



## **SEVENTH FRAMEWORK PROGRAMME**

### **Networked Media**

*Specific Targeted Research Project*

# **SMART**

(FP7-287583)

## **Search engine for Multimed*i*A environment generated conten*T***

### **D7.4 Open Source Software Portal**

Due date of deliverable: 31-07-2012

Actual submission date: 12-09-2012

Start date of project: 01-11-2011

Duration: 36 months

## Summary of the document

<b>Code:</b>	<b>D7.4 Open Source Software Portal</b>
<b>Last modification:</b>	7/09/2012
<b>State:</b>	Final
<b>Participant Partner(s):</b>	ATOS, GLA, AIT
<b>Author(s):</b>	Iadh Ounis (GLA), Craig Macdonald (GLA), Jose Miguel Garrido (ATOS), Dyaa Albakour (GLA), John Soldatos (AIT) Zvi Kons (IBM), Massimiliano Tarquini (S3LOG)
<b>Fragment:</b>	No
<b>Audience:</b>	<input checked="" type="checkbox"/> public <input type="checkbox"/> restricted <input type="checkbox"/> internal
<b>Abstract:</b>	This deliverable accompanies the establishment of the open source portal of the project at: <a href="http://opensoftware.smartfp7.eu/">http://opensoftware.smartfp7.eu/</a> . It illustrates the selected open source license, along with the governance scheme to be applied in the project. Furthermore, it reports on the rationale behind the selection of both the license and the governance scheme.
<b>Keywords:</b>	Open Source, Git, Mozilla Public License (MPL), Master-Governed Planning
<b>References:</b>	N/A

## Table of Contents

1	Executive Summary .....	5
1.1	Scope .....	5
1.2	Audience .....	5
1.3	Summary.....	5
1.4	Structure.....	6
2	Overview of the SMART Open Source Project .....	7
2.1	Rationale for releasing SMART as Open Source .....	7
2.2	Importance and Scope of the SMART Open Source Development Infrastructure .....	7
2.3	Importance and Scope of the SMART Open Source Governance Model .....	7
2.4	Importance and Scope of the SMART Open Source Licensing .....	8
3	SMART Open Source Development Infrastructure .....	9
3.1	Review of Main Open Source Tools .....	9
3.1.1	Website .....	9
3.1.2	Wiki .....	9
3.1.3	Version control and source code repository .....	9
3.1.4	Issue tracking.....	10
3.1.5	Mail lists and Forums.....	10
3.1.6	Public project hosting ("Forges").....	10
3.1.7	Self-hosted forges.....	12
3.2	Public websites and Self-hosted tools .....	12
3.3	Public and semi-public strategies .....	12
3.3.1	Public development.....	12
3.3.2	Semi-public Development.....	13
3.3.3	Strategies and tools .....	13
3.4	Recommendations and Selection .....	14
3.4.1	Git repository.....	15
3.4.2	Trac.....	16
3.4.3	Technical implementation .....	17
4	SMART Open Source Governance.....	18
4.1	Open Source Governance Methodologies and Collaboration Schemes .....	18
4.1.1	Collaborative planning .....	18
4.1.2	Master-governed planning .....	18
4.1.3	Use Cases .....	19



4.2	SMART Project Constraints, Recommendations and Selection .....	20
4.2.1	Constraints Associated with the Governance of the SMART Open Source.....	20
4.2.2	Recommendation and Selection.....	20
5	SMART Open Source Licensing .....	22
5.1	Open Source Philosophies .....	22
5.2	Main Open Source Licensing Requirements of the SMART Consortium .....	23
5.3	Open Source Licenses Considered by the SMART Consortium .....	24
5.3.1	GPL.....	25
5.3.2	LGPL.....	25
5.3.3	MPL.....	25
5.3.4	BSD & Apache .....	26
5.4	Recommendations and Selection .....	26
6	Conclusions .....	27

## 1 Executive Summary

### 1.1 Scope

SMART is developing an innovative multimedia search engine (along with an associated search framework) for environment generated content. Among the main objectives of the SMART project is to ensure the openness of the search framework, which will guarantee its extensibility in terms of (physical and virtual) sensors, sensor processing algorithms and presentation elements. As part of this objective, SMART will be releasing the implementation of its search framework as open source software (OSS). At the same time, SMART will endeavor to create and build up an open source community around the main SMART results. The development of this community hinges on the specification of a set of processes and tools that will support the building up, the collaboration and the governance of the community of the developers and the users of the SMART open source software. The purpose of the present deliverable is to present processes and tools associated with the setup, management and governance of the SMART open source community. In particular, the deliverable covers the following topics:

- The ICT tools that will be used in order to host and manage the SMART open source software project, including an open source software portal, a software versioning system, as well as tools for software documentation and contributors' collaboration.
- The processes that will be used for governing the SMART open source community. These include the main roles and responsibilities associated with the evolution of the SMART software and the regulation of the relevant contributions.
- The licensing scheme selected for the SMART open source project, which plays an important role for the future exploitation and sustainability of the project.

For all the above topics the deliverable illustrates the various options that were considered, along with the criteria that drove the final selection.

### 1.2 Audience

The target audience for this deliverable is manifold and includes:

- **The members of the consortium, notably the consortium members involved in the setting up of the SMART open source project:** The decisions presented in this deliverable will be taken into account by the SMART consortium members towards setting up the SMART open source project. Note that as part of this deliverable the consortium has also setup its open source portal, in line with the decisions presented in this document, which is the report that accompanies the open source portal implementation.
- **The open source community:** SMART intends to build a community of open source developers and users, which will (respectively) engage in the development/enhancement and use of the SMART open source project. The topics of the present document are therefore of interest to SMART users and/or developers, who would like to acquaint themselves with the SMART open source processes and tools. Note however that the information contained in this document will be also available on-line in the scope of the open source portal of the project.

### 1.3 Summary

This report is part of the deliverable D7.4, which deals with the setup of the SMART repository of open source code, and of the associated tools and techniques for open source development. The reports deals with three important aspects/choices of the SMART open source software, namely: (a) the selection and set-up of the appropriate environment and tools, (b) the selection of the open source licenses of the project and (c) the selection of a proper governance scheme for the SMART open source project and relevant community. For all three selections/choices, the deliverable illustrates possible options and justifies the final selection

of the SMART consortium. In terms of the open source development environment, SMART has setup a hosted web portal (available at: <http://opensoftware.smartfp7.eu/>). The portal comes with a range of popular tool facilitating development. In terms of licensing scheme, the consortium has opted for the Mozilla Public License (MPL), with a view to ensuring compatibility with background libraries used, while boosting possibilities of commercial exploitation of the SMART open source platform. Finally, in terms of the open source community governance, the SMART partners have specified a master-governed planning scheme, which emphasizes the leading role of a master (in the SMART case corresponding to a certain partner) in the evolution of SMART components or subprojects. The SMART consortium has appointed «master» partners for the two main subjects comprising the SMART system, namely the SMART edge node and the SMART search engine (based on the Terrier.org) search engine. The selected governance approach aims at facilitating integration and robustness at the early stages of the project. The consortium does not rule out changes or revisions to this governance scheme. The details of the selected licensing and governance schemes are described in later paragraphs of this document.

## 1.4 Structure

The deliverable is structured as follows:

- Section 2 illustrates the importance of the main processes that comprise the setup of the SMART open source project.
- Section 3 illustrates the ICT tools that comprise the SMART open source infrastructure, including the SMART portal, software version system and collaborative tools.
- Section 4 describes the processes that will ensure the proper governance of the open source community of the project.
- Section 5 reports on the process that led to selection of the SMART licensing scheme (based on the MPL license). The process includes the review/survey of popular licensing schemes for open source projects and their subsequent comparative analysis against the requirements of the SMART project.

