



Keeping Emulation Environments Portable

FP7-ICT-231954

Requirements and design documents for services and
architecture of emulation framework

Part A: Requirements document for services and architecture of
emulation framework

Deliverable number	D2.2a (part of bundle of two documents: D2.2a, D2.2b)
Nature	Report
Dissemination level	PU
Delivery date	Due: M9 (October 2009) Actual: 23 April 2010
Status	Final
Work package number	WP2
Lead beneficiary	CSM
Author(s)	Winfried Bergmeyer, Andreas Lange (CSM) Jeffrey van der Hoeven, Bart Kiers (KB) Bram Lohman (TES)



Document history

Revisions

Version	Date	Author	Changes
0.1	09-05-2009	Winfried Bergmeyer (CSM)	Created initial version
0.2	05-07-2009	Winfried Bergmeyer (CSM)	Updated URD based on comments from KB
0.3	23-09-2009	Bram Lohman (TES)	Updated requirements list based on The Hague meeting on 17-09-2009
0.4	16-10-2009	Jeffrey van der Hoeven (KB)	Updated URD with user definitions, scenarios and introduction
0.5	23-10-2009	Bart Kiers (KB) Jeffrey van der Hoeven (KB)	Updated scenarios and matched with requirements. Added description of EF.
0.6	26-10-2009	Winfried Bergmeyer (CSM)	Updated scenarios and use cases (UML)
0.7	30-10-2009	Bram Lohman (TES)	Revised and merged scenarios; revised requirements list; general spelling and grammar check; initial revision abbreviations/glossary. Implemented changes discussed at The Hague meeting on 28-10-2009
0.8	03-11-2009	Jeffrey van der Hoeven (KB)	Included executive summary; final review of changes
0.9	04-11-2009	Winfried Bergmeyer (CSM)	Update use cases (UML)
1.0	06-11-2009	Jeffrey van der Hoeven (KB)	Accepted last changes; corrected page numbering; updated illustration of emulator download overview
1.1	27-11-2009	Jeffrey van der Hoeven (KB)	Incorporated remarks from DNB (reviewer)
1.2	23-04-2010	Jeffrey van der Hoeven (KB)	Updated executive summary based on reviewer's comments (EU)

Reviews

Date	Name	Result
19-11-2009	Marcus Dindorf (DNB)	Approved with minor changes
13-11-2009	Jean-Philippe Humblot (BnF)	Approved with no changes

Signature/Approval

Approved by (signature)	Date

Accepted by at European Commission (signature)	Date

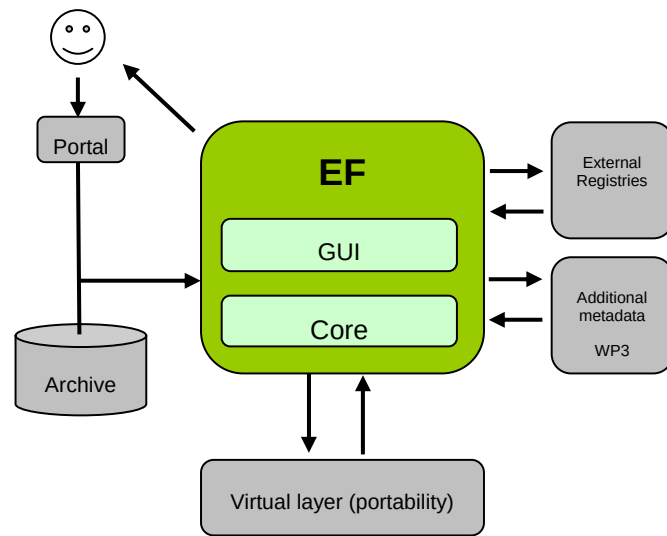
1 Executive Summary

This document describes the user requirements (functional and non-functional) for the Emulation Framework (EF) and forms the foundation of the framework and services to be developed. The figure below depicts the EF and its boundaries with other KEEP components and the outside world. With EF we mean the KEEP component that takes care of the recreation of an emulated environment in an automated and user-friendly way. This section summarizes the most important requirements defined for the EF in this document.

System users

The requirements of the EF are based on the needs of the users of the EF ranges from library & archive users, gamers and researchers to operators that have to manage the EF. Input has been gathered via an online survey and internal discussions. This led to several scenarios which have been translated into requirements. In the end, we defined three user roles:

- End-user : user of a cultural heritage institution where EF can be used.
- Administrator : the person that maintains and controls the EF.
- Machine : another computer system that would like to interact with the EF.



Functional requirements

The EF should be able to interact with existing archiving solutions. It should be possible to send a digital object and metadata to the EF which then renders this object in its original environment using emulation. The result should be presented to the user in an interactive way. To support emulation, at least two types of emulators will be incorporated, of which one is written in Java while another one should be capable of running on the KEEP VM.

The EF should be capable of dealing with at least two types of atomic digital objects (such as single files = mandatory) as well as two types of compound digital objects (such as applications and websites = desirable).

The end-user should be able to interact with the EF via a user-friendly GUI. This GUI should support functionality such as start and stop the emulation process and desirably offer some extra services (e.g. make screenshot, copy/paste text, record video/sound). The GUI should be accessible both via a stand-alone computer application (client) and in a web browser via a network connection.

The administrator should be able to configure the EF and update it with new emulators and external services. External services (if available) should be consulted to collect the required metadata.

Non-functional requirements

The GUI should be flexible enough to cope with future extensions. The EF itself should not rely on only one computer platform and deployment of the EF should be done via a single package containing all necessary components. A guide for end-users, as well as for administrators and developers should be available. The EF will be licensed under an open source licensing model.

Assumptions & Dependencies

Metadata will be gathered via several (external) sources. We assume that external services (e.g. metadata registries such as PRONOM, GDFR or PCR, and a software archive) will be available. If not, stubs will be created inside the EF which act like external services. If the externally available metadata is not sufficient, we rely on input from WP3 regarding additional bibliographic and preservation metadata.

Integration with the Planets Interoperability Framework is desired but depends on its availability. For example, the Planets Test bed is an external system and can only be used when it is hosted elsewhere and made available to us.

To recreate an emulation environment the original software is needed. We assume this software is available to us.

List of Related Documents

Description of Work	Overall project description
Architectural Design Document	Design for the EF
KEEP library user survey	Survey about library user needs

Abbreviations

Bibliothèque nationale de France	BnF
Koninklijke Bibliotheek	KB
Computerspiele Museum	CSM
Deutsche Nationalbibliothek	DNB
Tessella plc	TES
Work Package	WP
User Requirements Document	URD
Architectural Design Document	ADD
Emulation Framework	EF
Unified Modelling Language	UML
Graphical User Interface	GUI

2 Table of Contents

1 Executive Summary	4
2 Table of Contents	7
3 Introduction	9
Objectives and scope	9
Outline of this document	9
4 General description	11
User characteristics	12
4.1.1 Library user	12
4.1.2 Archiving user	14
4.1.3 (Computer) Gamer	15
4.1.4 Researcher	15
4.1.5 Emulator developer	15
4.1.6 Museum visitor	15
4.1.7 Operator	15
4.1.8 Machine-to-machine	15
Roles	15
5 Scenarios	17
Scenario 1: the end-user views an atomic digital object in the emulation framework	17
Scenario 2: the end-user views a compound digital object in the emulation framework	18
Scenario 3: the administrator installs and configures the emulation framework	18
Scenario 4: the end-user plays a game in the emulation framework, in several sessions	20
Scenario 5: the end-user cannot render a digital object	20
6 System context	21
Assumptions and Dependencies	21
6.1.1 Portal, digital archive and objects	22
6.1.2 External registries	22
6.1.3 Additional metadata	22
6.1.4 Software	22
6.1.5 Portability	23
7 Use cases	24
Use case 1: The end-user opens a document	24
Use case 2: The administrator sets up the EF	25

Use case 3: The administrator adds a new component.....	26
8 Functional requirements	27
8.1.1 GUI	27
8.1.2 External interfaces	28
8.1.3 Emulation process	28
8.1.4 Emulation environment	28
8.1.5 Usability	29
9 Non-functional requirements	30
9.1.1 Accessibility	30
9.1.2 Documentation.....	30
9.1.3 Licensing.....	30
10 Design and Implementation Constraints and Standards.....	31
11 Glossary.....	32

3 Introduction

This document describes the user requirements for the Emulation Framework (EF). A second (separate) document describes the design of the EF. Both documents (requirements plus design) form deliverable D2.2 of KEEP.

The requirements are based on a lengthy requirements elicitation process in which we investigated the user's needs concerning access to digital objects via emulation. Within the project all partners that will eventually make use of the EF were asked to contribute by writing user scenarios. Based on this input, use cases were defined and translated into requirements for the EF.

According to the Description of Work (DoW) the target groups for the EF are not only users in institutions like libraries, museums or archives but also individuals (DoW, Task 2.2 – Specification and design of services offered to the end user, p. 33). The sustainability risk evaluation of KEEP points out that, with respect to the limited market of cultural heritage institutions, KEEP could serve also the need for digital preservation and emulation solutions in companies and for individuals (DoW, p. 52).

Furthermore, an online survey was conducted amongst library users at the National Library of the Netherlands (KB). Users were asked how important they think digital publications are and in which way they would like to work with them. Furthermore, they were asked which kind of challenges they foresee in accessing digital content that has become obsolete.

The EF will be suitable for a wide range of users. To ensure all requirements are included, we made a distinction between various types of users. These user types have different needs concerning the desired functionalities of the EF. This has to be taken into account for during the development of the framework and the deployment of the EF at organisations.

Objectives and scope

The requirements defined in this document should guide the development of the EF in becoming highly functional and user-friendly. The current list of requirements reflects the needs of the users at the time of writing, but it is likely that changes will occur due to the research-oriented nature of this project.

This document is targeted at the EF and therefore does not address requirements for other components of the Emulation Access Platform (i.e. Transfer Tools Framework and Virtual layer).

Furthermore, this document only represents the technical aspect of the framework. It does not cover any legal, financial or organisational aspects that might be relevant when applying the EF to a specific organisation or to give access to its users.

Finally, assumptions are made about the availability of external services (e.g. metadata registries, software archive) on which the EF will rely. These are mentioned in more detail further on in this document.

Outline of this document

Before the requirements of the EF can be defined, a better understanding is needed of who the users of the system will be. The first chapter will address the various types of users and the way they will interact (i.e. their roles) with the system. In particular, some initial results from the library user survey are included to give a better understanding of what they think is required.

Based on the defined users, scenarios are described to define the way users might interact with the system. Following this the context of the EF as a system is projected and the scenarios are abstracted into use cases that define the use of the system on a higher level. Finally, the requirements are listed, organised into functional and non-functional lists.

At the end of this document a glossary is included describing the most important terms and definitions used in this document.

4 General description

The use of digital content is widespread and the variety of its appearance is enormous. Although many current computers can open most documents and applications, the threat is real that an increasing amount of content is becoming obsolete due to hardware deterioration, incompatible data types or lack of knowledge concerning old hardware and software (see figure 1). This became obvious when a large group of researchers in Europe were asked about the threats they think are important to counter¹. Eighty percent of the researchers noted the threat “lack of sustainable hardware, software or support of computer environment may make the information accessible” as either *important* or *very important*. Although this only entails researchers, other areas in information management painted a similar picture.

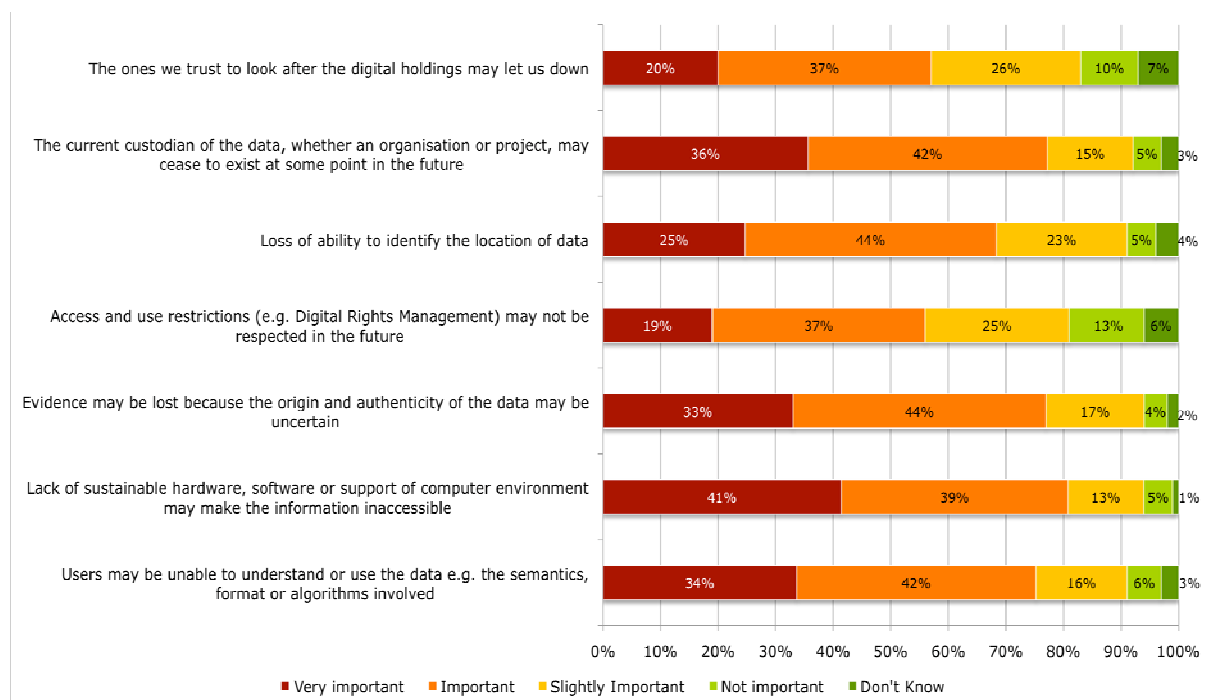


Figure 1: threats to preservation of digital data according to researchers (by PARSE.Insight)

This demonstrates the need for a more sustainable approach in accessing digital information. The KEEP Emulation Framework will try to counter the threat mentioned above by offering a system that gives access to both static and dynamic digital objects in their native computer environment. For this purpose, emulation will be used to recreate the native computer environment and the user will be offered additional services to allow a smooth and user-friendly interaction with the system.

For more information on the general outline of the EF, please see the DoW.

¹ PARSE.Insight *Interim Insight Report* (Sept 2009), available at: http://www.parse-insight.eu/downloads/PARSE-Insight_D3-5_InterimInsightReport_final.pdf

User characteristics

To understand what the EF should offer, a better understanding of the potential users of the system is needed. As four cultural heritage institutes (three libraries and one museum) are involved in the KEEP project, the first step is to identify their users and their needs. Further on other types of users are defined as well.

4.1.1 Library user

A survey has been distributed amongst a group of 400 library users of the National Library of the Netherlands (KB). The survey was available from the 16th of September 2009 until the 19th of October 2009, during which 106 people responded of whom 104 completed the survey; a response rate of 25%. To compare the results from the Netherlands with library users in other countries, the survey has also been distributed by the BnF and DNB, reaching French and German library users as well. At the time of writing, the results of the latter two distributions were not yet available, but it is expected that these results will be quite similar to those from the KB.

The full survey results will be available via the KEEP website. Hereafter, the highlights are shown that are relevant for the EF.

The first question of interest for the requirements elicitation is how library users would like to receive the requested publication (figure 2). The respondents answered that they are in favour of having both a printed and a digital copy of the publication. Twenty percent prefers digital above all else.

If you request a publication, how would you like to receive it?

