

Simulating The Effects of Migration Policies on Digital Repositories

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Informatik

eingereicht von

Christian Weihs

Matrikelnummer 9825447

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Andreas Rauber

Wien, 20.11.2015

(Unterschrift Verfasserin)

(Unterschrift Betreuung)

Simulating The Effects of Migration Policies on Digital Repositories

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering and Internet Computing

by

Christian Weihs

Registration Number 9825447

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Andreas Rauber

Vienna, 20.11.2015

(Signature of Author)

(Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Christian Weihs
Fernkorngasse 23

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum)

(Unterschrift Verfasserin)

Abstract

Planning and operating a digital repository is a laborious and complex project. Especially the estimation of system resources (storage space and processing power) is difficult. The stability of file formats (limited by the support period of software vendors) enforces multiple migrations to newer file formats over time, which leads to an increased need for storage space and processing power depending on the migration tools used.

In this thesis a simulator is presented that helps to estimate the longterm development of digital repositories. It is possible to describe the content of the repository and the rules for the migration processes and simulate the scenario over an arbitrary time span. After the simulation statistics and diagrams can get generated for analysis and comparison of the result.

The goal is to provide a tool that can evaluate different scenarios containing rules and tools for migrations based on a given repository configuration. It should be possible to analyze how different rules and tools impact the resource requirements of the archive, in particular whether there are unexpected resource peaks during the lifetime of the repository.

The usage of the simulator is shown on a sample configuration build upon an existing repository. Therefore the structure of the repository is analyzed using the format registry PRONOM to build a configuration that matches the archive closely. With this sample configuration several migration scenarios are tested and evaluated using the generated diagrams.

Kurzfassung

Die Planung und der Betrieb eines digitalen Repositories ist ein aufwändiges und komplexes Vorhaben. Insbesondere die Abschätzung der langfristig benötigten Systemressourcen (Speicherkapazität und Rechenleistung) ist ein schwieriges Unterfangen. Die Haltbarkeit der verwendeten Dateiformate erzwingt über den Jahrzehnte langen Betrieb immer wieder Migrationen in neuere Dateiformate, was abhängig von den verwendeten Werkzeugen einerseits entsprechend Rechenleistung benötigt, andererseits steigt damit auch der Speicherverbrauch, weil nach der Migration die Datei sowohl in der alten als auch in der neuen Version vorhanden ist.

In dieser Arbeit wird ein Simulator vorgestellt, der dabei hilft diese langfristigen Entwicklungen abzuschätzen. Es ist damit möglich den Archivinhalt und die notwendigen Migrationsvorgänge zu beschreiben und darauf aufbauend eine Simulation über einen beliebig langen Zeitraum durchzuführen. Im Anschluss werden Statistiken und Diagramme erzeugt um die Ergebnisse zu analysieren und mit ähnlichen Simulationen vergleichen zu können.

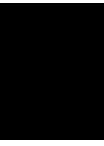
Das Ziel ist ein Werkzeug bereitzustellen, mit dem ausgehend von einem bestimmten Archivinhalt verschiedene Szenarien mit unterschiedlichen Migrationsregeln und den dazugehörigen Migrationstools evaluiert werden können. Dabei soll analysiert werden wie sich unterschiedliche Regeln und Tools auf den Ressourcenbedarf des Archivs auswirken, speziell ob zu bestimmten Zeitpunkten unerwartet hohe Zuwächse entstehen.

Die Verwendbarkeit wird anhand von Beispielsimulationen gezeigt. Dabei wird eine Konfiguration aufgebaut, die auf einem real existierenden Archiv basiert und mittels Zuhilfenahme der Formatregistry PRONOM erzeugt wird. Auf der Beispielkonfiguration wird die Verwendung von unterschiedlichen Regeln gezeigt und das Ergebnis der Simulation besprochen.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Problem Statement	2
1.3	Aim of the Work	2
1.4	Methodological Approach	3
1.5	Structure of the Work	3
2	State of the Art	5
2.1	Digital Preservation	5
2.2	Simulation	6
2.3	Preservation Planning	6
2.4	Format identification tools	7
2.5	Registries	7
2.6	Comparison and Summary of Existing Approaches	8
3	Implementation	9
3.1	General Description	9
3.2	Setup and Execution	14
3.3	Configuration	14
3.4	Gathering type information	21
3.5	Output	23
3.6	Generating Configurations	26
3.7	Performance	27
3.8	Summary	27
4	Tool Evaluation	31
4.1	General Issues	31
4.2	Doc2doc	32
4.3	Pdf2pdf	32
4.4	Jpeg2tiff	33
4.5	Summary	34
5	Case Study: EDW	35

5.1	General Issues	35
5.2	Sources of Sample Data	35
5.3	Sample Configuration	36
5.4	Experiments	37
5.5	Summary	41
6	Case Study: Danish Web Archive	43
6.1	General Issues	43
6.2	Basic simulation setup	44
6.3	Migration experiments	47
6.4	Summary	50
7	Summary and Future Work	51
7.1	Summary	51
7.2	Future Work	51
A	Appendix	53
A.1	EDW Ingests	53
	List of Figures	59
	Bibliography	61



Introduction

1.1 Motivation

We live in a digital world today. The amount of digital data grows with every day, as does the size of storage devices, which already store terabytes of data in a standard home computer. Only some decades ago the main media for storing information was paper. The big advantage of paper is that it is proven that it can carry the written information for hundreds of years. On the opposite digital storage devices are very transient. Today there is no known way for storing digital data for several decades on a single media. You always have to keep several copies, verify them and replace them from time to time.

The situation gets even more complex if you do not just want to preserve the information bit by bit, but keep the knowledge behind it accessible for future use. Historic paper documents are often difficult to read because over the centuries the languages changed and often you need the right cultural context to understand the text you are reading. The same applies to digital data, except that it does not take centuries but maybe only some years until there is no suitable software to open a certain document or no hardware to access the physical media.

So if you want to keep information accessible for a long period of time, you constantly have to monitor the file format it is stored in. Over the time the format can get unreadable for several reasons. The most common cause is that the software support is discontinued in favour for newer formats. This is specially a problem of proprietary binary formats, where you are bound to the software vendor. To circumvent problems with format readability it is necessary to migrate the files in time. Migration means to convert a file to another format using a conversion tool. The destination format can be a newer version of the same format family, for example a conversion from PDF 1.6 to PDF 1.7, or another format of another family, for example a conversion from Word Document to Open Document. The transition to another format family is usually done if another family has better preservation characteristics eg. longer release cycles or better format documentation. In some cases it might also be reasonable to migrate to several formats, for example a Word Document could be migrated to a newer Word version so that it is still editable, and in parallel it gets migrated to PDF for exactly preserving the layout.

For long term storage of data it is important to develop a good strategy, especially if the amount of data is big and ever-growing. You have to choose the moment migrations should take place, the destination format, the used tools and so on. This work should help to plan migrations and compare different policies. This is done by simulating a digital archive on which one can test different setups and compare the result.

1.2 Problem Statement

Setting up a digital repository is a very complex scenario. It is not enough to just physically store the elements and keep them save. Equally important is to keep the content accessible by doing migrations on a regular basis and schedule them for the future. This part is called preservation planning and is equally important to the physical preservation of the commissioned data.

Planning migrations for a digital repository is not a trivial task, as there are a lot of decisions that need to be made. At first the moment of time the migration should take place needs to get pinpointed. It can take place shortly before the software support for the file format ends, or as soon as a newer format is available, or even in the moment the file gets committed to the repository in case the used format is not suitable for long term storage.

The next decision is the destination format. Files can be stored in very different ways and not all ways are equally suitable for long time storage. Good file formats should be well documented and free from software patents. They also should be stable, so that new software will support them for a long time, and they should be robust to withstand a partial damage of the file in case of a hardware failure.

Another decision is which tool should be used to perform the migration. Usually there are several tools available for the task and they differ in accuracy, speed and the size of the produced result, so you have to choose carefully which tool should get used. Also it is important to decide whether to keep the original file after the migration or remove it to save storage.

All decisions have a huge impact on the repository, especially on the needs for storage sizes and processing power. This work should help to plan and compare different migration policies and show how a repository evolves when certain migration rules are applied. This is accomplished by simulating different scenarios and present the result in statistics and graphs so the user can easily compare the results of different simulations. At the end of the process there should be a clear view on what impact the different parameters have on the repository and the operator should be able to set up a good strategy for the probed archive.

It is important to mention that the accuracy of the used tools is not an issue here, as the migration processes are just simulated and not actually executed. Whether a tool is accurate enough or not has to be decided ahead, and only the impact that the tool has on needed storage and process time can get compared after the simulation. The same applies to the adequacy of existing file formats for digital preservation.

1.3 Aim of the Work

The aim of the work is to support maintainers of digital repositories in respect to preservation planning. This is done by providing a tool for comparing different migration policies and show-

ing the impact on digital archives. The tool (in the following called RepoSimulator) can be configured with different strategies in form of rule sets, which are then tested on a simulated archive. Of course the archive content itself can be configured in various ways to reflect the structure of the target system. After the simulation several statistics are created to show how well the set of rules performed and how the simulated archive will evolve in the tested scenario. If the statistics miss some special information the database itself can be used to gather more details. The archive metadata is stored in a standard SQL database in an easy to understand data model so it should be possible to collect as much details as needed. So in the end the maintainer of a repository should have a clear picture of how certain migration plans work with the data of his or her archive and be able to decide the best way to go.

1.4 Methodological Approach

The goal of this work is to get an insight on the changes a set of migration policies has on a digital archive. To predict the possible impact of migration strategies the configured archive gets simulated on a per file basis, which means that there is an entry in the simulation database for every file in the archive that represents the current state (size, filetype...), and if a rule applies for a certain file, the entry in the database gets modified to match the new situation. This can be performance expensive on large archives, but it simulates the archive very close to the real situation.

It was a conscious decision to do the simulation on a per file basis and not on groups of files, although this decreases the overall performance. The reason is that the real situation of a digital repository should get modeled as closely as possible.

The simulator is designed upon standard techniques to guarant that it can be easily adopted and developed further. It is written in Java using standard libraries wherever possible. The database is accessed using Hibernate, so although I tested only on a MySQL database, it should be no additional afford to use any SQL compatible database engine. The database model is kept as simple as possible to make it easy to analyze the data after the simulation run.

In the database every simulated file is represented as a database entry along with ingest date, file size and the used file type. The file type consists of a type family name and a subtype name specifying the current version of that family and the time frame in which this format is available. The usual approach on discontinuation of a subtype is to either migrate to a newer subtype in the same family or to switch to another family with better preservation characteristics (for example longer support time).

For presentation of the simulation results the chart engine JFreeChart was used to generate several diagrams and make the result visual accessible.

1.5 Structure of the Work

Chapter 2 describes the current state of the art and what similar approaches are currently available. It contains references about the difficulties of long term storage of data and preservation planning.

Chapter 3 shows how the RepoSimulator is built up and configured. The data model is described in detail and there are configuration examples. In the end the output of the simulator is documented with some sample diagrams and the way how they can get interpreted is describes.

In Chapter 4 the simulator is evaluated with several sample scenarios. It describes the way the sample data is gathered from an existing archive and how the simulator is configured. On the gathered real life data several different migration strategies are tested and the resulting statistics are compared.

Chapter 5 sums up the current state of the simulator and the usefulness of the results for preservation planning. It shows the constraints of the current implementation and ways the simulator can be extended in the future.

State of the Art

2.1 Digital Preservation

The preservation of information is a topic that mankind deals with since the beginning of its existence, yet it has drastically changed since we started to store information on digital media. In [11] it is explained that the main dangers for the accessibility of digital data is that it is very closely tied to the used technology. As long as the right technology exists it is no problem to access the data, but as technology evolves it may get inaccessible due to the lack of hard- and software support. [11] proposes three ways for facing this problem. One way is to preserve the technology, which means to keep old hardware running as long as there is information that need that designated hardware. This can be a very expensive task. The second way is to emulate the old system on newer hardware, and the third way is to migrate the information to a current format. This work focuses on the third way and tries to support the planning/scheduling of migrations by running simulations.

Preserving digital content is a large field of knowledge. The proceedings in [20] gives an overview about the personnel requirements organisations have to fulfill to be able to start preserving digital content.

There is an other article about the general topic on preserving digital data in [16] focusing on the threats digital material is exposed to. According to [16] the main threat digital content is endangered by is that either the storage media or the accessing software becomes obsolete. In the first case it is necessary to transfer the data to a new media, in the second case the data needs to be migrated to a newer file format as long as there is software available that supports the current format.

Although it is common sense that longterm storage of digital data is a very important topic, there is not much literature about simulating digital repository systems. Most papers about repository systems deal with theoretical lifetimes of the used hardware and how to physically keep the storage intact.

A very important part of digital preservation is choosing appropriate file formats for archiving. Different formats can bear very different risks for the data they carry. A significant aspect

of the usefulness of a file format is the tool support for that particular file format. Unfortunately that information is included only rudimentary in existing knowledge bases. The article in [?] tries to improve the current available information by clustering file formats to groups of similar formats in the hope of identifying additional software compatibilities, because usually tools support several similar formats for better information exchange.

2.2 Simulation

One of the few systems simulating aspects of a repository is ArchSim [5], which has a similar approach to the simulator presented in this work. Both try to simulate an archive instead of calculating the result, although it is theoretically possible to calculate it. In both cases the situation looks trivial in a small sample setup, but grows very complex when trying to set up a real life scenario and so it is easier to simulate the situation than doing theoretical calculations. Both simulators have a similar design, as they are based on an event driven algorithm.

Archsim is focusing on storage technology and the probability for the involved components to fail. Each component is modeled by a java object containing the failure probability of that component, and can therefore fire a failure event. The failure probabilities can follow different functions. The probability of not being able to interpret a format as it ages and becomes obsolete, for example, is modeled by a Weibull distribution. Different failure models can be assumed for different storage technologies. The failure of one component may lead to data loss, but it does not imply that. Eg. in case of a hard drive error the situation is recoverable as long as the data is stored on multiple drives, therefore a data loss can only happen if all storage devices fail before they can be replaced. Using a complex architecture of triggers allows efficient modelling of failure probabilities over long periods of time. The goal of the simulation is to determine the overall probability of the system that data loss happens. Therefore, the simulation has to be executed several times to see the distribution of the result.

ArchSim/C [6] explicitly models costs associated with operating archival storage, including costs for creating, operating, monitoring and repairing a complex storage system.

This way ArchSim and ArchSim/C give a good insight on how to set up a library system and where the most probable failure points are. The RepoSimulator on the opposite focuses on planning preservation policies, ie. when and how stored files have to get converted to other file formats. So it provides another part of the puzzle that is not captured by ArchSim.

