



Technical Architecture Report


Version 1

Authors

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

September 2012

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

This report provides a detailed outline of the functional components that comprise the architecture of the SCAPE project, in particular describing the interfaces required between each of these functional entities. It should be read in conjunction with the Technical Implementation Guidelines [59], which define project-wide technologies, coding best practices and recommendations for development environments and tools, and also in conjunction with other SCAPE deliverables as referenced throughout this document. This report is primarily aimed at those wishing to get an understanding of the general architecture of the SCAPE environment and how all the various development activities combine together, but it also serves as a means to capture the direction of development and ensure all developers understand the interconnections surrounding their work.

Table of Contents

Deliverable.....	i
Executive Summary.....	iii
1 Introduction.....	1
1.1 Open Source Development.....	2
1.2 Agile Development.....	2
1.3 Abbreviations.....	3
2 Architecture Overview.....	4
3 Automated Watch Component.....	8
3.1 Introduction.....	8
3.2 Functional Overview.....	8
3.3 Technical Overview.....	8
3.4 Sources.....	9
3.5 Source Adapters.....	9
3.6 Source Adaptor Manager.....	10
3.7 Data Merging and Linking.....	11
3.8 Knowledge Base.....	11
3.9 Automated Watch Client.....	12
3.10 Monitor Services.....	12
3.11 Assessment Service.....	12
3.12 Notification Service.....	13
3.13 The Repository Simulator.....	13
3.14 Required Interfaces.....	13
3.15 Packaging and Deploying.....	14
3.16 Roadmap.....	15
4 Automated Planning Component.....	16
4.1 Introduction.....	16
4.2 Functional Overview.....	16
4.3 Technical Overview.....	17

4.4	Automated Planning Tool.....	17
4.5	Web-based Analysis Tool	18
4.6	Machine Interpretable Policy Model	18
4.7	Required Interfaces	19
4.8	Packaging and Deploying	20
4.9	Roadmap	20
5	SCAPE Component Management.....	21
5.1	Introduction.....	21
5.2	Functional Overview.....	21
5.3	Technical Overview	21
5.4	SCAPE Components	21
5.5	SCAPE Component Catalogue	23
5.6	Taverna Workflow Modelling Environment.....	23
5.7	Required Interfaces	23
5.8	Packaging and Deploying	23
5.9	Roadmap	24
6	Digital Object Repository (DOR)	25
6.1	Introduction.....	25
6.2	Functional Overview.....	25
6.3	Technical Overview	25
6.4	Digital Object Model	26
6.5	Reference Repositories	26
6.6	Plan Management Logic.....	27
6.7	Data Layer.....	27
6.8	Loader Application	28
6.9	SCAPE Plan Management GUI	28
6.10	Required Interfaces	28
6.11	Roadmap	29
7	Execution Platform.....	31
7.1	Introduction.....	31
7.2	Functional Overview.....	31

7.3	Technical Overview	31
7.4	Parallel Preservation Components.....	32
7.5	Parallel Execution System	32
7.6	Taverna Engine	34
7.7	Job Execution Service	35
7.8	Microsoft Azure	35
7.9	Required Interfaces	36
7.10	Packaging and Deploying	36
7.11	Platform Instances.....	37
7.12	Roadmap	37
8	SCAPE Data Publication Platform	39
8.1	Functional Overview.....	39
8.2	Technical Overview	40
8.3	Roadmap	41
9	Conclusion	42
10	References	43

1 Introduction

The SCAPE project is developing 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 will be 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 are developing a platform tailored towards the automated planning of preservation plans, monitoring of knowledge impacting these plans, and the scalable execution of the preservation plan workflows on large content collections. The majority of development work is broadly divided into a number of key sub-components, Automated Watch, Automated Planning, Preservation Components, the Execution Platform and the Digital Object Repository, with validation of these developments occurring through three testbed repositories.

Automated Watch provides the mechanisms to monitor the content itself, designated user communities and other systems in order to provide actionable triggers for the planning component. Automated Planning provides the means to develop, monitor, execute and evolve preservation plans, the workflows of which, constructed from Preservation Component tools and services, will be executed on the Execution Platform. The Execution Platform provides scalable infrastructure, aiming to enhance the computational throughput and storage capacity of digital object managements systems through varying the number of computational nodes and enabling a parallel data processing approach, imperative to achieving reasonable processing times for large data sets. Execution management of preservation plan workflows and the storage of data sets will be via the Digital Object Repository.

This report provides an overview of the technical status and direction of the SCAPE project, giving details about these main components and how they are expected to interact with one another. In particular it identifies the main interface points between the various architectural components. It does not attempt to capture all background knowledge and experimentation used to drive the technical choices, instead associated background documents and reports will be referenced where appropriate.

This report starts, in Chapter 2, by providing a component-oriented overview of the SCAPE architecture, highlighting the main functional components described above, along with the expected interfaces between them. Each of these main functional components is then described in further detail in chapters 3 - 7, providing relevant functional and technical information, where known, and referencing specifications where these have already been defined. In particular, each component's main interfaces are described, along with details about how it is expected to be packaged and deployed, as well as a brief overview of future SCAPE milestones and deliverables of relevance. Finally Chapter 8 describes, in similar detail, a complementary component relating to the publishing of reproducible experimental results.

1.1 Open Source Development

Where possible, and where it makes sense, existing software and tools shall be developed and enhanced to add the functionality required by SCAPE. Changes should be offered back to the original development branch of the software, where it makes sense to do so, in order to encourage wider community support for maintaining the enhancements and to enable them to persist beyond the scope of the SCAPE project. As a good example, SCAPE have already enhanced Apache Tika™ [40] and successfully pushed these enhancements back into the main Apache Tika™ release.

Some enhancements may not be pertinent for such wider release however, in particular preservation specific enhancements which are not aligned with the original open source tool's agenda or roadmap. In such cases, SCAPE will have to develop and maintain its own fork of the code base, ensuring that this is kept synchronised with the original code base (i.e. the fork should be kept up to date with the original code base).

1.2 Agile Development

Many of the SCAPE software development work packages employ iterative development practices closer to Agile software development methodologies than a more traditional, so called, waterfall methodology. The project is not prescriptive as to the methods individual work packages or institutions employ, preferring to allow them to work as they would normally. However, the deliverables, milestones, and checkpoints from the project's Description of Work favour the release of early, simple prototypes which are refined through iterative cycles of development and testing, as well as integration testing with other components where required.

The main reason for this is the research nature of much of the projects software development. It would be difficult to be confident in a complete, up-front design process, when so many of the activities are trying to establish what is possible in terms of functionality and scale. The other reason is simply that many of the partners are more comfortable working in this manner.

This should be borne in mind when reading this report, as there are areas where the exact manner in which a piece of functionality will be implemented or the precise definition of an API is yet to be finalised. Many of the sub-projects and work packages have produced, or are in the process of producing early iterations of components. Testing of these, both individually and as an integrated whole, will inform and shape the final architecture.

In this spirit, this Architectural Report represents a current snapshot in time of the project architecture. The scope and number of components and component-level interfaces should be considered stable, although the maturity level of these interfaces, and in particular component-internal interfaces, varies. Precise definitions of operations are being refined on an on-going basis, and as such, this document will be updated as progress is made and individual component designs are adapted and finalised. There will be a second official release of the Architecture Document in M30 of the project (Deliverable D2.3).

1.3 Abbreviations

The following abbreviations are used throughout this report:

Table 1: Abbreviations

Abbreviation	Description
AIP	Archival Information Package
API	Application Programming Interface
CQL	Contextual Query Language
CSV	Comma Separated Values
DIP	Dissemination Information Package
DOM	Digital Object Model
DOR	Digital Object Repository
DROID	Digital Record Object Identification
GUI	Graphical User Interface
HDFS	Hadoop Distributed File System
HTTP	HyperText Transfer Protocol
JSON	JavaScript Object Notation
LSDR	Large Scale Digital Repository
METS	Metadata Encoding and Transmission Standard
OAIS	Open Archival Information System
PPC	Parallel Preservation Components
PPL	Program for parallel Preservation Load
PREMIS	PREservation Metadata: Implementation Strategies
REST	REpresentational State Transfer
SCAPE	SCAlable Preservation Environments
SDK	Software Development Kit
SIP	Submission Information Package
SOAP	Simple Object Access Protocol
SRU	Search/Retrieve via URL
SSH	Secure SHell
TCK	Technology Compatibility Kit
URI	Uniform Resource Identifier
WSDL	Web Service Definition Language
XML	eXtensible Markup Language

2 Architecture Overview

This section provides an overview of the SCAPE platform architecture, describing the top-level and sub-level components and, importantly, the interfaces required between them. These components and interfaces are described in further detail in the subsequent chapters.

Figure 1 shows an overview of the top-level components along with the interfaces needed to connect them. For ease of understanding, these interfaces are colour coded to match the component responsible for their implementation. For example, the Digital Object Repository is responsible for the Report, Plan Management and Data Connector APIs.

As can be seen, at a high level the SCAPE project consists of a number of interconnecting components each handling specific aspects of functionality to ensure the preservation of digital objects stored within a **Digital Object Repository (DOR)**. The **Execution Platform** manages and runs parallelised preservation workflows responsible for expedient and reliable execution of preservation actions on some data set within the DOR, and ensures the validity of the outcome. For example, a workflow may migrate all files of one format to another format and ensure that the relevant significant information is maintained, or it may perform file identification and characterisation on a large set of files using a tool such as DROID or Apache Tika™.

Workflow execution is initiated by the DOR through the **Job Execution Service API**. DOR data itself is accessed or referenced (with data access via an externally defined agent) through the **Data Connector API**, with the data either being uploaded directly to the Execution Platform in advance of workflow processing, or accessed directly from storage by the Execution Platform during processing – SCAPE supports either approach. The outputs from such workflow executions will filter back to the DOR (i.e. files of new formats) by way of the **Data Connector API** and/or be stored and published in the **SCAPE Data Publication Platform** repository through the **LDS³ API**.

Both the DOR and SCAPE Data Publication Platform repositories are used as **Sources** of information for the **Automated Watch** component (known as SCOUT), which builds and monitors a view of the world based on its input Sources in relation to institutional policies. Automated Watch constantly updates information from its Sources either by periodically polling for information on the **SourceAccess Pull API**, or by receiving pushed information via its **Source Push API** (the DOR also provides information via the **Report API**). Watch can then reassess its world view and notify the **Planning Component** if some predefined threshold to some criterion is met, for example, the cost of some specific migration software may have reduced to an acceptable level, or the number of files of a particular format may have increased to a level where an appropriate preservation planning action should be triggered. Such criteria are set by the Planning Component through the **Watch Request API**, assessed by the Watch Component, with corresponding actions triggered through the **Notify API**. Assessments too complex for the Watch Component's Boolean logic may be performed (by an operator) through the **External Assessment API**. The **Planning Component** is also responsible for creating, monitoring and testing preservation plans. It builds upon the PLATO planning tool [27] which provides a decision support tool for assessing the most appropriate preservation actions to perform on a content collection, along with audit evidence documenting the decision making procedure used to create the associated preservation plans. SCAPE aims to build upon this tool through use of the Automated Watch Component, machine interpretable policy models and content profiles to help automate the planning process.

Preservation Plans are built from preservation tools and workflows (Components) defined and stored in the **SCAPE Component Catalogue** (myExperiment [7]). Workflow Components themselves are



created in the *Taverna Modelling Environment* [23] and registered in the Component Catalogue via the *Component Registration API*. Components can then be accessed through the *Component Lookup API* and used to generate preservation plans in the Planning Component. Here these plans are assessed and "successful" plans can be uploaded to the Digital Object Repository, using the *Plan Management API*, for execution.

