Considerations on Informatics Systems with Free/Open Source Software: the Environmental Information Systems Example

Kostas Karatzas, Asterios Masouras, Athina Kaprara and Anastasios Bassoukos

Aristotle University, Department of Mechanical Engineering, Thessaloniki, Greece
kkara@eng.auth.gr, {oneiros, akap, abas}@isag.meng.auth.gr

Abstract. Improved access to environmental information is the basis for a higher degree of involvement of citizens and stakeholders in environmental decision-making. Environmental Information Systems play a key role in contemporary urban environmental management strategies, and are a prerequisite for the proper, timely information of the public; yet the fuzzy nature of environmental information requires for systems that can make optimum use of informatics and telecommunications infrastructures to address environmental management needs, while remaining open-ended, easy to use and inexpensive to implement and operate. In this paper, we will attempt to present the characteristics of Free/Open Source software that render it appropriate for use in developing Environmental Information Systems, accompanied by real world project examples.

1 Introduction

The Free/Open Source Software (FOSS) movement is a new software development paradigm that emerged in the last decade and relies directly on the volunteer efforts of geographically dispersed developers of varying affiliations and proficiencies. In contrast with previously established practices, FOSS recognizes a number of "freedoms" granted to the user regarding his ability to interact with the software and propagate its use. Unrestricted access to the software source code is a precondition for most of these freedoms, and it is implied that the usefulness and "life expectancy" of such software is dependent on the continual revision and adaptation of its source code.

FOSS developers minimize redundancy by reusing and adapting freely available best practice software and methodologies, thus concentrating investment on innovation. The support FOSS projects receive from the user-developer community serves to provide guidance, reduce maintenance costs and enhance software usability, therefore, the functionality and maintainability of software developed in this way is not impaired by artificial limitations. Furthermore, the service-oriented model of FOSS allows for a broad range of contractors to provide support, and since service and consulting fees are the only recurring expenditures, the Total Cost of Ownership for solutions based on FOSS software is kept low.

2 Using FOSS Software Resources

In the last decade, a new model has emerged for managing software projects that relies directly on the common efforts of geographically dispersed developers of varying affiliations and proficiencies. This software development paradigm is fuelled, in direct contrast with previously established business practices [1], by full disclosure of the source code, volunteer effort and a number of "freedoms" granted to the software user regarding his ability to interact with the software and propagate its use. It is these characteristics of Free/Open Source Software (FOSS), which render it flexible, economical and reusable, that make it appropriate for use towards building publicly funded ICT projects [2], especially those aiming at the dissemination of information to citizens, such as online environmental portals.

The definition of Free Software recognizes some fundamental freedoms as imparted by the author (the "licensor") (http://www.gnu.org/philosophy/free-sw.html) to the user (the "licensee"), inside a license agreement:

- The freedom to study how the program works, and the freedom to adapt the code according specific needs
- The freedom to improve the program (enlarge, add functions);
- The freedom to run the program, for any purpose and on any number of machines;
- The freedom to redistribute copies to other users.

The Open Source definition (http://www.opensource.org/docs/definition.php) further extended these principles. Open Source Software and Free Software are terminologies which have been used to describe software developed and distributed on the above and similar principles, with terms such as Libre software [3] used to group them together. Although the terms and their underlying movements are not fully interchangeable, mention of them throughout this paper should be taken to be, for purposes of simplicity.

Unrestricted access to the software source code is a precondition for most of these freedoms, and it is implied that the usefulness and potential for reuse of such software is dependent on the continual revision and adaptation of its source code. It follows then that the "life expectancy" of software developed in this way is a direct outcome of its popularity with developers (who will choose to devote time to improve functionality) and users (who provide constant feedback to developers on needed improvements and fixes).

The FOSS software community consists of individuals or groups of individuals who contribute to a particular FOSS product or technology: as a consequence of the previous statement, this also includes the users of the software. The FOSS process refers to the approach for developing and maintaining FOSS products and technologies, including software, computers, devices, technical formats, and computer languages.

The use of FOSS software towards building environmental information systems hinges on three points [4] providing benefits to users, developers and operators of the software: economy, quality and philosophy.