An other related work that deals with modeling repository systems is presented in [4] and has a focus on analyzing the reliability of system configurations for digital preservation. Again, the focus is on understanding the effect of component failures within a storage system.

2.3 Preservation Planning

Testing and evaluating the effect of preservation action has been more intensively addressed from a planning perspective [1, 2]. Specifying the requirements for a specific preservation challenge (also referred to as objectives) and measuring how well different tools perform on selected sample data provide solid evidence for decisions on which preservation action component to deploy, and that component's effect on the object (in terms of significant properties retained),

the storage space required, as well the the complexity of the deployment with respect to system and human resources that need to be provided.

During the process of migration, quality assurance is an important aspect. The work in [10] tries to automate this progress specialized on digital images by rendering the raw image data and comparing the resulting images.

Preservation planning thus, on the one hand, provides valuable input to the simulator, concerning information on the processing requirements of certain types of preservation action tools as well as the resulting changes in object storage size. On the other hand, the repository simulator presented in this paper provides valuable input to a preservation planning process by providing a basis to estimate the costs associated with a certain preservation action, thus effectively closing the loop between planning and evaluation in these criteria.

2.4 Format identification tools

A very important topic for planning migrations on the one hand, and gathering sample data on the other hand, is file type detection. The exact file type is needed to interpret the contents of the file the right way and select the right tools to process the file. The detection of the right file format is a complex task, because there is no standard in how the structure of digital data is stored. Usually the file extension gives a rough insight on the file type, but in many cases the information there is not sufficient as usually a whole format family shares the same extension, and if the source data is created by inexperienced users it can happen that the extension is wrong. In [8] there is a methodology proposed for the process of extracting the file type from the content. It is based on interpretation of several file characteristics, especially the beginning and the end of the data, as most file formats tend to store some versioning information there. A practical example of this approach is the format registry PRONOM [3]. In PRONOM the patterns for many important file type versions is stored and a tool (called DROID [14]) is provided to extract the type information from files. This information can then be used in the RepoSimulator as a basis for simulations.

2.5 Registries

The file type is also needed to detect the lifetime/support time, which is important for planning a digital repository, and to set up a realistic initial configuration by using collection profiling services. Format registries such as PRONOM [3], which provide consolidated information on the lifetime/support time for selected formats, as well as offering a basis for analyzing the evolution of formats are very helpful for this task, yet the information there is far from complete. There are several other attempts to create registries for file information, but as the task is a rather big one, none of them can be considered even close to complete. A good insight on the current stage of development and the different existing approaches gives [13].

After determining the correct file type, the next step is to choose the best tool for migration. Usually there are several tools available to choose from, which differ in quality, performance and destination file type, so it is often challenging to find the right one. In [12] a testing framework, called the PLANETS testbed, is introduced that helps to compare the quality of different tools

by testing them on a sample library. The tools are encapsulated in a standardized interface, so it is easy to add new tools to the system.

For RepoSim two characteristics of a tool are needed: The way the size of a file changes during the conversion, and how long the process will take.

2.6 Comparison and Summary of Existing Approaches

The existing approaches to archive simulation are mainly focused on failure prediction. That means on the one hand to calculate the costs that accumulate due to hardware failures estimated from statistical data on average hardware lifetime, and on the other hand to predict the probability of data loss because of exceeded software support for the used file formats, which renders the stored data inaccessible.

While the studies in [5] focuses on understanding system failure characteristics and associated costs, the simulator presented in this work focuses on understanding the evolution of individual files across a series of migrations into multiple branches, and the associated requirements in terms of storage and computational resources. So the approach to the topic of maintaining a digital repository system in this work is from the pure software view and excludes the subject matter on hardware details except for predicting the physical storage size and processor power needed for long term maintenance of repository systems.

The main task of the simulation is to calculate the changes in the archive after a migration takes place. Migration means to transform a source file according to certain migration rules to one or more destination files. Each migration step is followed by a change of the repository size. Usually the needed amount of additional storage space increases on each migration, especially if the source file is kept or if the file is migrated to multiple destination formats, for example to one format that preserves the layout of a text document closely and one that keeps the file search- and editable.

To sum up the current state of the art we already have several important fractions that support the planning of a digital archive. We have ArchSim [5] to help us calculate the costs of an digital archive. We can use PRONOM [3] to gather information about the used file types, and the PLANETS testbed [12] gives us an insight on the quality of different migration tools. These parts give a good support for planning a repository, what is missing is a tool that helps to schedule the migrations and to predict the impact of different tools on the whole archive.

This gap should be closed with this work. By simulating different scenarios the user will get an insight on the processes in his repository and be able to judge which strategy will work best in his situation.

Implementation

3.1 General Description

The RepoSimulator is realized in Java. For the storage of the simulated archive a MySQL database is used, but because it is accessed using the Hibernate Database Library any compatible database implementation can be used. The simulation is based on an event driven model. There are three main events that are processed during the simulation run:

- **ingest events:** That is where new files get stored in the archive.
- **migration events:** During these events a file gets migrated according to the rules defined in the configuration. Migration means that the file gets converted to another file type version and this converted file gets ingested into the repository. The old version of the file still remains in the repository but gets marked as a migrated version.
- **deletion events:** A set of rules can be defined to remove old versions of files left behind by the migration events to free up storage space. Of course the files are not deleted from the database but only marked as *deleted*, as we need the information about the file for later statistics.

Analogical to this events, stored files can reach three different states during their life cycle. For all states the timestamp when the file reaches it is stored for later statistics.

- **ACTIVE:** *Active* is the state when a file got newly ingested to the repository and belongs to the working area. This are the files the migration rules can trigger on.
- **MIGRATED:** When a file gets migrated, the new version(s) of the file get ingested as *active* files, meanwhile the remaining version is marked as *migrated*, and therefore does not trigger any migration rules anymore. In case the file should get migrated multiple times (for example a Word Document that gets migrated to PDF on ingest, and to a new

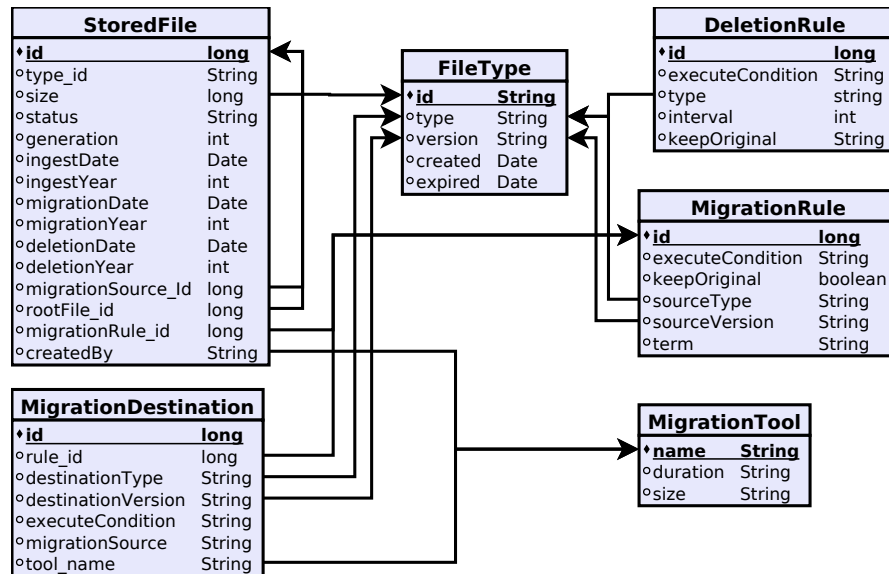


Figure 3.1: the structure of the database

Word version when the currently used version expires) then state of the file will not change to *migrated* until the last migration took place.

- **DELETED:** After the file is marked *deleted* the file is only ever accessed for statistical purposes, as it “officially” does not exist anymore.

At the end of the simulation statistical data gets collected from the database and rendered to some diagrams and csv files that can get processed further. If the generated statistics are insufficient the database can get analyzed for a better insight on the quality of the result.

Figure 3.1 shows the structure of the database. In the following the objects of the data model are described in detail:

- **FileType:** Describes the available file types. Every file type consists of a type family name (ie. “Word Document”) and a version name, which specifies the exact type version (ie. “Word 6.0”), as well as the average duration of validity and the periodicity with which new versions are released. It has the following attributes:
 - **id:** the id of the file type
 - **type:** the name of the type family

- **version:** the name of the type version
- **created:** the begin of the validity period
- **expired:** the end of the validity period
- **StoredFile:** Each ingested object is stored as a file stub (i.e. a file's profile) in the database. If a file gets migrated, a new file instance's profile is stored to the archive. Each file has the following attributes:
 - **id:** the id of the file
 - **type_id:** the file type version this file belongs to
 - **size:** the size of the file
 - **status:** the current status of the file
 - **generation:** a counter for how often this file was migrated. When a file gets added to the repository this field is set to 0 and gets increased by 1 on each migration.
 - **ingestDate:** the date the file was ingested
 - **ingestYear:** the year portion of the ingest date. This is a convenient attribute (as migrationYear and deletionYear) to keep database selects simple, because it is a common case to group certain files by year.
 - **migrationDate:** the date the file was migrated
 - **migrationYear:** the year portion of the migration date.
 - **deletionDate:** the date the file was deleted
 - **deletionYear:** the year portion of the deletion date.
 - **migrationRule:** the rule this file was migrated by
 - **migrationSource:** If the file was generated through migrating an other file, this attribute contains the id of the predecessor
 - **rootFile:** If a file gets migrated multiple times, this attribute stores the file as it was ingested to the repository.
 - **createdBy:** the tool which created this file. If this field is null it means that the file was ingested from outside and not created by any migration tool.
- **MigrationTool:** The main properties of a migration tool are the duration of the process and how the file size is altered during the migration.
 - **name:** the name of the tool
 - **duration:** how long the migration will take time (Can be calculated for example as a fraction of the file size.)
 - **size:** how the size of the file will change during the use of this tool

- **MigrationRule:** A migration rule defines when and how a migration should take place. This includes conditions which need to be met for the rule to be triggered (e.g a rule should only trigger for files smaller/larger than a certain size) and the scheduled moment of the migration
 - **id:** the id of the tool
 - **executeCondition:** the condition which must be fulfilled for the rule to trigger
 - **keepOriginal:** Defines whether the source file should be kept active so that further rules could trigger on them.
 - **sourceType:** the source type this rule should trigger for
 - **sourceVersion:** the version this rule should trigger for; can be set to “[all]” if it should trigger for every version of this type family.
 - **term:** the term when this rule should trigger
- **MigrationDestination:** Each rule can have a set of destinations to which the source file should get migrated to when the rule gets executed. Each destination has the following attributes:
 - **id:** the id of this destination
 - **rule_id:** the id of the rule to which this destination belongs to
 - **destinationType:** the type family to which the source file should get migrated to.
 - **destinationVersion:** the type version to which the source file should get migrated to. This can either be an exact version name, or “[minimal_step]” for a migration to the next available version, or “[maximal_step]” for a migration to the version with the longest expire time available.
 - **executeCondition:** the condition which must be fulfilled for this destination to trigger. This is for the case one wants to migrate to different file formats, but generate only certain destination files depending on e.g. the file size.
 - **migrationSource:** whether to use the CURRENT file version for the migration or to use the ROOT file for the migration, that is the file as it was ingested.
 - **tool_name:** the name of the tool used for the migration
- **DeletionRule:** A deletion rule defines when old versions of a file should be removed from the repository. This includes a condition which need to be met for the rule to be triggered (eg. a rule should only trigger for files smaller/larger than a certain size). The deletion rules are triggered once a year to do periodic cleanup.
 - **id:** the id of the rule
 - **executeCondition:** the condition which must be fulfilled for the rule to trigger
 - **sourceType:** the source type this rule should trigger for

- **keepOriginal:** Defines whether the root file should be kept even if the specified interval would include it.
- **interval:** Specifies which generations of a file should get deleted. Eg. `interval=3` would select every third generation (0, 3, 6 ...) for deletion.

To make the configuration flexible, many fields in the configuration are parsed with an expression language library, which means that any mathematical term can be used. Those fields are: the quantity and file size in the ingest configuration, the term and condition fields of the migration rule, and resulting change in file size and the computational cost of the migration process for each tool. Besides the usual mathematical operations there are several special keywords available that help adjust the expression to the aspired scenario. These keywords are:

- **currentsize:** the size of the currently processed file
- **generation:** the current generation number of the file, ie. how often this file was already migrated
- **version_expired:** the date the version of this file expires
- **version_created:** the date the version of this file was created
- **file_ingest:** the date the file was ingested
- **elapsedYears:** the number of years since beginning of the simulation
- **elapsedDays:** the number of days since beginning of the simulation
- **month:** the timespan of a month
- **year:** the timespan of a year
- **Dist:normal(mean,deviation):** calls a function for generating a normal distribution
- **Dist:weibull(alpha,beta):** calls a function for generating a Weibull distribution
- **Math:methodName(param):** With the prefix “Math:” every method of the Java class `java.lang.Math` is accessible providing a lot of mathematical functions which is useful in many cases.

These keywords help to make the configuration very flexible and dynamic. In the following section there are several examples how to use this feature in practice.

3.2 Setup and Execution

For using the simulator the following prerequisites are needed:

- A computer with a working java installation. The software is tested with JDK 1.7.0 but it should work with any current Java version.
- A database server. I used a MySQL server, but any database compatible to the hibernate framework should be usable.

Before starting with the first simulation, the database has to be configured in the file `hibernate.properties`. If you use a database other than MySQL you will also have to add the appropriate driver class to the classpath. The tables are created automatically, so you just need to provide an empty database.

For execution the following main classes are needed:

- `com.wec.reposimulator.RepoSimulator`: The simulator itself, which takes the directory with the configuration as parameter.
- `com.wec.reposimulator.gui.ConfigEditor`: The graphical configuration editor.
- `com.wec.reposimulator.StatisticsGenerator`: For generating the statistics.

If you rerun the simulation several times the database gets recreated every time, so you do not have to worry about cleaning up the previous runs.

3.3 Configuration

The configuration of a repository is done in plain text *ini*-files. There are basically four types of configuration files needed to specify all aspects of the simulation. For easier configuration there is a configuration editor shown in figure 3.2 which provides an easier access to the configuration files than just a text editor, although everything done in the configuration editor can easily be done by hand directly in the *ini*-files. In the following the core configuration possibilities are discussed showing sample listings of *ini*-files. The structure shown in the configuration editor is presented in the same way as in the *ini*-files, so everything discussed for the *ini*-files applies to the configuration editor the same way. The property names in the *ini*-files are case insensitive. To make them appear consistent they are converted to lowercase when saved through the configuration editor. It is important to mention that the shown configuration is intentionally kept as simple as possible because its only purpose is to show all configuration properties. See chapter 5 for more advanced samples.

