Albert-Ludwigs University of Freiburg

MASTER THESIS

Large-Scale, Transparent Format Migration System

Author:
Isgandar Valizada
Mtrk.Nr. 272 45 45

Supervisors:
Prof. Dr. Gerhard Schneider

Prof. Dr. Christian Schindelhauer

June 29, 2011

Erklärung

Hiermit erkläre ich, dass ich diese Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen/Hilfsmittel verwendet habe und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Abschlussarbeit nicht, auch nicht auszugsweise, bereits für eine andere Prüfung angefertigt wurde.

Ort, Datum

Unterschrift

Acknowledgement

I am thankful to Dirk von Suchodoletz and Klaus Rechert for their guidance and feedback which helped me to better structure my thesis.

I am also thankful to Roman Graf for his valuable hints concerning the Planets Interoperability Framework.

My thank goes to Kyle Retan who spent his time helping me in the correction of grammar and vocabulary mistakes.

Abstract

An important challenge in digital preservation is the problem of data format obsolescence. The digital objects eventually become inaccessible when the applications, which were properly interpreting their formats, cease to exist. Preserving these applications is an important step in solving above-mentioned problem. However, in order to enable the usage of the objects over time by newly developed tools, the objects must be periodically converted to more spread out formats. The work in this thesis is dedicated to the design of an easily extendible, transparent and large-scale oriented system which is able to perform such conversions. The system was designed as part of an existing digital preservation framework in order to enrich the framework with its migration functionality.

The interactive session replaying was chosen as the main approach in performing format conversions. According to this approach, the conversion process is carried out by computer interaction with the graphical user interface of the original, format-native applications. Their capability of exporting the loaded digital objects in a format other than their initial format is used. The order and nature of the input events necessary for this task are acquired once from the corresponding manual human interaction. The input events are then reproduced under the same or similar conditions for arbitrary digital objects subject to the same source-to-target format conversion.

The work in this thesis presents the structure of the system in general and focuses on the components responsible for the actual conversion. A prototype application has been developed and deployed in an existing digital preservation framework. During the experiments it has shown successful results when converting digital objects in a completely unattended way.

Contents

Introduction

The importance of Digital Preservation (DP) is rooted in Humanity's need to preserve its information heritage over time [1, p. 1]. Preserving information digitally was preferred to other options because digital representation of information is easy to store, process, refresh and replicate.

The storing of information involves the usage of the digital archives located on a media chosen according to the long-term preservation criteria [2, p. 4]. Digital Objects (DO) stored in an archive are internally structured according to their data formats. Accessing the content of DOs after their retrieval from the archive requires proper interpretation of their formats.

Since data formats become obsolete with time, the logical content of the corresponding DOs from that point on becomes inaccessible [3, p. 2] The solution to this problem lies in preserving the format-native, original applications [4, p. 4]. They are expected to have the most complete, built-in knowledge about their own data formats, and as a result, these applications should work more reliably with the corresponding DOs.

However, the accessibility of obsolete DOs only by their original applications limits the scope of their usage. It might be necessary to access the logical content of DOs with other tools non-native to their obsolete format. Therefore, it is important to be able to convert DOs from obsolete formats to formats of the same/similar type that are more spread out. The process of such conversion is referred to as Migration [1, p. 6].

Usually original applications implicitly support migrations, since many of them allow the saving/exporting of loaded DOs in a different format than their initial format. Such applications can support one or multiple migration input/output *scenarios*. Here *input* is the source format and *output* the target format, to which the source format is to be migrated.

For some of the scenarios original applications which could perform migrations in one step might not exist. Such scenarios would still be possible if the corresponding Migration Path(chain of intermediate migrations) starting at input format and ending at output format is available. Each intermediate migration would then have to be done by its corresponding original application.

Using the original applications, DOs can be migrated whenever they become obsolete in order to keep their logical content constantly accessible. However, performing migrations manually for every DO is not a feasible strategy, considering their large and constantly growing amount. Rather, the DOs have to be (periodically) migrated in large quantities and in an unattended way, in order to reduce the overall efforts [12, p. 155].

Original applications in most of the cases are suitable only for manual interactions with the user via GUI. Implementing the control of GUI elements to add the automation functionality to them is time-consuming and specific in each case [12, p. 155]. Therefore, some generalized automation mechanism is required which must function regardless of the original application used.

1.1 Goal and Structure

The purpose of this thesis is to develop a format migration system capable of performing large scale migrations. The system must be able to form the necessary migration path(-s) (if such is/are available). Afterward, using format native original applications it must migrate the DOs from one intermediate format to another until the output format is reached. The migration process is to be done transparently for the user and in a feasible, generalized way.

The system must also be responsible for all internal tasks related to the reliability of reaching the output format. It expects only the source DO(-s) of the input format, the scenario information and optional parameters to produce the resulting Migrated Object(-s) (MO). Having such a system would imply the ability to migrate large quantities of DOs in an efficient way.

Although the system could be used as a stand-alone preservation tool, the main scope of application for it are the existing digital preservation frameworks like Planets [14]. Having such a component integrated in them would contribute to the frameworks by providing the format migration functionality. In turn much of the frameworks' existing functionality would not need to be implemented and could be reused to enable complex preservation workflows involving not only migration.

The next chapter gives an overview of the basic building blocks of the system. Afterward, the requirements are clearly defined and followed by the design and implementation details. Further chapters contain the source code of the developed prototype and thesis conclusion.

Related Work

Constantly upgrading the original applications is time-consuming. A more suitable strategy would involve acquiring each tool once and be able to use it at any future point in time, without making any modifications to its program code. This requires adapting the modern environments for the application in such a way that it can be run as reliably as in its native environment, for which it was initially created. The term *environment* will refer to a combination of the operating system, the hardware available to it and installed third-party applications/libraries (if any).

Instead of adapting the real machine environment on which the application is to be run, creating a virtual one representing some environment native to the application can be created. The virtual environment would not require the real run-time installation of the operating system, software and the presence of hardware. It can therefore be formed much faster and on demand. The approach of creating virtual environments is related to the usage of Emulation technique in DP. [5, p. 55]

2.1 Hardware Emulation

Hardware emulators are the applications capable of creating the above-mentioned virtual environments. Their theoretical base relies on the Church-Turing-Thesis [6, p. 291]. On the low level they represent a software implementation of the hardware architecture and are capable of forming the so called Virtual Machines. For example, they implement the functionality of such computer components as: CPU, Memory managment- and I/O device subsystems. The

examples of emulator applications are Qemu [7] and Dioscuri [8].

In order to run a certain operating system on a virtual machine, the latter must be provided with a source for booting. The source could be, for example, an image file of a hard disk drive, where this system is installed. The operating system would run on the virtual machine by performing calls to the emulated hardware. Any applications/libraries installed on it could then be used as in the real environment.

The ability to run a certain operating system depends highly on the type of the emulated hardware. For example, the same emulator application could form two different virtual machines. One with the emulated x86 and one with x86-64 CPU architecture. This in turn defines what operating systems can work reliably on which of the virtual machines.

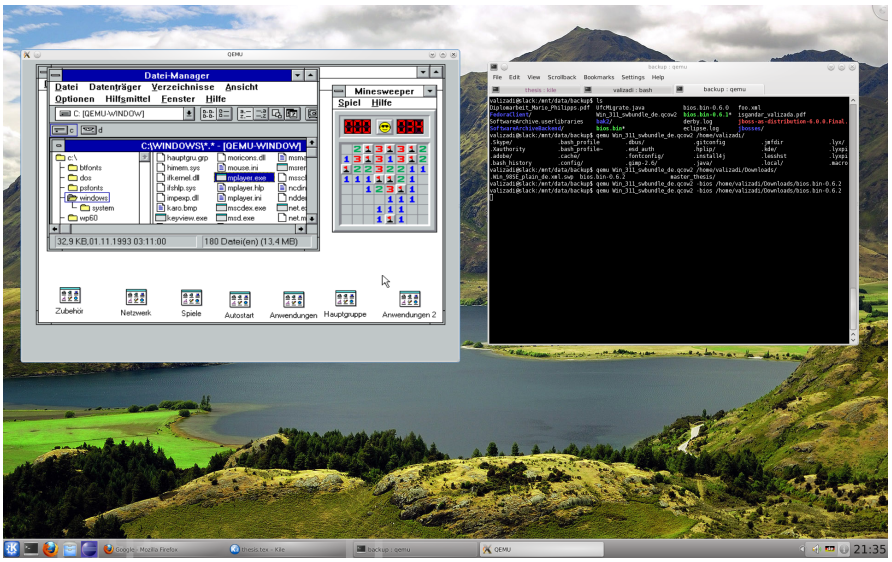

Figure 2.1: On the left: Emulating Windows 3.11 (release year 1993) on Linux Slackware 13.1 (release year 2010) via Qemu v12.0 emulator.

Although the constant reimplementation of the original applications can be avoided by using emulation, the deprecation of the emulators themselves remains a problem. In order to be able to use the obsolete emulators, it might be required to emulate the emulators themselves to eventually achieve the goal of supporting the necessary View Path [9, p. 8,9]. The recursive depth of such emulation would increase with time.

A more reasonable solution to the deprecation problem of the emulators is to upgrade them on demand. The amount of emulators being maintained can be reduced to several most reliable ones. Therefore, their periodic upgrade would not constitute a problem.

However, the newly upgraded versions of the emulators must guarantee their backward-compatibility with the previously used operating systems. If these systems become unusable at some point in time, then the original applications depending on them would become unusable as well. In general, an automated emulator testing system similar to [10] must be available to test the usability of operating systems with emulators.

2.2 Migration in the Emulated Environment

In order to perform the migration by the original tools in their native emulated environments, it is necessary to provide these tools with the digital objects on which they are to be operated. DOs must be injected into the emulated environment from the real one, and MOs produced by those applications must be extracted from it upon finishing.

Injection/Extraction of data into/from the emulated environments is possible via emulated data storage devices, e.g. via floppy-, hard disk- and cdrom drives. They can be thought of as a gateway for binary data exchange between the real and emulated environments. [11, p. 93] Usually emulators support the emulation of at least one storage device.

For example, in Qemu v12.0 it is possible to activate the emulation of a floppy drive with a virtual floppy disk inside the drive by such argument pair: "-fda floppy.img". Here *floppy.img* refers to a virtual floppy disk file prepared in the real environment, which inside contains binary data subject to injection into the emulated one.

Any modifications done to the virtual floppy disk inside the emulated environment will be reflected in the corresponding *floppy.img* file afterward. The file can be afterward analyzed in order to acquire the modified data.

Further, in this thesis, the emulated storage devices will be referred to as DO containers, unless their nature is otherwise indicated.

