



Job submission language and interface


Deliverable D5.2 of the SCAPE Project

Authors

Rainer Schmidt, AIT Austrian Institute of Technology

May 2013

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

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



Executive Summary

The Job Submission Service (JSS), provided by the execution environment, plays an important role in the SCAPE platform architecture. The JSS implements a network accessible interface for scheduling and executing workflows (jobs) on the SCAPE platform. In this document, we discuss the major functionality that must be supported by the Job Submission Service in the context of the SCAPE project.

The goal of this document is to identify the components that must be provided on the SCAPE platform and to show how these can be orchestrated by means of the Job Submission Service. This includes for example the identification, configuration, and execution of SCAPE components and their dependencies on the cluster as well as dealing with external data sources and APIs like provided by the SCAPE Digital Object Repository. We therefore explain the primary role of the JSS in the context of SCAPE and describe how it is integrated with the platform architecture. Furthermore, the document discusses how Apache Oozie, a job scheduling service for Apache Hadoop, can be applied as a SCAPE Platform JSS and provides a practical example for scheduling the execution of a preservation tool based on the SCAPE MapReduce tool wrapper.



Table of Contents

1	Introduction	Error! Bookmark not defined.
2	Concepts that must be considered.....	6
2.1	SCAPE Components.....	7
2.2	Platform Application Registry	8
2.3	Data Access, Protocols, and APIs	9
2.4	Specification and Submission of Jobs.....	10
3	Putting Everything Together	11
3.1	Applications available on the Cluster Front-End.....	11
3.2	Items residing on the Distributed File System	11
3.3	Sequential and Parallel Executions	12
4	Example Job using the MapReduce Tool Wrapper	12
4.1	Workflow Specification	13
4.2	The SCAPE Tool Specification Document	15
4.3	The Workflow Configuration Document.....	16
5	Conclusion	16
6	Glossary	18

1 Introduction

The SCAPE Preservation Platform consists of a number of complex system entities that have been built using mature software frameworks like Apache Hadoop, the Taverna Workflow Management Suite, and the Fedora Digital Asset Management System. The SCAPE platform architecture¹ defines SCAPE-specific higher-level services on top of these entities supporting the required concepts of the preservation system and ensuring interoperability among the various components.

The Job Submission Service (JSS), provided by the execution environment, plays an important role in the SCAPE platform architecture, as it implements a network accessible interface for scheduling and executing workflows (jobs) on the SCAPE platform. This document describes the primary role of the JSS in the context of SCAPE and explains how it is integrated with the platform architecture. Furthermore, we exemplify how Apache Oozie², a job scheduling service for Apache Hadoop, can be applied as a SCAPE Platform JSS.

The main duty of the Job Submission Service is providing a remotely accessible API for controlling the execution of SCAPE components (or workflows composed from those) on the platform's parallel data processing environment (called the Execution Platform) based on Hadoop jobs. Under the hood, the JSS functions as a job scheduler, which allocates computing tasks among the available hardware resources available within the execution platform. The concept of job schedulers for (HPC) cluster environments is well known and has a long history, examples for job schedulers are Portable Batch System (PBS)³ or the Maui Scheduler⁴. Apache Hadoop comes with a pluggable scheduling framework allowing a cluster administrator to choose between different implementations depending on the individual needs. Existing Hadoop-based scheduler implementations are the FIFO Scheduler, Fair Scheduler, or the Capacity Scheduler⁵. The latest MapReduce generation (YARN or MapReduce 2.0) has further improved Hadoop's resource management and scheduling strategies. However, the JSS works on top of these mechanisms and only provides a remotely accessible and programmable interface to the local cluster scheduler.

With respect to standardization and related work, the OASIS WS-Resource Framework⁶, a set of standards developed within the Grid computing community, provides mechanisms that can be utilized to remotely manage jobs on computing resources based on SOAP web services. The Globus Toolkit⁷ uses this standard for communicating with different cluster job schedulers within a computational grid infrastructure.