Plans uploaded to the DOR can be managed through the *SCAPE Plan Management GUI* which also utilises the Plan Management API. It is via this GUI, that preservation plans can be initiated on the Execution Platform (through the Job Execution Service API) for some content collection held in the DOR. A *Loader Application* provides the ability to initially load data in to the DOR in accordance with the SCAPE Digital Object Model for data.

Figure 2 captures these components in greater depth, indicating sub-components that perform the necessary functionality. Components coloured light-red and with underlined names are Sources to the Automated Watch component and therefore implement the Sources' API (unless otherwise stated in the discussion that follows). All these components, sub-components and interfaces will be described in detail in the following chapters.

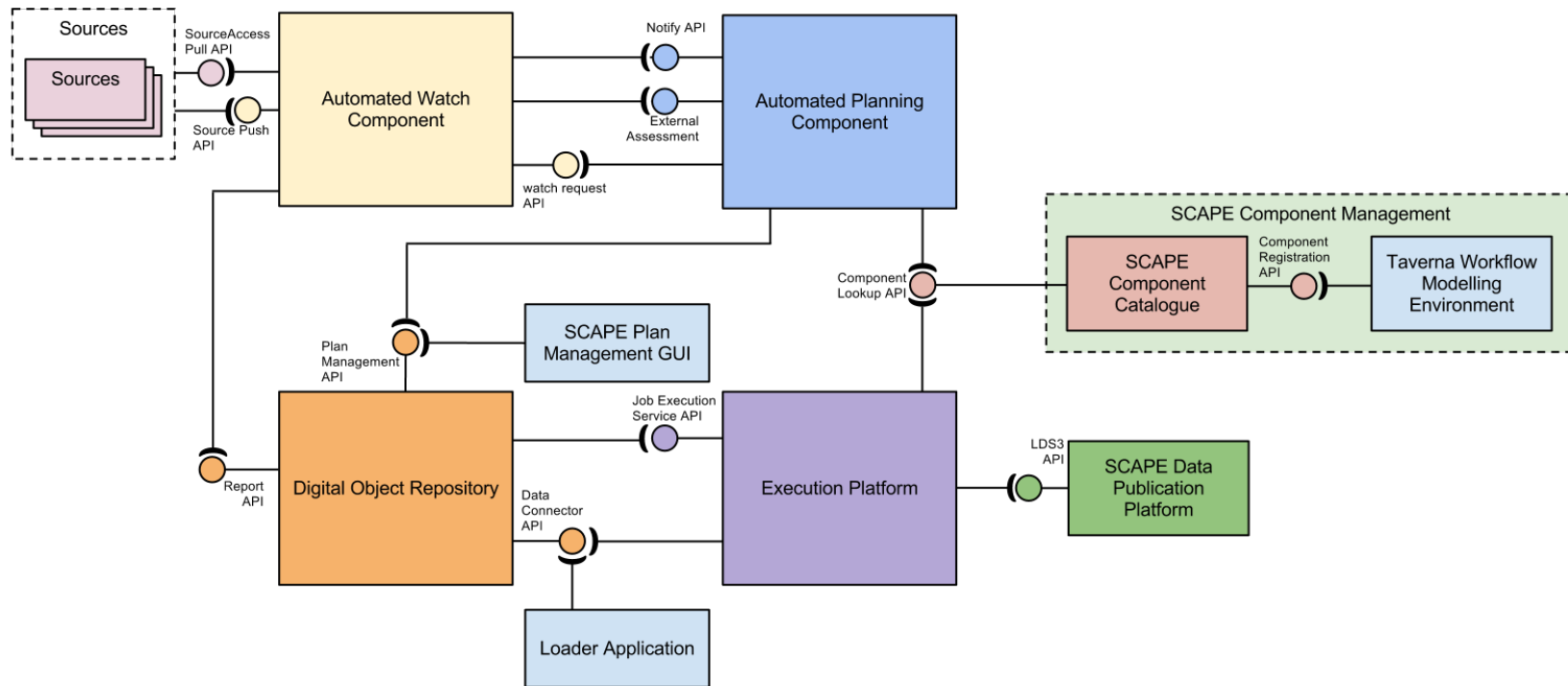


Figure 1: Colour coded top-level architectural diagram highlighting which component is responsible for which APIs

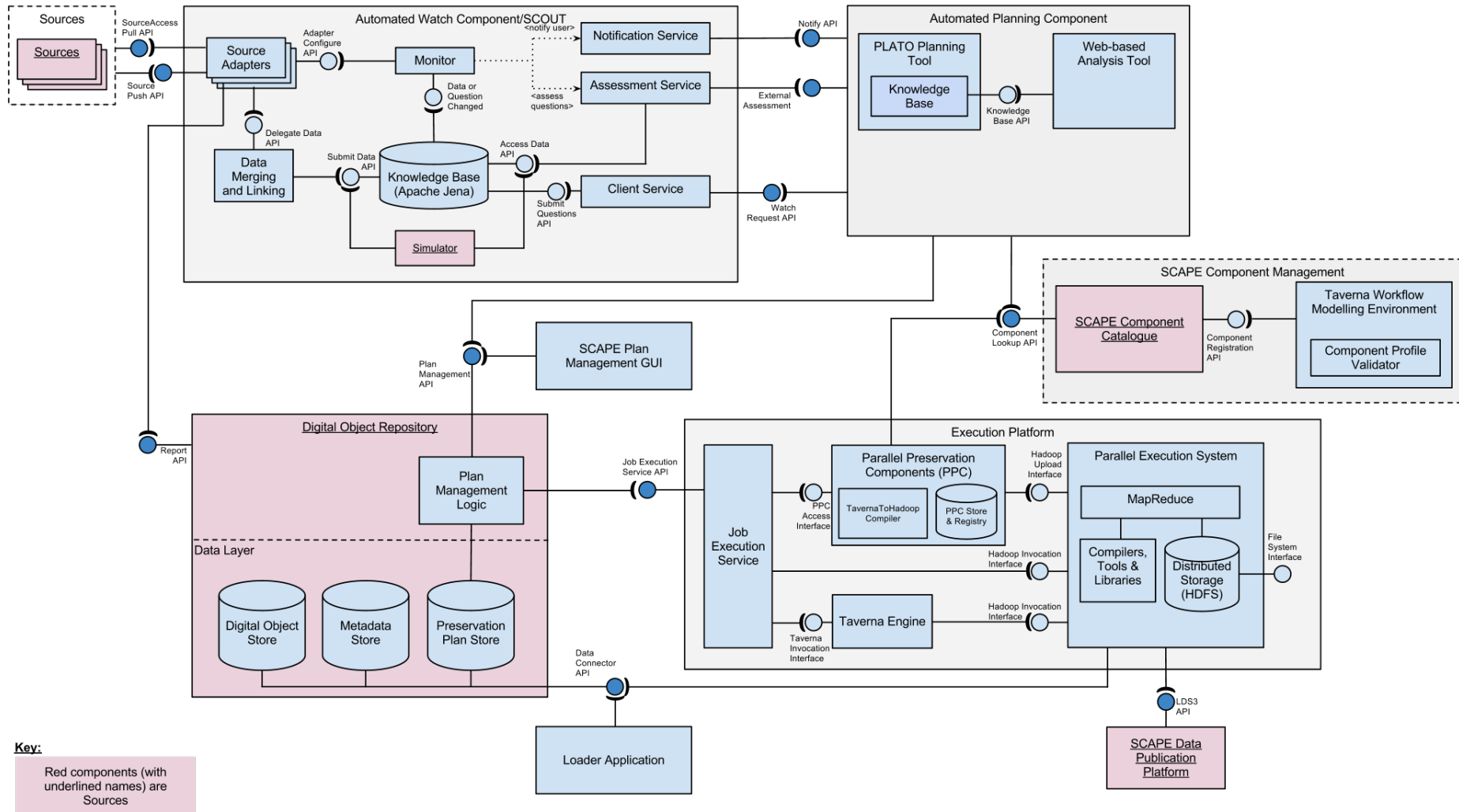


Figure 2: SCAPE component architectural diagram

3 Automated Watch Component

3.1 Introduction

The aim of Automated Watch is to substantially improve automated support for effective digital preservation watch. Specifically its aims are:

- To increase the breadth and scope of collected digital preservation information.
- To normalize and structure gathered information into a queryable Knowledge Base.
- To allow both human and automated systems to query and monitor information in the Knowledge Base.
- To develop software components that monitor information in the Knowledge Base for significant events.

3.2 Functional Overview

The Automated Watch component is an automated information gathering and monitoring system that:

- Gathers information relevant to digital preservation activities from potentially any source of information of interest to the planner, e.g. repositories, technical registries.
- Allows software agents and human operators to add information relevant to digital preservation activities.
- Allows software agents and human operators to ask questions about gathered information through Watch Requests.
- Provides an automated monitoring system that looks for changing information and re-answers questions to a schedule.
- Assesses answers against conditions and triggers submitted in Watch Requests, and alerts external agents when conditions are met.
- Provides a Repository Simulator that analyses the information gathered from a user's repository and projects its future state facilitating timely Preservation Planning for upcoming requirements.

3.3 Technical Overview

The Automated Watch component comprises a number of sub-components that each provides specific functionality to achieve the goal of monitoring the "state of the world" through various Sources of information and providing notifications to the planner. Information is gathered via pull adaptors, developed to normalize and aggregate data from external sources, alternatively sources can push information to the Automated Watch system via the push source API.

Adding new sources to the system means developing a compliant adaptor. A Data Merging and Linking sub-component adds provenance information and links to equivalent entities and properties held in the knowledge base. The knowledge base also contains pre-defined questions that can be answered from information held in the knowledge base. A planner can submit a Watch Request either synchronously or asynchronously, via the Watch Request API in order to query specific measurements of interest and receive notification when conditions are met. The query, conditions and means of notification are all parts of a Watch Request. Synchronous Watch Requests are used to query for a specific measurement at a specific point in time, blocking the requesting client until the

response is returned. Asynchronous requests tell the Automated Watch component to monitor for changes in specific measurements (by specifying Conditions) triggering a Notification such as an email, to the requesting client when such a change is detected. This approach does not block the requesting client.

Detailed design information for the Automated Watch system can be found in SCAPE D12.1 - Identification of triggers and preservation Watch component architecture, subcomponents and data model [41].

The following sections discuss the various sub-components involved in the Automated Watch Component and how they interact.

3.4 Sources

Although not strictly a part of the Automated Watch component, Sources are described here to aid understanding of the Automated Watch sub-components. A Source represents specific aspects of the world and provides measurements of the properties associated with it, and can be internal or external to the project. Key sources currently considered are:

- Format Registries
- SCAPE Preservation Components catalogue (MyExperiment)
- Policy models
- Repositories
- Experimental Results
- Content Profiles
- Human Knowledge
- Web Browser snapshots (being developed within Preservation Watch)
- A Repository Simulator that projects a repository's future state based upon repository trends (being developed within Preservation Watch)

Sources are coloured pink in Figure 1 implying that, although they may also connect to other SCAPE components, they will interact with Source Adapters through either the **Source Access Pull API** or the **Source Push API**. An exception would be the Digital Object Repository which implements a **Report API** for interaction with a relevant Source Adapter.

3.5 Source Adapters

3.5.1 Functional Overview

A Source Adapter gathers information from a Source and transforms it to the Entity/Property model adopted for the Knowledge Base. There are two approaches to achieving this, push or pull, the choice of which to use will depend on multiple factors such as whether the Source is Watch component agnostic or whether it is possible to create software to run on the Source.