Note that file sizes in the configuration have no special magnitude (bytes, kilobytes etc.) associated with them. They can be specified in any magnitude, as long as it is consistent in all configuration files. If not stated different I assume megabytes as the magnitude in the examples in this work.

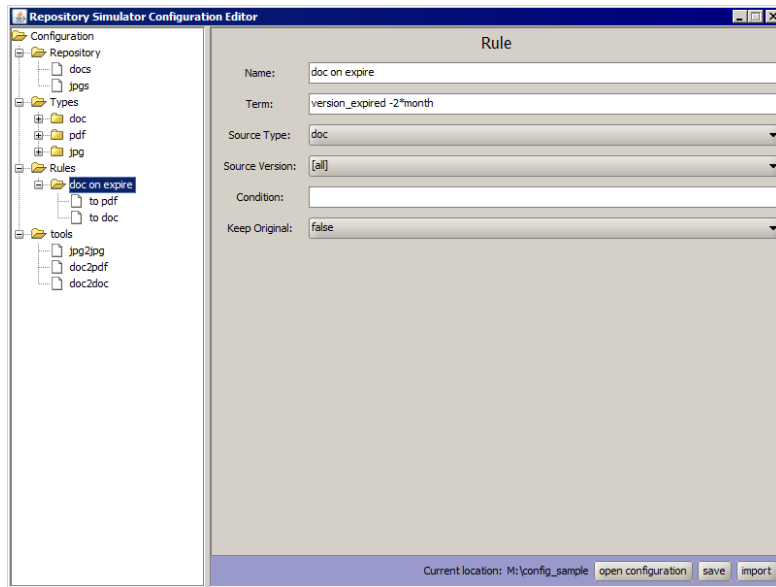


Figure 3.2: configuration editor with simple sample configuration

- **Repository:** In this file the start and the end of the simulation is configured, as well as (the sequence of) all ingest events. It basically describes the characteristics of the repository to be simulated.

The base configuration can be done by specifying groups of objects and their ingest characteristics by listing the number of objects, the mean and standard deviation in file size. This will create the according set of objects following eg. a Gaussian or Weibull distribution. Similarly, the timeline of the ingest process can be modelled, so not all files are ingested at the beginning of the simulation, but the repository can grow step by step via ingest of original objects (in addition to the migrated ones) during the simulation process. Furthermore, parameters allow the specification of the growth characteristics of an archive, both in terms of number of objects and the average file size. These can be specified via almost arbitrary complexity, ranging from simple linear growth to more complex functions fitting real-life growth curves. Additionally, in combination with the type family configuration described in more detail below, the ingested objects will be of a specific version of the given file type families according to the timestamp within the simulation progress. Starting the simulation then creates the respective “files” as simulated entities with the respective ingest timestamps in the database.

Listing 3.1 shows a sample repository configuration. In this case the simulation runs over 24 years starting in 2011/01/01 and ending 2035/01/01. Two types of files are ingested during the simulation. During the whole simulation files of the type *doc* are inserted with the quantity formula “ $100 + 100 * \text{elapsedYears}$ ”, which means 200 in the first year, 300 in the second year and so on. This ingest is repeated 25 times (successive_count = 25). In the simulation year 2020 ingests of type *jpg* starts with quantity of “ $10 * (\text{elapsedYears} + 20)$ ”.

These are inserted only every second year (`successive_intervall=2`) and repeated 8 times (`successive_count=8`). The file sizes are distributed following a normal (Gaussian) distribution. For both ingest definitions there is no type version specified, but with the keyword “[best]” it is defined that the best available version should be used, which will change over time during the simulation. Table 3.1 shows the number of ingests generated by this configuration.

Listing 3.1: Repository Configuration

```
[repository]
simulation_start = 2011/01/01
simulation_end = 2035/01/01
output_dir = output
name = 2011/01/01

[ingest1]
type = doc
quantity = 100 + 90 * elapsedYears
filesize = Dist:normal(10000,500)
ingest_date = 2011/01/01
successive_intervall = 1
successive_count = 25
name = docs
version = [best]

[ingest2]
type = jpg
quantity = 200 * (elapsedYears+20)
filesize = Dist:normal(30000,700)
ingest_date = 2020/01/01
successive_intervall = 2
successive_count = 8
name = jpgs
version = [best]
```

- **Filetype:** This describes a file type family. Each file type family consists of a family name and several type versions with a specified time frame of how long they are supported, as well as how frequently new subtypes are generated. This results in a set of available file types at each point during the simulation, with objects being ingested as new originals usually being created in the most recent version of a file format family available. Setting up realistic type information is a quite difficult task. See section 3.4 for more information on this problem set.

Listing 3.2 provides the configuration for the three filetypes *doc*, *pdf* and *jpg*. *doc* is valid for 3 years, every 2 years (`successive_intervall=2`) there is a new version, *pdf* is valid for

year	number of <i>doc</i> files	number of <i>jpg</i> files
2011	100	
2012	190	
2013	280	
2014	370	
2015	460	
2016	550	
2017	640	
2018	730	
2019	820	
2020	910	5800
2021	1000	
2022	1090	6200
2023	1180	
2024	1270	6600
2025	1360	
2026	1450	7000
2027	1540	
2028	1630	7400
2029	1720	
2030	1810	7800
2031	1900	
2032	1990	8200
2033	2080	
2034	2170	8600

Table 3.1: number of ingests created by listing 3.1

16 years and every 8 years a new version is available, and for *jpg* it is 11 years and a new version every 7 years. The attribute *successive_count* specifies how many following versions are generated. To give each generated version a unique name, a sequence number is appended to the specified version name. You can also specify several different versions for one format family, for example to simulate a container format like *avi* with different encodings inside. Table 3.2 shows the generated versions by this configuration.

Listing 3.2: file type configuration

```
[ filetype ]
name = doc

[ version1 ]
name = doc
created = 1999
```

```

expired = 2002
successive_intervall = 2
successive_count = 19

```

```

[ filetype ]
name = pdf

```

```

[ version1 ]
name = pdf
created = 1990
expired = 2006
successive_intervall = 7
successive_count = 7

```

```

[ filetype ]
name = jpg

```

```

[ version1 ]
name = jpg
created = 2000
expired = 2011
successive_intervall = 7
successive_count = 6

```

- **Migration rule:** The migration rule is the most important part of the configuration. Each rule has an effective date, a source type for which the rule should be triggered, and a condition (for example: *current file size is smaller than 5000*) that determines for which files the rule should be executed. This allows for a rather fine-grained specification of migration policies, eg. migrating smaller objects to multiple formats, whereas very large files might be migrated only within a single format family strand. Furthermore, migrations can either be based always on the most recent format version of each object, or always be based on the originally ingested object, ie. the root object, by setting the *source* parameter of the migration rule.

Beyond that one or more destination types along with the tools to be used for the simulated migration can be listed. For each destination type one can again specify a condition, so complex migration policies can be mapped into the simulation model.

Listing 3.3 provides two migration rules. The first one defines the migration of all files of the type family *doc* to the format family *pdf* and/or new versions of *doc*. The property *term* describes when the rule has to trigger, in this case two months before the respective sub-version of the file format expires. Two destination types are specified. The first one is a type of the family *pdf*. The subtype is not specified directly, but with the keyword *maximal step* it is indicated that we want to migrate to a type from that format family which is available at migration time and has the longest expiration time. The destination

subtype for doc is specified as *minimal step*, which means that the subtype with the next higher expiration date should be taken. This rule is also shown in figure 3.2 loaded into the configuration editor.

Both destinations have a condition specified. The migration to each destination is only executed if it evaluates to true. In this example files smaller than 15000 are migrated only to new *doc* versions and files larger than 10000 only to *pdf*. Files with a size between 10000 and 15000 are migrated to both destination formats. (This example is only supposed to demonstrate the flexibility of configurations, allowing to address space considerations that may appear in real preservation planning scenarios, when certain objects types that may be in demand by different user communities should be made available in different formats, whereas other, potentially very large files, should not be kept in duplicate versions. It is not supposed to represent a recommended preservation plan within the scope of this work. The same applies to the timing settings provided, ie. whether a migration should happen 2 months prior to the expiry date.)

The second rule in listing 3.3 takes care of the *jpg* type family. All *jpg* should get migrated to newer versions of *jpg* at term “ $version_expired - (version_expired - version_created) / 4$ ”, which means when the last quarter of the format version lifetime is reached.

Listing 3.3: migration rule configuration

```
[migrationrule]
description = migration for doc files
term = version_expired -2*month
source_type = doc
type = doc
name = doc on expire
condition =
keeporiginal = false
version = [all]

[destinationformat1]
destination_type = pdf
destination_version = [minimal_step]
condition = currentsize > 10000
tool = doc2pdf
source = current
name = to pdf

[destinationformat2]
destination_type = doc
destination_version = [maximal_step]
condition = currentsize < 15000
tool = doc2doc
source = root
```

```

name = to doc

[ migrationrule ]
name = jpgs on expire
type = jpg
term = version_expired - (version_expired - version_created) / 4
condition =
keeporiginal = false
version = [ all ]

[ destination0 ]
name = to jpg
destination_type = jpg
destination_version = [ minimal_step ]
tool = jpeg2jpg
condition =
source = current

```

- **Deletion rule:** The deletion rules specify when old generations of files should get deleted from the repository to save space. Deletion is a very important part of an archive as storage space is limited. Though storage devices grow bigger and bigger over time, in the same way does the amount of new data generated by people. So it is necessary to free space if possible.

The deletion rules are executed once a year to find matching files in the repository. Each rule can be restricted to a certain file type family. If no type is specified the rule applies to files of all types. With the property *interval* one can specify which generations should be deleted and must be a positive integer. If the generation number is the same or a multiple of the specified number the file gets deleted. You can also specify independently if the first generation should be kept or not (*keepOriginal*). Note that the newest (*active*) generation is never touched by the deletion process, only files with the *migrated* state are affected by these rules.

Listing 3.4 shows a deletion rule for the type family *jpg*. The *interval* is specified as “1”, so every generation is affected. A *condition* is set to “(*current_date* - *file_ingest*) > 5 * *year*”, which means that the file ingest date must be older than 5 years. The property *keepOriginal* is set to *false*, so the first generation will also get deleted.

Listing 3.4: deletion rule configuration

```

[ deletionrule ]
name = delete
type = jpg
interval = 1
condition = (current_date - file_ingest) > 5 * year
keeporiginal = false

```


- **Migration tool:** For each virtual migration tool one can specify how the size of the file is changed during the migration process and how long the migration will take. Both the file size and migration duration are specified using mathematical expressions. Thus, one is not bound to simple linear changes but more complex effects can be simulated. Both units are dimensionless, ie. as for the specification of the file sizes, these can be given in Bytes, Kilobytes, etc. More importantly, for the simulation of computational resources, either computation time or eg. processor cycles may be specified. The latter may prove useful when more realistic estimates of the computational requirements are required. By mapping operation cycles in a virtual unit, normalization factors may be applied to account for improvements in processing power over time. Still, in first experiments, specifying actual processing time, and then applying a normalization factor to account for improved computational facilities, seemed to be more easily accepted. Note, that the primary use of the effort simulation is not a precise determination of the HW requirements at a specific point in time in the future, but to capture the potential of cumulative effects resulting from certain preservation policies. These may stem, for example, from the difference of migrating on-ingest (usually leading to a more even spread of subsequent migrations) or on-expiry - resulting in strong peaks if all objects of a specific format version need to be migrated to, e.g. the subsequent version.

Two examples:

- **size=currentsize * Math.log(currentsize):** This specifies a logarithmic growth of the file. (The keyword “Math” in this string references the Java class `java.lang.Math`, which has methods for many mathematical operations, all accessible through this keyword.)
- **duration=Math.max(currentsize * 3, 18000):** The migration is estimated to take three times as long as the file is big, but at least 18000ms.

From this set of configurations the simulation of a repository’s evolution is started. For each (set of) files specified in the repository configuration, the according sets of files are “created” as database entries with the respective timestamps. For each of these the respective migrations based on the preservation planning triggers as specified in the migration rules setting are executed consecutively. Thus, for each file specification in the database meeting a migration condition the respective new file(s) are generated with new ingest timestamps and file sizes considering the migration time needed and the file size change incurred as specified in the respective migration tool specification. From these, the resulting hypothetical computational load (i.e. the number of files to be migrated at any specific point in time) and the required storage space for the accumulated archive can be calculated.

3.4 Gathering type information

The most important information for a reasonable simulation is about the used file types. Unfortunately that is the most difficult task. For setting up a good simulation configuration one

need to know the exact type version of each file in the repository, the family to which the format versions belongs, and the validity period of each version.

Determining the format version of a file is not trivial. The first place a user looks for this information is the file extension, but that gives only a very rough estimation. The first problem is that you cannot trust the extension. In most cases the extension is correct, but again and again there are cases where files have a wrong extension. That might happen by mistake when the creator stores the file, it can happen intentionally to bypass extension restraints ie. when mailing/uploading the file or it can be the result of error-prone software. Even if the extension is correct its expressiveness is rather weak. Usually all versions of one file type have the same extension, so you cannot derive the version from this attribute, but only the type family, but there are cases where versions belonging to the same family have different extensions, so not even the family is derivable.

So the only way of determining the version of a file is to look into it. The problem is that there is no standardized way to store the type information inside a file. So for detecting the file type of a file one needs a registry that stores the characteristics of as much file types as possible, and a tool that extracts this information from the file. The largest registry available at the moment is the registry of the National Archives of the United Kingdom and is called PRONOM [15] [3]. The PRONOM database currently contains over 800 entries and should cover most of the common file types. The entries are identified by the “PRONOM Persistent Unique Identifier” (PUID), which provides a good tagging for the file type versions.

For extracting the information from a file the National Archives of the United Kingdom provides a tool called DROID [14], which uses the PRONOM registry and delivers the PUID of a file.

Beside the exact file type version the date when the file was ingested to the repository and the size is needed. For this task I used the combination of the tools FITS [9] and C3PO [17]. FITS stands for “File Information Tool Set” and is a meta data collection utility that uses several third party tools, among them DROID, for gathering all sorts of meta data, ig. the EXIF header information of picture files. These information is then stored in an unified XML structure for further processing. C3PO (“Clever, Crafty Content Profiling of Objects”) then uses this XML files and stores the information in a database for easy access, and it can export the collected information as character separated values (*csv*).

The *csv* files produced by C3PO can be used to build up a configuration using the import function of the configuration editor. Figure 3.3 shows the editor with an imported configuration. For grouping the file type versions to file type families I used the mime type field of the C3PO output as it has been the only field that modeled this information correct in all my test sets.