Apache Oozie is a project that provides a REST based job coordinator that is integrated with the Hadoop stack. Oozie meets the SCAPE architecture design very well as it relies on REST-based service interfaces and it provides the required functionality of the JSS, which is mainly focused on supporting

¹ http://www.scape-project.eu/wp-content/uploads/2013/05/SCAPE_D4.1_IM_V1.0.pdf

² <http://oozie.apache.org/>

³ <http://www.mcs.anl.gov/research/projects/openpbs/>

⁴ <http://www.adaptivecomputing.com/products/open-source/maui/>

⁵ http://hadoop.apache.org/docs/r1.1.2/fair_scheduler.html

⁶ https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf

⁷ <http://www.globus.org/toolkit/>



remote job management on the platform. Oozie also defines and implements a language for describing Hadoop jobs as well as means to assemble and schedule more complex (DAG-based) workflows on the cluster.

The availability of a stable job coordinator like Oozie for Hadoop has motivated to evaluate its employment as a SCAPE Platform JSS. It has also become obvious that the Job Submission Service utilized in the context of SCAPE should be based on an established open-source project for reasons of efficiency and reliability, which is difficult to ensure when developing a service from scratch. Consequently, the “job submission language and interface” utilized in SCAPE will be based on already existing service interfaces and job description languages. The major challenge in providing the platform’s job submission service lies therefore in the application of existing description languages to those concepts required in SCAPE. This includes for example the identification, configuration, and execution of SCAPE components (and their dependencies) on the cluster. Another important aspect is dealing with external data sources and APIs like provided by the SCAPE Digital Object Repository.

In this document, we discuss the major functionality that must be supported by the Job Submission Service in the context of SCAPE. The goal is to identify the required components that must be provided on the SCAPE platform and to show how these can be orchestrated by means of the Job Submission Service as a workflow. The document evaluates the employment of Apache Oozie as a JSS for the SCAPE preservation platform and exemplifies how the mechanisms provided by the service can be applied to SCAPE concepts. It is however important to note that the SCAPE platform does not intend to prescribe a particular JSS implementation and that other job coordinators may be used in different deployments. We assume that the job description language and API will highly depend on the utilized job scheduling system and therefore should not be specified but rather be integrated by the SCAPE Platform.

2 Concepts that must be considered

The SCAPE platform architecture defines a number of concepts that are specific to the distributed preservation environment it establishes. Examples are workflows, components, services, and system entities. In order to deploy a specific preservation workflow on the platform, obviously a number of prerequisites must be met. These can include the development, deployment, and registration of a parallel preservation component (in the form of a MapReduce program), assembling a new workflow from existing preservation components, or preparing and loading content and/or metadata on the execution and storage platform.

A major requirement at this point is to identify the concrete mechanisms that must be implemented so that these concepts work together when a particular use-case is performed. In this document, we focus on the interactions and required components to remotely execute different preservation scenarios on the platform. In the following, we list these components and identify their roles and responsibilities. The goal is to identify the functionality that must be provided by the Job Submission Service and to derive requirements on the other involved system components, preservation component/workflows, and the user that wishes to perform a particular use case on the platform.

2.1 SCAPE Components

General Description

What is a SCAPE Component and how is it made available on the platform? SCAPE Components are implemented as annotated Taverna workflows that follow a particular component profile⁸. This is also supported by plugin⁹ for the Taverna workflow-modelling environment that provides graphical support for creating, validating and registering SCAPE preservation components with the SCAPE Component Catalogue. The component catalogue is based on the MyExperiment platform¹⁰ and provides a web-based application for registering, searching, and accessing SCAPE Components.

However, in order to deploy a SCAPE Component on the parallel execution environment it is required to turn the sequential Taverna workflow into a parallel application first. More precisely, the workflow must be executable on the Hadoop platform and all its runtime dependencies must be met. The effort required to “*parallelize*” a SCAPE Component depends on the individual requirements and the required effort to port an application can vary dramatically between different scenarios. For example, an image migration tool might be used as-is to convert a moderate amount of scanned images from one format to another by using a MapReduce tool wrapper. In order to analyse the format or calculate numerical weights for millions of web resources the parallel application must be much more tailored to the execution platform in order to be efficiently executed. As a consequence, it is often difficult to predict the behaviour of the parallel application based on a sequential workflow implementation. SCAPE has been working on various tools and examples that assist a tool/workflow developer in porting a sequential application on the platform. Typical approaches are the employment of MapReduce wrappers¹¹, using Taverna in headless or server mode, the development and employment of input formats that are supported on Hadoop platforms (e.g. for reading web archive files), or the translation of Taverna workflows to MapReduce applications.

