE

Emulation-as-a-Service is built as a distributed architecture that separates the different tasks required to re-enact a single digital object. This separation allows for every component to be maintained by respective specialists. Basically, the EaaS model is divided into the emulation service itself that handles the emulation task, and archives that provide digital objects. While common objects like operating systems and software can be shared in federated storage

[1]Keeping Emulation Environments Portable, `http://www.keep-project.eu/`, last retrieved 2/1/2014

[2]`https://olivearchive.org/`

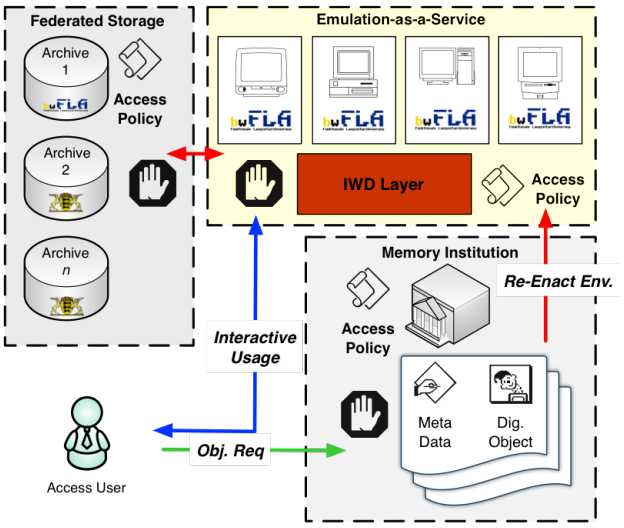

Figure 1: The EaaS distributed architecture with service provider and different archive providers.

archives, e.g. to share maintenance costs, digital objects preserved at a memory institution remain in the full control of these institutions (cf. Fig. 1).

In order to provide a public EaaS service model, an abstract description of how these different entities are to be coordinated is necessary. This description can then be used by an EaaS service's emulation components to bring together all the necessary bits and pieces to enable interactive user access to complex digital objects. Hence, the emulation component should not make any assumptions on the structure of the archive storing requested digital objects. Similarly, the archive or respectively its description must not assume a specific implementation of the emulation site. Finally, for emulation-based preservation of digital objects technical meta-data should be an abstract description of how to re-enact a specific computer environment that does not depend on a particular emulator software that will ultimately face the same digital obsolescence problem like all digital objects and technology.

3.1 Emulation Environment

To allow an individual computer environment to be replicable in the future, an abstract description of such a computer environment is required that is independent from emulator-specific configuration or its implementation details. Therefore, we introduce a comprehensive and abstract description of a computer system, the *emulation environment*. This technical metadata describes a computer environment to an extent that an emulation component can use it to reproduce the original environment. It includes the hardware architecture (platform) to be emulated as well as all devices that are optional to that platform (disk drives, sound cards, input devices, etc.).

These device descriptions might depend on external resources or assets like firmware ROM code or disk images that consist of binary data. For instance, an operating system, software and other digital objects are provided on emulated media

types such as virtual hard disks or CD-ROMs. After this data is created or retrieved from actual media or hardware and is preserved on a bit level, it has to be made available to the EaaS framework.

The data archives that provide preserved digital objects are not necessarily part of a specific EaaS service but can be provided by different data-centers or institutions. This means that all the digital objects required by an emulation environment may not be directly available for the EaaS service provider. Therefore, the data objects that are required by an emulation environment are referenced using *data bindings*.

These bindings reference the digital object using an URL that identifies the object's location or using a persistent identifier. Each binding is identified and accessed by the emulation component using an identifier unique within the emulation environment. To the emulation component these bindings are independent of the actual data location, access policy and other properties. This is achieved by the use of special data connectors that hide the complexity of actually accessing the digital object's URL and provide a simple, file-like access method to the data. Certain details of this access can still be specified by the emulation environment, though, for instance enforcing a specific transport protocol. While the data in an archive has to be read-only to guarantee long-term preservation constraints and to support efficient concurrent access, bindings always have to be writable from an emulator point of view. Technical restrictions in the emulated operating systems and saving customizations to the environment requires modifications to be tracked and stored for subsequent usage. To make the emulation environment metadata useful for archival and preservation purposes, it can also be extended with descriptive metadata like environment title, authoring information and creation dates. Similarly, the description may also contain information about what software is installed in the environment or which digital objects can be accessed.