The Source Adapters employed should map to the Sources being used. The SCOUT preservation watch project contains two reference adaptors described below.

3.5.2 PRONOM Adaptor - A Reference Format Registry Adaptor

There is a PRONOM source adaptor developed as part of the Preservation Watch process. The adaptor queries the PRONOM Linked Data SPARQL endpoint and transforms the returned JSON into

Entities/Properties for passing on to the Merging and Linking component. The Adaptor code is part of the SCOUT project [SW1].

3.5.3 C3PO Adaptor - A Reference Content Profile Adaptor

C3PO (Clever, Crafty Content Profiling of Objects) [SW2] is a content profiling tool. It doesn't perform any characterisation; instead it parses output from the FITS tool [43] and aggregates into a MongoDB [44] document database. There is also a tool that retrieves FITS records from the RODA repository for consumption by C3PO. C3PO also provides a web based tool to view and the aggregated data and a REST API for retrieving aggregated profile data.

The C3PO adaptor reads and parses the XML data retrieved from the C3PO REST API and retrieves a subset of the content profile data. The data is converted into the Automated Watch Entities and Properties before being passed to the Data Merging and Linking component, which enriches the data before inserting it into the Automated Watch Knowledge Base.

Both of these adaptors are in an early stage of development, but provide examples of how to develop a plug in adaptor for the Automated Watch system.

3.5.4 Other Source Adaptors

While there could be adaptors written for many different sources of information, the following will be developed as part of Automated Watch:

- **RODA Adaptor & eSciDoc Adaptor:** These are both repository adaptors that will gather information about a user's collection, e.g. Content Profiles.
- **Component Catalogue Adaptor:** The component catalogue adaptor will gather data from the SCAPE Preservation Component Catalogue via the Component Lookup API. The information gathered in the Knowledge Base will allow planners to find tools that meet their planning requirements, or alert them when tools that provide new functionality required for preservation activities that had not previously been available.
- **Policy Model Adaptor:** A policy model adaptor is assumed to be the form of the Automated Watch component required to incorporate the Machine Interpretable Policy Model into the Automated Watch Knowledge base. This source will provide data that allows the planner, or the Automated Planning system to monitor an organisation's repository contents checking for policy violations, and receive notifications if they occur.

3.6 Source Adaptor Manager

3.6.1 Functional Overview

In order to control the scheduling of Source Adaptors used by a particular Automated Watch instance a Source Adaptor Manager is being developed. This manager will allow an operator to:

- Install new adaptors.
- Upgrade installed adaptors.
- Enable or disable installed adaptors.
- Manage the scheduling of adaptors, i.e. how often the external sources are queried.

3.6.2 Technical Overview

This component provides a web API for the life-cycle management of the Source Adaptors and a web based front end that are part of the SCOUT project.

3.7 Data Merging and Linking

3.7.1 Functional Overview

This sub-component provides a data processing layer, sitting between the Source Adaptors and the Knowledge Base. The source adaptors convert data to fit the internal data model, but different source adaptors may present contradictory or incompatible data. The merging and linking component further process incoming data by:

- Adding provenance information to data
- Resolving inconsistencies between data sources.
- Providing additional cross-references between entities and properties.

It is the addition of the cross-references that will enable rich queries of the Knowledge Base.

3.7.2 Technical Overview

The component retrieves properties and entities from Sources using the Source Adaptor's plugin interface. It also provides the Push Source API to push information to it, and makes use of the Knowledge Base's **Submit Data** API to submit data for permanent storage in the Knowledge Base.

3.8 Knowledge Base

3.8.1 Functional Overview

The Knowledge Base is responsible for storing representation information about the world using a model based on Entities and Property Values. Ultimately, each Entity describes a specific set of values that are measurements of each Property at a specific moment in time. For example, for a "format" Entity, relevant Properties might be "name" (e.g. JPEG2000), "version" (e.g. 1.0), or "tool support" (e.g. limited); over time, the tool support for JPEG2000 may increase, therefore at a later point a new Entity may indicate "tool support" as "widespread". Relevant internal APIs are provided to store and retrieve data from the Knowledge Base, namely **Submit Data** and **Access Data**.

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

3.8.2 Technical Overview

It is planned to use RDF Linked Data as the model for storing data in the Knowledge Base, as this enables a generic and more flexible data representation than a relational data model.

The Knowledge Base uses Apache Jena [45], a Java framework for storing and querying large RDF datasets. Jena also provides support for OWL ontologies and a rule-based inference engine for reasoning with RDF and OWL data sources, useful for framing and answering the Watch Request questions. The SPARQL query language is planned to be used to represent Watch Request questions.

3.9 Automated Watch Client

3.9.1 Functional Overview

This is a web interface which provides the following functionality for planners:

- The manual addition of information to the knowledge base.
- Browsing of the knowledge base.
- The Submission of Watch Requests to the Automated Watch system.

3.9.2 Technical Overview

The Automated Watch Client is a Java Web application, packaged as part of the Automated Watch web application. The GUI provides an interface that allows the user browse the information in the knowledge base and to add new information through the Automated Watch Push API.

The GUI also provides a means by which human operators can submit Watch Requests. Watch Requests consist of:

- One or more pre-defined Questions that assess some aspect of the world through the querying of the Watch database.
- One or more Triggers that define Boolean conditions to be tested against the answers to the Watch Request's question set.
- One or more notifications that will alert external agents if the trigger conditions are met.

3.10 Monitor Services

3.10.1 Functional Overview

Monitoring services observe one or more information sources and recalculate answers to questions held in the Knowledge Base, when the results rely upon external information that has changed. These questions are predefined points of interest related to the information gathered from the sources. An example based upon a Component Catalogue source adaptor might be the number of tools fulfilling a particular criterion. As new tools are added to the catalogue the information will be gathered by the source adaptor, which in turn would be picked up by a Monitor Service. The service would then recalculate the answer to the question based upon the new information in the knowledge base.

3.10.2 Technical Overview

The Monitor sub-component provides a mechanism for continuously watching the Knowledge Base for changes to specific Watch Requests the client is interested in. To do this, it provides a **Data or Question Changed** interface for being notified about changes to the underlying data or the Watch Requests themselves. Upon receiving such an update, this sub-component will identify which Watch Requests require re-evaluation and instigate this re-evaluation through the **Assessment Service**.

3.11 Assessment Service

3.11.1 Functional Overview

The assessment service is responsible for evaluating Watch Requests utilising the latest information from the Knowledge Base, and the conditions associated with the Watch Request via the triggers.

There are two types of assessment, a preliminary assessment which is a simple test of a Boolean condition contained in a trigger. This may be all that is required for basic Watch Requests, if the trigger conditions are met by simple assessment test then the trigger will fire and notify external agents that an external assessment is required.

3.11.2 Technical Overview

The Assessment Service is a part of the Automated Watch Java Web Application (the SCOUT project). Access to the Knowledge Base is provided by the internal **Access Data** interface, and the information received is compared against a Watch Request Trigger to determine if a significant event has occurred. In many cases conditions to be assessed will be more complex than this requiring an external assessment service such as that offered by the Automated Planning Component through its **External Assessment** API. This would allow an existing preservation plan to be re-evaluated in the light of the new information, and assess whether the new state required action.

3.12 Notification Service

3.12.1 Functional Overview

The notification service is responsible for informing external entities of significant events as defined by the monitoring and assessment services. When the Monitor sub-component detects a significant event, based upon the questions and conditions stored in the Knowledge base, the Notification Service is used to alert interested parties. An interested party might be a human planner informed by email, or a software agent, typically the Automated Planning component.

3.12.2 Technical Overview

The Notification Service is again being developed in Java, as part of the automated watch SCOUT project. The notification component is extensible to allow different types of notifications to be offered, for example email, or HTTP API.

3.13 The Repository Simulator

3.13.1 Functional Overview

The Preservation Watch work package is also developing a repository simulator. This component will analyse repository metadata held in the Automated Watch knowledge base, and project the future state of the repository. Trends that may be detected might be accelerating storage requirements, or an increasing number of items in a particular format. Information about computational resources required to execute a preservation action across a set of content might also be analysed to establish how long it might take to run the action, or indeed establish if such a course of action is feasible.

3.13.2 Technical Overview

The Repository Simulator is a Java component, developed as part of the SCOUT Web application.

3.14 Required Interfaces

The Automated Watch component implements two external facing APIs: the Push Source Adaptor API and the Watch Request API.

3.14.1 Push Source Adaptor API

The Automated Watch Push API provides a means for third party software agents to add information to the Automated Watch Knowledge base without the development of a Source Adaptor. Note that push sources will not be controlled by the Source Adaptor manager, so that scheduling, and indeed enabling / disabling an unwanted push sources will have to be done by other means. The API may also be used internally by the Automated Watch Client to add new information via the web front end.

In the push model, the Source will send information to the Automated Watch component as and when it becomes available. Software must be developed for the Source component to achieve this, which in some circumstances is not possible. The pull model ideally relies on the Source component providing a network accessible API to enable a relevant Source Adapter to request information directly, most likely on a periodic basis, however if no such API exists, then the adapter will have to extract information from the format made available by the Source (for example, HTML parsing of a web page). The frequency with which data is requested by a Source Adapter is controlled by the Monitor sub-component through the internal **Adapter Configure** interface.

3.14.2 Watch Request API

This API will be used by external software agents to submit Watch Requests to the Automated Watch system. Typically the software agents will be:

1. The Automated Watch Client GUI.
2. The Automated Planning System.

A Watch Request is made up of a number of pre-defined questions, drawn from the Automated Watch knowledge base, and a number of triggers. Triggers are assessed against questions and, if the trigger conditions set in the Watch Request are satisfied, the planner or software agent is notified, for example by email. These questions and conditions are defined as SPARQL queries.

Both APIs are being implemented as RESTful services deployed with the Automated Watch Java Web Application.

3.15 Packaging and Deploying

The Automated Watch System is Java Web Application built from the GitHub Open Planets SCOUT Maven Project [SW3] and deployed as a Web Application Resource. The project relies upon the JBOSS Java EE 6 library and requires a dedicated JBOSS 7 server, rather than a Tomcat Servlet. RESTful services are provided through Jersey [46] an implementation of REST services for Java, and part of the GlassFish project [47].

3.16 Roadmap

Table 2: Upcoming Automated Watch Component Milestones/Deliverables

Milestone/Deliverable	Description	Due
MS55	First prototype of the simulation environment	M20
MS56	First version of the preservation watch core services	M22
MS57	First prototype of the watch component delivered including adaptors for repositories, and Web content	M28
D12.2	Final version of the Preservation Watch Component due	M38
D12.3	Final version of the Simulation Environment	M42

4 Automated Planning Component

4.1 Introduction

With current state-of-the-art procedures in digital preservation we can define organisational constraints and we can create plans that treat a homogenous sub-set of a large repository. PLANETS defined a Preservation Plan as follows:

“A preservation plan defines a series of preservation actions to be taken by a responsible institution due to an identified risk for a given set of digital objects or records (called collection). The Preservation Plan takes into account the preservation policies, legal obligations, organisational and technical constraints, user requirements and preservation goals and describes the preservation context, the evaluated preservation strategies and the resulting decision for one strategy, including the reasoning for the decision. It also specifies a series of steps or actions (called preservation action plan) along with responsibilities and rules and conditions for execution on the collection. Provided that the actions and their deployment as well as the technical environment allow it, this action plan is an executable workflow definition.”[48]

PLANETS also produced a preservation planning methodology, a structured workflow for creating, testing and evaluating preservation plans. The PLATO planning tool [27] developed within PLANETS follows this workflow to build preservation plans. PLATO produces an executable preservation plan along with audit evidence documenting the decision making procedures used in creating the plan[49]. However the plans were:

- Largely constructed manually, which could be a time intensive procedure.
- Were not normally applicable to all of an organisations holdings, but were restricted to a, normally homogenous, sub-set of a collection.
- Were not deployed and executed automatically in a repository.
- Had to be monitored manually for changes in best practice, collection profile, etc.