## **2 Overview of the SMART Open Source Project**

### **2.1 Rationale for releasing SMART as Open Source**

SMART is developing a search engine for multimedia environment generated content i.e. content derived from multiple sensors deployed at the physical environment, along with content stemming from social networks feeds processing. Among the objectives of the project is to create the search engine on the basis of an open and extensible framework that provides flexibility in terms of the physical and social sensors data streams to be integrated and searched. The rationale behind the openness of the framework is to allow third-parties (beyond the developers of the SMART consortium) to enhance the search engine on the basis of novel sensor processing components, reasoning algorithms and search applications. In order to pursue and accomplish this objective, SMART will release a major part of its software as open source. The open source nature of the SMART project is expected to drive a number of other benefits as well, in particular:

- It will allow for community development of the SMART software, which could lead to improved software quality, while at the same time boosting the sustainability and technological longevity of the project.
- It will provide a software blueprint for building non-trivial applications that leverage environment generated content. This blueprint could be exploited by enterprises in order to build added-value search applications in areas such as news, media, security and surveillance.

The decision to implement SMART as open source should be accompanied with key decisions associated with the tools and processes that will support the open source project. These tools and processes concern three main areas in particular:

- The selection of the ICT tools comprising the SMART open source development infrastructure.
- The governance scheme and processes of the SMART open source project.
- The license of the open source project.

The importance of these areas is outlined in following paragraphs. Note that several SMART partners have experience in setting up and contributing to open source projects, which has greatly facilitated the process of taking decisions associated with tooling, governance schemes and open source licenses. In this respect the contribution of partner GLA (who is the founder and main contributor to the open source Terrier.org project) has been substantial.

### **2.2 Importance and Scope of the SMART Open Source Development Infrastructure**

Open source projects involve smaller or larger communities of developers that engage in the design, production and documentation of software libraries comprising the open source project. These communities operate on the basis of a range of ICT tools, which provide documentation, collaboration, software versioning capabilities and more. The selection of these tools can therefore be crucial for the productivity of the open source community. Nowadays, there are several popular tool-suites, which can be used stand-alone or on-line. SMART has reviewed several tools and accordingly selected the ones that suited its needs. More details are provided in Section 3 later in this document.

### **2.3 Importance and Scope of the SMART Open Source Governance Model**

The governance model is another important element of an open source project. It specifies roles, responsibilities and processes regulating the activities of the open source community. Governance is particularly crucial when it comes to specifying the process of ensuring the quality of software contributions, as well as when it comes to integrating those contributions within the open source code base. A main challenge associated with open source governance is the proper balancing between collaboration and quality of integration. This is because collaborative models boost collaboration and democratic decisions, while at the same time the existence of integration masters can better ensure the quality of the integration. Section 4 of the document il-

illustrated the selected SMART governance model.

## **2.4 Importance and Scope of the SMART Open Source Licensing**

The selection of the licensing scheme of the SMART open source project has also been an important decision for the project, given that it has significant impact on the project's exploitation and sustainability strategy. Hence, the selection of the SMART open source licensing has considered a variety of factors including the partners' exploitation intention, but also compatibility and reuse of other open source libraries (such as the Terrier.org search engine which is a crucial element of the SMART open source project).



### 3 SMART Open Source Development Infrastructure

In this section we present the SMART Open Source Development Infrastructure. However, we also outline some considerations the main options considered prior to make the ultimate selection.

#### 3.1 Review of Main Open Source Tools

##### 3.1.1 Website

Any open source software project requires a web presence that can be accessed globally and should at least describe the project, identify its purposes and maintain the latest releases of the project (zip, .tar.gz etc).

It should also provide the necessary links for the documentation and any other tools used for the development of the project to help anyone learn more about it and enable those interested to become part of the project community and contribute to its development.

In the case the SMART project, we have the home page of the project in [www.smartfp7.eu](http://www.smartfp7.eu). The web page for developers will be a subsection of the home page.

##### 3.1.2 Wiki

A recent trend is using a wiki for the instructions and help about the application, working as the main part of the web page. There are several websites that offer free wiki service.

- **Wikia** (<http://www.wikia.com>) uses MediaWiki software.
- **Wikispaces** (<http://www.wikispaces.com/>) for educational purposes.
- **Wikidot** (<http://www.wikidot.com>) section for groups and collaboration

Apart from these specialized “wiki farms”, the “forges” usually feature their own wiki functionality (see section 3.1.6)

On the other hand, there are various wiki software packages that can be installed in any server by the developers (self-hosting):

- **Mediawiki**<sup>1</sup> – software used by Wikipedia, written in PHP.
- **MoinMoin**<sup>2</sup> – python wiki platform, used by Apache Consortium and Terrier. Can support private wiki pages.
- **Confluence**<sup>3</sup> – closed source wiki platform from Atlassian.

In addition to the help displayed on the wiki, it is necessary to display the automatically generated JavaDoc for the API of the library. It is only static html information, so the project home page could be used.

##### 3.1.3 Version control and source code repository

There are two main alternatives for version control are centralized and distributed systems. The two commonly used tools for those two options are Subversion and Git respectively:

**Subversion (SVN):**<sup>4</sup> The traditional solution, used in thousands of projects and supported by all devel-

---

<sup>1</sup> <http://www.mediawiki.org/wiki/MediaWiki>

<sup>2</sup> <http://moinmo.in/>

<sup>3</sup> <http://www.atlassian.com/software/confluence/overview>

opment tools. The main problem is that its way of working is obsolete, especially for big, collaborative projects. SVN is only an evolution of CVS, and CVS was an evolution of the first system, RCS.

**Git**<sup>5</sup>: is a relatively new system designed by Linus Torvalds. After testing existing open tools, none was good enough for developing the Linux kernel, and the commercial tool BitKeeper changed its terms of use. Git features a distributed model; more advanced than the client-server approach of SVN and CVS.

Other common free version control systems are Mercurial, Bazaar, Monotone or Arch. They use the same distributed philosophy as Git. Therefore the decision of choosing a version control system is usually making a choice between the classic SVN/CVS systems and the new Git philosophy.

### 3.1.4 Issue tracking

The issue (or bug) tracking systems maintain a list of bugs, new features or pending issues in general. Popular systems include:

- **Bugzilla**: (<http://www.bugzilla.org/>): it is both a portal and an installable application.
- **Jira** (<http://www.atlassian.com/software/jira/overview>): a commercial product, but free for open projects. The main advantages are of Jira are:
  - Customizable workflows.
  - Code integration, including with GitHub
  - Fancy reports and statistics.
- **Trac** (<http://trac.edgewall.org/>): an enhanced issue tracker that nowadays offers integration with SVN or Git, and with a wiki. The main advantage is that it is possible to add cross references between issues, code and wiki pages. It is open source, so it is possible to use it self-hosted, the same as Bugzilla.

### 3.1.5 Mail lists and Forums

The Mailing list helps the community to use email as a communication medium for supporting the community and end users (e.g. ask for help or documentation) or to collaborate on the development of the project (e.g. discuss issues/bugs that need to be resolved). Mailing lists can be used in combination with the aforementioned tools such as the wiki, the issue tracker or the forum. The problem with mailing lists as support tool is that the number of messages can grow quickly as the number of users increases, so usually another solution is needed for a mature community.

