



Technical Architecture Report


Version 2

Authors

Peter May (British Library), Carl Wilson (Open Planets Foundation)

March 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).

This work is licensed under a CC-BY-SA International License 

Executive Summary

The SCAPE project's focus has been to develop scalable tools, services and infrastructure for the efficient planning and execution of preservation strategies for large-scale, heterogeneous collections of complex digital objects. Through this, digital preservation state-of-the-art is enhanced threefold:

- by developing infrastructure and tools for scalable preservation actions;
- by developing a framework for automated, quality assured preservation workflows;
- by integrating these components with a policy-based preservation planning and watch system.

To achieve these advances SCAPE has developed an ecosystem of tools and services to facilitate the automated production of Preservation Plans, monitoring of knowledge which impacts these plans, and the scalable execution of the Preservation Plan workflows on large content collections.

The SCAPE ecosystem presents a modular infrastructure divided across five main entities, Automated Watch, Automated Planning, Component Management, the Digital Object Repository and the Execution Platform, and supported by additional SCAPE developed tools and services. Interactions between these entities is by way of SCAPE defined interfaces, enabling flexibility in entity implementation and facilitating institutions wishing to amalgamate their existing infrastructure with SCAPE technologies.

In particular, two main approaches to running Preservation Plan workflows are described, facilitating either direct execution or, following further optimisations, execution over the Hadoop parallel infrastructure. Direct invocation provides a high level of integration within the SCAPE ecosystem but is perhaps not well suited for large collections. Conversely, optimising the workflows for execution on Hadoop enhances the computational throughput achievable, making it more suited to very large collections, but reduces integration and requires additional development effort to achieve.

This report presents a technical overview of the SCAPE ecosystem, describing the main services developed and the interfaces they use to interact with each other. It is therefore primarily aimed at those wishing to get an understanding of the SCAPE ecosystem and its architecture, as well as for those who wish to integrate it within their existing infrastructure.

Table of Contents

Deliverable	i
Executive Summary	i
1 Introduction	1
1.1 Outline	2
1.2 Open Source Development	2
1.2.1 Revision Control and Issue Tracking	2
1.2.2 Continuous Integration	2
1.2.3 Packaging and Deployment	3
1.2.4 Contributions to External Open Source Projects	3
1.3 Deliverables Due Post D2.3 Release	3
2 Architecture and Integration Overview	4
2.1 SCAPE Ecosystem	4
2.1.1 Preservation Tools and Workflows	5
2.1.2 Supporting Tools and Services	5
2.2 Architecture	6
3 Automated Watch	8
3.1 Introduction	8
3.2 Functional Overview	8
3.3 Technical Overview	8
3.4 Source Pull Adaptors and Source Push API Clients	10
3.4.1 Sources	10
3.4.2 Functional Overview	10
3.4.3 Technical Overview	11
3.5 Knowledge Base	11
3.5.1 Functional Overview	11
3.5.2 Technical Overview	12
3.6 Watch Requests	12
3.6.1 Functional Overview	12

3.6.2	Technical Overview	13
3.7	Automated Watch Web Client	14
3.7.1	Functional Overview	14
3.7.2	Technical Overview	14
3.8	Public APIs	14
3.8.1	Source Push API	14
3.8.2	Watch Request API	14
3.9	Packaging and Deploying	15
4	Automated Planning	15
4.1	Introduction	15
4.2	Functional Overview	16
4.3	Technical Overview	17
4.3.1	Preservation Plans	17
4.4	Plato	18
4.4.1	Functional Overview	18
4.4.2	Technical Overview	18
4.5	Web-based Analysis Tool	19
4.5.1	Functional Overview	19
4.5.2	Technical Overview	19
4.6	Machine Interpretable Policy Model	19
4.6.1	Functional Overview	19
4.6.2	Technical Overview	19
4.7	Public APIs	19
4.7.1	Notify API	20
4.7.2	External Assessment API	20
4.8	Packaging and Deploying	20
5	Component Management	21
5.1	Introduction	21
5.2	Functional Overview	21
5.3	Technical Overview	22
5.3.1	Components	22
5.3.2	Component Profile Validator	23

5.3.3	Component Catalogue	23
5.3.4	Taverna Workbench	23
5.3.5	SCAPE Toolwrapper	24
5.4	Public APIs	25
5.4.1	Component API	25
5.5	Packaging and Deploying	25
5.5.1	Components	25
5.5.2	Component Tool Dependencies	25
6	Digital Object Repository	27
6.1	Introduction	27
6.2	Functional Overview	27
6.3	Technical Overview	27
6.3.1	Data Management Strategies	28
6.3.2	Digital Object Model	29
6.3.3	DOR Interface Logic	30
6.3.4	Data Layer	30
6.4	Public APIs	30
6.4.1	Data Connector API	30
6.4.2	Plan Management API	31
6.4.3	Report API	31
6.5	Reference Implementations	32
6.5.1	SCAPE Reference Repository	32
6.5.2	Other Open Source Reference Implementations	32
6.5.3	Technology Compatibility Kit	32
6.5.4	SCAPE Digital Object Model Implementation	33
7	Execution Platform	34
7.1	Introduction	34
7.2	Functional Overview	34
7.3	Technical Overview	35
7.4	Direct Preservation Plan Execution	36
7.4.1	Functional Overview	36
7.4.2	Technical Overview	36

7.4.3	Integration Overview	37
7.5	Scalability-Optimised Execution	38
7.5.1	Functional Overview	38
7.5.2	Technical Overview	38
7.5.3	Integration Overview	40
7.5.4	Installation of Tools on a Cluster	41
7.5.5	Extensions to the Hadoop Core Approach	41
7.6	Workflow Parallelisation Strategies	43
7.6.1	Taverna Workflow to MapReduce	43
7.6.2	Tools to MapReduce	44
7.7	Public APIs	45
7.7.1	Job Submission Service API	45
7.8	Packaging and Deploying	45
7.8.1	Platform Releases	46
7.9	Exemplar SCAPE Platform Instances	46
7.9.1	SCAPE Central Instance	46
7.9.2	Hadoop Local Instances	46
7.9.3	Data Centre/Cloud Deployment	47
8	Supporting Tools and Services	49
8.1	Plan Management GUI	49
8.1.1	Functional Overview	49
8.1.2	Technical Overview	49
8.2	Loader Application	50
8.2.1	Functional Overview	50
8.2.2	Technical Overview	50
8.3	C3PO	51
8.3.1	Functional Overview	51
8.3.2	Technical Overview	51
8.4	Repository Simulator	51
8.4.1	Functional Overview	51
8.4.2	Technical Overview	51
9	Conclusion	52



9.1	Remaining Work	52
10	References	54
10.1	SCAPE Deliverables	57
10.2	SCAPE Software References	57
11	Glossary	59

1 Introduction

The SCAPE project's focus has been to develop scalable tools, services and infrastructure for the efficient planning and execution of preservation strategies for large-scale, heterogeneous collections of complex digital objects. Through this, digital preservation state-of-the-art is enhanced in three ways:

- by developing infrastructure and tools for scalable preservation actions;
- by developing a framework for automated, quality assured preservation workflows;
- by integrating these components with a policy-based preservation planning and watch system.

To achieve these advances SCAPE has developed an ecosystem of tools and services to facilitate the automated production of Preservation Plans, monitoring of knowledge which impacts these plans, and the scalable execution of the Preservation Plan workflows on large content collections.

The SCAPE ecosystem presents a modular infrastructure divided across five main entities, Automated Watch, Automated Planning, Component Management, the Digital Object Repository and the Execution Platform. Interactions between these entities is by way of SCAPE defined APIs, enabling flexibility in entity implementation and facilitating institutions wishing to amalgamate their existing infrastructure with SCAPE technologies.

Automated Watch provides a service to monitor repository content and metadata, alongside information from selected user communities and other software systems, in order to provide actionable notifications for Automated Planning.

Automated Planning provides a service used to develop, monitor, and assess Preservation Plans. These plans contain executable workflows constructed from SCAPE Components, which themselves wrap digital preservation tools and services. Components are created and then shared through a catalogue provided by Component Management. Preservation Plans are stored in the Digital Object Repository alongside the Intellectual Entities they are enacted against.

Digital Object Repositories are based on existing OAIS compliant repository, but SCAPE adds a set of additional interfaces to enable Preservation Plan management, uniform Intellectual Entity handling and consistent event reporting with other SCAPE entities, regardless of the underlying repository implementation.

The Execution Platform provides the infrastructure for running Preservation Plan workflows, either directly or following further optimisation for parallel execution. Direct invocation provides a high level of integration within the SCAPE ecosystem but is perhaps not well suited for large collections. Conversely, optimising the workflows for execution on the Hadoop parallel execution infrastructure enhances the computational throughput achievable, making it more suited to very large collections, but reduces integration and requires additional development effort to achieve.

Additional SCAPE developed tools and services support these main entities offering a variety of functionalities. For example, some facilitate creation, management or packaging of SCAPE Components; another helps parallelisation of tool invocations on the scalable Execution Platform. As per the main entities, the use of any of these depends on an institution's goals.

This report presents a technical overview of the SCAPE ecosystem, describing the main services developed and the interfaces they use to interact with each other. It does not attempt to capture all of the background knowledge and experimentation that guided technical decisions, but instead provides references to the appropriate background material.

1.1 Outline

This report starts, in section 2, with an overview of the five main entities in the modular SCAPE ecosystem and how they are intended to interact with each other. Each of these five entities are then described in further detail in sections 3 - 7, which provides functional and technical overviews, along with a description of the main SCAPE interfaces that must be implemented. Finally, section 8 describes the additional SCAPE tools and services that complement these main entities.

1.2 Open Source Development

In general, SCAPE software projects have been developed as open source software projects, released under the terms of the Apache v 2.0 License [1]. The project has endeavoured to leverage publicly available software development services wherever possible, the rationale for this has been to:

- Minimize hosting and maintenance costs.
- Provide open, public access to the entire development process, facilitating distributed development and encouraging community contribution.
- Ensure that the same support for open source communities is provided by continuing to operate these services beyond the project lifetime.

The project has employed the software and services described below, while developing software.

1.2.1 Revision Control and Issue Tracking

The SCAPE project has encouraged the use of Git as a revision control tool for source code. Regardless of whether individual developers use Git, SCAPE policy dictates that every project's source code and documentation is made publicly available on GitHub [3]. GitHub is an online hosting service for Git source code repositories that provides tools to facilitate distributed development:

- A dedicated Issue Tracker for each project (see GitHub Issues [4]).
- A web site for each project (see GitHub Pages [5]).
- A team based security model that controls write access to source code, while allowing anyone to clone a project.
- A mechanism for external contributors to submit enhancements to projects for testing (see GitHub Pull Requests [6]).

SCAPE projects are added to the GitHub account [7] managed by the Open Planets Foundation [8] to ensure they are sustained beyond the lifetime of the project.

1.2.2 Continuous Integration

Continuous Integrations describes the process of automatically compiling software and executing unit tests each time a developer submits a code change. SCAPE has used Travis CI [9], a hosted Continuous Integration service that is integrated with GitHub and free to use for the open source community.

In addition to Travis, SCAPE also uses a Jenkins server [10] and a Sonar [11] server that, together, perform scheduled builds to:

- Perform code quality analysis and publish the results (see public Sonar Page [12]).
- Publish automatically generated documentation where possible (see JavaDoc Page [13]).
- Release software snapshots to the Sonatype Maven Repository [14] (see SCAPE snapshot repo page [15]).

1.2.3 Packaging and Deployment

Given the wide range of operating systems available the project had to make a pragmatic choice as to which to support. After consideration the Ubuntu 12.04 LTS [16] release was chosen because:

- The adoption of a free operating system de-complicates licensing issues associated with cloud based deployments.
- The linux ecosystem provides a large set of open source tools that have digital preservation application.
- The Ubuntu software repositories contain a large selection of pre-packaged software.
- The apt package management system used by Ubuntu is shared by other Debian derivatives.
- Ubuntu 12.04 is a Long Term Support release, guaranteed support for 5 years, up to April 2017.

This choice of OS led to the decision to package SCAPE outputs as apt [17] packages, as wide distribution of the software was envisaged. However this is not the case for all SCAPE software, since services such as PLATO and Scout are expected to be offered as single, hosted instances. In these cases the software products are packaged as Java Web Application Resources (WAR files), with deployment instructions in each project's README file.

Binary or installable SCAPE software packages are downloadable from Bintray [18], a hosted platform for publication and consumption of open source software releases.

1.2.4 Contributions to External Open Source Projects

On occasion SCAPE developers enhanced existing open source software and tools to provide functionality required for the project. Where practical, changes were submitted to the original development branch of a project, encouraging broader use of these enhancements, and ensuring support beyond the project lifetime. One example is SCAPE developed enhancements to the Apache Tika™ project [19].

1.3 Deliverables Due Post D2.3 Release

This report is written to reflect the end state the project, however, it is being written 6 months prior to the project end. As such, there are still milestones and deliverables to achieve (outlined in section 9.1) which, coupled with the research nature of the project, may cause some minor changes to the described architecture. This report should be considered stable but subject to possible amendments within the project's lifetime - even if just to remove this note.

2 Architecture and Integration Overview

The SCAPE infrastructure has been developed as a modular system, shown in Figure 2, enabling a full range of integration opportunities from full deployment of the entire SCAPE ecosystem to just the entities of interest. This section presents an overview of this architecture, describing the top-level architectural entities and the interfaces¹ used to facilitate the different integration scopes. Subsequent sections provide more detailed descriptions of each entity and interface.

2.1 SCAPE Ecosystem

The five main entities within the SCAPE ecosystem are:

- **Digital Object Repository:** a data store for digital objects, metadata and preservation plans.
- **Automated Watch (Scout):** system for monitoring and reasoning over sources of information to detect preservation risks and opportunities.
- **Automated Planning (Plato):** system applying a methodical approach to generating, testing and evaluating preservation plans.
- **Component Management:** creation tool and a catalogue service to create, manage and share SCAPE Components (essentially the building blocks for Preservation Plans).
- **Execution Platform:** infrastructure (software and hardware) for execution preservation workflows.

These five entities work together through a set of interfaces to effectively achieve the interactions² shown in Figure 1.

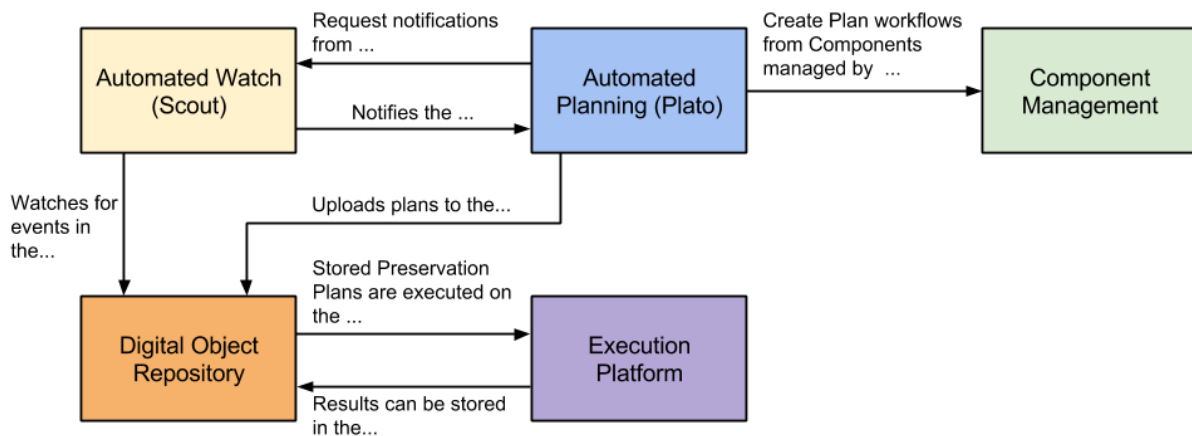


Figure 1: Effective interactions between SCAPE entities

Central to SCAPE is the Digital Object Repository (Section 6) which stores digital objects and associated metadata. Digital objects are made accessible, either directly or by reference, through the repository’s Data Connector API. Preservation Plans are important to an object’s provenance and are also stored within the DOR, accessible through the Plan Management API. Finally, information about events occurring in a repository, such as ingest or planning, are accessible through the Report API.

¹ To aid understanding, interfaces are colour coded to match the component responsible for implementing them. For example, the Report API, Plan Management API and Data Connector APIs are all coloured orange to indicate the Digital Object Repository is responsible for their implementation.

² It should be noted that these are not all the interactions, for example, Scout may watch other sources of information beyond just the Digital Object Repository.

These events are especially useful as a source of information for the Automated Watch system, Scout.

Scout (Section 3) gathers data from sources of information which, through manipulation by appropriate adaptors, it uses to construct and monitor a model view of the world. The Watch Request interface enables Scout's clients to set notification triggers on this model so they will then be notified when that trigger condition has been met, for example, when the number of TIFF files stored in the repository has reached a level requiring preservation plan reassessment.

Preservation Planning defines and assesses actionable workflows to run on specific sets of digital objects in order to aid their preservation (Section 4). These plans also document the decision making procedure used to develop the plan, providing audit trail and provenance information for those digital objects they have been applied to. Plato, the planning tool developed initially by the PLANETS project, guides users through a planning methodology to produce these Preservation Plans using SCAPE Components created and shared through the Component Management service (Section 5). Within SCAPE, the tool has been enhanced to help automate the planning process and produce plans executable on the Execution Platform.

The Execution Platform (Section 7) provides the computational infrastructure for running the Preservation Plans stored in the Digital Object Repository. Scalable execution across large datasets can be achieved through use of the Hadoop parallel execution environment, however this requires development effort to translate the Preservation Plan to a Hadoop application. The Taverna Server platform implementation enables Preservation Plans to be directly executed as they are, producing a high degree of integration within the SCAPE ecosystem, but runtime performance is not optimised for operation over large datasets. Plan execution is initiated through the interface defined by the platform technology used; at a layer of abstraction above the technology, this interface is known as the Job Submission Service.

Datasets themselves are either uploaded to the Execution Platform in advance of plan execution, or accessed directly from the Digital Object Repository or another external storage device. Results from such plan executions, e.g. migrated files, can be stored back in the Digital Object Repository.

2.1.1 Preservation Tools and Workflows

Preservation Plans contain, amongst other information, actionable Taverna workflows comprised of SCAPE Components. These components are themselves smaller Taverna workflows that conform to an agreed input and output interface definition. They typically wrap a preservation action or tool, such as a characterisation or quality assurance tool. The interface definitions facilitate the combination of these components into the larger Preservation Plan workflows.

SCAPE Components are stored in a web based Component Catalogue for reuse and sharing across institutions. Semantic annotations are added to them to provide extra provenance information and can be used to search for Components from the Component Catalogue. Section 5 provides more in depth details about SCAPE Components and the catalogue.

2.1.2 Supporting Tools and Services

The four main entities described above are supported by other tools and services, the main ones being:

- **Taverna Workbench:** User interface for graphically creating, editing and running Taverna workflows. Used within SCAPE to construct Preservation Plan workflows. See Section 5.
- **Component Catalogue:** A web service (myExperiment [20]) where SCAPE Components can be stored, queried and accessed for use in Preservation Plan workflows via a web interface. See

Section 5.

- **Toolwrapper:** A SCAPE tool to simplify the description, invocation and packaging of tools. It can also generate SCAPE Components from tool descriptions. See Section 5.
- **ToMaR (Tool-to-MapReduce execution wrapper):** A SCAPE tool which wraps command line tools for parallel execution on Hadoop. See Section 7.
- **Plan Management GUI:** A SCAPE web based application that interfaces with the Digital Object Repository and the Taverna Server Execution Platform to enable storage, access and execution of Preservation Plans. See Section 8.
- **Loader Application:** A SCAPE tool for facilitating upload of digital objects to the repository. See Section 8.
- **C3PO (Clever, Crafty Content Profiling of Objects):** A content profiling tool that processes metadata extracted from digital objects into an aggregated form suitable for Preservation Planning and Watch analysis. C3PO is also capable of providing representative samples from collections of content. See Section 8.
- **Repository Simulator:** A SCAPE tool and high-level language for describing the state of a repository and simulating the effects of preservation actions upon that repository. See Section 8.