A bit of a problem is the validity period of the file type versions. This information is available in the PRONOM registry, but it is difficult to obtain it automatically and is incomplete in many cases. For this reasons I decided to use the fist and the last ingest of a file belonging to that file type version to derive the validity period of that version, as this closely shows in which time frame compatible software was in use.

During the import of the *csv* data, file types, versions and ingests get configured following the structure of the initial scanned repository. After that migration rules and tools must be added manually to complete the configuration. After storing the configuration the simulation can be

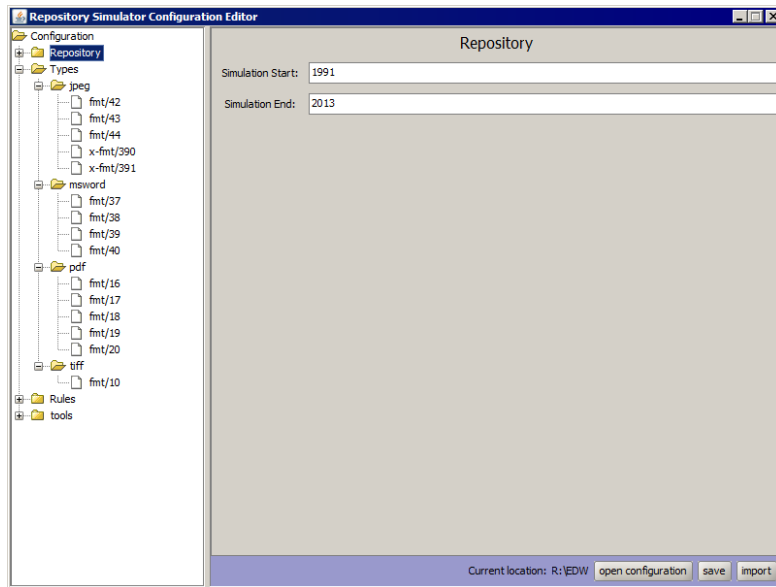


Figure 3.3: Configuration Editor with a configuration imported from csv data

started.

3.5 Output

After the simulation the result of the simulation must be analyzed, as it is the main goal of the simulator to show how well a certain configuration performed. The result is presented in 3 different ways:

- **Diagrams:** There are several diagrams produced by the simulator that show the main information about the simulation process.
- **csv Data:** For the case that the result of the simulation needs to get processed further the main data is exported in form of a csv file that can be imported into various tools.
- **the database:** If special information is needed that is not covered by the diagrams and the csv data the database itself can get investigated. It contains all information of the simulation process as on every ingest and every migration the simulated timestamp and used tools/rules are stored.

The diagrams and the csv file are generated by a separate tool that can analyze the database after the simulation and collect the needed data. In the following the diagrams produced by this statistics collector are shown. The diagrams are based on the sample configuration shown in the previous section.

Figure 3.4 shows how many migrations took place every year. This way it is easy to see whether the are well distributed over the years or if there are hot spots with extremely many

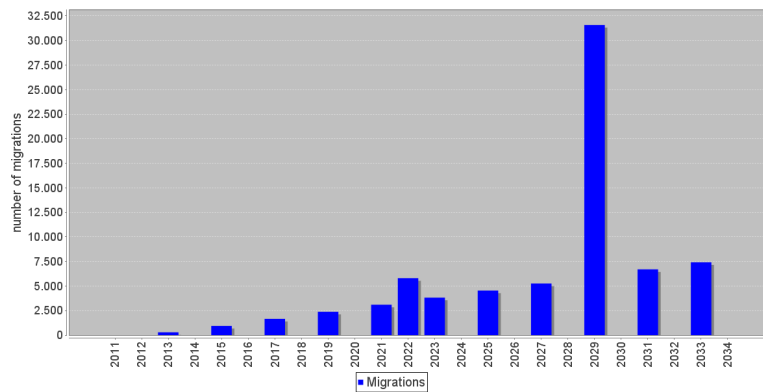


Figure 3.4: diagram of the number of simulations grouped by year

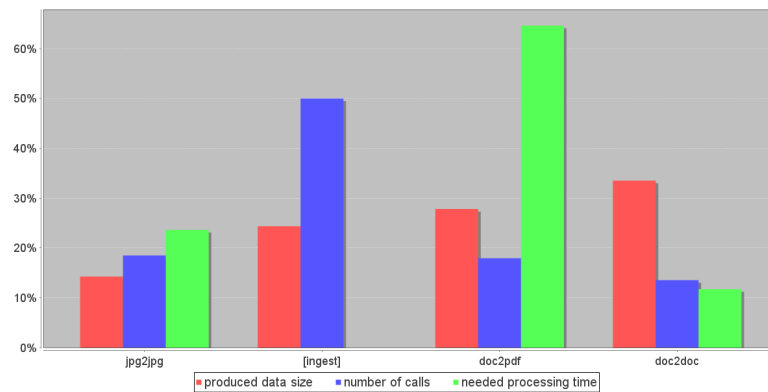


Figure 3.5: diagram of the accumulated storage volume and needed processing time generated by the used tools in comparison to the ingests in percent

migrations, which can lead to resource bottlenecks. In this case we see regular migrations every two years. These are the *doc* files that expire every two years. Besides that we have two irregular spikes - a smaller one in the year 2022 and a larger one in the year 2029. These are the years the *jpg* files get migrated. The first one is very small because at this moment we do not have many *jpg* files in the archive, as ingest of this type start only in 2020. In the year 2029 we already have a lot of *jpg* files in the archive, so the spike is very big. Additionally this overlaps with a *doc* migration point, leading to a big burst of migrations.

Figure 3.5 shows the usage of the tools used for the migrations. The left bar (red) shows how much data was generated by this tool, the middle bar (blue) shows how often the tool was called and the right bar (green) shows the needed processing time by this tool. For comparison the ingest operations are also shown in the diagram. In this case the diagram shows that the most data was produced by the tool *doc2doc*, but the difference to *doc2pdf* and *jpg2jpg* is not very high. In contrast, the processing time needed by *doc2pdf* is far more than *doc2doc*, so it may be recommended to search for a more efficient tool as replacement for *doc2pdf* if processing time

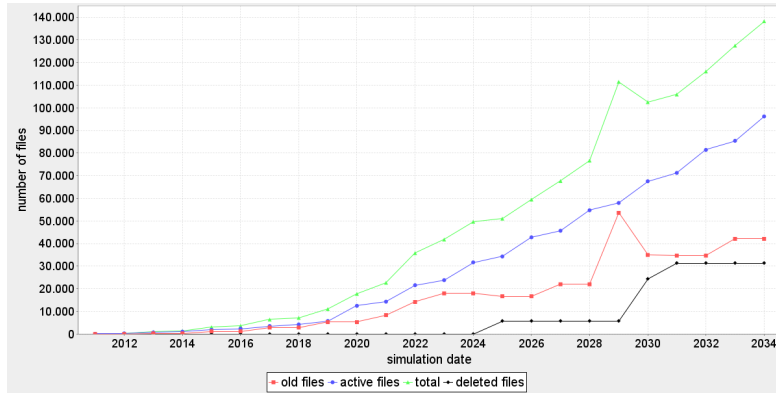


Figure 3.6: diagram of the number of files in the repository over the simulation time

reaches a critical amount.

In figure 3.6 the number of files in the archive is shown. The line with the blue circles shows the number of files that are still in use at that point of time, the line with the red squares shows the files that are left over after the migrations and the third line shows the sum of both. The black line with the diamonds shows the number of deleted files. In this example we see that the number of active files grows slowly but rather constantly, as we have only periodically ingest in this simple sample configuration. The old files grow nearly the same way in the beginning, but then the curve flattens around the year 2025. This is the time when the deletion rule kicks in for the first time, deleting the first *jpg* files from the repository. Then there is the most visible peak in 2029. We already discussed in the migration diagram that there are many *jpg* migrations and some *doc* migrations in this year, which leads to a massive growth of the old files left behind after the migrations. After this peak the graph goes down again. This shows the second wave of deletions triggered by the deletion rule. The graph of the total file number follows both the active file number and the old file number, as it is just the sum of both.

Note that this is just the number of the files and not the needed storage space. This number can be of interest if there are periodical tasks on the files independent from their size, eg. reviewing meta data or querying for files with certain attributes.

Figure 3.7 is similar to figure 3.6 but shows the sum of size of the files instead of the number. Again blue circles shows the files that are still in use, the line with the red squares the files that are left over after the migrations and the third line the sum of both. In this case the both diagrams are very similar because we do not have many rules defined in the configuration. Usually there are well visible spikes where important file format versions get migrated to other formats, especially if several migration waves overlay each other, showing hotspots with abruptly growing resource needs like in year 2029 in this case.

The generated diagrams give good oversight on what happened during the simulation and where the main problems can occur. Yet, in some cases it might be desirable to take a closer look into the processes inside the repository. In this cases the simulation result can be analyzed using SQL. The structure of the database was already discussed in section 3.1. A possible sample select is shown in figure 3.8. In this example the processing time for the tool *doc2pdf* was asked,

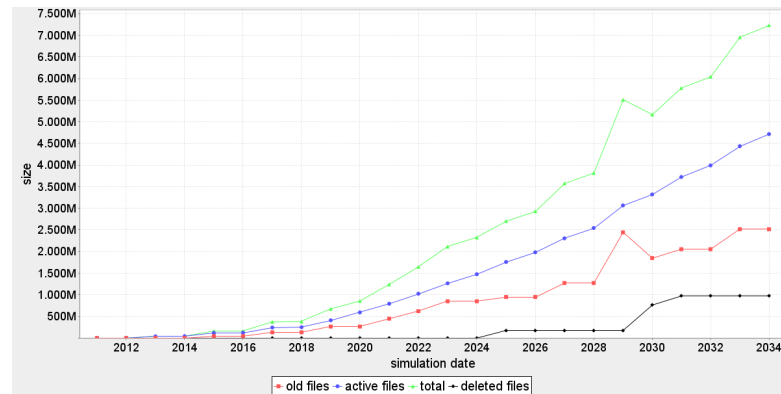


Figure 3.7: diagram of the total size of the repository over the simulation time

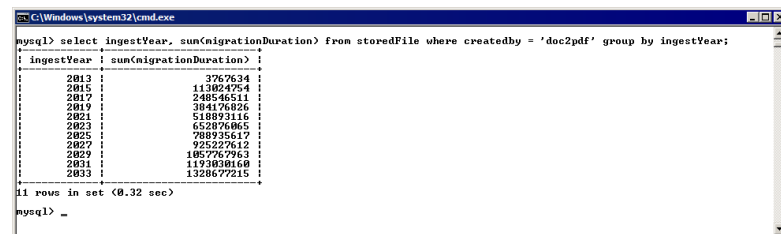


Figure 3.8: Gathering simulation results from the database using SQL

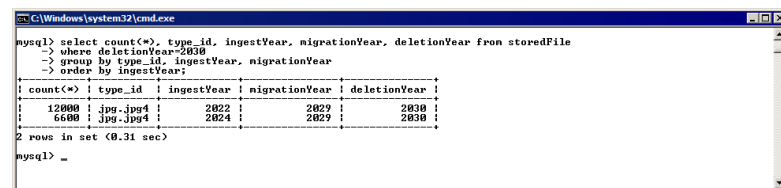


Figure 3.9: Using SQL to analyze the deletions in the year 2030

summed up for every year to get an impression on how far it increases over time.

A second example for deeper analysis is shown in figure 3.9. I used the shown select when writing the description for figure 3.6 (the number of files in the repository). I wanted to see which files were deleted in the year 2030, especially when they were ingested/migrated, because the increase of deletions in this year was more than I had expected.

3.6 Generating Configurations

A usual scenario is that there is an existing archive which should be the base of the simulation setup. How to reprocess existing data strongly depends on the type of information and how it is stored and usually is not possible without manual intervention. However there are tools that

support this process.

If the files are available through the file system and there is no information about the content of the files (as it was the case in the example in chapter 5), the tools FITS and C3PO can help to automate the operating sequence.

FITS (File Information Tool Set) is specialized in collecting metadata on large file collections. It does this by including several tools that extract information about the files and brings the output of the tools in a uniform xml structure.

The result of a FITS scan is one xml per file with all the metadata in it. As it is difficult to handle such an amount of separate files it is necessary to bring them into a more handy form. For this part I used the tool C3PO (Clever, Crafty Content Profiling of Objects). C3PO can read the xmls and store them in a database in a consolidated form, which means that the output of the distinct tools is combined to a single data record. This database can then be exported to a csv file with one line per input file, which is good for further processing.

For processing this csv file I have assembled a helping class that processes this file and produces the ini files needed for the simulation (`com.wec.reposimulator.util.C3poReader`). This class reads the file and generates ingest events and file type descriptions matching the described files. Usually the types have to be manually reworked as it just takes the first and the last usage of a type version for the validity period, which may match quite good for old versions that are not in use anymore, but it will not match for current versions, as there is no information on how long they will be usable in the future. (An approach how to predict the future behavior of file types based on the past development is discussed in chapter 6.)

The class `C3poReader` should work with the output of C3PO in most cases but will need some fine tuning occasionally. Beyond that it should provide a good guideline for processing the output of various tools.

3.7 Performance

The duration of a simulation depends on the configuration and the used hardware. On the configuration side the main factor is the number of ingests and migrations that are processed during the simulation. Both events have nearly the same processing time as they are very similar. With the current implementation the simulator can process on average 1500 events per second on a current multicore system. For simulations with several millions of ingests/migrations you have to expect a runtime in the range of an hour. If you are preparing a simulation bigger than that it is reasonable to reduce the ingest by a constant factor (10 or 100 for example) until you verified that the configuration is valid, and do only the final run with the full number.

3.8 Summary

The repository simulator provides flexible configuration possibilities that empowers the user to specify many different scenarios fitting closely to real world scenarios. The configuration is done in plain ini-files, so that it is human readable and comprehensible. For convenience a configuration editor helps to set up a valid configuration and import data gathered from existing archives.

The output is presented in several diagrams that show the main processes in the repository. They focus on size and number of the files and the number of migrations done every year. If the output does not show the wanted information the database itself can be consulted to reveal more special details on the process.

