Free/Open Services (F/O-Services)

G.R. Gangadharan, Politecnico di Milano, Italy

Vincenzo D'Andrea, University of Trento, Italy

Michael Weiss, Carleton University, Canada

Even if an apple is available freely on your way, you have to consider the issues of property rights before the consumption of it. If, for an apple, considering the rights of consumption is significant, then for software which is freely available, it becomes inevitable to consider the associated rights. The Free/Open Source Software (FOSS) [1] approach protects the unconditional rights of modification and redistribution by the collaborating developers, making the source code freely available. The freedom in software is reflected by the software license describing terms and conditions for use and distribution.

Service-oriented computing (SOC) represents the convergence of technological enablers with an understanding of cross-organizational business processes [2]. Services enhance the World Wide Web model not only for human use, but also for machine use by enabling application level interactions. Services deliver complex business processes and transactions as well as simple functions, allowing applications to be constructed on-the-fly and to be reused. Service composition [3] can be perceived as a way of developing a new service, a composite service, whose execution depends on the service(s) being composed. Services can be composed statically, at design time, when the services to be combined are selected, linked, and deployed. Alternatively, services can be selected and composed dynamically during runtime.

Though services are software fragments, services differ from software in several ways. Generally software serves as a stand-alone application. Services intend to make network-accessible operations available anywhere and at anytime. Unlike software, services are not resident in the recipient's environment. For that reason, in general service consumers have no access to the implementation details of a service, including whether or not a service uses other services, and what are these other services. Services are executed in the hosted infrastructure and a consumer sees only the result of execution.

The seamless proliferation of SOC demands the thoughts related to the ownership and distribution aspects, to enable widespread use of services. An approach similar to FOSS that opens the accessibility of the source code of services and the execution/use of services, would significantly enhance the understandability of the service composition process (including data and control flow) and allow the creation of derivative services. Adopting and adapting the principles of FOSS approach to SOC could enhance the widespread use of services. In this paper, we illustrate the concept of Free/Open services (F/O-Services), inspired by FOSS movement over SOC.

Unleashing F/O-Services

F/O-Services significantly enhance the way of usage and distribution a service as follows.

Service Usage

Service usage describes the freedom to use a service by other applications, for any purpose. The basics of F/O-Service allows the use of service by any other application, both service-oriented or not, in agreement with the given F/O-Service license. With the creation of F/O-Service, we are provided with the freedom to know how the service works and could be adapted to our needs, making the source code of service interface as well as service implementation freely available. A service is often created as a wrapper of another software application, in that case the availability of source code of the F/O-Service would not necessary include the source code of wrapped software (if, for instance, the wrapped software is a proprietary application).

Service Distribution

Service distribution describes the freedom to distribute a service as a new, separate service. The new service could be implemented as an independent entity or invoke the first one. Furthermore, this offers the freedom to improve the service, and release improvements to the public, so that the whole community benefits. F/O-Services allow to perform modifications on the interface and implementation of the service and thus, derived services are created. Derived services could be executed independently (together with separate interface and implementation) or could use the implementation of the parent service.

Following the definition of FOSS[1, 2], we define a F/O-Service as follows [14].

A F/O-Service should be free for use.

The source code of the interface (WSDL descriptions) as well as the implementation of a F/O-Service should be available.

The service implemented by creating a new service using the source code and interface of a F/O-Service should be freely distributable as an independent service. The modification of interface and implementation should be permitted.

The service using a F/O-Service as part of a composite service should be freely distributable as an independent service, even when using a separate interface. The modification of interface and implementation should be permitted.

Derived services and modified services must be allowed and be capable of distribution.

The license must not discriminate against any person or group of persons or any field of endeavor.

The license agreement must provide a F/O-Service "as is" with no warranties either to functional and/or non-functional properties or non-infringement of third party rights.

The license must not place restrictions on composition with other services and on distribution of composed services.

1. http://www.gnu.org/philosophy/free-sw.html

2. http://www.opensource.org/docs/osd

Exploring Freedom and Openness in F/O-Services

In this section, we analyze the possible scenarios in connection with F/O-Services using three dimensions to characterize different approaches: the possibility to modify the service interface, the possibility to modify the service implementation, and the possibility to execute a service independently.

By *execution independency*, a service can be executed in a different context or that can be owned and/or maintained by a different organization. By *execution dependency*, a new service can be created from a F/O-Service in such a way that the service needs not to be implemented again. The operations of a F/O-Service can be invoked and executed directly on the host of the F/O-Service itself by another services in execution dependent way.

Furthermore, a F/O-Service allows modification of interface or/and implementation. *Interface modification* is a common scenario in SOC as the source code of service interfaces must be generally available. The *modification of implementation* allows to create value added services beyond composition.