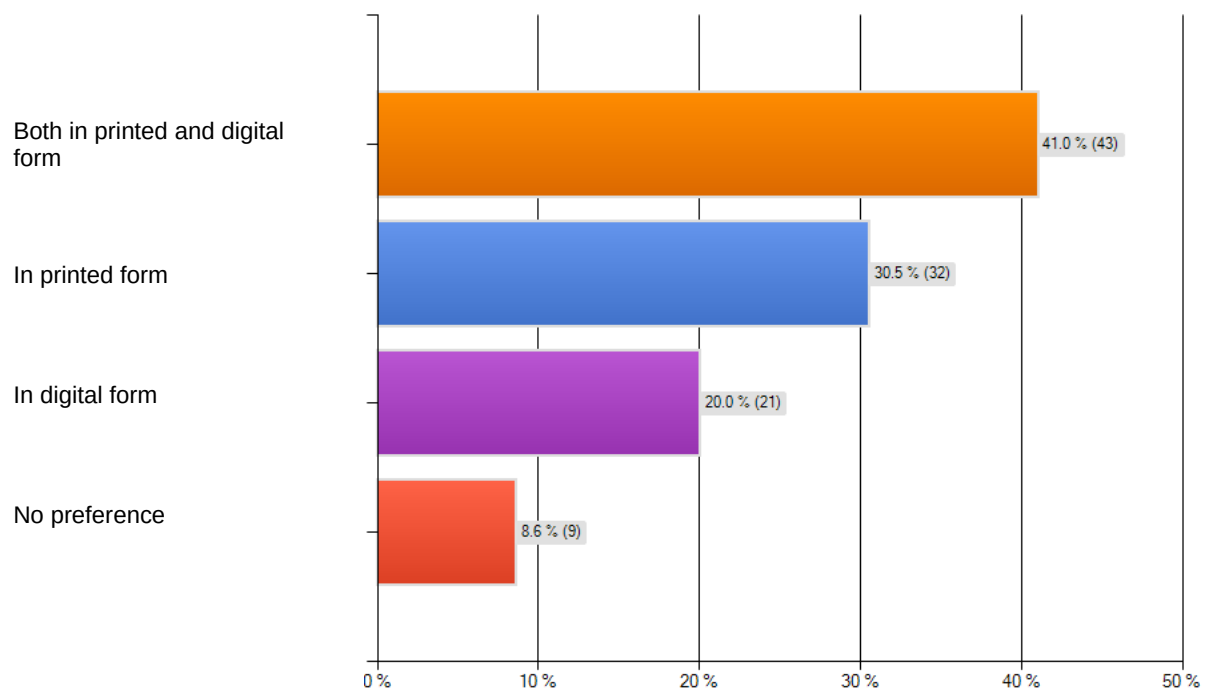
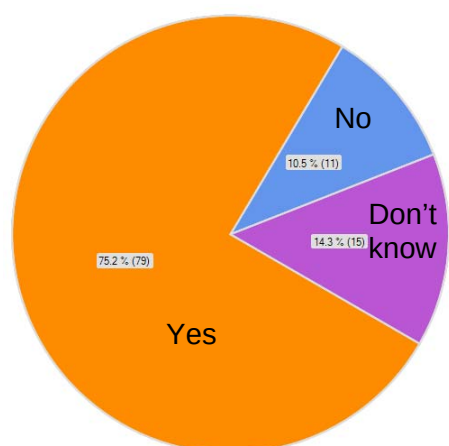


Figure 2: KEEP survey results (1)

Often, publications at libraries are not available in digital form yet, but in certain cases the printed publication contains digital supplements. These supplements vary from a floppy disk containing illustrations to a CD-ROM containing educational software that is highly dependent on a specific computer platform. We asked library users about their interest in those supplemental digital contents (figure 3).

Besides the publication, are you interested in supplemental digital files as well?



Based on the response there is a large interest in additional digital content. However, to access this content a special setup is often required, as the software cannot operate on current computers anymore.

Regarding user's own experience in dealing with old computer files and programs we asked them which kind of problems they expect to encounter (figure 4).

Figure 3: KEEP survey results (2)

What is the reason why you can't use the old files/applications anymore?

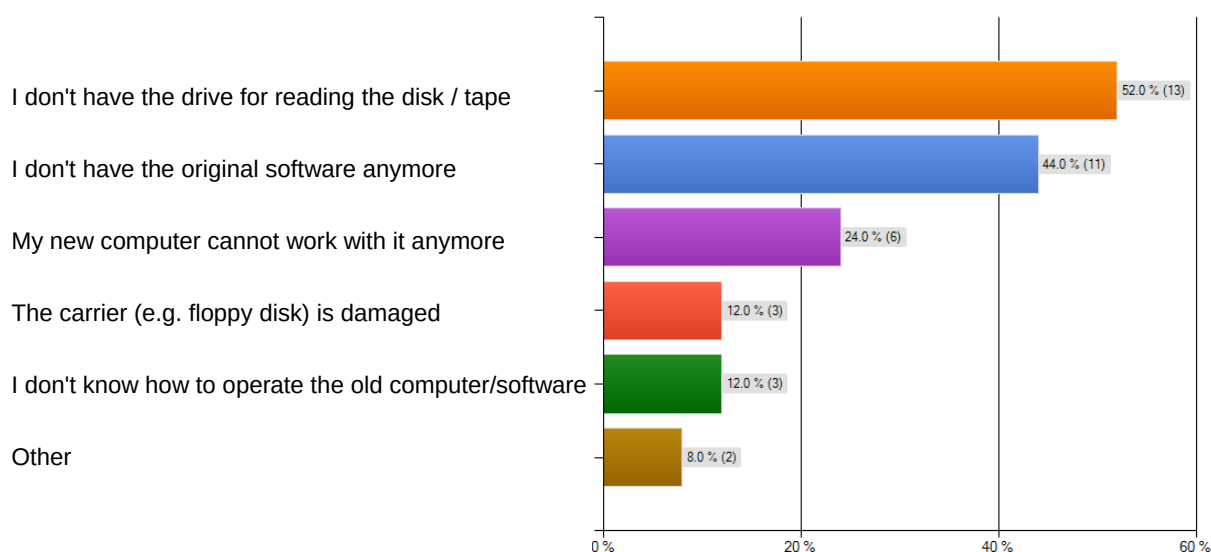


Figure 4: KEEP survey results (3)

Clearly, the main reasons mentioned by the library users are based on hardware and software problems. This indicates that older files and applications are becoming obsolete as current computer solutions are not appropriate for accessing older digital material.

Assuming access to old documents and applications is possible, we asked the library users which aspect of the digital object they are most interesting in (figure 5). The biggest interest is in text, which is not surprising for library users. However, 64% indicated images as an aspect to be preserved, whereas sound and functionality are mentioned in 26% and 15% of the responses, respectively. Although text and images can often be preserved via a migration process, sound and functionality are best preserved using emulation. Transcoding audio streams can be time consuming and often diminishes quality, while functionality is

application specific and cannot be migrated at all. Emulation seems the way forward in this area.

Which kind of aspects of that publication would you definitely want to keep?

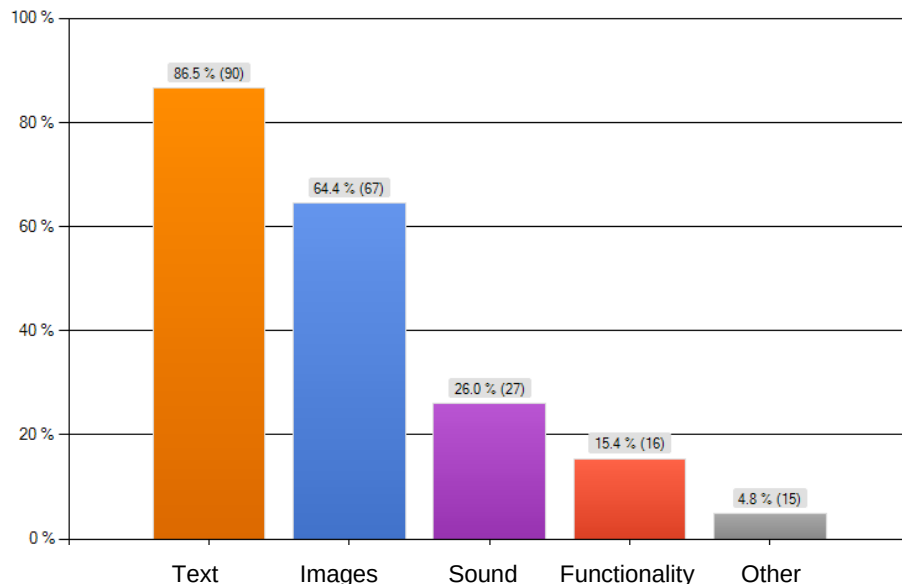


Figure 5: KEEP survey results (4)

We asked library users if they were still able to operate an older computer environment if it was available to them. Almost 40% answered “yes, but with a little support” and 18% answered “No, not at all”. Again, this clearly shows that even if people are offered an old computer again, they wouldn’t know how to interact with it. The EF can support people in operating with ancient computer environments.

Finally, we asked how this kind of “support” should look. Approximately 48% would prefer human assistance and 46% is in favour of having an online wizard or tutorial that helps them with the operating of older machines. Only 5% responded that support is not needed.

In summary, the respondents of the library users survey showed us that:

- there is a great interest in receiving publications in digital form
- additional digital files and applications stored on older media is of interest to them as well
- the majority of the respondents has problems accessing older files and applications
- text and images are their primary interest, but that sound and functionality definitely matter
- support is needed when operating with older computer environments.

These desires have to be taken into account when developing the EF.

4.1.2 Archiving user

Although KEEP does not include an archiving partner it does take archiving users into account. Users from archives are primarily interested in the original record because of its

legal status or simply because there is no substitute available. In those cases, migration is often not possible. Besides this special interest in the original record, in which the appearance of a record is of high importance, archiving users do not differ much from library users.

4.1.3 (Computer) Gamer

Computer games are special digital objects as they have a strong dependence on the software and hardware of the computer environment they run on. Games also feature an intense interaction with the user. The appearance and behaviour of a game is therefore crucial for how this user experiences the game. Computer games often put a high stress on computer systems and are therefore very useful test objects that should be taken into account.

4.1.4 Researcher

Research on digital artefacts (games, applications, simulations, multimedia etc.) is increasing. In a lot of cases emulation will be needed to access these sources. As researchers might also be interested in the context of the digital object and its environment, the EF will provide more contextual information (e.g. documentation) if available.

4.1.5 Emulator developer

Although emulator developers are not the intended users of the EF, they provide emulators that have to be taken into account by the EF. Therefore, emulators need to be created for specific computer environments and they need to be compatible with the EF.

4.1.6 Museum visitor

Just as with physical artefacts, certain digital objects and computer environments are becoming rare and therefore interesting to collect. The Computerspiele Museum (CSM) situated in Berlin has a large collection of computer games; to let their visitors experience these digital artefacts in its original setup, the EF could be used. Therefore, the EF should be capable of rendering older environments in such a way that computer games can be experienced during exhibitions.

4.1.7 Operator

The operator is the person that works in the back office (of a library, archive or museum) and has to deal with the day-to-day operation of the EF. They have to be able to control the EF and solve issues when they arise. They also secure the integration of the EF with other organisational systems (e.g. archiving systems).

4.1.8 Machine-to-machine

Although the end users of the EF are always people, the EF should provide an interface that is capable of handling external interaction with other systems (e.g. document management systems).

Roles

The abovementioned types cover a wide range of users of the EF. As many types overlap, these types were aggregated into three basic roles that a user can play:

- **End-user**
The end user role covers the library users, archiving users, gamers, researchers and museum visitors. This role has as main interest to get the system to render a digital object, and subsequently interact with this digital object via the system.

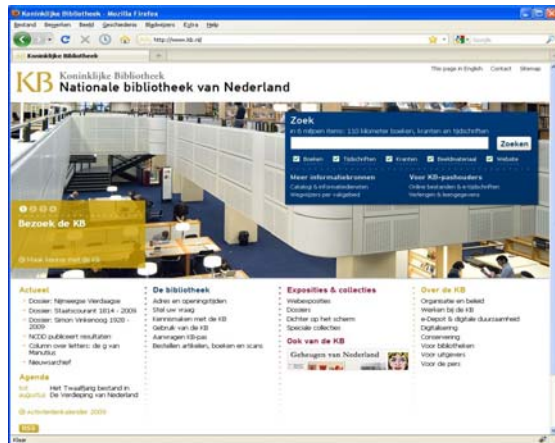
- Administrator
The administrator role represents the operator of the EF. This role covers any set up, configuration and maintenance of the system for use by end-users. The administrator itself will not tend to use the system as an end-user would.
- Machine
The machine role covers the machine-to-machine usage of the system. This role consists of any non-human interaction between the system and any external system.

The emulator developer user is not covered by any of the above roles as this kind of user is not directly using the EF but is rather involved in developing new tools for the EF. Focus on this user is left for future work.

5 Scenarios

This section contains several scenarios that describe how users, in one of the roles defined in the previous section, are envisioned to use the system. These scenarios have been used to form the basis of the requirements defined in section 8.

Scenario 1: the end-user views an atomic digital object in the emulation framework



After logging in to the portal of a library institution (figure 6), the user performs a search operation in the libraries' online catalogue with the use of specific query terms. This search returns a list of results that matches the query terms. From the results, the user selects a digital object representing a WordPerfect 5.1 document² that they would like to see rendered in the EF. The library catalogue passes the digital object and any associated metadata to the EF; the two systems communicate to ensure this object is handed over successfully.

Figure 6: Institutional portal to online catalogue

A new page showing the metadata (figure 7), as well as the pathway (the application needed to open the document, the operating system and the hardware architecture) needed to render the object is presented. This information is retrieved using external services such as DROID, PRONOM, Planets Core Registry or others. Not all pathways shown may be supported by the currently installed set of emulators, or software packages available from an external repository.

Once the user has selected a valid pathway, a new stand-alone window is launched by the system.

This new window displays the selected document in its original environment (figure 8). The window also presents a set of services to the user. Here a distinction is made between novice users and experienced users: the experienced user has access to more advanced settings and services; the novice user is limited to a basic set of functionality. This basic set includes making screenshots, copying text, recording a video, recording audio and keyboard logging. Furthermore, support is given to the user via an online wizard that helps the user operating the old emulated computer environment.

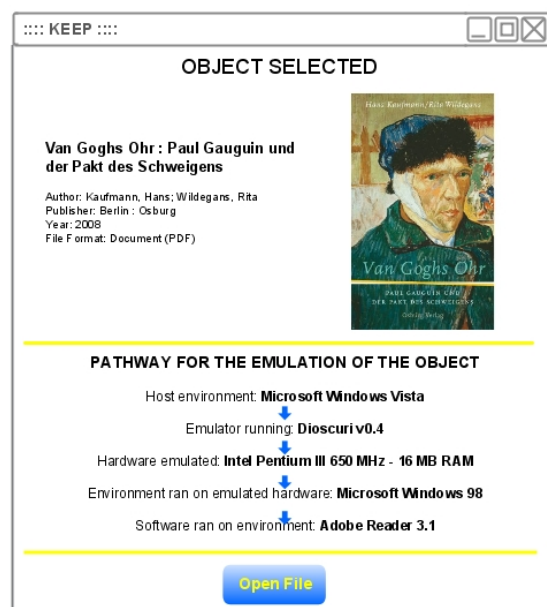


Figure 7: object metadata and pathways

² See also Report describing case studies (Planets PA/5 D3, 2008), http://www.planets-project.eu/docs/reports/Planets_PA5-D3_CaseStudies-final.pdf

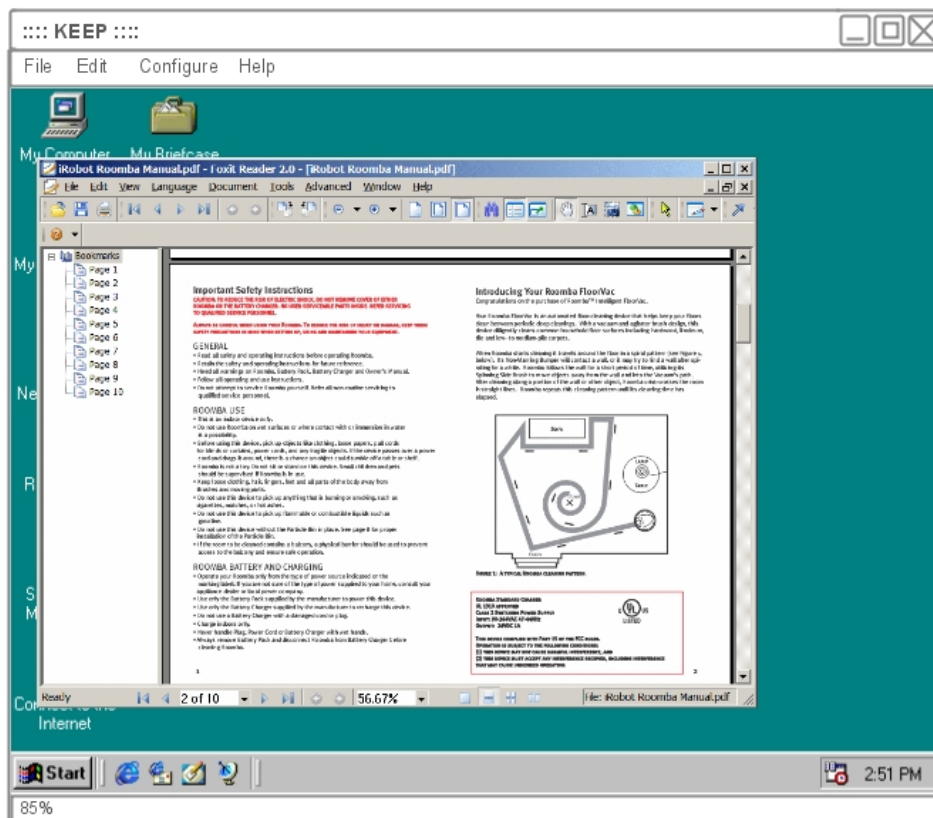


Figure 8: rendered digital object (example)

Scenario 2: the end-user views a compound digital object in the emulation framework

Using the same setup as in Scenario 1, the user searches for and selects a digital object representing a complete web page. This digital object is in the form of a WARC, a Web Archive container that comprises all types of web content including [HTML](#) web pages, [style sheets](#), [JavaScript](#), [images](#), and [video](#). The associated metadata defines a top-level entry point into the WARC (such as the main HTML page), which contains the relation between all the content.

