
# Technical Implementation Guidelines

Authors
Andrew Jackson (British Library)

July 2011

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


# Executive Summary

This report gives an outline of the technical architecture of the SCAPE project, and defines project-wide technologies, coding best practices, and recommendations for development environments and tools. It is primarily aimed at those developing tools under the Preservation Components work package, but also touches on broader development issues.


# Table of Contents

# 1 Technologies

The SCAPE architecture attempts to make a clean separation between the preservation component (tool) layer, the web layer, the platform layer and the application layer. The technologies and requirements are different at each level, and an initial roadmap is in place to ensure that these components can be fully integrated as the project proceeds.

## 1.1 The Application Layer

The primary user applications in the SCAPE architecture are Taverna (for the Testbeds) and PLATO (for Preservation Watch). Taverna will be used to develop the Testbed scenarios, has a mature integration framework for WSDL/SOAP services, and has more recent support for RESTful services and for integrating command-line tools (the latter as of version 2.3). PLATO will be used for preservation planning, and is needs to invoke preservation tools during the planning process. To so this, it simply requires a web service (WSDL or REST) with a stable API.

Please note that tool development should be driven by the needs of the Testbeds Scenarios, and so it must be possible to invoke tools from Taverna.

It is envisaged that simple command-line or GUI clients to local tools or remote services may be required later, in order to assist broader integration of the SCAPE outputs. Repository integration may need remote services or local tools (as per the ePrints PLATO integration). For general systems and web integration, a RESTful style would be preferred.

## 1.2 The Platform Layer

The Platform layer exposes the tools and Taverna workflows as remote services that run on a cluster of machines. This should allow the tools developed during this project to be tested and deployed at scale, processing large collections with relative ease. The Platform layer will be able to invoke Preservation Components either as local Java or command line tools. Initially, the Platform will be Apache Hadoop, with data stored in HDFS and/or HBase. Later on, Taverna workflows will be run directly on the cluster, and tools that have particular platform requirements may be invoked as 'remote' services from inside locally-deployed virtual machines constructed precisely for that purpose.

## 1.3 The Web Layer

In order to make the tools available during the development of Taverna workflows and during the PLATO planning process (or indeed during broader institutional or repository integration), it is necessary to have the SCAPE Preservation Components available over the web, either as WSDL or RESTful endpoints.

## 1.4 The Tool Layer

Preservation components should, in general, be developed as simple command-line tools or as Java classes implementing a standardised interface. Any command-line tools we write or adopt must compatible with the SCAPE Platform. The process of turning a tool into a remote service will be standardised by the Platform and TCC work packages, and tool developers should not need to write wrapper code or other web-layer components. Only limited standardisation of the tool inputs and outputs is absolutely required, but further standardisation will be encouraged in order to allow the results from different tools to be combined together more reliably.


## 2 Architectural Roadmap

The four layers describe above will work together best if we can find a way of ensuring that the preservation tools can be invoked in a reproducible manner, whether the call is being executed locally or remotely. We aim to do this by writing declarative specifications that describe the tools and indicate how they should be invoked from the command-line or as Java classes, and then standardising the way these specifications can be invoked as WSDL/SOAP or as RESTful services. To improve the integration further, we will propose interoperable data formats and consistent semantics across contexts where required. As the different layers start with different capabilities and will mature over time, we have a staged roadmap, as follows.

### 2.1 Year One

In order to meet its deadlines, the Testbed sub-project must be able to invoke tools from Taverna straight away. The fastest route to making this happen has been to start with the Axis2-based WSDL/SOAP wrapper code that was developed for the IMPACT project, and extend it to meet SCAPE's needs. This lead to the WSDL Tool Wrapper codebase that you can find here:

https://github.com/openplanets/scape/tree/master/xa-toolwrapper

This framework allows ad-hoc WSDL/SOAP web services to be deployed, based on simple configuration files that specify how to invoke a given command-line tool. Using this information, it builds a deployment package that exposes the tool as a web service, with the command parameters exposed as ports so that Taverna can show them. This permits loose integration via WSDL/SOAP to be set up fairly easily, with results returned as a pointer to a new file hosted at a URL local to the service, while service and execution metadata is returned in a standardised form.