After inspecting the results one should have a good prospect on the outcome of the simulation and it should be easily comparable to similar simulations leading to good decisions for further preservation planning.

type	version	created	expired
doc	doc1	1999	2002
doc	doc2	2001	2004
doc	doc3	2003	2006
doc	doc4	2005	2008
doc	doc5	2007	2010
doc	doc6	2009	2012
doc	doc7	2011	2014
doc	doc8	2013	2016
doc	doc9	2015	2018
doc	doc10	2017	2020
doc	doc11	2019	2022
doc	doc12	2021	2024
doc	doc13	2023	2026
doc	doc14	2025	2028
doc	doc15	2027	2030
doc	doc16	2029	2032
doc	doc17	2031	2034
doc	doc18	2033	2036
doc	doc19	2035	2038
jpg	jpg1	2000	2011
jpg	jpg2	2007	2018
jpg	jpg3	2014	2025
jpg	jpg4	2021	2032
jpg	jpg5	2028	2039
jpg	jpg6	2035	2046
pdf	pdf1	1990	2006
pdf	pdf2	1997	2013
pdf	pdf3	2004	2020
pdf	pdf4	2011	2027
pdf	pdf5	2018	2034
pdf	pdf6	2025	2041
pdf	pdf7	2032	2048

Table 3.2: types and versions generated by listing 3.2

Tool Evaluation

4.1 General Issues

For simulating migration events it is very important to evaluate migration tools in order to set up the parameters as close as possible to the real migrations. In the following chapters 5 and 6 there are two case studies that show the simulator in possible use cases, and for this simulations several tools are needed. In this chapter I will try to evaluate possible tools and set up good configurations for the following simulations.

For all tools the process is usually the same and consists of the following steps:

- Finding a tool that meets the needed criteria.
- Create a sample set of files to test the tool on.
- Execute the test.
- Analyze the result.

After this steps it is possible to set up an appropriate configuration for this tool. In the simulation configuration the tools can be named freely. I usually use the name convention “<source>2<destination>” instead of naming it after the used program so it is easier to see the usage.

A very important point during analyzing the result is to evaluate whether the resulting files have the appropriate quality for the storage process, which means that the it must contain all information that should be preserved. This aspect usually depends on the storage goals of the repository. In my examples I did a limited test on a random subset of files and compared the result visually.

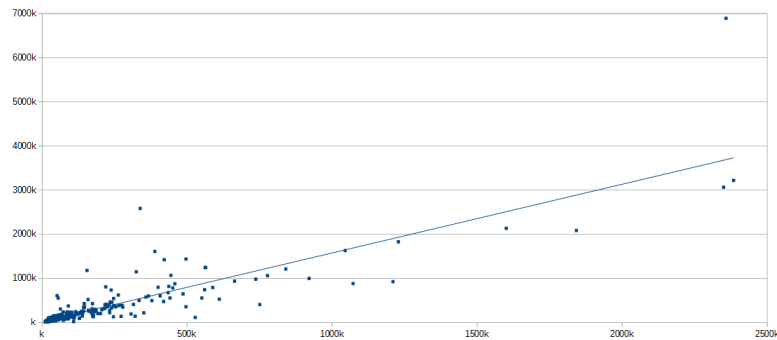


Figure 4.1: Regression Analysis of file growth when converting between Word97 to Word2003

4.2 Doc2doc

For the conversion between versions of the Microsoft Word format family I selected LibreOffice as conversion tool, as it supports many different versions and has a headless mode for batch conversion processes.

As sample set I collected 370 random files using Google and filtering to “filetype:doc”. The files are wide spread in different size and content. The size varies from about 11Kb to 5Mb with a total size of 76Mb.

In a first attempt I tried to convert all files back to “Word 95”, and then run several conversions to “Word 97”, “Word 2003” and “Word 2007”, and then average the result. Unfortunate that was not possible, as the conversion backend of LibreOffice proved to have problems with many files, resulting in program crashes or unreadable files. The only conversion step that worked on all input files was from “Word 97” to “Word 2003”, so I took that for my scenario.

The conversion took 59 minutes and 40 seconds, resulting in a total file size of 107Mb. A regression analysis of the file size growth (see 4.1) showed that the growth rate is linear (coefficient of correlation: 0.79) with a gradient of 40%. The resulting configuration is shown in listing 4.2

Listing 4.1: tool doc2doc

```
[tool]
name = doc2doc
size = currentsize * 1.40
duration = currentsize * 47
```

4.3 Pdf2pdf

For converting PDF files to the next higher version I used GhostScript. As input files I took the Word documents from the previous section and converted them to PDF 1.4 using LibreOffice. From there, I did consecutive conversions to version 1.5 and 1.6 using GhostScript.

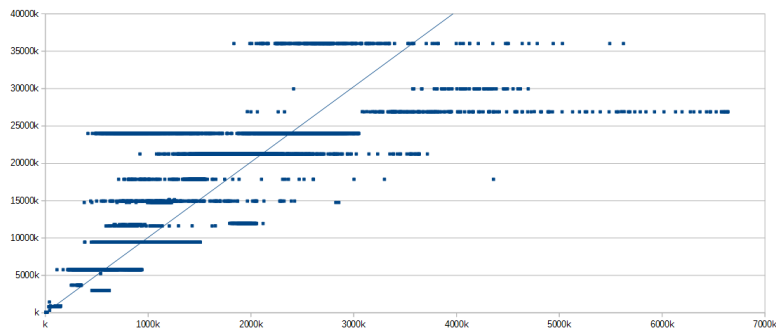


Figure 4.2: Regression Analysis of file growth when converting between jpg to tiff

In all three versions the file size stayed almost the same, only varying a few bytes. The reason therefore is that PDF is a very stable format, and new versions mainly add new features without changing existing file structures. (On the opposite to Word documents, where the file format was totally reworked between several version steps.) The conversion was very fast with 3,7 seconds per Mb.

Listing 4.2: tool pdf2pdf

```
[ tool ]
name = pdf2pdf
size = currentsize
duration = currentsize * 3,7
```

4.4 Jpeg2tiff

To set up a tool for the conversion of jpg files to tiff, I did an experiment on my private photo archive. To ensure that I have a good diversification over different image sources and over time, I only took a fraction of the archive which consists of pictures collected from various cameras on a yearly event over 9 years. The input for my experiment contained an amount of 13834 jpg files with a total size of 11.5GB. For converting the files to tiff I used IrfanView [19] in batch mode. The resulting images had a total size of 133GB. The regression analysis (4.2) shows again a linear growth rate (coefficient of correlation: 0.87) with a growth rate of the factor 10.09. The conversion took 67 minutes, which results in an average conversion time of 0,43 seconds per Mb.

Listing 4.3: tool pdf2pdf

```
[ tool ]
name = jpeg2tiff
size = currentsize * 10.09
duration = currentsize * 0.43
```

4.5 Summary

The examples above show how tool configurations can be created based upon the behaviour of real tools. Good tool configuration plays an important role during the simulation and can have a significant influence on the simulation.

Case Study: EDW

5.1 General Issues

The evaluation of the result is a rather difficult task. The main problem is that digital archives are usually very different in respect to their content. It is impossible to find one or more data sets that are representative for all archives, as every archive has a different preservation goal and different content sources. And depending on the duty of an archive there are different requirements in what is needed to preserve. In some cases the highest priority may be preserving the exact layout of the stored documents, in other cases maybe keeping the documents editable is most important. Those requirements lead to very different migration policies.

Another problem is that there are not many archives that are available in public, so it is difficult to find a source for sample data which is freely accessible and is big enough to collect enough data for a meaningful sample configuration.

In the following, I will show a configuration based on an existing archive to which I had been granted access for this work. I will add several rules and show the according output diagrams documenting the changes in the repository so one can get an insight how to work with the simulator and interpret the output.

5.2 Sources of Sample Data

In search for good sample data to test the repository simulator I got the permission to use the office archive of the Austrian childcare organisation “Katholische Jungschar der Erzdiözese Wien” [7] (in the following referred as “EDW”) for indexing the stored data and building up a sample configuration. The EDW archive has a size of about 40000 files, mainly consisting of text documents, but also including some images and other files.

The data was collected during the organisation of educational events. The archive consists of meeting protocols, preparation documents, information material for the participants, feedback discussions and some press photos. Since the year 2000 the archive also contains the preparation

materials for a magazin appearing four times a year. (The magazine also appeared before the year 2000, but there it was produced only with analog devices and so there was no data to collect.)

The archive is not very big, but it is interesting for several reasons: It is built not by computer trained personal but people using the software they are used to. The personal in the office consists of many people working at home voluntary, so they are not tied to use the software available in the office but usually use what is available on a standard home computer. And finally the events for that the data is collected are very similar over the years. To sum up, the reasons for data collection stays rather constant, but the technological background is constantly adapting, which should give us a clear picture of the technological change in the small office/home office area.

For gathering the type information and building up my sample configuration I used the tools FITS and C3PO. FITS is a tool that combines multiple meta data extraction tools and brings the output in a unified XML based structure. C3PO parses the output of FITS and normalizes it and stores it in a searchable database. The metadata can then be exported to a csv file, which is good for further processing.

5.3 Sample Configuration

The EDW archive contains 39440 files with a total size of 40.7 GB. FITS and C3PO are able to identify the exact file format version of 15145 of them. For the other files it was not possible to determine the matching PUID and therefore they cannot be used for this example. The identified files were distributed over several file type families. The four families “jpeg”, “msword”, “pdf” and “tiff” contain the most part of the files; there are only 527 files belonging to other families. To keep this example concise I filtered out those 527 files and focused on just the four main families, leaving 14618 files in the simulation. Table A.1 in the appendix shows the complete list of ingests generated from the remaining files along with the average size of the files.

A very important part of the configuration is the validity period of the file types and versions. It is difficult to predict how long a version will be supported, even the information about passed versions is only limited. PRONOM tries to collect that information for known file type families, but the information there is far from complete. This makes the estimation of the validity periods a very complex and broad topic, which is not covered by this work. In this simulation I tried to manually estimate the life cycle of the used types to get plausible values. The main assumption is that when a version is not used anymore, then the support for this version will expire soon after that. In this case the result much more reflects the real world than official support periods, because people in private areas usually do not adapt to new software very fast, and they often continue to use the same version as long as it fulfills their needs, even if there is no official support anymore.

Figure 5.1 shows the configuration editor with the types imported from the EDW archive and in table 5.1 you can see the types and versions listed in detail.

The clearest sequence arrangement of versions is observable within the *msword* family. Probably this is because *msword* files are tied to the Office Suite by Microsoft and so the usage

type	version	created	expired
jpeg	fmt/43	1997	—
jpeg	fmt/44	1997	—
jpeg	x-fmt/390	1997	2007
jpeg	x-fmt/391	2000	2012
jpeg	fmt/42	2002	2011
msword	fmt/37	1990	1993
msword	fmt/38	1991	2006
msword	fmt/39	1993	2008
msword	fmt/40	1997	—
pdf	fmt/16	1999	2006
pdf	fmt/17	1999	—
pdf	fmt/18	2001	—
pdf	fmt/19	2003	—
pdf	fmt/20	2003	—
tiff	fmt/10	1994	—

Table 5.1: types and versions in the EDW repository

of these formats depends strongly on Microsofts release cycle. The other formats are used by multiple software vendors so the picture is not that clear in that cases.

5.4 Experiments

To get a first insight of how my files are distributed in this archive, I ran the simulation once without any rules so I can see just the ingest events in the diagrams. In figure 5.2 the distribution of the files over the file types is shown: In the first years mainly files of the type msword are in the archive. Since the turn of the millennium also ingests of pdfs happen and increase over time while the msword documents stagnate in the end. Also the jpgs start around the year 2000 and climb up very fast.

As we see in table 5.1 there are several versions that run out during the simulation, we have to add some rules to the repository configuration, as this means that the files stored in this format versions will no longer be accessible because the software does not support them anymore. As a first scenario I defined a rule for each type family to migrate the files to the next higher version (ie. the version with the next higher expire date) some months before the current version expires (“minimal step size”) to keep all files accessible and up to date and we do not risk to loose the information inside the files.

In 5.3 we see the migrations caused by this rule set. One can see that the biggest fraction of migrations take place in the years 2005, 2006, 2008 and 2010. This is because around these dates format versions with many files expire and need to get translated to the next higher version. Especially in the year 2008 there are many migrations. In 5.5 we can see that these migrations

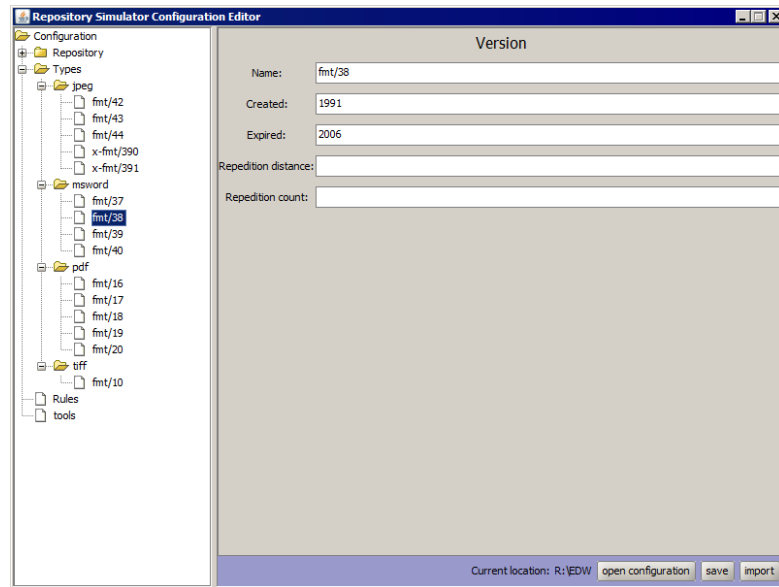


Figure 5.1: Configuration Editor with sample configuration

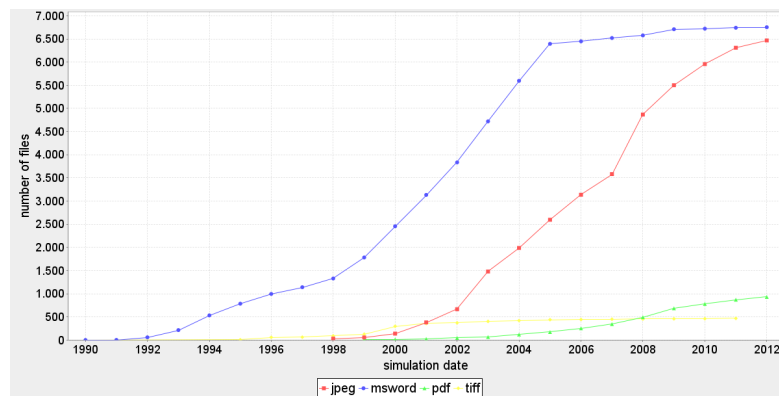


Figure 5.2: number of files per type

are mainly caused by the files of the type *mword*, as there is a big increase of files in the year 2008. The graph in figure 5.4 shows the total size of the archive. One can see that the total size increases drastically especially in the later years of the simulation where the most migrations take place and it is noticeable that there is already an amount of obsolete files in the repository. The total size of the archive reaches 10GB in the end of the simulation and the obsolete files have a portion of about 10%.