Taverna, the Component Catalogue, the Toolwrapper and ToMaR are described in the chapters they provide key functionality for; for example, Taverna Workbench and the Component Catalogue are both described in Component Management (Section 5). The final four - the Plan Management GUI, the Loader Application, C3PO, and the Repository Simulator - sit, architecturally, outside of the main SCAPE entities, interacting with them through interfaces³. These are therefore described in Supporting Tools and Services (Section 8).

2.2 Architecture

Figure 2 shows an overview of the SCAPE ecosystem, depicting the main entities and the interfaces between them, as well as a number of the supporting tools and services. Each of the main and supporting entities are described in further detail in the subsequent sections.

³ Apart from the prototype Repository Simulator which can be considered a standalone tool.



SCAPE

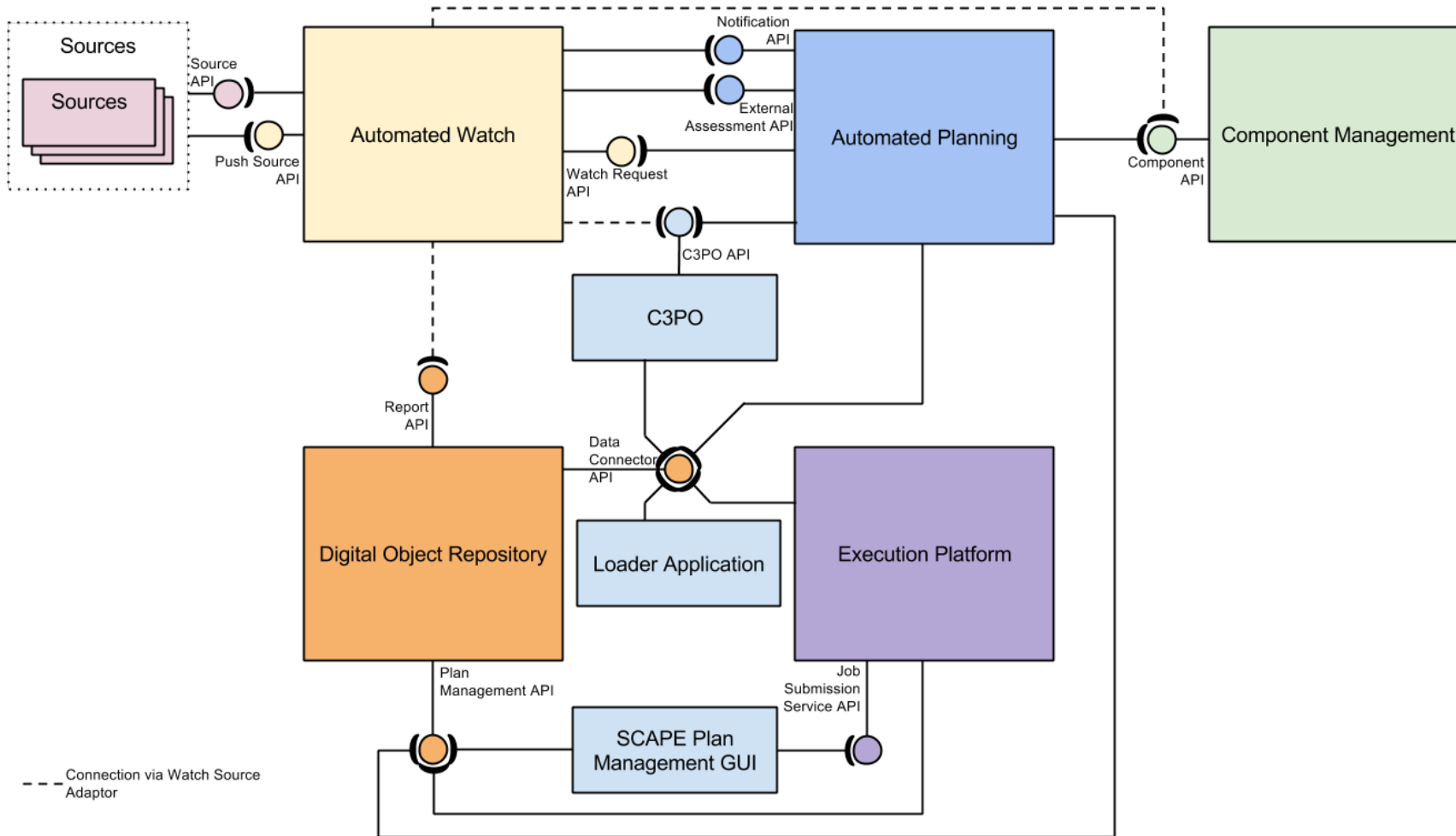


Figure 2: Overview of SCAPE ecosystem

3 Automated Watch

3.1 Introduction

Automated Watch provides support for monitoring the dynamically changing environments that impact the preservation of digital content, in order to detect preservation risks and opportunities. Domain information is obtained from external sources, transformed and added to a Knowledge Base. This Knowledge Base is a source of digital preservation information that can be browsed and queried by a planner - an agent who is interested in the change of the domain's state over time - such as an automated system like the Planning Component (Section 4), or a person.

The Automated Watch functionality is delivered by the Scout Project [SW1], an application that implements the Knowledge Base, supporting services and a web GUI. It is designed to be run as a centralised service, gathering information from multiple sources and providing notifications to its users. SCAPE aims to provide a hosted instance of Scout to encourage its use as a community resource.

This section provides a functional and technical overview of Automated Watch as a whole, before describing the services and APIs that comprise it in more detail.

3.2 Functional Overview

Scout gathers information, from potentially any source, using managed software adaptors to pull in data, or accepting data pushed by human operators or other software systems. Information and sources of interest to digital preservation planners might include:

- Preservation metadata, gathered from digital object repositories or content profiles.
- Format specific information, for example tool support for a particular format, obtained from technical registries.
- Institutional context, e.g. access requirements, taken from preservation policy.

Scout allows users to add new sources through development of new pull adaptors or push API clients, which gather and transform data making it compatible with the Knowledge Base model. Human operators can also use Scout's web GUI to add new data.

The structured data collected by Scout is fed into a linked-data Knowledge Base which is monitored for properties and trends of interest. External agents can ask questions about the gathered information by submitting Watch Requests. A Watch Request is made up of conditions, which are evaluated against pre-defined questions, and triggers, which notify the agent when conditions are met.

Scout also includes an automated monitoring system that addresses the dynamic nature of the information in the Knowledge Base. As new information is gathered, Scout checks for changing data and identifies questions that might be affected by the changes. The answers to these questions are then re-calculated, using the latest data.

3.3 Technical Overview

Scout is composed of sub-components that each provide specific functionality. Together they achieve the overall goal, monitoring the "state of the world" via various sources of information and providing notifications based on changes to that world view. Figure 3 gives an overview of Scout's components and their interaction with one another, the external information sources and the Automated Planning Entity.

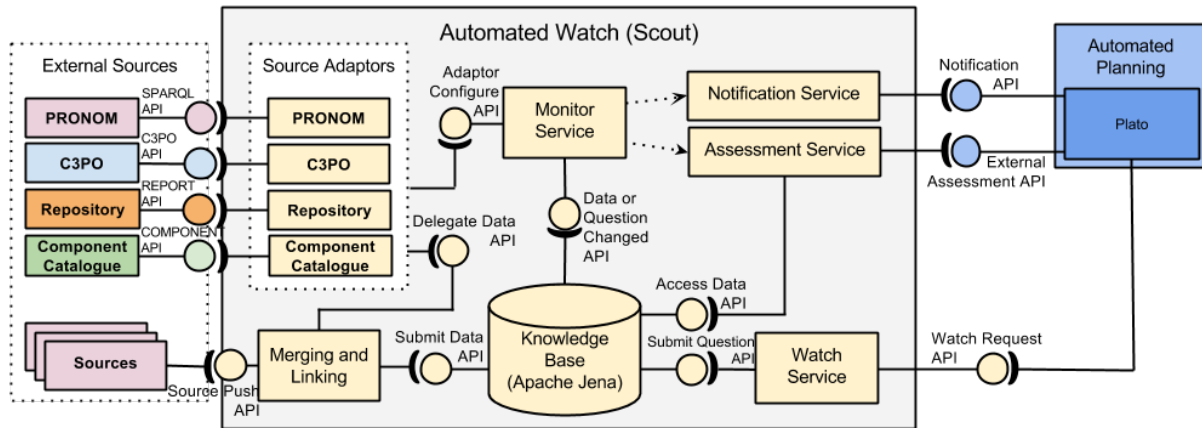


Figure 3: Automated Watch architecture and interaction

Scout is composed of sub-components that each provide specific functionality. Together they achieve the overall goal, monitoring the "state of the world" via various sources of information and providing notifications based on changes to that world view.

Information is collected in two distinct manners:

- Pull Adaptors, which normalize and aggregate data from externally published sources, and are administered by Scout.
- Push API Clients that perform the same normalization and aggregation, but submit data to Scout via its Push API endpoint.

The information gathered or submitted must be compatible with the flexible data model used by Scouts RDF Knowledge Base. The Knowledge Base holds data as typed Entities⁴, which have particular Properties. Properties, in turn, have Property Values which represent a measurement of the property at a particular time. Before adding new data to the Knowledge Base, Scout's Data Merging and Linking service may add relationship information, for example stating the relationship between equivalent entities from different sources. Scout also adds provenance information such as a timestamp and details about the source and adaptor. When Property Values change new measurements are added to the Knowledge Base while previous measurements are also retained. Over time Scout constructs a Knowledge Base that records changing domain information, facilitating repeatability of decision making, and informing future planning.

A planner wishing to query the Knowledge Base about specific measurements of interest submits a Watch Request to Scout. There are two types of Watch Request, synchronous and asynchronous. Synchronous requests contain queries that return a set of measurements gathered at a specific time, returning the answer immediately. Asynchronous requests contain additional conditions and triggers, allowing the planner to query measurements and receive notification when the defined conditions are met. Asynchronous requests do not block the requesting client, but the client must detect and act upon the notification.

The detailed design of Scout is presented in SCAPE deliverable D12.1 - Identification of triggers and preservation Watch component architecture, subcomponents and data model [SD8].

⁴ Note: the term Entity used within Scout has a different meaning to its use throughout the rest of this document. Within Scout it is a way of representing knowledge; within the document it implies an architectural component.

The following sections discuss the various sub-components involved in Scout and their interaction.

3.4 Source Pull Adaptors and Source Push API Clients

3.4.1 Sources

Although Sources are external to Scout, they are described to aid the readers understanding of Scout's sub-components. A Source represents specific aspects of the world that are of interest to digital preservation planners, and provides measurements of properties associated with it. A Source can be internal or external to the project, some key examples showing potential Sources are:

- Format Registries
- SCAPE Components Catalogue (myExperiment)
- Policy models
- Repositories
- Experimental Results
- Content Profiles
- Human Knowledge
- Web Browser snapshots

New Sources can be integrated with Scout by developing a pull adaptor managed by Scout, or an external client that submits data via the Source Push API.

3.4.2 Functional Overview

Source pull adaptors and push API clients gather information from a Source and transform it to conform to the Entity/Property data model used by the Knowledge Base (Section 3.5).

Pull adaptors are software plugins that can be installed and managed through Scout's web GUI (Section 3.7). They collect information from an accessible data point provided by the Source. This data point would typically be a REST API, for example the Digital Object Repository Report API (Section 6.4.3), but data could also be gathered from an RSS feed or parsed from an HTML page. Once an adaptor is registered with Scout, it manages the scheduling of data collection through its Monitor Service and an internal Adaptor Configure interface.

Push API clients are external to Scout and are typically implemented as part of the external Source system, for example as part of a reporting subsystem. A push API client would normally be developed when the Source system neither provides a suitable API or a publicly accessible data point to enable a Pull adaptor to be developed. Scout does not manage or schedule these clients, which only interact with Scout using its Push API (Section 3.8.1).

The Scout preservation watch project has developed three reference pull adaptors and a source API client that provide working examples of Scout plugin/client development and are described below.

PRONOM Adaptor - A Reference Format Registry Adaptor

SCAPE have developed a PRONOM source adaptor capable of querying the PRONOM linked-data SPARQL endpoint and transforming the returned JSON into Scout Entities and Properties which are added to the Knowledge base by way of the Merging and Linking component. The Adaptor code is part of the Scout project [SW2].

C3PO Adaptor - A Reference Content Profile Adaptor

The C3PO adaptor parses the XML content profile data, retrieved from the C3PO REST API, and creates a set Scout Entities and Properties for the profile information of interest. The information is added to the Knowledge Base where a planner may use it, for instance, to monitor trends concerning

particular formats, or digital objects with certain properties. The Adaptor code is part of the Scout project [SW3].

Report API Adaptor - A Reference Repository Adaptor

The Report API adaptor interacts with the Digital Object Repository Report API (Section 6.4.3). Any repository implementing the API exposes a queryable Source of its own event audit data. The adaptor provides Scout the ability to gather and record user interaction data from these repositories. This data might include information about ingest events, user access events, or plan execution events. A planner can then query Scout's Knowledge Base for information regarding trends in average ingest time for a content object, or user access.

Component Catalogue Adaptor - A Reference Tool Registry Adaptor

The Component Catalogue adaptor gathers data from the Component Catalogue (Section 5.3.3), using the Component API (Section 5.4.1). This information is used to assist planners to find tools that meet their planning requirements, or alert them when new tools provide functionality required for preservation activities that was previously unavailable.

Policy Model Push API Client - A Reference Institutional Policy Adaptor

The policy model adaptor is a push client that submits a data file of low level policy elements (expressed as Entities / Property Values I assume) to Scout. Policy data allows the planner to monitor an organisation's repository content checking for policy violations, and receive notifications if they occur.

3.4.3 Technical Overview

Pull adaptors are written as Java classes that implement an adaptor plugin interface that enables Scout to manage all aspects of the adaptors life-cycle. The API means that Pull adaptors can be registered, removed, disabled, enabled, and scheduled using Scout's web GUI.

Push API clients are typically developed as part of the Source system, indeed Scout does not need to be aware of them. They can be implemented in any manner as long as they are capable of calling Scouts Push API, and honouring its contract.

Both adaptors and clients are responsible for transforming data so that it conforms to the Scouts Entity/Property domain model, before submitting it to Scout's Knowledge Base.

3.5 Knowledge Base

3.5.1 Functional Overview

The Knowledge Base stores representation information about the world, employing a model based on Entities and Properties. An Entity is of a particular Entity Type, where the Entity Type has a specific set of Properties associated with it, and each Property is of a specified Data Type.

An Entity gathers together a set of Property Values, which are Property measurements taken at a particular moment in time. For a concrete example consider the Entity Type format. The Properties / Property Value pairs for an Entity of this type that represents the JP2K format would include:

- name = JPEG2000
- version = 1.0
- PUID = x-fmt/392 (where PUID is a PRONOM Unique ID [21]).
- tool support = limited

To ensure that new data submitted to Knowledge Base Scout provides a data processing layer, called the Data Merging and Linking service. This sits between the various Source Adaptors and Clients, and

the Knowledge Base. While adaptors and clients convert their data into Entities and Property Values before presenting them to Scout, Property Values may not be of the correct data type, or different external sources may submit contradictory or incompatible data simply because they are independent of Scout and one another so they are not guaranteed to be correct or consistent. The merging and linking service enriches incoming data by:

- Adding provenance information such as a timestamp, the name and version of the adaptor/client that collected the data, and details of the external source.
- Ensuring that Property Values submitted are consistent with the Entity Type definition, are of the correct data type.
- Resolving inconsistencies between data sources.
- Providing additional cross-references between entities and properties, these additional cross-references enable rich queries of the Knowledge Base.

Timestamping Property Value measurements means that new Entities and their Property measurements can supersede old ones without replacing them. Revisiting the JP2K Entity above, at a later time the tool support for JPEG2000 may increase and a new Entity might indicate "tool support" as "widespread". The previous Entity, indicating limited tool support is kept in the Knowledge Base and the timestamps provide a record of the change in tool support.

A history of all knowledge gathered is kept allowing the Knowledge Base to be queried for past data, enabling repeatability of the decision making process. The Knowledge Base also stores the questions posed by software agents or external users.

3.5.2 Technical Overview

Scouts Knowledge Base is implemented in Java using Apache Jena [22], a Java framework for building Linked Data applications. Jena provides an RDF API for dealing with triples, SPARQL support for RDF queries, a persistent triple store, and an Inference API used to reason over RDF data.

The Data Merging and Linking component is also implemented in Java as part of Scout, and uses Scouts internal Submit Data API to add information to the Knowledge Base.

Scouts Knowledge Base of domain information, built over time, provides a valuable resource for Preservation Planners. Planners wishing to make use of this resource may query, or indeed monitor the Knowledge Base by means of Watch Requests.

3.6 Watch Requests

Watch Requests are the primary means by which a Planner may query or monitor the information held in Scouts Knowledge Base. A Watch Request can be a one off query, returning an instant answer, a Synchronous Request, or a request to monitor some set of information against criteria over time informing the planner if the criteria are met, an Asynchronous request. This sections describes Watch Requests and the Scout services that provide monitoring and notification functionality.

3.6.1 Functional Overview

Planners wishing to ask questions of Scout do so by means of Watch Requests which, in their simplest form, are a composition of questions created by the planner. There are two types of Watch Request available to the Planner:

- **Synchronous Requests** contain only questions that are processed by Scout immediately. The answer is returned to the calling client, the requests are analogous to database queries.
- **Asynchronous Requests** are composed of questions, conditions and triggers, and represent

an instruction by the Planner to monitor some aspect of the Scout domain model and inform the Planner when the criteria defined in the Watch Request are met. Scout monitors these requests against the dynamically changing information in its Knowledge Base, and only notifies the Planner if the conditions of interest to them are met. Indeed it is possible that an Asynchronous Watch Request never notifies the Planner, presuming the conditions of the Request were never met.

To illustrate the difference consider a Planner with an interest in Open Source codec support for the JP2K format. Initially the planner may submit a Synchronous Watch Request that queries the Knowledge Base for such tools. If the returned answer offered no such tools, the planner may then elect to create an Asynchronous Request, instructing Scout to observe its domain model and notify the planner when tools fitting these criteria become available. The observe and notify functionality is provided by Scout services:

- **Monitoring services** that observe the domain model looking for changes.
- An **Assessment service** that re-evaluates a Watch Requests conditions and triggers..
- A **Notification service** that informs external agents when an event, as defined by a particular Watch Request occurs.

Scouts monitoring services check external information Sources for changes. When changes occur the service triggers the re-evaluation of Knowledge Base questions, ensuring the answers reflect the latest state of the world. The answer to the JP2K tool question might be calculated using data gathered from the Component Catalogue via a source adaptor. When new tools are registered with the catalogue the updated information is gathered by the source adaptor. This new information is detected by a Monitor Service which recalculates answers using the latest data and requests the re-evaluation of Watch Requests through Scouts Assessment service.

The Assessment service uses the latest information in the Knowledge Base and the trigger conditions defined by the Watch Requests to evaluate whether an event of interest to the Planner occurred. Scout can perform combinations of Boolean tests when assessing a Requests conditions, but some conditions may be too complex for Scouts preliminary local assessment. In such cases the planner may instruct Scout, via the Watch Request, to use an external assessment service such as an expert system or person, which performs the assessment and returns the result. In either case, once it has evaluated a Watch Requests Triggers the Assessment service will decide whether to notify the planner of an event via the Notification service.

The notification service is responsible for informing external entities of significant events as defined by Watch Requests, and detected by the monitoring and assessment services. An interested party might be a human planner informed by email, or a software agent, typically the Automated Planning component.

3.6.2 Technical Overview

Watch Requests are submitted to Scout via a REST API, the Watch Request API. A Synchronous Watch Request is composed of Questions about Entities, which return lists of Entity names, and Properties, which return lists of Property Values. Questions are asked as SPARQL queries that are performed on Scouts RDF Knowledge Base. A Synchronous Watch Requests blocks the calling client until Scout returns the answer, which may not be instantaneous depending on the complexity of the request.

Asynchronous Watch Requests are also submitted via Scouts Watch Request API. These extend the simple Synchronous Request, holding Conditions and Triggers in addition to Questions. Rather than posing Questions of Scout, Asynchronous Watch Requests instruct Scout to monitor some part of the domain model, and inform the planner it the defined conditions occur in the future. It makes no

sense for these requests to block the calling client as it may well not be known whether the Trigger Conditions will ever be met.

