SCAPE

Scalable Preservation Environments

Final version of the simulation environment

Authors

Kresimir Duretec (Vienna University of Technology)

September 2014

This work was partially supported by the SCAPE Project. The SCAPE project is co-funded by the European Union under FP7 ICT-2009.4.1 (Grant Agreement number 270137).

Executive Summary

This document accompanies the prototype implementation of the simulation environment. It concisely describes two main parts of the implementation: the simulation engine and the simulation language.

The implementation can be found here: https://github.com/openplanets/scape-simulator

Table of Contents

1 Introduction

Digital repositories are systems designated to hold digital content. Some of them have a special mission to preserve their content for the future generations. The goal of digital preservation is to keep content in an accessible and authentic form over a long term. To achieve this goal different methodologies and standards have been developed: OAIS (CCSDS, 2012), Preservation Planning, Preservation Watch (Becker, Faria, & Duretec, 2014).

One problem within digital repositories is that it is hard to anticipate consequences that might bring actions taken in present or in the future. While this might not be a crucial problem in small scale repositories, repositories storing large amounts of data (terabytes or more divided into a number of different collections) need a good understanding of potential outcomes when (not) applying a certain preservation action. Limited resources (economical, technical and human) are an important factor driving the decision making process. To understand the potential resources required for a certain collection several factors should be considered: new material is continuously ingested into a collection, collections are accessed by the community and preservation actions like migration happen at different points in time. Ingest will result in increasing the size of a collection in terms of number of objects and size in MB. It will also result in a diversification of collection in terms of different formats. This is a direct result of the evolving environment outside of a repository. The evolving environment will also affect the repository through access. The access outcomes will force repository owners to perform certain actions. For example the repository owners could migrate their collection when they notice that it is not accessible any more by the designated community. When they migrate, the new collection will require new resources. There are numerous questions that repository owners could ask. For example, is my planned resource upgrade in the next five years sufficient? If my collection is following certain trends in ingest and access, what is the estimated time for a first migration? When I migrate, will I have enough resources to keep the originals as well or they will need to be deleted?

To answer this kind of questions a tool that can simulate different aspects of a digital repository is required.

Simulation is the imitation of the operation of a real-world process or a system over time (Banks, 1998). Today, it is an important method in solving a number of different problems like analysing performance, validating designs and testing hypotheses. Indeed in some industries it is a required step before taking any major decision.

There are numerous reasons to use simulation:

- the problem is too complex to be solved analytically
- it is not possible or too expensive to perform experiments,
- the simulated system does not exist in a reality so a simulation is the only way to observe its behaviour

The idea of using simulation principles to better analyse digital repositories is not new. In (Crespo & Garcia-Molina, 2000) authors present a simulation tool (ArchSim) capable of evaluating a repository system implementation. The focus is on comparing different options such as disk reliability, error detection and preventive maintenance to estimate the Mean Time to a Failure (MTTF) of a whole repository. Similar work was done in (Christensen, 2005) where a simulation tool was built to evaluate potential design of the Danish web archive. More recent work was conducted during the PrestoPrime project[1]. In (Addis & Jacyno, 2010) authors describe a tool capable of simulating the costs and risks of using IT storage systems for the long-term archiving. The tool is now known as iModel[2] and is maintained by the PrestoCentre[3] organization.

While the main focus of these tools is physical preservation, there is only a limited support for logical preservation.

The goal of this work was to develop a simulation environment with a focus on logical preservation (migrations, format evolutions, ingest and potential collection change over time). Aspects such as resources are also considered.

2 Goals

In this chapter a formal list of goals is specified.

1. **Simulation environment is agnostic to simulation models**
   Providing one simulation model which simulates every important aspect is cumbersome because one model will not be applicable in every possible scenario and there may be too many parameters to define. In order to enable simulation of the different scenarios relevant to digital preservation, a desirable design goal of the simulation environment is to enable users to run simulations based on different models. This will enable the simulation of different aspects such as ingest, resource utilization, migration paths and others. This should enable combining the simulation of different aspects. For example, combining models that simulate ingest and format evolution to get a better understanding of collection evolution over time and possible migrations.
2. **Enable answering "what if" questions**
   A user might be interested in testing different scenarios. For example, are current resources sufficient to keep all the content for next five years, or what are the differences between deleting old files, after migrating them to a new format, and keeping them? To enable this, the simulation environment should allow specifying rules which simulate future needs.
3. **Visualize simulation results**
   Results visualization is an important step in a simulation process. In most cases different static graphs are sufficient.