To reduce the needed storage size one possible approach is to reduce the number of migrations, in the case we want to keep all old versions. So I ran the next simulation with migrations to the version with the longest expire time available (“maximal step size”). This should reduce the overall number of migrations and lead to less old versions in the repository and thereby reducing

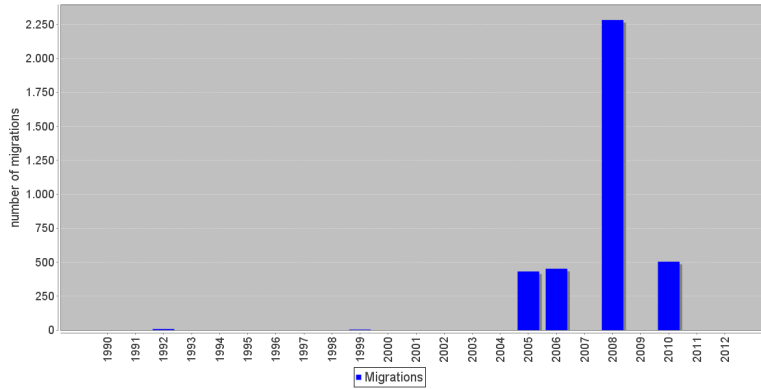


Figure 5.3: migrations with minimal step size

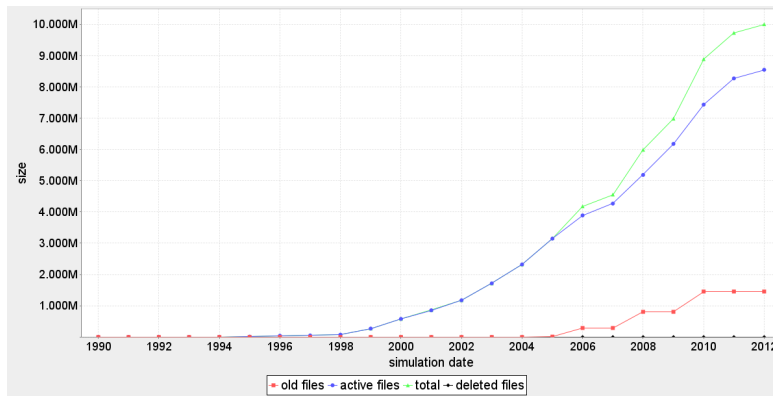


Figure 5.4: total size after simulation with minimal step size

the needed space of the archive.

Figures 5.6 and 5.7 show the result of this approach. In figure 5.6 we see that the number of migrations dropped as a result of this change. Note that we still have the migration peaks at nearly the same years as in the simulation with minimal steps. This is because the expire dates are still the same and the ingested files are distributed over all versions, so the peaks are the same. Figure 5.6 shows that the overall size of the archive dropped from about 10GB as shown in 5.4 to about 8.5GB. How much impact the change from “minimal step size” to “maximal step size” depends on how many overlapping versions there are in the repository. In this case it is a small but noticeable gain.

The next important step is to think about the reasonability of the used rules. In the last experiments the files belonging to the jpg family get more or less often migrated to newer versions of that family. That is probably not a wise choice, as jpg uses usually a lossy compression method and by migrating to a new version one risks that the picture is decompressed and recompressed, leading to information loss. It is considerable to migrate those files to the tiff format, as this format is lossless. An other argument for tiff over jpg is that it is much more stable, as we have

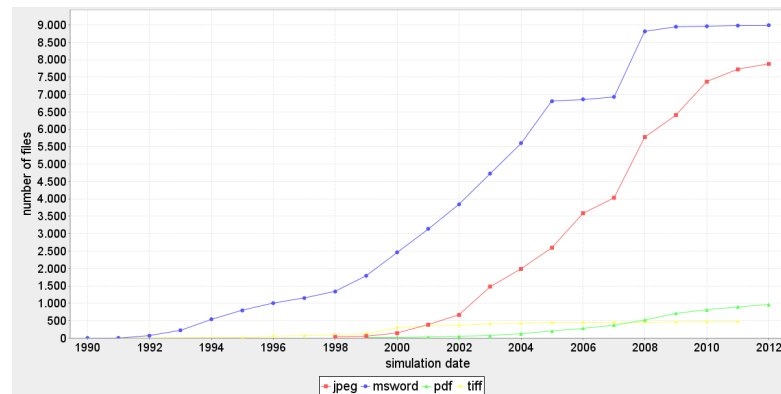


Figure 5.5: number of files after simulation with minimal step size

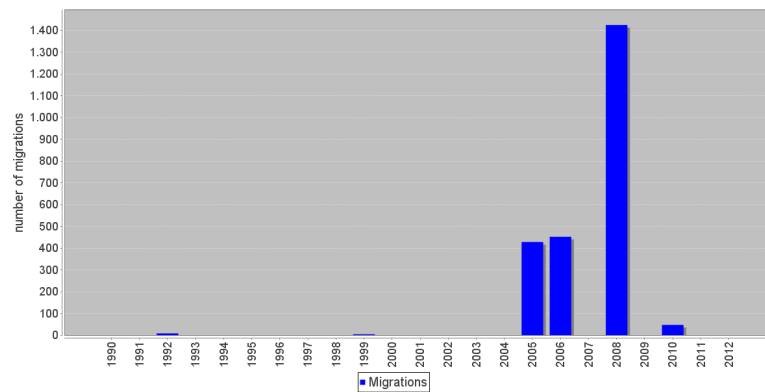


Figure 5.6: migrations with maximal step size

5 different versions of jpg in the EDW archive, but only one of tiff.

So the next step is to set up a rule for migrating jpg files to tiff. There are two possibilities when to do that: Either when the jpg version expires or straight after the ingest of the file. For the next experiment I ran both variants to compare the results.

Figure 5.8 shows the growth of the archive in case of migrating the jpg files on expire and 5.9 shows the result for migration on ingest. Of course, the size in the end of both simulations is virtually identical, yet the characteristics differ. In the first case we have the biggest increment in the overall size in the year 2006, where the size increases drastically over 100% of its previous size, whereas the rest of the characteristics are rather flat. When migrating on ingest, the size increases continuously over the time. This is because we have constant ingest of new pictures, but a huge amount of pictures expiring on the same date.

So, comparing the two results one can decide whether it is more important to postpone the increase of size by migrating the files on expire, or to distribute the increase over time to avoid peaks in the hardware requirements.

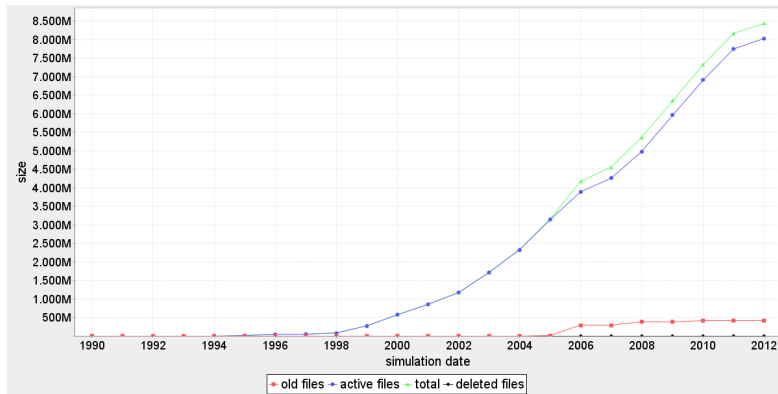


Figure 5.7: total size after simulation with maximal step size

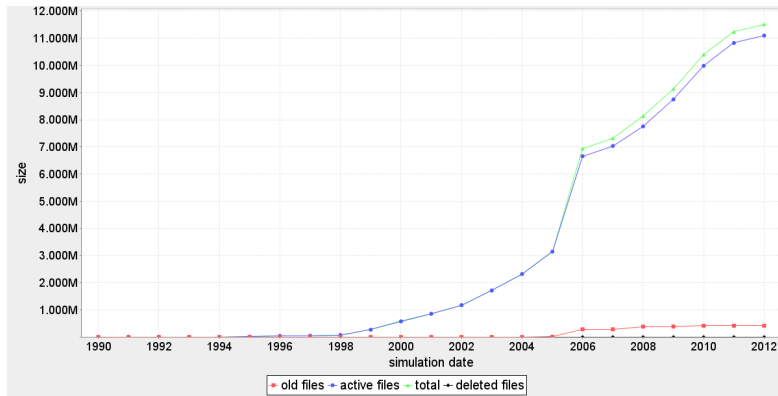


Figure 5.8: migrating jpg to TIFF on expire

5.5 Summary

These experiments show how one can work with the simulator and interpret the results. The graphs should give a clear look on the changes happening in the repository and give good advice on the decisions that have to be made when maintaining a repository. Note that this sample simulations are based on a very particular configuration and that the results with the same rule sets will differ on each repository, so it is important to set up the simulation matching the target archive as close as possible.

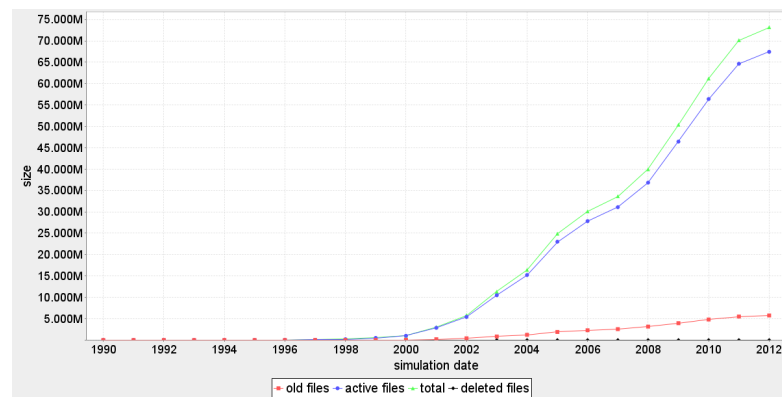


Figure 5.9: migrating jpg to TIFF on ingest

Case Study: Danish Web Archive

6.1 General Issues

For a second case study I got access to the Danish Web Archive. This archive is the result of a yearly web crawl, executed by the StatsBiblioteket (The State and University Library) in Aarhus, Denmark¹. It represents the yearly state of the Danish part of the World Wide Web. As this is a very large amount of data I only got a fixed subset of one million files per year, starting in the year 2005 and ending 2012. And I did not get the data files itself but they were preprocessed by FITS and I got the resulting xml files to work with.

The characteristics of the data set imply several limitations:

- As the data does not give an insight on the total size of the archive it is not possible to make predictions about the absolute development of the archive.
- Because of the short time frame predictions about the development of version usages are difficult.

The second limitation is especially problematic since the usage and life-cycle of each version is the basic configuration for the simulator. To solve this problem there is an interesting work written by Stefan Schindler [18], who analyzed this dataset and was able to extract the life-cycle of the type families HTML, PDF and FLASH by doing statistical analysis. His theory is based on the expectation that different type versions in each family show similar usage behaviour over time. And because every version has a different release date the 8 years represented in the dataset show different parts of the life-cycle of each version. Based on this theory he could expand the observation time frame and calculate the average usage for each type family. The result of his regression analysis is the description of the usage by the following polynomial functions²:

¹<http://www.statsbiblioteket.dk/>

²These functions were included in Schindlers paper only in form of plotted graphs, so I contacted the author to get them in textual form.

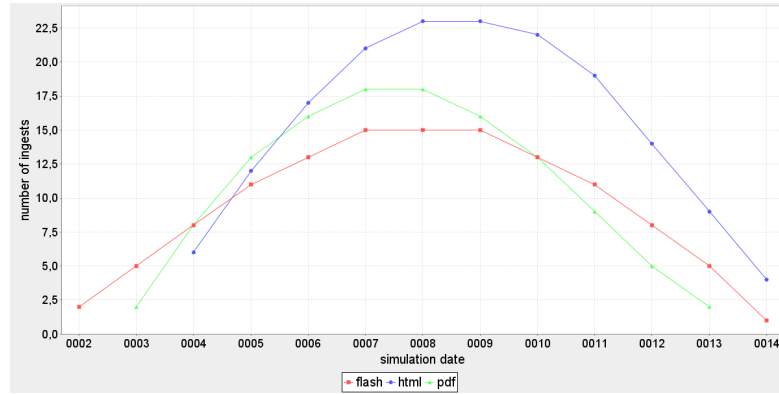


Figure 6.1: Basic setup: Ingests per file type

PDF:

$$y = 2.912 - 1.038e + 01 * x + 5.310e + 00 * x^2 - 7.522e - 01 * x^3 + 4.149e - 02 * x^4 - 7.965e - 04 * x^5$$

HTML:

$$y = 1.9385275 - 8.2927686 * x + 3.7782122 * x^2 - 0.4214922 * x^3 + 0.0165867 * x^4 - 0.0001897 * x^5$$

FLASH:

$$y = -2.990e - 01 - 1.299e - 01 * x + 1.060e + 00 * x^2 - 1.338e - 01 * x^3 + 4.444e - 03 * x^4 - 1.363e - 05 * x^5$$

6.2 Basic simulation setup

In this section I tried to set up a basic simulation based on Schindlers results. Fortunately his formulas can be used only with syntactical changes directly in the simulation. So for the first run I set up three file types and a repetitive ingest for each with the following file quantities:

PDF:

$$quantity = 2.912 - 1.038e + 01 * typeYear + 5.310e + 00 * Math : pow(typeYear, 2) - 7.522e - 01 * Math : pow(typeYear, 3) + 4.149e - 02 * Math : pow(typeYear, 4) - 7.965e - 04 * Math : pow(typeYear, 5)$$

HTML:

$$quantity = 1.9385275 - 8.2927686 * typeYear + 3.7782122 * Math : pow(typeYear, 2) - 0.4214922 * Math : pow(typeYear, 3) + 0.0165867 * Math : pow(typeYear, 4) - 0.0001897 * Math : pow(typeYear, 5)$$

FLASH:

$$quantity = -2.990e - 01 - 1.299e - 01 * typeYear + 1.060e + 00 * Math : pow(typeYear, 2) - 1.338e - 01 * Math : pow(typeYear, 3) + 4.444e - 03 * Math : pow(typeYear, 4) - 1.363e - 05 * Math : pow(typeYear, 5)$$

Figure 6.1 shows the ingests of this configuration. This is not a reasonable simulation at the moment, but just a check whether the formulas work as expected. And as one can see the graphs match the ones in Schindlers paper exactly.