Functionality

To summarize, a component on the SCAPE platform must be made available as a Hadoop-compliant application. The functionality of the parallel SCAPE Component may be implemented directly in Java or using a higher level language like Apache Hive¹² or Pig¹³. A SCAPE Component may however also just be implemented through a configuration file that is used by a MapReduce wrapper (based on the SCAPE tool specification language¹⁴). However, a component implementing a more complex workflow (e.g. a quality assured file format migration) might also require the execution of multiple MapReduce programs. Hence, we cannot assume a 1:1 relationship between SCAPE Components and MapReduce applications deployed on the cluster. Consequently, we see a need for employing a workflow language on the execution platform allowing us to represent “*parallelized*” SCAPE components as workflows that can be composed from multiple MapReduce applications. The

⁸ <https://github.com/openplanets/scape-component-profiles/blob/master/profiles/CharacterisationComponentProfile.xml>

⁹ <http://wiki.opf-labs.org/display/SP/Components>

¹⁰ <http://www.myexperiment.org/>

¹¹ <https://github.com/openplanets/scape/tree/master/pt-mapred>

¹² <http://hive.apache.org/>

¹³ <http://pig.apache.org/>

¹⁴ https://github.com/openplanets/scape/tree/master/scape-core/src/main/resources/eu/scape_project/core/model/toolspec

number and nature of the activities in the parallel workflow might however not directly correspond to those used in the sequential workflow implementation.

Requirements

- SCAPE Components must be registered with the SCAPE Component Catalogue and provide – beside other semantic descriptions – a specification on their context dependencies (required operating system, package dependencies)
- Platform (or parallel) components must execute as MapReduce applications on the cluster (which is the responsibility of the component developer)!
- Complex (or composite) workflows (i.e. involving multiple MapReduce applications) must be implemented as parallel workflows. The workflow language must be supported by the Job Submission Services.
- Platform components and their dependencies (like wrapped preservation tools) must be deployed on the execution platform prior to their registration with the platform's Application Registry.
- Deployed platform components must be registered with the platform's Application Registry developed in the context of PT.WP2 Application Provisioning.

2.2 Platform Application Registry

Functionality

The first and foremost requirement on the Platform Application Registry is to provide the mechanisms to register SCAPE Components when being made available on the cluster. This includes the assignment of an identifier (preferably the one used by the SCAPE Component Catalogue) that a client can use to identify a component and schedule it for execution. A first approach can be a simple directory service that organizes the platform components, for example in the form of: `"/components/$component_identifier/$component_impl"`. Components available within this registry can then be identified and (remotely) executed by a user. In a next phase, the component registry might also support additional functionality like performing unit tests, validating that a component's runtime dependencies are met, or retrieving semantic information from the component catalogue.

Requirements

- It must be possible to identify the components available on the platform from a client who wishes to execute them via the Job Submission Service. This functionality should be provided by the Application Registry developed in the context of PT.WP2 Application Provisioning.
- There must be a mechanism to resolve the identifier assigned by Platform Application registry using the corresponding component ID assigned by the SCAPE Component Catalogue (if not the same).
- The Platform Application Registry should take tool dependencies into account, reveal and/or validate them.
- There must be a defined procedure for cluster administrators to register and unregister platform components.
- There must be a procedure for users to browse the platform application registry. Besides the component identifier, the registry must provide sufficient information for a client to configure and execute the component on the platform using the JSS.

- The identification mechanism used by the application registry must also be applicable to composite applications (i.e. workflows).

2.3 Data Access, Protocols, and APIs

The Job Submission Service does not provide any specific means to transfer data between a data source/sink and the execution environment. Hence, it assumes that an application that is scheduled for execution is able to read and write the data it is supposed to process. Apache Hadoop provides a specific Java API for applications that access its distributed file system (HDFS). A number of frameworks like Apache HBase, Pig, and Hive provide data base operations on top of this layer which provide their specific data management interfaces. SCAPE additionally defines the Data Connector API which provides a service to manipulate digital objects maintained by content repositories. From the perspective of the Job Submission Service, two requirements exist: (a) the JSS must be capable of scheduling jobs that utilize different languages, frameworks, and data management APIs; and (b) the job configuration language provided by the JSS must be sufficiently expressive to configure the utilized input/output data locations.