Further no mechanism exists to relate preservation policies to preservation plans, correlation has to be done manually.

4.2 Functional Overview

The goals of SCAPE are to provide an automated planning component that is informed by:

1. The accumulated knowledge of previous preservation plans.
2. An organisation's digital preservation policy.
3. An organisation's digital collections.
4. Other queries performed on the Automated Watch Knowledge Base, e.g. queries of File Format Registry information.

The Automated Planning component comprises of :

- A new version of the PLATO Planning Tool [27]Building upon the existing PLATO tool but using the Automated Watch Component, the Policy Model and content profiles to automate the creation of preservation plans, and the management of the plans..

- A machine interpretable model of preservation policy elements. 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.

4.3 Technical Overview

The Automated Planning Tool is a Java Enterprise Application that is deployed on a JBOSS server [51]. It provides a web GUI that takes 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 model of low level, actionable policies rather than a software component. Policy models will be placed in the Automated Watch Knowledge base via a Source Adaptor, or possibly the use of a dedicated loader.

4.4 Automated Planning Tool

4.4.1 Functional Overview

The SCAPE Planning Component continues the development of the PLATO Planning tool used in the PLANETS project [27]. As described the PLANETS PLATO tool produces executable preservation plans to an established preservation planning methodology. The process of producing these plans had various shortcomings described in the introduction.

The planning tool addresses these by adding automated support throughout the established process. The manual GUI will still be supported for low level plan editing where necessary, or indeed if the user prefers it. However modules are being developed that will populate forms with recommended content during each planning step. This will mean a lighter planning process with less labour intensive form filling, and a lighter GUI.

A fast-track planning mode will also be introduced; this reduces user choice and reduces the 14 planning steps to 4 phases, with a single GUI page for each phase.

4.4.2 Technical Overview

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

- a Policy Model and a Watch Trigger provide the planner with all of the information required to describe the plans institutional context, the first planning step.
- an XML content profile can be uploaded that completes the PLATO "define samples" page.

The business logic in PLATO has been refactored so that flexible configurations of the workflow are quicker to implement, giving the option to create the light weight planning options described in the Functional Overview.

The PLATO tool developed in Planets contained an embedded tool execution engine, known as minimee. This allowed users to test and compare different tools, or different configurations of a tool against their planning requirements. While this will be part of the initial iterations of the planning tool, the aim is to integrate with the SCAPE Component Catalogue to encourage the use of established or experimental Taverna workflows developed by the Testbeds. Currently minimee offers real measurement of resource usage by tools, information not yet available from the Component Catalogue, or Data Publication Platform. As richer information becomes available from SCAPE components it is envisaged that the minimee platform will be switched off.

The automated planning work package is responsible for the development of the software component that imports instances of the machine interpretable policy model into the planning tool.

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. The Planning Tool holds a database containing preservation plans created by different institutions, these are processed and anonymised, before being 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 will not be part of the planning workflow.

4.6 Machine Interpretable Policy Model

4.6.1 Functional Overview

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

Only the control policies are guaranteed to be represented in the machine interpretable policy model. The development of the machine interpretable policy model is led by the development of a catalogue of policy elements.

4.6.2 Technical Overview

The policy element catalogue provides a semantic representation of generic policy elements that is understandable by preservation systems. The initial version of the policy catalogue lists a set of Guidance Policies which, by definition, will not appear in the machine interpretable model in their full form. Instead these must be broken down into sets of Procedural Policies, which in turn will be represented by sets of Control Policies that will be used to create the machine interpretable policy model. The iterative process of refining the catalogue will be undertaken by using the catalogue to express the real preservation policies of three partners representing the needs of Large Scale Digital Repositories, Web Archives, and Scientific Data Sets. Once validated the catalogue will be used to develop the machine interpretable model.

The machine interpretable policy model provides a source for the Automated Watch system and will inform the Automated Planning system. Standard tools such as RDF/OWL [50] will be used to define the terms used to describe and represent Control Policies and support policy reasoning. Similarly to the catalogue, the policy model will undergo an iterative process of testing and refinement while been used to model the various Testbed scenarios.

One purpose of the policy catalogue is to guide organisations in creating their own complete policy model. The iterative processes described above will be used to improve the process of creating institutional policies by either extending an existing tool, or developing a simple editor or a domain-specific language that is easier to write, if this is feasible. The process is complex, the aim is to simplify it as much as possible.

4.7 Required Interfaces

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

There are no recognised interfaces developed as part of the policy modelling workpackage. The Automated Watch and Automated Planning components are both responsible for developing software components that will interpret the model and base decisions upon the policy elements. The Policy Modelling work package is responsible for ensuring technical interoperability between these components and the policy model.

4.7.1 Notify API

The Notify API will be used by the Automated Watch component to inform the Planning Component when significant events occur, that have been defined through Watch Requests. Examples would be a notification that a plan requires updating, due to a change in the state of a repository, or that a new tool is available that provides previously unavailable functionality required by a preservation plan.

4.7.2 External Assessment API

The External Assessment API will be used by the Automated Watch component to make external assessments that are more complex than the state of 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 [51] application packaged as a Java Enterprise Application Resource.

There is a central instance of the Planning Tool hosted by TUWIEN [27], which is currently running version 3.0.1. The latest release will continue to be hosted here.

Organisations wishing to host their own PLATO [27] instance would first require a JBOSS server that has been installed and set up separately on which to host the application.

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.

There is a GitHub project [SW4] where the semantic model of low-level Control Policies is being developed. The project contains:

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

4.9 Roadmap

Table 3: Upcoming Automated Planning Component Milestones/Deliverables

Milestone/Deliverable	Description	Due
MS61	Initial version of automated policy-aware planning component	M18
D13.1	Final version of policy specification model	M30
MS62	Automated policy-aware planning component v2 with full lifecycle support	M32
D13.2	Catalogue of preservation policy elements	M36
MS63	Report on compliance validation	M40
D14.2	Final version of automated policy-aware planning component	M42

5 SCAPE Component Management

5.1 Introduction

Preservation Plans developed and tested within the Planning Component utilise SCAPE Components to provide preservation tools and actions. A SCAPE Component is a Workflow designed for execution on the SCAPE platform that, most likely, wraps a tool execution. For example, a SCAPE Component may exist to run DROID or Apache Tika™ file identification over a digital object. These Components are created using the Taverna Workbench and stored in the SCAPE Component Catalogue.

5.2 Functional Overview

SCAPE Components provide a consistent way to represent preservation actions so that they can easily be discovered and included into larger Preservation Plan workflows. The Taverna Workflow Modelling Environment provides a means to create these components, enables the addition of relevant profile information, and ensures that these Components have consistent inputs and outputs as defined by their respective Component Profile (to facilitate connectivity into larger workflows).

These Components are stored in the SCAPE Component Catalogue for:

- i) Monitoring by the Automated Watch component (it is therefore an Automated Watch Source);
- ii) Discovery and use by the Planning Component;
- iii) Compilation into Parallel Preservation Components for execution by the Execution Platform.

5.3 Technical Overview

The SCAPE Component Management module shown in Figure 1 and Figure 2 should be considered as an abstract entity surrounding the two main components: the SCAPE Component Catalogue, and the Taverna Workflow Modelling Environment.

SCAPE Components will be Taverna Workflows that adhere to a specific SCAPE Component Profile. Taverna Workbench will be utilised to create these workflows and will be updated to include a Component Profile validator. The SCAPE Component Catalogue storing the SCAPE Components will use the myExperiment web portal [7], and will present a RESTful APIs for registration and discovery of Components.

5.4 SCAPE Components

SCAPE is tasked with producing tools and preservation actions that address scalability issues and can be used on the SCAPE infrastructure. Part of this work is to develop and enhance the tools for scalable preservation actions, so for example, whilst many tools already exist that can be applied to aspects of digital preservation, such as JHOVE, DROID, FIDO, these existing tools have not been designed with the SCAPE Execution Platform in mind; they need adapting and enhancing so that they work effectively with the SCAPE platform. In addition to this is the potential need for new or enhanced tools as required by new workflows, for example to enhance Quality Assurance for some specific preservation action on some specific dataset (e.g. audio files from radio broadcasts).

Developed and enhanced tools are deployed within SCAPE as SCAPE Components. These are Taverna workflows that adhere to a specific SCAPE Component Profile.

5.4.1 Workflow

A workflow is a sequence of steps or operations on some input that execute according to the defined flow and combine to perform some complex operation, for example a workflow may take a file location URL as input and pass this to DROID to identify the specified file, the output of which is returned to the user. Building upon this, this workflow may be used as part of another workflow where the identification output is used to control flow within the larger workflow, for example, if a file is identified as an image file, it may undergo optical character recognition before and after file format migration with a comparison to provide some metric on the quality of the migration.

SCAPE use Taverna Workflows[23], making use of the Taverna Workflow Modelling Environment (See Chapter 1.1) for users to produce workflows using a GUI. In general, these workflows can invoke SOAP/WSDL or REST web services, local Java code, external tools via SSH, or other sub-workflows. SCAPE has developed a "toolwrapper" to wrap local tools as web services for use within workflows [SW7].

Taverna Workflows are hosted in the SCAPE Component Catalogue (myExperiment [8]), where they will be a *Source* for the Automated Watch component and can be searched for and utilised by the Planning tool. SCAPE Component workflows required by a Preservation Plan workflow for execution can also be discovered by the SCAPE Execution Platform and transformed into Parallel Preservation Components (see Chapter 7.4)..

5.4.2 Component Profiles

To enable interoperability between tools, automation of preservation processes, and discoverability by Planning and Watch, SCAPE Components need to provide a standardised interface. Such an interface is provided by the Taverna Workflows adhering to defined input/output interfaces and by annotating them with a common, standardised vocabulary. Defined combinations of interfaces and annotations form SCAPE Preservation Component Profiles; SCAPE have already defined a number of these profiles, extending common ports and annotations, for: migration action components; characterisation components; quality assurance object comparison components; quality assurance property comparison components; validation components; and executable plans. These Profiles are defined [15].

A Profile has four different areas to check:

1. **Input ports:** Expected input ports of the workflow;
2. **Output ports:** Expected output ports of the workflow;
3. **Taverna Activities:** Taverna activities that must be present for the workflow, e.g. external tool services that are used;
4. **Annotations:** Workflow level annotations.

As an example, consider a Migration Action Component, with respect to annotations it builds upon the common elements such as the Component's name, version and ID (a full list can be seen [15]) with details about the Migration Paths that this component supports, i.e. the file types this component can migrate from and to. The profile also specifies the particular input ports that must be defined, where *path_from* and *path_to* specify the path of the file to migrate and the path to migrate it to, as well as a *parameter* input port to detail any specific options to apply (e.g. tool specific command line options/flags). Output ports are also specified, specifically the *path_from* and *path_to*, which have the same meaning as before. Finally the Taverna Activity defines the external tool service used to perform the migration.