The Monitor sub-component provides a Data or Question Changed interface for requesting notification about changes to underlying data or the Watch Requests themselves. Upon receiving an update, this sub-component will identify which Watch Requests require re-evaluation and trigger this re-evaluation through the Assessment Service.

The Assessment Service requests an External Assessment via an implementation of the External Assessment API. This mechanism allows for the possibility of submitting Watch Requests whose assessment is too complex for Scouts internal Boolean logic. These can still be monitored by Scout but the complex assessment is carried out by an external expert. The Assessment service retrieves data from the Knowledge Base via the Access Data interface.

The Notification Service is simple and extensible, allowing the development of different types of notification, for example email, or HTTP API.

3.7 Automated Watch Web Client

3.7.1 Functional Overview

The Scout application provides a web interface which delivers the following functionality for planners:

- The ability to manually add information to Scouts Knowledge Base.
- Browsing of the knowledge base.
- The Submission of Watch Requests to the Automated Watch system.
- Installation, upgrade and management of Source Adaptors.
- The definition of new Entity Types and Properties for Scouts domain model.

3.7.2 Technical Overview

The Automated Watch Client is a Java Web application, packaged as part of the Automated Watch web application.

3.8 Public APIs

The Automated Watch component implements two public APIs: the Source Push API and the Watch Request API.

3.8.1 Source Push API

The Source Push API provides a RESTful interface for use by external agents wishing to submit data to the Knowledge Base. This is carried out by developing an external client, rather than a Source Adaptor integrated with Scout. Note that push sources are not controlled by the Source Adaptor manager, meaning the scheduling, enabling and disabling of push sources is managed by the source system(s). The API is also used internally by the Automated Watch Client (Section 3.7) to add new information via the web front end. Full details of this API can be found in [22].

3.8.2 Watch Request API

This is a RESTful API used by external software agents wishing to submit Watch Requests to the Automated Watch system. Typical clients include the Automated Watch Client GUI, or the Automated Planning System (Section 4).

Full details about this API can be found in [22].

3.9 Packaging and Deploying

Scout, the Automated Watch system, is a Web Application, with source code available online [SW1]. The system was developed as a Java Maven project and is deployed as a Java Web Application Resource (WAR) on a Java Servlet container such as Tomcat [23].

RESTful services are provided through Jersey [24], an implementation of REST services for Java and part of the GlassFish project [25].

While it is possible for individual organisations to install and support their own Scout instance, the service was designed and developed with a central hosted instance in mind. This provides a shared Knowledge base that will, in time, contain the information accumulated from many different sources, which could be shared between institutions.

4 Automated Planning

4.1 Introduction

Automated Planning has continued the development of the preservation planning methodology and the Plato Planning tool used in the PLANETS project [26]. Preservation planning is the process of assessing digital collections for risk and applying preservation methods and techniques to mitigate those risks. Proper planning should take into consideration all relevant factors including:

- Institutional preservation policies which provide constraints and drivers for operational preservation planning.
- The legal obligation of the institution.
- Institutional requirements, for example covering access to collections and security.
- Institutional constraints, such as limited staff or financial resource, or technical constraints.

Planning is labour intensive when performed manually. Plato offers an automated planning workflow based upon an established planning methodology. SCAPE has developed Plato in order to further automate the planning process by:

- Importing information from external sources, for instance content profiles, or elements of institutional policy.
- Integrating Plato with the Component Catalogue (Section 5) and the Scout Automated Watch (Section 3) service developed as part of SCAPE.
- Incorporating planning functionality in repository systems, so that plans are fed back into repositories for auditing and monitoring.

Upon completion of the planning process Plato produces a Preservation Plan that contains:

- Contextual information including the reason for creating the plan, institutional context, and a description of the collection.
- Evidence describing the preservation strategies evaluated and giving the reason for the decision to choose a particular strategy.
- A series of steps or actions, called a Preservation Action Plan.
- Roles, responsibilities, rules and conditions for execution on the collection.
- An Executable Preservation Plan that can be deployed and run on a Taverna Server instance.

Plato is provided as a single instance hosted online (see [28]).

Automated Planning has also developed a machine interpretable model of actionable, low level preservation policy elements to inform and automate the planning process and provide information for the Scout Knowledge Base (Section 3.5).

This section provides a functional and technical overview of the tools and services that constitute the Automated Planning entity, specifically the Plato Planning tool and the web-based analysis tool. A brief overview of SCAPE's machine interpretable Policy Model is given before describing the APIs presented by the Planning Entity.

4.2 Functional Overview

Automated Planning integrates a number of SCAPE Entities with the Plato planning tool and its planning methodology. A Plato user developing a new preservation plan can start by importing institutional context information from the Policy Model. An XML content profile, from C3PO (Section 8.3) can be used to populate the define samples stage of the planning process. The Plato decision making process requires an objective tree of evaluation criteria that is used to evaluate the available preservation strategies and choose the best alternative. Some of the fine grained decision criteria that populate the objective tree can be derived from the Policy Models actionable control policy elements.

When users start evaluating preservation tools for use in the Preservation Action Plan they are able to search SCAPe's Component Catalogue for wrapped preservation tools that meet the criteria required by the plans objectives. Any candidate Components can be tested on the sample content and evaluated against the decision criteria in the objective tree. The information gathered during this testing is added to the Preservation Plan in order to provide documentary evidence to support the final decisions taken.

An example plan might consider options for migrating a set of TIFF image files to the JP2K format for storage and access efficiency reasons. Plato can be used to evaluate whether the target format meets the desired storage / access requirements. This can be done by querying format registries or by examining the characteristics of converted sample content and comparing them to the criteria in the objective tree created to assess the proposed solution.

If migration is considered as a strategy the user can evaluate preservation tools, wrapped as SCAPE Components, chosen from the Component Catalogue, to be used in a Preservation Action Plan. Again the evidence gathered regarding Component quality and performance is evaluated against the criteria in the objective tree.

Once a SCAPE Component has been selected an Executable Plan can be generated. Such a plan would typically involve the component chosen to carry out the format migration alongside components to assure the quality of the migration. The Executable Plan is wrapped in the Preservation Action Plan, which also specifies the content the action is to be applied to and quality criteria for the migrated objects.

A successful conclusion to the planning process leaves the user with a Preservation Plan that contains a Preservation Action Plan and an Executable Preservation plan. The final Preservation Plan is stored in Plato's database where a user can designate them as private, i.e. inaccessible by other users, or public, meaning they become a resource that can be used by others to inform their own planning. A Preservation Plan can also be deployed to a SCAPE repository, via the Plan Management API (Section 6.4.2), to be scheduled for execution. Plato can also deploy triggers to Scouts Watch Request API, in order to monitor the plan.

Plato also provides the ability to submit Watch Requests (Section 3.6)

4.3 Technical Overview

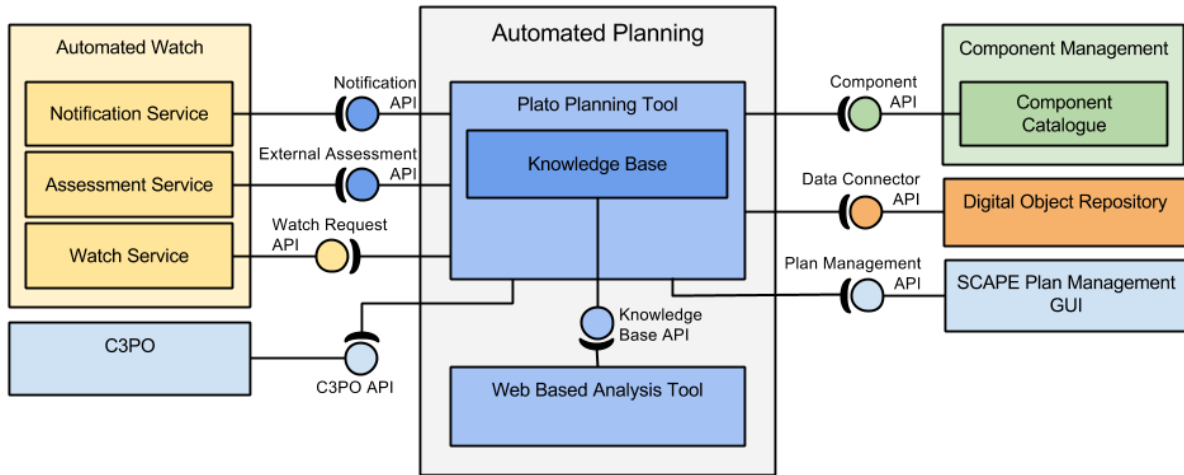


Figure 4: Automated Planning architecture and interactions

Figure 4 shows an overview of The Automated Planning entity and its interactions with other SCAPE entities. Automated Watch comprises:

- A new version of the Plato Planning Tool [27] building upon the existing PLATO tool but using the Automated Watch Component, the Policy Model, Taverna Workflows and content profiles to automate the creation and management of preservation plans.
- A machine interpretable model of preservation policy elements [SW5] (Section 4.6). Modelling preservation policies from the top down as a catalogue of high level policy elements, and from the bottom up as a machine interpretable model of actionable low level policy elements in order to inform and automate the planning process, and provide information to the Automated Watch Knowledge Base.
- An analysis tool for mining the results of previous preservation plans querying past preservation plans and provide decision support to the planning process.

PLATO provides a web GUI that guides the user through a streamlined planning methodology and produces Preservation Plans. The Analysis Tool is part of the same Java EE application providing a web GUI that allows stored Preservation Plans to be analysed for trends.

The machine interpretable policy model is an RDF / OWL [30] model of low level, actionable policies rather than a software component.

4.3.1 Preservation Plans

A Preservation Plan documents the decision process which led to the selection of a preservation action. They can be serialised to XML based on the Plato XML Schema definition [28]. Preservation Plans also contain the chosen preservation which is part of the Preservation Action Plan [29].

The Preservation Action Plan contains all the information necessary to apply the preservation action to the objects in the repository:

- **Objects:** The plan identifies the objects that the preservation action should be applied to. These might be identified by enumeration of object IDs, or an SRU query that selects a subset of the collection. This information can be used to request the content from a Data Connector API implementation.
- **Executable Plan:** This is a Taverna Workflow that conforms to the “Executable Plan”

Component Profile. The Executable Plan applies the preservation action to an object and provides measures used to monitor the quality of the action.

- **Quality Level Descriptions:** These give expected levels of quality and are derived from the decision criteria and their utility functions. They are encoded as Schematron schemas and can be used to validate the quality measures produced by the Executable Plan.

Preservation plans deployed to repositories are stored as Archive Information Packages (AIPs – see Section 6). Provenance information and plan execution details are added to the digital provenance section of the AIP for executed plans.

4.4 Plato

4.4.1 Functional Overview

Plato adds automated support throughout the established planning process. While the manual GUI is still supported for low level plan editing where necessity or user preference dictates, modules have been developed that populate forms with suggested content during each planning step. This has led a lighter planning process with less laborious form filling and a simpler GUI.

The key to a simpler more efficient planning process is the ability to import relevant information from other sources to support or automate the decision making process. For example:

- A Watch Trigger (Section 3.6) provides the planner with information required to start planning (e.g. control policies and concerned content set).
- The Policy Model (Section 4.6) contains information describing the institutional context which can be applied at the first planning step.
- Control policies describe requirements on the objects, which are used to derive decision criteria.
- An XML content profile can be uploaded that completes the Plato "define samples" page.
- Plato is now integrated with the SCAPE Component Catalogue encouraging the use of established or experimental Taverna workflows.

4.4.2 Technical Overview

Plato's business logic has been refactored, making flexible workflow configurations easier to implement. This facilitated the development of the lightweight planning options covered in the Functional Overview.

Plato now uses a variety of APIs and technologies to improve automation of the planning process and integrate preservation planning with repositories:

- Plato can derive decision criteria from control policies.
- Plato uses the Data Connector API to retrieve sample content from the repository, based on samples selected from a C3PO (Section 8.3) content profile.
- Plato uses myExperiments REST API to make semantic queries of the Component Catalogues for annotated Components to test during planning and use in Executable Plans.
- The Plan Management API is used to deploy preservation plans to a SCAPE repository.
- Watch Triggers are derived from decision criteria and their utility functions. These are encoded as SPARQL queries within Watch Requests which are submitted to Scouts Watch Request API. These Watch Requests are used to monitor plans deployed to a repository.

The Notify API and the External Assessment API, both described at the end of this section, will allow Scout to inform Plato when Watch Request conditions have been met, and to request External Assessment from Plato.

4.5 Web-based Analysis Tool

4.5.1 Functional Overview

This web-based tool supports the systematic and repeatable assessment of decision criteria and is fully compatible with the PLATO planning tool [27]. It enables decision makers to share their experiences and in turn build upon knowledge shared by others. Plato holds a database containing preservation plans created by different institutions. Public plans are processed and then presented to the planner along with a number of features facilitating systematic analysis.

4.5.2 Technical Overview

The analysis tool is a separate GUI from the planning tool, and while the indicators offered by the tool will be used for some of the planning automation modules, the GUI itself is not be part of the planning workflow.

4.6 Machine Interpretable Policy Model

4.6.1 Functional Overview

Preservation policies are institutional governance statements that constrain or drive operations for Preservation Planning but may also have other effects outside of operational planning. For Planning and Watch policy elements have been divided into 3 classes:

1. **Guidance Policies:** are strategic, high level policies expressed in natural language and can't be expressed in machine interpretable form as they require human interpretation
2. **Procedural Policies:** model the relation between guidance policies and control policies. They can be represented in a formal model as the relation between guidance and control policies
3. **Control Policies:** are specific and actionable, and can be represented in a semantic model

Only the control policies are guaranteed to be represented in the machine interpretable policy model.

The Policy Model can be used by Plato to automate steps in the planning process by providing contextual information about the institutional environment and decision criteria for evaluating the Preservation Plan itself.

4.6.2 Technical Overview

The machine interpretable policy model provides a source for the Automated Watch system and informs the Automated Planning system. Standard technologies such as RDF / OWL [30] have been used to define the terms that describe and represent Control Policies and support policy reasoning. The policy model has undergone an iterative process of testing and refinement while being used to model the various scenario workflows.

One purpose of the policy catalogue is to guide organisations in creating their own complete policy model. This iterative process was used to improve the process of creating institutional policies by simplifying a complex process.

4.7 Public APIs

The Automated Planning component **must** implement two APIs, the Notify API and the External Assessment API.

4.7.1 Notify API

The Notify API will be used by Automated Watch to inform the Planning Component when significant events occur, as defined by Watch Requests. Examples might be flagging that a plan requires updating, due to a change repository's state, or a new tool provides previously unavailable functionality required by a preservation plan.

4.7.2 External Assessment API

The External Assessment API will be used by Automated Watch to request external assessments more complex than the Boolean conditions checked by Automated Watch's own assessment services. The API will provide access to existing preservation plans, in order to match the simple Questions and their conditions to criteria and evaluate whether the changes result in a reevaluation of alternatives. The basis for the assessment will rely on a utility function as provided by the planning component.

4.8 Packaging and Deploying

The Automated Planning Tool is a JBOSS [31] application packaged as a Java Enterprise Application Resource.

There is a central hosted instance of the Planning Tool [27], the latest release will continue to be hosted here.

Organisations wishing to host their own Plato instance would first require a JBOSS server that has been installed and set up separately on which to host the application (see the project's ReadMe [SW4]).

The Web Analysis module is a separate Web Application Resource file within the Planning Suite EAR that can be deployed with Plato allowing the user to analyse their own plans. It will also be available through the PLATO central instance that hosts a large collection of previous Preservation Plans to analyse.

The semantic model of low-level Control Policies has being developed as a GitHub project [SW5]. The project contains:

- The current version of the policy model ontology.
- Some example properties, criteria, objectives, and scenarios.
- Some experimental queries developed in Java.

5 Component Management

5.1 Introduction

Component Management encompasses several tools and a catalogue service for creating, storing and cross-organisational sharing of SCAPE Components. SCAPE Components provide preservation tools and actions as building blocks for constructing Preservation Plan workflows⁵, for example a Component may encapsulate Apache Tika’s file identification capability, or a migration action such as ImageMagick’s convert. Components can be created using the Taverna Workbench or the SCAPE Toolwrapper and stored in the SCAPE Component Catalogue.

This section presents a functional and technical overview of Component Management, before describing Components in more detail, how they’re generated and how they may be shared and reused between organisations.

5.2 Functional Overview

To create a more integrated ecosystem, Automated Planning needs to be able to efficiently create executable plans using the best available preservation actions. To facilitate this, Automated Watch needs to be able to monitor the preservation actions available, checking for new tools or updates to existing ones. Achieving both Planning and Watch goals is challenging if just considering the tools; SCAPE’s approach provides assistance by encapsulating preservation actions within standardised workflow fragments⁶ with annotations to describe their behaviour. In this way, SCAPE Components provide a consistent way to represent preservation actions so that they are easily discoverable from a catalogue and useable in Preservation Plan workflows. Figure 5 demonstrates this concept of Components (i.e. preservation actions) being selected from a Component Catalogue and incorporated into a Preservation Plan workflow.

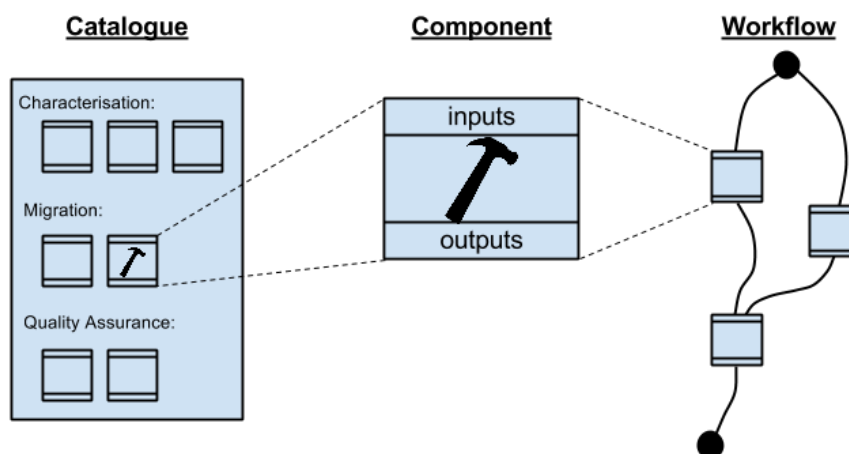


Figure 5: Relationship between tools, Components, Workflows, and the Catalogue

⁵ A workflow is a sequence of operations on some input that execute according to the defined flow to perform some operation.

⁶ Components are described as workflow fragments, however they are fully functional workflows themselves.

Support has been added to Taverna Workbench (and Server) for Components, providing the ability to both create Components and use them to build larger workflows. Alternatively, the SCAPE Toolwrapper can generate Components from SCAPE tool descriptions. These tools are described further in the sections 5.3.4 and 5.3.5 respectively.

Once created, Components can be registered and uploaded to the SCAPE Component Catalogue (Section 5.3.3) which provides validation, storage, semantic search/discovery and access to Components. This catalogue is a centralised service meaning Components registered with it can be shared by the digital preservation community.

5.3 Technical Overview

The representation of Component Management in Figure 2 is an abstract module, which groups the Component Catalogue, the Taverna Workbench, the SCAPE Toolwrapper, and a library for validating Components, as shown in Figure 6.