The EF contacts external registries to determine the pathway, fulfilling all of the software and hardware needs for all the content included in the WARC. Once a valid pathway has been selected, the workflow continues as in Scenario 1: an appropriate software package fulfilling the software needs is downloaded and prepared from an external repository; the correct emulator is configured and set up to run the software package, and once these requirements are fulfilled the selected object is displayed in the selected environment.

Scenario 3: the administrator installs and configures the emulation framework

The system administrator of a library institution is preparing to install the EF to be used in their institution. They download the single installation package and follow the included instructions to install it on their system. Once installed, they log in to the EF as administrator for set up and configuration.

In the administrator set-up screen, several configuration options such as external registry connections (e.g. IP addresses) and end-user display options are present. Once completed,

the administrator selects “Check for Updates” from a list of commands. An “Update in Progress” dialog box is showed to indicate the communication between the EF and a remote server containing new software updates for the EF. In addition to textual information, the dialog box displays an active progress bar, showing the ongoing connection with the remote server, and a “Cancel” button.

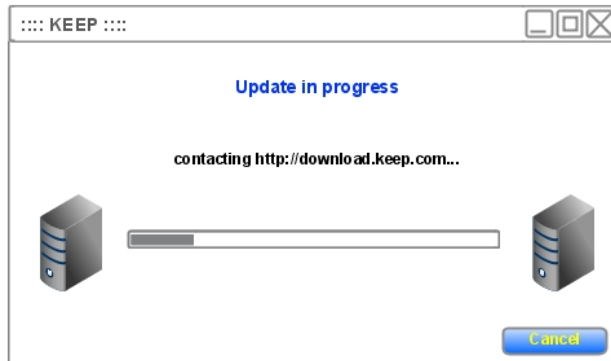


Figure 9: EF “update in progress” dialog box

representing a distinct component. Below the grid two buttons, named “Cancel” and “Download”, are displayed. The administrator checks the checkbox of the components they wish to download and finally clicks the “Download” button, which prompts the system to download the selected components and an automatic installation is launched which updates the EF.

Once the updating operation has finally terminated, the “Update in Progress” dialogue box is replaced by an “Updates Available” dialog box. This dialogue box displays the entire list of available emulators ready to be downloaded and installed within the EF.

Attributes of the grid includes the name, the version, the type, the author and the date of creation of the component to be downloaded, but double-clicking on a row provides more information. An unchecked check box is provided in each row,

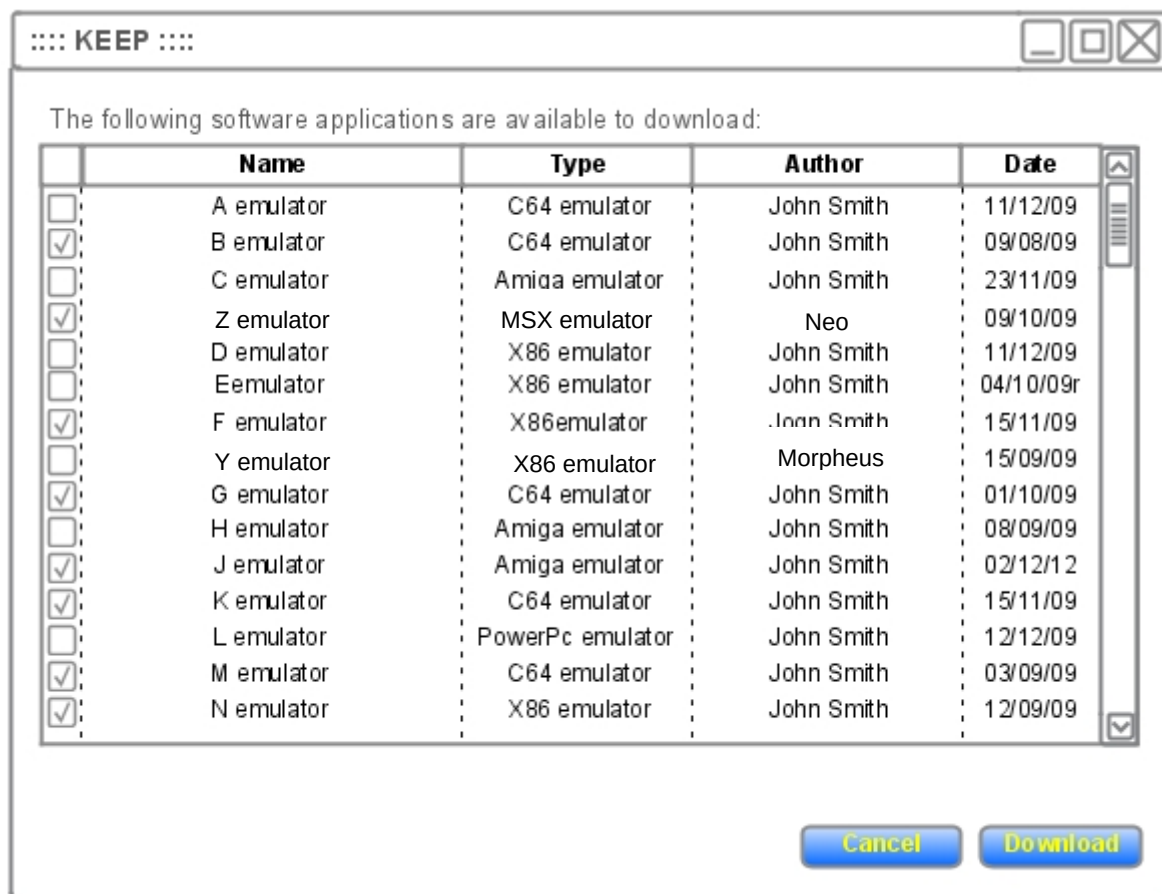


Figure 10: EF dialog box offering an overview of available emulators

Scenario 4: the end-user plays a game in the emulation framework, in several sessions

The end-user starts the EF to run an old console game for which they have the disk image available. Selecting the d64 file (Commodore 64 game), the EF attempts to identify the image using external registries. If successful, the EF shows the possible emulation pathways, ordered by user rating, allowing the end-user to select the one he/she wants. If the external registries are not available or unable to generate a pathway the EF allows the end-user to define a custom pathway to run the digital object. This pathway can be saved to disk, as well as uploaded to an external database.

The EF enables the end-user to select the options provided by the emulator, such as screen resolution and execution speed. During play, the EF allows the end-user to create screenshots, audio and video recordings, and maximise the screen display.

As it is a lengthy game, the end-user decides to continue playing at a later time. The EF saves the state of the emulator to a save file. Before exiting, the user also enters their rating of the pathway, which is added to the available ratings.

The next time the end-user runs the EF, he/she can load the state file, continuing the game where the end-user left of.

Scenario 5: the end-user cannot render a digital object

The end-user opens a digital object, but all pathways found by the EF are unsupported: the correct emulator is not available to render the object. Using the 'Contact' page in the EF, the end-user notifies the administrator of this deficiency in the configuration.

However, the end-user is still able to save the digital object, its associated metadata and their preferred (unsupported) emulation pathway information. This information is sent to another institution, also using the EF, where another user can load the saved information, and provided the second end-user's configuration supports the pathway (i.e. is able to run the required emulator), the second end-user can render the digital object successfully, including taking screenshots, audio and video recordings, logging keyboard activity, saving game states or downloading emulation pathway metadata for documentation. This functionality is available from a generic, emulator-independent menu.

6 System context

The EF is one of the major components of the Emulation Access Platform as described in the DoW. The task of the EF is to provide users access to digital objects of any kind via emulation.

Figure 11 shows a high-level overview of the system (green, in scope) and its boundaries (grey, outside scope). Within the EF a distinction is made in *Core* and *GUI*. The core is responsible for managing emulation processes while the GUI provides a rendering environment plus additional services to the user.

The interaction of the system with its boundaries is defined in the following, simplified view: when a user selects a digital object via the portal of the institution, the object plus metadata will be retrieved from the digital archive and transmitted to the EF. The EF will identify the object and will try to reconstruct which computer environment (operating system, hardware, applications, etc.) is needed to render the object. For this purpose, external registries such as PRONOM, Planets Core Registry or UDFR are consulted.

However, more information is needed to help the user operate an old computer environment (e.g. manuals, tutorials, keyboard overlays). KEEP will create a database containing a limited set of additional metadata (done by WP3, outside scope).

Finally, emulators need to be sustainable. WP4 (outside scope) will provide a virtual layer that allows emulators to run independently from their computer environment.

For all these external systems, the EF will define a common interface (API) or will implement the API offered by the external source.

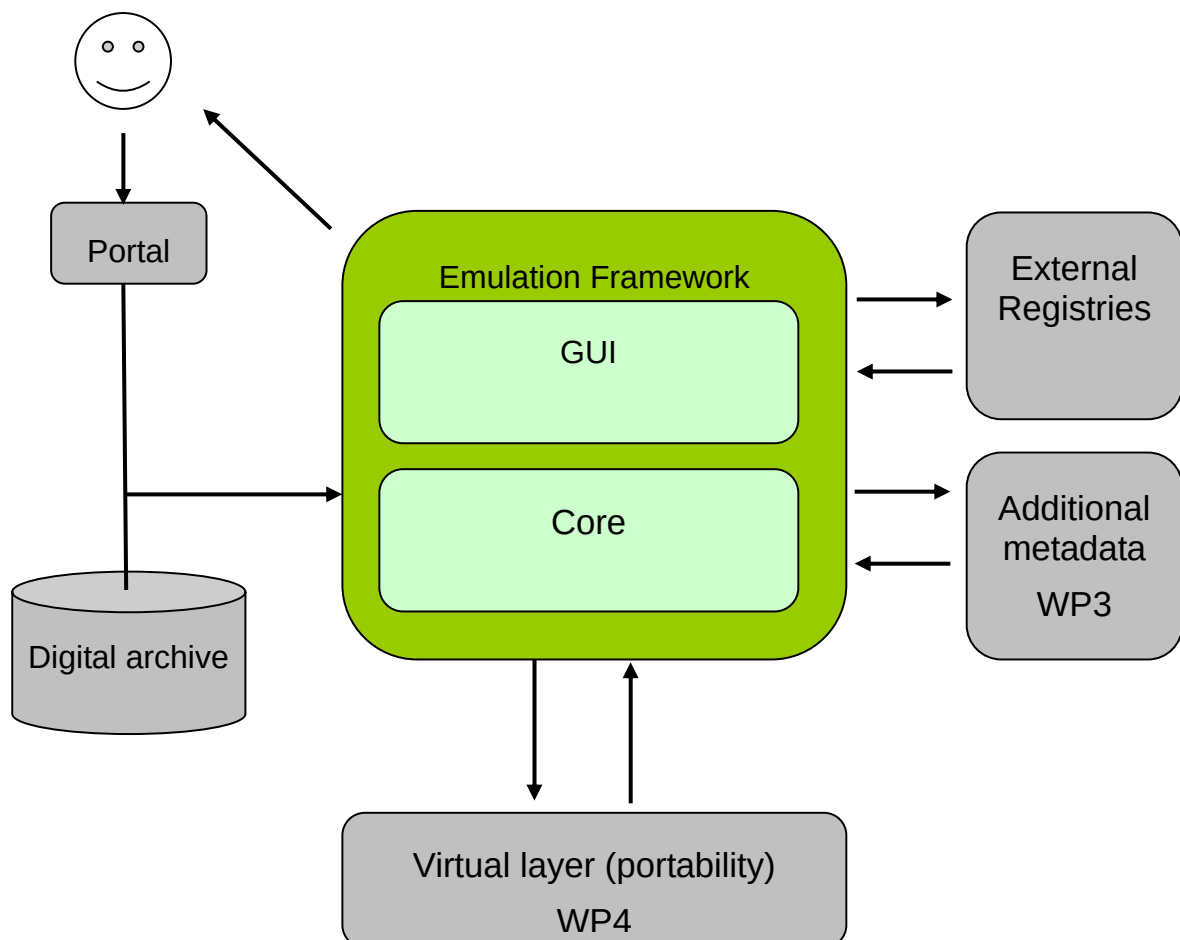


Figure 11: EF system overview and system-boundaries

Assumptions and Dependencies

6.1.1 Portal, digital archive and objects

The assumption has been made that users of the EF are able to select and retrieve a digital object via a portal from a digital archive or from any other digital collection available to the user. Both digital object and metadata will be made available to the EF using a predefined API.

6.1.2 External registries

To render a digital object, metadata about the object and its native computer environment is necessary. We will assume that several external registries will exist that can be used by the EF. Currently, three such registries are known that offer technical metadata about hardware and software dependencies.

PRONOM³ is a resource that can offer information about file formats, software products and other technical components required to support long-term access to electronic records and other digital objects of cultural, historical or business value.

Within Planets⁴, an extension of PRONOM is under development, called the Planets Core Registry (PCR). This registry will not only contain information about technical dependencies, but also offers services that can be applied to migrate or emulate a digital object. KEEP will make sure that it is registered here as well.

A third initiative for a registry is planned and is called the Unified Digital Formats Registry (UDFR)⁵. UDFR will provide sustainable distributed services to store, discover, and deliver representation information about digital formats. However, at the moment the UDFR only exists on paper, and it is unsure if KEEP will be able to use it in the near future.

Besides the above metadata registries, other external registries might be available in the future. One of those will be the Planets Test bed in which Preservation Tools (such as emulators and migration tools) can be tested and evaluated. Evaluation reports are available and can be used by other services (such as KEEP) to gather which tools operate properly. This information can support the EF in determining which emulator is best for a certain computer environment. Although the Test bed is not yet available as an operational service the EF should take advantage of its potential value.

If there is no external registry with information about a special file format or the file format is an image format, used for different environments and objects (for example FDI or iso) the EF should offer the user the option to enter an emulation pathway and store it for further use.

6.1.3 Additional metadata

Although the previously mentioned registries offer technical metadata to help the EF in constructing a rendering environment, more information is needed to ensure that the user can understand how to operate with that environment. Therefore, additional (bibliographic and preservation) metadata is needed. Work package 3 (tasks 3.1, 3.2 and 3.3) will address this lack of metadata and will develop a dedicated database that complements the information of other external registries.

³ PRONOM, <http://www.nationalarchives.gov.uk/PRONOM/Default.aspx>

⁴ Planets, <http://www.planets-project.eu>

⁵ UDFR, <http://www.gdfr.info/udfr.html>

6.1.4 Software

When an original computer environment has to be recreated using emulation, the original software such as BIOS, operating system, applications, drivers and other utilities is required. The EF will offer functionality to run this software, but it does not have the intention to provide this software. A lot of software is proprietary and cannot be shared without an appropriate license. The deployment of the EF will come without proprietary software. The institution at which the EF is deployed will need to provide the necessary software for the framework to enable emulation of the desired computer environment. The assumption has been made that some sort of “software archive” is available for the EF to search and access.