3 Implementation

The simulation environment is divided into two main sections: the simulation engine and a simulation language. This division separates the concepts of the simulation domain from the concepts of the digital preservation domain. This separation of concerns brings two main advantages. Firstly it removes all the peculiarities of simulation from the end user. Secondly, all the details of a

[1] http://www.prestoprime.org

[2] http://www.prestocentre.org/library/tools/imodel

[3] http://prestocentre.org

simulation model are defined in one single sharable file. This is expected to improve the reproducibility of simulation results due to several reasons. The simulation model is easy to share and can be considered as a documentation of the simulation. Furthermore the simulation model contains all the details which remove any ambiguities often present when the simulation is described only in text.

3.1 Simulation engine

The simulation engine is a simple java application which implements the discrete event simulation model.

In a **discrete-event simulation,** the model state changes only at discrete points in time. A modelled system is executed over time, thus creating a history of the observed system, rather than solving it analytically. This is particularly applicable for digital repositories enabling the observation of future repository evolution based on some assumptions.

In simplified form, the main building blocks for discrete-event simulation models are **entity** and **event**.

An **entity** is a possible unit that can exist in a simulation. During the simulation different entities can appear and disappear. All entities can have attributes. Attribute values at specific time define the state of an entity, so an entity has a state which changes over time.

Each entity attribute in the simulation engine is represented as a key-value pair where the key denotes the unique identifier of a specific attribute belonging to a specific entity and the value represents its current value.

An **event** represents an action carried out by or on an entity during the simulation.

Events in a simulation engine are stored in a queue where they are executed according to their priority (determined by the scheduled time).

3.2 Simulation language

A domain specific language was developed in order to define simulation models. The purpose of the language is to support the definition of a range of simulation models and to hide the simulation details from the end user. The simulation model expressed in a defined language can be used to document the whole simulation process.

The domain model covered by the implemented grammar is shown in Figure 1. The whole simulation is represented by the **Simulation** element which contains basic properties like the start and end date of the simulation and how many iterations should be performed. Start and end dates are defined in the form of month/year. The simulation is scheduled so that one clock tick, from in the simulation engine, is considered to be one month.