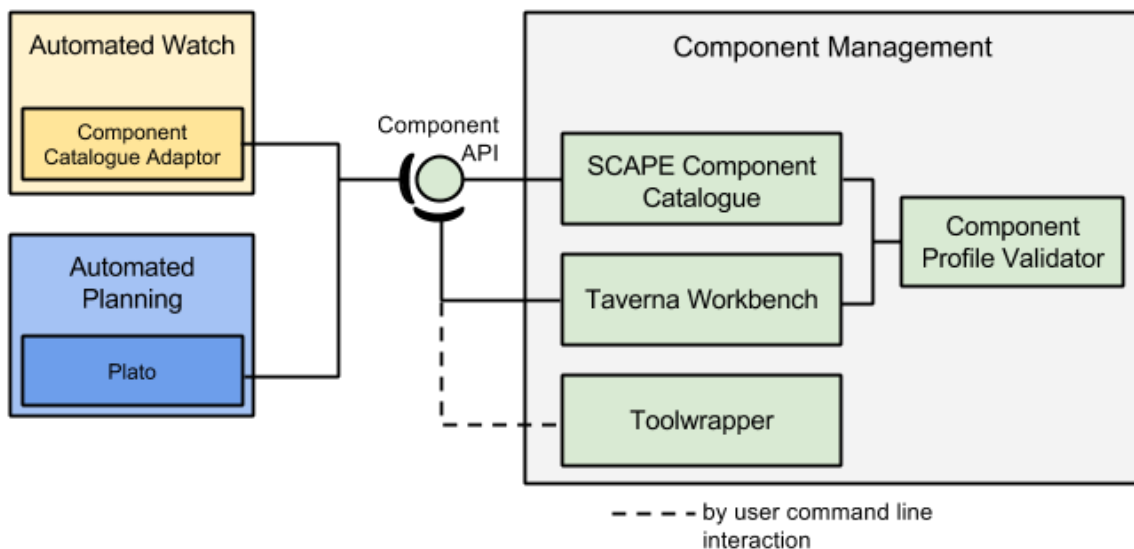


Figure 6: Component Management architecture and interaction

5.3.1 Components

SCAPE Components are XML serialised Taverna workflow fragments that encapsulate a preservation action, but are stored independently of any workflows they are used in. They either wrap command line tools or services with post-processing, are implemented as workflows, or are composed of other Components. A Component conforms to a Profile which enables discoverability and interoperability. Conformance to a Profile can be tested using the Component Profile Validator. Components are organised into Component Families, where each family is a collection of Components sharing the same Component Profile.

Component Profiles

A Component Profile is an XML specification that describes: mandatory and optional input and output ports; required annotations for example Title, Description, Author; and semantic annotations such as values accepted by input ports or measures provided by output ports.

SCAPE have defined a number of these Profiles:

- **Characterisation:** for tools such as Tika or DROID.
- **Migration Action:** for tools like ImageMagick's convert.

- **Object Quality Assurance:** for object based comparisons such as ImageMagick's compare.
- **Property Quality Assurance:** for property comparisons, e.g. using Jpylyzer output to compare image dimensions.
- **Executable Plan:** for the invocation of Executable Plans contained in Preservation Plans.

As an example, on top of common elements such as a title, version and ID a Migration Action Component adds annotations describing the Migration Paths it supports. The Profile also specifies input ports for the Source Object and Parameters used to specify options. Output ports are prescribed for the Target Object and the execution status. A Taverna Activity defines the external tool or service used to perform the migration.

Further details about Component Profiles, with exemplar Components can be found in [SW6].

Component Families

Families are collections of Components that adhere to the same Component Profile. They group components according to structural semantics, e.g. whether they perform migration, characterisation, or quality assurance. This organisation helps facilitate Components use in interactive environments.

Components must be associated with a Component Family which in turn must be associated with a Component Profile.

5.3.2 Component Profile Validator

Components can be checked for conformance against a Profile using the Validator⁷. This is designed as a library routine that is called from both the Component Catalogue and Taverna Workbench.

5.3.3 Component Catalogue

The Component Catalogue, for hosting Components, is the myExperiment [20] web application with an extended REST API to handle the creation and management of Components, Component Profiles and Component Families, including providing discovery and access to them. In particular, it enables semantic searches for Components through SPARQL queries.

The Component Catalogue (via the Component API) is used by:

- Scout, to monitor for new and updated components.
- Plato, to search for and identify suitable Components that meet criteria required by a Preservation Plan's objectives.
- Taverna Workbench, to upload new (or updated) Components or to access Components for incorporating into Preservation Plan workflows.
- SCAPE Toolwrapper, to upload generated Components to the Catalogue.

Further details about the Component API can be found in Section 5.4.1.

5.3.4 Taverna Workbench

Taverna Workbench [32] provides the first of two recommended ways to create and publish Components⁸.

The workbench provides an Eclipse [33] based GUI design tool for a visual, drag-and-drop approach to developing Taverna workflows which can invoke SOAP/WSDL or REST web services, local Java code, local tools, external tools and services via SSH, or other workflows. Through SCAPE, the

⁷ Note: At the time of writing the Component Profile Validator is to be completed.

⁸ The second being the Toolwrapper, which follows.

workbench has been enhanced to support the creation and use of Components. Components can be added to and retrieved from the Catalogue through the workbench, which internally makes use of the Catalogue's Component API (Section 5.4.1).

The workbench also incorporates an execution engine for Taverna workflows that, through the GUI, enacts workflows against supplied inputs whilst presenting a visualisation of execution progress to the user. Upon complete, the outputs of the workflow execution can be examined. Due to the live visualisation of workflow execution, the workbench does not scale up to very large executions. Hence, Taverna Server is provided which is a specialised execution engine without a graphical interface [34].

Further information about Taverna Workbench (and Server) with respect to the SCAPE project can be found in SCAPE deliverable D7.2: Workflow Modelling Environment [SD6].

5.3.5 SCAPE Toolwrapper

Another recommended approach to creating and publishing Components is to use the SCAPE Toolwrapper [SW7].

The Toolwrapper is a Java tool developed to simplify the following tasks:

- **Tool description:** Tools are described through tool-specification files which include semantic elements covering licensing, dependencies, inputs, outputs, and default parameters amongst others.
- **Simplified tool invocation:** Tools are wrapped as command-line bash scripts that standardise their invocation. By this means tools share a consistent method of invocation, so that users don't have to understand the technical specifics of each tool. Wrapped tools also support pipe composition, allowing data streaming even if the original tools lacked such functionality.
- **Generation of artefacts:** The Toolwrapper can generate SCAPE Components conforming to a Component Profile through definitions provided in a component-specification file, as well as Taverna Workflows for demonstrations or prototyping.
- **Upload of Components to the Catalogue:** A utility is included to enable upload of Components to the Component Catalogue.
- **Packaging:** The Toolwrapper can also generate Debian software packages for the tool invocations that simplify their distribution and installation across Debian Linux based execution environments (Section 5.5 for further details about packaging).

Tool and Component Specifications

Tools and their invocations are described in an XML tool-specification document, called a "toolspec"⁹, which conforms to a "toolspec" schema. This document includes the tool's command along with that command's inputs and outputs, for example an ImageMagick file format migration toolspec will specify ImageMagick's convert binary with parameters for the input and output files. Existing toolspecs, defined by SCAPE, can be found in [SW8].

Additional metadata is required for the Toolwrapper to create SCAPE Components. This information is defined in an XML Component-specification document, called a "componentspec". This enables the additional information required in a Component Profile to be specified.

⁹ The same toolspec document can also be used with the SCAPE Tool-to-MapReduce tool, ToMaR, described in section 7.6.2.

Development examples of toolspecs and componentspecs, the toolspec and componentspec schemas¹⁰ and instructions on how to use the Toolwrapper can be found in the Toolwrapper project [SW7].

5.4 Public APIs

The following API provides the interface to the Component Catalogue.

5.4.1 Component API

MyExperiment already implements a number of RESTful interfaces to programmatically interact with it, for example for creating, accessing and deleting workflows [35]. The Component API [36] is an extension of those interfaces that provides for the creation, management and discovery of Components, Component Families and Component Profiles, specifically:

- **Components:** The API allows the creation of Components, the creation of a new version of a Component, the retrieval of the definition of a Component, the deletion of a Component, and the lookup of a Component by either its name (within a Component Family) or its semantic properties (including its annotations) using SPARQL queries.
- **Component Profiles:** The API allows the creation, enumeration and deletion of Component Profiles (Component Families must be associated with a Profile, therefore deletion is only possible if the profile is unused by any Component Families).
- **Component Families:** The API allows the creation of a Component Family, the enumeration of the list of Component Families, and the deletion of a Component Family (which, as Components must be associated with a Family, means the deletion of all Components within that Family).

The API is XML based, with creation requests accepting XML defining the relevant Component, Profile or Family. Similarly, retrieving lists of Components, etc. returns XML documents with references to the requested items.

5.5 Packaging and Deploying

5.5.1 Components

The myExperiment platform provides a single accessible instance of the Component Catalogue for storing, discovering, retrieving, and sharing SCAPE components with no need for institutions to host their own instance.

As discussed previously, Components can be created and uploaded to this Catalogue via Taverna Workbench of the SCAPE Toolwrapper via the Component API. This same API also enables Component discovery and access by Taverna, Scout or Plato.

5.5.2 Component Tool Dependencies

Components typically wrap tool invocations and as such will have a dependency on a supporting tool or service. This means that both Components and their dependencies need to be installed on the computation nodes in the Execution Platform.

The Toolwrapper (Section 5.3.5) can produce Debian packages allowing tool invocations to easily be installed using the Advanced Packaging Toolkit (APT) [17] package management system. APT is a collection of tools that allows an end-user to manage and install software and its dependencies

¹⁰ Schemas are located in the toolwrapper-data/src/main/resources directory of [SW7].

through a single command, or click of a mouse. APT can also be configured to enable automatic updates and removal of software packages. The process for building a Debian package and deploying it to a package repository is fully described in D5.1 [SD2].

The Toolwrapper uses a toolspec to generate a Debian package for the Component, but the specification also holds the Component's dependency information in an installation section which records:

- the identifier for the tool dependency;
- the Operating System(s) that supports the tool, e.g. Debian;
- the package management system used to install the tool, for example dpkg (the underlying package manager for apt);
- an APT software repository that holds the tool, and the name of the tool package.

While SCAPE has concentrated on producing Debian packages, the toolspec syntax is extensible so can be used to record details for other package management systems. This allows the host system to install a Component and its dependencies via its native package management system.

In order to increase the adoption of its results the SCAPE project has worked to make software as easy to install and maintain as possible. Delivering tools as Apt packages will make the process of installing and using Components and tools as simple as possible. This is expected to broaden adoption by the Digital Preservation community, particularly amongst practitioners and other less technical users.

6 Digital Object Repository

6.1 Introduction

A Digital Object Repository (DOR) is an OAIS compliant repository [37]¹¹ that provides a data management solution for storing content and metadata about digital objects, as well as Preservation Plans which form part of the provenance information for digital objects. It interacts with the Execution Platform to carry out these preservation actions on selected content.

This section presents a functional description and technical overview of the architecture of the Digital Object Repository, before describing the three main interfaces a DOR must implement. Finally, open-source reference repository implementations are given.

6.2 Functional Overview

The Digital Object Repository is responsible for helping its users deposit, curate, preserve and access digital objects. It achieves this through packaging digital objects as OAIS Information Packages [37], helping ingest Submission Information Packages (SIPs), archive Archival Information Packages (AIPs), and enabling access to Dissemination Information Packages (DIPs).

Information Packages used within SCAPE comprise of the data to be preserved, such as images or audio/video files, and metadata representing technical, administrative, structural and preservation information related to the data. Essentially this makes the DOR responsible for storing the content and the metadata of a digital object, as well as maintaining the semantic relationship between digital objects.

Preservation Plans (Section 4.3.1) provide provenance information for digital objects about the actions that have taken place on them. Plans must therefore also be stored in the DOR and referenced within the AIP for each digital object they have been applied to. A DOR supports this by providing a Plan Management interface which enables new or updated plans to be uploaded.

Execution of Preservation Plans, over some set of digital objects held in the repository, occurs on the Execution Platform (Section 7). In addition to uploading Plans, the Plan Management interface also provides access to them in order to facilitate their execution, as well as providing access to a Plan's execution state to monitor its execution progress. The digital objects (AIPs) a plan executes on can be retrieved from the DOR (either as binary files or as references to files) by way of the Data Connector interface.

Finally, details about ingest, access and plan execution is important information for Preservation Planning purposes. The SCAPE Report interface enables retrieval of this information from the repository, providing a source of information for both Scout and Plato (Sections 3 and 4 respectively).

6.3 Technical Overview

The actual storage mechanism is repository implementation dependent. The key aspect to enabling a repository's use within SCAPE is conformant implementation of the Digital Object Repository's three defined HTTP APIs and use of the underlying SCAPE data model (Section 6.3.2). Implementation of these APIs will enable any repository to be used within SCAPE, enabling other entities to

¹¹ The 2003 OAIS reference model has been superseded by the 2012 version [39], compliance to which has yet to be determined.

access/reference the data, store or access Preservation Plans as well as monitor their execution status, and finally to gather reporting information on DOR events.

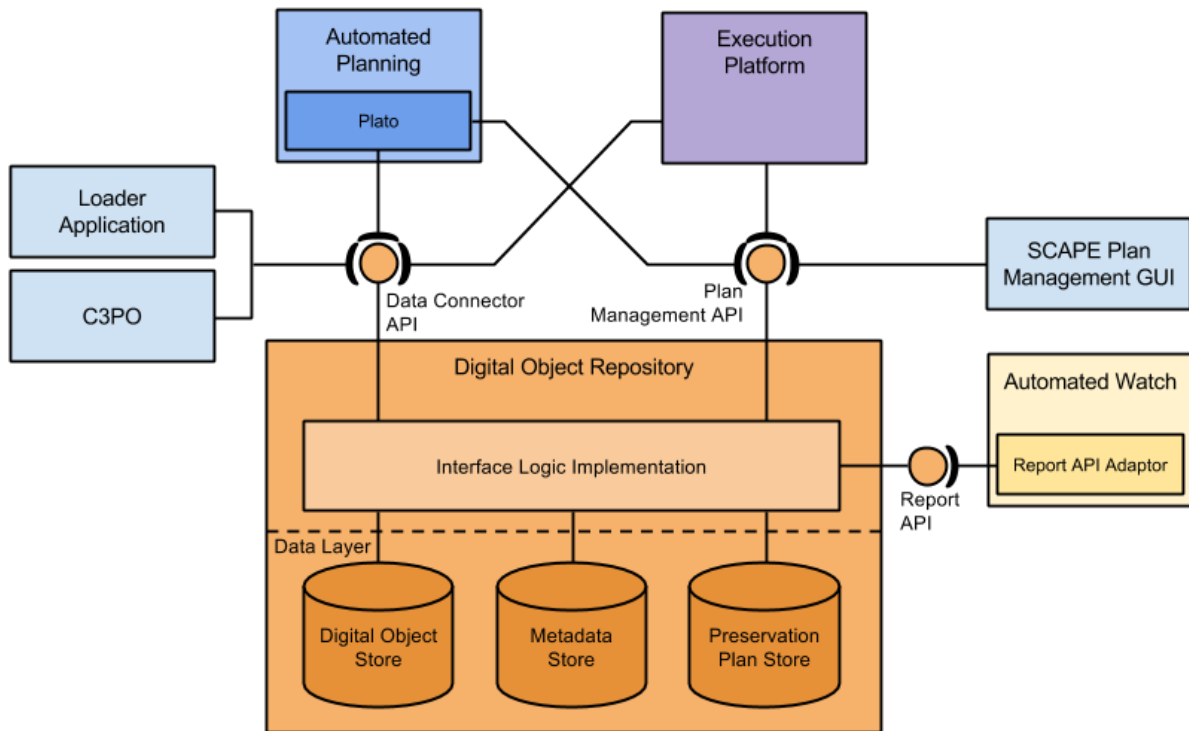


Figure 7: Digital Object Repository architecture and interactions

More specifically, a DOR exposes services to other SCAPE entities through these APIs. Retrieval of information about user events (ingest, access, plan execution) that take place within the DOR, for use by Scout, happens through the Report API (Section 6.4.3). Preservation Plans are submitted, updated, accessed and their execution status monitored through the Plan Management API (Section 6.4.2). These plans operate on digital objects stored in the DOR, therefore the exchange of these objects with the Execution Platform occurs via the Data Connector API (Section 6.4.1), where content may be copied and stored directly to the Execution Platform's storage system, and where outputs (either portions or the entire outcome) of a Preservation Plan's execution can be preserved in the DOR. As such, a DOR employs a defined data exchange model - the Digital Object Model (Section 6.3.2) - to ensure a consistent and well-understood information exchange across the API, thereby ensuring that any repository can be integrated into SCAPE ecosystem despite its underlying object model.

6.3.1 Data Management Strategies

The heterogeneity of collections sizes poses a challenge for good data access performance: while fetching small collections through a HTTP exposed API for workflow execution is a practical approach, the performance to fetch very large collections will be poor. Therefore, the following two storage strategies are available, letting stakeholders choose the most fitting approach to suit their needs:

1. **Managed Content:** When using this strategy, files are only accessible via the Data Connector API; for any operations involving files the "client systems" are required to use the Data Connector API to retrieve or update the binary data from the DOR. This functionality is exposed to the clients by a HTTP endpoint. This strategy, however, is neither very well suited

for large amounts of data, nor for geographically separated storage and computation clusters because of the necessary I/O overhead for data retrieval.

2. **Referenced Content:** When using this strategy, the DOR is integrated with clients by storing files in a file system directly accessible by them, with only file reference URIs passed via the API¹². This enables the Execution Platform to handle large files without having to first move them from the storage to the cluster before computation can take place. It could be viewed as an “on-demand” access of the content direct from its storage location.

The first approach enables preservation actions to be performed on an external Execution Platform (e.g., a Hadoop computation cluster) which is only in use for a limited time, this includes a platform which is external to the organisation. In this case the Execution Platform and storage are physical and/or geographical distinct entities, and so the export of data to the platform is performed via the Data Connector API. This does present a couple of drawbacks, firstly it introduces a bottleneck due to the export of data over a network to the Execution Platform, with a worst-case scenario being to move the entire content and then re-import the results. Secondly, an institution’s policy might prevent exporting the data to a third party, for example an external computation cluster such as provided by Amazon Elastic Compute Cloud web service or Microsoft’s Azure platform.

The second approach is more eligible for running a Hadoop cluster with access to files stored in the associated HDFS file system¹³. The computation cluster and repository (storage) are not distinct instances in this scenario, requiring the cluster hardware to act as both storage and computation platform for the preserved Intellectual Entities.

Batch loading of Intellectual Entities into a DOR will be supported by the SCAPE Loader Application (Section 8.2), which handles validation and error logging, and makes use of a HTTP endpoint for ingesting objects into the repository.

6.3.2 Digital Object Model

Existing repositories already provide their own Digital Object Model for effective storage of digital content and metadata. Such models typically form the basis for OAIS Information Packages (SIPs, AIPs and DIPs) [37] used within those repositories. However, this diversity of models is a hindrance to the Execution Platform in terms of being able to successfully integrate with every repository. Instead, a common DOM is required.

The Digital Object Model is described in detail in [SD7], essentially however, it is based on a combination of a METS XML container [39] and PREMIS preservation metadata [40]. Each Intellectual Entity is represented by one METS file, and each Representation and File will be described by administrative metadata.

Within SCAPE, SIPs, AIPs and DIPs are METS files adhering to the profile defined in [SD7]. This profile defines the mandatory, optional and forbidden elements along with the metadata schemas that should be used for metadata (e.g. descriptive metadata must only use Dublin Core terms, and rights metadata must only use PREMIS rights schema). Each METS document must be assigned a globally resolvable, persistent and unique identifier (recorded in the OBJID attribute), although no specific schema is prescribed.

¹² This includes the “hybrid” situation whereby a computational cluster could also be the DOR, the obvious example being the use of HDFS as the repository storage.

¹³ Although files could also be stored in some form of directly accessible network-attached storage device and referenced for use in the computation cluster.

As an ingestion package, the SIP is slightly more flexible, in terms of the minimum elements that should be present in the METS file, than the AIP or DIP. For example, no <amdSec> element is required in a SIP. Furthermore, no METS identifier is needed assuming that one will be assigned to the AIP by the repository. Both the AIP and DIP however, have the same profile containing technical and digital preservation metadata and, potentially, information about the preservation plan associated with the Intellectual Entity.

6.3.3 DOR Interface Logic

As previously mentioned, the Digital Object Repository provides access to events that occur within the repository, management and access to stored digital objects, and the ability to store and access Preservation Plans. This functionality is achieved through three APIs, described in section 6.4. The DOR Interface Logic is an abstract entity that contains repository-specific implementations of the necessary logic to enable the functionality provided by these three APIs.