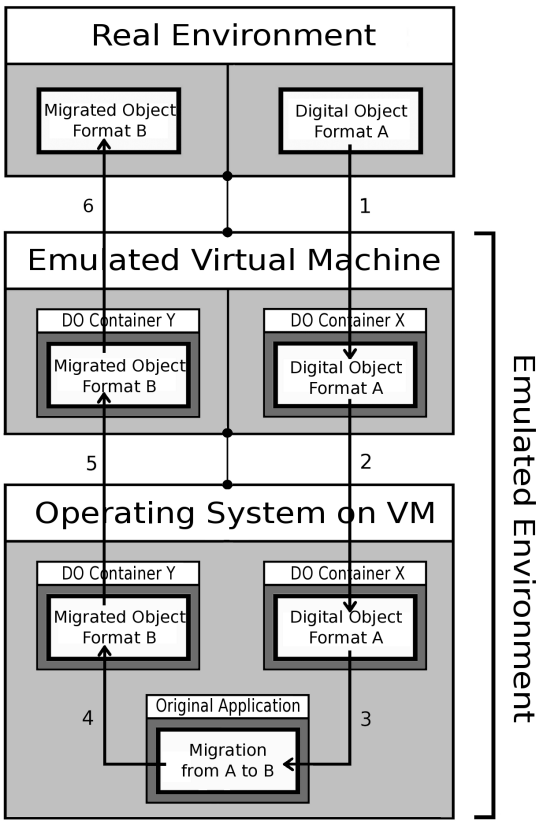

Figure 2.2: The typical workflow of migration by emulation involving DO containers.

Another possibility for the exchange of data between the real and emulated environments is the the usage of a network connection between them. [11, p. 93]

2.3 Unattended Migrations in the Emulated Environment

The interaction with the GUI of the original applications represents a series of user input events, such as mouse clicks/movements and key strokes. These events can be simulated using remote desktop control systems like VNC, allowing them to be performed in an unattended manner [12, p. 159]. This can be done by modified versions of remote desktop client applications, which are able to send input events acquired not from the user by his own manual interaction but, for example, from a binary file.

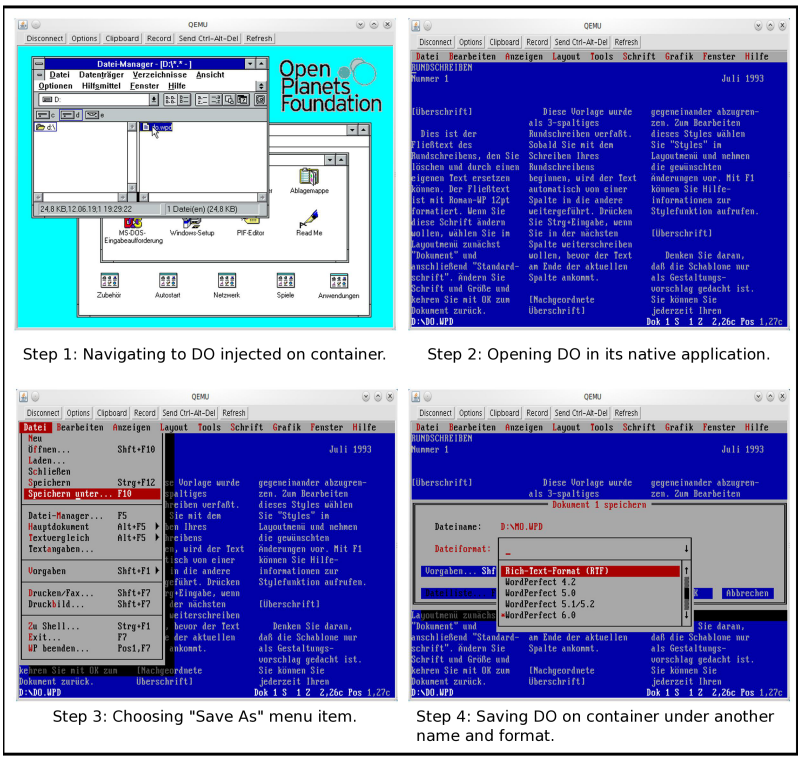

Figure 2.3: Migrating Word Perfect 6.0 document to Rich Text Format document via GUI interaction in the emulated Windows 3.11 operating system.

Being able to simulate manual interaction with the environment opens the possibility to perform unattended migrations by computer interaction with the GUI of the original applications and the operating system which they run in. This requires the order and nature of the input events for that task to be correctly determined. Acquiring the correct order and nature of these events

can be done by *recording* corresponding manual human interaction through remote control systems and saving it to a file.

Reproducing the same interaction programmatically in an unattended way would then be referred to as *replaying*. In this thesis the applications performing the recording and replaying will be referred to as Interactive Session Recorders/Replayers respectively. An example of an application that combines them both is Vncplay. [16]

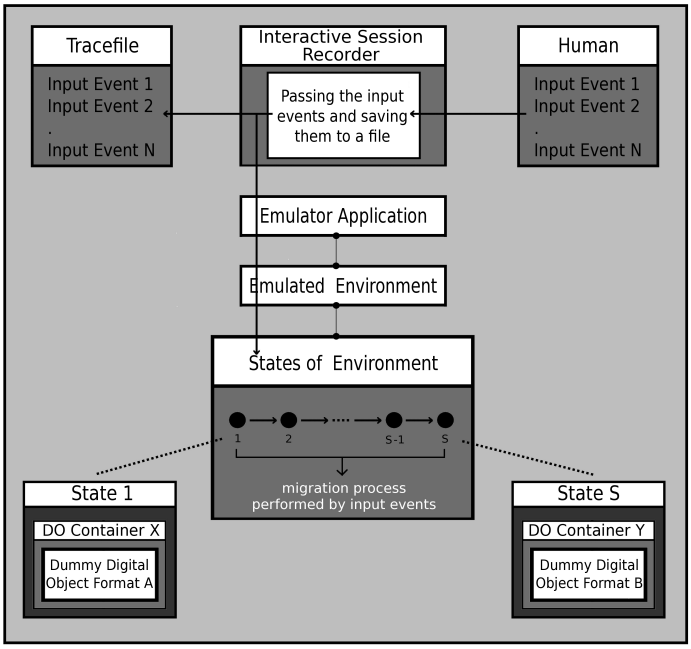

Figure 2.4: Recording the interactive migration session of the dummy DO.

In order for the interactive session recorders/replayers to be able to connect to the emulated environment, the corresponding emulator application must support activation of remote desktop access for its underlying Virtual Machine. For example, in Qemu v12.0 this can be done by specifying "-vnc :n" argument pair, where n denotes the number of the opened VNC [13] desktop.

To increase a reliability of the replaying process, each input event is also to be bound with precondition and an expected outcome. In such a case, during replay no next input event is sent for remote processing until the expected outcome of the previous one is observed in the environment. Therefore, each input event can be thought of as a trigger for change to the next abstract

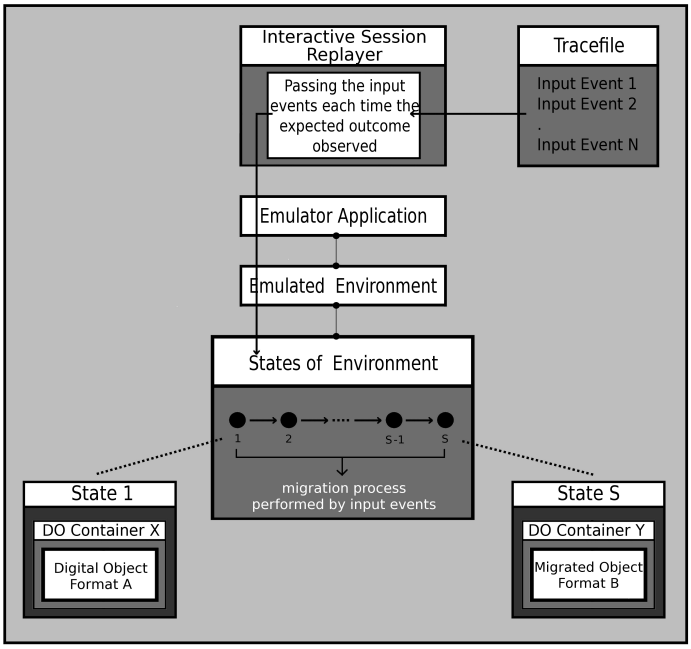

Figure 2.5: Replaying the prerecorded interactive migration session for a DO of interest.

state of environment. In case of a VNC system, the outcome is to be analyzed from the frame (screenshot) of the remote desktop.

The advantage of input event based interaction is that no original application specific API must be implemented and used for this task. In many of the cases such manipulation might be even impossible due to the proprietary nature of the applications.

The approach of unattended migrations via interactive session recording/replaying in the emulated environment was discussed in [12] [15]. Its feasibility was proven during experiments. The system suggested in the thesis is based primarily on this approach.

Design

The system assumes the existence of a common information storage accessible by its front-end and back-end components. The back-end component is used for the contribution of the migration specific information (e.g. interactive session recording). The front-end component uses this information in order to perform data migration (e.g. interactive session replaying). Hereinafter, the former will be referred to as Scenario Registration Component and the latter as the Migration Component.

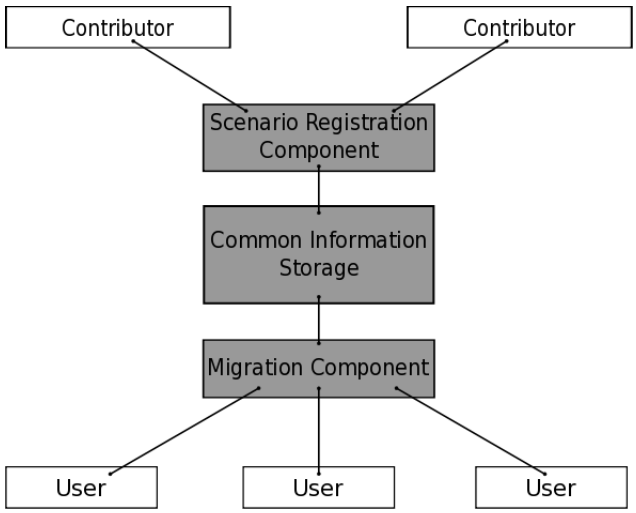

Figure 3.1: Structure of the system described on an abstract level.

The scenario registration component is not involved in the actual migration process. It cannot be used in any preservation workflow and, therefore, its integration into the digital preservation framework is not necessary. Further work in the thesis is devoted to the design of the Migration Component and the related challenges.

3.1 Requirements

Functional.

1. Transparency: The component must be able to perform migration process in an unattended way. It must separate the inner workflow from the user.
2. Large-scale: It must be able to perform migrations of large quantities of DOs subject to the same input/output scenario in one turn.
3. Wide-Accessibility: The component must be publicly available.
4. Integration: The component must be part of the existing preservation framework to enrich it with its format migration functionality.
5. Extensibility: Enriching the component with the support of new migration scenarios must not lead to the change of its functionality. It must be achievable in run-time, without the need of its temporary deactivation.
6. Robustness: An error handling mechanism must be present to increase the robustness of the migration workflow.

Non-Functional.

1. Flexibility: The component should be designed flexible enough to provide support for any third migration related approaches.
2. User-Feedback: It should accept and analyze user-feedback (if specified) upon finishing the migration process.
3. Supported Scenario List: It should provide user with the list of supported migrations if requested, preferably according to the specified filter.
4. Learning: A basic mechanism for run-time strategy improvement should be present.