The emulation environment is the basic building block to orchestrate the different components of the EaaS framework. It allows for separation of the emulation component and the archive and makes it possible to view emulation environments as a real document that can be referred to and be collaborated on. Changes made to an emulation environment can be ingested back into an archive which makes them again available as a new, derived environment.

3.2 Persistent Identification

While the emulation component heavily relies on the availability of data, the origin if this data does not matter. In the case of archives provided outside of the EaaS service provider, using static references to an archive to link the emulation environment with associated data is not feasible and would complicate migration to other EaaS or storage providers. Especially when implementing the archive component using dynamic Cloud storage solutions that can be allocated on-demand, referencing data by its network location (i.e. IP or host name) is not applicable as data can move to another host and may only be available for a limited time at the specified network location.

To solve this problem it makes more sense to ignore image locations altogether and refer to data using a unique and persistent identifier (PI) such as Uniform Resource Name (URN), Digital Object Identifier (DOI), or The Handle System (HDL) [1]. If the archive moves to another host or some digital objects move to another archive (or are distributed among many archives), the same PI can be used to resolve all available image locations, allowing load-balancing and dynamic allocation of resources in the cloud.

3.3 Persistent User Sessions

Once objects are stored in an archive and an appropriate environment has been created to access these objects, the environment should be immutable and cannot be modified except explicitly by an administrational interface. This guarantees that a memory institution's digital assets are unaltered by the EaaS service and remain available in the future. It also allows efficient concurrent access handling without the need to implement a complex and possibly expensive data and session management to avoid interfering with other users' sessions.

This immutability, however, is not easy to handle for most emulated environments. Just booting the operating system may change an environment in unpredictable ways. When the emulated software writes parts of this data and reads it again, however, it probably expects this data to represent its modifications. Also, users that want to interact with the environment must be able to change or customize it permanently. Therefore, data connectors have to provide write access for the emulation service while they cannot write the data back directly to the serving archive.

4. IMPLEMENTATION

The outlined requirements are used to orchestrate several components required to make digital objects in auxiliary archives accessible by an EaaS service instance. Individual data bindings that represent a single digital object are connected to by using *data connectors* on the EaaS site that are configured by the binding specification in the emulation environment. They can then be referenced by URLs of the form `binding://identifier`, e.g. to define a hard disk's data. Data connectors provide a generic interface between the archives and the actual emulation software to access heterogeneous data sources. They implement the network transport protocol, handle network connectivity and provide all the input and output operations that are common for a standard local file, like reading, writing and random access. Optionally, they also provide methods to authenticate the current user session to the archive if this is necessary to access protected digital objects. Different data connectors can be provided to support different network transport and authentication protocols in order to access different memory institutions' archives.

This concept requires some support from the archive to make archived objects accessible from the EaaS framework. Usually, digital objects from archives are not accessible directly as a single bit-copy of the original medium. Elaborate housekeeping information and further metadata is usually stored alongside the original object. To allow data connectors to access the individual digital object over the network, an archive server component has to be deployed at the memory institution's site that translates the internal data structures

used to archive the digital object to a network protocol suitable for accessing these objects. This archive component hides the complexities of bookkeeping and accessing preserved objects while granting or restricting access to individual objects. Consequently, the archive component can be highly specific to the needs and structure of the archive that are usually determined by the archiving institution. At the same time, it enables the EaaS service to access the raw data of individual digital objects in a unified way.

The distributed nature of this approach requires an efficient network transport of data to allow for immediate data access and usability. However, digital objects stored in archives can be quite large in size. When representing a hard disk image, the installed operating system, together with installed software, can easily grow up to several GB in size. Even with today's network bandwidths, copying these digital objects in full to the EaaS service may take minutes and derogates the user experience. While the archived amount of data is usually large, the data that is actually accessed frequently can be very small. In a typical emulator scenario, read access to virtual hard disk images is block-aligned and only very few blocks are read by the emulated system [15]. Transferring only these blocks instead of the whole disk image file is typically more efficient, especially for larger files.