Currently, the ONB is hosting a number of services for the Testbed sub-project, thus ensuring that the current SCAPE workflows (http://www.myexperiment.org/groups/490.html) can be executed locally without installing anything other than Taverna itself. At the current time, all tools should be deployed as services in this way. Note that the process of deploying a tool does not require any software development, just the creation of a new set of configuration files in the xa-toolwrapper project. See the tool wrapper instructions for more details.

### 2.2 Year Two

Although Taverna allows ad-hoc service integration, there have been ongoing efforts to standardise different types of services as hot-swappable components, making it easy to shift between different services that perform the same operation. This aims very close to the original Planets approach, where the strongly-typed web services described the standardised forms for preservation actions like Migrate, Identify and Characterise. However, the Planets interfaces suffered by being too complex, inflexible, and clumsy, especially in the way the digital objects were transferred (as a SOAP attachment, by default). By shifting to typed, extensible tool specifications (backed by Java interfaces per type), we can combine the best of all of these approaches, leading to a lightweight deployment procedure that efficiently exploits local data while allowing remote integration to be supported as required.

During year one, for adoption in year two, we will work with the Taverna External Tools & Components efforts to build these standardised interfaces. This will be done by defining a shared schema for tool specifications. These simple XML definitions describe individual tools and how to invoke them to perform different types of actions. The specifications are based on those used by the

Taverna External Tools plugin (http://www.mygrid.org.uk/dev/wiki/display/developer/Calling+external+commands+from+Taverna), which look like this:

```
<program name="bourne_shell_script" description="execute shell script"
      command="/bin/sh input">
    <input name="shell_script">
        <file path="input" />
    </input>
</program>
```

These tool specifications will be extended to meet the SCAPE project's needs, and will allow us to use standardised software to invoke these tools locally, or to expose these tools as web services. In the latter case, we will target RESTful deployment, as this is the easiest to integrate across a wide range of contexts. In both cases, we can use the standard invoker to automatically collect performance metrics during execution, re-using and extending the methods used by the MiniMe registry from the Planets version of PLATO.

Furthermore, by adopting these re-usable tool specifications, we make it easy to share information on tool invocation patterns reliably, e.g. via Email or for sharing via GitHub. It also provides a route by which tool parameters may be standardised across different tools that perform the same operation. Note also that, for format conversion, this approach is similar to (and broadly compatible with) that of the NCSA Polyglot system (http://isda.ncsa.uiuc.edu/NARA/conversion.html) and its associated Conversion Software Registry (http://isda.ncsa.uiuc.edu/NARA/csrAbout.html).

# 3 Development Best Practices

During code development, Java is generally the preferred platform language, as this makes sharing code across contexts easier. This does not, however, mean that the tools themselves must be developed in Java, and indeed the tool specification approach is designed to avoid making this necessary. In particular, it is preferable to extend an existing project in the language that the developers are currently using, rather than attempting to port to Java. Nevertheless, Java should be considered as the default language for new developments.

## 3.1 Java Development Best Practices

We will initially target Java 6, and Java 7 will be evaluated as soon as possible after its release.

Any Java code should follow Oracle's *Code Conventions for the Java Programming Language* (http://www.oracle.com/technetwork/java/codeconv-138413.html). Conformance to those conventions will be evaluated using the Checkstyle tool (http://checkstyle.sourceforge.net/).

All code should be accompanied by unit tests. The Clover code coverage tool (http://www.atlassian.com/software/clover/) will be used to assess how well the supplied tests cover the codebase.

The status of the codebase will be monitored via an integrated build server. This automatically builds the code after each commit to the code repository, and ensures that any changes that break the build are brought to the attention of the author of the code as soon as possible. See the section on Build Management below.