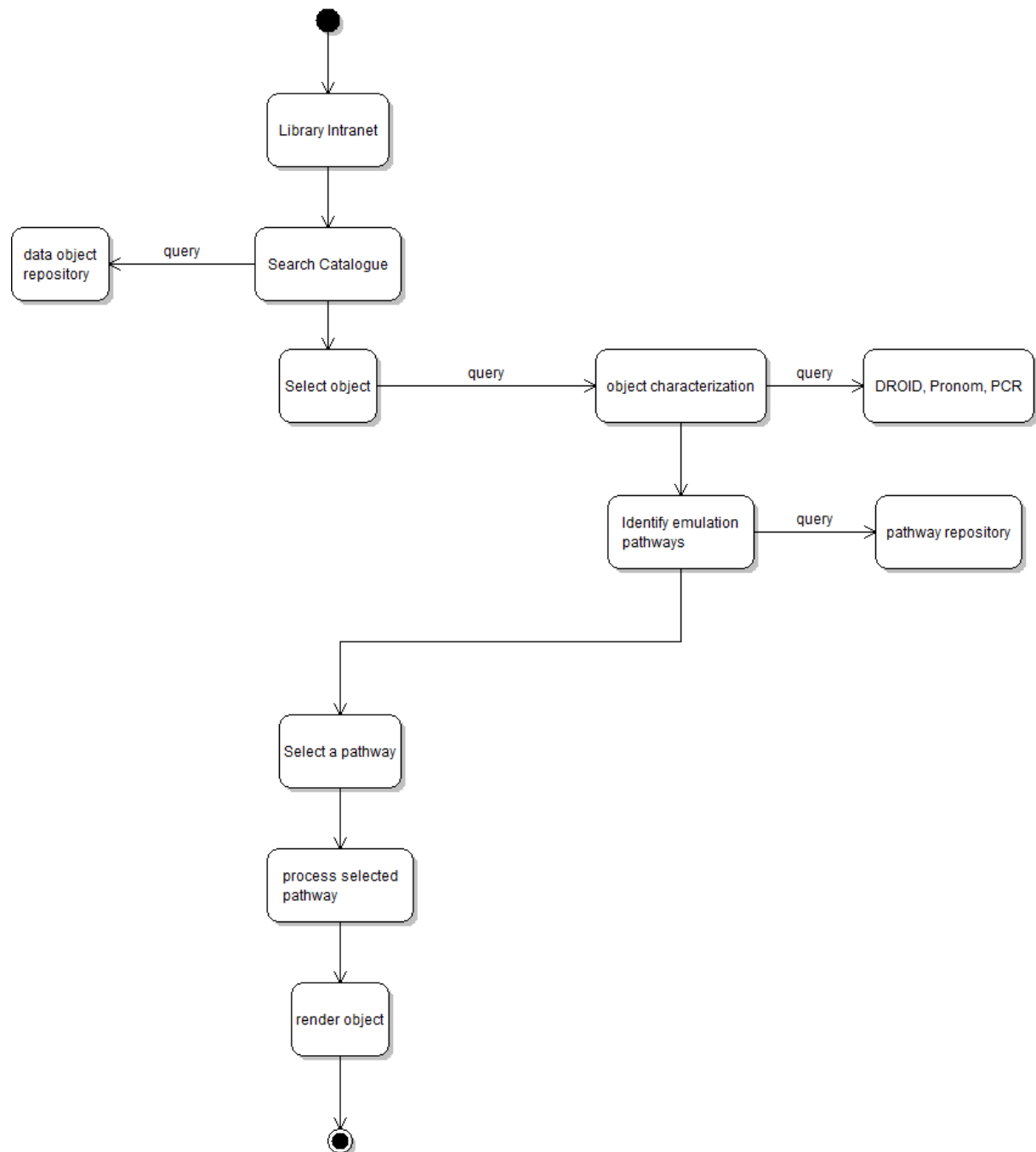
6.1.5 Portability

The EF will take care of running emulation processes correctly, not only now but also in the future. To do so, it is assumed that a virtual layer will be available that can be used by the EF to run emulators in a sustainable environment. Furthermore, this virtual layer will not draw restrictions on the functionality of the EF. If so, these restrictions should be explicitly defined to be able to measure the impact of it on the rest of the system.

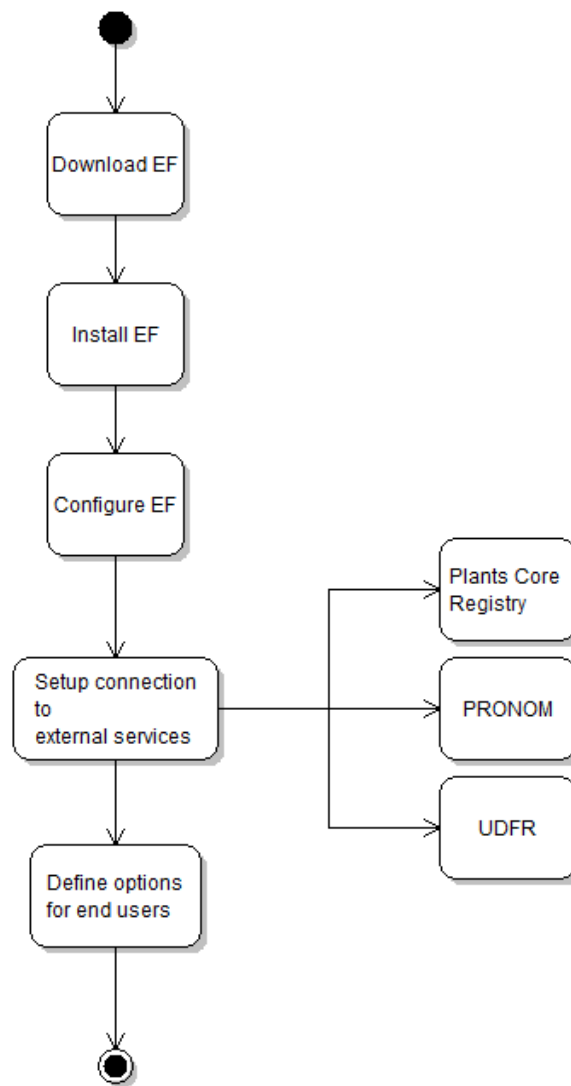
7 Use cases

Based on the user roles and scenarios described in previous sections, the workflow of different roles has been translated into use cases. The use cases are written using the Unified Modelling Language (UML).

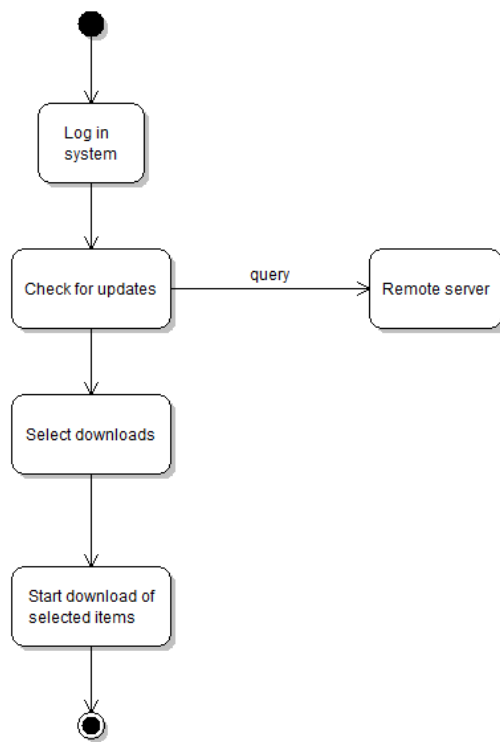
Use case 1: The end-user opens a document



Use case 2: The administrator sets up the EF



Use case 3: The administrator adds a new component



8 Functional requirements

The following two sections describe the requirements as they have been abstracted from the scenarios. The first section addresses all functional requirements; the second section non-functional requirements. All requirements have been categorised.

The categories have the following meaning:

- M Mandatory requirement. This feature must be built into the final product.
- D Desirable requirement. This feature should be built into the final product unless its cost is too high.
- O Optional requirement. This feature can be built into the final product at the Project Manager's discretion.
- E Possible future enhancement. This feature is recorded here so that the idea is not lost, but it will not be included in this project.

8.1.1 GUI

Requirement	Description	Priority	Scenario
F1.1	The end-user and admin should be able to control the EF via a single GUI	M	All
F1.2	The GUI should be able to run as an application	M	S3, S4
F1.3	The GUI should be able to run in a web browser	M	S1, S2
F1.4	The GUI should support common usability standards such as the IBM Common User Access	M	All
F1.5	The GUI should be able to show/hide several level of complexity for setting the emulation process options	M	S4
F1.6	The GUI should have a generic menu for common emulator functionality (e.g. pause, resume, screenshot, record video)	D	S5
F1.7	The GUI should display the supported emulation pathways	M	All
F1.8	The GUI should be able to display the unsupported emulation pathways	D	All
F1.9	The GUI should be able to display information (technical metadata) about the selected emulation pathways	M	S1, S2, S5
F1.10	The GUI should be able to display information (descriptive metadata) about the object to be rendered, if any	M	S1, S2, S5
F1.11	The GUI should offer the ability to contact the administrator	D	S5

8.1.2 External interfaces

Requirement	Description	Priority	Scenario
F2.1	The EF should offer an API for accepting digital object and metadata, preferably standardised.	M	S1, S2
F2.2	The EF API should acknowledge the receipt of the digital object and metadata	M	S1, S2
F2.3	Given an atomic digital object and metadata, the EF should be able to connect to one or more technical registries to generate a list of possible emulation pathways	M	S1, S4
F2.4	Given an compound digital object and metadata, the EF should be able to connect to one or more technical registries to generate a list of possible emulation pathways	D	S2
F2.5	EF should be able to access, search, and download content from external 'software package' repositories	E	S1, S2
F2.6	EF should be able to access, search and download external certified emulators	E	S3
F2.7	EF should be able to use the Planets Test bed to retrieve emulator performance information	D	S1, S2

8.1.3 Emulation process

Requirement	Description	Priority	Scenario
F3.1	The EF should be able to run at least two types of emulators	M	S1, S2, S4
F3.2	The EF should be able to run at least one Java-based emulator	M	S1, S2
F3.3	The EF should be able to run at least one Olonys VM-based emulator	M	S4

8.1.4 Emulation environment

Requirement	Description	Priority	Scenario
F4.1	EF should be able to prepare a 'software package' based on available pathway, digital object and any additional metadata	M	S2
F4.2	EF should be able to handle at least two types of atomic digital objects	M	S1, S4
F4.3	EF should be able to handle at least two types of compound digital objects	D	S2
F4.4	EF should keep a Preservation Fidelity factor for each emulation pathway	E	S4
F4.5	The Preservation Fidelity factor should be updated via feedback from end-users at the end of the emulation process	E	S4

F4.6	EF should be able to display the atomic digital object in the emulated environment	M	S1, S4
F4.7	EF should be able to display the compound digital object in the emulated environment	D	S2
F4.8	EF should be able to configure the selected emulator	M	S4

8.1.5 Usability

Requirement	Description	Priority	Scenario
F5.1	The end-user should be able to start and stop emulation processes	M	S4
F5.2	The end-user should be able to configure the emulation process before it starts	E	S4
F5.3	The end-user should not be required to configure the emulation process, i.e. the EF provides a default configuration	M	S1, S2
F5.4	The end-user should be able to record audio from the emulation process via the EF	D	S1, S5
F5.5	The end-user should be able to record video from the emulation process via the EF	D	S1, S5
F5.6	The end-user should be able to copy text from the emulation process via the EF	D	S1, S5
F5.7	The end-user should be able to record his/her keyboard input via the EF	E	S1, S5
F5.8	The end-user should be able to save/restore emulation sessions	E	S4
F5.9	The end-user should be able to view the emulated processes in "full screen" mode	O	S4
F5.10	EF should provide end-users with help/troubleshooting facilities	O	S1
F5.11	EF should provide an administrator account requiring authorisation	D	S3
F5.12	The administrator should be able to configure the connection to the technical registries	M	S3
F5.13	The administrator should be able to configure the amount of information displayed to the end-user	M	S1, S2, S4
F5.14	The end-user should be able to construct a custom pathway in the EF	E	S4
F5.15	The end-user should be able to save and load custom pathways	E	S4
F5.16	The EF can upload custom pathways to a central database	E	S4

9 Non-functional requirements

9.1.1 Accessibility

Requirement	Description	Priority	Scenario
N1.1	The GUI should be extensible	M	
N1.2	The EF should be platform independent	M	All
N1.3	The distribution should be done through a single package	M	S3
N1.4	The EF should be able to initialise itself within 60 seconds and should be able to load a 700 MB ISO image file within a further 120 seconds	M	All

9.1.2 Documentation

Requirement	Description	Priority	Scenario
N2.1	Guide containing information for end-users, including how to start, use and stop the system	M	S1, S2
N2.2	Guide containing information for administrators, including how to set up and configure the system	M	S3
N2.3	Guide containing information for developers, including how to maintain and write new software for the system, and API specifications	M	

9.1.3 Licensing

Requirement	Description	Priority	Scenario
N3.1	The EF will be open-source	D	
N3.2	The software will be covered by a suitable open-source license	M	

10 Design and Implementation Constraints and Standards

See deliverables D2.2b (ADD) and D5.1 (Integration) of KEEP.

11 Glossary

Atomic Digital Object

A digital object consisting of a single file

Compound Digital Object

A digital object consisting of a container format (such as ISO or ZIP), containing a collection of compound or atomic digital objects

Disk image

A disk image is a computer file containing a bit-level copy of the bits on a physical media carrier (e.g. floppy, CD-ROM, cassette, tape)

Emulation Framework

The system to be build, specified by these requirements, consisting of a front-end GUI and back-end core engine

Metadata

Data describing properties of the digital object

Pathway

The complete hardware and software stack needed to render a digital object. This usually consists of four layers (digital object, rendering application, OS, hardware platform), but may have more or less, e.g. a console game only has two: digital object and hardware platform

Software package

The software layers of the pathway, i.e. the rendering application and operating system

Preservation Fidelity factor

A user-generated indicator of how well a pathway supports the selected digital object



Keeping Emulation Environments Portable
FP7-ICT-231954

Requirements and design documents for services and architecture of emulation framework

Part B: Architectural Design for the Core Emulation Framework

Deliverable number	D2.2b (part of bundle of two documents: D2.2a, D2.2b)
Nature	Report
Dissemination level	PU
Delivery date	Due date : 30 October 2009 va.io.File)" updateRegistri
Status	Draft / Finalized / Reviewed / <u>Final</u>
Work package number	WP2
Lead beneficiary	TES
Author(s)	Bram Lohman (TES) David Michel (TES)



Document history

Revisions

Version	Date	Author	Changes
0.1	25-05-2009	Bram Lohman (TES)	1st version written.
0.2	08-06-2009	Bram Lohman (TES)	Expanded component descriptions. Incorporated comments from Software Architect.
0.3	11-08-2009	Bram Lohman (TES)	Expanded Characteriser and Controller sections
0.4	24-09-2009	Bram Lohman (TES)	Update design following URD meeting, 17-09-2009, KB
0.5	02-11-2009	Bram Lohman (TES)	Updated based on latest revision of URD
0.6	01-12-2009	Bram Lohman (TES)	Updated based on reviewer's comments (JOG)
0.7	6-12-2009	Bram Lohman (TES)	Updated based on reviewer's comments (KB)
1.1	22-03-2010	Bram Lohman (TES)	Updated design based on latest Scrum prototype (Sprint 4, 17-03-2010)
1.2	20-04-2010	Bram Lohman (TES)	Updated executive summary based on reviewer's comments (EU)

Reviews

Date	Name	Result
04-12-2009	Jeffrey van der Hoeven (KB)	Approved with minor changes
01-12-2009	Vincent Joguin (JOG)	Minor changes suggested

Signature/Approval

Approved by (signature)	Date

Accepted by at European Commission (signature)	Date



--	--

1 Executive Summary

This document describes the architectural design for the Core Emulation Framework, part of the Emulation Access Platform of the KEEP project. The architectural design outlines how the Core Emulation Framework should be developed.

The objectives and scope of the document are as follows: the design will follow the Scrum methodology, an iterative software development framework; major design decisions and motives will be documented when made; detailed component design will be updated when complete; the document covers only the Core component, and not the corresponding GUI.

The system context includes connections to external interfaces (both inputs and outputs) such as Technical Registries used for pathway generation, a Software Archives for software package retrieval, an Emulator Download Area for downloading emulator packages, Digital Archives/Transfer Tool Framework for supplying the digital object, and the Graphical User Interface to control the system.

The overall system functionality, to provide the user with the best available environment to render a digital object, is broken down into several steps, described, and shown in several workflow diagrams.

The design of the system components is described, starting with the Kernel component, which is the top level component and communicates with each of the remaining components: the Characteriser, which identifies digital objects and connects to external registries to retrieve the necessary information for running them in the EF; the Downloader, which connects to an external emulator database to download and update the emulator software which runs in the EF; and the Controller, which contains the logic to configure, run and output the results of each of the emulators running in the EF.

The data access layer, where the data is stored, and the database tables described.

The two prototypes mandated by the Description of Work and their delivery dates (month 19 and month 24, respectively) are detailed.

Required services of the external components are described, amongst which the external Emulator Download Area, the registries (such as PRONOM/Planets Core Registry and UDFR); the software archives; the machine and GUI API.

System-wide features such as the use of Java as the development language and unit tests as a testing / debugging strategy are covered.

The development environment, with the choice of Java as development language, H2 as internal database, the choice of Dioscuri (x86) and Vice (Commodore 64) emulators, and the emulator configuration strategy is fully described in section 9.

A glossary containing definitions of terms used in this document is included at the end of the document.