Where some form of user interaction may be required, for example, to manage and invoke Preservation Plans, this may also be implemented by a DOR and considered within scope of this abstract entity. However, in the case of this Preservation Plan example, an external GUI such as the SCAPE Plan Management GUI (Section 8.1) could be used instead.

6.3.4 Data Layer

The data layer conceptualises the need for the DOR to store three types of information: Digital Objects, metadata and Preservation Plans. Therefore, although Figure 2 represents these as three separate stores, this does not need to be the case. The actual storage configuration is repository implementation dependent, with the implementation of the three defined SCAPE interfaces being most important from a SCAPE architectural perspective.

The need to store three types of information gives rise to three conceptual stores:

- **Digital Object Store:** This conceptual entity is responsible for storing and making accessible digital object content.
- **Metadata Store:** This conceptual entity is responsible for storing and making accessible metadata information relating to the digital objects stored in the Digital Object Store.
- **Preservation Plan Store:** This conceptual entity is responsible for storing Preservation Plans executable on the SCAPE Execution Platform.

A DOR may provide implementation specific functionality to load data into the repository (outside the scope of SCAPE), alternatively however, an administrator may use the SCAPE Loader Application (which makes use of the Data Connector API) to ingest digital objects (Section 8.2).

6.4 Public APIs

Repository systems **must** implement three APIs, the Data Connector API, the Report API and the Plan Management API, to be used within the SCAPE ecosystem.

6.4.1 Data Connector API

To standardise access to digital objects within the SCAPE ecosystem, the Data Connector API exposes a RESTful API to the user (e.g. a developer or an application) to ingest, update and retrieve such objects. Specifically it aims to address the following use cases:

- *Batch ingest of Intellectual Entities via the SCAPE Loader Application:* see section 8.2.
- *Request for Intellectual Entities by the computation cluster:* Access to the metadata and binary content of Intellectual Entities is provided through the Data Connector API.

- *Update Intellectual Entities with provenance information:* Preservation actions executed on the computation cluster that need to update the provenance metadata of an Intellectual Entity representation can do so via the Data Connector API.

The API relies on the well-defined format of Intellectual Entities as described by the SCAPE DOM [SD7], and specifies a number of HTTP endpoints which provide functionality for: ingesting, updating, searching for and retrieving Intellectual Entities; updating and retrieving metadata records; searching and retrieving Representations of Intellectual Entities or Files; and retrieving the lifecycle status of an Intellectual Entity. Searching of objects is achieved through a SRU (Search/Retrieve via URL [41]) search endpoint.

The Data Connector API is defined fully in [SD7].

6.4.2 Plan Management API

The Plan Management API is a RESTful API for storing, finding and retrieving Preservation Plans in the DOR and for describing their execution state. Its purpose is to integrate the DOR with the preservation planning tool Plato. Specifically, it aims to address the following use cases:

- **Reserve a new Plan Identifier:** A new identifier has to be issued by the DOR before a new Preservation Plan is created. This identifier is incorporated into the plan and used when that plan subsequently gets deployed to the platform or needs to be updated.
- **Deploying new plans in the Repository:** New Preservation Plans created using Plato need to be able to be uploaded into the DOR in order for them to be executed on the Execution Platform.
- **Change existing plans:** Preservation Plan adjustments can be made and stored in the DOR with a Plan versioning system used to ensure appropriate provenance information for digital objects is preserved.
- **Retrieve a specific plan based on some criteria:** In the simplest case the agent has a preservation plan identifier and wants to fetch this plan, for example, for manipulation using Plato. In more complex cases, the agent might want to search for plans based on some plan properties such as a description.
- **Get plan execution and lifecycle state information:** Provides a client with operational (e.g. enabled/disabled) and execution status information about a preservation plan.

Preservation Plans can be searched for, based on their significant properties, through a SRU (Search/Retrieve via URL [41]) endpoint. The queries themselves are represented using Contextual Query Language (CQL) [42].

The Plan Management API is fully defined in [43].

6.4.3 Report API

Scout (Automated Watch, section 3) monitors DORs, amongst other sources, for information about their contents and the events that take place on them. The Report API provides a unified way for Scout to receive information about user interaction events happening within any DOR, relieving Scout from having to define a tailored connection to each different DOR.

Specifically, the user events reported by the API relate to the Ingest, Access and Planning OAI functional entities [37], and describe events surrounding the start and finish of ingest, viewing and download of descriptive metadata or a representation, and details about what plans were executed.

The Report API uses the OAI-PMH protocol [44] which enables a client, i.e. Scout, to retrieve a list of all the events a repository is able to report, to retrieve a single event, all events of a specified type or all events that occurred within a specified time period. The API specification does introduce a

restriction (an OAI-PMH *Record* can only belong to one *Set*) which implementers must adhere to. Events are exposed in XML using PREMIS schema, however, in order to adhere to the OAI-PMH specification, any API implementation must also support event information distributed in Dublin Core format [45].

The Report API and associated events are defined in [46].

6.5 Reference Implementations

6.5.1 SCAPE Reference Repository

The SCAPE DOR reference implementation is based on Fedora 4, which is an open source repository system currently under alpha release (currently alpha release 3) [47]. It provides insight and guidance on how to implement the three main APIs required by a DOR, as well as demonstrates the data structure (SCAPE Digital Object Model) support needed by DORs in order to facilitate ingest and access of Intellectual Entities.

The SCAPE APIs implemented for Fedora 4 are:

- Data Connector API implementation [SW9].
- Plan Management API implementation [SW10].
- Report API implementation - *currently waiting for OAI-PMH support in Fedora 4*

A ready-to-use WAR file containing a pre-beta version of Fedora 4 along with the implemented SCAPE APIs is available for download - see the installation and usage instructions [SW9].

6.5.2 Other Open Source Reference Implementations

API implementations for a few other repositories have also been completed, and provide additional guidance on implementing the APIs across different repository types.

RODA

RODA [48] is another open source repository for ingesting, managing and providing access to digital content, built upon Fedora Commons 2 repository.

All three SCAPE APIs are implemented for RODA:

- Data Connector API implementation - see subfolder in [SW11].
- Plan Management API implementation - see subfolder in [SW11].
- Report API implementation - see subfolder in [SW11].

A SCAPE specific version of RODA with the implemented SCAPE APIs is available [SW12].

DOMS

Another open source repository, the Document Object Management System is based on Fedora Commons 3 [49].

All three SCAPE APIs are planned to be implemented, with the Data Connector API currently under development (although this will not support Bitstreams).

6.5.3 Technology Compatibility Kit

To aid developers in creating and testing their implementation of the Data Connector API, a Technology Compatibility Kit (TCK) has been developed which mocks a SCAPE Data Connector API, exposing its HTTP endpoints [SW13]. The TCK also contains a suite of tests to act as a client to the Data Connector to test creation, retrieval and updating of objects within a repository.



6.5.4 SCAPE Digital Object Model Implementation

A Java implementation of the SCAPE Digital Object Model has been developed to help developers serialize and deserialize Intellectual Entities to and from METS representations [SW14].

7 Execution Platform

7.1 Introduction

The Execution Platform (EP) provides the computational infrastructure to execute workflows¹⁴ of preservation actions over a set of digital objects accessed from the DOR. SCAPE has implemented two approaches to achieve this, the choice of which to use depends a lot on the desired level of scalable execution, as well as on the level of technical effort and expertise an institution has available.

The first approach enables execution of Taverna Workflows contained in the Preservation Plans in an execution environment not suited for operation over large data sets - a non-scalable execution environment. This integrates the EP, DOR and Planning and Watch services, with minimal technical effort. However, this approach is not optimised for scalable execution.

The second approach relies on this extra development effort to produce versions of the SCAPE Components and the associated workflows optimised for execution on a scalable execution environment.

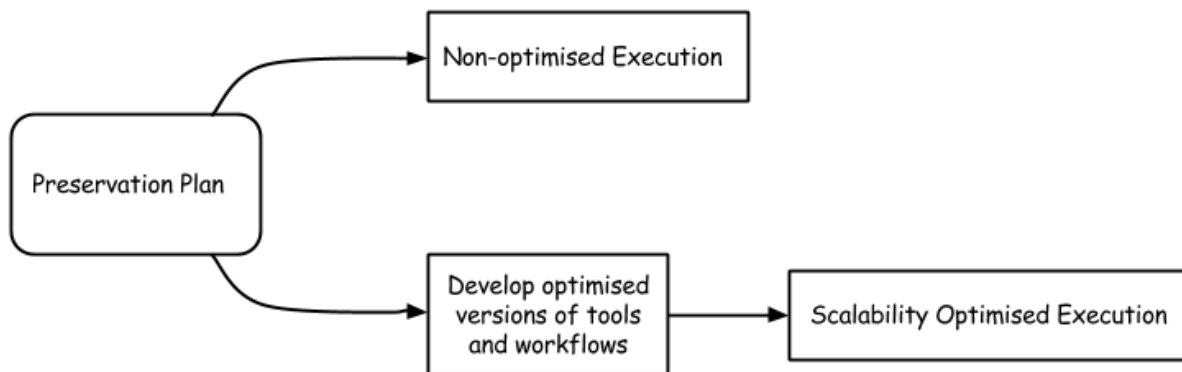


Figure 8: The two pathways for executing a Preservation Plan with SCAPE

A functional and technical overview of the Execution Platform is presented next, providing grounding for the subsequent sections which discuss the two execution approaches in further detail. These later sections indicate the specific technologies chosen and, where necessary, adapted by SCAPE. Following this, parallelisation approaches worked on within SCAPE are described. These attempt to bridge the gap between the non-optimised and the scalability optimised approaches.

7.2 Functional Overview

Ultimately, the Execution Platform provides the computational resources to execute preservation actions. SCAPE is interested in the scalable execution of such actions, primarily to enable fast and efficient processing of large volumes of content - the second approach described above. However, it is recognised that this approach requires specialist development skills to optimise tools and workflows for scalable execution. The first approach enables a pathway to executing Preservation

¹⁴ In this section, a workflow is intended to mean one or more preservation actions applied in some sequence, whether explicitly defined as a Taverna workflow or simply as an application of a sequence of actions by a user. If only a Taverna workflow is intended, this will be explicitly stated as such.

Plans directly, sacrificing scalability-optimised execution in favour of providing an integrated approach from planning a workflow through to executing it.

The first approach:

- Enables software agents and human operators to execute Preservation Plans constructed of non-optimised tool invocations.

The second approach:

- Enhances computational throughput based on the use of clusters of computational nodes (scaling out), rather than expensive server machines (scaling up).
- Executes and monitors the parallelised execution of preservation actions and workflows across the cluster.
- Is capable of distributing data storage across this cluster so as to reduce the effect of network traffic on computation time.
- Supports the coordinated and parallel execution of existing preservation tools and workflows (albeit with appropriate optimisation development).

Makes results from workflows and experiments available via the cluster’s storage layer; these can be exported to the DOR via the Data Connector API (Section 6.4.1).

7.3 Technical Overview

The actual technology employed for the Execution Platform depends on the execution approach chosen, however both pathways follow the same abstraction layers presented in Figure 9. Each layer in this abstracted view is a placeholder representing functionality that is to be fulfilled by a specific implementation. The functionality is described in this section; sections 7.4 and 7.5 describe the specific technologies employed at each layer for the two execution approaches described above.

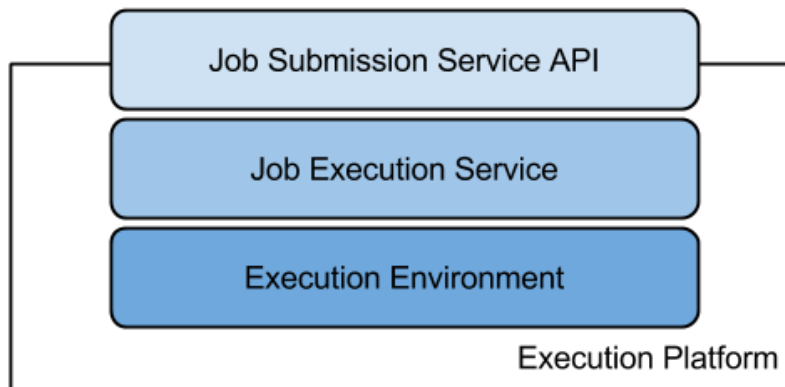


Figure 9: Abstract layers of the Execution Platform

Job Submission Service API

The Job Submission Service (JSS) API provides the entry point to the Execution Platform, implementing a remotely accessible interface to enable a user or client application to schedule and execute workflows (jobs) on the Execution Environment. The interface depends on the underlying JES and Execution Platform, but typical examples would be the Hadoop API provided over a SSH connection, or the Apache Oozie REST API [51] or Taverna-Server REST API [69] over HTTP.

Job Execution Service

The Job Execution Service (JES) functions as a job scheduler, allocating computing tasks amongst the available hardware resources available within the execution platform. For SCAPE, this particularly

focuses on the management and scheduling of workflows. Typical examples of the JES would be Taverna-Server [34] or Hadoop [52].

Execution Environment

The Execution Environment provides the physical infrastructure to perform computation. It is the necessary software and hardware that the Job Execution Service will allocate computation tasks to. Examples would be the software and hardware to run and execute the Taverna-Server, or the computational nodes provided for a Hadoop cluster.

7.4 Direct Preservation Plan Execution

7.4.1 Functional Overview

Preservation Plans incorporate an actionable Taverna workflow constructed from SCAPE Components (Section 5.3.1). SCAPE has enhanced the Taverna-Server implementation to enable these Taverna workflows to be executed, instigated by a client such as the Plan Management GUI (Section 8.1). This approach provides an integrated means to plan, store and execute Preservation Plans.

7.4.2 Technical Overview

Figure 10 depicts the technologies involved in the direct Preservation Plan Execution approach along with the input directed to the API (i.e., a Preservation Action Plan).

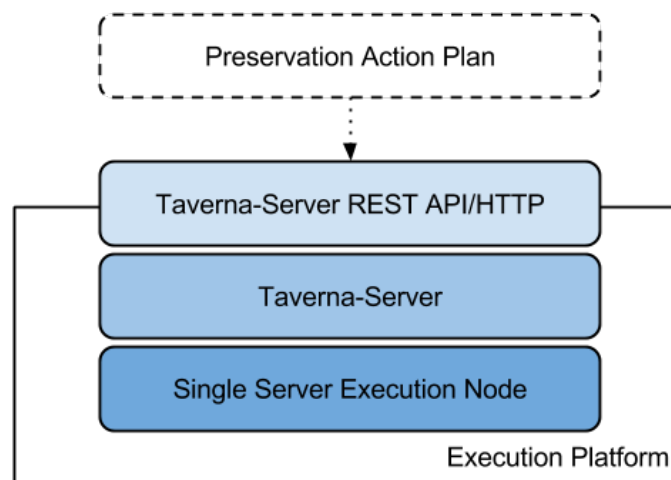


Figure 10: Technologies for Direct Preservation Plan Execution

Taverna Server 2.5.3 [34] is an execution engine that takes user-supplied Taverna workflows and enacts them against supplied inputs, yielding the outputs of the workflow and providing descriptions of the activity that took place during the execution. In terms of SCAPE however, the user is supplying a Preservation Action Plan - an object containing a collection of digital objects, a Taverna workflow to apply to each digital object, and (optionally) a definition of how to assess successful output from the workflow - rather than a direct Taverna Workflow, and so the Taverna Server has been enhanced to enable this.

Job Submission Service API

Taverna Server 2.5.3 interfaces have been enhanced with an additional interface (the “SCAPE Execution Service” interface in [SD6]) allowing Taverna Server to directly accept and execute SCAPE Preservation Action Plans. This REST API provides the following abilities:

- Submit new Preservation Action Plan for enactment
- Retrieve details of current Preservation Action Plan enactment jobs
- Retrieve enactment information about a specific Preservation Action Plan
- Destroy (terminate and remove) a Preservation Action Plan enactment
- Retrieve Preservation Action Plan completion notification address
- Set Preservation Action Plan completion notification address

Full details about the API can be found in [SD6]. Source code for a build of Taverna Server with an implementation of this API is available [SW15].

Job Execution Service

Because the workflow in a Preservation Action Plan only describes the transformations and measurements to be applied to a single digital object, Taverna Server has been enhanced to support the “SCAPE Execution Service” RESTful interface by transforming a supplied Preservation Action Plan into a fully enactable Taverna workflow. Specifically, the enactment workflow has the Preservation Action Plan workflow as a sub-workflow of itself, and incorporates in the iteration over the entire specified collection of digital objects, the access to the repository holding the digital objects, and the preparation of a report describing whether each of the transformations was successful.

This enactment workflow is then fed into the execution engine inside Taverna Server and the Plan Management Service is notified, via the Plan Management API (Section 6.4.2), that the Plan workflow is being executed. Once execution is complete, another notification is sent to the Plan Management Service with the results.

Precise details about the PAP transformation into an enactment workflow can be found in [SD6].

Execution Environment

Taverna Server is a Taverna workflow execution engine, and is not intrinsically designed to manage clusters of computation nodes¹⁵. It therefore operates over shared memory architectures, i.e. a typical PC or Server setup. Software wise, it has the following requirements:

- Unix-like operating system (e.g., Linux, OSX). Running Linux inside a virtual machine works. Running directly on Windows is not supported.
- Java 7 (or later) with the Java Cryptography Extension installed.
- Tomcat 6 (or recent version).

Further details can be found in the associated Taverna Server readme [SW15].

7.4.3 Integration Overview

Figure 11 shows the intended integration between the Digital Object Repository and the Taverna Server, indicating how Preservation Plans stored in the repository can be executed.

¹⁵ A Taverna Workflow could orchestrate Hadoop jobs, however in this approach it is Hadoop which controls and manages distribution and execution of the MapReduce application across the cluster along with the data.

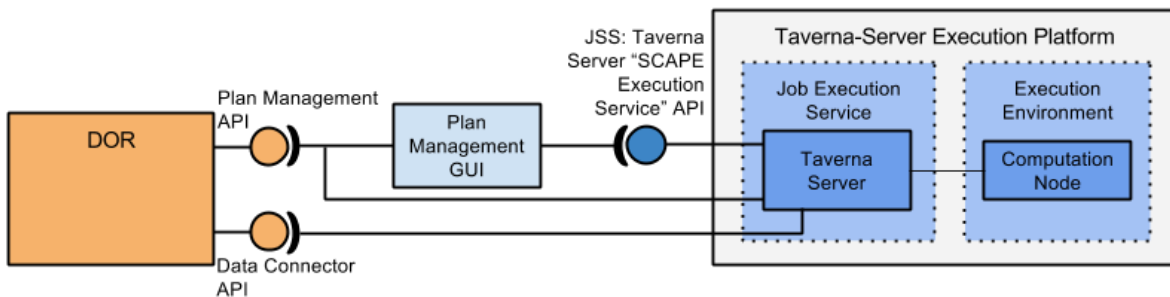


Figure 11: Taverna-Server integration with the Digital Object Repository

Specifically, a user can interact with the Plan Management GUI to identify Preservation Plans of interest, and through this GUI, initiate their execution on the Taverna Server. SCAPE enhancements to Taverna Server's support such execution by manipulation of the Preservation Plan's underlying Taverna workflow, as described in [SD6]. Taverna Server makes the relevant calls back to the DOR's Plan Management API to set the execution status of a Plan, which can subsequently be displayed to the user via the Plan Management GUI. Equally the necessary data can be retrieved from the DOR via the Data Connector API.

7.5 Scalability-Optimised Execution

7.5.1 Functional Overview

SCAPE targets scalable execution of Preservation Plans through the use of a parallel execution system (Hadoop [52]) to enhance computational throughput by scaling out the number of computational nodes processing preservation actions on digital objects at the same time.

It should be recognised, however, that optimised parallel execution is not as simple as executing a Taverna workflow across each node in the cluster. Whilst this is indeed technically possible, experimentation has shown that Java workflows specifically written for Hadoop have better performance [53]. The effort required to produce these optimised workflows depends a lot on the use case it implements and the preservation action tools used. To help assist with this effort, SCAPE have investigated the use of existing Hadoop related workflow projects, such as Apache Oozie [54], or Apache PIG [55], to help define workflows using higher level languages (Section 7.5.5). The project has also researched and developed parallelisation strategies for translating Taverna workflows to MapReduce jobs (PPL-translator: prototype direct translation tool; and translation to Apache Pig approaches), and for enabling command-line tool invocation across Hadoop (ToMaR: Tool-to-MapReduce wrapper). These parallelisation strategies are discussed in section 7.6; first however, the underlying Hadoop based Execution Platform is described.