2.1 Economy

Reusing and adapting freely available best practice software, instead of resorting to monolithic proprietary solutions or developing everything from scratch leads to minimizing redundancy in development efforts (thus, by extension, concentrating investment on innovation) and obtaining the best value for the citizens' money, while relying on the community to spark developer interest in the software and provide user feedback reduces maintenance costs and prolongs its' useful life cycle. A corollary of this is that the functionality and maintainability of the software is not impaired by artificial limitations (i.e. not intrinsic to the software itself), such as expiring licenses and financial plights to the parent company.

The Total Cost of Ownership (TCO) of solutions based on FOSS from a contractor point of view is alleviated [3], [5], since consulting fees are seen as fully useful expenditures, in contrast with licensing fees which mostly serve as instruments of amortization on behalf of developing companies. Since Environmental Information System development is often supported by public funding, no such amortization must burden their future customers past project end. For the service-oriented model of FOSS it should be noted that costs of support and maintenance can be contracted out to a range of suppliers, as per the competitive nature of the market enforced by source disclosure [6].

2.2 Quality

The main objective in software engineering is not necessarily to spend less but rather to obtain a higher quality for the same amount of money, and aim to enforce the best possible safeguards for quality and safety in the product. Avoiding to "reinvent the wheel" by using funds to develop new applications rather than re-inventing already developed parts, as pointed out elsewhere, speeds up technological innovation -as is also the case with the increased cooperation and full source code disclosure and availability required by FOSS tenets. The scientific results of such projects are subject to peer scrutiny and directly comparable with similar endeavors, thus ensuring conformance with best practices to a large degree. Finally, as has been repeatedly demonstrated in recent years [7], [8], software security concerns are better addressed through a continuous process of issue disclosure and user-developer cooperation in order to overcome them.

2.3 Philosophy

Reliance on proprietary software for science results in vendor "lock-in" as regards to data formats, making it difficult to pursue common protocols for data interchange and storage, for instance, as it is required by modern systems dealing with the problem of environmental data heterogeneity [9]. In contrast, FOSS developers and proponents promote the use of open scientific protocols, through their use in applications, as a means of consolidating researcher efforts, minimizing the cost and dependencies of technical innovation.

In promoting best practices, transparency and quality control in software design, the FOSS movement serves also to further collaboration between public bodies, professional communities and the private sector in the interests of creating a flexible and lasting service environment for the public. The free dissemination of technological advances (both in terms of cost and material availability) relating to informatics services, although not a panacea, can be seen to eventually help eclipse the digital divide [10], by allowing poorer countries to "catch up". Finally, the source code itself can be seen as literature [11], [12], thus as a human endeavor to be preserved as a cultural heritage for the future generations.

3 Ecological Information Systems

The holistic nature of ecological information systems requires them to deal with the problem of environmental data heterogeneity, by adopting common protocols for data interchange and storage. FOSS promotes the use of open data standards as a means of consolidating researcher efforts and increasing technical interoperability. Thus it can be demonstrated that the right of public access to environmental information, as has been defined in contemporary legislation, is better served by utilizing open, flexible and low cost dissemination platforms that make use of software developed by the community. In the following chapter, examples of FOSS applications are presented, all related to air quality management systems and all addressing problems that converge into the need of openness, flexibility, adaptability, resource optimum, environmental management solutions.

4 Project Examples

In the following, we present a sampling of informatics projects funded under EC initiatives that strived to produce environmental-ecological information systems for public use. Proceeding from the early recognized need for open architectures, platform independence and common data protocols, these systems gradually came to fulfill and embrace the philosophy and principles of the FOSS model, while development of the last example mentioned was largely based on FOSS software tools

4.1 The APNEE/APNEE-TU Projects

The APNEE project (http://www.apnee.org) [13], [14], contributed to the European research on public information systems and services, by developing citizen-centered dynamic information services aimed at providing intelligence about the ambient environment. These services advise the citizen about the air quality in terms of an air quality index and offer guidelines for behavioural change. Awareness services are based upon an array of information channels to reach the citizen. APNEE further utilises various intuitive presentation formats to convey information. The

configuration of such ambient technologies and the selection specific information channels has been evaluated in field trials in different European regions.