List of Related Documents

Description of Work [DoW]	Overall project description
User Requirements Document [URD]	Requirements for the Emulation Framework
Scrum Product Backlog [SPB]	Development tasks and prototypes

Abbreviations

Bibliothèque nationale de France	BnF
Joguin SAS	JOG
Koninklijke Bibliotheek	KB
Deutsche Nationalbibliothek	DNB
Tessella plc	TES
Person Month	PM
Work Package	WP
Description of Work	DoW
User Requirements Document	URD
Architectural Design Document	ADD
Emulation Framework	EF
Unified Modelling Language	UML

2 Table of Contents

1 Executive Summary	4
2 Table of Contents	6
3 Introduction	8
Objectives and scope	8
Outline of this document	8
4 System overview	9
Overview and Context	9
1.1 Inputs	11
1.2 Outputs	12
1.3 Major System Functionality	12
Architectural Summary	15
5 System Components	16
Kernel	16
1.1 Business Logic	16
1.2 Class Descriptions	16
1.3 External Interfaces	16
Characteriser	17
1.4 Business Logic	17
1.5 Class Descriptions	19
1.6 External Interfaces	19
Downloader	19
1.7 Business Logic	19
1.8 Class Descriptions	20
1.9 External Interfaces	20
Controller	20
1.10 Business Logic	20
1.11 Class Descriptions	20
1.12 External Interfaces	22
Data Access Layer	22
1.13 Emulator Database	22
6 Prototypes	24
7 External components	25
Emulator Download Area	25

Technical registries	25
1.1 PRONOM / Planets Core Registry	26
1.2 UDFR	27
Characterisation	27
1.3 FITS	27
Software archives	27
GUI	28
Machine API	32
8 System-Wide Features	33
1.1 Language	33
1.2 Error-Handling	33
1.3 Automated Debugging, Testing and Diagnostics	33
1.4 Speed and Capacity	33
1.5 Statutory and Regulatory	33
9 Development Environment	34
Development language	34
Internal database	34
Choice of emulators	34
Emulator configuration	35
10 Glossary	37

3 Introduction

This document describes the design of the Core Emulation Framework described in [DoW].

The design is based on the user requirements and the user scenarios specified in [URD].

This document forms the basis of the planning of the development phase. Since the development follows an iterative approach, the results from the various development iterations will be used to update the design to come to a complete architectural design. At the end of the development the document can be used as a developer's guide and as a basis for writing a system maintenance guide.

Objectives and scope

This design will follow the agile software development methodology, which is based on iterative development. The chosen methodology is Scrum, an iterative incremental framework. The final system will be the result of several iterations of prototypes, expanding and adding complexity during each iteration. Prototypes may implement a solution to a particular issue that seems viable at the time, but at a later stage may hinder further development. It is not unlikely that development will backtrack to resolve these issues and result in different prototypes and / or a different design strategy.

Any such major design decisions will be documented here for future reference.

Detailed design for many of the components of the prototypes has been delayed until development, as there are many unknowns in both requirements and external systems that need to be addressed. When this information becomes available the design will be updated, and subsequent prototypes will be based on this updated design.

As such, the architectural design is not expected to be complete until the end of the project.

Motives for design decisions made during the project's development phase will also be noted in this design to ensure the complete design process is documented.

The focus of this design is on the Core Emulation Framework; it does not cover the Graphical User Interface, which has been classed as a separate component of the Emulation Framework. The requirements of the GUI are also specified in [URD].

Outline of this document

This release of the design document focuses on a high-level design of the components. Several assumptions have been made, and will be labelled as such.

The overall structure of the document is as follows:

Section 4 contains an overview of the architecture.

Section 5 describes the major system components.

Section 6 lists the prototypes that will be produced in the development phase.

Section 7 has information on external components.

4 System overview

Overview and Context

The diagram of the Emulation Access Platform is provided below. This shows the proposed system architecture of the extraction tools and Transfer Tool Framework (1) which provides the digital objects used for input in the Emulation Framework; the Emulation Framework itself (2, 3), of which the Core Emulation Framework is the deliverable of this ADD; and the virtual machine layer (4), on top of which the Emulation Framework will run.

EMULATION ACCESS PLATFORM

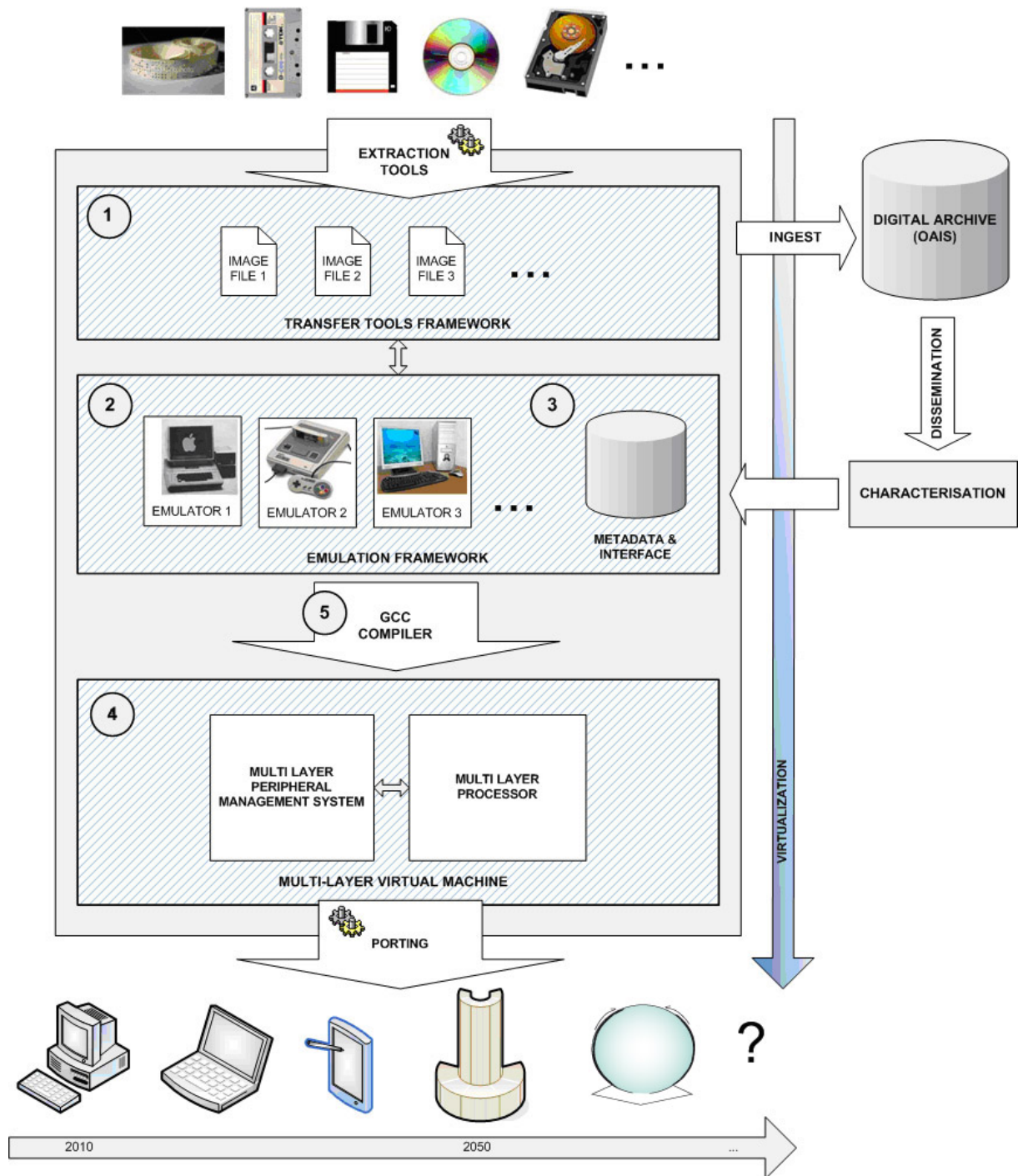


Figure 1: Emulation Access Platform

A simplified context model diagram of the Core Emulation Framework, showing the flow of data between the system and its environment across the system boundaries is as follows:

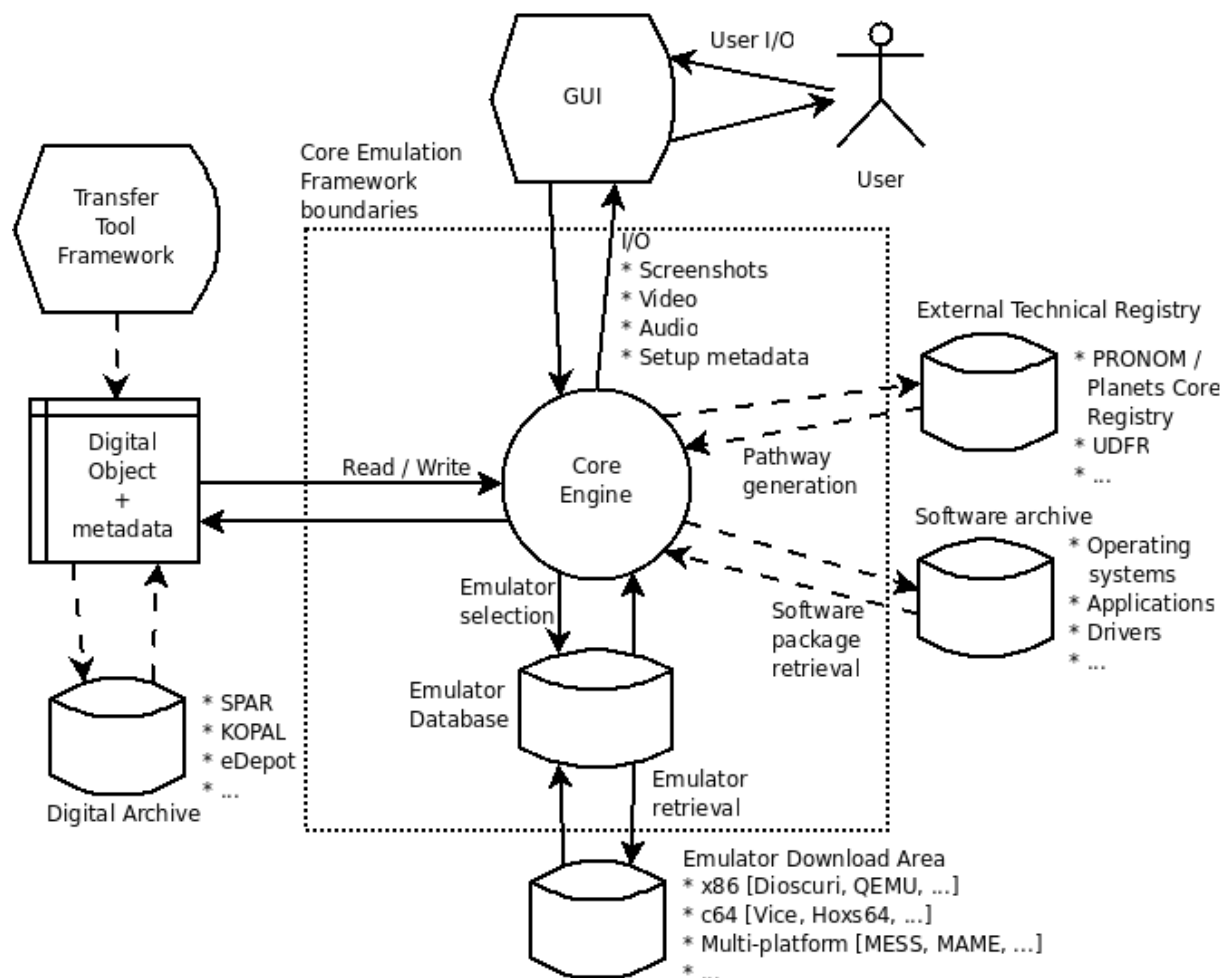


Figure 2: Emulation Framework system context diagram

1.1 Inputs

Figure 2 shows the external inputs of the Core Emulation Framework. Dashed arrows indicate optional inputs; these may not be necessary to complete the rendering of the digital object.

The human user will interact with the Core Emulation Framework via an externally developed GUI. This GUI will act as the intermediate for all input/output the user has for the system. For automated communication with environments, such as digital archiving systems, the Core EF will provide an API.

The starting point for the Core EF is the digital object. This digital object is in the form of a computer file, and can be classed as two types: as an 'atomic' file (e.g. PDF file or Word document) or a 'compound' file (e.g. a compressed collection such as a ZIP/RAR file, or a standardized image format such as an ISO 9660 image or cartridge image). The digital object may or may not include appropriate metadata describing it. This object is likely to come from a digital archive located at the memory institutions, such as DNB's Kopal, BnF's SPAR, or KB's e-Depot.

External technical registries provide the Core EF with a pathway describing the required hardware, software and application to render the digital objects. Examples of such registries are PRONOM¹ / Planets Core Registry or the UDFR².

The Core EF renders the selected digital object using emulators. These emulators are downloaded from the Emulator Download Area, an external database containing (certified) emulator packages. An emulator package contains executable files to run the emulator, and associated metadata describing the emulator's hardware, configuration, etc. The downloaded emulator packages are stored in a local database connected to the Core Engine.

The pathway for rendering a digital object consists not only of a description of a hardware emulator, but also a description of software dependencies including operating system, application and drivers. This software stack is retrieved from an external software archive, either as a complete stack or as separate components. The software archive is likely to be located at the memory institution; although shown as a separate system in the above figure for clarity, it may be part of a single system comprising the digital archive, Emulator Download Area and software archive.

1.2 Outputs

All inputs mentioned in section 1.1 also act as outputs. Again, dashed arrows indicate optional outputs.

Any user interaction with the Core EF will be shown as updates in the GUI. Also, any emulator outputs, such as audio and video, are also returned to the GUI. Other functionality such as capturing screenshots will also be presented to the user via the GUI.

The digital object may not only be read from but also written to. This allows the user to save any progress made.

The technical registries will need to be supplied with either the full digital object or a signature uniquely identifying it, for them to look up pathway information.

Prior to downloading emulator packages, one or more queries will be sent to the Emulator Download Area for a list of available emulators.

Similarly, the software archive will need to be queried for available content before software can be retrieved from it.

1.3 Major System Functionality

The functionality of the Core EF is to provide the user with the best available environment to render the digital object. Broken down into several steps, an administrator sets up the Core EF so the local Emulator Database contains one or more emulators used to provide the necessary environments, as shown in Figure 3

¹ PRONOM at The National Archives, <http://www.nationalarchives.gov.uk/pronom/>

² United Digital Formats Registry, <http://www.gdfr.info/udfr.html>

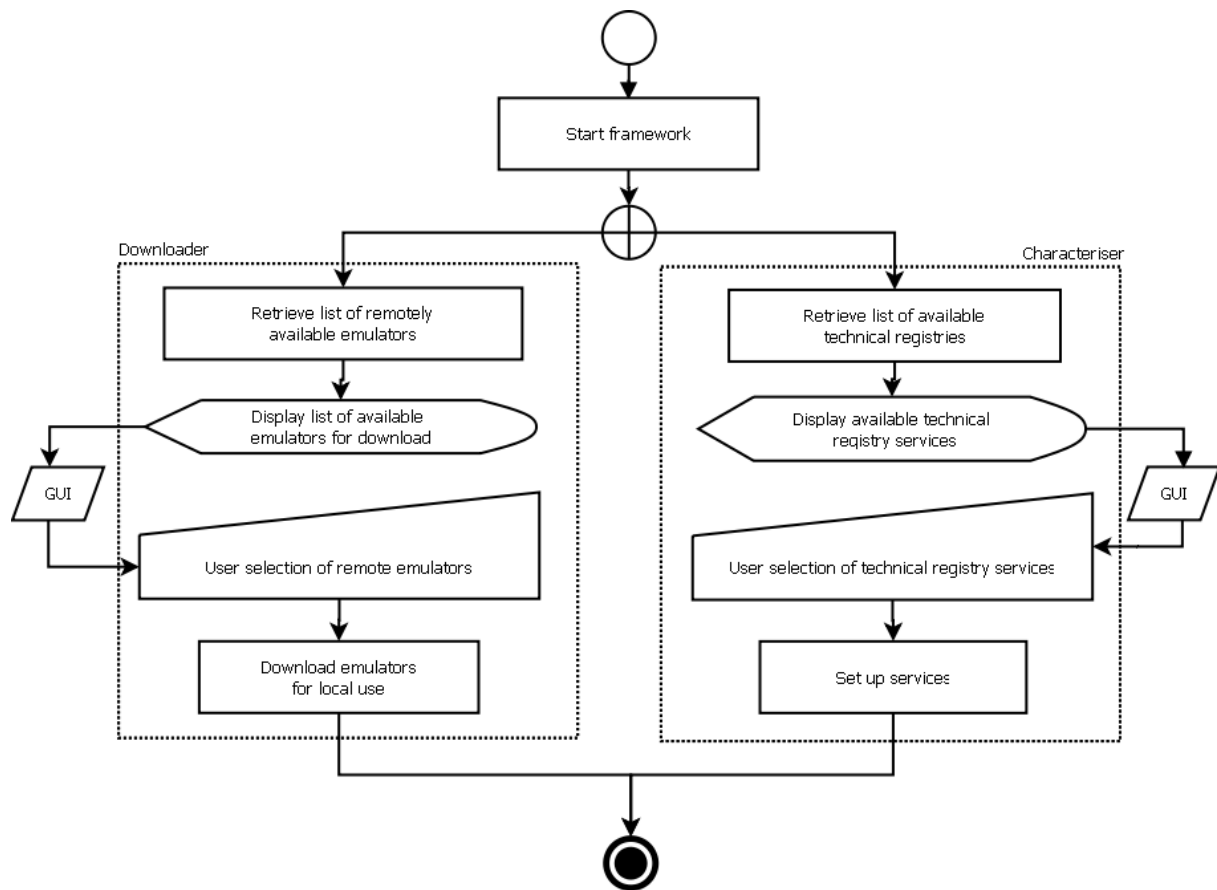


Figure 3: Core EF 'setup/configuration' workflow diagram

A user can then start the Core EF, providing it with a digital object, which may include identifying and/or characterising metadata. The Core EF may need to contact an external registry to obtain more information about the digital object; it will provide the registry with the required information identifying the digital object (which may vary between registries), and ask for the possible pathways. These pathways will be presented to the user for selection and/or configuration; not all pathways may be viable depending on the available emulators and software. Once the configuration and selection is complete, the pathway will be used to set up the environment for the digital object. The environment usually consists of the digital object, the rendering application and operating system plus any dependencies such as plug-ins or fonts, and the emulator.