The Taverna Workflow Modelling Environment will provide a *Component Profile Validator* to validate these components against the defined profiles.

5.5 SCAPE Component Catalogue

5.5.1 Functional Overview

The SCAPE Component Catalogue stores SCAPE Components to enable their discoverability by the Planning Component and the Execution Platform, and for monitoring by the Watch Component.

5.5.2 Technical Overview

The myExperiment platform [7] will be used to store SCAPE Components. Appropriate modifications will be made to its REST API [16] to enable registration of components and discoverability.

5.6 Taverna Workflow Modelling Environment

5.6.1 Functional Overview

The Taverna Workflow Modelling Environment comprises a graphical workbench for creating and modifying workflows, adding metadata and component profile information, and registering Components in the SCAPE Component Catalogue.

5.6.2 Technical Overview

The Taverna Workbench [23] will be used and is a Java based open source tool for designing and executing scientific workflows. It will require modification to enable validation of components against the SCAPE Component Profiles and to easily enable registration of Components to the SCAPE Component Catalogue.

5.7 Required Interfaces

The SCAPE Component Catalogue **must** implement two APIs: the *Component Lookup API* and the *Component Registration API*. Both these APIs will be integrated into myExperiment's (the SCAPE Component Catalogue's) REST API [16].

5.7.1 Component Lookup API

The Component Lookup API provides a mechanism for the Planning and Execution Platform components to discover and access SCAPE components. This has yet to be defined.

5.7.2 Component Registration API

This API provides a means to register SCAPE Components (Taverna Workflows) in the SCAPE Component Catalogue. This will be achieved using the existing myExperiment plugin for Taverna which utilises the functionality defined in [17].

5.8 Packaging and Deploying

SCAPE will use a dedicated version of the myExperiment platform [7] to host and share SCAPE Components. This web-based site enables a common, accessible location for uploading, discovering and retrieving SCAPE components without the need for institutions to separately install their own catalogue.

The Components themselves are workflows which wrap locally and/or remotely installed software applications (tools), therefore a major challenge is how to make these workflows, and more specifically the tools they rely upon, available on the scalable Execution Platform. More specifically, any such solution should ensure reliable and convenient installation across multiple computing nodes that form the Execution Platform, including automatic resolution and installation of all necessary tool dependencies.

5.8.1 Packaging and Deploying Component Tools

As defined by the Component Profiles, SCAPE Component workflows must provide annotations that indicate which tools they depend upon. This information can be used to indicate which tools must be deployed to the Execution Platform.

To enable easy distribution, installation and updates to the tools that SCAPE Components depend upon, the Debian (Linux) software packaging and package management system will be employed. This provides a standardised and integrated way to manage and install software ensuring that all dependencies of that tool are also installed. Through this process an end-user can easily install software through a single command (or click in a GUI), passing responsibility to the package manager to download the package, resolve dependencies, and install the software. Such a system 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 [26].

5.9 Roadmap

This section briefly outlines upcoming and future work that needs addressing and relates to the Taverna Workflow Modelling Environment, SCAPE Components, and the SCAPE Component Catalogue. Table 2 provides a summary of these upcoming milestones and deliverables.

The Taverna Workbench is available for use already, however enhancements are needed for SCAPE use, in particular, capturing provenance information about workflow runs, which should be recorded and persisted in order to perform provenance analyses on the data in the main repository, for example enabling a trace of the set of transformations applied to an image. The requirements of this work are captured in [22] and will be implemented in Taverna. Component Profiles also need capturing and validating to ensure that SCAPE Components present standardised interfaces to facilitate combination into larger workflows.

SCAPE Components will be stored and shared via the SCAPE Component Catalogue, based on the myExperiment site, with design and implementation documentation due in M24 and in deliverable D7.3 (M40). Of particular importance is the Component Lookup API, which needs defining in coordination with the Execution Platform and Planning components.

Table 4: Upcoming SCAPE Component Catalogue Milestones/Deliverables

Milestone/Deliverable	Description	Due
MS40	Design and Implementation of the Component Catalogue	M24
MS41	Final Preservation Workflow Sharing Platform	M42
D7.1	Design of Provenance Component	M20
D7.2	Workflow Modelling Environment	M36
D7.3	Design and implementation of the preservation component catalogue	M40

6 Digital Object Repository (DOR)

6.1 Introduction

A Digital Object Repository (DOR) is an OAIS compliant repository [28]¹, providing a data management solution for storing the content and metadata of digital objects as well as preservation plans.

6.2 Functional Overview

The DOR is responsible for helping its user community deposit, curate, preserve and access digital objects.

Digital objects themselves are comprised of content - the actual data to be preserved such as images or audio/video files - and metadata representing the technical, administrative, structural and preservation information. Therefore the DOR is responsible for storing the content and the metadata of a digital object, as well as maintaining the semantic relationship between digital objects.

The DOR also enables management and invocation of Preservation Plans on the SCAPE Execution Platform.

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 SCAPE Digital Object Repositories defined APIs. Implementation of these will enable any repository to be used within SCAPE, enabling other components to access/reference the data, invoke and monitor Preservation Plan executions and gather reporting information.

The DOR exposes its services through three well-defined HTTP APIs detailed in section 6.10: the Data Connector API, the Report API and the Plan Management API. The Data Connector API enables access to the digital objects and preservation plans. The Report API enables the Watch Component to monitor a repository, and the Plan Management API enables management and invocation of Preservation Plans.

There are some performance considerations regarding accessing digital objects through a HTTP interface that should be considered however. In particular, the request duration overhead when requesting binary content via HTTP varies depending on the size of the requested content. With small sized content the overhead is negligible, however with large binary content the overhead becomes significant. To accommodate this, SCAPE defines two strategies, letting stakeholders make the most appropriate choice to suit their needs: a *Managed Content* approach whereby files are accessible only through the Data Connector API; or a *Referenced Content* approach whereby files are stored in a file system directly accessible by the SCAPE platform and the Data Connector API merely passes references to this content. The former approach is not suited to large amounts of data or where storage and computation are geographically separated because of the IO overhead for data

¹ The 2003 OAIS reference model has been superseded by the 2012 version [29], compliance to which has yet to be determined.

retrieval; the latter, on the other hand is suitable for large files as they can be handled (by reference) without having to moving them between machines, however it does mean that the storage file system must be directly accessible to the platform.

In order to provide efficient computation, the DOR may store (or replicate) its content directly to the Execution Platform's storage system. It may also store outcomes of workflows (or parts thereof) that have been executed against the DOR's contents, so it is vital that the DOR employ a suitable data model and scalable object store. Transfer of data to the Execution Platform's storage system (i.e. HDFS) is the administrator's responsibility.

Batch loading of data into a DOR will be supported by a loader application (see section 6.8), which handles validation, error logging and retrying, and makes use of a HTTP endpoint for ingesting objects into the repository.

6.4 Digital Object Model

Existing repositories already provide their own Digital Object Model for effective storage of digital content and metadata. Such diversity is a hindrance to the SCAPE platform in terms of being able to successfully integrate with every repository. Instead, a common DOM is required.

The SCAPE Digital Object Model is described in detail in [30], essentially however, it is based on a combination of a METS XML container [31] and PREMIS preservation metadata [32]. Each Intellectual Entity is represented by one METS file, and each Representation and File will be described by administrative metadata.

The OAIS model [28] describes, at an abstract level, the requirements that a long-term preservation archival system must fulfil. Within this model is the notion of three Information Packages: Submission Information Package (SIP); Archival Information Package (AIP); and Dissemination Information Package (DIP). Within SCAPE, these packages are METS files adhering to the profile defined in [30], which 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.

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.

6.4.1 Preservation Plans

Preservation plans can be serialised to XML based on the PLATO XML Schema definition [19] - this schema needs updating to reflect updates needed within SCAPE. The plan itself is stored as an AIP in the repository. Executed plans have their provenance information and plan execution details stored in the digital provenance section of the AIP.

6.5 Reference Repositories

Four repositories are targeted as reference implementations for the SCAPE repository:

- Fedora/eSciDoc

- Fedora/DOMS
- Fedora/RODA
- Rosetta

The Fedora-based eSciDoc repository will be used as a reference implementation, with DOMS and RODA implementing the necessary functionality based on this reference implementation.

6.6 Plan Management Logic

6.6.1 Functional Overview

The Digital Object Repository provides the ability to manage and invoke Preservation Plans using its data on the SCAPE Execution Platform. The Plan Management Logic sub-component provides any necessary logic to enable this functionality, including presenting an externally accessible Plan Management API, interacting with the Execution Platform's Job Execution Service API, and accessing and/or updating a Preservation Plan Store.

Some form of user interaction is required to manage and invoke Preservation Plans; this may be implemented by a repository as part of this sub-component, or alternatively an external GUI (such as the SCAPE Plan Management GUI – see section 6.9) may be used.

6.6.2 Technical Overview

The Plan Management Logic sub-component will present an externally accessible Plan Management API as described in section 6.10.3.

6.7 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; what is important, from a SCAPE architecture perspective, is the functionality of the repository and the interfaces to access and reference the digital objects.

A DOR may provide 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 (see section 6.8).

6.7.1 Digital Object Store

This conceptual entity is responsible for storing and making accessible digital object content .

6.7.2 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.

6.7.3 Preservation Plan Store

This conceptual entity is responsible for storing Preservation Plans executable on the SCAPE Execution Platform.

6.8 Loader Application

6.8.1 Functional Overview

The Loader Application is an external (to the DOR) component that provides a means for an administrator to load digital objects into a DOR. It uses the Data Connector API provided by the DOR to enable the loader application to work with any DOR.

6.8.2 Technical Overview

The Loader Application makes use of the Data Connector API HTTP endpoints to ingest data into the repository. Authentication is achieved through HTTP Basic Authentication, with encrypted communication using HTTP over SSL/TLS being highly recommended. Full details about the RESTful API are described in the Connector API specification [4].

A reference implementation of the Loader Application will be developed within SCAPE, resulting in an SDK that can be wrapped by a GUI or accessed through a command line interface. This implementation will address two main use cases: the ingest of managed content (where the SIP includes the metadata and binary object files); and the ingest of referenced content (where the SIP includes only the metadata and has URI references to previously uploaded binary object files).

SIPs are expected to be created prior to uploading in accordance with the SCAPE Digital Object Model. They can be created manually, or by a SCAPE SIP creation tool (still to be developed). The application is designed to support deposit of digital objects regardless of their size, allowing both for a SIP to be POSTed to the repository, or for a reference to its location to be POSTed and for the repository to retrieve it directly. The Loader Application specification is specified [13].

6.9 SCAPE Plan Management GUI

6.9.1 Functional Overview

As a means to view and control execution of preservation plans on the SCAPE Execution platform, some form of user interface is required. This does not have to be a graphical interface, however SCAPE have produced an example GUI [12].

6.9.2 Technical Overview

The SCAPE Plan Management GUI will utilise the Plan Management API provided by the DOR to manage the plans available, and to initiate their execution. By utilising only this API, it will be possible for this GUI to be used by any SCAPE compliant DOR.

6.10 Required Interfaces

Repository systems **must** implement three APIs, the Data Connector API, the Report API and the Plan Management API, to be used within the SCAPE platform. Any repository implementing these APIs should be able to be used in a SCAPE platform.

6.10.1 Data Connector API

The Data Connector API integrates different repositories with the various SCAPE components, allowing these components to access the repository content and preservation plans. It does this by exposing a RESTful interface via HTTP services. Discovery of objects is via an SRU (Search/Retrieve via URL) search endpoint [2]. This Data Connector API has been defined [4].