Functionality

From the application perspective, there are hardly any restrictions on the utilized protocols and APIs (e.g. file, hdfs, and http) used to transfer, store, and access data. For reasons of efficiency, it is however advisable to rely on data IO mechanisms that are based on the Hadoop stack for applications that handle significant data loads. In a complex scenario, however, sequential activities or interactions with external services might be required, for example to trigger a data-intensive calculation. Consequently, it is required that the Job Submission Services also supports the execution of sequential components like Java programs or scripts on the cluster. In general, it is assumed that parallel (data-intensive) applications will make use of Hadoop-based mechanisms for handling data IO during the execution. While sequential APIs will be utilized before/after such executions in pre/post-processing steps. An example for a pre-processing activity in SCAPE is a workflow that sequentially retrieves digital objects (in the form of METS files) from the repository, processes the data referenced by these METS records on the HDFS using a MapReduce application, and sequentially updates the repository by depositing a set of newly generated METS records.

Requirements

- Services (like the Data Connector API) must provide a client component (e.g. in the form of an executable Java archive) that can be added to a JSS workflow as a sequential pre/post processing activity. This component should for example support the transfer of a set of SCAPE Digital Objects to a configurable HDFS location. The client component should be configurable based on command-line parameters.
- It is the application developer's responsibility to organize the output data of a calculation (say a file characterization) in a way that it can be loaded into a desired data sink (say a SCAPE digital object repository) using the provided data transfer client libraries (e.g. supporting to ingest of METS records into the repository). This can be achieved, for example, by implementing a MapReduce application that generates METS records (as output) using the SCAPE Digital Object Model (Java API) and the SCAPE METSFileFormat for Hadoop.
- In the context of SCAPE, there is a significant difference between the role of a "component developer" ensuring that a particular tool or piece of functionality (e.g. ImageMagick convert) is available on the parallel execution environment and the role of an

“application/workflow developer” that implements a scenario/use-case on the platform based on the available components. It is the responsibility of the workflow developer to ensure the integration of the various components used in the workflow (like data source, data cleaning, processing, and data sink components).

2.4 Specification and Submission of Jobs

The Job Submission Service provides a network accessible interface for executing applications on the platform’s execution environment. In the first place, this means that the functionality that is provided to a local user must be made available to remote users via the JSS. This is implemented by the JSS through a corresponding job configuration language. The language allows a user to specify the commands that should be remotely executed on the cluster based on a job descriptor document. The descriptor can be submitted via the JSS and its execution can be monitored, as described in the Platform Architecture (SCAPE deliverable D4.1).

Functionality

The functionality available to a local Hadoop user basically involves (1) the execution of file system operation on the HDFS, and (2) scheduling the execution of MapReduce applications on the cluster. Depending on the installed data analytics frameworks, a MapReduce application may be implemented and executed using different environments like Java MapReduce, PHP/Bash scripts (Streaming API), or data base operations (Pig, Hive). Each of these frameworks provides different tools used to configure and issue a particular action. The JSS must therefore provide a job configuration language that is sufficiently expressive for this environment supporting job descriptions for the different tools and frameworks.

Another functionality that is essential to SCAPE scenario developers is the specification of workflows that are assembled from parallel components. These workflows are composed on the level of Hadoop jobs (as opposed to workflows that are implemented by parallel components). The JSS supports the specification of these workflows and enacts them directly on the cluster (as compared to an approach where the workflow is orchestrated from a client computer). In SCAPE, complex scenarios involving multiple processing steps and/or data sources/sinks can be implemented using the workflows support provided by the JSS. Using Apache Oozie, these workflows can be directly stored on the HDFS file system enabling their registration with the SCAPE Platform Application Registry. The JSS workflow support can for example be used to compose, store, and execute SCAPE preservation plans on the SCAPE Platform.