From this point on the Core EF will act as an intermediary between the user and the underlying emulator, providing several functions independent of the emulator selection such as audio, screen and video capture.

Figure 4 shows the workflow for the above steps.

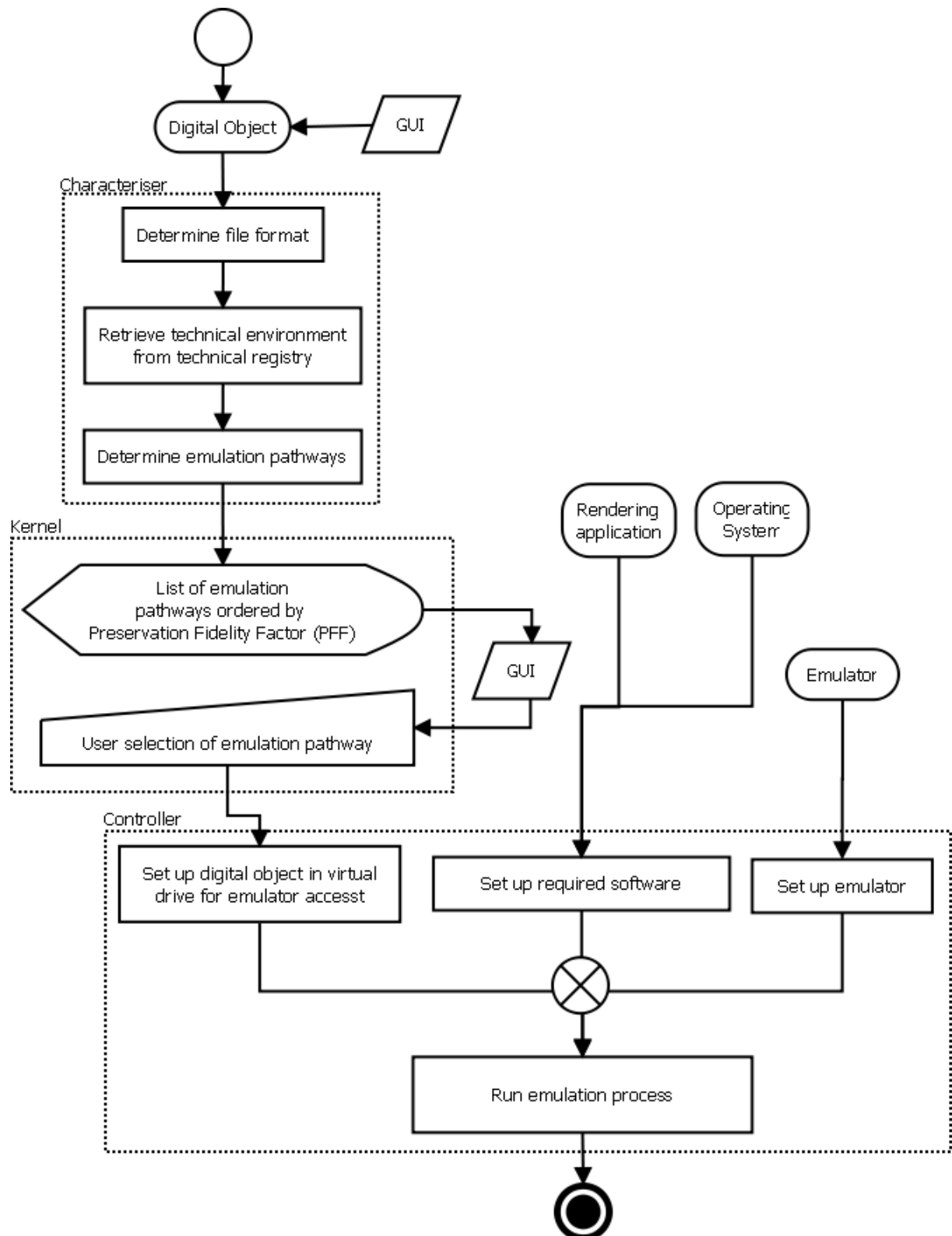


Figure 4: Core EF 'normal use' workflow diagram

The dotted boxes around the workflows indicate the components responsible for the logic of the workflow step.

Architectural Summary

The architecture of the main components in the system is shown in the UML Component diagram below.

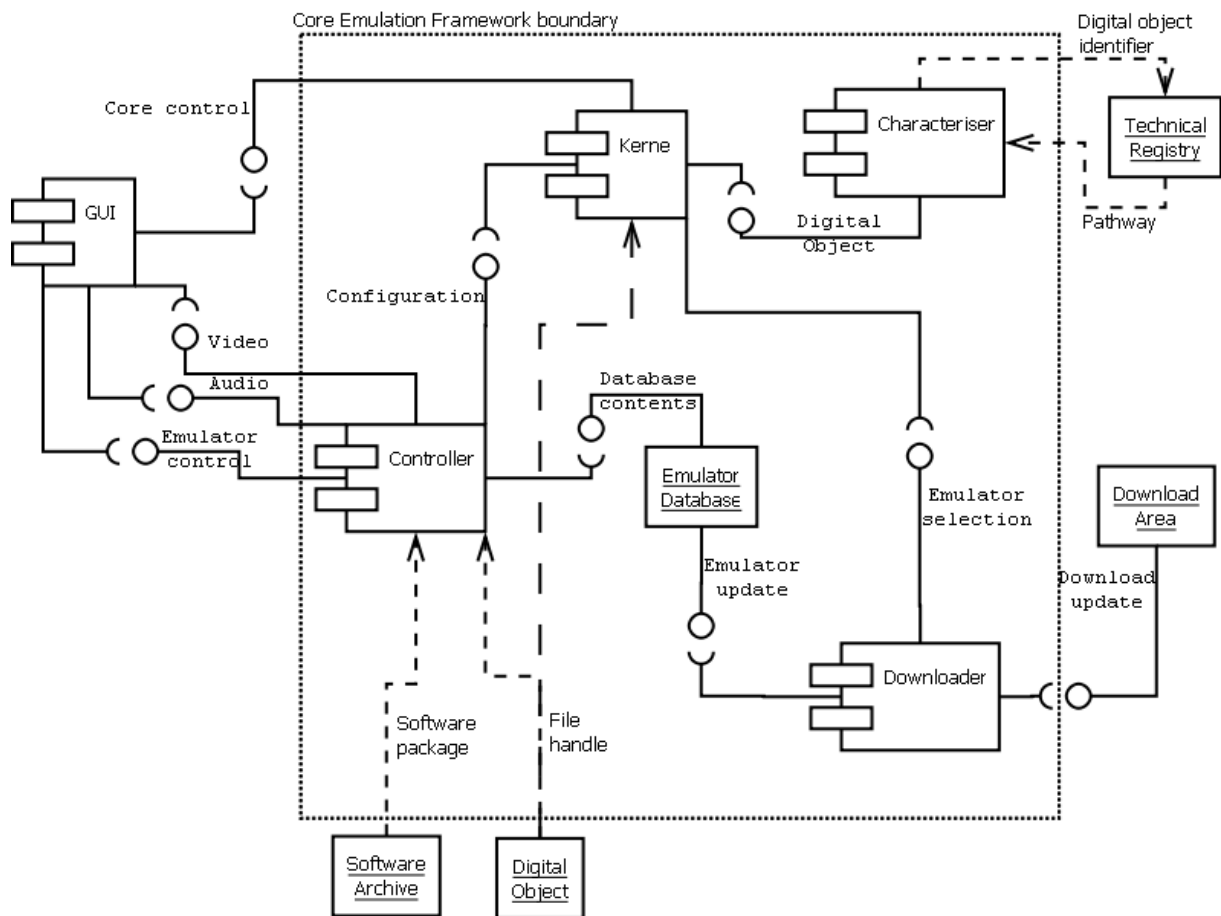


Figure 5: Core EF UML Component diagram

The socket connections show interfaces between components, while dashed arrows indicate dependencies of components on files or databases. Components outside the Core EF boundaries are not part of the system to be developed.

Communication between the Core EF and external components will occur according to the external component's interface. These interfaces are included in this document as a reference.

This diagram is a high-level overview of the system components, and as such may not include all connections between the different components. Detailed sub-level components and their functionality will be outlined in the following sections.

5 System Components

Kernel

1.1 Business Logic

The Kernel is the central component of the Core Emulation Framework. It ties together the internal components, and acts as the interface between the GUI and the Core EF, providing a path for the communication between the user's input and the component's outputs.

The Kernel component will implement the following functionality:

- Handle GUI input
- Delegate work to other components
- Implement the Machine API

The implementation of a central component with well-defined communication between internal components allows a modular structure to be built, and allows for easy replacement of internal components. This modular structure also simplifies adding new functionality over time.

As such, the Kernel is intended to translate high-level calls from the GUI into the required collection of calls required for underlying components to complete the request. For example, a GUI call for generation of pathways will result in the Kernel calling functionality to generate a digital object identifier, pass this to the Characteriser and return the result in the agreed form to the GUI.

1.2 Class Descriptions

The Kernel will act as a façade for functionality of the other components; as such it will contain classes for each of the internal components. These components are defined in the following sections, and so class descriptions of the Kernel can be based on the descriptions of these components.

1.3 External Interfaces

The Kernel component provides an interface for the communication between the GUI and the Core. As business logic is separated from input and presentation, the interface will follow the Model-View-Controller (MVC) architectural pattern. Examples of frameworks in Java that implement MVC are Java Swing³ for GUI frameworks and Spring⁴ or Struts⁵ as web-based frameworks.

The Core EF model provides at the time of writing the following functionality:

MVC

- Register and remove observers
- Initialisation

Emulator

³ <http://java.sun.com/javase/6/docs/technotes/guides/swing>

⁴ <http://www.springframework.org/>

⁵ <http://struts.apache.org/>

- Update the emulator list based on local packages
- Start an emulator based on a digital object
- Get and set emulator configuration options
- Clean up temporary files used by emulators

Technical Registries

- Update the registry list based on a configuration file
- Configure the registry list based on user input
- Characterise a digital object
- Retrieve a list of pathways for a file format

The full specification of the Model is included in the section External components: GUI

As the GUI and Core EF are written in the same language, the Core EF code can be simply integrated within the GUI code, i.e. using an external jar file so that the Core EF's public classes can be used.

Characteriser

1.4 Business Logic

The Characteriser provides functionality to connect to external technical registry databases for collecting metadata and pathways of user-selected digital objects.

The Characteriser provides the following functionality:

- Add/remove repositories
- Connect to external repositories and retrieve digital object metadata / pathways
- Convert external repository information from custom format into internal, common format

The Characteriser implements the APIs provided by external registries to retrieve digital object metadata. This will include functionality to convert the retrieved metadata into a generic internal format.

Currently the only available external registry is PRONOM, and the Characteriser uses the freely available FITS tool for characterisation of digital objects. These are described in section 7.

A flowchart of the functionality is depicted in the following figure:

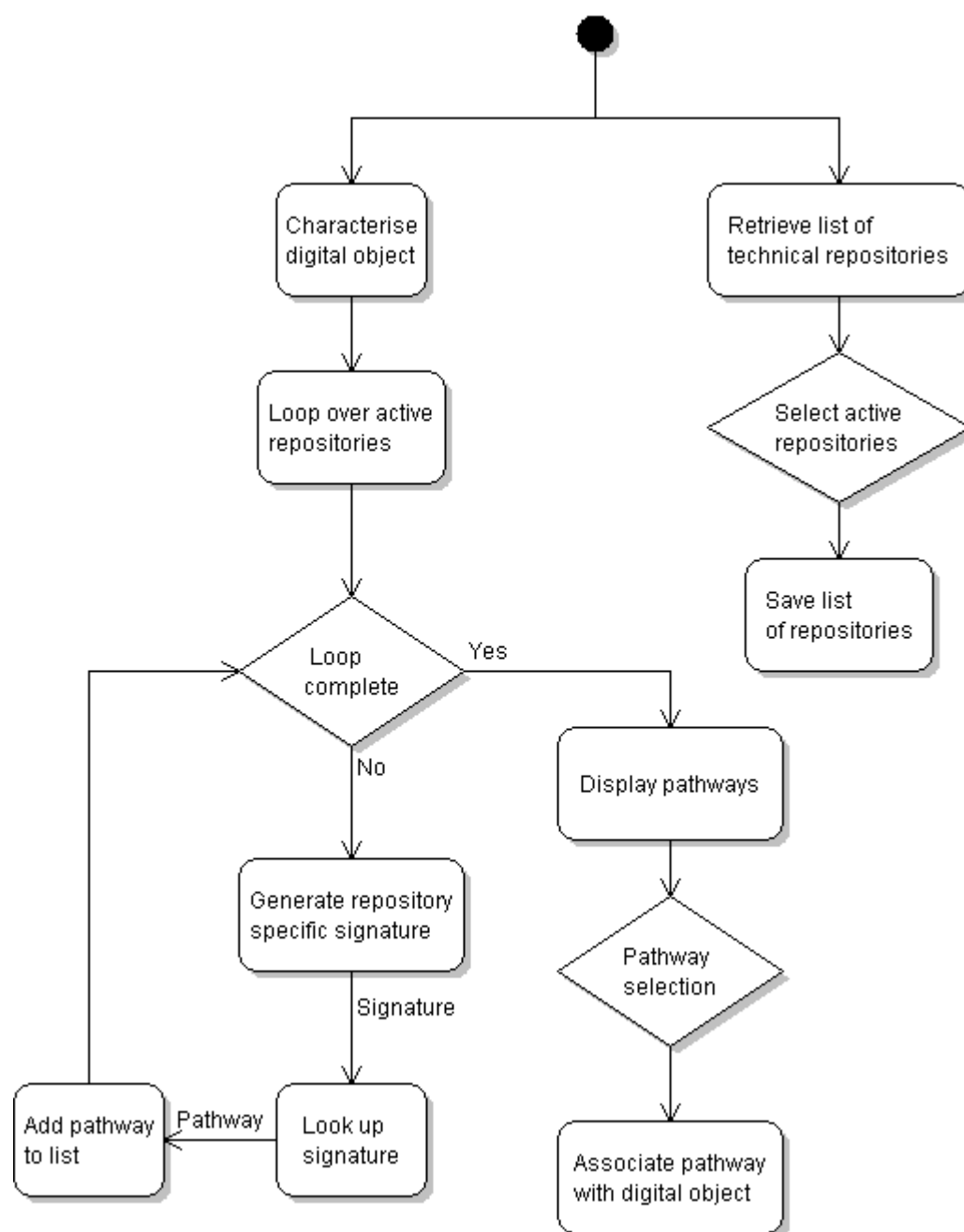


Figure 6: Characteriser flowchart

The characteriser has an internal catalogue of repositories containing the name, version, a short summary and URL of each repository. The user selects which repositories are queried for pathways for the digital object.

For each external repository, a class will implement the generic Characteriser repository interface, which contains common methods expected to be supported by the repositories (generate identifier, lookup identifier, etc.) This allows the system to loop over the implemented repositories and query them using generic methods.

When the system looks up a digital object using the specific implementations of the generic interface, the results are amalgamated. The repository-specific pathways are converted to an internal Core EF format, which is used by other Core EF components. During the conversion, the system will also try to remove any discrepancies between repository results.

The final selection is passed back to the Kernel for display to the user.