It was apparent from the beginning of the APNEE project that a flexible, modular and cost-effective architecture was needed, to support the environmental information needs of urban agglomerations through easy-to-use and easy-to-access interfaces that would allow a measure of personalization / customization in order to prove attractive to citizens. For this reason, development of the APNEE regional server was based on FOSS technologies.

APNEE / APNEE-TU is composed of a set of reference core modules, including the database, the service triggers, the regional server application and basic functionality modules (licensed as FOSS), as well as proprietary extension modules developed by telecommunication partners to provide services based on local ICT infrastructure conditions. Although the core modules are considered to be the heart of the system, yet, they may be "by-passed" or not implemented, in cases where only the electronic services are of interest for installation and operational usage, provided that there is a database and a pull and push scheme provided via alternative software infrastructures. The modules and the interfaces that have been developed to build the regional server are the following:

- *APNEE Environmental Database*: the database forms the back-end of all APNEE-TU services and consists of a schema for environmental data series, as well as warnings, medical advises, pollutant information; spatial data for the WebGIS component, and user information and personalization data for subscribers. APNEE-TU provides an object-relational persistence layer to allow cooperation with a variety of FOSS and commercial RDBMS systems. The environmental database is thoroughly described in APNEE D3.
- *APNEE Regional Server*: the centerpiece of the APNEE platform, the regional server provides a web-based anchoring point for APNEE services, configured and localized as per the needs of each installation site; also provides administrative interfaces for a variety of functionalities, such as subscription to the newsletter, and email services.
- *Push services*: these services consist of modules that are executed on when changes in the database occur. What kind of database change that will launch a module, is configured in the trigger. Push services consist of sending SMS and email messages to the citizen periodically, or upon user specified conditions, and are mostly used to send out alerts and warnings.
- *Pull services*: these services are used whenever another application requests information from the database. This includes requests made from users via WWW, PDA, or WAP, and requests from automatic processes using the XML-RPC or SOAP interface.

Implementation of the APNEE regional server was based on Java servlets [15], server-based web applications, and a variety of technologies made available by the Apache Foundation (Fig. 1). The APNEE regional server programmatically provided service front-ends to the environmental database through an object relational persistence layer (Apache Torque [16]), as well as Web Services [17] interfaces (via XML-RPC [18] and Apache Axis [19]) Development of the regional server application itself was based on Jakarta Turbine [20], a modular service oriented web

application framework: Jakarta Velocity, a dynamic HTML-based template engine; and Apache Ant, an automatic build and deployment scripting system. Finally, Jakarta Tomcat was chosen as the default servlet container for both development and production use.

In recognition of the value of the FOSS software paradigm towards building informatics systems for the public, and in order to preserve and disseminate development efforts, the APNEE-TU project produced a "reference implementation", composed of the environmental database, regional server and core modules, which is licensed as FOSS.

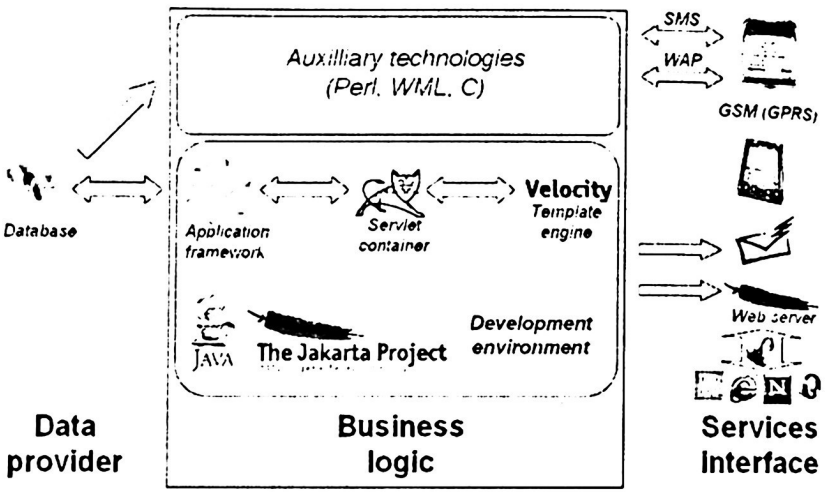

Fig. 1. The APNEE-TU services architecture materialisation for Thessaloniki, Greece