3.2 Use Cases

The migration component supports three use cases:

1.
   - Input: request to acquire the list of supported migration scenarios; preference filter;
   - Output: list of supported migration scenarios and the meta-information for each of them;
2.
   - Input: an archive file (e.g. ZIP type) of DOs; the requested migration scenario; optional execution parameters;
   - Output: an archive of corresponding MOs or null; migration meta-information; session identifier;
3.
   - Input: session identifier; feedback about the migration session corresponding to the identifier;
   - Output: empty response

3.3 Actors

The potential actors accessing this component are:

1. Digital preservation interoperability framework, serving as a middle layer between the end user and the component.
2. Third applications requiring migration functionality.
3. End users accessing the component by means of any web-service consuming tools.

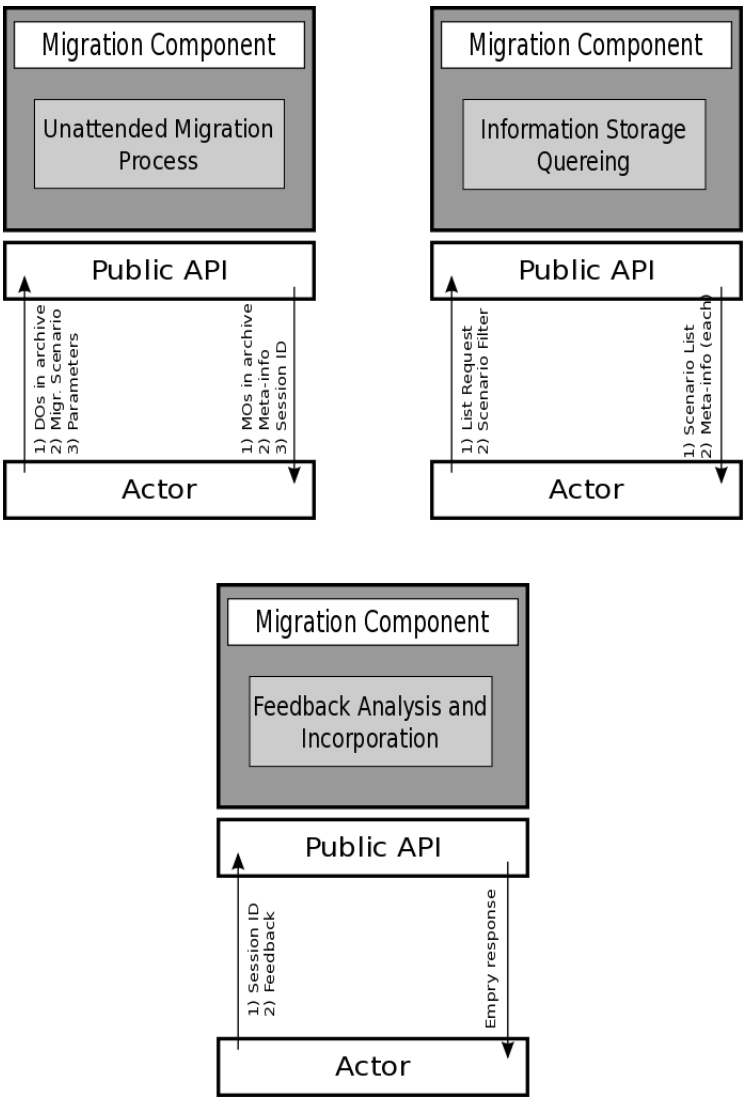

Figure 3.2: Demonstration of the possible use cases of the migration component.

3.4 Usage Example

In cases when the format of the obsolete DO is unknown, there is no possibility to choose the suitable migration scenario. In such cases the input format must first be identified. This task can be performed by characterization tools in DP. An example of such a characterization tool is Droid. [17] It connects to the Pronom [17] format registry in order to acquire the necessary information and identify the format of the DO.

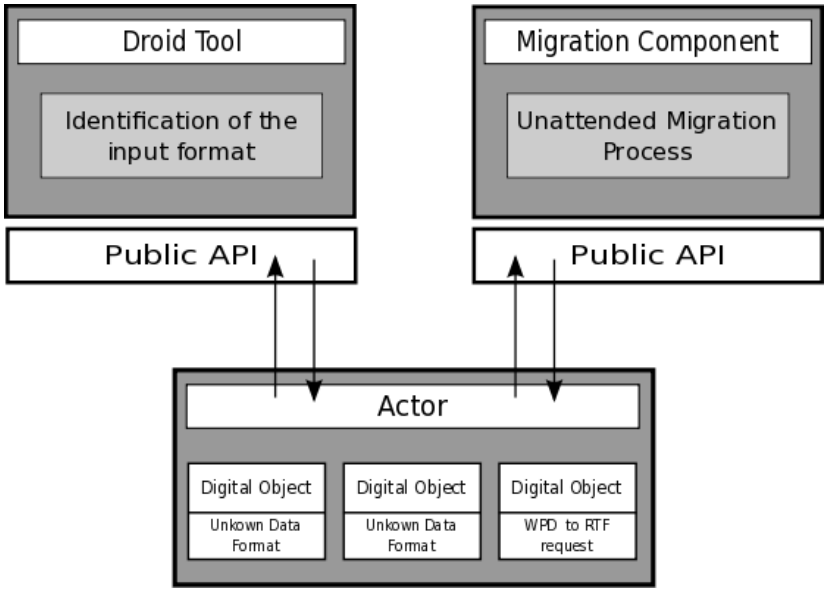

Figure 3.3: The actor uses the migration component in a preservation workflow, which includes DO characterization.

The actor could incorporate both the characterization tool and the migration component in the preservation workflow. The DOs would be first identified and then migrated according to the desired scenario. In practice, the workflow could consist of much more complex elements than the one being described.

Although in this example the actor could be represented by the end-user, a more realistic case is when this workflow is performed by the digital preservation framework. The framework would acquire the DO(-s) as an input, perform their characterization and migration hiding the corresponding workflow from the end-user.

Implementation

The inner workflow of this component is to be hidden from the user through the public API used for its invocation. Implementation of the methods for guiding the migration process in run-time are not assumed. Such design is in accordance to the **transparency** requirement.

In order to ensure the **wide-accessibility** requirement, this component is to be implemented in form of a web service. Its integration into existing DP frameworks would allow its usage both as a stand-alone tool and as a part of more sophisticated preservation workflows. The Planets Interoperability Framework (IF) [14] is a suitable candidate for the **integration** requirement.

In this framework each preservation tool is invokable according to one of the predefined code interfaces. The chosen interface depends on the tool's role in the scope of the DP (e.g. object migration/characterization/viewing/comparison). On the low level, the migration component would then represent the implementation of *Migrate*[1] interface of Planets IF. This interface contains the following method, which is invoked for the component's use-cases. This method represents the above-mentioned public API of the migration component and has the following code representation:

```
MigrateResult migrate(DigitalObject    digitalObject,
                      java.net.URI     inputFormat,
                      java.net.URI     outputFormat,
                      List<Parameter> parameters);
```

[1]Presented java code interface corresponds to Planets IF version of June 2010

Invocation is classified according to the previously defined use-cases. The use-case representing the unattended migrations is of particular interest and will be discussed first.

4.1 Migration Use-Case

The migration component queries the Scenario Database (Common Information Storage) in order to acquire the information necessary at further steps. This database is common to all parts of the system. The information, which is to be retrieved from it at the current step, is the possible migration paths.

The database needs to maintain information about all atomic migration scenarios available to it. An atomic scenario refers to a scenario that does not involve intermediate migrations. Depending on the user-requested migration, the corresponding complete migration paths which are formed from the atomic migration scenarios, are to be calculated. At this step the path preferences specified by the user in the parameter list can be taken into account. After the calculation, the component receives the path in form of ordered lists of atomic scenario identifiers.

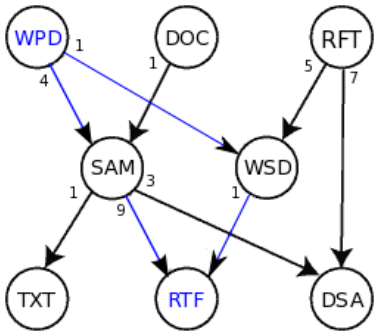

Figure 4.1: User requested the WPD-to-RTF migration. Two possible migration paths have been calculated according to the information in the scenario database. Both of them will be returned.

This problem can be represented in the form of a graph path calculation. In this graph the nodes correspond to different data formats and directed edges to available atomic migration scenarios. Finding the correct migration paths corresponds to finding the paths in this graph which would connect

the nodes of input and output formats specified by the user. Each path also has a certain cost associated with it, which is calculated from the costs of its edges. The total path cost would then define its efficiency value.

4.1.1 Scenario Bundles

By querying the Preservation Database using the identifiers of atomic scenarios it is possible to acquire scenario specific data bundles. These bundles represent a set of data used for invocation of the services responsible for the actual migration. On the code level they would represent tuples in a relational database. Their contents are the following:

- Meta-Information: This object incorporates meta description of the scenario, e.g. input/output formats, efficiency level, single/multiple DO support, author, timestamp. The input/output format identifiers need to be defined according to the conventional format registry system (e.g. Pronom).
- Service Endpoint: Network location of the service, which is able to perform this scenario.
- Parameters: Arguments of key/value form that need to be passed to ensure the successful scenario execution.

Depending on the endpoint of the bundle, the separate service corresponding to it will be invoked by the migration component at later stages. The inner representation of the parameter object depends on the service being invoked and might be different in each case. Such design of the scenario bundles satisfies the **flexibility** and **extensibility** requirements at the same time.

The former requirement is satisfied due to the fact that the migration execution is delegated to a separate service. Its inner working mechanism is not important for the invoking side. This implies that in addition to the service performing migration by interactive session replaying, services implementing other approaches can also be easily incorporated.

The latter requirement is satisfied due to the fact that enriching the system with a new migration scenario requires only the addition of the corresponding entry in the scenario database. The registration of a new bundle is to be done by the Scenario Registration Component. Registering a new bundle by adding a new database entry would require neither the reimplementation of the system's functionality nor its temporary deactivation.

4.1.2 Robustness

Among all received migration paths the one with the lowest cost should be chosen for execution by the migration component. However, even paths with the lowest costs are able to fail. In simple cases the failure of some edge can be overcome by retrying it. In each case of retry or complete failure, the efficiency level of the corresponding edge should be decreased. This mechanism would contribute to the **learning** requirement.

If the same edge fails too many times the path must be recalculated starting from the current node. In such a case the arc cost of a failing atomic migration scenario should be set to infinity. The path recalculation could be performed by using real-time planning algorithms (e.g. D* Lite [18]).

Furthermore, when a certain migration path is chosen, then all scenario bundles corresponding to its intermediate atomic migrations must be marked as *being executed*. This would imply the *write-lock* request for the bundle data and any data referenced from it. Such a mechanism would prevent potential path termination due to run-time bundle modification (or its referenced data).

4.1.3 Path execution

At this step it is assumed that the migration path is chosen and approved for execution. The corresponding bundles are accessible and write-locked. The next task is to perform the actual migration through external migration services.

These services need to implement the common interface similar to *Migrate* of Planets IF in order to be invokable by the migration component. In such a case they would contain a method used for their invocation by the migration component, which would look similar to the following:

```
MigrateResult performMigration(DigitalObject    digitalObject,
                               List<Parameter> parameters);
```

The input/output formats are to be set in the parameter list if necessary. The services must expect an archive of DOs with parameters and return the archive of MOs (or null). The archiving algorithm used for DO/MO storage is to be chosen by convention for all system and officially declared for the system users.