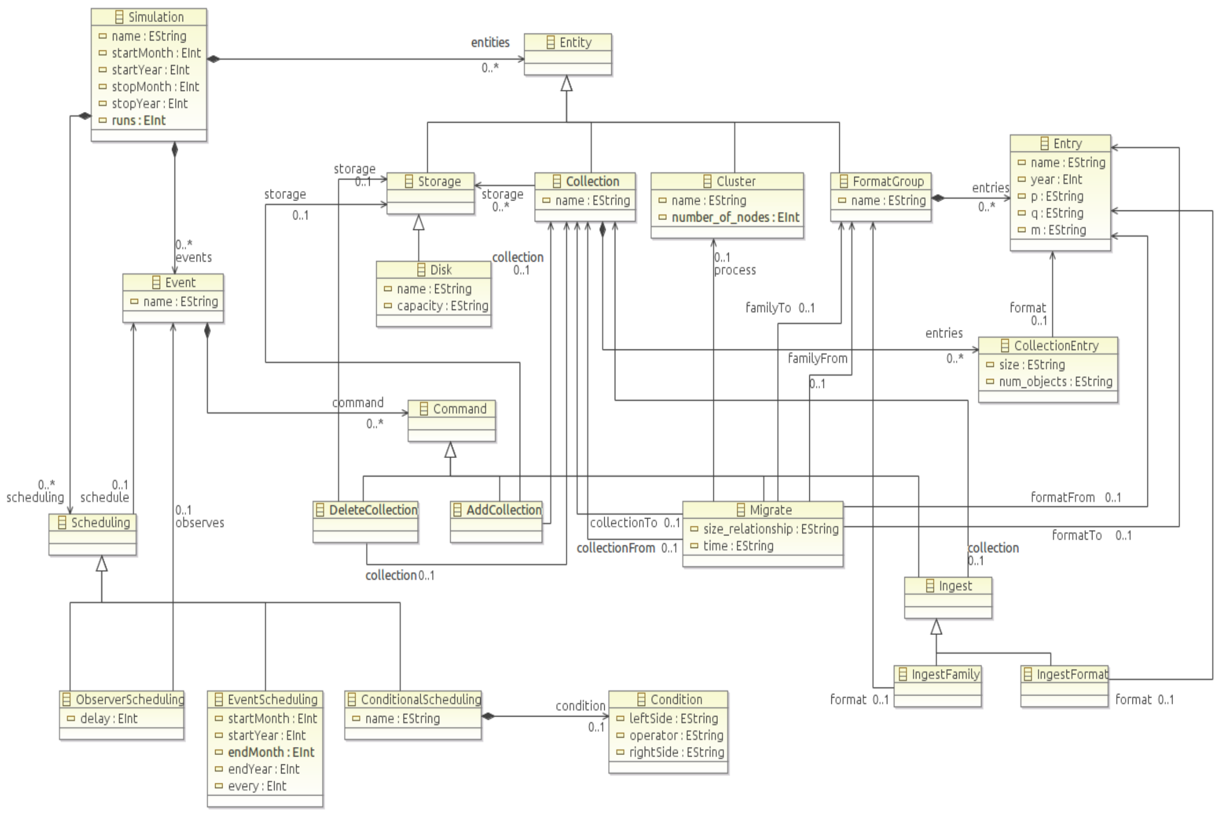

Figure 1 Domain model of the simulation language

Three main parts of the simulation can be distinguished:

- entities
- events
- scheduling of events

Supported entities in the language are: **FormatGroup**, **Collection**, **Storage** and **Cluster**.

Using the model it is possible to define a number of different formats, collections, storage and processing units. The use of these elements is optional, so not all entities need to be considered. For example a simulation model could omit storage aspects.

In the simulation engine entities are represented by their state variables. Events are the elements in a simulation which change these state variables. An arbitrary number of events can be defined within a simulation model. The operations that events can perform are currently limited to the ingest of new elements to a collection, migration of specific format to another format from a specific collection to another, and moving a collection to or from specific storage.

Defined events are executed at specific time slots during a simulation run. These slots are defined by scheduling an event in several ways. Combined with scheduling an event by simply defining its start and end times, it is also possible to define events which are executed after a certain event is executed or when a certain condition is met.

In the next chapter the main elements are described individually with accompanying grammar.

3.3 FormatGroup

The main purpose of the **FormatGroup** is to enable the simulation of the evolution (in terms of percentage use) of different formats by grouping them. The configuration properties are quite simple but still powerful enough to support different scenarios. For example it is possible to simulate evolution of a group of image formats (e.g. GIF, PNG, JPEG, JPEG2000), or the focus can on a single format family simulating multiple format versions (e.g. simulating all the versions of a PDF format).

A **FormatGroup** is defined by a unique name and a list of **Entry** elements which represent the different formats in the group. Each **Entry** has a unique name and several parameters. These parameters denote the release year of the format and the three (p, q, m) parameters of the Bass diffusion model (Bass, 1969). Those parameters denote p – the coefficient of innovation, q –the coefficient of imitation and m – the ultimate market potential.

The grammar (defined in the Xtext language) is

```
FormatGroup:
    'FORMATGROUP' name=ID
    (entries+=Entry)+
    'ENDFORMATGROUP';
Entry:
    '(' name=ID ',' year=INT ',' p=Number ',' q=Number ',' m=Number
')';
```

Each **Entry** will be represented by a percentage in a simulation state. In the case of the PDF format family the percentage of PDF version 1.4 in year 2014 would denote how many people are still using that version of PDF format.

The outcome of this specification is a series of automatically generated **Events** which will change the percentages of given entries according to the Bass curve.