Therefore, the network transport protocol has to support random data access and sparse reads without the need for actually copying the whole data file. While direct file access provides these features if a digital object is locally available to the EaaS service, it is not applicable in the general case of separate emulation and archive servers. Special-purpose network file systems like NFS (Network File System) or SMB (Server Message Block) provide file-like access to remotely exported files over the network. They, however, require a complex setup in the host operating system of both, the emulation service itself and the archive servers at the memory institutions. Additionally, this setup has to be done for every archive server that has to be available to an individual emulation component.

In contrast, the Network Block Device (NBD) [6] protocol provides a simple client/server architecture that allows direct access to single digital objects as well as random access to the data stream within these objects. Furthermore, it can be completely implemented and run without administrational privileges on the host operating system and has a very simple software design that does not require a complex infrastructure on the archive servers.

4.1 Handle It!

In order to access digital objects, the emulation environment needs to reference these objects in the emulation environment. Individual objects are identified in the NBD server by using unique export names. Consequently, a URL schema of the form `nbd:<hostname>:<port>:exportname=<name>` can be used to declare the network location of an individual digital object.

While this NBD URL schema directly identifies the digital object and the archive where the digital object can be found, the data references are bound to the actual network location. In a long-term preservation scenario, where emulation environments, once curated, should last longer than a single computer system that acts as the NBD server, this approach has obvious drawbacks. Furthermore, the Cloud structure of EaaS allows for interchanging any component that participates in the preservation effort, thus allowing for load-balancing and fail-safety. This advantage of distributed systems is offset by static, hostname-bound references.

Therefore, the Handle System is used as persistent object identifier throughout our reference implementation to identify resources. The Handle System provides a complete technological framework to deal with these identifiers (or "Handles" (HDL) in the Handle System) and constitutes a federated infrastructure that allows the resolution of individual Handles using decentralized Handle Services. Each institution that wants to participate in the Handle System is assigned a prefix and can host a Handle Service. Handles are then resolved by a central resolver by forwarding requests to these services according to the Handle's prefix. As the Handle System, as a sole technological provider, does not pose any strict requirements to the data associated with Handles, this system is used as a PI technology.

Each Handle consists of a set of typed records that the Handle server has to return upon request. While there are some predefined record types like "URL" or "EMAIL", individual Handle Services are able and encouraged to define their own record types that fit their needs. As currently the only information required in bwFLA is the actual network location, the URL type is used to encode the actual NBD URL. Because there can be more than one record of the same type in a Handle, several of these URLs can point to different archive providers or provide different transport types. Handles are then referred to in the emulation environment using URLs of the form `hdl:11270/61fecaebea36...` where `11270` is the prefix registered to the bwFLA project and the following string an arbitrary identifier.

The Handle Service resolving Handles for the bwFLA prefix is installed locally on one of the network nodes that run the bwFLA software, but is available globally. While it makes sense for owners of digital objects to make use of similar Handle Services to consistently refer to their objects independently from the archives that host the object, it is not expected that this is an inherent part of the EaaS infrastructure. As Handles are used throughout the emulation environments to identify data, the Handle Service has to be independent of a specific EaaS provider in order to preserve these emulation environments and possibly migrate them to different EaaS providers. This can be achieved either by each owner of digital objects to register his own Handle prefix, or to provide a institutionalized service similar to the DOI foundation that is more suitable for the needs of data expected by the EaaS framework.

4.2 Persistent User Sessions

The concept of interacting with re-enacted environment is an important part of the EaaS framework. Both, base systems provided by the service provider that curators can build their own environment on and users that interact with the final environment require modifications to an existing environment. Only saving these modifications and making them accessible to others makes sharing of resources possible, re-

ducing maintenance costs. At the same time it opens new possibilities for community-based curation efforts to allow contemporary witnesses to fine-tune and improve the user experience of digital objects like art or software [9].