Requirements

- The JSS must provide a job specification language that is capable to enact applications developed for the different Hadoop data analytic frameworks/languages utilized in SCAPE.
- The JSS must also be able to enact sequential programs like those required to export data sets from a digital object repository.
- The JSS must be integrated with the Platform Application Registry allowing a user to select an application based on its SCAPE Component identifier.
- The JSS must provide means to specify workflows that are composed from multiple SCAPE Components and other (sequential) applications running on the cluster.
- The JSS must be implemented as a REST-based service that provides the functionality described in the Platform Architecture (SCAPE deliverable D4.1).

3 Putting Everything Together

This document discusses the relevant concepts for executing SCAPE Components on the SCAPE Platform, which is based on the Apache Hadoop software stack. Figure 1 provides a high-level view of the integrated system with respect to the discussed concepts in particular.

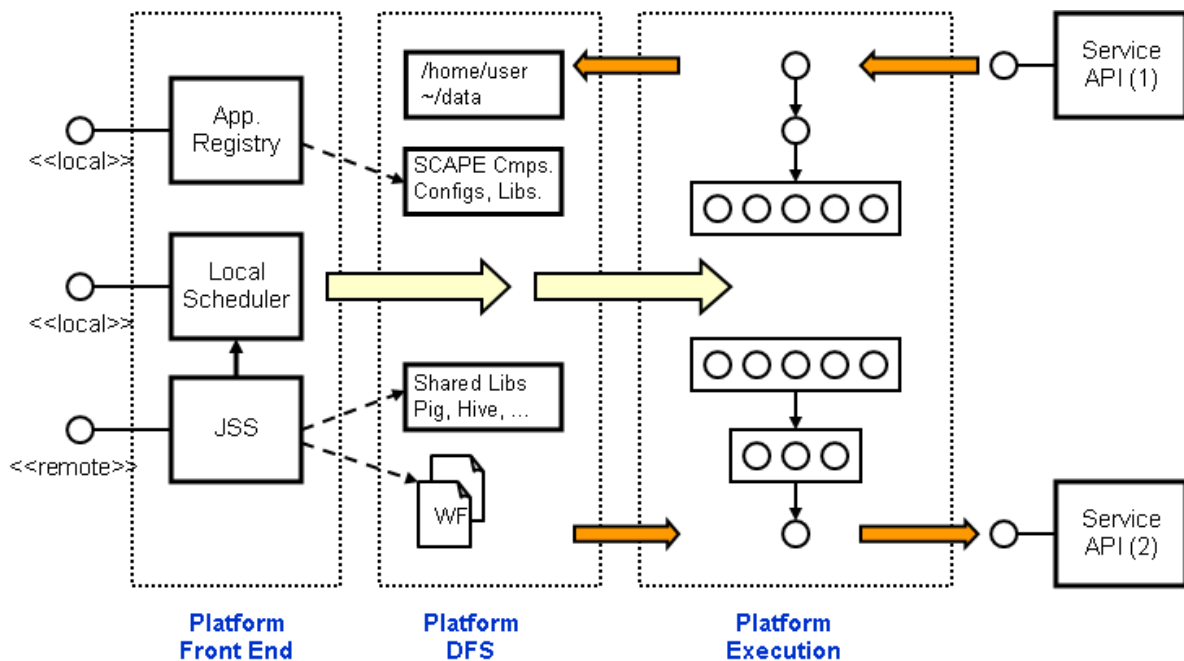


Figure 1: Executing SCAPE Components on the Platform

3.1 Applications available on the Cluster Front-End

The Application Registry, the Job Submission Services, and the Local Scheduler are applications that are available to a user from the cluster front-end node. The local scheduling system is typically provided through the execution environment (Hadoop) and available to a user through a command-line interface. The Job Submission Service is a server-sided workflow scheduler which can be provided through Apache Oozie. Oozie is available to local and remote users (through a client program) using its Web Service API. The Application Registry is a SCAPE specific command-line program that helps administrators (and users) to manage SCAPE Components that are deployed on the cluster.