If the path is longer than a single edge, then the migration will be performed in multiple turns. Upon finishing a certain intermediate scenario, an archive with intermediate objects is acquired. It is in turn sent to another or the same migration service on the chain in order to perform the next scenario. The picture below illustrates an example of complex migration.

The final result of the last scenario is prepared as a response for the end user of the migration component. It should be accompanied by the meta information of the performed migration. Additionally, the user must receive an identifier, which uniquely identifies the finished migration session.

4.1.4 Block/Interactive Migrations

Until now it was assumed that the migrations performed by the services have an atomic nature. They accept an archive of DOs and produce an archive of MOs. However, not all migration services might be able to migrate multiple number of files in one turn.

The data stored in the meta-information object of each bundle must include the information about the capabilities of the corresponding service. Cases when the service supports only single migrations must be distinguished. In such cases the migration must be done iteratively for each DO. Therefore

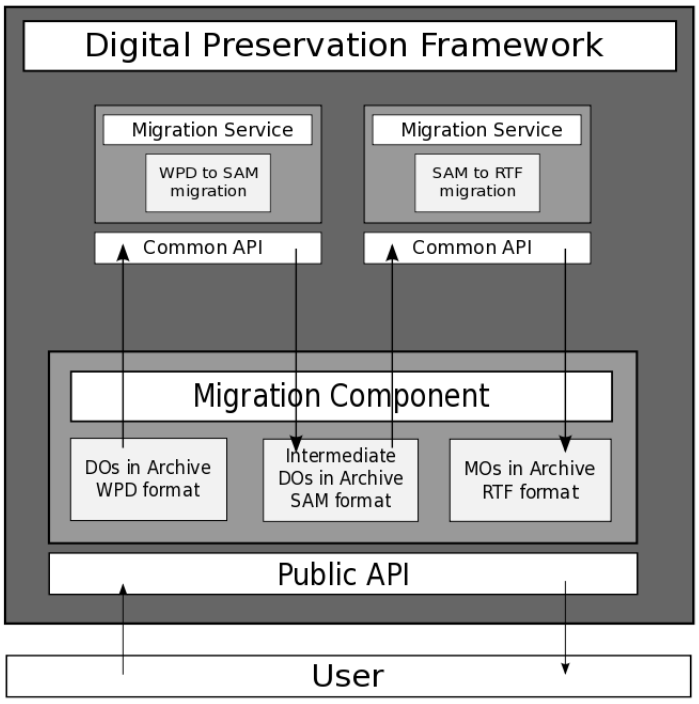

Figure 4.2: Migration involving intermediate scenario. By an *archive* in this context a package of files stored together is assumed.

passing an archive for the injection of DOs must be done only in cases when the service used on this step supports such migrations. Regardless of the case, block/iterative migrations ensure the system's requirement of **large-scale** migrations.

4.1.5 Migration Component Workflow Example

As an example, the WPD (Word Perfect 6.0) to RTF (Rich Text Format) migration will be discussed. User invokes the migration component by providing an input: an archive of DOs, input format - WPD, output format - RTF, parameter that denotes maximum path length of 2.

The migration component queries the Scenario Database and receives five different migration paths. Among them only one path satisfies the maximum migration path length condition. The path is the following: WPD to SAM; SAM to RTF. The migration component retrieves the bundles corresponding to these scenarios using their identifiers. When retrieved for performing the migration, the bundles are automatically marked as *being executed*.

At first a WPD to SAM migration is performed. The endpoint contained in the corresponding bundle is used to invoke the responsible migration service. The meta-information of the bundle contains the field indicating that this service can perform migrations of multiple DOs in one turn. Therefore, the DOs are passed in an archive accompanied with the execution parameters of the bundle. The WPD to SAM migration is performed, and an archive of MOs is returned along with the success status.

An archive of acquired intermediate MOs of SAM format is sent for the second scenario of the path. According to the meta-information in the SAM-to-RTF bundle, the service is unable to perform the migration of multiple DOs in one turn. Therefore, the archive of intermediate MOs is extracted to the local storage. From there the intermediate DOs are migrated one-by-one and the results of the migration are collected.

The resulting DOs of RTF format are packed into an archive. The unique session identifier corresponding to this path is generated. Then the MOs archive, the session identifier and any meta-information are returned to the user as a final result.

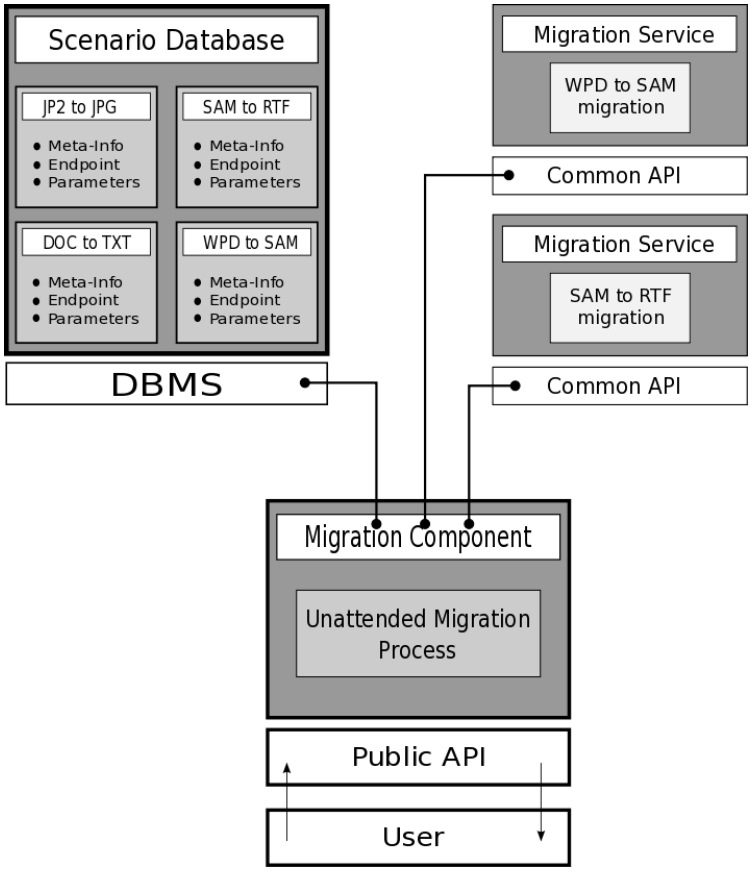

Figure 4.3: The interaction of the migration component with other elements of the system.

4.2 Replayer Service

This section describes the working principle of the migration service based on the approach of interactive session replaying. As any other service invokable by the migration component, it implements the common API. According to the API it receives a list of key/value parameters taken from the bundle corresponding to the desired scenario. The parameters in case of the Replayer Service are the following:

- OS-Image: An image file of a bootable hard disk with an operating system, original migration tool(-s) and any third applicaions/libraries installed.

- Tracefile: An object containing the recorded interactive migration session and the information about the DO container used for data injection during recording.
- Emulator: An object containing a list of endpoints of the emulation services, supporting the OS-Image. For each service it contains another list with identifiers of emulator applications, which are compatible with this image. It could also contain image specific information and/or secondary objects [19, p. 6] necessary for the successful generation of the View Path (e.g. bios file location, specific emulator directives).

The retrieval of the bundles corresponding to this type of service is depicted below. The objects stored in the parameter list represent only the references to the real locations. The real objects will be retrieved at different stages of the replayer service workflow.

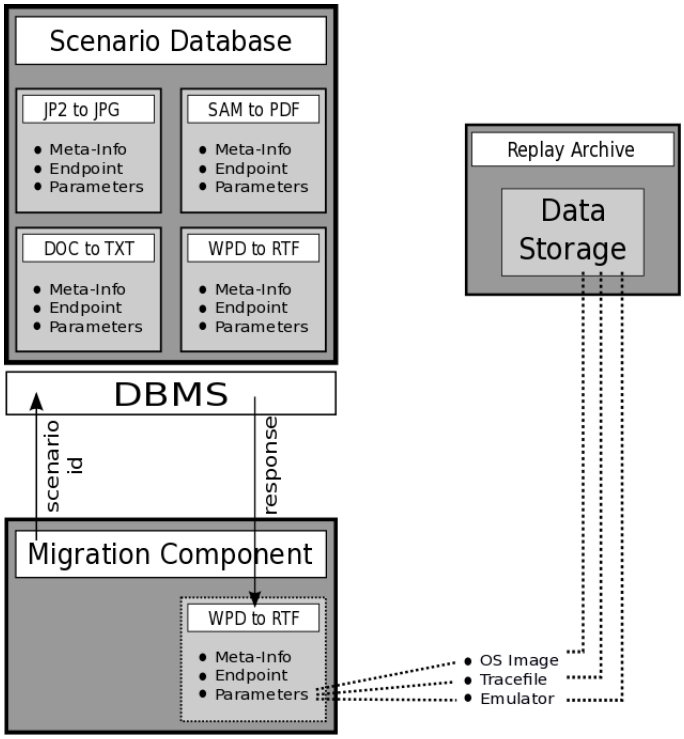

Figure 4.4: Retrieving the desired scenario bundle corresponding to the replayer service.

The filenames of the DOs play an important role in the case of migration in the emulated environment. The reason for this is that not all operating systems are able to handle long file names and/or special characters. Therefore, the DOs received in the archive must be first renamed using some conventional and *safe* naming scheme. For example $do\{N\}.\{fmt\}$, where N denotes their id (incremental) and *fmt* the format extension.

4.2.1 Container Preparation

Further sections describe the workflow of the replayer service starting from the container preparation. The creation of a DO container and injection/extraction of DOs into/from it is delegated to a separate subunit. It is crucial that the filenames of the DOs do not change after injection/extraction.

There are different possibilities for DO container creation. Optical devices are either read-only or require additional software for burning data to them. Floppy disks are a relatively easy solution, but they are limited in size. In this thesis the task of data injection/extraction is performed through the usage of hard disk drives. They are modifiable and their size can be adjusted to the desired degree. This allows for injection of the large-quantities of DOs into the emulated environment.

The procedure for empty hard disk creation can be summarized with the following steps:

1. Allocating the desired amount of space for the disk image.
2. Making only a single partition in the disk. The first partition in the hard disk starts at 32256 byte offset. This offset value is a crucial information, which later allows to identify the location of the data in the image file.
3. Formatting the single partition starting from the above-mentioned offset. Selection of the file system type is an important step. Its type defines, which operating systems could use it for the emulation of their data storage.

The size and the filesystem of the hard disk must be chosen wisely by the replayer service. The size would depend on the DOs being injected. The filesystem type should be retrieved from the tracefile object of the corresponding scenario bundle.

In the Appendix of this thesis there are two related code listings. The first one is used for the creation of virtual hard disks and the second for data transportation. Future implementations of the system would need to contain this functionality in the form of code libraries.

The creation of DO container can be done in the following way:

```
ArrayList<String> cmd = new ArrayList<String>();
cmd.add(muProps.m_hddCreate);
cmd.add(fSys);
cmd.add(dataSize);
cmd.add(tmpDisk.getAbsolutePath());
(new ProcessRunner(cmd)).run();
cmd.clear();
```

In this code the "hdd_create.sh" script is being invoked. It is provided with the desired filesystem and size of the disk. The last argument denotes the disk's absolute path, i.e. the location and the name of the resulting disk image.