## 3.2 Development Environment & Tools

The supported build tool is Maven 2. By handling the build and the dependencies in this way, we can support a wide range of development environments without managing IDE-specific configuration files. SCAPE Developers IDEs include Eclipse, NetBeans, IntelliJ and good old CLI. However, Eclipse should be considered the default IDE, as this is the development platform that the SCAPE partners have the most experience with.

If you are using Eclipse, here is a list of useful plugins that will help you develop SCAPE code.

- http://www.eclipse.org/m2e/ for Maven support.
- http://www.eclipse.org/egit/ for Git integration.
- http://www.atlassian.com/software/ideconnector/ for integration with Atlassian tools, e.g. JIRA, Bamboo, etc.
- http://wiki.apache.org/hadoop/EclipsePlugIn for integration with Hadoop.
- http://eclipse-cs.sourceforge.net/ for using the Checkstyle tool.
- http://www.javaforge.com/project/HGE MercurialEclipse for working with Mercurial repositories.
- http://subclipse.tigris.org/ for working with Subversion repositories.
- http://pydev.org/ for Python or Jython development.
- http://www.aptana.com/products/radrails for working with Javascript, Ruby, HTML and mobile platforms.

### 3.2.1 Development Resources

Along with the basic build tools, the SCAPE project is using the following resources to help manage the code, hosted by the Open Planets Foundation.

- http://wiki.opf-labs.org/display/SP The public SCAPE wiki, powered by Confluence.
- http://jira.opf-labs.org/browse/SP The SCAPE issue tracker, powered by JIRA.

## 3.3 Code Repositories & Conventions

All code will default to open source and be publicly available by default. If particular pieces of code are found to have dependencies or legal restrictions inconsistent with this approach, they will be moved elsewhere.

### 3.3.1 Repositories

The initial plan is to use a central repository where we all work together, hosted by GitHub at https://github.com/openplanets/scape. All developers will have write access. Code will be arranged by work package (see below). Any pull requests will be fielded by the Technical Coordinator.

By default, any new code should be developed in that repository. Of course in some cases, e.g. if the work involves forking an existing project, this may not be appropriate. As the functional requirements become clearer, the code may be re-factored along functional lines rather than by work package. As the code stabilises and matures, we may switch to a more tightly managed system where the core repo has limited commit access and larger modifications are managed via pull requests. Similarly, if a single shared repository is not working well, we can split the code into sub-repositories as required.

Furthermore, separate repositories can be set up for specific purposes, and of course in some cases it will be necessary to use external repositories of various kinds (e.g. when extending existing tools).

Please let the Technical Coordinator know about any external repositories, so that the code can be tracked and integrated into the build as necessary.

### 3.3.2 Naming Conventions

Java code developed under the SCAPE project should belong to the package

```
eu.scape_project
```

and it is recommended that, within that package, the sub-package names reflect the project structure. i.e.

```
eu.scape_project.(sub-project).(work-package)
```

e.g.

```
eu.scape_project.pc.cc
eu.scape_project.pc.pa
eu.scape_project.pc.qa
eu.scape_project.xa
```

Similarly, it is recommended the code held in the main Git repository is held as distinct maven projects per task, but where the name of the top-level folder reflect the sub-project and work package as well as the task:

```
/pc-cc-opjpg-xmldump-patch
/pc-qa-bitwiser
/xa-toolwrapper
```

### 3.3.3 Licensing

Any new developments should, in general, be released under the Apache 2.0 licence. For example, all files should have a header similar to this example Java comment (taking care to modify the year as appropriate):

```
/*
 * Copyright 2011 The SCAPE Project Consortium
 *
 *   Licensed under the Apache License, Version 2.0 (the "License");
 *   you may not use this file except in compliance with the License.
 *   You may obtain a copy of the License at
 *
 *       http://www.apache.org/licenses/LICENSE-2.0
 *
 *   Unless required by applicable law or agreed to in writing, software
 *   distributed under the License is distributed on an "AS IS" BASIS,
 *   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either expressed or implied.
 *   See the License for the specific language governing permissions and
 *   limitations under the License.
 */
```