Forums are good tools for supporting external users. The forum is really important from the last stages of the development, when the project is used by people not in the project.

Some of the functions of the forum are better covered by the issue tracking system, but usually we can't expect that all the users are able to create an issue report. Some open source projects receive a lot of useful bug reports using the issue tracker directly, but the forum is a more "human-friendly" way of providing bug reports and support to users.

In the Terrier project, the forum has been more successful than the mailing list – it has the advantages of not spamming users' inboxes, which is important for a support role. Users mostly submit forum posts and when they are identified as bugs, the moderators of the forum ask them to post an issue to the bug tracker, or they post the issue themselves.

The main disadvantage of forums is that they can be a bit chaotic and usually the forum needs one or more moderators.

### 3.1.6 Public project hosting ("Forges")

One popular option is to use "forges" which are repositories for open source hosting. Usually they offer code

---

<sup>4</sup> <http://subversion.apache.org/>

<sup>5</sup> <http://git-scm.com/>

hosting and many other tools previously discussed open source software development, trying to offer an integrated environment for the developer.

**SourceForge** (<https://sourceforge.net/>) was the first and even today it is the biggest of the forges. Recently, they have launched a “new” SourceForge 2.0 with new characteristics, as a Wiki or Git.

- **Code Hosting**
- **Issue tracking** (now called tickets)
- **Wiki**
- **Forum**
- **Blog**
- **Mailing lists**
- **Subversion and Git**

SourceForge is a web-based source code repository, which centralizes the development and management of open source software. A common criticism is the unintuitive navigation, it easy to get lost in the pages.

**GitHub:** A commercial forge, but free for open source projects.

- **Code Hosting**
- **Issue tracking** (Jira can use GitHub)
- **Wiki** (using their own system)
- **Teams**
- **Git** (also SVN)

The main advantage of GitHub is that they have very good review characteristics for the code. In the other hand, it is not possible to create private zone, because that is the feature that GitHub sells to clients (<http://marmoush.com/2011/10/04/sourceforge-vs-github-2011/>). GitHub is surpassing the alternatives in popularity (<http://www.readwriteweb.com/hack/2011/06/github-has-passed-sourceforge.php>). Several developers recommend GitHub, but usually is a matter of “taste” or “feeling” more than a rational thing.

**Google Code:** (<http://code.google.com>) has all the good characteristics (Wiki, issue tracking, Git and SVN). It hostess many Google or Android related projects, but it is open to all open projects. A key advantage is the simplicity.

It is evolving to be part of the general infrastructure for developers (<https://developers.google.com/>)

Apart from these forges, there are several others. Usually the features are a bit more limited and the mission is more specialized, but probably any of these owns merits enough to be a good alternative if needed.

- **Alioth** (<http://alioth.debian.org/>) is based over FusionForge. It is intended mainly for projects related to Debian.
- **GNU Savannah** (<http://savannah.gnu.org/>) is quite similar, intended for projects of the FSF.
- **BerliOS** (<http://www.berlios.de/>) is infrastructure depending from Fraunhofer that offers several services for open source developers, including a forge.
- **CodePlex** (<http://www.codeplex.com/>) is the forge for open source developers using Microsoft products.
- **Gitorious** (<http://gitorious.org/>) is another good forge based around Git.
- **JavaForge** (<http://www.javaforge.com/project/11>) is a full featured forge for projects not only using

Java.

- **JoinUp** (old OSOR.eu) (<https://joinup.ec.europa.eu/>) is an EC portal for open source solutions for public administrations, and includes a forge.
- **LaunchPad** (<https://launchpad.net/>) is sponsored by Canonical Inc and it is quite related to Ubuntu. They use Bazaar instead of SVN or Git

### 3.1.7 Self-hosted forges

With self-hosted forges, the initial originators of the project need to set up a server and install the forge for open source development. They also need to maintain the forge afterwards.

**FusionForge** is a self-hosted forge based on the public SourceForge. Fusion forge integrates several other utilities to offer the basic services that the “classic” Source Forge offers: code hosting (both SVN and Git), issue tracking, forum, user management and even a wiki. The forge is available to deploy privately via a package for Ubuntu, but the installation typically requires various customizations.

Other possible option is to choose some tools and integrate it, creating a “forge” with the needed tools.

## 3.2 Public websites and Self-hosted tools

As we see in the last section, for each tool, usually we find the possibility of using a public website, like Github, or self-hosting the tools.

In this last case, a website is created for the project and each individual tool such as the code repository or the wiki is deployed on-demand as a standalone part of a greater website. The advantages and disadvantages of such a process are as follows:

### Advantages

- The project can maintain all tools within a single website umbrella, thereby ensuring uniform branding.
- The project has flexibility in choosing the suitable tool for the project needs.

### Disadvantages

- Increased cost of installing/maintaining separate tools.
- Hosting costs. A very popular project can be downloaded by many people, and the network traffic and hosting infrastructure cost can be very high.
- Possible lack of integration. For instance, multiple logins for different tools can confuse users.

## 3.3 Public and semi-public strategies

We identify two main streams of development strategies both rely on the same principia but differ in objectives and management, namely *public development* and *semi-public development*. These development strategies mainly differ in the degree of openness to the community.

### 3.3.1 Public development

In this strategy, the source code (especially the source code repository) is publicly available at all times throughout the life-cycle of the project. The development process is entirely transparent to the active contributors as well as to the wider community, i.e. any interested individual or organization would have access to the source code and the development process and they may also become part of the process and contribute to the project and the source code.

For example, the popular Apache<sup>6</sup> software foundation has built a community that developed a number of

---

<sup>6</sup> <http://www.apache.org/>

open source software products using this strategy.

Advantages:

- The entire development process is open to the community and they become stakeholders of the project.

Disadvantages:

- Initial originators of the project (in our case, the consortium) have less control on the visibility of the planning and development process, as it is open to the entire community and anyone can become part of the planning and management process.
- Any code mistakenly committed by a developer has defacto been released, as it will always be available in the source code repository.

### 3.3.2 Semi-public Development

With this strategy, the project is managed by a number of individuals or organisations that may hide parts of the development process or source code from the wider community of contributors and/or users. For example, incremental releases are made available to public but private releases that contain un-disclosed code may be kept private to the managers. This allows decisions on what components need to be kept private to the initial originators.

The Terrier platform is developed with this strategy where public releases are always available and contributions from the entire community are welcome but the source code repository is kept private, and releases are planned and prepared in private.

The OpinionFinder project<sup>7</sup> is an extreme example of the strategy where the planning, management and the contribution to the source code are only done by the originators of the project and the releases are available to the community. Another example is MySQL.

Advantages:

- The initial originators, in our case the consortium, retain full control on the management and planning process and its visibility to the community.

Disadvantages:

- The community is less involved in the planning process.
- No committers outside of the consortium.

### 3.3.3 Strategies and tools

The strategy is important because it limits the possible elections. The tool or set of tool chosen must be able to implement the strategy. Most part of public tools, like Source Forge or GitHub, are clearly oriented for a public development strategy, and they don't offer private repositories or they ask a fee for it.

But a public website can be used in a semi-public strategy, providing that the source code repository is self-hosted. A common tendency is a semi-public strategy, combining a self-hosted development with the use of Google Code for distributing the public releases.

In the case of SMART, the development team is distributed, so the source code repository must be available using Internet for the developers. As the repository must be available, it is possible to use the same server computer for hosting other tools.