A single EaaS instance not only consists of the digital objects themselves but also includes the emulation environment as orchestration and management metadata. Modifications to this metadata can easily be handled because the emulation environment can simply be copied due to its small file size. If the user attaches new drives or otherwise modifies the metadata, a new emulation environment can be created that includes the new hardware as well as the configuration of the base system. In most cases, however, the hardware environment does not change but the data on hard disks or other drives does. For example, installing software or configuring the software environment result in modifications to the underlying data. Also, just booting the operating system may change an environment in unpredictable ways and users that want to interact with the environment may change certain aspects of it. When the emulated software writes parts of this data and reads it again, it expects this data to represent its modifications.

As digital objects are not to be modified directly in the archive, a mechanism to store modifications locally at the EC while reading unchanged data from the archive has to be implemented. Such a transparent write mechanism can be achieved using a copy-on-write access strategy. While NBD allows for arbitrary parts of the data to be read upon request, not requiring any data to be provided locally, data that is written through the data connector is tracked and stored in a local data structure. If a read operation requests a part of data that is already in this data structure, the previously changed version of the data should be returned to the emulation component. Similarly, parts of data that are not in this data structure were never modified and must be read from the original archive server. Over time, a running user session has its own local version of the data, but only those parts of data that were written are actually copied.

We used the qcow2 container format[3], part of the QEMU project, to keep track of local changes to the digital object. Besides supporting copy-on-write, it features an open documentation as well as a widely used and tested reference implementation with a comprehensive API, the QEMU Block Driver. The qcow2 format allows to store all changed data blocks and the respective metadata for tracking these changes in a single file. To define where the original blocks (before copy-on-write) can be found, a *backing file* definition is used. QEMU's Block Driver API provides a continuous view on this qcow2 container, transparently choosing either the backing file or the copy-on-write data structures as source.

This mechanism allows modifications of data to be stored separately and independent from the original digital object during an EaaS user session, allowing to keep every digital object in its original state as it was preserved. Once the session has finished, these changes can be retrieved from the emulation component and used to create a new, derived

[3]The QCOW2 Image Format, `https://people.gnome.org/~markmc/qcow-image-format.html`, last access 8/15/14.

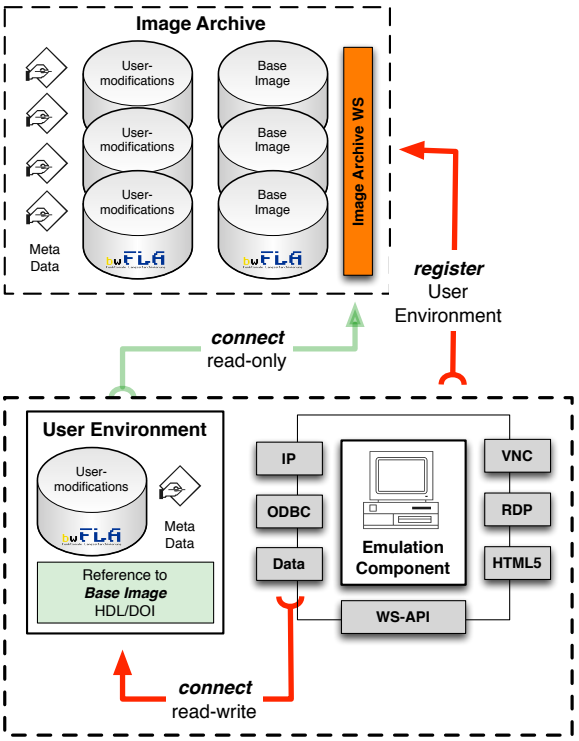

Figure 2: Data access workflow for derived environments. The eser environment exists only at the EaaS service provider until it is explicitly registered at the archive (if allowed).

data object (cf. Fig. 2). As any Block Driver format is allowed in the backing file of a qcow2 container, the backing file can also be a qcow2 container again. This allows "chaining" a series of modifications as copy-on-write files that only contain the actually modified data. This greatly facilitates efficient storage of derived environments as a single qcow2 container can directly be used in a binding without having to combine the original data and the modifications to a consolidated stream of data. However, this makes such bindings rely not only on the availability of the qcow2 container with the modifications, but also on the original data the qcow2 container refers to. Therefore, consolidation is still possible and directly supported by the tools that QEMU provides to handle qcow2 files.