Injection of the DOs using the script is done according to the following code:

```
ArrayList<String> cmd = new ArrayList<String>();
cmd.add(muProps.m_hddIo);
cmd.add("i");
cmd.add(tmpDisk.getAbsolutePath());
cmd.add(tmpDob.getAbsolutePath());
(new ProcessRunner(cmd)).run();
cmd.clear();
```

The script "hdd_io.sh" is being invoked and provided with the "i" argument, denoting data injection. The arguments after, refer to the absolute path of the disk image and the DOs for injection.

4.2.2 Emulation

The next step is to start the emulated environment and enable the remote desktop control in it. This task is performed by one of the emulation services listed in the emulator object of the corresponding scenario bundle.

When the emulation service is chosen, it is provided with all the necessary information contained in the emulator object of the bundle. The service is also provided with the DO container file for its injection into the emulated environment. If the emulation was started successfully, DO container injected and the remote desktop control enabled, then the service replies with success response. Additionally it returns the port for remote control clients and the unique session id. In case of failure a different emulation service or a different emulator of the same service (if any) could be tried.

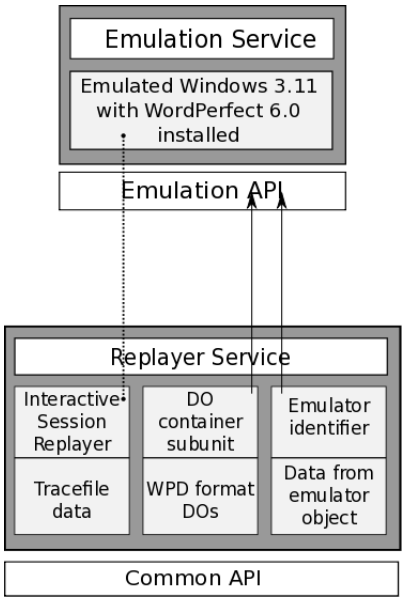

Figure 4.5: Request for activation of the emulated environment and connection to it via interactive session replayer.

The port handling mechanism must be present. The service should not provide remote control access for multiple different environments, using the same port. The port handling method used in the current implementation of the service is enclosed in the Appendix.

Depending on the emulator identifier, chosen by the replayer service according to the data in the emulator object, the corresponding executable location is retrieved by the service. This information could be taken from an XML configuration file containing such correspondence on the emulation service side. The invocation of the emulator with remote control support can be demonstrated by this sample code listing:

```
Integer freeVncPort = getFreeVncPort();
cmd.add(muProps.m_qemuExec);
cmd.add("-vnc");
cmd.add(":" + new Integer(freeVncPort - portMin));
cmd.add("-drive");
cmd.add("file=" + scenario.m_image + ",snapshot=on");
cmd.add("-hdb");
cmd.add(tmpDisk.getAbsolutePath());
qemu = new ProcessRunner(new ArrayList<String>(cmd));
qemuThread = new Thread(qemu);
qemuThread.start();
```

In this code the emulator is being invoked and provided with the instruction to enable VNC support for its emulated environment. The VNC desktop number is formed from the value returned by the function, which is guaranteed to return a free port.

Upon request and using the session id, this service must terminate the emulated environment, detach the DO container and return it to the replayer service. In order to avoid the resource leakage this service should also expect the consumer to send "keep-alive messages". If no message has arrived within the predefined timeout then the emulated environment must be terminated and all resources freed.

4.2.3 Replaying

If the replayer service receives the success response from the emulation service and the opened port information, it can then invoke its interactive session replayer to perform the migration. As an input the replayer must receive the tracefile with the previously recorded interactive session, the endpoint of the emulated environment and the port for connection.

```
cmd.add("java");
```

```
cmd.add("-jar");
cmd.add(muProps.m_vncplay);
cmd.add("HOST");
cmd.add(InetAddress.getLocalHost().getHostAddress());
cmd.add("PORT");
cmd.add((new Integer(freeVncPort)).toString());
cmd.add("autoplay");
cmd.add("true");
cmd.add("tracefile");
cmd.add(scenario.m_tracefile);
ProcessRunner vncplay = new ProcessRunner(cmd);
vncplay.run();
```

In this code the interactive session replayer connects to the emulated environment on the specified port. The replaying process starts and the DOs are being migrated from the input to output format according to the migration scenario. Inside the emulated environment the produced MOs are placed on the DO container. The interactive session replayer finishes its work and the replayer service requests emulation termination. The emulation service finalizes the environment and returns the DO container with migrated objects.

4.2.4 Response Forming

This step requires the replayer service to extract the objects from the received DO container. For this task the DO container subunit is invoked and the contents are extracted to the local storage of the replayer service.

```
cmd.add(muProps.m_hddIo);
cmd.add("e");
cmd.add(tmpDisk.getAbsolutePath());
cmd.add(tmpIoDir.getAbsolutePath());
(new ProcessRunner(cmd)).run();
cmd.clear();
```

The MOs must be identified by the matching of their filenames. By convention all produced MOs must be named according to a predefined naming scheme. An example of such could be $mo\{N\}.\{fmt\}$, where N denotes its number and *fmt* the format.

The naming scheme must be taken into consideration during the interactive session recording as well. This implies that, the contributor performing the recording of the migration session for its later usage should produce MOs according to the conventional sheme. Otherwise, the MOs will not be identified when the corresponding session is reproduced for other DOs by the replayer service.

When MOs are identified among all contents of the container, they are then renamed to their initial file names but with an output format extension. The resulting objects are packed to an archive file and returned to the migration component.

4.2.5 Replayer Service Workflow Example

In this section the sample workflow involving the WPD to RTF migration will be discussed. The operating system and the original tool used, in this case are Windows 3.11 and WordPerfect 6.0 installed on it. According to the workflow, the migration component invokes the replay service by providing it with the archived DOs and the parameters. It will be assumed that the prerecorded WPD-to-RTF migration scenario is able to migrate the desired number of DOs in one turn.

The replay service parses the parameter list and acquires the references to the three necessary objects: OS-Image, Tracefile and Emulator. It then extracts the received DOs from the archive to the local storage, while calculating their total size. The DOs are renamed according to the predefined naming scheme and the correspondence between the real and virtual names is stored separately. The next step is the DO container creation and injection of data.

The DO container subunit is invoked and provided with the information about the desired filesystem of the hard disk drive and its size. The size is set to a value big enough to hold the DOs and the MOs, which will be produced in the emulated environment. The DO container subunit acquires the input data and produces the hard disk image file. It then injects the DOs, without changing their filenames. After the operation is completed, it returns the prepared container to the replay service.

When the DO container is prepared, the emulated environment can be started. The replay service acquires one of the emulation service endpoints stored in the scenario bundle and one of the suitable emulator ids associated with it. It then invokes the respective emulation service by providing the following: the emulator id, the operating system image of OS-Image object, any secondary objects and/or directives necessary for the emulation of this operating system, and the prepared container, with DOs in it.

The emulation service receives the input data and starts the emulated Windows 3.11 with Word Perfect 6.0 installed in it. It also attaches the the hard disk container with the DOs. The remote desktop control is being enabled. It then sends a success response with a port number for connection as well as the session identifier. The replay service acquires the response and initiates the periodic sending of keep-alive messages each time specifying the session identifier.

The replay service then starts its interactive session replayer. It is provided with the network locaton of the emulator service and the port for connection. The interactive session replayer acquires the input events from the tracefile object and automatically passes them on to the emulated environment.

These events correspond to a manually prerecorded WPD-to-RTF migration using WordPerfect 6.0 original application in the emulated Windows 3.11. The input events lead to the following actions in the emulated environment:

1. The WordPerfect 6.0 is executed via mouse click on its icon.
2. The "Open" menu of Word Perfect 6.0 is chosen.
3. In the opened dialog box, the DO on the attached hard disk drive is chosen and loaded.
4. The "Save As" menu of Word Perfect 6.0 is chosen.
5. In the opened dialog box the new file name is chosen according to the conventional naming scheme and the DO is saved under RTF format on the same hard disk. Thus the MO is produced at this point.

This set of actions is performed for each DO automatically. (For the questions

regarding migration cycles in execution of the replay process, please refer to the *Outlook* section of the *Conclusion* chapter.) After sending the last input event and observing its expected outcome, the interactive session replayer finishes its work. The replay service sends a request for the termination of the emulated environment, using the session identifier. The emulation service frees all resources, detaches the hard disk image with the newly produced RTF format MOs and returns it to the replay service.

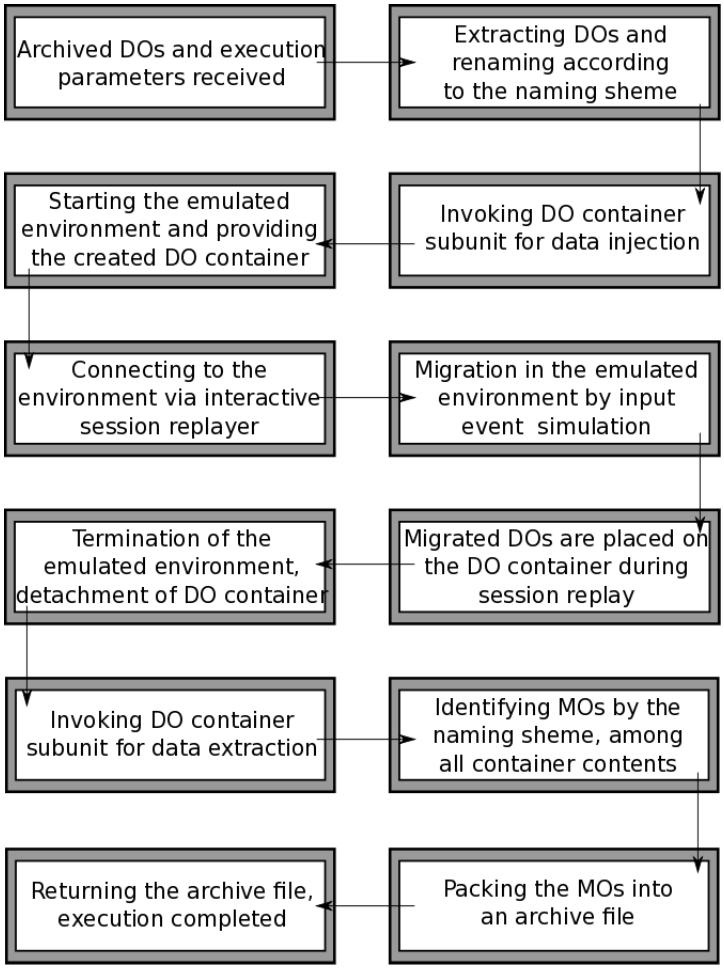

Figure 4.6: Typical workflow of the replayer service.

The replay service invokes its DO container subunit to extract the contents of the hard disk image. The contents are extracted to a local storage. Afterward the MOs are identified according to the conventional naming scheme. They are renamed to their initial file names but with RTF extension and packed into an archive file. The archive file is then returned to the migration

component. At this point the execution of the replay service is completed.