6.10.2 Report API

The Report API enables communication between a DOR and the Automated Watch component.

The Automated Watch component must monitor repositories, amongst other sources, for information about their contents and the actions that take place on them. In general terms, the Automated Watch component defines Source Adapters to collect information from each source, however as each repository has its own internal information structure and naming schemes, the Automated Watch component would have to create a new Source Adapter for each repository. To prevent this, integration between Automated Watch and repositories is split into two parts: a *Report API* that is implemented by every repository and provides a unified interface enabling Automated Watch to retrieve information about events taking place in the repository; and a *repository Source Adapter*, implemented by the Automated Watch component, that connects to the Report API.

From the repositories point of view, the Report API is sufficient to enable the repository to be used as an Automated Watch input, i.e. the repository does not need to implement the `ISourceAccess` API.

The events exposed and the methods that must be implemented by this Report API are defined in [5] and based upon the OAI-PMH protocol [6].

6.10.3 Plan Management API

The Plan Management API provides HTTP endpoints for retrieval and management of Preservation Plans from the SCAPE digital object repository. Plans are represented using XML and can be searched for, based on their significant properties, using SRU (Search/Retrieve via URL) [2] searching through the relevant endpoint. Queries are represented using Contextual Query Language (CQL) [3].

Endpoints are defined in [1], along with relevant HTTP status codes.

6.11 Roadmap

Of particular importance as a SCAPE output is the DOR reference implementation, which will provide insight and guidance on how to implement the three main APIs required by a DOR, as well as demonstrate the data structure ('content model') support needed by DORs in order to trace provenance information and digital object versions. The reference implementation will be based on eSciDoc, with DOMS and RODA implementing the necessary functionality based on this reference implementation in order to demonstrate the adaptability of Fedora-based repositories and give credence to the reference implementation's approach. The upcoming milestones and deliverables reflect this work.

In addition, a Technology Compatibility Kit (TCK) will also be developed to enable repository developers to test their implementation of the Data Connector API. The TCK will consist of a HTTP Client to test the various Data Connector endpoints, essentially mocking an implementation of a client and testing creation, retrieval and updating of objects within the repository, in accordance with the specification.

An example SCAPE Plan Management GUI is available [12], however this is currently only a front-end GUI with no connection to the DOR. An appropriate user interface (GUI or otherwise) is required to manage Preservation Plan executions via the Plan Management API.

A reference Loader Application will be developed as a means to load digital objects into a DOR. This will make use of the Data Connector API.

Table 5: Upcoming Digital Object Repository Milestones/Deliverables

Milestone/Deliverable	Description	Due
MS42	Loader Application Reference Implementation Deployed on Shared TestBed	M24
MS43	Preservation-Aware Content Models Reference Implementation	M30
MS44	Reference Implementation with Interface to Executable Workflows	M36
D8.1	Recommendations for Preservation-aware Content Models	M36
D8.2	Reference implementation of DOR with interfaces to preservation components, workflows, and execution	M42

7 Execution Platform

7.1 Introduction

The SCAPE Execution Platform provides the necessary infrastructure to execute preservation plans and store appropriate digital objects in a scalable manner so as to aid execution. The goal is to enhance the scalability of storage capacity and computational throughput based on the use of clusters of computational nodes, rather than single machines. These clusters, with appropriate control and workflows, will enable fast and efficient parallel processing of large numbers of digital objects by enabling tools, for example file identification tools, to execute on multiple digital objects at the same time.

An alternative execution platform service is also being utilised within SCAPE and is based on the Microsoft Azure platform [18]. In essence, this provides a similar processing concept to the SCAPE Hadoop based Execution System, whereby multiple computational nodes are utilised to increase computational throughput through parallelisation. Azure provides the ability to reliably (replicated across three computers in the Azure data centre) store data close to these computational nodes, along with the ability to define and manage applications that process this data.

The Azure platform is currently only being used to investigate the requirements for cloud-based architectures to support scalable migration of document formats.

7.2 Functional Overview

To achieve this aim, the SCAPE Execution Platform provides the environment for executing and controlling execution of parallel programs on a large amount of data in a scalable manner. As such, it:

- Executes and monitors the execution of parallel programs across a cluster of computational nodes.
- Is capable of distributing data storage across this cluster of nodes 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 adaptations/compilation).
- Generates parallel programs from simple SCAPE Components that are executable on the platform.
- Stores Parallel Preservation Components (parallel programs) ready for execution
- Enables software agents and human operators to execute Preservation Plans constructed of Parallel Preservation Components.
- Uploads publishable results from workflows and experiments to the SCAPE Data Publication Platform.

7.3 Technical Overview

As can be seen in Figure 2, the Execution Platform consists of four main sub-components: Parallel Preservation Components, the Parallel Execution System, a Taverna Engine and the Job Execution Service.

Parallel Preservation Components are an optimisation of workflows for execution on the Parallel Execution System. They form the building blocks of Preservation Plans. The Parallel Execution System is responsible for execution of Parallel Preservation Components in a parallel manner. Orchestration of the execution of these parallel components may be performed by a Taverna

Workflow executed on a Taverna Engine. The Job Execution Service is responsible for enabling initiation and monitoring of Preservation Plan workflows.

Sections 7.4 to 7.7 describe each of these sub-components in further detail.

7.4 Parallel Preservation Components

7.4.1 Functional Overview

A Parallel Preservation Component (PPC) is a parallel program implementation of a SCAPE Component. It provides a program specifically able to be executed on the SCAPE Execution Platform.

The underlying SCAPE Component is a Taverna Workflow that wraps and controls some tool invocation or preservation action. These components can be built up to define more complex workflows or components (and ultimately a workflow as part of a Preservation Plan). The SCAPE Components (Taverna Workflows) are not optimised for execution on the SCAPE Parallel Execution System however, and as such, need to be generated and stored as appropriate parallel programs.

The Parallel Preservation Components module provides the ability to compile such components to parallel programs (using the “TavernaToHadoop Compiler”), as well as the ability to store these components (in the “PPC Store” – see Figure 2) ready for execution.

7.4.2 Technical Overview

SCAPE Components are XML Taverna workflows that wrap tool invocations or other preservation actions, for example a SCAPE Component workflow may wrap invocation of the Apache Tika™ file identification and parsing tool. These are stored in the SCAPE Component Catalogue for searching and inclusion in larger workflows, however to execute with greater efficiency on the SCAPE Parallel Execution System, these should be compiled to a suitable parallel program, i.e., a Hadoop MapReduce job.

Such parallel programs can be hand-written, however the TavernaToHadoop Compiler provides a means to compile Taverna Workflows to MapReduce jobs. This Java based system uses templates per Taverna activity to convert a workflow into MapReduce Java code. The TavernaToHadoop code is available from GitHub [SW5]. It is in the early stages of development and so is currently only capable of converting simple workflows consisting of single input and output ports surrounding a Beanshell activity.

The platform administrator is responsible for deploying such Parallel Preservation Components to the platform, using an appropriate Hadoop Upload Interface, along with any tools (e.g. Apache Tika™) they depend upon. A simple registry will be maintained to indicate which components are supported by the Execution Platform.

7.5 Parallel Execution System

7.5.1 Functional Overview

The Parallel Execution System sub-component is responsible for providing the infrastructure needed for performing data-intensive computations, and more specifically for the execution of Parallel Preservation Components. It makes use of multiple nodes for storing and processing data in order to increase the computational throughput, whilst maintaining coordination over the tasks to be completed. These connected nodes are known as a cluster.

The output from executing Parallel Preservation Components may be stored on the Parallel Execution System internal storage, transferred back to the Digital Object Repository, or, in the case of publishable results from reproducible experiments, added to the SCAPE Data Publication Platform.

7.5.2 Technical Overview

The SCAPE Parallel Execution System is essentially Apache Hadoop with its associated Apache Hadoop Distributed File System (HDFS), which together provide flexible, scalable and reliable parallel processing and storage. In particular, this combination enables a close proximity between the data and processing nodes, reducing transport overhead and thereby enabling high computational throughput.

7.5.2.1 Apache Hadoop

Apache Hadoop primarily consists of two main sub-projects: MapReduce and the Hadoop Distributed File System (HDFS). MapReduce provides a parallel-processing mechanism that allows Hadoop to process large data sets in a scalable manner. It has components to manage MapReduce jobs, aiming to ensure that 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 commodity hardware. Importantly, it stores data on the same nodes that perform the computation, thereby boosting performance.

7.5.2.2 MapReduce

MapReduce is a framework for parallel processing of large datasets across a large number of computers, or *nodes*. It is divided into two steps: the **Map** step is where the input dataset is divided and shared out amongst worker nodes, where each worker node computes an answer to part of the problem; the **Reduce** step then collects and combines all the partial-answers into one.

Further details about MapReduce can be found in a MapReduce tutorial [11].

7.5.2.3 Hadoop Distributed File System (HDFS)

HDFS is a distributed and scalable file system designed to run on a cluster of machines. A cluster typically comprises of a *Namenode* server, that manages the cluster's file system and access to the files therein, and a number of *Datanodes* (typically one per node), that manage the storage on each node.

Files are split into one or more blocks, where each block is usually a multiple of 64MB, and stored across multiple datanodes. Replication of individual blocks across multiple nodes achieves reliability of the data.

The same nodes are also used for computation in the MapReduce cluster, and because of the close connectivity between these layers, MapReduce jobs can often be scheduled to execute on the same nodes as the actual data, thereby reducing the amount of network data traffic and improving performance.

7.5.2.4 Hadoop Version used within SCAPE

SCAPE, in particular the Central Instances (See Chapter 7.11.1), currently use the patched distribution of Apache Hadoop provided by Cloudera [53]. Specifically, the CDH3 update 2. The Cloudera distribution is used as this is kept up to date with patches solving various bugs and security/performance improvements that are available before a major Apache release. Furthermore, they provide good documentation.

CDH3 update 2 provides:

- Hadoop version 0.20.2
- HBase version 0.90.4
- Zookeeper version 3.3.3
- Hoop

7.5.2.5 Hadoop/Taverna Workflow Integration

Hadoop is designed to operate on large data files rather than many small files, leading to questions over its performance ability when processing large SCAPE datasets. A number of experiments have been performed [20],[21] using Hadoop to try to ascertain the effects of file size versus number of files on Hadoop performance. One study [20] looked at file identification performance of files contained within ARC archive files, comparing the ARC file size versus the number of ARC files, but also looking at the performance impact from executing tools via a Java API or via the command line (through direct tool execution or via a JAR file). The results indicate that: a) increased data file size offers improved processing performance compared with smaller, more numerous files; and b) MapReduce jobs using a tool's Java API provides significantly better performance than invoking a command line tool (either a program or a JAR file). This is likely due to the start-up costs incurred when initiating an external tool, for example, the cost from starting a JVM to execute a JAR. Where possible, tool development should focus on creating Components that utilise tool APIs for execution.

This study is complemented by [21] which investigates the best approach to apply workflows to the Hadoop execution platform. Two possibilities present themselves: i) use Taverna as a scheduler and execute Taverna activities (sub-components of a workflow) compiled as MapReduce applications on the Hadoop cluster; or ii) use Hadoop as a scheduler to run a “driver” workflow, compiled for the platform and comprised of multiple MapReduce programs, on the cluster. There are some advantages and disadvantages to both approaches, such as whether it is necessary to pass HDFS file references between Hadoop and Taverna or the level of integration with Taverna; performance wise however, this investigation suggests using Hadoop as a scheduler and running entire, compiled workflows is significantly faster (despite the need to create an initial sequence file for processing) than using Taverna as the scheduler, and relates back to the fact that Hadoop is designed to work on large input files (e.g. the sequence file) than many small files (see [21] for further details). Under this approach, Taverna Workflows do need to be compiled as MapReduce programs, either through manual creation or through automatic workflow compilation using the TavernaToHadoop compiler. As indicated in Chapter 7.4 however, the TavernaToHadoop compiler is currently in its infancy, and so both scheduling approaches are supported by the Execution Platform.