Alternatively, a filesystem-based approach like UnionFS[4] could be used to track, store and maintain changes made to a system. These unification filesystems "stack" several modification layers on top of each other. While a filesystem-based approach offers convenient tools to track individual files, the metadata required to reconstruct these changes is implementation specific. Using a simple, block-oriented approach of maintaining a virtual disk's differential changes has some advantages in a digital preservation scenario, due to its simple meta-data structure. The result of changed blocks are a simple entries in a block mapping table (c.f. Listing 1 which defines which file the data should be read from. This simple

[4]A Stackable Unification File System, `http://unionfs.filesystems.org/`

representation allows a manual reconstruction, even if the original implementation is not available anymore.

Listing 1: An excerpt from the block mapping table used in qcow2.

```
Offset      Length     Mapped to   File
0           0x10000    0x270000    derived.qcow2
0x10000     0x10000    0x60000     base.qcow2
0x10000000  0x10000    0xab0000    base.qcow2
0x20000000  0x210000   0x50000     derived.qcow2
0x20210000  0x800000   0x2b0000    base.qcow2
0x30000000  0x10000    0xac0000    base.qcow2
0x3ffe0000  0x20000    0x80000     base.qcow2
```

Once the data modifications and the changed emulation environment are retrieved after a session, both can be stored again in an archive to make this derived environment available. If there is no efficient transparent write support and a full copy is used instead, the changed copy can be used directly. In case of a copy-on-write approach, only those chunks of data that actually were changed by the user have to be retrieved. These, however, reference and remain dependent on the original, unmodified digital object. It can then be accessed like any other archived environment.

4.3 Collection containers

Sometimes it is useful to archive several individual data objects combined in a single container. For example, when a software is distributed on more than one installation medium, all the images belong to the same software with each single one of them useless without all the other. To make this collection one single digital object, they can all be tied together into a container format, e.g. a UDF image or a tar archive. To refer to this new digital object and access individual images from it, the data connectors in our reference implementation support a mechanism to access the contents of containers.

To determine whether a digital object is a container, the data references can be used. If only the `binding://name` form is used, the digital object is accessed directly. As soon as a reference of the form `binding://name/subobject` is used to make use of a sub-object, the binding `name` is used as a container, requiring the use of the "collection connector" to access the data. To avoid implementing the NBD access protocol twice, this collection connector can be used on top of the NBD connector.

4.4 Example

Listing 2 shows an example emulation environment from our reference implementation describing an IBM OS/2 system. Apart from some management information like the title or an ID, it identifies the system architecture (line 4) and includes a drive specification (lines 9–17). The drive specification tells the EC about the virtual disk interface to use (line 11) and all necessary bus information. To refer to the data contained in the virtual hard disk, a special URI scheme referring to a binding is used instead of the actual location of the virtual hard disk image (line 10). This binding (lines 29–33) is then defined in terms of an HDL reference with automatic transport protocol negotiation in case the HDL resolves to more than one transport method (line 31). Finally, the binding also selects the copy-on-write access method (line 32) instead of a full copy, essentially enforcing a failure if none of the archives support random-seek read access. A second drive (lines 19–27) together with another binding (lines 35-38) demonstrates how the binding mechanism can be used for larger collections of floppy images for which it makes sense to archive them in one single container (e.g. as a tar archive or a UDF image). Subcomponents of this container can be accessed directly in the emulation environment with the EC providing an appropriate data connector to unpack this container.

Using this information, the EaaS framework can determine an EC suitable for emulating the requested system architecture (x86 PC). The EC then instantiates a suitable emulator configuration and connects to all defined bindings by the mechanisms described above. Additional environment configuration like an attached CD-ROM containing some digital artifact could be added by simply adding another `<drive>` element and choosing the correct PI for the CD-ROM. Likewise, the binding-mechanism also makes it possible to declare ROM-images or similar data.