---

<sup>7</sup> <http://code.google.com/p/opinionfinder/>

### 3.4 Recommendations and Selection

According to the semi-public strategy, some parts of the portal must be only accessible by the developer team. As we stated in section 3.3.3, this make a self-hosted portal a sensible alternative, so the implemented solution is a self-hosted web portal.

- We can offer all the assets related to the project under a unified brand, our smartfp7.eu domain.
- Allow most liberty of election for the tools
- More flexible, we can create as many repositories/sub-projects as needed, and we can define fine-grained access policies access according to different purposes.
- It is possible to use the same repository for the non-open source parts of SMART.

The portal is installed in the server [opensource.smartfp7.eu](http://opensource.smartfp7.eu) . The solution is implemented mainly using two open source tools:

- Trac
- Git

(See sections 3.4.1 and 3.4.2 for more details about Git and Trac)

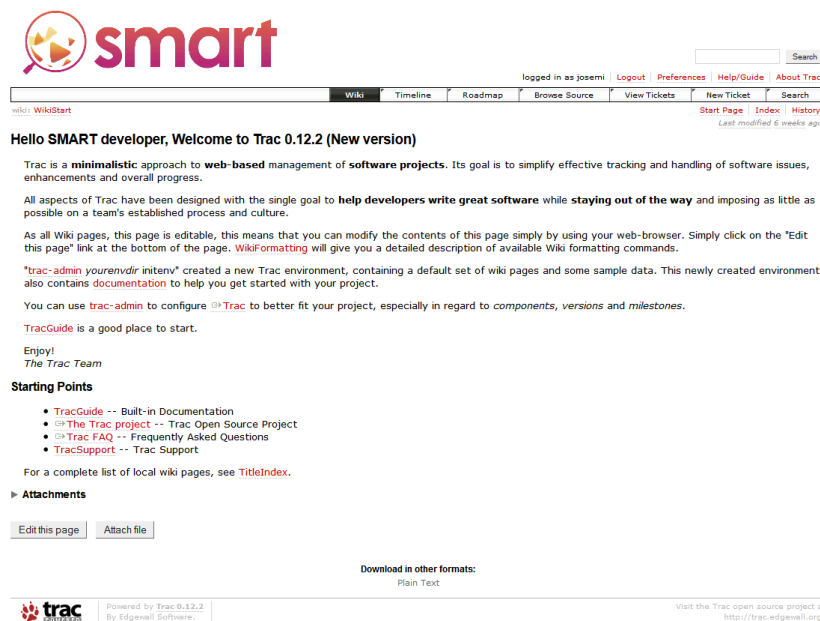


Figure 1: Main page of an example portal implemented using Trac

Some other services are available in the portal:

- A general purpose **web server**: It resolves the problem of publishing documentation automatically generated from the source code. The results from tools as JavaDoc are directly publishable as web pages.
- Some **mail list** for SMART developers: we have currently using some mail lists in the SMART project, mainly for consortium members. As we state previously, as the number of developers grows, the mail list become less useful, so we can implement a forum in the future if needed.
- A **WebDAV** repository: Some files and contents are not suitable for the code repository, which as its own name implies, it is optimized for source code. But at the same time, it is important to share it with other developers because a complete system needs these files. Videos, sound contents large binary files in general are the best examples. Usually a FTP server was em-

ployed, but a WebDAV repository is a better solution, and it is more suitable for integration in the web portal.

Particular effort was put about the integration of users, one of the traditional problems of the use of several tools at the same time. In this case, the tool election and the configuration it is tailored in a way that it is possible to use the same user/password for accessing Trac, Git or WebDAV.

### 3.4.1 Git repository

As we stated in section, Git is more advanced than SVN, Git uses a new approach that overcome some of the old limitations of the traditional SVN/CVS philosophy.

#### Advantages

- Git is distributed; each developer has his own local repository and each repository can commit changes in the code with each other, not only with only a central repository.
- Git has a better a mechanism for creating and merging branches. Git allows a non-linear development workflow. In the case of SMART, the developers can use a private repository for every-day work and upload complete releases to a public repository
- As Git is distributed, the failure of a central repository is not a problem.
- A Git repository can communicate using several pre-existent protocols, like http or ssh.
- One of the main requirements of Git was the speed. Git is able to work with really complex projects, like the source code of the Linux kernel. Git is clearly quicker than SVN, especially for big repositories.

#### Disadvantages

- A problem with Git is that the developer needs to change their way of thinking<sup>8</sup>, by having to use a local repository, after many years of taking SVN/CVS as reference.
- Git was invented for the use in a Linux environment and it is implemented at low level using Unix system calls. Therefore, porting Git to BSD and Mac OS X was straightforward, but the implementation of Git for Windows was not direct. In any case, some solutions appeared; so it is not especially difficult to use Git in windows nowadays.

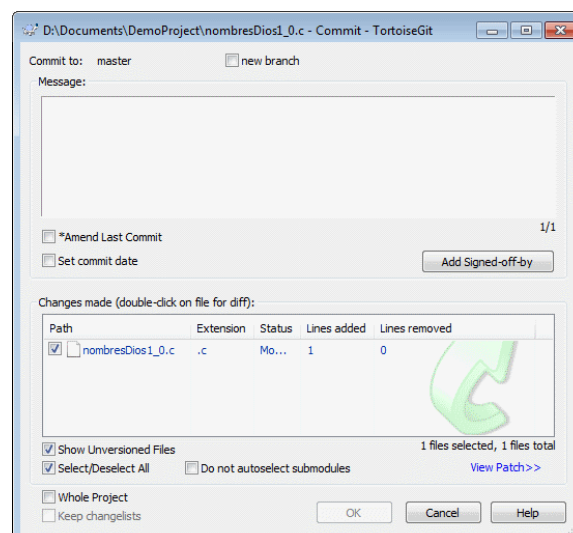


Figure 2: Example of tool: TortoiseGIT for Windows

<sup>8</sup> [https://git.wiki.kernel.org/articles/s/v/n/SvnMigration\\_f3dd.html](https://git.wiki.kernel.org/articles/s/v/n/SvnMigration_f3dd.html)

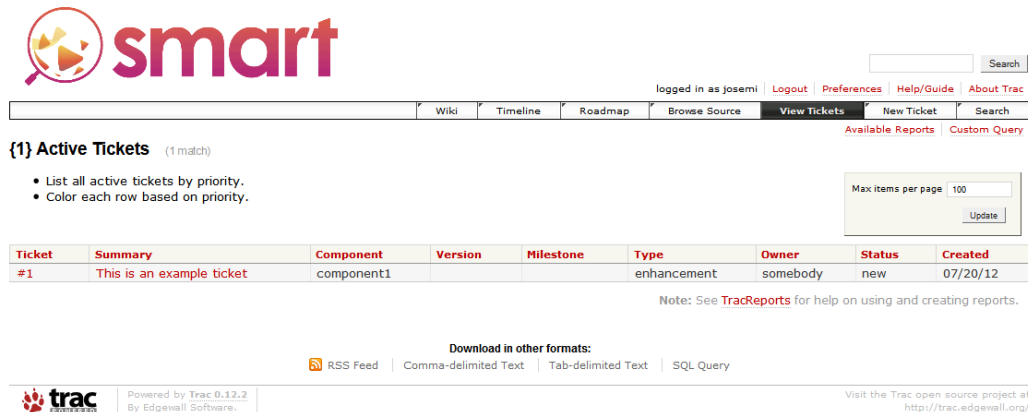


As we stated before, given the distributed character of Git, each SMART developer will need to install a local Git repository in his/her computer. The Git repository hosted by the project is only a point of encounter for the communication between different developers; it is not really a central repository in the SVN sense.