Now, we exemplify the freedom and openness exclusively associated with F/O-Services based on possible combinations of modification (or not) of service implementation, modification (or not) of service interface, and independent (or dependent) execution as follows (see Figure 1).

In all scenarios, we consider S_A as a F/O-Service providing a spell checking operation for words, say, *Spell*(*word*), by wrapping PWP (a fictitious name for a word processor) spell checker API.

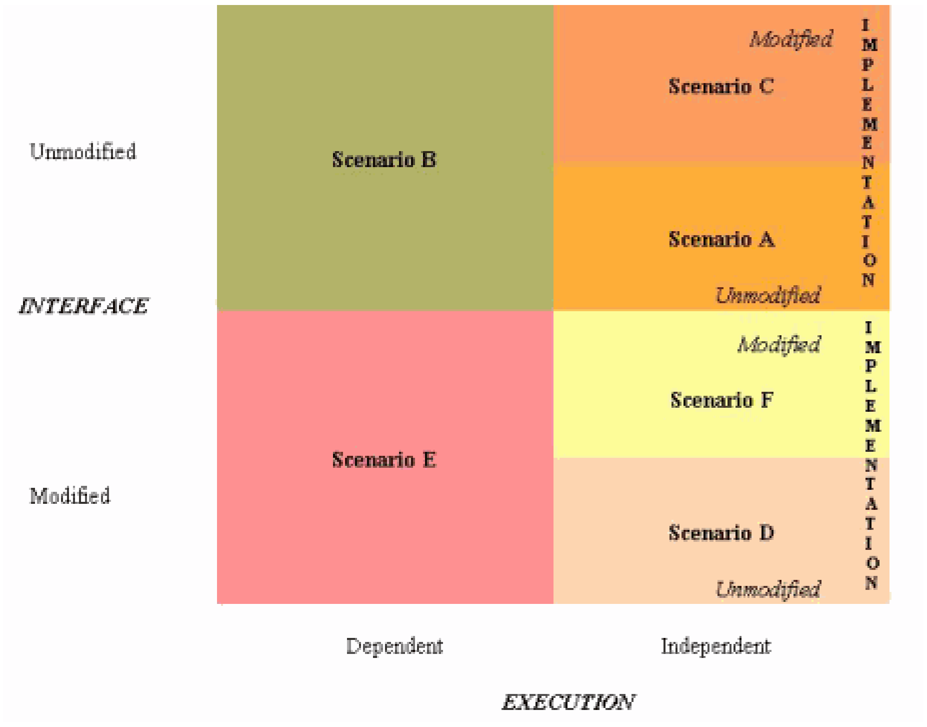

Figure 1. Exemplifying Freedom and Openness in F/O-Services

Scenario A

Description:

- The simplest method enabling free usage and distribution of a service.
- May require simple attribution to the parent service.

Example:

Let S_B be an independent service, providing the same *Spell*(*word*), created by replicating the source code of implementation and interface of S_A. Albeit S_A and S_B are performing same operations, S_A and S_B are two different services, executed separately (possibly could be a part of the information system of a different organization).

Scenario B

Description:

- A common scenario in SOC.
- Adds value to a service by distributing the service, not requiring to implement the service again.

Example:

Let S_B be a service providing a spell checking operation *Spell*(*word*) for words, using (copying) the interface *Spell*(*word*) of S_A. S_B is designed in such a way that *Spell*(*word*) of S_B directly invokes the operation of S_A, executing on the host of S_A itself.

From a service consumer's perspective, S_A and S_B are providing exactly the same *Spell*(*word*) interface. Thus, they can be interchangeable in an application on the consumer side. However, implementations of S_A and S_B are not distinguishable. Theoretically, there will not be any noticeable differences in performances of both services, apart from possible network latency between S_A and S_B or different hardware performances for the hosts of S_A and S_B.

Scenario C

Description:

- As an entirely new service from a F/O-Service keeping its interface unchanged and modifying the implementation.

Example:

Let S_B be an another independent service, providing the same *Spell*(*word*), created by replicating the interface of S_A. However, S_B provides the operation *Spell*(*word*) by wrapping QWQ spell check API (QWQ is a fictitious name for another word processor.). Albeit S_A and S_B are performing the same operations, S_A and S_B are two different services, executed separately.

From a service consumer perspective, there could be differences in the performance of S_A and S_B, same as in scenario B but also depending on the performances of the different word processors in use.

Scenario D

Description:

- Creates a new independent service only by modifying its interface (by not allowing or not interesting to change the implementation).

Example:

Let S_B be another independent service, created by replicating the source code of service implementation and modifying the interface of S_A, to provide a spell checking operation in Italian language, say *Ortografia(parole)*. In this case, S_B translates the interface of S_A and results in the Italian version of S_A as an independent service.