Listing 2: An example emulation environment configuration.

```
 1 <emuEnvironment xmlns="EmuEnvironment">
 2  <uuid>2016</uuid>
 3  <title>IBM OS/2 2.11</title>
 4  <arch>i386</arch>
 5  <description>
 6   ...
 7  </description>
 8
 9  <drive>
10   <url>binding://system_hdd</url>
11   <iface>ide</iface>
12   <bus>0</bus>
13   <unit>0</unit>
14   <type>disk</type>
15   <boot>true</boot>
16   <plugged>true</plugged>
17  </drive>
18
19  <drive>
20   <url>binding://floppys/disk1.img</url>
21   <iface>fdc</iface>
22   <bus>0</bus>
23   <unit>0</unit>
24   <type>floppy</type>
25   <boot>false</boot>
26   <plugged>true</plugged>
27  </drive>
28
29  <binding id="system_hdd">
30   <url>hdl:11270/0ecd47a3...</url>
31   <transport>auto</transport>
32   <access>cow</access>
33  </binding>
34
35  <binding id="floppys">
36   <url>hdl:11270/c41d0444...</url>
37   <transport>auto</transport>
38   <access>cow</access>
39  </binding>
40 </emuEnvironment>
```

For digital preservation purposes, it is often not sufficient to have this functional description of an environment. If

any component of the emulation environment (especially the data referred to by bindings) is lost, the original purpose of the environment can no longer be determined. Therefore, the `<description>` element in the emulation environment contains a behavioral description of the emulated computer system like operating system, installed software, special configuration and customization this software underwent and other curation information. Using this archival information, a curator could, if he had access to all single original software components, re-create the complete environment.

5. USE-CASES AND EXAMPLES

To provide a better understanding of the EaaS image-archive interfaces and prototypical implementation, the following three use-cases demonstrate how the current implementation can be used in practical scenarios. An obvious scenario is the creation of so called derivatives of emulated computer systems, i.e. specifically adapted system environments suitable to render a specific object or to be used in a specific context. In a similar scenario a data object is injected into the environment which is then modified for later access, i.e. installation of a viewer application and adding the object to the autostart folder. Finally, an existing hard disk image (e.g. an image of a real machine's hard disk) is ingested into the system. This scenario requires, besides the technical adaption of the hardware environment suitable to be run in an emulator, private files are to be removed before public access.

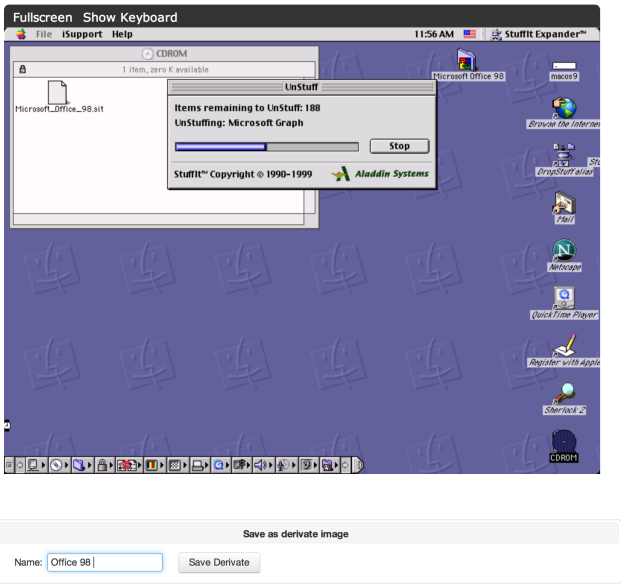

Figure 3: Installing uploaded software package and creating a derivative environment.

5.1 Derivatives – Tailored Runtime Environments

Typically, an EaaS provider provides a set of ready-made environments, so-called base images. These images contain a basic OS installation which has been configured to be run on a certain emulated platform. Depending on the user's requirements, additional software and/or configuration may be required, e.g. the installation of certain software frameworks, text processing or image manipulation software. To do so, the user is able to upload a software installation package, which is then injected into the emulated environment, e.g. as CD-ROM or DVD medium. Once the software is installed, the modified environment can be saved and made accessible for object rendering or similar purposes (cf. Fig. 3).