### 3.4.2 Trac

We call Trac an issue tracker in the section 3.1.4, but Trac goes further than Bugzilla or other issue trackers, it is a complete tool for project management. Trac gives us a very complete set of features:

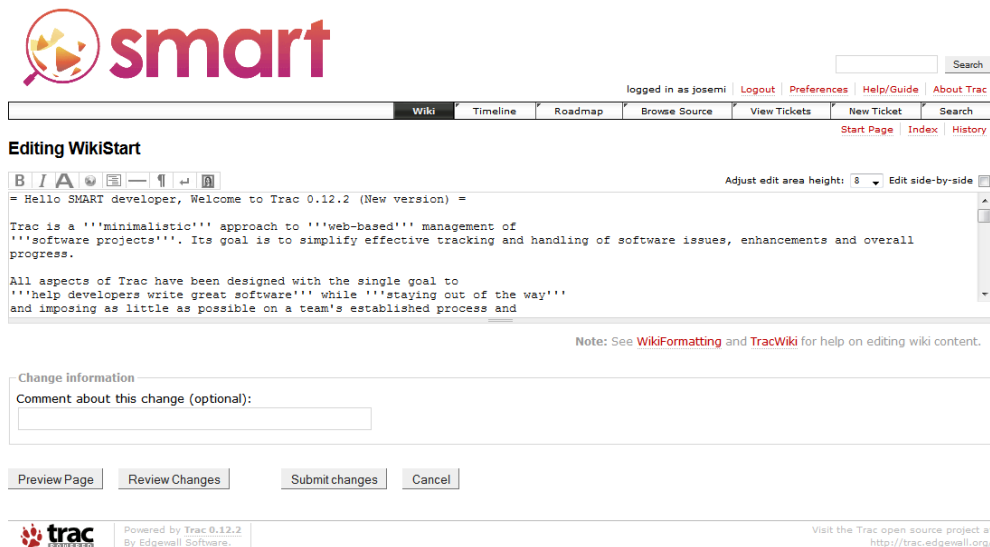
- Issue tracker: Trac features all the characteristics of a modern issue tracker. We can create “tickers”, assign it to a developer, add more information or attach files to the ticket, change the state of the ticker o generate a report.



The screenshot shows the Trac web interface. At the top is the SMART logo and a search bar. Below the logo, there's a navigation bar with links like Wiki, Timeline, Roadmap, Browse Source, View Tickets, New Ticket, and Search. The main content area displays a list of active tickets. A table shows one ticket with details like Summary, Component, Version, Milestone, Type, Owner, Status, and Created. Below the table, there are links for RSS Feed, Comma-delimited Text, Tab-delimited Text, and SQL Query. The footer includes the Trac logo, version information (Trac 0.12.2 by Edgewall Software), and a link to the Trac open source project.

Figure 3: Issue tracking in Trac

- Wiki for internal: Trac is also a wiki server. Trac has all the characteristics of a modern wiki, and it uses the syntax from Moin Moin, a wiki system used by Terrier.
- Wiki for external: We can also use the wiki characteristics for creating a public repository for external developers of the SMART system.



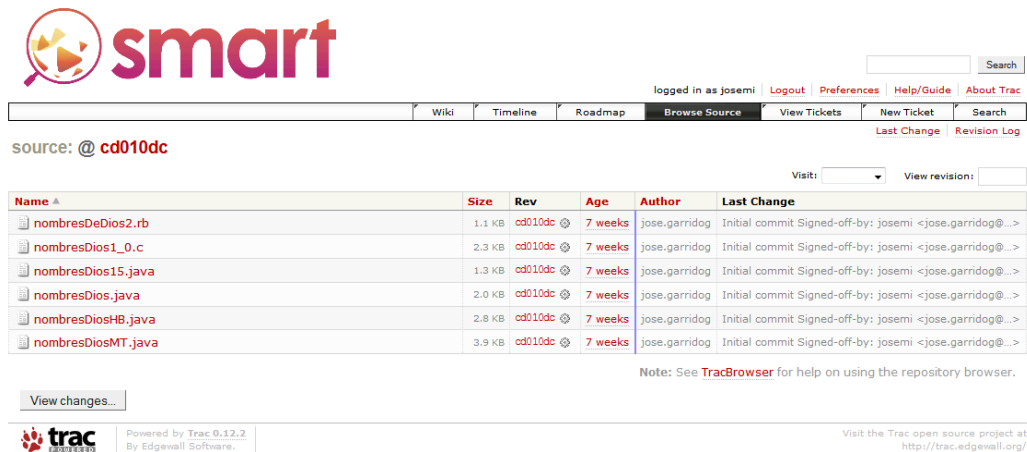
The screenshot shows the Trac wiki interface. At the top is the SMART logo and a search bar. Below the logo, there's a navigation bar with links like Wiki, Timeline, Roadmap, Browse Source, View Tickets, New Ticket, and Search. The main content area displays the 'Editing WikiStart' page. It includes a text editor with a rich text toolbar and a source code view. The text content describes the SMART project and its goals. Below the editor, there's a 'Change Information' section with a comment field. At the bottom, there are buttons for Preview Page, Review Changes, Submit changes, and Cancel. The footer includes the Trac logo, version information (Trac 0.12.2 by Edgewall Software), and a link to the Trac open source project.

Figure 4: Trac as a wiki: editing an article

- File repository: the wiki can store some files, for instance a PDF or a MS Word.



- Source code browser for Git: We can see the source code in the repository using a web browser.



Name	Size	Rev	Age	Author	Last Change
nombresDeDios2.rb	1.1 KB	cd010dc	7 weeks	jose.garridog	Initial commit Signed-off-by: josemi <jose.garridog@...>
nombresDios1_0.c	2.3 KB	cd010dc	7 weeks	jose.garridog	Initial commit Signed-off-by: josemi <jose.garridog@...>
nombresDios15.java	1.3 KB	cd010dc	7 weeks	jose.garridog	Initial commit Signed-off-by: josemi <jose.garridog@...>
nombresDios.java	2.0 KB	cd010dc	7 weeks	jose.garridog	Initial commit Signed-off-by: josemi <jose.garridog@...>
nombresDiosHB.java	2.8 KB	cd010dc	7 weeks	jose.garridog	Initial commit Signed-off-by: josemi <jose.garridog@...>
nombresDiosMT.java	3.9 KB	cd010dc	7 weeks	jose.garridog	Initial commit Signed-off-by: josemi <jose.garridog@...>

Figure 5: Code browser

- Project management: Trac features some tools for project management. For instance, it is possible to define milestones. For Trac, a milestone is a collection of issues to solve before a date.
- Timeline: Trac stores a complete record of the actions, so it is possible to see the progression of the project
- Integration: Trac is an integrated system, it is possible to include references to the code in a ticket, or to a ticket in the wiki, or include a reference to a ticket in the comment of the commit using Git.

Trac is not perfect, in any case. The current version of Trac (0.12) has some issues that the Trac team is trying to resolve.

- The Git support in Trac is not as good as SVN support. In the current version (0.12), Git is available using an external plug-in. The future 1.0 version will feature native Git support.
- Not all the functionality of Trac is available using the graphical interface. Some task must be carried out using text commands.