3.2 Items residing on the Distributed File System

The Job Submission Service is required to cope with the various Hadoop-related libraries needed to successfully launch a job through the execution environment’s scheduling system. A major concern is



for example to build up the correct Java Classpath required by an application and/or the underlying framework it uses. Apache Oozie also organizes the workflow documents as well as the referenced MapReduce applications through HDFS. The SCAPE Application Registry will naturally take a similar approach¹⁵ to organize SCAPE Components (most importantly workflows documents, Java libraries, and configuration files) on the cluster. The goal is to enable a workflow designer to implement a SCAPE scenario by assembling a workflow from parallel SCAPE Component (using a defined formalism).

Apache Oozie supports different authentication methods allowing the JSS to execute jobs under user permissions. This allows a user to organize the input data for a particular job under the user's HDFS home directory. Data will typically reside on HDFS prior to the execution but can also be loaded onto the HDFS as part of a workflow. Jobs that are scheduled for execution on the MapReduce cluster will also access the software libraries they depend on mostly using the distributed file system (HDFS). Exceptions include applications depending on native tools that must be made available on every cluster node prior to the execution.

3.3 Sequential and Parallel Executions

MapReduce applications (e.g. developed in Java or a Hadoop-compliant scripting languages) are shipped to the cluster nodes via HDFS and instantiated as parallel programs on the cluster through the MapReduce scheduling system (illustrated by the yellow arrow in Figure 1). A particular workflow may however comprise sequential and as well as parallel activities (illustrated in Figure 1 - Platform Execution). Oozie supports the specification of sequential libraries or scripts within a workflow and executes them in the form of a single Map task on a cluster node. Parallel tasks are performed by the execution environment based on concurrently scheduled Map and Reduce tasks.

The MapReduce execution environment is directly embedded into the distributed file system allowing the cluster nodes to access files on HDFS very efficiently. It is however possible to implement data transfers from an external service to the distributed file system as part of a workflow, if desired. This approach can be beneficial in cases where only metadata needs to be transferred as well as for automatically scheduled reoccurring jobs. The workflow examples in Figure 1 include sequential activities that transfer input data from an external service into HDFS and vice versa (illustrated by orange arrows in Figure 1). In SCAPE, workflows that must handle the import/export of digital (metadata) objects (maintained within a content repository) might be implemented analogously using the SCAPE Data Connector API.

4 Example Job using the MapReduce Tool Wrapper

A common requirement in SCAPE is the execution of command-line applications over large volumes of data. This includes for example file format migrations, format identification, and the characterization and quality assurance of digital items (described in more detail on the corresponding project Wiki pages¹⁶).

¹⁵ <http://blog.cloudera.com/blog/2012/12/how-to-use-the-sharelib-in-apache-oozie/>

¹⁶ <http://wiki.opf-labs.org/display/SP/Scenarios>



The SCAPE MapReduce Tool Wrapper provides a useful utility that allows a user to concurrently execute pre-installed command-line applications using Hadoop. The tool wrapper supports data residing on HDFS as well as file systems mounted over the network. A tool execution is configured based on the SCAPE Tool Specification Language (or toolspec for short). The toolspec specifies required information like package-name, supported actions, and corresponding command-line patterns for a particular tool. The toolspec used in combination with the MapReduce tool wrapper provides a handy way to execute actions that rely on command-line tools on the platform's execution environment.

4.1 Workflow Specification

In the following, we provide a simple workflow example that shows how the MapReduce tool wrapper can be executed using Apache Oozie. It is not intended to provide an overview of Oozie and its workflow specification language in the context of this document. For more detailed information on Apache Oozie the reader is referred to the Oozie project home page¹⁷. Please also note, that the example provided here, assumes that the tool wrapper is executed as a MapReduce application (in form of a Java Archive) residing in a user's home directory. The detailed means by which an application is turned into a SCAPE Component and registered with the platform using the SCAPE Application Registry is not subject to this example.