The next step is to add new versions in each family in proper intervals. According to the table of release years in Schindlers paper new versions appear virtually every year. Additionally the quantity of the ingests must be adapted, as the result of the formulas is just a percentage and not an absolute value. For this I summed up the usage of each version in the dataset. The tables 6.1, 6.2 and 6.3 show the collected data. As one can see in the data, the ingests of the different

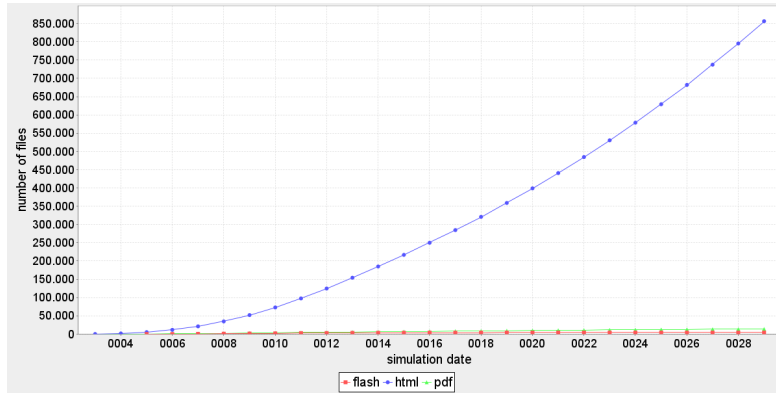


Figure 6.2: total number of files per type family

families are not constant, but HTML keeps growing while PDF is dropping over the years and FLASH nearly vanishes in the last years. I reflect that behavior in the quantity formulas and ended up with the following configuration³:

PDF:

$$quantity = 81 * Math : pow(0.969, elapsedYears) * (2.912 - 1.038e + 01 * typeYear + 5.310e + 00 * Math : pow(typeYear, 2) - 7.522e - 01 * Math : pow(typeYear, 3) + 4.149e - 02 * Math : pow(typeYear, 4) - 7.965e - 04 * Math : pow(typeYear, 5))$$

HTML:

$$quantity = 928 * Math : pow(1.048, elapsedYears) * (1.9385275 - 8.2927686 * typeYear + 3.7782122 * Math : pow(typeYear, 2) - 0.4214922 * Math : pow(typeYear, 3) + 0.0165867 * Math : pow(typeYear, 4) - 0.0001897 * Math : pow(typeYear, 5))$$

FLASH:

$$quantity = 3171 * Math : pow(0.695, elapsedYears) * (-2.990e - 01 - 1.299e - 01 * typeYear + 1.060e + 00 * Math : pow(typeYear, 2) - 1.338e - 01 * Math : pow(typeYear, 3) + 4.444e - 03 * Math : pow(typeYear, 4) - 1.363e - 05 * Math : pow(typeYear, 5))$$

³Because this is a rather big simulation I reduced the number of ingests to 10% to speed up the simulation runs.

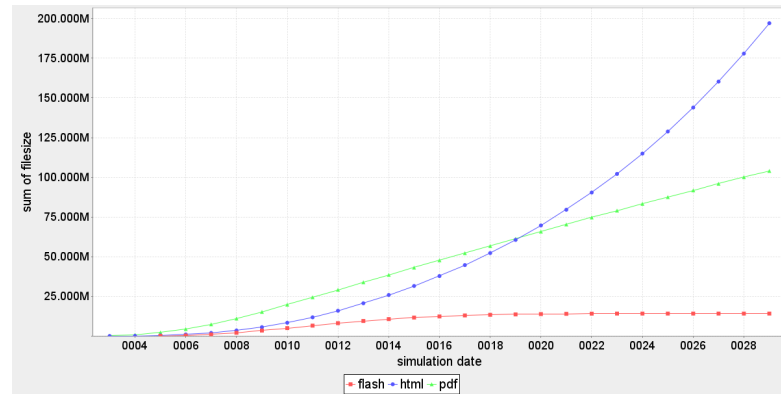


Figure 6.3: total size of files per type family

	fmt/14	fmt/15	fmt/16	fmt/17	fmt/18	fmt/19	fmt/20	fmt/95
2005	9	44	1265	2863	2143	483	39	
2006	11	24	1244	3401	3202	560	222	
2007	3	37	1076	3183	2777	414	236	
2008	3	22	621	3005	3769	507	421	2
2009	3	8	404	2009	2476	630	256	1
2010	3	12	64	1301	1168	1419	180	2
2011		5	36	691	515	1834	89	3
2012		7	42	605	652	1614	119	1

Table 6.1: number of ingests for the family PDF

	fmt/96	fmt/97	fmt/98	fmt/99	fmt/100	fmt/102	fmt/103
2005	195682	607	5936	34568	64788		
2006	162994	187	4436	28325	79281		
2007	192776	154	5007	37177	95021	26811	
2008	128732	127	4147	27164	70835	50322	1258
2009	106774	90	3964	25118	72247	60945	2053
2010	165105	164	7369	37459	110778	127039	4006
2011	182639	141	15349	34627	132945	184252	6739
2012	161491	190	10125	25981	114888	166795	5943

Table 6.2: number of ingests for the family HTML

	fmt/104	fmt/105	fmt/106	fmt/107	fmt/108	fmt/109
2005	2	6	62	2396	2058	363
2006		1	76	936	1216	316
2007		2	46	759	820	206
2008		1	32	479	511	256
2009		5	22	281	308	164
2010		1	4	52	153	58
2011			4	15	187	16
2012			7	42	126	24

Table 6.3: number of ingests for the family FLASH

In figure 6.2 and 6.3 the result of this configuration is shown. Note that I started with an empty archive and no version defined in the beginning, and then successive added new versions and the corresponding ingests. After about 12 years the simulation reaches a fully established set of versions and ingests, and from that moment it matches the original dataset as close as possible. Of course the result looks very continuous and periodic as it is based on smoothed curves.

It is interesting to see, that although FLASH and PDF have a very low fraction of files in the repository compared to HTML (Which is not a big surprise, as it is a web archive.) but they do have a considerable amount of storage size, because PDF and FLASH have a much higher average file size.

6.3 Migration experiments

As usual the next step is to schedule migrations for the disappearing file versions. I have done three different scenarios for comparison.

- Scenario 1: I assume that when the usage of a version drops under 10 percent, then this version will expire soon and it is time to migrate to a newer version, and in this case to the next higher version. The moment of migration is reached 11 years after version release in case of PDF, 13 years after release in case of HTML and 12 years in case of FLASH. Listing 6.1 shows the rule configuration for PDF as example.
- Scenario 2: In this case migration takes place in the same moment as in scenario 1, but destination version is the best available version.
- Scenario 3: As a third option I tried to migrate every file on ingest straight to the best version available. As this is not reasonable in the first years of a version, I postponed the migration for the first 5 years. Because the maximum usage of each version lies several years after the 5 year mark this should migrate the majority of files on ingest. Listing 6.2 shows a sample rule. Additionally I kept the rules from scenario 3, as we still need to conserve the files at the end of the version life-cycle, although this should happen clearly later.

In the figures 6.4, 6.5 and 6.6 the migrations of this scenarios are shown, and figures 6.7, 6.8 and 6.9 show the total size of the archive. It is easy to see that in scenario 1 there are very many migrations and the overall size is growing very fast and high. Scenario 2 shows that there is a big gain when migrating to the best possible version as the resulting migrations drop by nearly 90%. There is a notable boost of migrations in the year 24. This is where the oldest files in the archive get migrated for the second time.

A bit surprising is scenario 3, as I would have expected to get a rather smooth resulting graph containing only small irregularities. In this case the planned migrations pile up more than I had expected. So in this scenario one has to be prepared to have unsteady resource needs over the years.

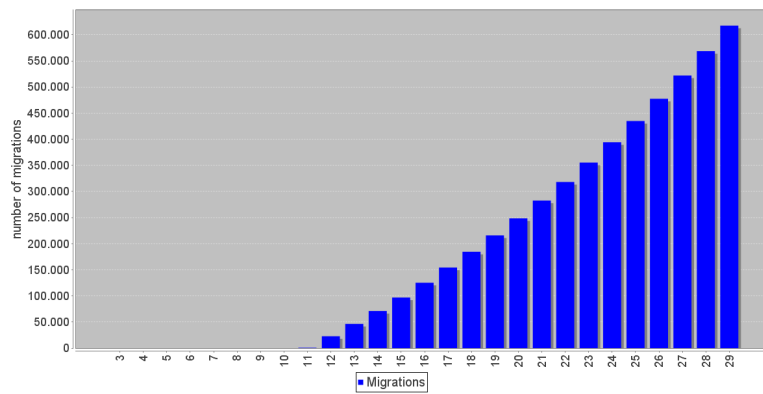


Figure 6.4: scenario 1 - number of migrations

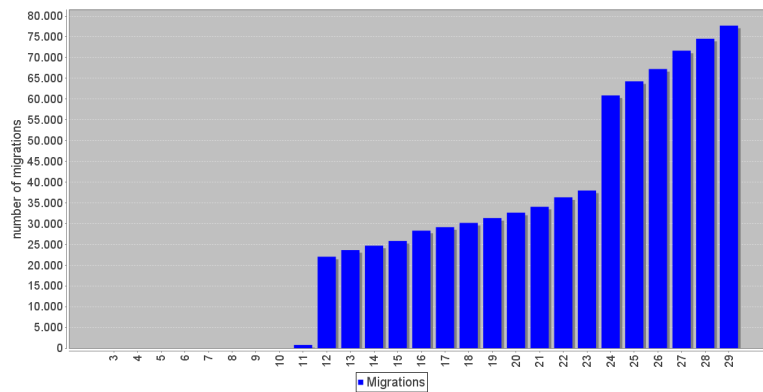


Figure 6.5: scenario 2 - number of migrations

Listing 6.1: scenario 1 - sample rule

```
[ migrationrule ]
name = pdf
type = pdf
term = version_created + 12*year
keeporiginal = false

[ destination0 ]
name = pdf
destination_type = pdf
destination_version = [ minimal_step ]
tool = pdf2pdf
source = current
```

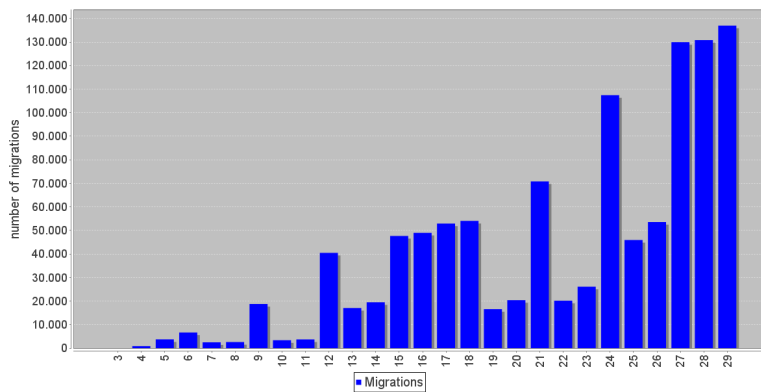


Figure 6.6: scenario 3 - number of migrations

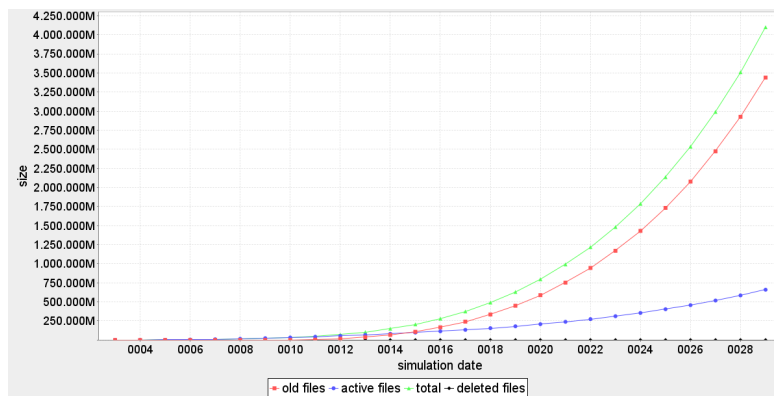


Figure 6.7: scenario 1 - sum of file size

Listing 6.2: scenario 3 - sample rule

```
[ migrationrule ]
name = pdf
type = pdf
term = Math:max( file_ingest , version_created+year*5)
condition = generation == 0
keeporiginal = false

[ destination0 ]
name = pdf
destination_type = pdf
destination_version = [ maximal_step ]
tool = pdf2pdf
source = current
```

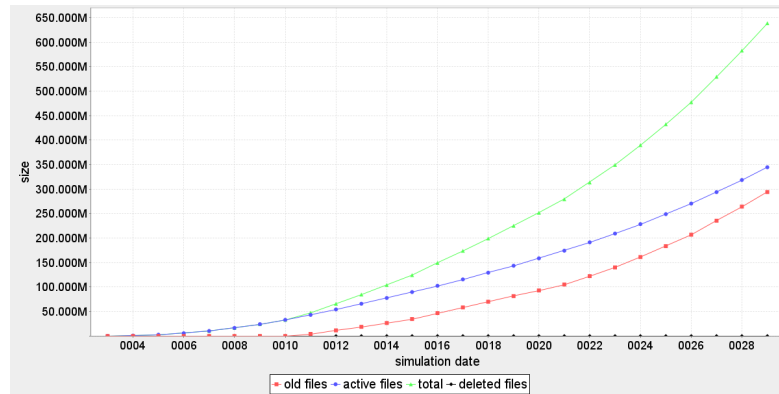


Figure 6.8: scenario 2 - sum of file size

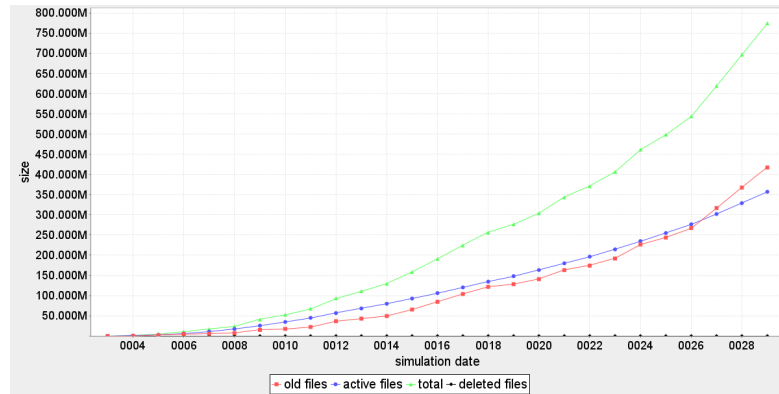


Figure 6.9: scenario 3 - sum of file size

6.4 Summary

This chapter shows how to set up simulation scenarios based on statistically analyzed data. It demonstrates that the simulator can work with complex formulas resulting from regression analysis or similar statistical tools. It also shows that in some cases the result is more chaotic as expected.

Summary and Future Work

7.1 Summary

The way how the growing amount of data produced every day gets preserved for future generations is one of the most important topics in the field of electronic data processing. Yet the longterm storage of digital information is still an underestimated afford.