### 3.4.3 Technical implementation

In technical terms, the repository is an Ubuntu 12.4 LTS Server machine running in a professional hosting service. We choose Ubuntu because it is likely the most popular version of Linux that offers patches and security updates for free. Using a LTS (Long term support) version, Ubuntu creates security patches for 5 years, so we will have security support guaranteed for the life of the current phase of the project.

We use the Ubuntu version of the applications (Git, Trac, Apache...), we installed it from Ubuntu repository, not directly from each creator webpage. This way, we assure the installation of the security patches for all the elements in the server. That is very important, because we have deployed a web application available from the Internet, not an intranet, so the security against intrusion is an important matter.

The unified login system is implemented using Apache. All the user/password information is stored in a file that it is accessed by Apache. Given the relatively small number of the developers (the consortium in this first phase), this simple solution is good enough by the moment.

Trac is a web program implemented in Python, and it is server using Apache. The Trac web pages for each repository are stored in the directory `/var/lib/trac` of the server.

Git can be accessed remotely using several means, like file sharing, ssh servers and daemons. For the SMART repository, we use WebDAV, what in turn is implemented by Apache. Accessing Git this way, we can use Apache as the means of access for all the tools.

## 4 **SMART Open Source Governance**

### 4.1 **Open Source Governance Methodologies and Collaboration Schemes**

In principle, open source development promotes *agile* software development methodologies. *Agile* means that the development is not restricted to a central planning and one process from start to finish. Agile methodologies have an iterative and incremental character and rely on rapid prototyping and evolutionary development making them flexible to changes in the requirements that may emerge throughout the project life-cycle.

Usually open source projects adopt agile methodologies that allow collocated teams and individuals to plan and deliver incremental and open source software.

We identify two main streams of open source development management that both use agile methodologies but differ in the degree of centrality and transparency in decision making. We outline the framework of these management streams.

In practice variations of the described processes apply in real open source projects and some projects may adopt hybrid approaches.

#### 4.1.1 **Collaborative planning**

Under these category of management, the individual or the team who wish to make changes or add a feature submit their plan to the community (usually via an online system such as an issue tracker) and the community can then provide feedback and vote to accept the plan.

The developers then write the code and make it available for review and test. The community can then monitor the status of the feature as it is written and provide feedback.

##### Advantages:

- The entire process is transparent to the entire community.
- Decisions are taken collaboratively.

##### Disadvantages:

- Speed of development is highly dependent on good communication between the developers.

#### 4.1.2 **Master-governed planning**

Under this category, more power is given to certain individuals (masters) who are responsible for allowing code into specific part of the project.

Teams or individuals who wish to add a feature or change an existing feature may discuss the ideas with the community or just communicate with the masters.

They send their code to the master and he/she can leverage the community to accept it or not. He/she has the final say with or without discussing the issue with the community.

##### Advantages:

- Less prone to integration errors.
- Rapid development, as decisions are taken quickly.

#### Disadvantages:

- The master becomes a bottleneck. It requires highly talented and passionate developers who are willing to lead the project.

#### 4.1.3 Use Cases

In the following table we compare four different open source projects in terms of their management and governance methodologies. One of the projects is the Terrier search engine, which is among the background projects for SMART.

	<b>Ubuntu</b>	<b>Linux</b>	<b>Hadoop</b>	<b>Terrier</b>
Methodology	Collaborative planning	Master-governed	Master-governed	Hybrid
Process of developing a new feature/release	New features are called Blueprints. The blueprint online system allows the community to provide feedback on the initial plan and vote on it. Also they can later monitor the code as it is written	Linus Torvalds (original creator) makes the releases of new versions, also called the "vanilla" or "mainline" kernels, meaning that they contain the main, generic branch of development. This branch is officially released as a new version approximately every three months, after Torvalds does an initial round of integrating major changes made by all other programmers, and several rounds of bug-fix pre-releases.	Patches should be attached to an issue report. A reviewer checks the code and rejects until they are happy. Once it is committed the issue is closed.  A 'king' then manages what is committed to the branch for release	Within the Terrier team committing new code is done by agreement.  If patches are contributed from the entire community the team leader examines the code and commits
Governance	Ubuntu Community Council and Ubuntu Technical Board.  Members of both boards are nominated by the project sponsors and then voted by the community.	Linux Kernel organisation supported by a number of sponsors.	Apache Software Foundation (Also sponsored by Web companies like Google, Yahoo, etc.)	Governed by the Terrier team in the University of Glasgow

**Table 1: Open Source Management and Governance Examples**

## 4.2 SMART Project Constraints, Recommendations and Selection

### 4.2.1 Constraints Associated with the Governance of the SMART Open Source

In this paragraph we identify some constraints on the SMART project with regards to the planning and governance of the open source development of the SMART framework. These constraints will help us to identify the right the development methodology and the governance process.

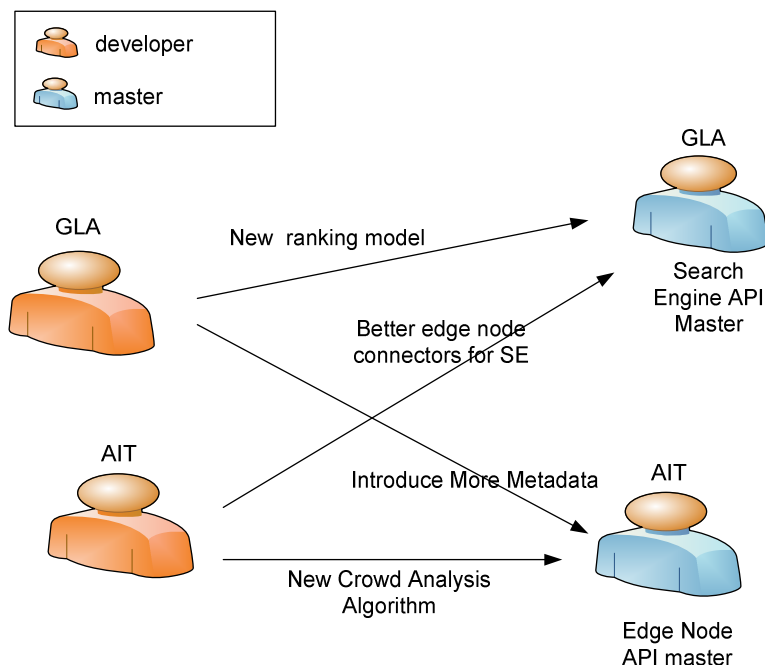
The following constraints are identified:

- Strict deadlines, deliverables have fixed dates that are not possible to change if needed. Conversely open source releases may be more flexible to changes and delays.
- Initial originators (the consortium teams) are located in distant locations and physical meetings are expensive.
- The development strategy is semi-public (See the Open Source Development document). The releases are available to the public but the planning of the future releases and the code repository is private.

### 4.2.2 Recommendation and Selection

In the light of the previous discussion, the consortium decided to adopt a management style where corresponding master of certain parts of the project are appointed (edge node, search engine, user interface, etc.). Based on the selected management style new features (patches) suggested by the consortium will be reviewed by corresponding master. They will then make decisions on the design and whether to accept or reject the patches. New releases are planned by the consortium where masters of each section can then decide on consensus which patches go to the new release.

The figure below shows different scenarios of interactions between the developers and the masters across different teams.



**Figure 6: Interactions between Masters and Developers in the scope of the SMART Governance Model**

As part of the selected governance scheme the consortium has assigned the master roles for two main sub-

projects (i.e. edge node and search engine) as follows:

- AIT will act as a master for the edge node subproject of the SMART open source.
- GLA will act as a master for the search engine subproject of the SMART open source.