The workflow specification provided in Table 1 implements a migration scenario using the PostScript interpreter *Ghostsript* as a tool and *ps2pdf* as the action to convert a set of ps-files residing in an HDFS input directory into pdf-Files. The workflow consists of two actions: the first action is invoking a shell script that generates an input file for the tool wrapper, the second actions starts the tool wrapper implemented as a MapReduce Java program. There are a few things to note:

- Variables in the workflow declared in the form $\$(variable_name)$ will be assigned a value at runtime by the user as explained later in this example.
- The shell script *generate-input-file.sh* is dynamically shipped to the node where it will be executed using the `<file> tag`¹⁸.
- The tool wrapper presently only implements a mapper function (eu.scape_project.pt.mapred.CLIWrapper.CLIMapper).
- The tool and action that will be executed by the tool wrapper are specified as Hadoop JobConf properties using the `<configuration>` element.

```
##workflow.xml##
workflow-app name='ps2pdf-wf' xmlns="uri:oozie:workflow:0.2">
  <start to='generate-input-file'/>
  <action name="generate-input-file">
    <shell xmlns="uri:oozie:shell-action:0.2">
      <job-tracker>${jobTracker}</job-tracker>
      <name-node>${nameNode}</name-node>
      <exec>generate-input-file.sh</exec>
      <argument>${inputDir}</argument>
```

¹⁷ <http://oozie.apache.org/>

¹⁸ <http://blog.cloudera.com/blog/2013/03/how-to-use-oozie-shell-and-java-actions/>

```

    <argument>${outputDir}</argument>
    <argument>${generatedInputFile}</argument>
    <capture-output/>
    <file>generate-input-file.sh</file>
  </shell>
  <ok to="ps2pdf"/>
  <error to="fail"/>
</action>
<action name='ps2pdf'>
  <map-reduce>
    <job-tracker>${jobTracker}</job-tracker>
    <name-node>${nameNode}</name-node>
    <prepare>
  </prepare>
    <configuration>
      <property>
        <name>mapred.mapper.class</name>
        <value>eu.scape_project.pt.mapred.CLIMapper</value>
      </property>
      <property>
        <name>scape.tool.toolspec</name>
        <value>ghostscript.xml</value>
      </property>
      <property>
        <name>scape.tool.action</name>
        <value>ps2pdf</value>
      </property>
      <property>
        <name>mapred.input.file</name>
        <value>${generatedInputFile}</value>
      </property>
    </configuration>
  </map-reduce>
  <ok to='end'/>
  <error to='end'/>
</action>
<kill name='kill'>
  <message>${wf:errorCode("ps2pdf")}</message>
</kill>
  <end name='end'/>
</workflow-app>

```

Table 1: workflow.xml

4.2 The SCAPE Tool Specification Document

Table 2 shows an excerpt of the SCAPE Toolspec for Ghostscript providing two actions for ps2pdf migrations. Other actions like for example ps2pdfa would be typically also included into this document. The toolspec is provided as a file residing on HDFS and utilized by the MapReduce tool wrapper to execute a desired command-line pattern against the files being processed. For a detailed definition of the SCAPE Tool Specification Language the reader is referred to the toolspec code project¹⁹.

```
##ghostscript.xml##
<?xml version="1.0" encoding="utf-8" ?>
<tool xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://scape-project.eu/tool tool-1.0_draft.xsd" xmlns="http://scape-project.eu/tool"
xmlns:xlink="http://www.w3.org/1999/xlink" schemaVersion="1.0" name="ghostscript"
version="1.0.1" >
  <installation>
    <dependency operatingSystemName="Debian">ps2pdf</dependency>
    <license type="Apache Licence 2.0">Apache License, Version 2.0</license>
  </installation>
  <operations>
    <operation name="ps2pdf">
      <description>Converts postscript to pdf</description>
      <command>ps2pdf ${input} ${output}</command>
      <inputs>
        <input name="input" required="true">
          <description>Reference to input file</description>
        </input>
      </inputs>
      <outputs>
        <output name="output" required="true">
          <description>Reference to output file</description>
        </output>
      </outputs>
    </operation>
    <operation name="convert-streamed">
      <description>Converts postscript to pdf reading from stdin outputting to stdout</description>
      <command>ps2pdf - -</command>
      <inputs>
        <stdin required="true">
          <description>Input stream</description>
        </stdin>
      </inputs>
      <outputs>
        <stdout>

```

¹⁹ https://github.com/openplanets/scape/tree/master/scape-core/src/main/resources/eu/scape_project/core/model/toolspec

```

    <description>Output stream</description>
  </stdout>
</outputs>
</operation>
</operations>
</tool>