4.3 Scenario Retrieval Use-Case

The second use-case to be discussed is the request for returning the list of supported atomic scenarios. Upon receiving the request in the form of a corresponding parameter, the migration component queries the Scenario Database and returns the meta-information object of each scenario bundle to the user. Furthermore, a predefined filter could be specified as an additional parameter during invocation. This filter would correspond to a desire of acquiring only those scenarios which satisfy the supplied conditions. For example, in case of the replayer service type bundles, the usage of only certain operating systems/applications/emulators could be specified.

On the code level returning the list of supported migrations could be done by an implementation of *describe* method of Planets IF *Migrate* interface. Its code representation is the following:

```
@WebMethod(operationName = PlanetsService.NAME + "Describe",
    action = PlanetsServices.NS
        + "/" + PlanetsService.NAME + "/" + "Describe")
@WebResult(name = PlanetsService.NAME + "Description",
    targetNamespace = PlanetsServices.NS
        + "/" + PlanetsService.NAME, partName = PlanetsService.
            NAME
        + "Description")
@ResponseWrapper(className = "eu.planets_project.services."
        + PlanetsService.NAME + "DescribeResponse")
ServiceDescription describe();
```

This method is used for providing a brief description of the deployed preservation tools, including their supported migration scenarios. Inside it could encapsulate the access to the scenario database in order to acquire the necessary information and form the response. However, it should be taken into account that the method accepts no parameters. Thus it is not possible to specify a desired filter. If the presence of filter functionality is important then a workaround could be the usage of the *migrate* method for the scenario retrieval as well. In such a case no digital object and no input/output

formats would be specified. The only input data would be the information in the parameter list indicating scenario retrieval request.

```
MigrateResult migrate(DigitalObject    digitalObject,
                      java.net.URI     inputFormat,
                      java.net.URI     outputFormat,
                      List<Parameter> parameters);
```

4.4 User Feedback Use-Case

Another use-case is the request for accepting feedback about the performed migration. The migration component is unable to judge the quality of the MOs produced by the migration services. Only their presence or absence determine success or failure at that stage.

Therefore, it is important to receive feedback from the user, denoting the quality of the produced MOs. The feedback could contain the rating value ranging from 0 to 10. Additionally, the user could specify the text description of the feedback, which could be later analyzed by the DP staff. Cases in which the feedback indicates total success could be used in order to increase the efficiency value of each migration service involved in the corresponding migration path. Such a mechanism is in accordance with the **learning** requirement. However, in cases when the migration failed, there is no possibility to identify the failing edge other than by testing each of them.

On the code level the feedback use-case is similar to that of scenario retrieval. The first parameter for the migrate method should indicate the feedback request and the following ones its content in the form of ranking and text description. Additional parameter in the list should correspond to a session identifier. Although this use-case can be performed manually by the user, in practice the Planets IF could provide means for accomplishing this task in a convenient way via web-interface.

Conclusion

The work in the thesis presented the format migration system which is be able to perform migrations of large-quantities of DO transparently for the user. It was designed primarily for the task of being integrated into an existing DP Framework. By deploying the system withing the Planets IF the framework would be enriched by the format migration functionality. The following sections describe open questions which would need further attention.

5.1 Outlook

Additional work must be dedicated to the methods of creating and adding the scenario bundles to the database. The task of bundle registration is performed by the Scenario Registration Component. Through this component's public API, contributors should be able to request the adding of their own scenario bundles. This step would involve specifying the endpoint of the migration performing service, the meta-information of the scenario and the parameters. Additional use-cases of such a component could be temporary deactivation, modification or deletion of the bundles upon request.

The creation of the bundles' contents depends on the type of the approach used by the corresponding migration services. This section will further focus on the bundles corresponding to the Replay Service.

5.1.1 Replay Bundles

Creation of the replay bundles involves specifying the location of the OS-Image, Tracefile and Emulator objects in the parameter list. These values should be formed during the migration scenario recording. Scenario recording would involve the steps similar to the ones described in the Replay Service section of the thesis. The major difference would be the usage of the interactive session recorder instead of the replayer.

During the session recording, the contributors should be able to choose the operating system image (depending on the desired original application(-s)), the DO container type and the corresponding emulator. The contributors would perform the migrations manually, and their input events would be saved to file.

After the recording is finished, the objects could be automatically packed into the parameter list. They would also be accompanied with the endpoint of the replay service. The meta-information should be formed according to the contributors description of the migration session. After all parts of the replay bundle are ready, it can be automatically registered by a call to the Scenario Registration Component. After its registration, the corresponding scenario would be available for the Migration Component as one of the edges of the migration path graph.

The user-friendliness is a crucial aspect of the application performing the creation of replay bundles. Recording of the session should be available to a wide-range of contributors with no in-depth knowledge of computer science. By connecting to the replay archives through the Replay Bundle Creator Component, the web-based/graphical user interface would provide users with a list of available image files, emulators and any other secondary objects. It should also provide a means of entering brief description of the session, including the input/output formats, according to conventional format registry.

One of the important issues concerning session recording is the formation of the initial efficiency factor and its addition to the meta-information object. It may be formed, depending on the duration of the session, number of input events and value manually assigned by the contributor. Basing the factor on duration only might give false results due to varying system performance

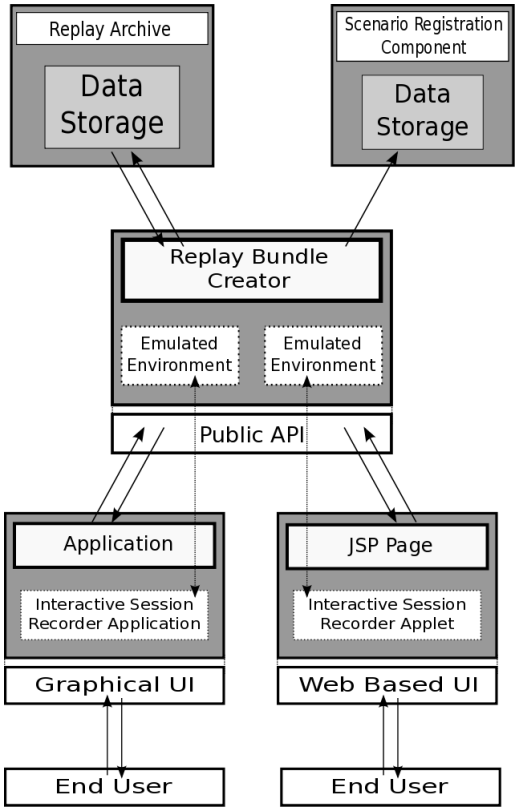

Figure 5.1: Connection between the contributors, replay bundle creation and registration process.

during recording and replaying. Duration value should then be accompanied by the average CPU/RAM statistic acquired periodically during recording.

The prototype of the replay bundle creator with web-based UI was developed in the scope of [15] and is depicted below.

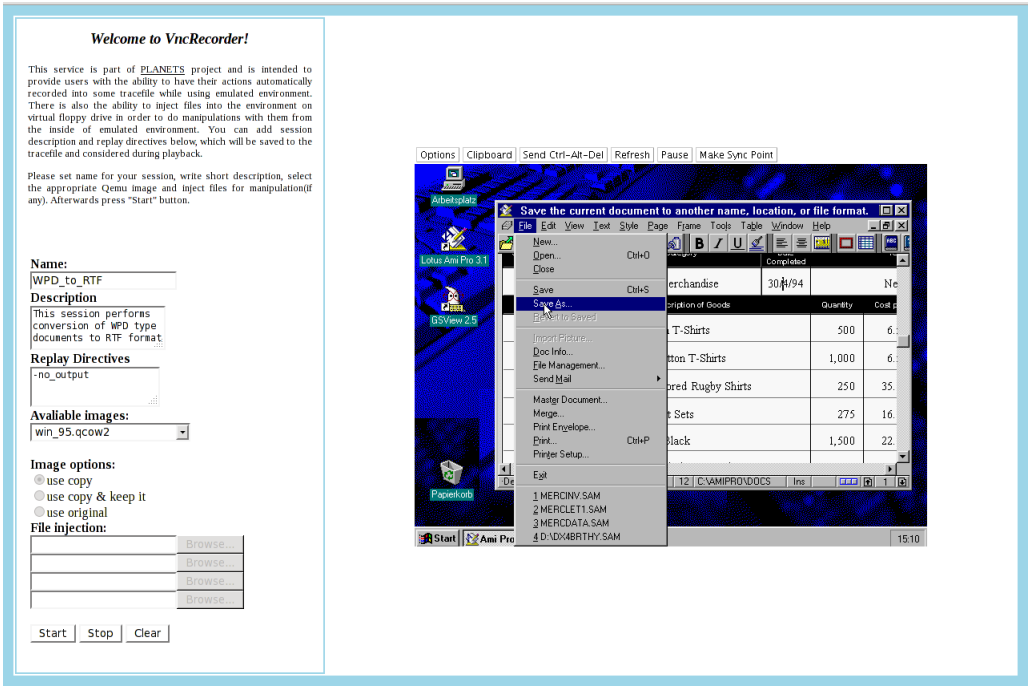

Figure 5.2: Contributor specified the name of the scenario and its description and chose the image file. The dummy DO was injected in order to produce the dummy MO manually and save the corresponding input events to a file.

5.1.2 Cyclic Regions in the Tracefile

If a certain tracefile represents the recorded migration session of a single DO, then it contains an exact region of input events, which is responsible for the production of the MO. This region could be repeated depending on the number of DOs injected for migration. On the low level it would represent the markers in the tracefile, which denote the existance of a cycle. In such a case these markers should be specified during the session recording by the contributor.

```
9 id=503 button=0 when=1307637076988 modifiers=0 type=java.awt.
    event.MouseEvent y=93 x=469
14 id=503 button=0 when=1307637077002 modifiers=0 type=java.awt.
    event.MouseEvent y=93 x=469
/* REPEAT SECTION BEGIN*/
415 id=501 button=1 when=1307637077417 modifiers=16 type=java.
    awt.event.MouseEvent y=93 x=469
```

```
19 id=502 button=1 when=1307637077436 modifiers=16 type=java.awt
    .event.MouseEvent y=93 x=469
710 id=401 when=1307637078146 keycode=10 keychar=10 modifiers=0
    type=java.awt.event.KeyEvent
.....................................................
.....................................................
.....................................................
74 id=402 when=1307637078220 keycode=10 keychar=10 modifiers=0
    type=java.awt.event.KeyEvent
516 id=503 button=0 when=1307637078736 modifiers=0 type=java.awt
    .event.MouseEvent y=93 x=469
72 id=503 button=0 when=1307637078808 modifiers=0 type=java.awt.
    event.MouseEvent y=93 x=469
/* REPEAT SECTION END*/
37 id=503 button=0 when=1307637078845 modifiers=0 type=java.awt.
    event.MouseEvent y=93 x=469
10 id=503 button=0 when=1307637078855 modifiers=0 type=java.awt.
    event.MouseEvent y=93 x=469
13 id=503 button=0 when=1307637078868 modifiers=0 type=java.awt.
    event.MouseEvent y=94 x=469
10 id=503 button=0 when=1307637078878 modifiers=0 type=java.awt.
    event.MouseEvent y=94 x=471
13 id=503 button=0 when=1307637078891 modifiers=0 type=java.awt.
    event.MouseEvent y=94 x=472
```

If the interactive session replayer processing this tracefile encounters the markers, it repeats the inner region for the specified number of times. The number of iterations would depend on the number of DOs. The corresponding value could be passed on to the interactive session replayer as an execution parameter.