In any case where new subprojects will be identified/produced, the consortium will assign a master for them as well.

Note however that the selected management style may be revised (on a merit and need basis) during later stages of the project. Decisions for changing/revising the governance scheme of the project will be based on the feedback from evaluating the experience and the practical aspects of the actual execution of the selected governance process. Hence, the consortium might opt for a more collaborative governance schemes during later stages as more developers gain experience with the SMART technologies.

After the official end date of the project, SMART would have built an open source community. It is envisaged that the industrial and the academic partners will allocate resources to support the project in line with their objectives.

## 5 **SMART Open Source Licensing**

### 5.1 **Open Source Philosophies**

There are many definitions about what is exactly an open source licence.

The first point of discussion is the name itself. Some people call it “free software”, as Richard Stallman and the Free Software Foundation (FSF)<sup>9</sup>, the creators of GNU software. They talk about 4 freedoms in the GNU philosophy<sup>10</sup>:

- Freedom 0: The freedom to run the program for any purpose.
- Freedom 1: The freedom to study how the program works, and change it to make it do what you wish.
- Freedom 2: The freedom to redistribute copies so you can help your neighbour.
- Freedom 3: The freedom to improve the program, and release your improvements (and modified versions in general) to the public, so that the whole community benefits.

The first freedom only requires that the program is “freeware” – i.e. it can be received at no cost or for an optional fee, yet with certain restrictions in terms of its use. The next 3 freedoms are characteristics of the real free software according to the FSF. In particular, these stipulate that the source code must be freely available, while also regulating the conditions under which improvement and enhancements could be released (e.g., whether they have also to be freely available).

Other people in the community call it “open source software”, such as Bruce Perens and the Open Software Initiative<sup>11</sup> (OSI). The definition according to OSI is more specific, comprised by 10 characteristics. Very briefly the important characteristics for SMART are (from the OSI webpage<sup>12</sup>):

1. *Free Redistribution*: The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale.
2. *Source Code*: The program must include source code, and must allow distribution in source code as well as compiled form.
3. *Derived Works*: The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.
4. *License Must Not Restrict Other Software*: The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be open-source software.

In practice, the OSI recognises that open source software may be commercialised, extended, or deployed, as long as the open source part is available. We use the OSI definition and the OSI terminology because it is considered less restrictive and friendlier for commercial companies.

There are many open software licenses. Usually, we can classify them into restrictive or permissive:

- Restrictive licenses
  - Strong *copyleft* (GPL): Redistribution of modifications or derivative works must be made under the same license

---

<sup>9</sup> <http://www.fsf.org/>

<sup>10</sup> <http://www.gnu.org/philosophy/free-sw.html>

<sup>11</sup> <http://www.opensource.org/>

<sup>12</sup> <http://opensource.org/docs/osd>

- Weak *copyleft* (LGPL, MPL, EPL): The redistribution must remain with the original license, but they can be combined with other licenses
- Permissive licenses (BSD, MIT, Apache)
  - Redistribution of any modified code from the original licensed can be delivered under any type of license (even proprietary)

The main danger of permissive licenses is that the code can be “stolen” and included in proprietary applications by a third party. In the other end, restrictive licensed as GPL are accused of being “viral”, as any (for instance) GPL components or work derived from a GPL work must remain as GPL (and have the source code available).

## 5.2 Main Open Source Licensing Requirements of the SMART Consortium

For the purposes of the SMART consortium, the choice of open source license should be made while considering several requirements. These requirements are based on the intended stakeholder and 3<sup>rd</sup> party commercialisation routes. In particular, the DoW (B2.1) identifies key actions for assuring impact that are relevant:

- A. Supporting and building an open source community around the project
- B. Executing the partner’s exploitation plans, namely:
  - i. Product support services, value-added consultancy
  - ii. Service provision, Smart city deployment
  - iii. Research dissemination from the framework

From these actions, we identify the following desirable requirements for the open source license. It is important to mention here that these requirements are fixed and the consortium needs to prioritise these requirements if they cannot be all met at once.

### 1. OSI-approved open source license

The open source SMART framework should adopt a reputable open source license, such as one accepted by the OSI, such that an open source community can be formed (A), and that product support services and consultancy services can be made (B(i)).

### 2. Redistributable in unmodified form on a commercial basis

Partners and other 3<sup>rd</sup> parties should be able to commercialise products based on the SMART (B(ii)).

### 3. Redistributable in modified form on a commercial basis:

- a. Without (changed) source code
- b. With (changed) source code

Partners and other 3<sup>rd</sup> parties should be able to commercialise products based on the SMART (B(ii)) should experience no risk to their commercial derivative works from viral licenses.

### 4. Compatible with current Terrier license (MPL v1.1)

As the Terrier search engine platform is a key component of the SMART solution, the chosen license should be compatible with the license used by Terrier. This also supports GLA’s current dissemination plans from the framework (B(iii)).

### 5. Can use libraries without license contamination

Various open/free source libraries exist that may be beneficial to the SMART project which may be licensed under different open source licenses. Any chosen license should consider if such libraries cause the source code of the main product to be contaminated.

## 6. Maximise the chances of acquiring improvements from 3rd parties who redistribute the code commercially.

Although the license should be commercial friendly (requirement 3) as this is strategic to the consortium, it should maximise the chances of obtaining improvements from 3<sup>rd</sup> parties who redistribute the framework in commercial applications.

In the following section, we discuss some common open source licenses, their pro/cons, and how they are compatible with the requirements identified here.

## 5.3 Open Source Licenses Considered by the SMART Consortium

The consortium considered the following representative sample of popular open source licenses:

- GPL: Gnu Publish License (<http://www.gnu.org/copyleft/gpl.html>)
- LGPL: GNU Lesser General Public License (<http://www.gnu.org/licenses/lgpl.html>)
- MPL: Mozilla Public License (<http://www.mozilla.org/MPL/>)
- BSD: Berkeley Software Distribution ([http://en.wikipedia.org/wiki/BSD\\_licenses](http://en.wikipedia.org/wiki/BSD_licenses))
- Apache license (<http://www.apache.org/licenses/>)

The table below lists those licenses and how they meet the requirements of the SMART framework.

Requirement	GPL	LGPL	MPL	BSD family (e.g. Apache)
1. OSI-approved	Yes	Yes	Yes	Yes
2. Redistributable in unmodified form on a commercial basis	Only if the source code is available and all referenced libraries are GPLed.	Yes, even if the referenced libraries are proprietary.	Yes, even if the referenced libraries are proprietary.	Yes, even if the referenced libraries are proprietary.
3. Redistributable in modified form on a commercial basis	Redistributable only if the derived software is GPL and the source code of changes is available	Redistributable only if the derived software is GPL or LGPL and the source code of the changes is available	Redistributable but the changes should be available under the MPL license.	Yes (Redistributable <i>with or without</i> releasing the source code of changes)
4. Compatible with Current Terrier License	No	Yes	Yes	Yes
5. Can use libraries without license contamination	No, can only use GPLed libraries except for system libraries.	Less viral than GPL	Yes, new files containing no MPL-licensed code are not Modifications, and therefore do not need to be distributed	Yes



			under the terms of the MPL.	
6. Chances of acquiring improvements from 3rd parties who redistribute the code commercially	Very high	High (LGPL ensures that any changes on the library level should be committed).	High (MPL ensures that any changes on the file level should be committed).	Low (No enforcement for committing changes)

**Table 2: Comparative Overview of the main licenses considered by the SMART Consortium**

### 5.3.1 GPL

GPL is a restrictive license as it enforces the distribution of any derivatives under the GPL license and therefore considered “viral”. Derivative works include those using a GPLd library (although a few libraries using GPL with an exception, such as the GPL classpath exception). For this reason, the GPL does not meet most of the requirements we identified above and therefore it is not recommended for the SMART framework.