1.5 Class Descriptions

The Characteriser classes and their relations are shown in the figure below:

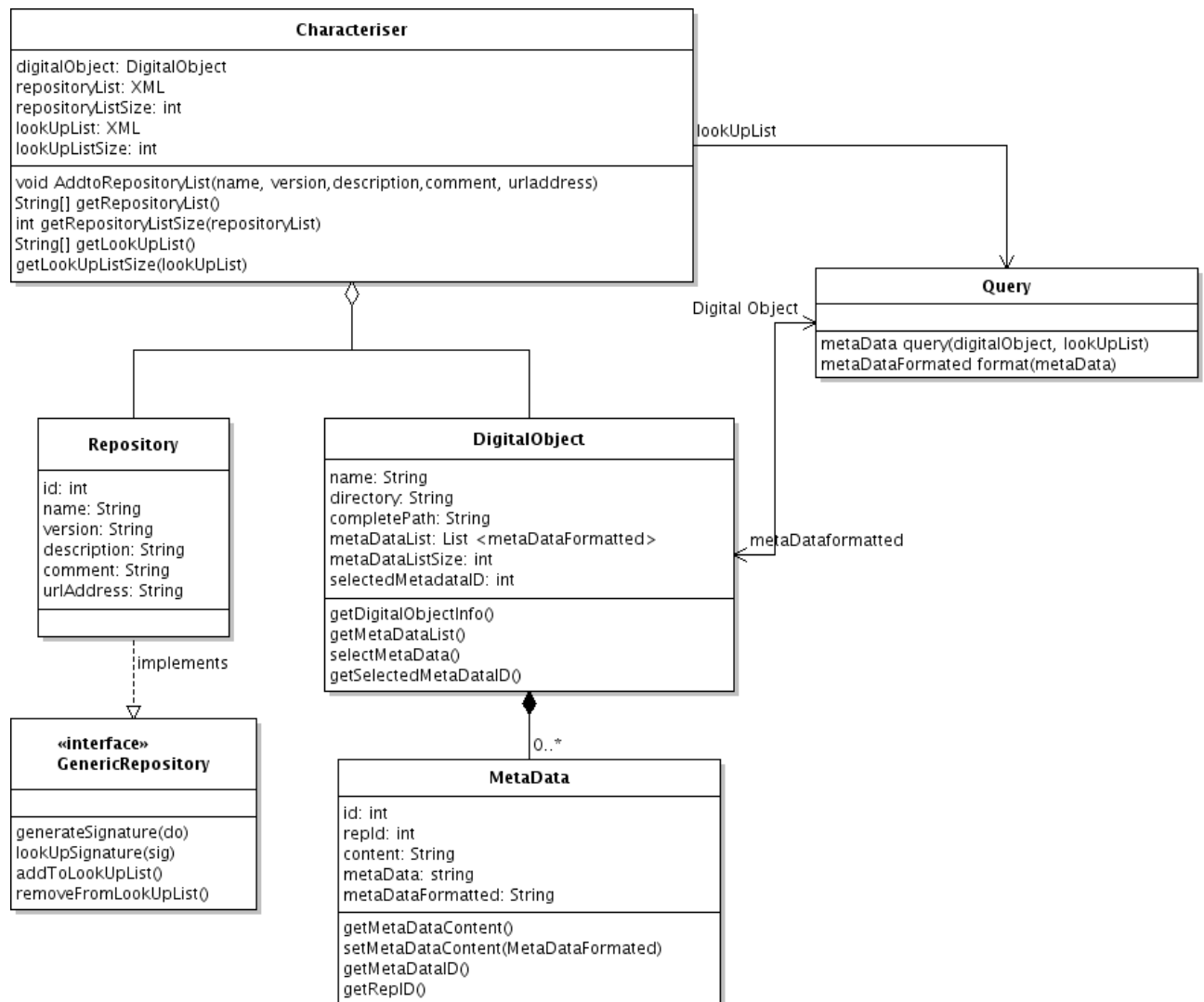


Figure 7: Characteriser class diagram

1.6 External Interfaces

See section 7 for APIs to external registries.

Downloader

1.7 Business Logic

The Downloader retrieves certified emulator packages (which contain specific metadata) from a central location and places these in the local Emulator Database. The emulators are then available for use within the Core EF; the metadata is used to determine which emulator can fulfil the pathway requirements.

The Downloader provides the following functionality:

- Contact Emulator Download Area for a list of emulators
- Download and store emulator packages in the local Emulator Database

- Remove downloaded emulator packages from the Emulator Database
- Update Emulator Database with emulator metadata
- Check for upgrades of existing emulators
- Implement emulator-specific configuration requirements

The Downloader implements the API provided by the Emulator Download Area to contact, retrieve, and download packages from the external database.

It also implements database logic to insert the retrieved packages and metadata into the local Emulator Database.

1.8 Class Descriptions

Classes in the Downloader will be based on the API of the Emulator Download Area. However, at the time of writing this API was not yet available, and so no class diagram or description could be made.

1.9 External Interfaces

An interface with the Emulator Download Area database is set up to retrieve emulator packages. An interface with the local Emulator Database is set up to insert emulators and metadata. The Emulator Database tables are described in section 1.13.

See section 7 for the API to the Emulator Download Area.

Controller

1.10 Business Logic

The Controller configures and runs the emulator, contained in the internal Emulator Database, which has been selected based on the pathway. It presents the emulator's video and audio output to the GUI and accepts GUI input for controlling the emulator.

The Controller component will implement the following functionality:

- Configure, start and run selected emulators
- Handle user input to emulator
- Present video and audio results of emulator to user
- Load user-selected digital object into emulator
- Record video and audio from emulator

The Controller runs the selected emulator executable based on the user pathway selection. It retrieves this emulator from the Emulator Database and sets it up with the correct configuration. Audio and video streams from the emulator are made accessible to the GUI, and can also be used to record from.

Input from the GUI is accepted and passed on to the running emulator.

1.11 Class Descriptions

The following diagram illustrates the classes that make up the Controller.

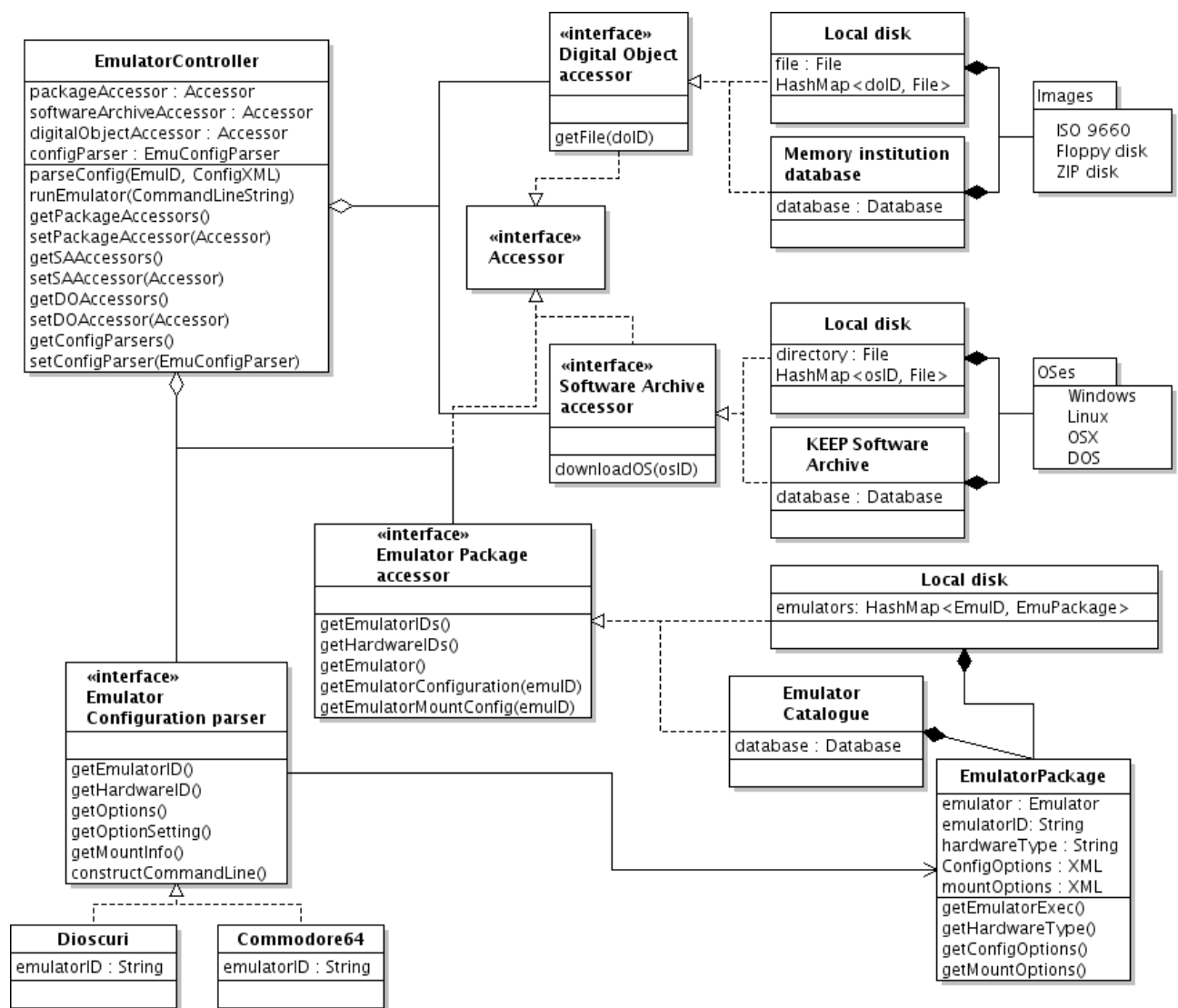


Figure 8: Controller class diagram

A short explanation of the classes is as follows: the controller receives an emulator configuration (e.g. XML). To parse this, each specific emulator will implement the Emulator Configuration parser interface; specific parsers are written because it is assumed the configuration options vary immensely between emulators.

The parser will know how to construct a command line string to start the emulator (using either command line arguments or by generating an option file). It also knows how to mount the operating system and digital object onto the emulator.

There are several 'accessors' that access the various (external) systems/objects: the digital object accessor for the file to mount; the software archive accessor for any required operating system; the package accessor for the emulator executables and metadata. Each of these accessors is envisioned to have several implementations: the 'local disk' implementation is envisioned for the user who simply does not require intranet/internet access and runs files (digital objects, operating system images) directly from their computer; for more complex interaction the database implementations provide access to intranet/internet databases that contain files to be accessed. Each memory institution will have a custom database implementation for access to their repository.

Currently the class diagram does not include video and audio recording options nor emulator input handling; this will be added in later prototypes.

1.12 External Interfaces

An interface with the Emulator Database is set up to retrieve emulator executables. The Emulator Database tables are described in section 1.13.

Data Access Layer

There are several places where the Core EF accesses data.

Locally there is a database containing downloaded emulator packages and their metadata, called the Emulator Database.

Externally, there is a list of databases/repositories that the Characteriser component accesses for retrieval of pathways and metadata. These databases, such as PRONOM/Planets Core Registry or UDFR, are expected to have an open API available that can be implemented in the Characteriser. These APIs are described in section 7.

There is also an external database of certified emulator packages, the Emulator Download Area. The Downloader will contact this to download emulators for use within the framework. This API is also described in section 7.

1.13 Emulator Database

This database contains data relevant to each emulator contained within the Emulator Database.

Emulators

Attribute	Data Type	Description
Emulator_ID	Large, unsigned integer	Unique emulator identifier from Emulator Download Area
Name	Variable length string (up to 255 characters)	Name of the emulator
Version	Large, unsigned integer	Emulator version
Exec_Type	Variable length string (up to 255 characters)	Type of executable (jar, exe, etc.)
Exec_Name	Variable length string (up to 255 characters)	Name of executable file
Exec_Dir	Variable length string (up to 255 characters)	Directory in which executable is located
Description	Variable length string (up to 255 characters)	Short description of the emulator
Package_name	Variable length string (up to 255 characters)	Name of the emulator package
Package_type	Variable length string (up to 255 characters)	Type of package compression (zip, rar, etc.)
Package	Binary	Binary package file

Hardware types

Attribute	Data Type	Description
Hardware_ID	Large, unsigned integer	Primary key

Name	Variable length string (up to 255 characters)	Hardware platform type
------	---	------------------------

Emus-hardware

Attribute	Data Type	Description
Emulator_ID	Large, unsigned integer	Emulators primary key
Hardware_ID	Large, unsigned integer	Hardware primary key

Registries

Attribute	Data Type	Description
Registry_ID	Large, unsigned integer	Emulators primary key
Name	Variable length string (up to 255 characters)	Name of the registry
URL	Variable length string (up to 255 characters)	URL of the registry
Classname	Variable length string (up to 255 characters)	Java .class file associated with registry
Enabled	Boolean	Registry is used during pathway lookup
Description	Variable length string (up to 255 characters)	Short description of the registry
Comment	Variable length string (up to 255 characters)	Short comment of the registry

6 Prototypes

The design of the Core EF is based around a sequence of prototypes, where each subsequent iteration includes more functionality. The final product should provide the functionality of the Core EF as described in [URD].

This approach was chosen to facilitate design, allow simple implementation to show where bottlenecks lie, and defer implementation until user requirements are properly elicited.

[DoW] mandates the release of two prototypes:

Sequence	Details	Delivery date
1	Prototype of working Core EF with set of emulator modules for hardware components of the selected target environments	M19 (August 2010)
2	Second prototype of Core EF with set of emulator modules for hardware components of the selected target environments	M24 (January 2011)

The Scrum Product Backlog [SPB] estimates a larger number of prototypes, as there will be around 10 four-week sprints, each resulting in a working prototype. It is expected a small number of these will be publicly released, of which two will be the [DoW] mandated prototype releases named above.

7 External components

Emulator Download Area

The Emulator Download Area is a database containing certified packages of emulators. The certification indicates it is known to run successfully in the Emulation Framework, and the package contains the required metadata described in the database tables in section 1.13.

At the time of writing, no API for the Emulator Download Area is available. However, any API that will be developed should support at least the following:

- a web-service that takes a blank input and returns a list of emulators, including their version and an identification number for each emulator.
- a web-service that takes a blank input and returns a list of supported hardware, including an identification number for each hardware type.
- a web-service that takes an identifier of a hardware type as an input and returns a list of emulators, including their version and an identification number, which supports the hardware type.
- a web-service that takes an identifier of an emulator as an input and returns a binary file containing the emulator and metadata.

Technical registries

The Emulation Framework may use several registries to gather pathway information if the metadata provided with the digital object provides insufficient. These registries should support at least the following:

Contents

- a web-service that takes a blank input and returns a list of file formats, including a version number and an identification number for each format.
- a web-service that takes a blank input and returns a list of software packages, including a version number and an identification number for each package.
- a web-service that takes a blank input and returns a list of operating systems, including a version number and an identification number for each OS.
- a web-service that takes a blank input and returns a list of hardware types, including a version number and an identification number for each type.
- a web-service that takes a blank input and returns a list of emulators, including a version number and an identification number for each emulator.

File formats

- a web-service that takes an identifier of a file format as an input and returns a list of software packages, including their version and an identification number, which support the file format.
- a web-service that takes an identifier of a file format as an input and returns a list of operating systems, including their version and an identification number, which support the file format.

- a web-service that takes an identifier of a file format as an input and returns a list of hardware types, including their version and an identification number, which support the file format.
- a web-service that takes an identifier of a file format as an input and returns a list of emulators, including their version and an identification number, which support the file format.
- a web-service that takes an identifier of a file format as an input and returns a list of pathways, consisting of some or all of the software packages, hardware types, and emulators, including their version and an identification number, which support the file format.

Software packages

- a web-service that takes an identifier of a software package as an input and returns a list of operating systems, including their version and an identification number, which support the software package.
- a web-service that takes an identifier of a software package as an input and returns a list of hardware types, including their version and an identification number, which support the software package.
- a web-service that takes an identifier of a software package as an input and returns a list of emulators, including their version and an identification number, which support the software package.
- a web-service that takes an identifier of a software package as an input and returns a list of pathways, consisting of some or all of the hardware types and emulators, including their version and an identification number, which support the software package.

Operating systems

- a web-service that takes an identifier of an operating system as an input and returns a list of hardware types, including their version and an identification number, which support the operating system.
- a web-service that takes an identifier of an operating system as an input and returns a list of emulators, including their version and an identification number, which support the operating system.
- a web-service that takes an identifier of an operating system as an input and returns a list of pathways, consisting of some or all of the hardware types and emulators, including their version and an identification number, which support the operating system.

Hardware types

- a web-service that takes an identifier of a hardware type as an input and returns a list of emulators, including their version and an identification number, which support the hardware type.

Whether reverse lookups of the pathways are also required (e.g. given an emulator identifier, return the hardware types it supports) will be determined at a later stage.

1.1 PRONOM / Planets Core Registry

PRONOM is an on-line information system about data file formats and their supporting software products. Originally developed to support the access to and long-term preservation of electronic records held by the National Archives, PRONOM has been made available as a resource for anyone requiring access to this type of information.

The PRONOM API is described in several WSDLs⁶, Web Services Description Language files, an XML based protocol for information exchange in decentralized and distributed environments; however, the required methods above are in the process of implementation. When the API becomes available, it will be included here as a reference.

DROID (Digital Record Object Identification) is a software tool developed by The National Archives to perform automated batch identification of file formats. It is one of a planned series of tools utilising PRONOM to provide specific digital preservation services. DROID uses internal (byte sequence) and external (file extension) signatures to identify and report the specific file format versions of digital files. These signatures are stored in an XML signature file, generated from information recorded in the PRONOM technical registry.

1.2 UDFR

As of this writing, UDFR is still in the conceptual stage, and as such does not have a public API available yet.

Characterisation

1.3 FITS

The Core EF uses the open-source File Information Tool Set (FITS)⁷, which identifies, validates, and extracts technical metadata for various file formats. It wraps several third-party open source tools.