7.5.2 Technical Overview

Figure 12 depicts the technologies involved in the scalability-optimised execution approach, with submission of a MapReduce Applications to the Hadoop Core stack.

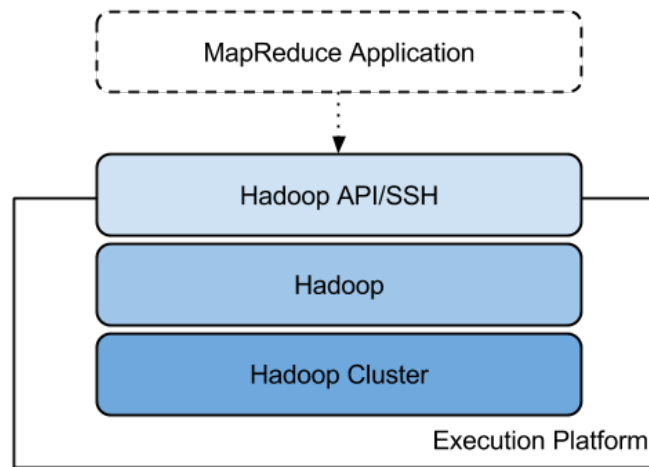


Figure 12: Technologies for Parallel Execution

Apache Hadoop primarily consists of two main sub-projects: MapReduce and the Hadoop Distributed File System (HDFS).

MapReduce is a framework for parallel processing of large data sets in a scalable manner across a large number of computers, or *nodes*. It is divided into a *Map* step, where the input dataset is divided and shared out amongst worker nodes which each compute an answer to part of the problem, and a *Reduce* step which collects and combines all the partial-answers into one¹⁶. Hadoop manages MapReduce jobs, aiming to ensure computation occurs on the same node that data is stored, or failing that, on as close a node as possible to minimise network latency issues.

Data storage is managed by HDFS, a Java based, distributed file system that provides reliable data storage across a cluster of commodity hardware. Files are split into one or more blocks, where each block is usually a multiple of 64MB, and stored across multiple data nodes. Replication of individual blocks across multiple nodes achieves reliability of the data. Importantly, data is stored on the same nodes that perform the computation¹⁷, thereby reducing the amount of network data traffic and boosting performance.

Job Submission Service API

Hadoop provides a script for performing all hadoop commands, from initiating a MapReduce application to manipulating data within HDFS¹⁸. These commands are used via an SSH connection to Hadoop from a remote client.

Job Execution Service

The Apache Hadoop project provides the main framework for scalable storage and computation, making it a good platform, given its approach to bringing parallelised computation to the data¹⁹, for processing of preservation workflows on large datasets. However, the Hadoop ecosystem includes

¹⁶ Further details about MapReduce can be found in a MapReduce tutorial [57].

¹⁷ Despite the data being replicated across multiple nodes, the computation performed on that data, in general, only happens on one node. However, Hadoop has fault tolerance built in and can a) restart failed tasks on another node, or b) spawn a second instance of a slow running task on another node, and use the result from the first to finish - known as *speculative execution*.

¹⁸ Run the script is bin/hadoop to get more details about options.

¹⁹ Rather than taking the data to the processing.

other related technologies which are also of interest to the project, such as Apache Pig [55] and Apache Oozie [54]. Through Apache directly, these tools can all be separately downloaded and installed, however some commercial organisations, such as Cloudera and Hortonworks, provide distribution bundles encompassing Apache Hadoop plus related technologies.

Specifically, SCAPE targets the distribution provided by Cloudera, called CDH (Cloudera Distribution including Hadoop) [57]. When development started on the project, the current version was CDH3 (update 2²⁰), which is based on Hadoop version 0.20.2 and MapReduce version 1 (MRv1). Although CDH3 has now come to the end of maintenance, during the main development lifetime of the SCAPE project, Cloudera kept this distribution up to date with patches solving various bugs and security/performance improvements that are available before a major Apache release. CDH distributions therefore present a well-specified and well-maintained release of Hadoop for SCAPE purposes, and consequently it is used for the Central Instance and some partner organisation's clusters (Section 7.9).

Since adopting CDH3 however, Cloudera have continued to develop their product. At the time of writing, CDH4 is the current stable release of CDH (with CDH5 in beta). CDH4 is based on Hadoop 2.0 and, more importantly, a new version of MapReduce (MRv2) which is essentially a rewrite of the MapReduce infrastructure. MRv2 includes a new resource management component, called YARN (Yet Another Resource Negotiator) [58], which offers improvements in scalability for resource management and job scheduling. However, Cloudera do not consider MRv2 to be stable enough for production environments, and recommend the continued use of MRv1. CDH4 does offer MRv1 support, however given SCAPE's intention of continuing to use MRv1, the project has decided the advantages of migrating to this CDH4 version are currently minimal, and focus should remain on delivering results for the existing chosen platform version.

CDH3 update 2 includes (as well as other technologies):

- Hadoop version 0.20.2
- HBase version 0.90.4
- Apache Pig version 0.8.1
- Apache Oozie pre-Apache Incubator version

Execution Environment

Hadoop is purposefully designed to operate on commodity hardware, enabling construction of a cluster of computational nodes from affordable PCs. That said, many institutions prefer to operate Hadoop clusters over multi-core servers, particularly making use of virtualisation software to quickly setup and deploy nodes.

CDH3 prerequisites are fully defined in [60], but essentially they are:

- Linux OS: Ubuntu, Debian, Red Hat compatible or SUSE
- Java 1.6

7.5.3 Integration Overview

Figure 13 indicates the integration required with a Hadoop based Execution Platform. In particular this diagram highlights the user interaction required to take a Preservation Plan stored in the Digital Object Repository and optimise it for execution in a parallel Execution Platform.

²⁰ The latest update of CDH3 is update 6. This is still based on Hadoop 0.20.2, although other software is likely to have been updated (e.g. HBase), see the release notes for further details [60].

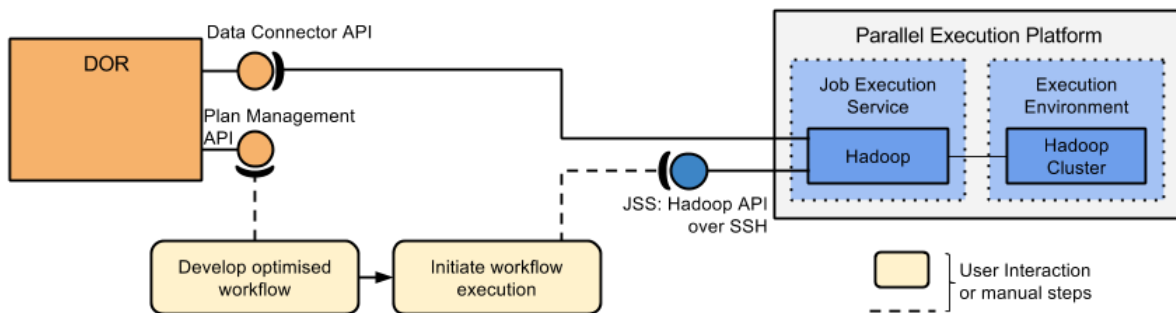


Figure 13: Hadoop Core integration with the Digital Object Repository

To run a Preservation Plan workflow on a parallel Execution Platform in an optimal way it will need to be translated to, or re-written as, a MapReduce application. There are a few main options to performing this re-write, but all require some level of development effort to achieve:

- Rewrite the workflow by hand to a MapReduce application.
- Take advantage of extension technologies to core Hadoop and rewrite as an Apache Oozie workflow or an Apache Pig script (Section 7.6 for both).
- Use the prototype SCAPE Taverna-to-Pig Translator to generate Pig scripts (Section 7.6.1).
- Use the SCAPE ToMaR tool which is designed to simplify parallel execution of command line tool invocations (Section 7.6.2).

Once created, the user can initiate execution of this MapReduce application on a Hadoop cluster through Hadoop's usual command line mechanism. Alternatively, Hadoop jobs could be initiated through elements in a Taverna workflow; there are numerous blog posts which describe this approach [53][61][62].

7.5.4 Installation of Tools on a Cluster

As described in section 5.5 the underlying tool dependencies involved in a workflow can be packaged into apt packages for installation on cluster nodes using the Debian package management.

7.5.5 Extensions to the Hadoop Core Approach

The Hadoop approach described above requires the generation of MapReduce applications (translated from a Preservation Action Plan, for example) in order to execute a scalability optimised workflow of preservation actions. However there are many other technologies built on top of Hadoop designed to facilitate development of MapReduce programs and workflows. Two projects that have received focus within SCAPE are Apache Oozie [54] and Apache Pig [55].

Apache Oozie is a workflow scheduler system designed to manage Hadoop jobs. More specifically, it is designed to run Oozie workflows represented as a sequence of actions arranged in a Directed Acyclical Graph (DAG). Actions trigger the execution of computation tasks, such as MapReduce tasks, Hadoop file systems tasks, Pig scripts, or Oozie sub-workflows. A more in-depth account of Oozie is available via its documentation pages [54].

Apache Pig is a platform for processing large data sets comprising a high-level language for expressing the computation, called Pig Latin, and a compiler for producing sequences of MapReduce jobs from the Pig scripts. Pig Latin has a large amount of built in functions, however these can be extended by a developer through the creation of User Defined Functions (UDFs) in either Java, Jython, Python, JavaScript, Ruby or Groovy. This ability to write UDFs which can easily be distributed across a Hadoop cluster, coupled with the compilation to MapReduce jobs for execution is

particularly appealing to SCAPE for scalability-optimised execution. However, generation of UDF's and the associated Pig Scripts does require specialist development support.

Figure 14 shows these technologies applied against the abstract layers described earlier. It should be noted, that whilst shown as separate stacks, Pig scripts can also be used within Oozie workflows. It should also be noted that these tools often present alternative means to working with them, for example Oozie has a client command line interface and also a Java API [63] for programmatic invocation.

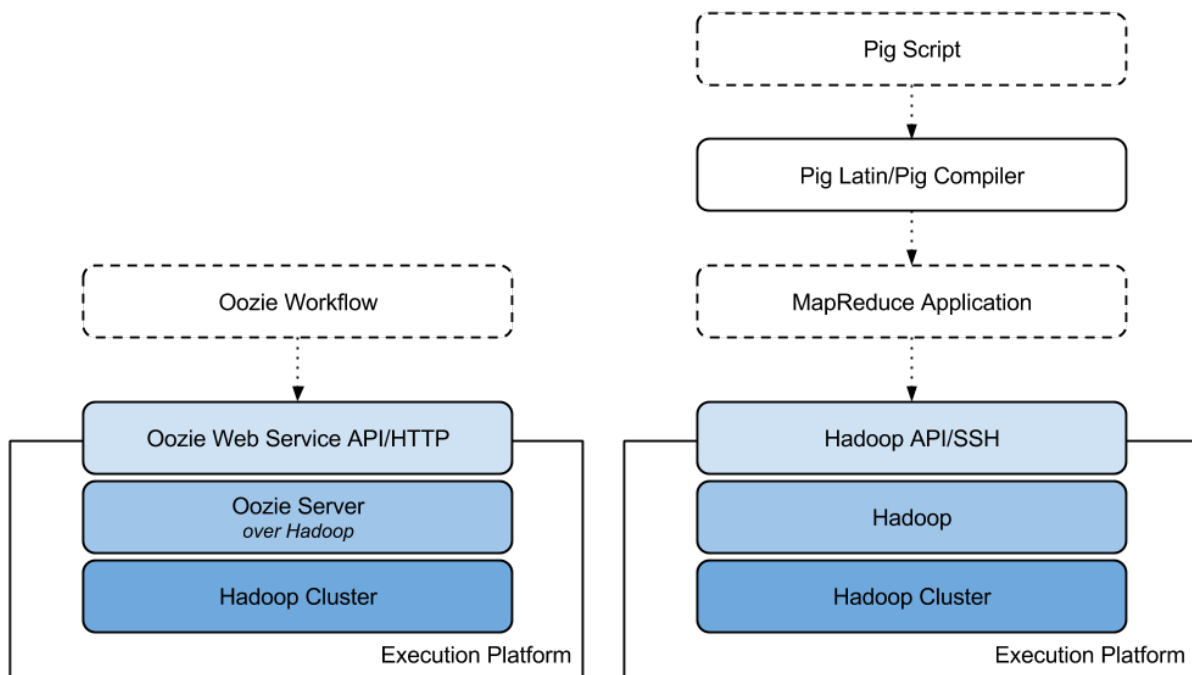


Figure 14: Apache Oozie (left) and Pig (right) technologies as applicable to the SCAPE Execution Platform

Job Submission Service API

Apache Oozie Server provides a HTTP REST JSON Web Services API [51] which, amongst other functionality, provides end points for submitting, managing and retrieving information about Oozie workflows. This includes the ability to submit workflows containing MapReduce or Pig actions.

Apache Pig runs as a client side application, where, if set to MapReduce mode, compiles Pig scripts to MapReduce jobs and executes them on a Hadoop cluster. The Execution Platform underlying this, therefore, is the same as for the Hadoop Core approach described earlier.

Job Execution Service

Apache Oozie has a Server²¹ which is responsible for launching and controlling jobs, as well as provision of the Web Services API. It is responsible for communicating and initiating jobs on Hadoop. Oozie comes already installed with Cloudera's CHD3 distribution, however this is not the latest release. It can also be installed separately [64].

²¹ It also has a client which for launching jobs via a command line interface.

For the Apache Pig approach, the Execution Platform is simply executing MapReduce Applications as per the Hadoop core approach described in section 7.5.2.

Execution Environment

All computation tasks triggered by Oozie actions are executed on Hadoop MapReduce over a Hadoop Cluster. This is similar for Pig, which is just executing MapReduce jobs. The environment needed for either approach is therefore as described for the Execution Environment in section 7.5.2.

7.6 Workflow Parallelisation Strategies

As has been touched upon earlier, a key challenge is the optimisation of tools and workflows for scalable execution. Section 7.4 provides an integrated approach from creating Preservation Plans through to executing them on the Taverna Server Execution Platform. However, to achieve higher levels of computation throughput on increasingly larger data sets, some form of parallel processing system will be required, with tools and workflows optimised for it. It should be noted that the strategy used to parallelise an individual workflow depends a lot on the use case and should be selected on a case-by-case basis.

This section considers strategies for adapting tools or workflows towards optimised execution on the Hadoop platform.

7.6.1 Taverna Workflow to MapReduce

PPL-Translator

Initial SCAPE investigations developed a prototype translator program, called the PPL (Program for parallel Preservation Load), to facilitate the parallel execution of Taverna workflows. In essence, it takes a workflow and generates a class consisting of a linear list of MapReduce jobs for execution on a Hadoop cluster. It does this by using the SCUFL2 API [65] to traverse the workflow, building up a list of workflow elements which are translated to MapReduce jobs using templates. Further details are available in [SD3] or from the source code [SW16].

Several investigations were conducted, comparing the compilation of workflows down to a series of MapReduce jobs versus in one Map job, and also a comparison of automatic translation (via the PPL-translator) with a manually implemented equivalent MapReduce job. Details about the investigations and results can be found in [SD3] and are summarised in [SD4], but ultimately it was concluded that processing overhead should be minimized by jointly executing all processes that can be by combining them into one execution of a Map or Reduce phase.

Manual creation of MapReduce jobs was found to result in significantly faster execution times than automatic compilation (using this PPL-translator program). One factor responsible for the difference is that the automatically compiled version employs beanshell scripts to execute the logic, meaning that every Reduce call creates a new beanshell to parse the script. The manually created job did not employ beanshells at all, emphasising the benefits of developer support in optimising workflows for scalable execution.

Sustainable Approaches using Existing Technologies

Subsequent analysis of this approach to automatic Taverna workflow to MapReduce jobs concluded a number of problems:

- Most Taverna workflows currently in use by the Taverna and SCAPE communities define user defined functions (UDFs) in ways that run contrary to the MapReduce paradigm (primarily the use of locally installed tools)
- Only a subset of all operators in Taverna can be compiled to the MapReduce paradigm

- The compilation and optimization of complex workflows is highly difficult and costly in terms of development effort.

These are described in greater detail in [SD4]. Importantly however, it was noted that many of these issues have been recognized and subject to research by a large and active community in the field of scalable processing. Apache Pig [55] is one such open-source project that provides a high level language for easily defining data processing programs, the compilation environment to translate them to MapReduce jobs, and the ability to extend it with user defined functions which can easily be distributed amongst the cluster. Leveraging this technology and its underlying community support would provide a more robust and sustainable approach.

Use of Apache Pig

With developer support, Pig Latin scripts can manually be written to reproduce the functionality of a Preservation workflow. From here, the Apache Pig compilation engine can be used to generate MapReduce jobs for parallel execution on Hadoop. Details about this can be found from Apache Pig documentation [55]. An example, manually written, script is shown in [SD4].

Whilst Pig Latin is a high level language, it should be evident that it will still require specialist software development skills and experience in order to write the Apache Pig workflows. To help facilitate this process, SCAPE are currently investigating automatic Taverna to Pig Script translation. This translator works in a similar way to the PPL-Translator, using the SCUFL2 API to traverse the workflow and translate elements to generate an appropriate Pig Script. A more complete description is given in [SD5]²².

As shown in Figure 15, such a tool could then be incorporated into the Taverna Server approach (Section 7.4), enabling an integrated approach from Preservation Plan creation through to scalability-optimised execution. A description of this Taverna integration Hadoop is touched upon as future work within SCAPE deliverable D7.2 - Workflow Modelling Environment [SD6].

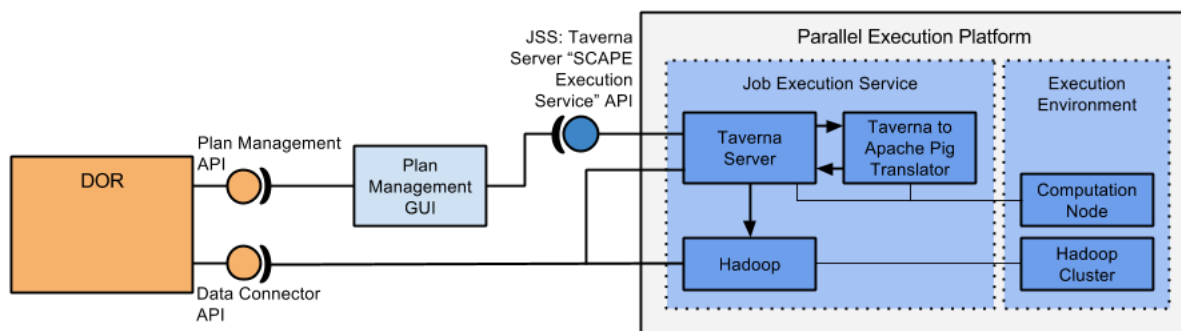


Figure 15: Optimised execution of a Preservation Plan through integration of Taverna Server, a Taverna to Pig Translator and Hadoop.

7.6.2 Tools to MapReduce

Parallelisation optimisation has so far focused on the workflow level, specifically, translating a Taverna workflow into some other form for execution in a Hadoop environment. These Taverna workflows comprise SCAPE Components (Section 5.3.1), which at their heart typically wrap some tool invocation, e.g. Apache Tika [19] or FITS [66]. However, consideration should also be given to the distributed execution of these tools, particularly for command line tool invocations.

ToMaR - Tool to MapReduce Wrapper

²² At the time of writing, this deliverable is in draft format.

SCAPE's ToMaR is a Tool-to-MapReduce wrapper which easily enables the distributed execution of command line tools across Hadoop [SW17]. It is particularly suited for those with little or no programming experience where, for existing SCAPE Components²³, only a text based control file needs to be created describing a single tool invocation (the *action*) per line.

In the simplest case, ToMaR is initiated through the Hadoop API, resulting in tool execution across the Hadoop cluster based on distributing the invocations in the specified control file. One Hadoop map task processes one line from the control file; as described in that line, it fetches the specified toolspec (the same toolspec as used by the Toolwrapper - see Section 5.3.5), copies the specified remote files to the local file system and executes the specified action's command, depositing the results of that action in the declared remote location. Parallelisation therefore happens through concurrent execution of multiple map tasks across the Hadoop cluster.