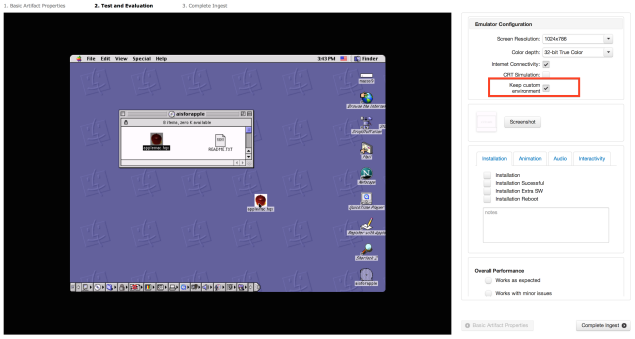

Figure 4: Ingest of CD-ROM art. Object is copied to the compouter's desktop and added as "autostart" object.

5.2 Object-specific Customization

In case of complex CD-ROM objects with rich multimedia content from the 90s and early 2000s such as encyclopedias and teaching software, typically a custom viewer application has to be installed to be able to render its content. For these objects, an already prepared environment (installed software, autostart of the application (cf. Fig. **??**)) would be useful and would surely improve the user experience during access as "implicit" knowledge on using an outdated environment is not required anymore to make use of the object. Since the number of archived media is large, duplicating for instance a Microsoft Windows environment for every one of them would add a few GB of data to each object. Usually, neither the object's information content nor the current or expected user demand justify these extra costs. Using derivatives of base images, however, only a few MB are required for each customized environment since only changed parts of the virtual image are to be stored for each object. In the case of the aforementioned collection of multimedia CD-ROMs, the derivate size varies between 348kB and 54MB.

5.3 Authenticity vs. Redaction

Another scenario of increasing importance is the preservation complete user system like the personal computer of Villem Flusser in the Villem Flusser Archive [5]. Such complete system environments usually can be achieved by creating a hard disk image of the existing computer and use this image as the virtual hard disk for EaaS. Such hard disk images can, however, contain personal data of the computer's owner. While EaaS aims at providing interactive access to complete software environments, it is impossible to restrict this "interactiveness", e.g. to forbid access to a certain directory directly from the user interface. Instead, our approach to this problem is to create a derivative work with all the personal data being stripped from the system. This allows users with sufficient access permissions (e.g. family

[5] Villem Flusser Archive, `http://www.flusser-archive.org/`

or close friends) to access the original system including personal data, while the general public only sees a computer with all the personal data removed. The redacted version of the disk image is inextricably linked to the original image, such that any action of the redaction process can be audited.

6. CONCLUSION & OUTLOOK

The presented architecture and implementation provides means to connect an external archive to an EaaS infrastructure and to curate its objects using emulation-based preservation workflows. It provides a functional view on both, data and the hardware configuration of a computer system instead of specifying a direct network location or hardware model, both of which may be meaningless in the far future.

At the same time, the EaaS service allows to make preserved environments accessible to a broad audience and provides a community-centered curation approach in which changes made by individual users to improve the authenticity of an environment can easily be made available to the rest of the community without losing the original version of the environment. This also makes it possible to track improvements and understand how computer systems and software works, allowing for a better restoration process in the future.

The interfaces and architecture presented in this paper also provide several features to overcome common problems in a distributed network. First, large digital objects can be accessed efficiently over the network. First, digital objects can now be efficiently accessed over the network. Together with a location-independent PI to reference data, this allows for a complete separation of the archive and the emulation services, also on an organizational level. New digital objects do not need to be registered at the EaaS service provider and the emulation service does not require direct access to the archive's storage backend in order to re-enact a single object's behavior and utility. Digital objects can rather be used directly after making them available using either their NBD network location directly, or, preferably, after they have been registered at some PI service. As this service is usually not dependent on the implementation of a specific EaaS framework, this is a much more versatile approach. This also leads to the possibility of quickly adding new archives to the system without having to coordinate with the EaaS service provider. The pure archive component can easily be implemented on any platform and does not rely on specific features to be available. The reference implementation should be able to run on any POSIX compatible system with network access without any modifications. Therefore, using EaaS and the proposed data management concept, object owners are able to present their objects (interactively) without actually releasing the environment and, more importantly, the intellectual property to the user. This is a required feature for digital art and similar digital assets: to provide access to an almost unlimited amount of users in order to unfold its potential impact on today's society, e.g. to use and interact with a piece of digital art, without anyone being able to copy it. The owner remains in control of the object and is able to restrict access any time simply by restricting access to their archive.