FITS uses the following open source tools:

- Jhove⁸
- Exiftool⁹
- National Library of New Zealand Metadata Extractor¹⁰
- DROID¹¹
- FFIdent¹²
- File Utility¹³

FITS also supplies two original tools

- FileInfo
- XmlMetadata

⁶ <http://gforge.planets-project.eu/gf/project/pronom/scmsvn/?action=browse&view=rev&revision=5>

⁷ <http://code.google.com/p/fits/>

⁸ <http://hul.harvard.edu/jhove/>

⁹ <http://www.sno.phy.queensu.ca/~phil/exiftool/>

¹⁰ <http://meta-extractor.sourceforge.net/>

¹¹ <http://droid.sourceforge.net/>

¹² <http://schmidt.devlib.org/ffident/index.html>

¹³ <http://unixhelp.ed.ac.uk/CGI/man-cgi?file>

Software archives

At the time of writing, no public software archives were known to exist. However, memory institutions may have digital depots that provide this functionality. Such a software archive should support at least the following:

Contents

- a web-service that takes a blank input and returns a list of file formats, including a version number and an identification number for each file format.
- a web-service that takes a blank input and returns a list of software packages, including a version number and an identification number for each package.
- a web-service that takes a blank input and returns a list of operating systems, including a version number and an identification number for each OS.
- a web-service that takes an identifier of a software package as an input and returns a list of available plug-ins, including a version number and an identification number for each plug-in.
- a web-service that takes an identifier of an operating system as an input and returns a list of available plug-ins, including a version number and an identification number for each plug-in.

File formats

- a web-service that takes an identifier of a file format as an input and returns a binary file containing the complete software stack, consisting of some or all of the software package, software package plug-ins, operating system and metadata.

Software packages

- a web-service that takes an identifier of a software package as an input and returns a binary file containing the software package and metadata.
- a web-service that takes an identifier of a software package and a set of identifiers of software package plug-ins as inputs and returns a binary file containing the software package, including plug-ins and metadata.
- a web-service that takes an identifier of a software package and a set of identifiers of software package plug-ins as inputs and returns a binary file containing the complete software stack, consisting of some or all of the software package, software package plug-ins, operating system and metadata.

Operating systems

- a web-service that takes an identifier of an operating system as an input and returns a binary file containing the operating system and metadata.
- a web-service that takes an identifier of an operating system and a set of identifiers of operating system plug-ins as inputs and returns a binary file containing the operating system, including plug-ins and metadata.

GUI

The full Javadoc API for the Model provided by the Core EF is as follows:

Method Summary	
ConfigBuilderDirector	autoConfigureEmulator (BigInteger emuID)

BigInteger	autoSelectEmulator (List<BigInteger> emuIDs) Select automatically an emulator from the list of potential emulators
String	characteriseObj () Characterise the digital object
void	cleanUp () Clean up the temporary created files/folders
List<BigInteger>	getEmulatorIDs (Pathway pathway) Select emulators according to the pathway
List< EmuOption >	getEmuOptions (ConfigBuilderDirector configBuilderDirector) Get the list of all available emulator options
Pathway	getPathway (String format) Get the pathway for a given file format
Properties	getProperties () Get the properties file
boolean	init () Initialise the model.
void	registerObserver (CoreObserver coreObs) Registers an observer
void	removeObserver (CoreObserver coreObs) Removes an observer
void	runEmulator (ConfigBuilderDirector configBuilderDirector) Run the chosen emulator with the appropriate configuration
void	setDigitalObject (File file) Set the digital object
void	setEmuOptions (List< EmuOption > options, ConfigBuilderDirector configBuilderDirector) Set the list of emulator options
boolean	start (File file) Start a digital object in an automatically selected emulator
boolean	stop () Stop the Core Emulator Framework engine
HashMap<BigInteger, String>	updateEmus () Update the local database with the latest emulator versions from the Emulator Download Area
List<Registry>	updateRegistries (File file) Update the local database with the latest registry information from the registry catalogue xml.
void	updateRegistries (List<RegistryCatalogue.Registry> listReg) Update the local database with a list of Registries

Method Detail

getProperties

Properties **getProperties()**

Get the properties file

Returns:

Properties the properties java object

registerObserver

void **registerObserver**([CoreObserver](#) coreObs)

Registers an observer

Parameters:

coreObs - A Core Emulation Framework observer

removeObserver

void **removeObserver**([CoreObserver](#) coreObs)

Removes an observer

Parameters:

coreObs - A Core Emulation Framework observer

init

boolean **init()**

Initialise the model. Calls any methods required to set up and configure the model, such as loading of parameters, connecting to accessors, etc.

Returns:

True if initialisation successful, false otherwise

setDigitalObject

void **setDigitalObject**(File file)

Set the digital object

Parameters:

file - File representing the digital object to render via emulation

characteriseObj

String **characteriseObj()**

Characterise the digital object

Returns:

String The file format, e.g. 'jpeg' or 'xml'

getPathway

[Pathway](#) **getPathway**(String format)

Get the pathway for a given file format

Parameters:

format - String representing the file format

getEmulatorIDs

List<BigInteger> **getEmulatorIDs**([Pathway](#) pathway)

Select emulators according to the pathway

Parameters:

pathway -

Returns:

list of emulator IDs

autoSelectEmulator

BigInteger **autoSelectEmulator**(List<BigInteger> emuIDs)

Select automatically an emulator from the list of potential emulators

Parameters:

emuIDs - list of emulator IDs

Returns:

BigInteger ID of the emulator

autoConfigureEmulator

[ConfigBuilderDirector](#) **autoConfigureEmulator**(BigInteger emuID)

runEmulator

void **runEmulator**([ConfigBuilderDirector](#) configBuilderDirector)

Run the chosen emulator with the appropriate configuration

Parameters:

configBuilderDirector - configuration builder director

start

boolean **start**(File file)

Start a digital object in an automatically selected emulator

Parameters:

file - File the file to start

Returns:

True if engine started without error, false otherwise

stop

boolean **stop()**

Stop the Core Emulator Framework engine

Returns:

True if engine stopped without error, false otherwise

cleanUp

void **cleanUp()**

Clean up the temporary created files/folders

updateEmus

HashMap<BigInteger,String> **updateEmus()**

Update the local database with the latest emulator versions from the Emulator Download Area

Returns:

HashMap containing emulator ID and name/version, if any

updateRegistries

List<RegistryCatalogue.Registry> **updateRegistries**(File file)

Update the local database with the latest registry information from the registry catalogue xml.

Parameters:

file - File A XML file containing registries

Returns:

List of Registries

updateRegistries

void **updateRegistries**(List<RegistryCatalogue.Registry> listReg)

Update the local database with a list of Registries

Parameters:

listReg - List of Registry

getEmuOptions

List<[EmuOption](#)>

getEmuOptions([ConfigBuilderDirector](#) configBuilderDirector)

Get the list of all available emulator options

Parameters:

configBuilderDirector - configBuilderDirector

Returns:

List

setEmuOptions

```
void setEmuOptions(List<EmuOption> options,  
                  ConfigBuilderDirector configBuilderDirector)
```

Set the list of emulator options

Parameters:

options - List representing the list of emulator options

configBuilderDirector - configBuilderDirector

Machine API

As referenced in section 1.1, the Core EF should provide a machine API for automated communication. This API is still in development, but should contain at least the following:

- a (web) service that takes a binary file and an XML metadata file as inputs and returns an acknowledgment on successful acceptance.
- a (web) service that takes a binary set of files and an XML metadata file as inputs and returns an acknowledgment on successful acceptance.

8 System-Wide Features

1.1 Language

The external GUI will contain all of the language displayed to the user. However, error messages from the Core EF may be passed to the GUI, but these will consist of an identifier and a description, allowing translation in the GUI based on identifier if necessary. The descriptions of errors will be in English.

1.2 Error-Handling

Any errors will be passed back to the external GUI for further handling and possible display to the user. These error messages will consist of a unique identifier and an optional description.

1.3 Automated Debugging, Testing and Diagnostics

The major methods of classes will have their own automatic self-checking tests using the jUnit framework to enable frequent, extensive unit tests to be run throughout development. Each library will have a test harness user interface to enable it to be executed in isolation.

1.4 Speed and Capacity

As emulation can be very CPU-intensive, high-end hardware and optimisation of code can lead to better results. The Core EF is not concerned with these issues, however, as the emulators themselves are not part of the code to be developed and thus beyond the scope of this document.

Also, the Core EF is intended to run on a virtualised layer. Optimisations in this layer or during compilation of the Core EF for this layer will have far greater effect than optimising the EF code. Again, this virtual layer is beyond the scope of this document, so no special precautions will have to be made within the Core EF.

Connections to external systems will have time-out values associated with them to ensure the system returns within a specified time.

1.5 Statutory and Regulatory

The KEEP consortium is not aware of any Regulatory and Statutory requirements relating to this project. Legal studies regarding emulation have been carried out as part of the KEEP project and the design of the Core EF will adhere to these results. However, the Core EF is only a harness for running emulators and no specific investigation has been carried out to confirm the legality of it.

9 Development Environment

Development language

The choice of development language is limited to Java or C/C++ [DoW]. Both languages allow the use of very similar libraries and are both object-oriented. They both seem adequate for this project and the difficult parts of the project seem equally difficult in both languages: controlling video/audio output of separate processes (external executables) seems to be very difficult, if not impossible for any given language/technology.

It is expected that C/C++ is easier to compile with GCC, for running the Core EF on a virtualisation layer. However, portability to other operating systems such as Microsoft Windows or OS/X is currently not as easy as Java, although possible via MinGW/Cygwin. Depending on how close the Core EF will incorporate existing emulators, C++ might be a better choice for embedding emulators that are mostly written in C/C++.

Java is likely to be easier to use for connecting to external web services and databases. Portability is not an issue with Java, although GCC is not yet entirely compatible with Java's graphic libraries (Swing/AWT). If in any case the integration with the virtualisation layer fails, or GCC and/or Linux cannot be ported to the virtual machine layer, the code remains highly portable and can be run using the Java virtual machine.

As the aim of the KEEP Emulation Access Platform is to be portable and durable, and it is unknown what support the virtual machine layer will provide to either C/C++ or Java code, Java has been chosen as the development language.

Internal database

The choice for internal database to be used within Java can be one of many simple databases such as H2, HSQL, Derby, or Firebird. As complete emulator executables may be stored inside this database, it is important it has a good performance with large amounts of binary data, such as BLOBs.

A database abstraction layer will be developed that translates the development language calls to SQL; this should ensure that the code is independent of the choice of internal database, allowing for easy replacement if for any reason a particular database is preferred.

Choice of emulators

The requirements for the Core EF specify that it should be able to run at least two types of emulators, of which one is a Java-based emulator, and another is an Olonys-VM based emulator.

Two emulators were chosen to fulfill these requirements: Dioscuri¹⁴ and VICE¹⁵.

Dioscuri was chosen as the team possessed extensive knowledge of the emulator. As it was expected that changes to the emulator may be needed to fully integrate it with the Core EF, this knowledge could prove invaluable. Furthermore, Dioscuri is written in Java, fulfilling the requirement to support Java-based emulator. It emulates the x86 platform, currently the most widely-used computer platform.

¹⁴ <http://dioscuri.sourceforge.net/>

¹⁵ <http://www.viceteam.org/>

VICE was chosen to fulfill the remaining requirement to support a different type of emulator. VICE emulates the Commodore's 8-bit family of computers. From a usage perspective, the Commodore is widely supported in the emulation community, as well as being a platform for which very accurate emulators have been written. VICE is open source, so changes can be made to integrate it with the Core EF if necessary; furthermore it is written in C/C++, which makes it straightforward to (cross) compile, making it available for many platforms which may ease the porting to Olonys-VM.

Emulator configuration

Currently, to configure the emulators included in the Core EF, a separate emulator configuration is included in the (certified) emulator package for each emulator. This highly intensive task is not durable: it requires constant updates to the configuration when the emulator is modified. This method only works for configuration options already available in the emulators; although for open source emulators the source code can be modified, this will not work for closed source emulators.

To solve the problem of individual emulator configuration, a future solution would be to suggest that emulators included in the EF support a configuration API, i.e. each emulator implements a specified API that would allow the framework to call the methods in the API to control the emulator. This way, emulator developers have more control over the configuration, and the EF can rely on a generic API to control every emulator.

As configuration complexity differs between emulators, it is suggested that the configuration API is broken down into three levels of complexity: at the lowest, simplest level, only the very basic configuration parameters, such as mounting of disk and BIOS images, starting and stopping the emulator, are supported. Higher levels include all parameters of lower levels, and adding configuration of peripherals such as mouse/keyboard. At the highest complexity level, all the lower level parameters are supported, as well as video/chip timings, screen settings, etc. This hierarchy is displayed in Figure 9, which is a suggested configuration design using XML files and Java interfaces.

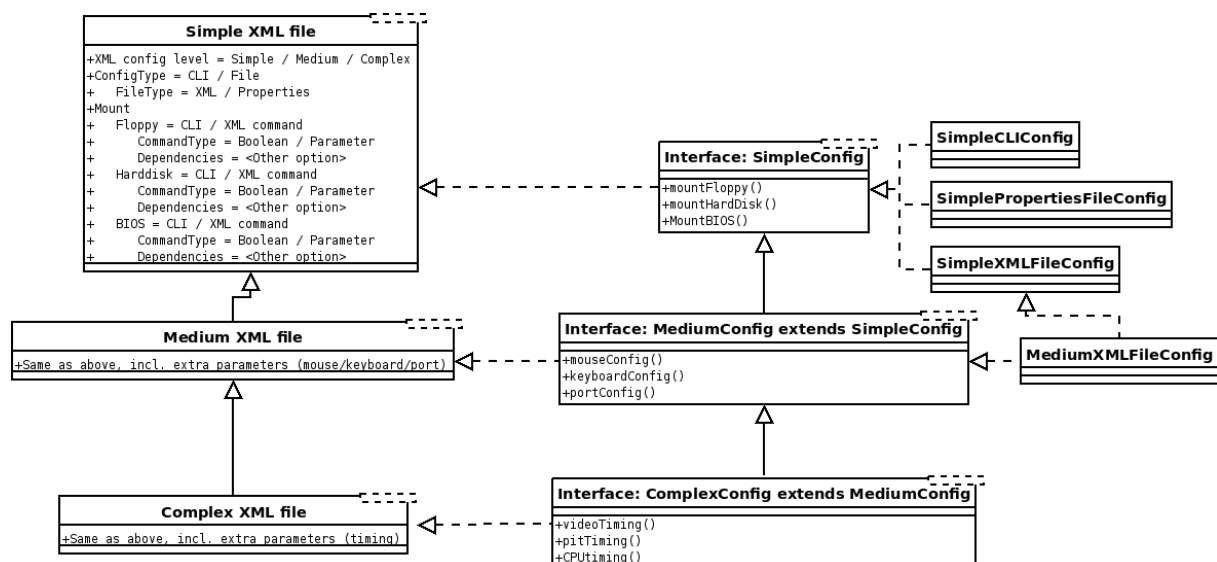


Figure 9: Emulator configuration design

This hierarchy of configuration would allow emulator developers to choose which level they can or want to implement, and so would allow the EF even for the simplest configuration to control the emulators.

As the EF and every emulator is expected to run on the Olonys VM in the future, the VM will act as an intermediary between the emulators and the EF, as the user will control the EF. The EF passes the required calls to the VM, which in turn will pass them on to the running emulator. In this respect, the API described above will be implemented between the VM and the EF as well as between the VM and the emulators.

10 Glossary

Architectural Design Document

The high-level design document for the entire system (this document)

Atomic Digital Object

A digital object consisting of a single file

Compound Digital Object

A digital object consisting of a container format (such as ISO or ZIP), containing a collection of compound or atomic digital objects

Core Emulation Framework

The part of the framework not including the GUI, nor the emulators that run in it, nor the transfer tools or the virtual layer on which it will run. These are all separate systems within the Emulation Access Platform. This document outlines the design of the Core Emulation Framework.

Digital Object

An object composed of a set of bit sequences, e.g. a single document such as a PDF file, or an image of a (console) game, etc.

Emulation Framework

A framework that offers emulation services for digital preservation. Its main functionality is to allow a user to load a digital object and select one of a number of pathways in the framework to render the digital object in its original environment. It consists of a front-end GUI and back-end Core EF. It is part of the Emulation Access Platform

Metadata

Data about other data; more specifically, data describing properties of the digital object

Pathway

A structured description of the complete hardware and software stack needed to render a digital object. This usually consists of four layers (digital object, rendering application, OS, hardware platform), but may have more or less, e.g. some console game only have two: digital object and hardware platform

The terms pathway and viewpath can be considered to be used interchangeably

Software package

The software layers of the pathway, i.e. the rendering application and operating system

Preservation Fidelity factor

A user-generated indicator of how well a pathway supports the selected digital object