A slightly more advanced case would be to use a Taverna workflow to control invocation of ToMaR, as used in this example of characterising a Web Archive using FITS [67]. Or it could even be initiated through an Oozie job.

A more detailed explanation of how ToMaR works and how to use it can be found in the project's ReadMe [SW17].

7.7 Public APIs

The Execution Platform must provide a concrete Job Submission Service API implementation, although, as indicated in sections 7.4.2, 7.5.2 and 7.5.4, the exact nature of this implementation will be dependent on the execution approach chosen.

7.7.1 Job Submission Service API

The Job Submission Service API provides the entry point to the Execution Platform, implementing a remotely accessible interface to enable a user or client application to schedule and execute workflows on the Execution Platform. The actual interface implemented depends on the underlying platform technology, but typical examples would be the Hadoop API [50] provided over a SSH connection, or the Apache Oozie REST API [51] or Taverna Server REST API [68] over HTTP.

Clients are responsible for initiating execution of a Preservation Plan workflow (or optimised version thereof) by way of this API. A Job Submission Service can be used by multiple clients enabling one Execution Platform to provide execution services for Preservation workflows from multiple Digital Object Repositories.

As an example, within SCAPE, the Plan Management GUI (Section 8.1) acts as a client and is able to initiate execution of Preservation Plans on the Taverna Server through its SCAPE Execution Service API. Where Hadoop MapReduce Applications are defined however, the Hadoop command API is used over a SSH connection to the cluster.

7.8 Packaging and Deploying

No specific deployment or infrastructure is prescribed by SCAPE, and indeed the intention is for the platform to be versatile enough to suit individual institution needs. The system may be hosted using private or institutionally shared hardware, by an external data centre, or it may be deployed on an IaaS infrastructure through virtualization.

²³ Specifically, ToMaR requires a toolspec description, as would be used to create a Component using the SCAPE Toolwrapper (Section 5.3.5).

7.8.1 Platform Releases

SCAPE have released a pre-configured virtual machine image of the SCAPE Execution Platform, providing an experimental environment that combines Taverna Workbench, the Platform's MapReduce tool executer (ToMaR), and a set of preservation tools together with a number of examples. This released is described in detail in [SD1].

The idea of this release is to provide a basic environment that can be easily installed on a desktop computer allowing users to experiment with different technologies and examples. The virtual machine release may also be deployed as a cluster on multiple machines and be extended with additional SCAPE software components.

The release is presently available as an image for VirtualBox [69] which utilizes the Ubuntu 10.04.4 LTS Linux distribution as OS²⁴. The image is presently available as a downloadable package to SCAPE participants only and will be made publicly available once licensing restrictions have been clarified.

7.9 Exemplar SCAPE Platform Instances

SCAPE work with a number of different deployment installations, but these can essentially be categorised into the Central Instance and institutional Local Instances.

7.9.1 SCAPE Central Instance

The SCAPE Central Instance provides SCAPE project members with a pre-configured Hadoop infrastructure upon which to experiment with platform software, as well as to test and benchmark tools and workflows.

The infrastructure²⁵ of the Central Instance consists of three dual-core AMD 1.6GHz (total 6 CPU cores), low consumption nodes, each with 8GB RAM and 15TB storage (5x 3TB HDDs). Emphasis is put on the consumption aspect of the nodes, whereby commodity hardware is used in an innovative no-rack environment with no active cooling required²⁶.

Cloudera CDH3 provisions the cluster with Apache Hadoop, along with Apache Pig and a pre-Apache Incubator version of Apache Oozie.

7.9.2 Hadoop Local Instances

Local Instances are platform instances setup and maintained by an institution primarily to evaluate their own data sets, and helping to alleviate licensing restrictions which can often prevent the data from being uploaded to an offsite repository. In addition, by implementing a platform instance, institutions will be able to validate SCAPE's component-oriented architecture and the ability to deploy the SCAPE platform across various hardware and software platforms (e.g. using DOR's other than the SCAPE reference implementations). The following are descriptions of two high-level system setups used by different institutions within SCAPE, and should be considered as examples rather than prescribed or promoted ideals:

²⁴ An upgrade of the VM image to the latest Ubuntu LTS release together with an update of the SCAPE platform components is planned for the final project months.

²⁵ Infrastructure updates are planned but have not been implemented at the time of writing.

²⁶ Nodes are not enclosed in boxes, enabling free air flow.

Setup 1:

A cluster comprising 10 virtual nodes (total 10 CPU cores) with an aggregated HDFS capacity of about 4TB (maximum 400GB per node). The platform is running Apache Hadoop (0.20.2-cdh3u2). A Fedora Commons-based repository is being added.

Setup 2:

Multiple Hadoop instances are used for different stages of development. Initial development occurs on developer PCs (2-4 cores, 1-4GB RAM) running a pseudo-distributed single-node Hadoop instance in a Virtual Machine. HDFS capacity is in the 10's of GB range. Complementing this is a testing/deployment cluster built around a multicore server with 32 CPUs, 224GB RAM and ~27TB of HDD space and supported by VMWare ESXi hypervisor. This server is currently configured with 30 single-CPU nodes (6GB RAM and 500GB HDD each) comprising: 1 manager, 1 namenode/jobtracker and 28 datanodes/tasktrackers.

7.9.3 Data Centre/Cloud Deployment

Deployment of SCAPE tools in large data centres or in Cloud Computing environments raises specific challenges, such as dynamic allocation of components to compute nodes, monitoring the platform, and ensuring quality of service. These challenges can be alleviated in part by utilising a combination of tools, such as node deployment systems, automation or configuration management systems, and Infrastructure monitoring tools.

Node deployment systems, such as Cobbler [70], help administrators to automate the installation and configuration management of multiple, networked computers from a centralised point, for both bare metal servers and on virtualised computing resources. This can be complemented by automation software, such as Puppet [71], which allow the evolution of SCAPE software packages by providing "high-level" recipes describing the tools and relations between them. SCAPE Components can easily be provisioned to computing nodes with minimal human intervention, providing a more deterministic software deployment process and ensuring that software is deployed as expected by the developers, meeting all the required expectations. Finally, the integrated Puppet Configuration Management system natively provides capabilities for integration with and deployment of the Nagios monitoring solution [72], enabling administrators to monitor infrastructure for problems and ultimately provide a better quality of service.

SCAPE Cloud Deployment Toolkit for Eucalyptus and Amazon Web Services

SCAPE have started building a proof-of-concept toolkit which aims at providing puppet modules for deploying critical SCAPE software in Cloud Environments. It will be aimed at demonstrating the operation of selected SCAPE software tools and services in Cloud Environments, focusing on Eucalyptus and Amazon EC2, and fostering their scalability for providing on demand computing capacity.

The toolkit is composed of:

- A GUI (web based portal) for the management of an SCAPE Platform deployment on Eucalyptus based clouds and, in a subsequent stage, on Amazon Web Services EC2
- Puppet and PuppetDB Rest APIs
- Abstracted EC2 and Eucalyptus APIs for providing a uniform programming environment, thereby ensuring portability

Puppet recipes for the following SCAPE tools and services have been created [SW18]:

- Taverna Server
- Tomcat Server

- Jpylyzer
- Pagelyzer
- xcorrSound

HDInsight: Hadoop on Azure

Windows Azure HDInsight is a service that provides cloud based Hadoop (versions 1.0.3, 1.2.0 and 2.2) clusters for analyzing big data [73]. It is built around Microsoft’s Azure computing platform [74], which is used to build, host and scale applications across a global network of Microsoft-managed data centres, potentially providing a simpler route to a fully featured Hadoop cluster. Figure 16 provides an overview of these technologies with reference to the abstracted SCAPE Execution Platform layers defined in Section 7.2.

Implementations of Apache Pig and Apache Oozie would enable the use of the previously described approaches (Section 7.5.4) to develop scalable preservation workflows. A fully featured HDInsight emulator provides a local, single-node, development environment which can be used to define Hadoop jobs before moving over to HDInsight for production [75].

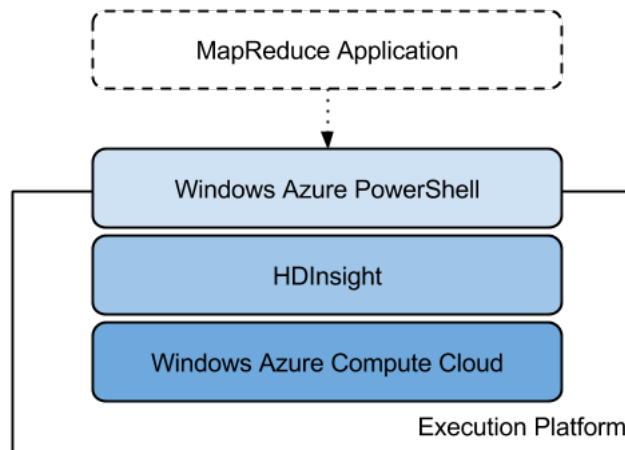


Figure 16: Technologies for Parallel Execution with HDInsight

At the time of writing, access has only just been gained to HDInsight, and so SCAPE are just beginning to investigate how easily existing preservation tools and workflows can be deployed on the platform.

8 Supporting Tools and Services

This section describes four tools and services which support the Watch, Planning, Digital Object Repository, and Platform main entities in the SCAPE ecosystem. These are:

- **Plan Management GUI:** GUI supporting user interaction with Preservation Plans.
- **Loader Application:** Application for loading Intellectual Entities into the repository.
- **C3PO:** A content profiling tool for digital collections.
- **Repository Simulator:** A tool for simulating effects on repository states.

8.1 Plan Management GUI

This Web Application is an reference implementation client application to the Plan Management API and Job Submission Service API. Different client implementations are entirely possible and not restricted by SCAPE, for example, a command line client, similar functionality being integrated into a repository interface or even splitting the plan upload/retrieval and execution into two separate user interfaces.

8.1.1 Functional Overview

This SCAPE Plan Management GUI Web Application [SW19] enables a user to store or update Preservation Plans in the DOR as well as to retrieve existing plans and easily initiate their execution on the Execution Platform. Essentially, this implementation acts as a user interface to bridge the gap between plan creation in Plato, plan storage in the DOR and plan execution on the Execution Platform, as shown in Figure 17.

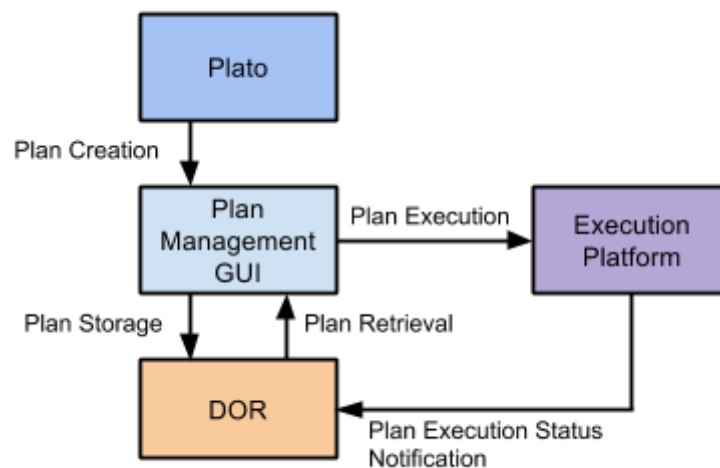


Figure 17: Flow of Preservation Plans through the Plan Management GUI from creation in Plato, storage in the DOR, to execution on the Execution Platform

8.1.2 Technical Overview

The Plan Management GUI is a JavaScript based web application that builds as a WAR file. It requires a servlet container, such as Tomcat, to deploy it.

Upload and execution of Preservation Plans occurs via JavaScript HTTP calls to the appropriate Plan Management API and Job Submission Service endpoints, respectively. In particular, with regards to plan execution, the JavaScript code extracts the Preservation Action Plan portion from the Preservation Plan and calls the appropriate Job Submission Service API (Section 7.7.1) to deliver that

PAP to Taverna Server and initiate its execution. Taverna Server notifies the DOR's Plan Management Service that execution has started and also provides notification of execution results.

Source code and a pre-packaged WAR file of the Plan Management GUI are both available for download [SW19].

8.2 Loader Application

8.2.1 Functional Overview

The SCAPE Loader Application [SW20] is an external (to the DOR) service that provides a means for an administrator to load a large number of digital objects into a DOR whilst being able to monitor and report on the ingest process. Specifically, the Loader Application enables the loading of large numbers of Intellectual Entities in an efficient manner; loading of the associated binary files is not handled by this application.

Intellectual Entities are described by a METS file and loaded using the SCAPE Data Connector API (Section 6.4.1), implemented by the DOR. This enables the Loader Application to work with any DOR, releasing burden on the repository provider to implement a specific loader application for their repository.

8.2.2 Technical Overview

The SCAPE Loader Application ingests SIPs which are METS documents conforming to the profile defined by the SCAPE Digital Object Model specification [SD7]. One SIP represents one Intellectual Entity, and each representation, file and bitstream thereof is described by technical metadata contained within the SIP.

SIPs are expected to be created prior to uploading. Once created, they are picked up by the Loader Application from a source location and registered for ingest into the repository. Alternatively, SIPs can be stored in a Hadoop SequenceFile²⁷ - a key/value pair flat file - which the Loader Application can use to ingest the Intellectual Entities into the repository. The command line interface of the Loader Application offers a way to configure the path of the source directory, as well as the REST endpoints of the repository to be used.

The Loader Application offers three distinct services:

1. **Add:** Adds SIPs to the Loader Application's registry (e.g. HSQLDB) while reading a source directory, registering them for ingest into the repository.
2. **Ingest:** Triggers a HTTP POST request for each SIP, ingesting each Intellectual Entity asynchronously into the repository.
3. **Get State:** Retrieves the status of the SIP during the ingest process and records it in the Loader Application's registry enabling the user to be informed about the current progress.

Ingest of SIPs into the DOR is via the Data Connector API HTTP endpoints (Section 6.4.1). The actual binary files (e.g., TIFFs, PDFs, WAVs) which the SIPs pertain to will not be directly ingested into the repository by the Loader Application, but merely referenced in the METS description of each Intellectual Entity; the application's focus is to ensure correct ingestion of the Intellectual Entities.

Source code for the SCAPE Loader Application is available [SW20]. This is a command line application however it could be easily extended to provide a GUI.

²⁷ This SequenceFile may reside on HDFS or on the local file system.

8.3 C3PO

8.3.1 Functional Overview

C3PO (Clever, Crafty Content Profiling of Objects) [SW21] processes metadata extracted from files in a digital collection and generates a content profile. C3PO doesn't perform characterisation; it parses and aggregates the output of characterisation tools then aggregates and persists metadata.

C3PO provides a command line utility for importing metadata, generating content profiles, selecting representative samples and exporting profile data. C3PO's web application provides a browser based GUI for content profile visualisation and analysis, and a REST API for the retrieval of aggregated profile data.

8.3.2 Technical Overview

C3PO's extensible design allows for the easy integration of different metadata formats generated by new characterisation tools. It also enables users to develop a new persistence layer for storing metadata, although this requires more effort than integrating a new metadata format.

The process of importing of metadata is split between a gatherer interface and a metadata adaptor class that extends an abstract class. The gatherer is responsible connecting to a source and reading in raw metadata. C3PO currently has a single gatherer implementation that collects information from the local file system.

The adaptors transform gathered metadata into a form compatible with C3PO's internal data model. Integrating a new characterisation tool often only requires the implementation of a new adaptor that parses the tool output for a single content object at a time and returns a metadata data element which the persistence layer is capable of storing. The current version of C3PO has adaptors that can import the output of the Apache Tika™ [19] and FITS [66] characterisation tools.

C3PO currently provides a MongoDB [76] backed metadata persistence layer, and a Mongo installation is a prerequisite before installing C3PO.

8.4 Repository Simulator

8.4.1 Functional Overview

One of the major challenges when dealing with huge amount of heterogeneous data is to anticipate the future state of a repository. Simulation methods are useful for addressing this kind of a problem because they offer a cheap way of testing repository responses in different scenarios. Consequently interested users can easily get a good overview of possible future resource requirements, with the potential to identify trends such as accelerating storage requirements.

The goal of the simulation tool is to provide an environment which can support simulating different assumptions about the repository and the events that can occur on it. For example, this could be assumptions about the format profile of collections, and the effects of ingesting other collection material. To enable this, the simulation environment contains both a simulation language for modelling these assumptions, and a simulation engine for enacting them.

8.4.2 Technical Overview

The Repository Simulator is a standalone (from the rest of SCAPE) Java tool that runs as an Eclipse plugin. It combines the high-level simulation language for defining a repository in a prescribed state and for defining a set of actions to perform on that repository, along with a simulation engine to enact those actions. The simulation engine is based on discrete event simulation principles, i.e.

system operation is modelled as a discrete sequence of events in time, with each event marking a change of state in the system, and no state change occurring between events [77].

Source code for the Simulator can be found in [SW22]. The simulation environment is described in detail in the SCAPE deliverable D12.3 [SD9]²⁸.

9 Conclusion

The SCAPE project has focussed on developing scalable tools, services and infrastructure for the efficient planning and execution of preservation strategies for large-scale, heterogeneous collections of complex digital objects, in an effort to enhance the digital preservation state-of-the-art.

SCAPE developed an ecosystem of tools and services to facilitate the automated production of Preservation Plans, monitoring of knowledge which impacts these plans, and the scalable execution of the Preservation Plan workflows on large content collections. This ecosystem presents a modular infrastructure divided across five main entities, Automated Watch, Automated Planning, Component Management, the Digital Object Repository and the Execution Platform. Interactions between these entities is by way of SCAPE defined interfaces, enabling flexibility in entity implementation and facilitating institutions wishing to amalgamate their existing infrastructure with SCAPE technologies.

Two main approaches to running Preservation Plan workflows are described, which facilitate either direct execution or, following further optimisations, execution over the Hadoop parallel infrastructure. Direct invocation provides a high level of integration within the SCAPE ecosystem but is perhaps not well suited for large collections. Conversely, optimising the workflows for execution on Hadoop enhances the computational throughput achievable, making it more suited to very large collections, but reduces integration and requires additional development effort to achieve.

Overall, this report has documented this ecosystem of tools and services. It started by describing the overall architecture of this ecosystem and the intended integration of the main entities. It then described each main entity in detail, providing functional and technical explanations along with descriptions of the interfaces that entities must implement. Finally, additional SCAPE developed tools and services that complement the main entities were described.

The intention is, by having read this report, practitioners will have gained enough knowledge and understanding to help them to implement all or part of the SCAPE ecosystem within their organisations.

9.1 Remaining Work

At the time of writing, SCAPE has a further 6 months until completion and so a number of milestones and deliverables remain to be completed. These are listed in Table 1 below.

To summarise the remaining work, most SCAPE entities (e.g. Watch, Planning, Digital Object Repository, Platform) have to deliver final versions of their software/services. Planning has to report on the evaluations of Plato against preservation models such as OAIS [37] and compliance with trustworthy repository criteria such as TRAC [78]. The Component Catalogue (Section 5.3.3) design and implementation needs to be documented. Finally, tasks from SCAPE-Enlarged about the anonymised data ingestion prototype need to be finalised and reported on, as well as prototypes are reports on the deployment of SCAPE within Data Centres and cloud platforms.

²⁸ This deliverable is not due until the end of the project (September 2014).