3.4 Storage

The **Storage** entity defines the storage types that can be used to store different **Collection** entities. Currently only one type of storage is supported which is simply denoted as **Disk**. This type is defined by its unique name and capacity. In a simulation state there are two variables representing it: capacity and used capacity.

The grammar (defined in the Xtext language) is

```
Storage:
    Disk;
Disk:
    'DISKS' name=ID 'capacity' '=' capacity=Number 'GB' 'ENDDISKS';
```

3.5 Cluster

A **Cluster** is the only entity type that represents computational resource in the simulation. It is defined by the number of nodes and number of nodes used in the simulation state.

The grammar (defined in the Xtext language) is

```
Cluster:
    'CLUSTER' name=ID 'number_of_nodes' '=' number_of_nodes=INT
    'ENDCLUSTER';
```

3.6 Collection

A **Collection** simulates a collection of digital objects in a repository. In a simulation model it is represented with a unique name and a list of **CollectionEntry** elements. **A CollectionEntry** is an element which denotes homogeneous parts of a collection. It points to one of the **Entry** elements from one of the defined **FormatGroup** elements. Furthermore, it defines the size of that part of a collection and the number of objects. For example, by pointing to PDF version 1.4 in the PDF **FormatGroup** it is possible to denote the initial size and number of objects (which are PDF 1.4s) in some collection.

The grammar (defined in the Xtext language) is

```
Collection:
    'COLLECTION' name=ID
    (entries+=CollectionEntry)*
    ('stored:' (storage+=[Storage] (',' storage+=[Storage])*))*
    'ENDCOLLECTION';

CollectionEntry:
    '(' 'format' '=' format=[Entry|QName] ',' 'size' '=' size=Number
'GB' ',' 'number_of_objects' '=' num_objects=Number ')';
```

Each **CollectionEntry** is represented by the number of objects and its size in the simulation state. The whole **Collection** is also represented by the number of objects and size. These numbers denote the sum of all objects in a collection and their respective size.

A **Collection** can optionally be stored on an arbitrary number of storage elements. This enables simulating redundant backups in a repository.

3.7 Event

An **Event** is an element which changes the state variables of defined entities. Each **Event** is defined by its unique name and a list of commands. Currently the commands supported are: **Ingest , Migrate, DeleteCollection,** and **AddCollection**.

The grammar (defined in the Xtext language) is

```
Event:
    'EVENT' name=ID
    (command+=Command)*
    'ENDEVENT';

Command:
    Ingest | Migrate | DeleteCollection | AddCollection;
```

3.7.1 Ingest

The **Ingest** event enables modelling the ingest process by defining the number of objects and the size of each object to be ingested into a specific **Collection**. The number of objects and size can be expressed as a number or by using one of the pre-defined distributions (**Uniform,** or **Normal**).

There are two types of Ingest event, the first (**IngestFamily)** simulates the ingest of the whole **FormatGroup** into a specific **Collection**. In this case every **Entry** will be ingested into a **Collection** where the number of objects per Entry is calculated according to its current usage percentage defined by the **Entry.** For example if there is 1000 PDF objects to be ingested and PDF version 1.4 is currently at 20% of usage that will result in the ingest of 200 PDF version 1.4 objects.

The second method is ingesting a specific **Entry**.

The grammar (defined in the Xtext language) is

```
Ingest:
    IngestFamily | IngestFormat;

IngestFamily:
    'ingest' 'group' format=[FormatGroup] '(' num_of_objects=Num ','
size=Num 'MB' ')' 'to' collection=[Collection];

IngestFormat:
    'ingest' 'format' format=[Entry|QName] '(' num_of_objects=Num ','
size=Num 'MB' ')' 'to' collection=[Collection];
```

3.7.2 Migrate

Migration is defined by the **Migrate** element. It is possible to migrate a part of the collection or the whole collection to another collection. When defining a **Migrate** element it is possible to specify a single **Entry** that is to be migrated or to specify the whole **FormatGroup**. This is useful when simulating the normalization of a collection. For example migrate all PDF versions to a PDF/A version. The configurability of the environment allows a user to model migrations that don't have much sense, it is up to the user to ensure that the migration is reasonable.