The simulation tool presented in this thesis is a helpful tool for planning migration policies and maintaining a digital repository. The configuration offers flexible configuration possibilities to model real world situations very close. File types, ingests and migrations can be defined by easy understandable rules. It is possible to build up a configuration based on an existing archive to facilitate the configuration process. There are detailed diagrams that document the simulation result so that the simulator can be used to compare different scenarios and provide a good reasoning base for future decisions. For deeper analysis the database can be used to clarify details of the simulation results.

The configuration is very dynamic to give the user a big scope of development and allow to build up very detailed settings. This provides the potential to explore the differences between configuration changes and facilitates the decision between different strategies.

7.2 Future Work

The simulator is already a feature rich tool. Yet I am sure that there are many things that can be developed further. The most important aspect is the system architecture of the simulated repository. The current implementation reflects the repository as a single core system. As today multicore systems and cloud computing are standard technologies, a possible enhancement would be to reflect that in the simulation.

Also, switching to a multicore representation can possible open the way to performance gain if the simulation process is also split up to multiple processes. The current structure does not allow processing multiple migration events in parallel. Although the performance of the current

implementation is sufficiently optimized, execution time is something one should always keep in mind, as faster execution facilitates a higher number of simulation runs and so refines the information gained with the simulator.

Another way to gain performance is to base the simulation not on single files but on group of files. This path gives a tradeoff between performance and level of detail.

Appendix

A.1 EDW Ingests

year	type	version	number	avg. size
1990	msword	fmt/38	1	6198
1990	tiff	fmt/10	1	60928
1991	msword	fmt/37	2	28213
1992	msword	fmt/37	6	5778
1992	msword	fmt/38	49	5062
1993	msword	fmt/38	156	6622
1994	msword	fmt/38	164	7368
1994	msword	fmt/39	154	14013
1995	msword	fmt/38	5	7609
1995	msword	fmt/39	247	17874
1995	tiff	fmt/10	5	885366
1996	msword	fmt/38	5	15092
1996	msword	fmt/39	206	62745
1996	tiff	fmt/10	52	312340
1997	msword	fmt/38	6	20215
1997	msword	fmt/39	139	51134
1997	tiff	fmt/10	5	24452
1998	jpeg	fmt/43	4	341375
1998	jpeg	fmt/44	1	61664
1998	jpeg	x-fmt/390	26	383247
1998	msword	fmt/39	178	38766
1998	msword	fmt/40	11	26951
1998	tiff	fmt/10	28	248994

1999	jpeg	fmt/43	2	22039
1999	jpeg	fmt/44	2	212371
1999	jpeg	x-fmt/390	22	427738
1999	msword	fmt/38	2	5272
1999	msword	fmt/39	318	169278
1999	msword	fmt/40	133	359151
1999	pdf	fmt/15	4	564213
1999	pdf	fmt/16	3	457670
1999	pdf	fmt/17	4	5010633
1999	tiff	fmt/10	36	1628169
2000	jpeg	fmt/43	33	80694
2000	jpeg	fmt/44	22	163637
2000	jpeg	x-fmt/390	28	420696
2000	msword	fmt/39	122	46766
2000	msword	fmt/40	552	56423
2000	pdf	fmt/16	1	68608
2000	pdf	fmt/17	3	3077858
2000	tiff	fmt/10	170	1439781
2001	jpeg	fmt/43	38	310011
2001	jpeg	fmt/44	21	814293
2001	jpeg	x-fmt/390	14	781386
2001	jpeg	x-fmt/391	168	655768
2001	msword	fmt/38	4	6039
2001	msword	fmt/39	29	113561
2001	msword	fmt/40	644	53314
2001	pdf	fmt/16	8	406859
2001	pdf	fmt/17	1	524640
2001	tiff	fmt/10	59	1446516
2002	jpeg	fmt/42	1	64048
2002	jpeg	fmt/43	54	770857
2002	jpeg	fmt/44	46	355685
2002	jpeg	x-fmt/390	156	910074
2002	jpeg	x-fmt/391	27	435577
2002	msword	fmt/38	3	-2082
2002	msword	fmt/39	22	110696
2002	msword	fmt/40	677	65610
2002	pdf	fmt/16	5	-6566
2002	pdf	fmt/17	15	1486559
2002	pdf	fmt/18	3	1955406
2002	tiff	fmt/10	15	2197932
2003	jpeg	fmt/42	5	-5664

2003	jpeg	fmt/43	284	435754
2003	jpeg	fmt/44	124	421888
2003	jpeg	x-fmt/390	131	423441
2003	jpeg	x-fmt/391	269	732423
2003	jpeg	x-fmt/398	1	739795
2003	msword	fmt/38	1	3072
2003	msword	fmt/39	1	15360
2003	msword	fmt/40	884	58423
2003	pdf	fmt/16	2	712679
2003	pdf	fmt/17	14	366180
2003	pdf	fmt/18	3	132084
2003	tiff	fmt/10	32	1639119
2004	jpeg	fmt/42	1	23821
2004	jpeg	fmt/43	160	761104
2004	jpeg	fmt/44	94	888965
2004	jpeg	x-fmt/390	67	480100
2004	jpeg	x-fmt/391	186	814849
2004	msword	fmt/39	1	32768
2004	msword	fmt/40	874	134790
2004	pdf	fmt/16	2	39409
2004	pdf	fmt/17	15	776409
2004	pdf	fmt/18	34	977140
2004	pdf	fmt/19	1	307533
2004	pdf	fmt/20	1	5173614
2004	tiff	fmt/10	17	2829516
2005	jpeg	fmt/42	29	845894
2005	jpeg	fmt/43	173	1399597
2005	jpeg	fmt/44	86	1424481
2005	jpeg	x-fmt/390	4	503407
2005	jpeg	x-fmt/391	317	845701
2005	jpeg	x-fmt/398	1	175595
2005	msword	fmt/38	2	10961
2005	msword	fmt/40	796	124139
2005	pdf	fmt/16	1	85312
2005	pdf	fmt/17	12	938911
2005	pdf	fmt/18	40	579620
2005	pdf	fmt/19	1	254430
2005	tiff	fmt/10	19	1364188
2006	jpeg	fmt/42	1	62125
2006	jpeg	fmt/43	140	752995
2006	jpeg	fmt/44	182	465563

2006	jpeg	x-fmt/390	4	288125
2006	jpeg	x-fmt/391	213	973754
2006	msword	fmt/39	1	94208
2006	msword	fmt/40	53	385329
2006	pdf	fmt/17	27	3343877
2006	pdf	fmt/18	51	1619013
2006	pdf	fmt/19	2	63369
2006	tiff	fmt/10	3	4647554
2007	jpeg	fmt/43	188	692688
2007	jpeg	fmt/44	99	547809
2007	jpeg	x-fmt/391	158	560165
2007	msword	fmt/39	1	108032
2007	msword	fmt/40	70	71292
2007	pdf	fmt/17	30	678976
2007	pdf	fmt/18	61	863198
2007	pdf	fmt/19	1	213599
2007	tiff	fmt/10	8	3295319
2008	jpeg	fmt/42	1	1185408
2008	jpeg	fmt/43	1004	223673
2008	jpeg	fmt/44	83	760278
2008	jpeg	x-fmt/391	195	962739
2008	jpeg	x-fmt/398	3	144954
2008	msword	fmt/39	1	233472
2008	msword	fmt/40	55	78832
2008	pdf	fmt/17	52	1683621
2008	pdf	fmt/18	87	811600
2008	pdf	fmt/19	8	587313
2008	pdf	fmt/20	2	3877433
2008	tiff	fmt/10	4	1162671
2009	jpeg	fmt/42	5	199677
2009	jpeg	fmt/43	414	988417
2009	jpeg	fmt/44	72	450455
2009	jpeg	x-fmt/391	142	2616863
2009	msword	fmt/40	134	46374
2009	pdf	fmt/17	66	961133
2009	pdf	fmt/18	71	1094754
2009	pdf	fmt/19	52	571789
2009	pdf	fmt/20	1	520284
2009	tiff	fmt/10	4	605538
2010	jpeg	fmt/42	4	588862
2010	jpeg	fmt/43	296	2271690

2010	jpeg	fmt/44	125	1175749
2010	jpeg	x-fmt/391	30	1109096
2010	msword	fmt/40	13	81390
2010	pdf	fmt/17	25	311271
2010	pdf	fmt/18	28	1008762
2010	pdf	fmt/19	41	741405
2010	pdf	fmt/20	2	3654008
2010	tiff	fmt/10	2	217495
2011	jpeg	fmt/43	280	2209347
2011	jpeg	fmt/44	54	879990
2011	jpeg	x-fmt/391	20	1744173
2011	msword	fmt/40	19	137143
2011	pdf	fmt/17	19	3246043
2011	pdf	fmt/18	16	1449189
2011	pdf	fmt/19	52	783319
2011	pdf	fmt/20	3	1300193
2011	tiff	fmt/10	8	683133
2012	jpeg	fmt/43	77	1532755
2012	jpeg	fmt/44	76	1455911
2012	jpeg	x-fmt/391	4	2442977
2012	msword	fmt/40	11	33890
2012	pdf	fmt/17	19	286390
2012	pdf	fmt/18	10	539541
2012	pdf	fmt/19	31	667525
2012	pdf	fmt/20	1	4491089

Table A.1: ingests in the EDW repository

List of Figures

3.1	the structure of the database	10
3.2	configuration editor with simple sample configuration	15
3.3	Configuration Editor with a configuration imported from csv data	23
3.4	diagram of the number of simulations grouped by year	24
3.5	diagram of the accumulated storage volume and needed processing time generated by the used tools in comparison to the ingests in percent	24
3.6	diagram of the number of files in the repository over the simulation time	25
3.7	diagram of the total size of the repository over the simulation time	26
3.8	Gathering simulation results from the database using SQL	26
3.9	Using SQL to analyze the deletions in the year 2030	26
4.1	Regression Analysis of file growth when converting between Word97 to Word2003	32
4.2	Regression Analysis of file growth when converting between jpg to tiff	33
5.1	Configuration Editor with sample configuration	38
5.2	number of files per type	38
5.3	migrations with minimal step size	39
5.4	total size after simulation with minimal step size	39
5.5	number of files after simulation with minimal step size	40
5.6	migrations with maximal step size	40
5.7	total size after simulation with maximal step size	41
5.8	migrating jpg to TIFF on expire	41
5.9	migrating jpg to TIFF on ingest	42
6.1	Basic setup: Ingests per file type	44
6.2	total number of files per type family	45
6.3	total size of files per type family	46
6.4	scenario 1 - number of migrations	48
6.5	scenario 2 - number of migrations	48
6.6	scenario 3 - number of migrations	49
6.7	scenario 1 - sum of file size	49
6.8	scenario 2 - sum of file size	50
6.9	scenario 3 - sum of file size	50

Bibliography

- [1] Christoph Becker, Hannes Kulovits, Michael Kraxner, Riccardo Gottardi, Andreas Rauber, and Randolph Welte. Adding quality-awareness to evaluate migration web-services and remote emulation for digital preservation. In Maristella Agosti, Jose Borbinha, Sarantos Kapidakis, Christos Papatheodorou, and Giannis Tsakonas, editors, *Proceedings of the 13th European Conference on Digital Libraries (ECDL 2009)*, volume 5714 of *LNCS*, pages 39–50. Springer, September 2009.
- [2] Christoph Becker and Andreas Rauber. Decision criteria in digital preservation: What to measure and how. *Journal of the American Society for Information Science and Technology (JASIST)*, 2011.
- [3] Tim Brody, Leslie Carr, Jessie Hey, Adrian Brown, and Steve Hitchcock. PRONOM-ROAR: Adding format profiles to a repository registry to inform preservation services. *International Journal of Digital Curation*, 2(2), November 2007.
- [4] Panos Constantopoulos, Martin Doerr, and Meropi Petraki. Reliability modelling for long term digital preservation. In *Proceedings of the 9th DELOS Network of Excellence Thematic Workshop on Digital Repositories: Interoperability and Common Services*, Heraklion, Greece, May 11-13 2005.
- [5] Arturo Crespo. *Archival repositories for digital libraries*. PhD thesis, Stanford University, March 2003.
- [6] Arturo Crespo and Hector Garcia-Molina. Cost-driven design for archival repositories. In *Proceedings of the First ACM/IEEE Joint Conference on Digital Libraries (JCDL'01)*, pages 363–372, Roanoke, Virginia, USA, 2001. ACM Press.
- [7] Katholische Jungschar der Erzdioezese Wien.
<http://wien.jungschar.at/>
visited on January 8th 2013.
- [8] R. Dhanalakshmi and C. Chellappan. File format identification and information extraction. In *Proceedings of the World Congress on Nature Biologically Inspired Computing, 2009. NaBIC 2009*, pages 1497 –1501, Dec. 2009.

- [9] Office for Information Systems Harvard University Library. File information tool set (fits). <http://code.google.com/p/fits/> visited on January 8th 2013.
- [10] Artur Kulmukhametov, Markus Plangg, and Christoph Becker. Automated quality assurance for migration of born-digital images. In *Archiving Conference*, volume 2014, pages 73–78. Society for Imaging Science and Technology, 2014.
- [11] Kyong-Ho Lee, Oliver Slattery, Richang Lu, Xiao Tang, and Victor Mccrary. The State of the Art and Practice in Digital Preservation. *Journal of Research of the National Institute of Standards and Technology*, 107(1):93–106, January 2002.
- [12] A. Lindley, A.N. Jackson, and B. Aitken. A collaborative research environment for digital preservation - the planets testbed. In *Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE), 2010 19th IEEE International Workshop on*, pages 197–202, June 2010.
- [13] Gary McGath. The format registry problem. *code4lib*, 2013. <http://journal.code4lib.org/articles/8029>.
- [14] National Archives of the United Kingdom. Digital record and object identification (droid). <http://digital-preservation.github.com/droid/> visited on January 8th 2013.
- [15] National Archives of the United Kingdom. The technical registry pronom. <http://www.nationalarchives.gov.uk/PRONOM> visited on January 8th 2013.
- [16] Linda Dailey Paulson. Libraries face the challenge of archiving digital material. *Computer*, 43(5):16–19, May 2010.
- [17] Petar Petrov. Clever, crafty content profiling of objects (c3po). <http://ifs.tuwien.ac.at/imp/c3po> visited on January 8th 2013.
- [18] Stefan Schindler. File format analysis. Master’s thesis, Technical University of Vienna, 2014.
- [19] Irfan Skiljan. Irfanview. <http://www.irfanview.net/> visited on January 23th 2013.
- [20] Brent West. The digital dark ages: Preserving history in the era of electronic records. *AIS Electronic Library (AISeL)*, 2014.