### 5.3.2 LGPL

The LGPL is quite similar to the GPL (“viral”) for the internal code of the library, but explicitly allows the linking with external modules not GPLed. Some of the most famous libraries in the open source community are LGPL, like the C standard library for GNU C itself.

However the use of LGPL is now discouraged, even by GNU (mainly in preference of GPL and its inherent stronger support for free derivatives<sup>13</sup>). In our case, the LGPL can be too restrictive for enterprise use. We can mix the library with the proprietary modules of SMART, but it enforces the use of LGPL for all the future derivate works modifying the library itself.

### 5.3.3 MPL

The Mozilla Public License is a “copyleft” license quite similar to LGPL, but the file is the limit, not the whole library. So it is possible to combine MPL code with proprietary code with fewer restrictions, so it is more “company friendly”.

The main disadvantage of MPL used to be that it was impossible to combine it with GPL code. Fortunately, the MPL 2.0 (published last January) has addressed this disadvantage<sup>14,15</sup>.

The main advantage for the SMART project is that Terrier uses MPL (1.1). If we choose MPL, we have to decide between the 1.1 and 2.0. It makes sense to choose 2.0 for any new software, as there is no incompatibility between MPL 1.1 and 2.0.

It should be noted that MPL can have some minor implications on the commercial use. For example, one of the use cases suggested by S3Log is security. Implementing such a system for security purposes may require additional protection measures such as encryption of data and communication between different parts of the system. This may require integration of encryption code directly into the files at different levels. However, the encryption code might be classified, proprietary or controlled by different regulation and therefore it can't be published. In some cases, the encryption itself can be taken out from the OS files and only the function calls are kept in those files but this also might not be always possible as sometimes it would be impossible even to publish that a specific feature is implemented (e.g. for legal requirements). Nevertheless, it is still possible implement the encryption in separate proprietary code, e.g. by implementing a high level interface that the framework provides.

<sup>13</sup> <http://www.gnu.org/philosophy/why-not-lgpl.html>

<sup>14</sup> <http://www.mozilla.org/MPL/2.0/>

<sup>15</sup> <http://www.mozilla.org/MPL/2.0/FAQ.html>

#### 5.3.4 BSD & Apache

Another family of licences is the BSD license and derivatives, such as the MIT license or the now popular Apache licence. The main difference is that using the BSD license, the free code can be mixed with proprietary code and included in commercial products without restrictions, even in products from third parties, and without returning the improvements to the community.

For some open source advocates, this is the equivalent of stealing and they discourage the use of BSD, because the free code can be “stolen” from the community. There are many examples of this behaviour. For instance, many years ago Microsoft took the BSD-licensed code from the BSD Unix and included it as the TCP stack of Microsoft Windows. Apple took the code of the kernel of BSD Unix as the core of the kernel of his own Mac OS X. (Apple created the open Darwin project, but by public relationship considerations). More recently, Apple took the Apache licensed code of Readability and included it in their proprietary Safari browser and iPad.

The Apache license “is permissive like BSD, but (unlike BSD) actually happens to mention the rights under copyright law and gives you a license under those rights. In other words, it actually knows what it is doing unlike some of the other permissive licenses”<sup>16</sup>. Moreover, “the Apache 2.0 licenses contain a patent grant, which means that the authors of the code are giving the receiver any rights that you need for the authors' patents that happen to be in the code that you are using”<sup>17</sup>.

In our case, the major disadvantage of the BSD/Apache licenses is that they can make it easy for any third party to make money from research funded by the tax payer without contributing to the open source community.

### 5.4 Recommendations and Selection

As a summary the MPL and BSD licenses are the most suitable ones for our requirements. However the requirements should be prioritised by the consortium to be able to make a decision on which license to choose. In particular, requirement 3(a) can be met by BSD licenses and partially by MPL, however MPL meets requirement 6 better than BSD licenses.

**In light of how each license addresses the requirements listed in Section 3, we recommend the MPL2.0 license for the SMART open source framework.**

Based on the requirements above, the participants in the SMART project will have almost no restrictions on any third-party software library. However for some licenses there might be limitations/constraints on using the 3<sup>rd</sup> libraries within the SMART framework. The table below lists some OSI approved licenses and any limitations that consortium members should be aware of. In particular, for each license, we consider three different checks that need to be considered before using a 3<sup>rd</sup> party library with that license.

3 <sup>rd</sup> party Library License	Use in SMART MPLeD code	Use in SMART proprietary code	Release changes to library
GPL	Yes	No	Yes
LGPL	Yes	Yes	Yes
MPL	Yes	Yes	Yes
BSD	Yes	Yes	No
Apache	Yes	Yes	No

**Table 3: Checks associated with the use of 3rd party libraries in SMART**

<sup>16</sup> <http://blogs.zdnet.com/Burnette/?p=192>

<sup>17</sup> <http://stackoverflow.com/questions/40100/apache-licence-vs-bsd-vs-mit>

## 6 Conclusions

SMART will be releasing its multimedia search engine for environment generated content as open source software. Therefore, the setup of the project's open source portal is an important deliverable and milestone of the project, given that it is a prerequisite for releasing the SMART software as open sources. Note however that the setup of the project's open source portal is associated with some key decisions driving the open source development processes, as well as evolution and exploitation of the open source components of the project. As part of this deliverable the SMART consortium has dealt with all these issues, with a view to setting up the open source properly, while at the same time commencing the open source development.

The open source portal of the project has been made available at: [opensoftware.smartfp7.eu](http://opensoftware.smartfp7.eu). It is primarily based on two open source tools, Trac (for bug tracking) and Git (for versioning). A hosted solution (web-based) has been implemented. Following the establishment of this portal, all developers within the project have been creating accounts and acquainting themselves with the development environment and tools. The portal will host several of the coming deliverables of the SMART project (notably the open source prototypes).

Along with the establishment of the open source portal, the consortium has also selected the license of the open source software. Following an analysis of popular licensing schemes in terms of their pros/cons and their alignment to the SMART targets/goals, MPL2.0 has been selected. This licensing scheme provides a good balance between openness and opportunities for commercial exploitation, given that it is a business friendly license. Hence, it satisfies requirements for compliance with background projects, while at the same time ensuring that the companies of the consortium (notably the solution providers/integrators ATOS, S3LOG, IBM, TELESTO) will be able to build exploitable solutions on top of the SMART platform (without having to release their own/proprietary add-ons as open source).

SMART has also investigated the open source development processes, as part of wider governance schemes associated with the project development. The aim of the investigation was to balance between ease of integration, robustness and the participatory nature of the various processes. Instead of a collaborative planning approach (based on the equal participation/collaboration of all contributors), the consortium has opted for a master-governed planning approach. The latter emphasizes the leading role of a master in the integration process. In this way, complex integration tasks are facilitated and the code is likely to be more robust, especially if the master is a highly skilled and dedicated individual. The selected master-governed approach is expected to boost the quality and rapid evolution of the project during early stages. SMART may consider a shift to the collaborative-planning approach (or even a hybrid approach) in later stages of the project.

The present deliverable is expected to be valuable to SMART consortium members and third-parties (e.g., community members) engaging with the SMART open source development endeavor.