Scenario E

Description:

- Allows another services to modify the interface and to invoke operations in a dependent way.

Example:

Consider a service S_B with an interface providing *Spell*(*sentence*). For every execution of a word in a sentence, *Spell*(*word*) of S_A is repeatedly invoked. Thus, for spell checking of a given prose *Spell*(*sentence*), S_B invocates *Spell*(*word*) of S_A repetitively for each words. Here, S_A as a F/O-Service allows S_B for certain operations to execute directly on the host of S_A.

Scenario F

Description:

- The most permissive case of freedom/openness offered by F/O-Services (as a result of derivation).

Example:

From the given F/O-Service S_A, we create a new service S_B, by modifying interface and implementation of S_A. The interface of S_B provides *Spell*(*sentence*) which composes *parser*() to split the given sentence and *Spell*(*word*) to spell check a word reusing the operation of S_A. Now, S_A and S_B are two different services, executed independently. *Spell*(*sentence*) of S_B is derived from *Spell*(*word*) of S_A and is value added by having an own additional functionality *parser*().

Extending F/O-Services by Dependency

Services provide universal interoperability, manifested by the web-like network of services created by the composition of services into more complex services. However, the opaque nature of services often hides the details of operations from the service consumer. The consumer could neither see anything beyond the interface nor understand about the services being composed in a composite service. While in the tradition of software encapsulation this is considered desirable, we claim that it may prove too restrictive when applied to services as we will elaborate below.

We define dependency between services as the description of the interactions of a service with other services. Interactions do not have a direction per se, but a dependency does. A dependency link is directed from the service consumer to the service provider and expresses the dependency of the consumer on the provider. Consider a service S_A, which composes the services S_B, S_C, and S_D (see Figure 2). Further, these services compose S_E, S_F, S_G, and S_H. Given the service S_A, we could not understand what the services are being composed in S_A.

If we make the dependencies of services open, we could achieve a service, whose service internals are completely exposed to the consumer. In Figure 2, circles represent services and arrows represent their dependencies. From the given dependency graph, we could recognize the complete hierarchy of composed services. This approach is quite similar to white box description of components [4].

Opening dependencies implies only the provision of a list of composed services, and differs from fully exposing the application logic used to compose them. There are at least two notions of openness in the context of dependencies.

1. The service declares which services it uses, but this does not imply a right for the consumers to invoke those services directly, if the composition as an assembly is itself an artifact that the service provider wishes to protect with restrictions and

2. The service allows others to reuse the relationships with other services it has. Restrictions imposed by the component services apply, of course.

However, these notions are not all-encompassing. For instance, there may be intellectual rights attached to the selection of services during composition. These could, for instance, require the composer using a F/O-Service to allow similar freedom to users of the composite service.

Also, expressing the openness of dependencies depicts licensing compliance issues of a service with other services. In the dependency graph of our example, consider that licenses of S_F and S_H impose restrictions on S_C and S_D respectively. The license of S_C should comply to S_F and the license of S_D should comply to S_H. The licensing clauses of S_A should comply to the licensing clauses of S_C and S_D. Thus, S_F and S_H are the minimum licensing sets for composing the licensing accountability of S_A. However, we have assumed the licenses of S_B, S_E, and S_G will not impose any restrictions on the licensing of composition of services in this example (indicated by dashed lines in Figure 2). Thus, the licensing accountability graph is the sub-graph of a dependency graph showing the services that require licensing compliances among themselves (indicated by straight lines).