7.6 Taverna Engine

7.6.1 Functional Overview

The Taverna Engine provides a means to run Taverna workflows which orchestrate MapReduce applications for execution on the SCAPE Parallel Execution System (i.e. Hadoop).

It should be noted that this use is distinct from running a Taverna Engine on each node of a cluster (so as to enable a MapReduce job to invoke a Taverna Workflow); in this latter case, the Taverna Engine should be considered as a SCAPE Component and the Taverna Engine packaged and deployed as in Chapter 5.8.

7.6.2 Technical Overview

The Taverna Engine will run on a local machine or a server close to the Parallel Execution System (the Hadoop cluster). Taverna provides a Command Line Tool for running workflows from the command line [54], or a Web application Archive (WAR) for setting up a Taverna Server to remotely execute workflows [55].

Within a workflow a Hadoop job can be initiated through a Taverna Tool service, which specifies the Hadoop execution command line. A good example is the Hadoop OCR parser workflow created to demonstrate the ability to call Hadoop jobs from Taverna workflows [56]. This example workflow also highlights the use of SequenceFiles in order to help alleviate Hadoop's "Small File Problem" [57].

7.7 Job Execution Service

7.7.1 Functional Overview

Execution of the Preservation Plan workflows (and therefore their associated Parallel Preservation Components) is initiated and managed by clients through the Job Execution Service. Specifically, the Job Execution Service understands SCAPE concepts such as Preservation Components, Data Connector API URLs, and (potentially) the SCAPE Data Publication Platform. As part of its functionality, the Job Execution Service will not try to resolve the data to be operated on however, instead it would merely generate an appropriate input file understandable by a Parallel Preservation Component, based on the input URI provided to it from the DOR (see Chapter 1); the user is responsible for ensuring that the data, i.e. digital objects, were accessible by the Execution Platform.

7.7.2 Technical Overview

The Job Execution Service presents an external RESTful interface, the [Job Execution Service API \(see Chapter 7.9.1\)](#), which is used by a client (i.e. a Digital Object Repository) to initiate and monitor the execution of Preservation Plan workflows.

Internally, the Job Execution Service will need to be aware of which Parallel Preservation Components it has stored and therefore can execute. Depending on the nature of the workflow orchestration (either Hadoop or Taverna scheduling), the Service will either invoke a "driver" workflow program (appropriately compiled for Hadoop) directly on the Parallel Execution System, or it will invoke a Taverna workflow on the Taverna Engine.

In terms of Job Execution Service implementation it is unclear (at present) how much will be provided by Hadoop. The latest release of Hadoop MapReduce (called YARN) provides support for service-based Job submission and Resource Management which could possibly be used; this is currently being investigated for usefulness and, if useful, to what extent it would require extending.

7.8 Microsoft Azure

An alternative platform is also being utilised within SCAPE, and is worth describing here as an example of an alternative execution platform.

Windows Azure is a flexible cloud computing platform (Platform as a Service) that is used to build, host and scale applications across a global network of Microsoft-managed datacentres. It is possible to build applications using any language, tool or framework with features and services being exposed via open REST protocols. Azure provides a robust messaging system that allows for existing IT infrastructures to be integrated with applications running within the Azure environment, enabling the creation of scalable distributed applications and hybrid solutions that run across both cloud and on-premise environments.

Azure allows for applications to be scaled up or down as required, with resource usage management available in real time. Application code can be reliably hosted and scaled out, either vertically or horizontally, within compute roles. Data storage is available via relational SQL databases, NoSQL table stores or unstructured Blob stores, with the option to use Hadoop and business intelligence services to data-mine it. Further details about Azure can be found in [18].

7.8.1 Microsoft Azure within SCAPE

Within SCAPE, Windows Azure is used as an alternative SCAPE platform, primarily for the investigation of requirements necessary for cloud-based architectures to support scalable migration of document formats. SCAPE Preservation Actions are run within Azure Worker Roles with communication via internal endpoints. A Worker Role can be thought of as a process within an OS which is managed by Windows Azure, i.e. updating, spawning, etc. As the name suggests these are typically used for background processing of data. In addition to these Preservation Actions, Word Automation Services, which provide “server-side conversion of documents into formats that are supported by the Microsoft Word client application” [58], are run within Virtual Machine (VM) Roles to provide efficient batch processing of Word format related conversions. VM Roles can be considered more like dedicated instances of an OS that a user needs to manage and maintain themselves.

Further details about the use of Microsoft Azure within SCAPE are still to be defined.

7.9 Required Interfaces

The Execution Platform must implement the Job Execution Service API.

7.9.1 Job Execution Service API

The Job Execution Service API provides a REST interface for executing and monitoring Preservation Plan workflows on the Parallel Execution System. The Digital Object Repository acts as a client to this service, and is responsible for initiating execution of a Preservation Plan against the data that it manages. To help reduce the effects of network latency this data should reside on the Parallel Execution System's Distributed Storage network prior to execution, however it is the user's responsibility for ensuring that this is the case. Such data transfer (from DOR to Distributed Storage) may not be practical or worthwhile however, and so the SCAPE Execution Platform enables data to be accessed directly from a repository.

A Job Execution Service can be used by multiple clients enabling one platform to provide execution services for multiple Digital Object Repositories.

The API is yet to be defined and documented, with Milestone 32 and Deliverable 5.2 providing focus for this work.

7.10 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 virtualisation.

7.10.1 Platform Releases

Platform Concept Release software was released in the summer 2012 consisting of the Central Instance platforms and a MapReduce tool wrapper enabling a user to easily execute command line

applications as MapReduce jobs. The software is currently available from the SCAPE GitHub repository [SW6].

This tool is used with a Hadoop installation (for example, Hadoop can be installed on a PC using a virtual machine such as VirtualBox running Ubuntu) and can execute command line applications such as the Unix File command or FITS file identification as a MapReduce job. The command to be executed is specified in a toolspec file (and passed in as an argument when executing the MapReduce job).

The first platform release is due in M24.

7.11 Platform Instances

There are two types of platform instance currently perceived within SCAPE: Central Instances; and Local Instances.

7.11.1 Central Instances

Central Instances are designed to provide SCAPE participants with pre-configured infrastructure upon which to experiment with platform software, to test and benchmark tools, workflows and Testbed scenarios, as well as to provide a platform for public demonstrations. Two instances are currently available, one from AIT and the other from IMF.

The AIT instance initially comprises a cluster of 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. Further details about how to connect to this cluster are described in [9].

The IMF 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). Details about how to connect and use this cluster are described in [10].

7.11.2 Local Instances

Local Instances are platform instances setup and maintained by an institution primarily to evaluate their own data sets. This typically occurs when an institution has licensing restrictions on the data preventing it from being uploaded to a public repository. 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).

Such instances may or may not be available to other SCAPE members.

7.12 Roadmap

The Execution Platform component forms a large and important section of work. A previous milestone has delivered an initial platform concept release, consisting of the Central Instance platforms and a MapReduce tool wrapper software. A first platform release is due in M24 followed by D4.1 deliverable providing details on the design of the Execution Platform including its main components, layering and interactions.

Table 6: Upcoming Execution Platform Milestones/Deliverables

Milestone/Deliverable	Description	Due
MS27	First Platform Release	M24
D4.1	Architecture Design	M26
D4.2	Final Release	M36

Of concern to the Execution Platform is the means to execute workflows with high performance in order to process the large datasets exposed by SCAPE partners. The approach taken within SCAPE (based on experimental evidence) is to compile workflows to Parallel Preservation Components for execution on the Hadoop based Parallel Execution System, requiring the need for a Taverna-to-Hadoop compiler. An initial version of this is due M20 which will then continue to be developed.

The Parallel Preservation sub-component will need to integrate with the SCAPE Component Catalogue, so it is imperative that an appropriate Component Lookup API is defined in a timely fashion.

Table 7: Upcoming Parallel Preservation Component Milestones/Deliverables

Milestone/Deliverable	Description	Due
MS34	Initial Translator for Taverna Workflows into PPL Algebra	M20
MS35	Executing PPL on Hadoop	M21
MS36	Enhanced compiler and optimiser for Taverna Workflows	M30
MS37	Final evaluation of parallelisation approaches for preservation	M38
D6.1	Report on the Feasibility of Parallelising Preservation Processes	M26
D6.2	Demonstrator and Report on Workflow Compilation and Parallel Execution	M34
D6.3	Optimisation of preservation processes	M38

The Execution Platform component is responsible for providing an interface for initiating execution of Preservation Plans and monitoring their progress. The Job Execution Service API necessary for this is yet to be defined, although a prototype is due in M24.

Table 8: Upcoming Job Execution Service Component Milestones/Deliverables

Milestone/Deliverable	Description	Due
MS32	Job Execution Service Prototype	M24
D5.2	Job Submission and Language Interface	M28

Finally, at the time of writing, details of how Microsoft Azure will integrate within SCAPE are currently unclear.

8 SCAPE Data Publication Platform

8.1 Functional Overview

The SCAPE Data Publication Platform provides a scalable means to publish linked-data results from experiments and workflows whilst recording provenance and versioning information about the results, e.g. who published the results, when were they published, what tools were used. Providing this additional metadata establishes trust in the data, and provides access to historical information enabling decision processes based on this data to be reviewed.

This repository and publishing point for SCAPE experimental results allows them to be historically referenced. As an example, consider a workflow executing the DROID file identification tool over a sample file set. When executed with particular versions of DROID, or with different signature file, DROID may incorrectly identify specific file formats (e.g. Microsoft Word docx); as tool and signature files development iterates, inaccuracies will be corrected (although new ones may be introduced). The file format identification coverage of any specific version of the software, or signature file is therefore hard to ascertain without referencing experimental results.

Another example of experimental results for publication are comparable metrics for digital preservation tools or workflows. These metrics could be performance based, e.g. tool X takes two hours to convert data set Y to PDF, while tool Z took four hours, or quality based, e.g. tool X lost the headers and footers from the document pages, while tool Y retained them. As the SCAPE Preservation Component workpackages and TestBeds continue to develop new tools and workflows, they aim to produce just this type of data

The SCAPE Data Publication Platform aims to store experimental results with additional temporal information making it possible to capture and publish changing tool behaviour in a form where the associated risks can be discovered and reported by the Automated Watch component. Data could either be pushed from the Publication Platform to the Automated Watch Knowledge Base via the Watch Push API, or a Watch Source Adaptor for the experimental data could be developed.

8.1.1 Why Linked Data?

Automating the Watch component as much as possible, in particular the access and retrieval of Source information, would greatly improve scalability (and reliability) of this component. Therefore, in terms of accessing experimental results data, such information ideally needs to be in a self-describing form capable of being consumed by other computing components (i.e. Automated Watch). This open, identifiable data enables the generation of new knowledge through linking multiple datasets and complex reasoning, for example P2's linking of PRONOM and DBpedia enabled answers to questions such as "What tool can open a particular file?".