Of course, when extending existing code, you should NOT remove the existing header and replace it with this unless you are the copyright holder. If you cannot use the Apache 2.0 licence (e.g. when extending an existing project that is under the GPL), or have any other questions about licensing, you should get in touch with the Technical Coordinator

### 3.3.4 Using the Approved Useful Software Registry

The Approved Useful Software Register is a list of the SCAPE project's dependencies on third-party software tools and their associated licenses.

https://portal.ait.ac.at/sites/Scape/Management/Lists/IPR%20Registry/Allitemsg.aspx

Any software your code depends on should appear on that list. The purpose is so that all parties can have an opportunity to object to any items whose licensing conditions would prohibit further legitimate use of the software as per the consortium agreement.

## 3.4 Build Management

Currently, the Technical Coordinator is the build manager, and the Open Planets Foundation is providing the automated testing and continuous integration server:

http://bamboo.opf-labs.org/browse/SP

The build manager role will shortly be taken over by the Internet Memory Foundation, and the build server is expected to migrate to IMF systems.

## 3.5 Large-scale Testing Infrastructure

As per the description of work, the SCAPE project will set up a central instance of the SCAPE platform, hosted by the Internet Memory Foundation. The system manager for the central platform will make it available to SCAPE partners so that they can test their tools and software at scale.

## 3.6 Publishing our outputs via Maven

Maven artefacts will be pushed to the central Maven repo via Sonatype, if appropriate. See this ticket for details.

As well as publishing our own artefacts so that they can be re-used, we will also pursue the possibility of getting other related projects to publish their artefacts through Sonatype, and indeed help them do so. This will encourage code reuse and help ensure that the network of software dependencies we need will be preserved over time. If we cannot publish JARs directly, we can also use Sonatype to upload third-party JARs.

## 3.7 Working with Third-Parties

When extending or improving existing tools, the Technical Coordinator should be made aware of the tool being adopted and of the identities of the people involved (on both sides). In particular, the Technical Coordinator will wish to know about the licensing terms and the location of the source code repository. We will also offer out build integration and testing services to the third-party projects, as appropriate.

In general, it is recommended that SCAPE project partners work closely with the third-party developers to agree and document the distinct features that they wish to add to the third-party project (e.g. as issues in the SCAPE JIRA tracker). Specific branches should be constructed for each feature, and a road-map should be agreed specifying the terms and time-scales under which these feature branches should be folded into the core project.

If this is not possible, the SCAPE project partners may wish to consider forking the original project and created a SCAPE variant of the original work. However, this is not encourage in general, as we


are aiming to improve the quality of existing tools rather than invest heavily in branches that may not be supported once the project draws to a close. We are not just preserving digital objects. We are also preserving code, and the representation information enshrined by the code is extremely valuable and its sustainability should be a primary goal.

# 4 Communications & Feedback

In general, if you don't know who else to get in touch with, get in touch with the Technical Coordinator.

Communication will be via github.com and the OPF Labs resources. See Getting started for details.

## 4.1 Developer Workshops

The first developer workshop has been announced here, with details here. Please keep an eye on the SCAPE and/or OPF web sites for future announcements. More information will be added under the Events page.

## 4.2 Twitter

People in SCAPE on twitter include:

- @SCAPEproject and see also #SCAPEProject.
- @anjacks0n

## 4.3 Mailing Lists

The Technical Coordinator also runs the Technical Coordination Committee, and two mailing lists for technical discussion. For TCC discussions:

tcc@list.scape-project.eu
http://list.scape-project.eu/cgi-bin/mailman/listinfo/tcc

and for general SCAPE technical discussions

techie@list.scape-project.eu
http://list.scape-project.eu/cgi-bin/mailman/listinfo/techie

Both lists are fully public, and open to non-project members.