However, such a strategy would require proper automatic file naming by the cyclic region of input events. Since the region is being repeated, then it would always produce the MOs under the same file name and location. In such a case each MO would be constantly overwritten by its successor.

Further studies should investigate this possibility. If this strategy were considered unreliable for large-scale migrations, then the session recording would assume future migrations to consist of only single DOs. In this case the migrations of large number of DOs could be done via iterative invocations of the replayer service.

Appendix

5.2 Hard Disk Creation Script

```
#!/usr/bin/bash
set -o errexit
set -o noclobber

# PARAM/VAR HANDLING:
usage='hdd_create.sh fsys size hdd'
if [ "$#" -eq 0 ]; then echo -e "$usage"; exit 0; fi
fsys="$1"
size="$2"
hdd="$3"
dev=''
fd=''
ok=0

# CLEANUP HANDLING
cleanup()
{
  code=$?

  set +e
  sync; losetup -d "$dev"; sync;
  exec {fd}>&-
  if [ "$ok" -ne 1 ]; then rm -f "$hdd"; fi
  set -e

  exit $code
}

# GET FREE LOOP DEVICE
getdev()
{
```

```
  secs=60

  set +e
  for (( i=0; i<$secs; ++i ))
  do
    dev=$(losetup "$@")
    if [ -b "$dev" ]; then return 0; fi
    sleep 1
  done
  set -e

  false
}

# CREATING HDD FILE:
exec {fd}>"$hdd"
trap cleanup ERR INT KILL TERM QUIT
dd if=/dev/zero of=/dev/fd/"$fd" bs=256K count=$(($size*4))

# MAKING PARTITION:
getdev --show -f "$hdd"
sfdisk -D "$dev" << EOF
,,$fsys
EOF

# FORMATTING PARTITION:
sync; losetup -d "$dev"; sync;
getdev --show -f -o 32256 "$hdd"

fmt=""
case $fsys in
  6) fmt="mkdosfs $dev";;
  b) fmt="mkdosfs -F 32 $dev";;
  *) false
esac
$fmt

ok=1
echo "HDD CREATED"
cleanup
```

5.3 Hard Disk Input/Output Script

```
#!/usr/bin/bash
set -o errexit
```

```
# PARAMS/VARS HANDLING:
usage='hdd_io.sh op hdd io'
if [ "$#" -eq 0 ]; then echo -e "$usage"; exit 0; fi
op="$1"
hdd="$2"
shift 2
mpt=''
dev=''



# CLEANUP HANDLING
trap cleanup ERR INT KILL TERM QUIT
cleanup()
{
  code=$?

  set +e
  sync
  pumount "$dev"
  sync
  losetup -d "$dev"
  sync
  rm -r -f "$mpt"
  set -e

  exit $code
}



# GET FREE LOOP DEVICE
getdev()
{
  secs=60

  set +e
  for (( i=0; i<$secs; ++i ))
  do
    dev=$(losetup "$@")
    if [ -b "$dev" ]; then return 0; fi
    sleep 1
  done
  set -e

  false
```

```
}



# BIND AND MOUNT HDD
if [ ! -e "$hdd" ]; then false; fi
getdev --show -f -o 32256 "$hdd"
mpt="/media/.${$}${RANDOM}.tmp"
pmount "$dev" $(basename "$mpt");



# DATA INPUT/OUTPUT
if [ "$op" = 'i' ]; then cp -r -n "$@" "$mpt";
elif [ "$op" = 'e' ]; then cp -r -n "$mpt"/* "$1";
else false; fi
echo "I/O SUCCESS"
cleanup
```

5.4 Migration Component

```
package eu.planets_project.services.migration.ufcmigrate;

import java.io.IOException;
import java.lang.InterruptedException;
import java.io.File;
import java.net.URISyntaxException;
import java.net.URI;
import java.net.URL;
import java.lang.Integer;
import java.lang.Long;
import java.lang.Process;
import java.util.ArrayList;
import java.util.Random;
import java.util.List;
import java.util.TreeSet;
import java.util.Properties;
import java.util.logging.Logger;
import javax.ejb.Stateless;
import javax.jws.WebService;
import javax.annotation.Resource;
import javax.xml.ws.WebServiceContext;
import javax.xml.ws.soap.MTOM;
import com.sun.xml.ws.developer.StreamingAttachment;
```

```
import eu.planets_project.ifr.core.techreg.formats.
    FormatRegistryFactory;
import eu.planets_project.ifr.core.techreg.formats.
    FormatRegistry;
import eu.planets_project.services.datatypes.MigrationPath;
import eu.planets_project.services.PlanetsServices;
import eu.planets_project.services.datatypes.Content;
import eu.planets_project.services.datatypes.DigitalObject;
import eu.planets_project.services.datatypes.Parameter;
import eu.planets_project.services.datatypes.ServiceDescription;
import eu.planets_project.services.datatypes.ServiceReport;
import eu.planets_project.services.datatypes.ServiceReport.Type;
import eu.planets_project.services.datatypes.ServiceReport.
    Status;
import eu.planets_project.services.migrate.Migrate;
import eu.planets_project.services.migrate.MigrateResult;
import eu.planets_project.services.utils.ProcessRunner;
import eu.planets_project.services.utils.ServiceUtils;
import eu.planets_project.services.utils.ZipUtils;
import eu.planets_project.services.migration.ufcmigrate.
    MigrationUnit;




/**
 * @author Isgandar Valizada
 */
@Stateless
@MTOM
@StreamingAttachment(parseEagerly=true, memoryThreshold=
    ServiceUtils.JAXWS_SIZE_THRESHOLD)
@WebService(name = UfcMigrate.NAME, serviceName = Migrate.NAME,
    targetNamespace = PlanetsServices.NS, endpointInterface = "eu
    .planets_project.services.migrate.Migrate")
public class UfcMigrate implements Migrate
{
  /** The service name. */
  public static final String NAME = "UfcMigrate";

  /** planets format registry  */
  public static final FormatRegistry fmtReg =
      FormatRegistryFactory.getFormatRegistry();

  /** logging performing object */
    private static final Logger log = Logger.getLogger(
        UfcMigrate.class.getName());
```

```
/** logging performing object */
  private static final MigrationUnit migrUnit = new
     MigrationUnit();



/**
   * {@inheritDoc}
   * @see eu.planets_project.services.migrate.Migrate#migrate(
      eu.planets_project.services.datatypes.DigitalObject,
   *       java.net.URI, java.net.URI, eu.planets_project.
      services.datatypes.Parameter)
   */
public MigrateResult migrate(DigitalObject digitalObject, URI
    inputFormat, URI outputFormat, List<Parameter> parameters)
{
  DigitalObject migratedObject = null;
  ServiceReport migrationReport = null;

  try
  {
    migratedObject = migrUnit.doMigration(digitalObject,
       inputFormat, outputFormat);
    migrationReport = new ServiceReport(Type.INFO, Status.
       SUCCESS, "migration performed, successful execution");
  }
  catch (Exception e)
  {
    e.printStackTrace();
    migrationReport = new ServiceReport(Type.ERROR, Status.
       SUCCESS, "error occured, details in server log");
  }

  return new MigrateResult(migratedObject, migrationReport);
}



/**
   * {@inheritDoc}
   * @see eu.planets_project.services.PlanetsService#describe
      ()
   */
  public ServiceDescription describe()
{
      ServiceDescription.Builder mds = new ServiceDescription.
         Builder(NAME, Migrate.class.getCanonicalName());
      mds.classname(this.getClass().getCanonicalName());
```

```
        mds.author("Isgandar Valizada <I.Valizada@googlemail.com
            >");
    mds.description("UfcMigrate service.");

    FormatRegistry fmtRegistry = FormatRegistryFactory.
        getFormatRegistry();
    List<MigrationPath> mgPaths = new ArrayList<MigrationPath>()
        ;
    mgPaths.add(new MigrationPath(fmtReg.createExtensionUri("ANY
        "), fmtReg.createExtensionUri("ANY"), null));
    mds.paths(mgPaths.toArray(new MigrationPath[] {}));

    return mds.build();
    }
}
```