Linked data could therefore be of major benefit to the Automated Watch component, but there are well-known challenges with using linked data, especially when concerned with digital preservation. In particular, trust and provenance information are hard to come by; data is represented by RDF triples which describe the relationship (predicate) between some subject and an object (value), however there is no notion of who published this information and when. Relatedly, most data in linked datasets represents only the current knowledge - it is hard to get historic data. To help overcome these challenges the Linked Data Simple Storage Specification has been defined [33] and shall be used as a convenient, scalable means to store and publish SCAPE workflow data.

8.1.2 The Data Publication Process

A little needs to be said about the process of publishing SCAPE experimental data. The Data Publication Platform is not intended to store the type of temporary data that is generated while developing and testing a tool or workflow. It's designed to provide a permanent home for significant experimental data that is of value to others in the digital preservation field, e.g. preservation planners, or tool developers. The experimental results published should be the results of reproducible digital preservation experiments performed on open data sets. The first part of the process is the gathering of experimental data, the form of the data is not important as long as it is machine interpretable, i.e. CSV, XML, JSON, etc. are all suitable. It is important that the experiment is performed on a data set that can be openly shared, for example the GovDocs corpus. Experiments performed on private data sets are not reproducible and the results will not be considered for publication.

The data set and results can now be considered for publication. Details of the data set, and where it can be obtained will be published. Specific loaders will have to be developed to convert the data into a form suitable for loading into the LDS³ (see below) store.

8.2 Technical Overview

8.2.1 Linked Data Simple Storage Specification

Building on the P2-Registry [34], the Linked Data Simple Storage Specification (LDS³) [33] provides a system for automating the process of publishing data, whilst helping to maintain trust and versioning information. It does this by extending the triple based RDF model to a quad model, known as a named graph, utilising the fourth dimension to convey facts about the author, publisher, publication time, etc. This is enforced by LDS³, which automatically annotates hosted data with publisher and publication time alleviating the user of this task. Resources (e.g. people, file formats, etc.) cannot be directly created, updated or deleted, and instead have to be described in a published document, i.e. a named graph.

Named graphs are versioned through a combination of GUID and time stamp in the URI scheme used to reference data publications. In this manner, both specific time stamped versions of a publication can be retrieved from storage, as well as the latest version (no time stamp specified).

LDS³ provides a HTTP CRUD (Create, Retrieve, Update, Delete) based interface. Data is HTTP POSTed to the server, returning the location of the created resource. An additional (edit-) IRI is also returned that is used to update or delete the document; this, coupled with the fact that a user can only manipulate data through published documents, means that such amendments are restricted to only that data which a specific user added. All HTTP REST requests must be signed as per the approach employed by Amazon's Simple Storage Service (S3) [35]. This signs only the request portion of a transaction meaning there is no performance degradation as only uni-direction communication is required from client to server.

The full specification is available at [33].

8.2.2 Reference Implementation

A reference implementation of the LDS³ specification has been developed, utilising existing libraries where possible. The OAuth2 module [39] is used for users to register and obtain authentication key-pairs used in authenticating requests. Document annotation is performed by the Graphite library [36]. The quad store, 4store [37], is used to store the quads, enabling their indexing and querying. A patched version of the Puelia-PHP application [25] (which is a PHP implementation of the Linked-

Data API [38]) is used to handle incoming requests in accordance with a dataset configuration file, which details a URI pattern to match and a corresponding SPARQL query to execute; the patch enables retrieval of named graphs from the document URL, a dated URI or an edit-IRI.

8.2.2.1 Linking With Scape

The LDS³ specification [33] imposes some restrictions on clients, such as that they must be able to understand the "Location" HTTP headers, suggesting that clients require some LDS³ specific logic, and therefore a LDS³ client module will be required on the Platform to publish data. Furthermore, whilst the interface for creating and updating publications within the LDS³ server can be done through HTTP requests (that conform to the specification), the Automated Watch component expects Sources to implement a Push or Pull interface, which is likely to differ from the LDS³ REST based API. A simple adapter may be required to interface between the Automated Watch component and the SCAPE Data Publication Platform.

8.3 Roadmap

A reference implementation of the LDS3 specification has been developed. Appropriate connection with the Automated Watch component needs to be considered, potentially requiring a simple adapter to provide the interface.

Table 9: Upcoming SCAPE Data Publication Platform Milestones

Milestone/Deliverable	Description	Due
MS89	Result Evaluation Framework (REF) containing Identification Data	M25

9 Conclusion

The SCAPE project is 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.

To achieve these advances SCAPE are developing a platform tailored towards the automated planning of preservation plans, monitoring of knowledge impacting these plans, and the scalable execution of the Preservation Plan workflows on large content collections. The majority of development work is broadly divided into a number of key sub-components, Automated Watch, Automated Planning, Preservation Components, the Execution Platform and the Digital Object Repository.

This report has provided a detailed overview of the functional components that comprise the architecture of the SCAPE project, in particular describing the interfaces required between each of these functional entities. Specific details about the interface APIs are not discussed, rather the relevant SCAPE API documents are referenced instead (where APIs have been defined). Roadmaps are also provided to give an indication of the upcoming work relevant to each component.

10 References

- [1] "Plan Management API", F. Asseg, M. Hahn, 2012, SCAPE.
- [2] "Search/Retrieve via URL", <http://www.loc.gov/standards/sru/>
- [3] "SRU Contextual Query language", <http://www.loc.gov/standards/sru/specs/cql.html>
- [4] "Connector API", F. Asseg, M. Hahn, 2012, SCAPE.
- [5] "Report API Specification", R. Castro, M. Ferreira, L. Faria, F. Asseg, P. Petrov, 2012, SCAPE.
- [6] <http://www.openarchives.org/OAI/openarchivesprotocol.html>
- [7] <http://www.myexperiment.org>
- [8] "Preservation Components Profile", SCAPE wiki, v. 25
- [9] SCAPE logging into the AIT Cluster.
- [10] "Deployment infrastructure (hosting, storage, security)", S. Barton, R. Schmidt, 2012, SCAPE,
- [11] "MapReduce Tutorial",
http://hadoop.apache.org/common/docs/r0.20.2/mapred_tutorial.html
- [12] "Plan Management Mock-Up", SCAPE wiki, v. 4.
- [13] "SCAPE - Loader Application", Y. Brama, R. Castro, F. Asseg, M. Hahn, 2012, SCAPE.
- [14] "LSDR Executable Workflows for Experimental Execution", D16.1, C. Wilson, P. May, S. Schlarb, B. Jurik
- [15] <http://wiki.myexperiment.org/index.php/Developer:API>
- [16] http://wiki.myexperiment.org/index.php/Developer:WorkflowsResource#Create_workflow
- [17] "Windows Azure", <http://www.windowsazure.com/en-us/develop/overview/>
- [18] <http://www.its.tuwien.ac.at/dp/plato/schemas/plato-3.0.1.xsd>
- [19] ARG.GZ & Tika Hadoop Experiment, M. Raditsch
- [20] "Evaluation of String Operations in Taverna and Hadoop", M. Schenck, 2012, v1.1.
- [21] Requirements documents for provenance component, (SCAPE Checkpoint).
- [22] <http://www.taverna.org.uk/>
- [23] <http://code.google.com/p/puelia-php/>
- [24] "Guidelines for deploying preservation tools and environments", D5.1, R. Schmidt, D. Tarrant, R. Castro, M. Ferreira, H. Silva, 2012,
- [25] "Plato", <http://www.ifs.tuwien.ac.at/dp/plato/intro.html>
- [26] 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)
- [27] 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
- [28] "SCAPE Digital Object Model", M. Hahn, F. Asseg, N. Shirwinter, R. Castro,
- [29] <http://www.loc.gov/standards/mets/>
- [30] <http://www.loc.gov/standards/premis/>
- [31] "LDS³: Linked Data Simple Storage Specification", <http://www.lids3.org/Specification>
- [32] "Preserv2 – File Format Registry", <http://p2-registry.ecs.soton.ac.uk/>
- [33] "Amazon Simple Storage Solution", <http://aws.amazon.com/s3/>
- [34] "Graphite PHP Linked Data Library", <http://graphite.ecs.soton.ac.uk/>
- [35] <http://4store.org/>
- [36] "Linked Data API", <http://code.google.com/p/linked-data-api/>

- [37] [“The oauth 2.0 authorization framework”, D. Recordon, D. Hardt, IETF, 2011.](#)
- [38] [Apache Tika™, http://tika.apache.org/](http://tika.apache.org/)
- [39] [D12.1 “Identification of Triggers and Preservation Watch Component Architecture, Subcomponents and Data Model. K Durutec, L. Faria, P. Petrov, C. Becker 2012 SCAPE](#)
- [40] [“File Information Tool Set \(FITS\)”, http://code.google.com/p/fits/](http://code.google.com/p/fits/)
- [41] [“MongoDB”, http://www.mongodb.org/](http://www.mongodb.org/)
- [42] [“Apache Jena”, http://jena.apache.org/](http://jena.apache.org/)
- [43] [“Jersey”, http://jersey.java.net/](http://jersey.java.net/)
- [44] [“GlassFish”, http://glassfish.java.net/](http://glassfish.java.net/)
- [45] [“Systematic planning for digital preservation: Evaluating potential strategies and building preservation plans”, C. Becker, H. Kulovits, M. Guttenbrunner, S. Strodl, A. Rauber, H. Hofman, http://publik.tuwien.ac.at/files/PubDat_180752.pdf](#)
- [46] [“Plato: A Service Oriented Decision Support System for Preservation Planning”, C. Becker, H. Hofman, http://publik.tuwien.ac.at/files/PubDat_170832.pdf](#)
- [47] [“OWL Working Group”, http://www.w3.org/2007/OWL/wiki/OWL_Working_Group](http://www.w3.org/2007/OWL/wiki/OWL_Working_Group)
- [48] [“JBoss”, http://www.jboss.org/jbossas](http://www.jboss.org/jbossas)
- [49] [“Apache Hadoop”, http://hadoop.apache.org/](http://hadoop.apache.org/)
- [50] [“Cloudera Distribution including Apache Hadoop”, http://www.cloudera.com/hadoop/](http://www.cloudera.com/hadoop/)
- [51] [“Taverna Command Line Tool”, http://www.taverna.org.uk/download/command-line-tool/](http://www.taverna.org.uk/download/command-line-tool/)
- [52] [“Taverna Server”, http://www.taverna.org.uk/download/server/](http://www.taverna.org.uk/download/server/)
- [53] [“Hadoop hOCR parser workflows”, http://www.myexperiment.org/workflows/3069.html](http://www.myexperiment.org/workflows/3069.html)
- [54] [“Hadoop’s Small File Problem”, http://www.cloudera.com/blog/2009/02/the-small-files-problem/](http://www.cloudera.com/blog/2009/02/the-small-files-problem/)
- [55] [“Word Automation Services”, http://msdn.microsoft.com/en-us/library/ee558830.aspx](http://msdn.microsoft.com/en-us/library/ee558830.aspx)
- [56] [“Technical Implementation Guidelines”, D2.1, A. Jackson, 2011](#)
- [57] SCAPE Software References

The following links provide references to software modules and components developed in SCAPE.

- [SW1] <https://github.com/openplanets/scout/tree/integration/adaptors/pronom-adaptor>
- [SW2] <https://github.com/openplanets/scout/tree/integration/adaptors/c3po-adaptor>
- [SW3] <https://github.com/openplanets/scout>
- [SW4] <https://github.com/openplanets/policies>
- [SW5] <https://github.com/schenck/taverna-to-hadoop>
- [SW6] <https://github.com/downloads/openplanets/scape/pt-mapred-demo.tar.gz>
- [SW7] <https://github.com/openplanets/scape-tool-framework>