4.2 The ECOSMES Project

In the frame of the e-LCA project (http://www.elca.enea.it/), a system consisting of web-based applications for supporting the adoption of Integrated Product Policies in SMEs was developed. The authors undertook the development of a number of modules-applications, including news, events, consultancies, a documents file manager, an eco-products list, a newsletter, a mail notification service, and a contacts and links section, on the basis of predefined functional specifications. These modules represent a significant part of the ECOSMEs web site (http://www.ecosmes.net/) that is designed to help SMEs carry out assessments on their products to establish their environmental impacts and assist with eco-design.

Tools and technologies used

From the start of the project, several FOSS tools suggested themselves for use in rapid prototyping and development, due to the intense constraints externally imposed on the development team:

Tomcat was naturally selected as the application container, as the team had extensive previous experience with its use in past projects. Tomcat being the reference implementation of web application servlet containers, and having proven a robust choice for both development and production use it was felt that it would be a more than adequate choice.

Torque [16] was chosen as the Object-Relational bridge. While it is considered too intrusive by some, as business objects will need to inherit from Torque-generated classes, it was also utilised in the APNEE project and the team had fluency in using it.

CVS [21] was used for source code revision control, and it is considered an industry standard. While *Subversion* [22] was also considered for use, it was rejected because at project start it was still an unproven technology.

Eclipse [23] was chosen as the IDE. Eclipse was a relatively new addition to the team's toolbox, with less than two months usage before the project start, but proved an extremely usable and productive tool, especially where extensive automated (eg. refactoring) or team (eg. code review) operations where concerned.

Lomboz [24] was the plug-in used to add J2EE capabilities to Eclipse.

SysDeo offered its excellent Tomcat integration plug-in for Eclipse, of which heavy use was made.

PostgreSQL [25] was the database specified by the technical specifications; as the team had been using it as a default development and production RDBMS, it was felt that its use would be more than adequate for the requirements of the project.

The development process

While the team had previous experience with CVS, it was in the context of slow, medium-sized updates of about once per week to once per month. The pressure on the team, combined with the ease-of-use of productive tools like Eclipse, amalgamated to give the development process an almost Extreme-Programming-esque [26] quality, as the process was almost completely driven by Continuous Integration, concurrent, multiple commits, and sometimes even pair programming.

The final web application delivered consists of more that 24000 written lines of code plus 65000 lines of automatically generated code via the development tools spread out over 426 Java classes.

Experience gained

Due to the constrained schedule of the project, several other technologies were considered but rejected due to retraining costs that would have to be incurred. In hindsight however, it is apparent that their adoption would have improved the teams' overall productivity. For example, a good portion of the project involves data form entry, verification and presentation, for which Struts would seem the obvious choice; however, questions about possible integration issues with the rest of the application were not resolved early enough, resulting in its exclusion from the project. The team also learned to rely on its tools and make optimal use of each member's potential,

while face-to-face dialogue emerged as the most potent communication medium in time limited situations. Finally, valuable insights were gained into the complexities of implementing true multilingual web applications, which can only be derived by actual development of similar solutions.

5 Conclusions

The need for collaborative information management modules that would allow for implementing a homogenized, service-based, user perspective of heterogeneous environmental-ecological data and computational resources was the main drive towards modular and open software architectures in projects such as the ones previously mentioned. It was also made apparent that project design and development could benefit from the use of Free/Open Source Software. This was mainly initiated within the last decade via the usage of Internet based communication infrastructures, leading to the flourishing of platform-independent (eg. Java based) software module implementations and the need for cost-effective and reliable informatics solutions. APNEE/APNEE-TU verified the trends outlined above and for the first time provided a flexible, cost-effective working solution for implementing a public environmental information ICT infrastructure, making use of resources developed by the FOSS community. In addition, the ECOSMEs project experience verified the power, flexibility and effectiveness of FOSS under intense and demanding software development procedures.