The grammar (defined in the Xtext language) is:

```
Migrate:
    'migrate' 'from' collectionFrom=[Collection] 'group'
familyFrom=[FormatGroup] ('format' formatFrom=[Entry|QName])* 'to'
    collectionTo=[Collection]
    'family' familyTo=[FormatGroup] 'format' formatTo=[Entry|QName]
'with' 'size_relationship' '=' size_relationship=Number
    'time_per_object' '=' time=Number 'ms' 'on' 'cluster'
process=[Cluster];
```

3.7.3 Add or remove a collection

To simulate different redundancies it is possible to add or remove a **Collection** to particular **Storage.** This provides the possibility of covering different scenarios such as decreasing the number of backups for collections that are no longer a priority.

The definition of the command is quite straight forward simply referencing a **Collection** and a **Storage** element.

The grammar (defined in the Xtext language) is:

```
DeleteCollection:
    'delete' collection=[Collection] 'from' storage=[Storage];


AddCollection:
    'store' collection=[Collection] 'to' storage=[Storage];
```

3.8 Scheduling of an event

Three different types of event scheduling are available: scheduling of an **Event** (**EventScheduling**), scheduling an **Event** when another **Event** occurs (**ObserverScheduling**) and scheduling an **Event** when a certain condition is satisfied (**ConditionalScheduling**).

Simple scheduling of an **Event** is carried out by defining a start year and month and an end year and month. Another parameter denotes how often (in terms of months) a specific **Event** should be executed.

ObserverScheduling enables scheduling an **Event** in reaction to some other **Event** that may occur. Its only parameter is the delay which specifies after how many months the **Event** should be executed. For example we might want to delete a specific **Collection** from some specific **Storage** 6 months after the **Collection** has been migrated.

ConditionalScheduling enables scheduling an **Event** when a certain condition is met. For example, we might decide to migrate all the documents held as PDF version 1.1 when the percentage of use drops below 1%.

```
Scheduling:
    EventScheduling | ObserverScheduling | ConditionalScheduling;

EventScheduling:
    'SCHEDULE' schedule=[Event]
    ('start' '=' startMonth=INT '/' startYear=INT )?
    ('end' '=' endMonth=INT '/' endYear=INT )?
    ('every' '=' every=INT )?
    'ENDSCHEDULE';

ObserverScheduling:
    observes=[Event] '=>' schedule=[Event]
    ('delay' '=' delay=INT ';')?;

ConditionalScheduling:
    'CONDITION' name=ID 'when' (condition=Condition) '=>'
schedule=[Event]
    'ENDCONDITION';
```

4 The tool implementation

The tool is implemented as an Eclipse[4] plugin. This supports a rich graphical user interface based on the Eclipse environment. The plugin extends the Eclipse environment in two ways. It provides a textual editor for defining simulation models and a view which displays different simulation results as

[4] https://www.eclipse.org/

charts. The language itself is implemented in the Xtext language workbench[5] which supports definition of new domain specific languages.

More details about the implementation and instructions on how to use it can be found here: https://github.com/openplanets/scape-simulator

5 http://www.eclipse.org/Xtext/

5 Bibliography

Addis, M., & Jacyno, M. (2010). D2.1.2 Tools for modeling and simulating migration-based preservation.

Banks, J. (1998). *Handbook of Simulation: Principles, Methodology, Advances, Applications and Practice.* John Wiley & Sons, Inc.

Bass, F. M. (1969). A New Product Growth for Model Consumer Durables. *Management Science*.

Becker, C., Faria, L., & Duretec, K. (2014). Scalable Decision Support for Digital Preservation. *OCLC Systems & services*.

CCSDS. (2012). *Reference Model for an Open Archival Information System (OAIS).*

Christensen, N. H. (2005). Preserving the bits of the Danish Internet. *5th International Web Archiving Workshop.* Vienna.

Crespo, A., & Garcia-Molina, H. (2000). Modeling Archival Repositories for Digital Libraries. *ECDL Proceedings of the 4th European Conference on Research and Advanced Technology for Digital Libraries* (pp. 190-205). Springer-Verlag.