Second, the use of a copy-on-write mechanism together with a transport protocol that allows fragmented access improves the user experience. Instead of having to wait for a full copy of the digital objects to be made, only minimal amounts of data have to be transferred in order to make the environment usable immediately after the initialization. Furthermore, it allows a community-centered curation approach in which changes made by individual users to improve the authenticity of an environment can easily be made available to the rest of the community without losing the original version of the environment. This also makes it possible to track improvements and understand how computer systems and software works, allowing for a better restoration process in the future.

Finally, a more structured emulation environment allows for a more future-proof approach to emulation-based preservation. The emulation environment separates the functional description of a hardware system and the archival metadata required to understand the system. Each of them can be exchanged independently from each other, either using a different approach to describe the hardware in a possible future EaaS solution, or using a different preservation metadata that describes how the environment was built and preserved and how it can be used.

7. REFERENCES

[1] Arms, W. Y., May 2001. Uniform resource names: Handles, purls, and digital object identifiers. *Commun. ACM 44*, 5 (May 2001), 68–.

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

[3] Guttenbrunner, M., and Rauber, A., 11 2011. Re-awakening the philips videopac: From an old tape to a vintage feeling on a modern screen. In *Proceedings of the 8th International Conference on Preservation of Digital Objects (iPres 2011)* (11 2011), pp. 250–251. Posterpresentation: iPres 2011 - 8th International Conference on Preservation of Digital Objects.

[4] Guttenbrunner, M., and Rauber, A., May 2012. A measurement framework for evaluating emulators for digital preservation. *ACM Trans. Inf. Syst. 30*, 2 (May 2012), 14:1–14:28.

[5] Lohman, B., Kiers, B., Michel, D., and van der Hoeven, J., 2011. Emulation as a business solution: The emulation framework. In *8th International Conference on Preservation of Digital Objects (iPRES2011)* (2011), National Library Board Singapore and Nanyang Technology University, pp. 425–428.

[6] Machek, P., 1997. Network block device. http://atrey.karlin.mff.cuni.cz/~Davel/nbd/nbd.htm.

[7] Marsland, T., and Demco, G., 1978. A case study of computer emulation. *Canadian Journal of Operational Research and Information Processing 2* (1978), 16.

[8] Pinchbeck, D., Anderson, D., Delve, J., Alemu, G., Ciuffreda, A., and Lange, A., 2009. 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).

[9] Rechert, K., Espenschied, D., Valizada, I., Liebetraut,

T., Russler, N., and von Suchodoletz, D., 2013. An architecture for community-based curation and presentation of complex digital objects. In *Digital Libraries: Social Media and Community Networks, ICADL 2013* (2013), Springer, pp. 103–112.

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

[11] Reichherzer, T., and Brown, G., june 2006. Quantifying software requirements for supporting archived office documents using emulation. In *Digital Libraries, 2006. JCDL '06. Proceedings of the 6th ACM/IEEE-CS Joint Conference on* (june 2006), pp. 86–94.

[12] Rothenberg, J., 1995. Ensuring the longevity of digital information. *Scientific American 272*, 1 (1995), 42–47.

[13] Satyanarayanan, M., Bala, V., St. Clair, G., and Linke, E., 2011. Collaborating with executable content across space and time. *7th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*, October (2011), 528–537.

[14] Scott, J., 2012. What a wonder is a terrible monitor. Online `http://ascii.textfiles.com/archives/3786`.

[15] Tang, C., 2011. Fvd: a high-performance virtual machine image format for cloud. In *Proceedings of the 2011 USENIX conference on USENIX annual technical conference* (Berkeley, CA, USA, 2011), USENIXATC'11, USENIX Association, pp. 18–24.

[16] van der Hoeven, J., van Diessen, R., and van der Meer, K., 2005. Development of a universal virtual computer (uvc) for long-term preservation of digital objects. *Journal of Information Science 31*, 3 (2005), 196–208.

[17] van der Hoeven, J., and van Wijngaarden, H., 2005. Modular emulation as a viable preservation strategy. In *Proceedings of the 9th European Conference on Research and Advanced Technology for Digital Libraries* (Berlin, Heidelberg, 2005), ECDL'05, Springer-Verlag, pp. 485–486.

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