6 Acknowledgements

The authors greatly acknowledge the European Commission for supporting research projects APNEE and APNEE-TU (DG Infso, Systems and Services for the Citizens) and project ecoSMEs (DG Infso, e-content). Prof. Nicolas Moussiopoulos who co-ordinated the participation of the Laboratory of Heat Transfer and Environmental Engineering (LHTEE) of the Aristotle University of Thessaloniki to these projects is also acknowledged, as the authors were affiliated to LHTEE during the time they performed the work described in the present paper.

References

1. Raymond, E. S.: The Cathedral and the Bazaar (2000). http://catb.org/~esr/writings/cathedral bazaar/
2. Berlecon Research, ProActive Int.: /Free/Libre and Open Source Software (FLOSS) Survey and Study (2002). http://www.infonomics.nl/FLOSS/index.htm
3. EWGLS-European Working Group on Libre Software: Free Software / Open Source: Information Society Opportunities for Europe (2001), http://eu.conecta.it/paper.pdf
4. Interchange of Data between Administrations -European Commission, DG Enterprise: Pooling Open Source Software An IDA Feasibility Study (2002).

http://europa.eu.int/ISPO/ida/jsps/index.jsp?fuseAction=showDocument&parent=crossreference&documentID=550

5. Mitre Corporation: A Business Case Study Of Open Source Software (2002). http://www.mitre.org/work/tech_papers/tech_papers_01/kenwood_software/kenwood_software.pdf
6. Lerner, J., Tirole, J.: The Simple Economics of Open Source (2002). http://ideas.repec.org/p/nbr/nberwo/7600.html
7. Perens, B.: Why Security through Obscurity Won't Work (2001). http://slashdot.org/features/980720/0819202.shtml
8. Schneier, B.: Full Disclosure, Crypto-Gram Newsletter issue 111 (2001). http://www.counterpane.com/crypto-gram-0111.html
9. Visser U., Stuckenschmidt H., Wache H., Vögele Th.: Using Environmental Information Efficiently: Sharing Data and Knowledge from Heterogeneous Sources (2001). http://citeseer.nj.nec.com/309536.html
10. Schauer, Th.: The Sustainable Information Society, Visions and Risks (2003). http://www.global-society dialogue.org/saskia.pdf
11. Knuth, D.E.: Literate Programming. CSLI Lecture Notes 27. ISBN 0-937073- 80- Stanford (1992)
12. Graham, P.: Hackers and Painters (2003). http://www.paulgraham.com/hp.html
13. Bohler, T., Karatzas, K., Peinel, G., Rose, Th., Jose, R.S.: Providing multi-modal access to environmental data-customisable information services for disseminating urban air quality information in APNEE. Computers, Environment and Urban Systems 26(1) (2002) 39-61.
14. Karatzas, K., Masouras, A., Kaprara, A., Bassoukos, A., Papaioannou, I., Slini, Th., Moussiopoulos, N.: Environmental Information Systems and the Concept of Environmental Informatics. In: Scharl, A. (ed.): Environmental Online Communication. Advanced Information and Knowledge Processing Series. Springer-Verlag, Berlin Heidelberg New York (2004) 3-10
15. Servlets: Java Servlets technology. http://java.sun.com/products/servlet/
16. Torque: a Java persistence layer. http://db.apache.org/torque/
17. Apache Web Services Project. http://ws.apache.org
18. XML-RPC, XML-based Remote Procedure Calls. http://www.xmlrpc.com/
19. Axis: SOAP implementation for Java. http://ws.apache.org/axis/
20. Turbine – a web application framework. http://jakarta.apache.org/turbine/
21. CVS – Concurrent Versioning system. http://www.gnu.org/software/cvs/
22. Subversion – a Version Control System. http://subversion.tigris.org/
23. Eclipse: An Extensible IDE. http://www.eclipse.org/
24. Lomboz, an J2EE plugin for Eclipse. http://forge.objectweb.org/projects/lomboz
25. PostgreSQL, an Open Source Relational Database Management System. http://www.postgresql.org/
26. XP: Extreme Programming, an Agile development methodology. http://www.extremeprogramming.org/