Table 1: Remaining Milestones and Deliverables

Milestone/ Deliverable	Description	Due
Watch	(Section 3)	
D12.2	Final version of the Preservation Watch Component due	M38
D12.3	Final version of the Simulation Environment	M42
Planning	(Section 4)	
MS63	Report on compliance validation	M40
D14.2	Final version of automated policy-aware planning component	M42
Components	(Section 5)	
D7.3	Design and implementation of the preservation component catalogue	M40
DOR	(Section 6)	
MS97	Final anonymized ingestion prototype	M42
D8.2	Reference implementation of DOR with interfaces to preservation components, workflows, and execution	M44
D8.3	Initial anonymized ingestion prototype	M40
Platform	(Section 7)	
MS41	Final preservation workflow sharing platform	M42
MS94	Cross-platform execution prototype	M44
D4.3	Final Data Center Deployments	M40
D5.3	Application Integration and Provisioning	M42

10 References

- [1] Apache v 2.0 Licence, <http://www.apache.org/licenses/LICENSE-2.0.html>.
- [2] Git revision control, <http://git-scm.com/>.
- [3] GitHub, <https://github.com/>.
- [4] GitHub Issues, <https://github.com/blog/831-issues-2-0-the-next-generation>.
- [5] GitHub Pages, <http://pages.github.com/>.
- [6] GitHub Pull Requests, <https://help.github.com/articles/using-pull-requests>.
- [7] Open Planets Foundation GitHub Organisation, <https://github.com/openplanets>.
- [8] Open Planets Foundation, <http://openplanetsfoundation.org/>.
- [9] Travis-CI, <https://travis-ci.org/>.
- [10] OPF Jenkins Server, <http://jenkins.opf-labs.org>.
- [11] OPF Sonar Server, <http://sonar.opf-labs.org>.
- [12] OPF Sonar SCAPE QA, <http://sonar.opf-labs.org/dashboard/?did=6>.
- [13] SCAPE JavaDoc sites, <http://projects.opf-labs.org/scape/>.
- [14] Sonatype Central Repository, <http://central.sonatype.org/>.
- [15] Maven Snapshot Repository SCAPE root ,
<http://oss.sonatype.org/content/repositories/snapshots/eu/scape-project/>.
- [16] Ubuntu 12.04 LTS, <http://releases.ubuntu.com/12.04/>.
- [17] Advanced Packaging Tool, http://en.wikipedia.org/wiki/Advanced_Packaging_Tool.
- [18] Bintray, <https://bintray.com/>.
- [19] Apache Tika™, <http://tika.apache.org/http://tika.apache.org/>.
- [20] MyExperiment, <http://www.myexperiment.org>.
- [21] PRONOM Unique ID (PUID),
<http://www.nationalarchives.gov.uk/aboutapps/pronom/puid.htm>.
- [22] Scout REST API, http://projects.opf-labs.org/scape/scout/apidocs/eu/scape_project/watch/rest/WatchClient.html
- [23] Apache Jena, <http://jena.apache.org/http://jena.apache.org/>.
- [24] Apache Tomcat, <http://tomcat.apache.org/>.
- [25] Jersey, <http://jersey.java.net>.
- [26] GlassFish, <http://glassfish.java.net/http://glassfish.java.net/>.
- [27] Planets Project, <http://www.planets-project.eu/>.
- [28] Plato, <http://www.ifs.tuwien.ac.at/dp/plato/intro.html>.
- [29] Plato XML Schema, <http://ifs.tuwien.ac.at/dp/plato/schemas/plato-V4.xsd>.
- [30] Plato Preservation Action Plan Schema,
<http://ifs.tuwien.ac.at/dp/plato/schemas/preservationActionPlan-V1.xsd>.

- [31] W3C Semantic Web, <http://www.w3.org/standards/semanticweb/>.
- [32] JBoss, <http://www.jboss.org/>.
- [33] Taverna Workbench, <http://www.taverna.org.uk/download/workbench/>.
- [34] Eclipse, <https://www.eclipse.org>.
- [35] Taverna Server, <http://www.taverna.org.uk/download/server/>.
- [36] MyExperiment REST API, <http://wiki.myexperiment.org/index.php/Developer:API>.
- [37] MyExperiment Components API, <http://wiki.myexperiment.org/index.php/Developer:Components>.
- [38] ISO 14721:2003 "Space data and information transfer systems -- Open archival information system -- Reference model", http://www.iso.org/iso/catalogue_detail.htm?csnumber=24683 (this has been superseded by the 2012 model).
- [39] ISO 14721:2012 "Space data and information transfer systems -- Open archival information system -- Reference model", http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=57284.
http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=57284
- [40] METS, <http://www.loc.gov/standards/mets/>.
<http://www.loc.gov/standards/mets/>
- [41] PREMIS, <http://www.loc.gov/standards/premis/>.
<http://www.loc.gov/standards/premis/>
- [42] Search/Retrieve via URL (SRU), <http://www.loc.gov/standards/sru/>.
- [43] SRU Contextual Query language (CQL), <http://www.loc.gov/standards/sru/specs/cql.html>
<http://www.loc.gov/standards/sru/specs/cql.html>
- [44] "Plan Management API", F. Asseg, M. Hahn, 2012, SCAPE, https://github.com/openplanets/scape-apis/blob/master/Plan%20Management%20API_V1.0.pdf
- [45] OAI-PMH, <http://www.openarchives.org/OAI/openarchivesprotocol.html>
<http://www.openarchives.org/OAI/openarchivesprotocol.html>
- [46] "Dublin Core Metadata Element Set, Version 1.1: Reference Description", <http://dublincore.org/documents/1999/07/02/dces/>.
- [47] "Report API Specification", R. Castro, M. Ferreira, L. Faria, F. Asseg, P. Petrov, 2012, SCAPE, https://github.com/openplanets/scape-apis/blob/master/ReportAPI_V1.0.pdf.
- [48] Fedora Repository Wiki home, <https://wiki.duraspace.org/display/FF/Fedora+Repository+Home>.
- [49] RODA - Repository of Authentic Digital Objects, <http://www.roda-community.org/>.
- [50] Document Object Management System (DOMS), <http://wiki.statsbiblioteket.dk/domswiki/>.
- [51] Hadoop API, <https://hadoop.apache.org/docs/r2.2.0/api/>.
- [52] Apache Oozie Web Services API, <https://oozie.apache.org/docs/4.0.0/WebServicesAPI.html>
- [53] Apache Hadoop, <http://hadoop.apache.org/>.
<http://hadoop.apache.org/>

- [54] OPF Blog: “Mixing Hadoop and Taverna”, W. Palmer, 14-02-2013, <http://www.openplanetsfoundation.org/blogs/2013-02-14-mixing-hadoop-and-taverna>.
- [55] Apache Oozie, <http://oozie.apache.org/>.
- [56] Apache Pig, <http://pig.apache.org/>.
- [57] “MapReduce Tutorial”,
https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.htmlhttp://hadoop.apache.org/mon/docs/r0.20.2/mapred_tutorial.html.
- [58] “Cloudera Distribution including Apache Hadoop”,
<http://www.cloudera.com/hadoop/><http://www.cloudera.com/hadoop/>
- [59] CDH4 and MapReduce, http://www.cloudera.com/content/cloudera-content/cloudera-docs/CDH4/4.2.2/CDH4-Installation-Guide/cdh4ig_topic_4_1.html.
- [60] CDH3 Update 6, <http://www.cloudera.com/content/cloudera-content/cloudera-docs/CDH3/CDH3u6/CDH3-Release-Notes/whats-new-u6.html>.
- [61] CDH3 Installation Guide, <http://www.cloudera.com/content/cloudera-content/cloudera-docs/CDH3/CDH3u6/CDH3-Installation-Guide>.
- [62] OPF Blog: “Web Archive FITS Characterisation using ToMaR”, S.Schlarb, 16-12-2013, <http://www.openplanetsfoundation.org/blogs/2013-12-16-web-archive-fits-characterisation-using-tomar>
- [63] OPF Blog: “DROID File Format Identification using Hadoop”, S.Schlarb, 24-05-2013, <http://www.openplanetsfoundation.org/blogs/2013-05-24-droid-file-format-identification-using-hadoop>
- [64] Apache Oozie Java API, <https://oozie.apache.org/docs/4.0.0/client/apidocs/org/apache/oozie/client/OozieClient.html>.
- [65] Apache Oozie Quick Start Guide, https://oozie.apache.org/docs/4.0.0/DG_QuickStart.html
- [66] SCUFL2 API, <http://dev.mygrid.org.uk/wiki/display/developer/SCUFL2+API>
- [67] File Information Tool Set (FITS), <http://projects.iq.harvard.edu/fits>
- [68] Web Archive FITS Characterisation using ToMaR, <http://www.openplanetsfoundation.org/blogs/2013-12-16-web-archive-fits-characterisation-using-tomar>
- [69] Taverna Server REST API, <http://dev.mygrid.org.uk/wiki/display/taverna/REST+API>
- [70] VirtualBox, <https://www.virtualbox.org/>
- [71] Cobbler, <http://www.cobblerd.org/>
- [72] Puppet, <http://puppetlabs.com/>
- [73] Nagios Infrastructure Monitoring Solution, <http://www.nagios.org/>
- [74] HDInsight, <http://www.windowsazure.com/en-us/documentation/services/hdinsight/>
- [75] Windows Azure, <http://www.windowsazure.com/en-us/develop/overview/>
- [76] HDInsight Emulator, <http://www.windowsazure.com/en-us/documentation/articles/hdinsight-get-started-emulator/>
- [77] MongoDB, <http://www.mongodb.org/>

- [78] Discrete Event Simulation, revision: 18:10, 20 March 2014, http://en.wikipedia.org/w/index.php?title=Discrete_event_simulation&oldid=600482215
- [79] TRAC and TDR Checklists, Center for Research Libraries, <http://www.crl.edu/archiving-preservation/digital-archives/metrics-assessing-and-certifying-0/>

10.1 SCAPE Deliverables

The following links provide references to SCAPE specific deliverables referenced within this report. All deliverables are (or will shortly be) available from:

<http://www.scape-project.eu/category/deliverable>.

- [SD1] "D4.2: Final Platform Release", R. Schmidt, M. Rella, 31-01-2014, v1.0
- [SD2] "D5.1: Guidelines for deploying preservation tools and environments", R. Schmidt, D. Tarrant, R. Castro, M. Ferreira, H. Silva, 2012,
- [SD3] "D6.1: Report on the feasibility of parallelising preservation processes", M. Schenck, 03-04-2013, v1.2
- [SD4] "D6.2: Demonstrator and report on workflow compilation and parallel execution", A. Akbik, 28-11-2013, v1.0
- [SD5] "D6.3: Optimisation of Preservation Processes", A. Akbik (v1.0 due 31-03-2014)
- [SD6] "D7.2: Workflow modelling environment", D. Fellows, 31-01-2014, v1.0
- [SD7] "D8.1: Recommendations for Preservation-aware Digital Object Model", M. Hahn, F. Asseg, N. Shirwinter, R. Castro, 31-01-2014, v1.0
- [SD8] "D12.1: Identification of Triggers and Preservation Watch Component Architecture, Subcomponents and Data Model", K. Duretec, L. Faria, P. Petrov, C. Becker, 27-01-2012, v1.0
- [SD9] "D12.3: Final Version of the Simulation Environment", K. Duretec (v1.0 due 30-09-2014)

10.2 SCAPE Software References

The following links provide references to software tools and services developed in SCAPE and described or referenced in this report (i.e. this is not a complete list of SCAPE software).

- [SW1] Scout <https://github.com/openplanets/scout>
- [SW2] Scout PRONOM Adaptor <https://github.com/openplanets/scout/tree/integration/adaptors/pronom-adaptor>
- [SW3] Scout C3PO Adaptor <https://github.com/openplanets/scout/tree/integration/adaptors/c3po-adaptor>
- [SW4] Plato <https://github.com/openplanets/plato>
- [SW5] Policies <https://github.com/openplanets/policies>
- [SW6] SCAPE Component Profiles <https://github.com/openplanets/scape-component-profiles>
- [SW7] Toolwrapper <https://github.com/openplanets/scape-toolwrapper>
- [SW8] Toolspecs <https://github.com/openplanets/scape-toolspecs>
- [SW9] Data Connector API Reference Implementation <https://github.com/openplanets/scape-fcrepo4-connector>
- [SW10] Plan Management API Reference Implementation

- <https://github.com/openplanets/scape-fcrepo4-planmanagement>
- [SW11] RODA
<https://github.com/openplanets/roda>
- [SW12] SCAPE specific RODA
https://github.com/openplanets/roda/tree/master/roda-core/roda-core-services/src/main/java/eu/scape_project/roda/core
- [SW13] Technology Compatibility Kit (TCK)
<https://github.com/openplanets/scape-tck>
- [SW14] SCAPE Digital Object Model
<https://github.com/openplanets/scape-platform-datamodel>
- [SW15] Taverna Server with SCAPE Execution Service API
<https://github.com/myGrid/taverna-server/tree/scape-execution-interface>
- [SW16] PPL-Translator: Taverna to Hadoop Compiler
<https://github.com/openplanets/taverna-to-hadoop>
- [SW17] Tool-to-MapReduce Wrapper
<https://github.com/openplanets/tomar>
- [SW18] SCAPE Puppet recipes
<https://bitbucket.org/scapeuvt/puppet-modules/src>
- [SW19] SCAPE Plan Management Web Application
<https://github.com/openplanets/scape-planmanagement-webapp>
- [SW20] SCAPE Loader Application
<https://github.com/openplanets/loader-app>
- [SW21] C3PO
<https://github.com/openplanets/c3po>
- [SW22] SCAPE Repository Simulator
<https://github.com/openplanets/scape-simulator>

11 Glossary

The following terms and abbreviations are used throughout this report:

Table 2: Terms and Definitions

Term/ Abbreviation	Description
AIP	Archival Information Package
API	Application Programming Interface
APT	Advanced Packaging Toolkit
Automated Planning	A systematic and semi-automatic process that provides the ability to assess the impact of influencers and specify actionable preservation plans that define concrete courses of actions and the directives governing their execution. This is the operative management of obsolescence and maximizing expected value with minimal costs.
Automated Watch	A systematic and semi-automatic process that provides the ability to monitor external and internal entities for changes having a potential impact on preservation and to provide notification. The Automated Watch entity denotes the architectural software component that supports the Automated Watch process.
Bitstream	A bitstream is contiguous or non-contiguous data within a file that has meaningful common properties for preservation purposes. A bitstream cannot be transformed into a standalone file without the addition of file structure (headers, and so forth) and/or reformatting to comply with a particular file format.
(SCAPE) Components	SCAPE Components are Taverna Components, identified by the SCAPE Preservation Components sub-project, that conform to the general SCAPE requirements for having annotation of their behaviour, inputs and outputs. SCAPE components may be stored in the SCAPE Component Catalogue.
Component Catalogue	The Component Catalogue is a searchable repository for the definitions of SCAPE Components, Component Families and Component Profiles. The Component Catalogue is implemented by the myExperiment service [20] and implements the Component Service API [36].
Component Management	Tools and the Component Catalogue Service encompassing the creation, storage and cross-organisational sharing of SCAPE Components.

Component Profile	A definition of an interface that a Component should conform to. A component profile defines what input ports and output ports the component must have, what inputs and outputs may be optionally present, and what semantic annotations may be attributed to the component and its ports.
CQL	Contextual Query Language
CSV	Comma Separated Values
Digital Object Model	A data exchange model, based on METS and PREMIS, to encapsulate digital objects and ensure a consistent and well-understood information exchange between SCAPE entities.
Digital Object Repository	An OAIS Compliant repository that provides a data management solution for storing content and metadata about digital objects, as well as Preservation Plans. DORs implement three interfaces (see Section 6.4): Plan Management API; Data Connector API; and the Report API.
DIP	Dissemination Information Package
DOM	Digital Object Model
DOR	Digital Object Repository
DROID	Digital Record Object Identification
Entity (architectural)	SCAPE architectural elements, e.g. the Execution Platform, the Digital Object Repository, etc. This term is used to avoid overloading the term “components”, distinguishing “architectural components” from “SCAPE Components”.
Entity (Scout)	A domain object that represents something of interest to Automated Watch, for example an entity may represent the JPEG2000 file format. Entities may have Properties.
Execution Environment	An abstract layer of the Execution Platform which provides a placeholder representing functionality to be fulfilled by a specific technology. The Execution Environment provides the physical infrastructure to perform computation. An example might be the nodes of a Hadoop cluster.
Execution Platform	An infrastructure that provides the computational resources to enact a Preservation workflow and execute Preservation actions. Abstracted into three layers: the Execution Environment; the Job Execution Service and the Job Submission Service API.
File	A file is a named and ordered sequence of bytes that is known by an operating

	system. A file can be zero or more bytes and has a file format, access permissions, and file system characteristics such as size and last modification date.
GUI	Graphical User Interface
HDFS	Hadoop Distributed File System
HTTP	HyperText Transfer Protocol
Intellectual Entity	A set of content that is considered a single intellectual unit for purposes of management and description – for example, a particular book, map, photograph, or database. An intellectual entity may have one or more digital representations.
Job Execution Service	An abstract layer of the Execution Platform which provides a placeholder representing functionality to be fulfilled by a specific technology. The Job Execution Service provides job scheduling functionality, allocating computing tasks amongst the available hardware resources available within the Execution Environment. An example might be Taverna-Server [35] or Hadoop [53].
Job Submission Service API	An abstract layer of the Execution Platform which provides a placeholder representing functionality to be fulfilled by a specific technology. Provides the entry point to the Execution Platform, implementing a remotely accessible interface to enable a user or client application to schedule and execute workflows (jobs) on the Execution Environment. The exact interface depends on the underlying Job Execution Service and Execution Platform, but typical examples would be the Hadoop API provided over a SSH connection, or the Taverna-Server REST API over HTTP.
JSON	JavaScript Object Notation
LSDR	Large Scale Digital Repository
METS	Metadata Encoding and Transmission Standard
OAIS	Open Archival Information System
OWL	Web Ontology Language
PPC	Parallel Preservation Components
PPL	Program for parallel Preservation Load
PREMIS	PREservation Metadata: Implementation Strategies

Preservation Plan	<p>A preservation plan is a live document that defines a series of preservation actions to be taken by a responsible institution due to an identified risk for a set of digital objects or records (called a collection).</p> <p>It is defined by Plato and stored in a Plan Management Service.</p>
Preservation Action Plan	<p>A Preservation Action Plan is part of a Preservation Plan and describes a set of digital objects, an operation (typically a transformation) to apply to each of them, and a rule that allows the determination of whether the operation on a particular digital object was successful. Success is determined on the basis of characteristics measured on the instantiation of the digital object, what it was transformed into, or the comparison of what it was and what it became.</p> <p>A Preservation Action Plan does not describe how to instantiate the DO, where to archive successful transformations, or where to report the outcome of applying the Preservation Action Plan.</p>
Properties (Scout)	<p>Describes a certain “quality” of an Entity, for example Properties of the JPEG2000 file format might be a common name (with value: JPEG2000) or an indication of the level of tool support (with example value: limited).</p>
RDF	Resource Description Framework
Representation	<p>A Representation is the set of files, including structural metadata, needed for a complete and reasonable rendition of an intellectual entity. There can be more than one representation for the same intellectual entity.</p>
REST	REpresentational State Transfer
SCAPE	SCAlable Preservation Environments
SDK	Software Development Kit
SIP	Submission Information Package
SOAP	Simple Object Access Protocol
Source (Scout)	<p>A source represents specific aspects of the world that are of interest to digital preservation planners, providing measurements about properties associated with it. Sources can be internal or external, i.e. they can be part of the organization responsible for preservation or part of the outside world, such as Format Registries, SCAPE Component Catalogue or Human Knowledge.</p>
SRU	Search/Retrieve via URL
SSH	Secure SHell

Taverna Component	<p>Taverna components are Taverna workflow fragments that are stored independently of the workflows that they are used in, and that are semantically annotated with information about what the behaviour of the workflow fragment is. They are logically related to a programming language shared library, though the mechanisms involved differ.</p> <p>Taverna components are stored in a component repository. This can either be a local directory, or a remote service that supports the Taverna Component API (e.g., the SCAPE Component Catalogue). Only components that are stored in a publicly accessible service can be used by a Taverna workflow that has been sent to a system that was not originally used to create it.</p>
Taverna Server	<p>Taverna Server is a multi-user service that can execute Taverna workflows. Clients do not need to understand those workflows in order to execute them.</p>
Taverna Workbench	<p>The Taverna Workbench is a desktop application for creating, editing and executing Taverna workflows.</p>
Taverna Workflow	<p>A Taverna workflow is a parallel data-processing program that can be executed by Taverna Workbench or Taverna Server. It is stored as an XML file, and has a graphical rendering.</p>
TCK	<p>Technology Compatibility Kit</p>
URI	<p>Uniform Resource Identifier</p>
WSDL	<p>Web Service Definition Language</p>
XML	<p>eXtensible Markup Language</p>