5.5 Replayer Unit

```
package eu.planets_project.services.migration.ufcmigrate;

import java.io.IOException;
import java.io.File;
import java.io.*;
import java.net.*;
import javax.xml.parsers.*;
import javax.xml.xpath.*;
import javax.xml.xpath.XPathExpressionException;
import org.w3c.dom.*;
import org.xml.sax.*;
import org.apache.commons.io.*;
import java.lang.Integer;
import java.lang.Runtime;
import java.lang.Process;
import java.util.Properties;
import java.util.ArrayList;
import java.util.UUID;
import java.util.zip.*;
import eu.planets_project.ifr.core.techreg.formats.
    FormatRegistryFactory;
import eu.planets_project.ifr.core.techreg.formats.
    FormatRegistry;
import eu.planets_project.services.datatypes.DigitalObject;
import eu.planets_project.services.utils.ProcessRunner;
import eu.planets_project.services.utils.DigitalObjectUtils;
```

```
import eu.planets_project.services.datatypes.
    DigitalObjectContent;
import eu.planets_project.services.datatypes.Content;
import eu.planets_project.services.utils.ZipUtils;




/**
 * @author Isgandar Valizada
 */
public class MigrationUnit
{
  private static final int portMin = 5900;
  private static final int portMax = 6000;
  private static int vncPort = portMin;



  private class MUProperties
  {
    public String m_qemuExec;
    public String m_qemuBios;
    public String m_hddDisk;
    public String m_hddIo;
    public String m_vncplay;
    public String m_scenarios;

    public MUProperties() throws IOException
    {
      Properties props = new Properties();
      props.load(this.getClass().getResourceAsStream("/eu/
          planets_project/services/migration/ufcmigrate/
          ufcmigrate.properties"));

      m_qemuExec = props.getProperty("mu.qemu.exec");
      m_qemuBios = props.getProperty("mu.qemu.bios");
      m_hddDisk = props.getProperty("mu.hdd.disk");
      m_hddIo = props.getProperty("mu.hdd.io");
      m_vncplay = props.getProperty("mu.vncplay");
      m_scenarios = props.getProperty("mu.scenarios");
    }
  }



  private class Scenario
  {
```

```
  public String m_image;
  public String m_tracefile;

  public Scenario(String scenarios, String inputFormat, String
      outputFormat) throws SAXException,
     ParserConfigurationException, IOException,
     XPathExpressionException
  {
    DocumentBuilderFactory docBuildFact =
       DocumentBuilderFactory.newInstance();
    docBuildFact.setNamespaceAware(true);
    DocumentBuilder docBuild = docBuildFact.newDocumentBuilder
       ();
    Document doc = docBuild.parse(scenarios);

    XPathFactory xpathFact = XPathFactory.newInstance();
      XPath xpath = xpathFact.newXPath();
      XPathExpression xpathExpr = xpath.compile("/xml/
         scenarios/scenario");

      Object exprRes = xpathExpr.evaluate(doc, XPathConstants.
         NODESET);
      NodeList nodeList = (NodeList) exprRes;

    for (int i = 0; i < nodeList.getLength(); ++i)
    {
      NamedNodeMap nodeAttrs = nodeList.item(i).getAttributes
         ();
      String in = nodeAttrs.getNamedItem("in").getNodeValue();
      String out = nodeAttrs.getNamedItem("out").getNodeValue
         ();

      if(in.equalsIgnoreCase(inputFormat) && out.
         equalsIgnoreCase(outputFormat))
      {
        m_image = nodeAttrs.getNamedItem("image").getNodeValue
           ();
        m_tracefile = nodeAttrs.getNamedItem("tracefile").
           getNodeValue();
      }
    }
  }
}



synchronized private Integer getFreeVncPort() throws Exception
{
  if(portMin >= portMax)
```

```
    return null;

  String localIp = null;
  try
  {
    localIp = InetAddress.getLocalHost().getHostAddress();
  }
  catch(UnknownHostException e)
  {
    e.printStackTrace();
    return null;
  }

  final int TRIES = 50;
  for(int i = 0; i < TRIES; ++i)
  {
    Socket vncSocket = null;
    try
    {
      vncSocket = new Socket(localIp, vncPort);
      vncPort = ((vncPort + 1) < portMax) ? ++vncPort :
          portMin;
    }
    catch (IOException e)
    {
      return vncPort;
    }
    finally
    {
      try
      {
        if(vncSocket != null)
          vncSocket.close();
      }
      catch(IOException e)
      {
        e.printStackTrace();
      }
    }
  }

  return null;
}



synchronized private File createTempDir() throws IOException
{
  final int TRIES = 20;
```

```
    for(int i = 0; i < TRIES; ++i)
    {
      File tmpDir = new File(System.getProperty("java.io.tmpdir
          ") + "/" + (System.nanoTime() + String.valueOf(UUID.
          randomUUID())).replaceAll("[^\\p{L}\\p{N}]", "") + ".
          ufcmigrate");
      if(tmpDir.mkdirs()) return tmpDir;
    }

    return null;
  }



  public DigitalObject doMigration(DigitalObject digitalObject,
     URI inputFormat, URI outputFormat)
  {
    DigitalObject migratedObject = null;
    File tmpDir = null;
    ProcessRunner qemu = null;
    Thread qemuThread = null;
    boolean created = false;

    try
    {
      // getting the file extensions from the DOB uri
      FormatRegistry fmtReg = FormatRegistryFactory.
          getFormatRegistry();
      String fmtInExt  = fmtReg.getFirstExtension(inputFormat).
          toLowerCase();
      String fmtOutExt = fmtReg.getFirstExtension(outputFormat).
          toLowerCase();

      // getting scenario & properties data
      MUProperties muProps = new MUProperties();
      Scenario scenario = new Scenario(muProps.m_scenarios,
          fmtInExt, fmtOutExt);
      ArrayList<String> cmd = new ArrayList<String>();

      // creating main temporary directory
      tmpDir = createTempDir();

      // creating directory for data io
      File tmpIoDir = new File(tmpDir.getAbsolutePath() + "/" +
          "iodir");
      created = tmpIoDir.mkdirs();

      // copying empty hdd drive
      // not to harm the original
```

```
File tmpDisk = new File(tmpDir + "/" + "hdd");
FileUtils.copyFile(new File(muProps.m_hddDisk), tmpDisk);

// writing dob to a temporary file
// for injection purpose
File tmpDob = new File(tmpIoDir + "/" + "dob." + fmtInExt)
    ;
DigitalObjectUtils.toFile(digitalObject, tmpDob);

// injecting dob into copied hdd
cmd.add(muProps.m_hddIo);
cmd.add("i");
cmd.add(tmpDisk.getAbsolutePath());
cmd.add(tmpDob.getAbsolutePath());
(new ProcessRunner(cmd)).run();
cmd.clear();

// starting qemu with vnc enabled and
// hdd injected, bios is optional
Integer freeVncPort = getFreeVncPort();
cmd.add(muProps.m_qemuExec);
cmd.add("-vnc");
cmd.add(":" + new Integer(freeVncPort - portMin));
cmd.add("-drive");
cmd.add("file=" + scenario.m_image + ",snapshot=on");
if(muProps.m_qemuBios != null)
{
  cmd.add("-bios");
    cmd.add(muProps.m_qemuBios);
}
cmd.add("-hdb");
cmd.add(tmpDisk.getAbsolutePath());
qemu = new ProcessRunner(new ArrayList<String>(cmd));
qemuThread = new Thread(qemu);
qemuThread.start();
cmd.clear();

// running vncplay and waiting
// for it to finish
cmd.add("java");
cmd.add("-jar");
cmd.add(muProps.m_vncplay);
cmd.add("HOST");
cmd.add(InetAddress.getLocalHost().getHostAddress());
cmd.add("PORT");
cmd.add((new Integer(freeVncPort)).toString());
cmd.add("autoplay");
cmd.add("true");
cmd.add("tracefile");
```

```
cmd.add(scenario.m_tracefile);
ProcessRunner vncplay = new ProcessRunner(cmd);
vncplay.run();
cmd.clear();

// extracting container contents
cmd.add(muProps.m_hddIo);
cmd.add("e");
cmd.add(tmpDisk.getAbsolutePath());
cmd.add(tmpIoDir.getAbsolutePath());
(new ProcessRunner(cmd)).run();
cmd.clear();

// identifying migrated object(-s)
ArrayList mobs = new ArrayList<File>();
for(File fl: tmpIoDir.listFiles())
  if(FilenameUtils.removeExtension(fl.getName()).
      equalsIgnoreCase("mob"))
    mobs.add(fl);

// returning (if any) single file
// or zip archive
if(mobs.size() == 1)
  migratedObject = new DigitalObject.Builder(Content.
      byValue((File) mobs.get(0))).format(outputFormat).
      title("mob." + fmtOutExt).build();
else
  if(mobs.size() > 1)
  {
    File zipRes = new File(tmpIoDir.getAbsolutePath() +
        "/" + "mob.zip");
    ZipOutputStream zipData = new ZipOutputStream(new
        FileOutputStream(zipRes));

    for(File mob: (ArrayList<File>) mobs)
    {
      ZipEntry mobEntry = new ZipEntry(mob.getName());
      zipData.putNextEntry(mobEntry);

      try
      {
        CopyUtils.copy(new FileInputStream(mob.
            getAbsolutePath()), zipData);
      }
      catch (IOException e)
      {
        e.printStackTrace();
      }
```

```
            zipData.closeEntry();
          }

          zipData.flush();
          zipData.close();

          migratedObject = new DigitalObject.Builder(Content.
              byValue((zipRes))).format(fmtReg.createExtensionUri
              ("zip")).title("mob.zip").build();
        }
    }
    catch(Exception e)
    {
      e.printStackTrace();
    }

    // cleanup iff tmp dir
    // was created by us
    if(created)
    {
      try
      {
        FileUtils.deleteDirectory(tmpDir);
      }
      catch(IOException e)
      {
        e.printStackTrace();
      }
    }

    if(qemu != null)
      qemu.setTimeout(0);

    return migratedObject;
  }
}
```

References

[1] John Garret et al., *Preserving Digital Information: Report of the Task Force on Archiving of Digital Information.* Commissioned by the Commission on Preservation and Access and the Research Libraries Group, Inc., Washington DC: Commission on Preservation and Access, May 01, 1996

[2] Adrian Brown, *Selecting storage media for long-term preservation.* The National Archives of UK, Digital Preservation Guidance Note: 2, August, 2008

[3] Terry Kuny, *A Digital Dark Ages? Challenges in the Preservation of Electronic Information.* 63RD IFLA Council and General Conference, August 27, 1997

[4] Ken Quick and Mike Maxwell, *Ending Digital Obsolescence.* Affiliated Computer Services, Inc. (ACS). Dallas, Texas, January 20, 2005

[5] Remco Verdegem and Jeffrey van der Hoeven *Emulation: To be or not to be.* IS and T Conference on Archiving 2006. Ottawa, Canada, May 23-26, 2006

[6] Stewart Granger, *Digital Preservation and Emulation: from theory to practice.* Cultural Heritage and Technologies in the Third Millenium: ICHIM 2001, September, 2001

[7] Fabrice Bellard, *QEMU, a Fast and Portable Dynamic Translator.* FREENIX Track: 2005 USENIX Annual Technical Conference, 2005

[8] Jeffrey van der Hoeven, *Dioscuri: emulator for digital preservation.* D-Lib Magazine, 13(11/12), Corporation for National Research Initiatives (CNRI), 2007

[9] Dirk von Suchodoletz *Requirements towards Emulation as a Long-term Preservation Strategy.* Faculty of Engeneering, University of Freiburg, Germany, http://hdl.handle.net/10760/14860 (last accessed: 29 June 2011) July, 2009

[10] Nana Tchayep, A, *Emulatoren-Testing fuer die digitale Langzeitarchivierung.* Master Thesis, Faculty of Engeneering, University of Freiburg, Germany 3-Mar-2011

[11] Randolph Welte, *Funktionale Langzeitarchivierung digitaler Objekte. Entwicklung eines Demonstrators zur Internetnutzung emulierter Ablaufumgebungen.* Dissertation zur Erlangung des Doktorgrades der Fakultaet fuer Angewandte Wissenschaften der Alber-Ludwigs-Universitaet Freiburg im Breisgau July, 2008

[12] Klaus Rechert, Dirk von Suchodoletz, Randolph Welte, Maurice van den Dobbelsteen, Bill Roberts, Jeffrey van der Hoeven et al., *Novel Workflows for Abstract Handling of Complex Interaction Processes in Digital Preservation.* iPRES 2009: the Sixth International Conference on Preservation of Digital Objects. October 05, 2009

[13] Tristan Richardson, (James Weatherall, Andy Harter and Ken Wood also helped in the design of the RFB protocol) *The RFB Protocol.* RealVnc Ltd. November, 2010

[14] Ross King, Rainer Schmidt, Andrew N. Jacksonm, Carl Wilson and Fabian Steeg *The Planets Interoperability Framework* Proceedings of the 13th European Conference on Digital Libraries (ECDL09) 2009

[15] Klaus Rechert, Dirk von Suchodoletz, Randolph Welte, Felix Ruzzoli, Isgandar Valizada, *Reliable Preservation of Interactive Environments and Workflows.* The European Conference on Research and Advanced Technology for Digital Libraries, The 14th conference, University of Glasgow. September, 2010

[16] Nickolai Zeldovich, Ramesh Chandra *Interactive Performance Measurement with VNCplay.* Computer Science Department, Standford University.

[17] The National Archives TNA *The technical registry PRONOM Online resource.* http://www.nationalarchives.gov.uk/pronom (last accessed: 29 June 2011) 2010

[18] Sven Koenig and Maxim Likhachev, *Fast Replanning for Navigation in Unknown Terrain.*, IEEE Transactions on Robotics and Automation 2002

[19] Mario Philips, *Entwurf und Implementierung eines Softwarearchivs fuer die digitale Langzeitarchieveirung*. Diplomarbeit, University of Freiburg, http://www.ks.uni-freiburg.de/download/diplomarbeit/SS10/08-sw-arch-mphilipps/ (last accessed: 29 June 2011) 23 Juli 2010