```

Table 2: Tool Specification for Ghostscript ps2pdf actions

4.3 The Workflow Configuration Document

While the workflow specification and the libraries, scripts, and property files it relies on are stored on the HDFS, the workflow is instantiated and configured by a user dynamically. In order to schedule a workflow for execution, the user submits a job configuration (here the job.properties file) to the Oozie REST interface. This can be done for example using a provided command-line interface. The file shown in Table 3 provides a configuration for the example described above, simply specifying the utilized Hadoop instance, the path to the workflow, and the utilized HDFS input and output directories.

```

##job.properties#
nameNode=hdfs://172.20.30.40:8020
jobTracker=172.20.30.40:8021
oozie.wf.application.path=${nameNode}/user/${user.name}/ps2pdf-wf
inputDir=${nameNode}/user/${user.name}/ps2pdf-in
outputDir=${nameNode}/user/${user.name}/ps2pdf-out
generatedInputFile=${nameNode}/user/${user.name}/ps2pdf-in/input-file.txt

```

Table 3: Job configuration file

5 Conclusion

The document discusses the role of the SCAPE Job Submission Service (JSS), a network accessible interface for executing jobs on the SCAPE platform. A major concern was the identification of the relevant SCAPE concepts and the development of an integrated approach that can be deployed on top of Apache Hadoop. In this context, the document evaluates the application of Apache Oozie as a Job Submission Service for the SCAPE platform.

Based on this evaluation, it has clearly turned out that the SCAPE Job Submission Services should be based on an established open-source project for reasons of efficiency and reliability. The major challenge for the SCAPE Platform lies in the integration of the existing software and description logic to the concepts implemented in SCAPE. It has turned out that the mechanisms provided by Oozie correspond well to what is required on the SCAPE platform with respect to job specification and execution.



Another important goal of this document was also to discuss the requirements and means by which SCAPE components can be deployed and orchestrated on the Hadoop platform. We conclude that (a) SCAPE Components will be implemented using a Hadoop-compliant workflow language on the platform; (b) activities used in these workflows will be (implemented and) executed as MapReduce applications on the platform; (c) it is not required that the number and nature of the activities in the parallel workflow directly correspond to those used in a sequential version of the workflow implementation.

We see the primary role of the SCAPE Job Submission Service to define the job and workflow description language, to handle the scheduling and enactment of activities implemented using different Hadoop-based analytic frameworks and languages, and to provide a network accessible interface to the client environment.

6 Glossary

SCAPE Platform	The SCAPE platform is an integrated system that includes various components like the execution environment, the digital object repository, and the component catalogue. The Platform Execution Environment provides a scalable backend for digital preservation systems allowing them to carry out the processing of data-intensive workloads. The platform reference implementation has been built on top of existing frameworks like Apache Hadoop, the Taverna Workbench, and the Fedora Commons repository. The platform architecture specifies a set of preservation specific higher-level services that are built on top of these components.
SCAPE Component Catalogue	The SCAPE Component Catalogue provides a service to register and look-up semantically enriched workflows that comply with SCAPE (workflow) component profiles and the SCAPE ontology. Workflows registered and stored using the SCAPE Component Catalogue are implemented and annotated using the Taverna Workbench. These workflows are however not required to run (efficiently) within the Platform Execution Environment (i.e. Apache Hadoop).
SCAPE Components	SCAPE components are entries in the SCAPE Component Catalogue. Components provide semantic and technical metadata describing for example the functionality, ports, and dependencies of a component. A component description is however independent from a concrete deployment of the component.
Platform Application Registry	The Platform Application Registry maintains parallel versions of SCAPE components that are deployed on the platform. The registry takes only the deployment of applications (i.e. executable components) on a specific execution platform instance into account. Each entry in the application registry is required to provide a pointer to the corresponding general component description provided by the SCAPE Component Catalogue.
Platform Job Submission Service	The Platform Job Submission Service provides a network accessible REST interface for scheduling jobs on the Platform Execution Environment. The job submission service interprets a job or workflow description language and handles the scheduling and enactment of the described activities on behalf of a client.