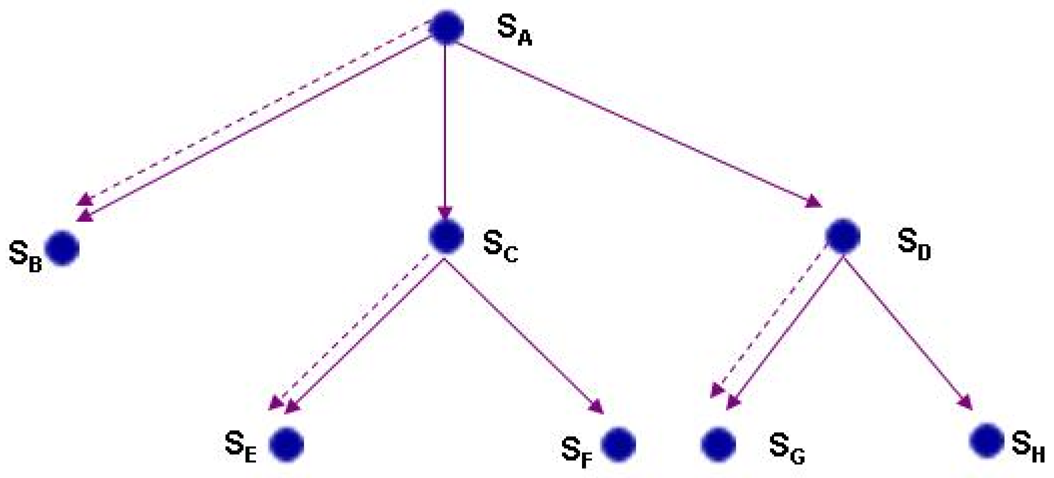

Figure 2. Service Dependability and Licensing Accountability

Business Models for F/O-Services

Free services inspired by Free Software licenses could make value addition by composition, resulting composed services as `free'. Thus, free services (with free licenses) could create a chain effect on composition of services to be free, even if one of the composing service may be not `free'. Making services (monetarily) free could be highly beneficial for government sectors, education, and non-profitable organizations to explore and enjoy the benefits of services.

Making F/O-Services may raise an emergent question of how a service provider could profit by providing services. For this, we propose the following set of business models for the sustainability of F/O-Services. There exists a wide range of business models for software [5] as well as for services [6].

Following are some of the possible business models for F/O-Services.

Accessorizing

A service may be free in the sense of no cost. However, it could motivate the consumer to purchase something. For example, a service providing map and route information freely may require the service user to install a Global Positioning System device in his/her vehicle. Also, F/O-Services can make revenue from training, consulting, and custom development.

Sell It, Free It

Like traditional commercial software, services can begin their life cycle as closed and then later, can be converted as F/O-Services when appropriate.

Brand Licensing

A F/O-Service provider can charge other service providers/ aggregators/ consumers for the right to use its brand names and trademarks in creating derivative services. This is one of the common business models in practice of the FOSS community [7].

Dual Licensing

Dual licensing is a business model for FOSS exploitation based on the idea of simultaneous use of both FOSS and proprietary licenses. Following the dual licensing strategy, a service can be licensed under a F/O-Service license as well as a proprietary license. In addition to delivering complementary revenue streams, the dual licensing strategy captures a large user base.

Intermediary and Shared Infrastructure Models

The intermediary and shared infrastructure models [8] can also be adapted to F/O-Services. One type of intermediary is a service aggregator. It adds value by composing other services so that a new functionality arises that was not available before. An example is a context-aware service that combines location sensing with location-specific information services. Intermediaries may also add value through the pre-selection of component services and managing their quality. A shared infrastructure service is an open service jointly developed by service users or providers for their common usage. In this case, it is more economic to share the development costs rather than developing the capabilities provided by the services individually.

Differentiated Use Model

A service could be used for consumption (by the end user) or for value addition (composition or derivation by another service). A service provider could come across a model of offering a base service for free to end users, but charging for value added capabilities or vice versa. However, technologically, it is difficult to differentiate these types of uses. The use of services could be charged by subscription or pay-per-use models [9].

Service Hosting

Service hosting is another business strategy for F/O-Services. A F/O-Service provider could host the services defined by others, thus making a viable business opportunity. A service host provides the capacity for executing a F/O-Service. There are different options for the service host to be remunerated: charging users a fee, charging providers of the service a fee, or by adapting the quality of service in exchange for remuneration. For instance, a free hosting service could be offered as time-limited, or only offer a certain number of executions per day. There is also an opportunity to generate revenue through direct (embedded advertising) or indirect (resale of demographics) marketing campaigns.

Selling Infowares

Selling infowares is an exclusive business model for F/O-Services. As F/O-Services allow derivation of services freely, the restrictions over usage or value addition of the services might seem illogical at the first sight. But, of course, there is data associated with the service. Services intend to use the data and could be referred as data driven applications. Thus, services are infoware [10], more than software. We view services as a combination of software and data. the use of services signifies the access of software as well as the data. However, copying of a service refers to copying of the associated software only. A F/O-Service placing restrictions on usage/value addition owning a unique source of data would be a sustainable source for financial income.

Summary

In this article, we have proposed the concept of Free/Open Services, a distinctive advance over the concept of service-oriented computing, inspired by Free/Open Source Software approach. F/O-Services allow the access to the source code of interface and implementation of services, making freely distributable composite services and derivative services. Though FOSS approach does not discriminate in the uses of software, the dynamic binding and execution of services could enforce certain restrictions for the execution/usage of F/O-Services. Thus, F/O-Services can be restricted by a service provider. Furthermore, F/O-Services enable the creation of fully transparent composite services and allow people and other services to access them. We have proposed a set of business models for F/O-Services, adapted from the business strategies of services and FOSS.

The wedding of services with FOSS would be beneficial for both communities, spreading services `free'ly. Some informal and unstructured discussions about the concepts of free/open services are budding in several web logs [11, 12, 13]. However, there is no unambiguous conceptualization of F/O-Services. In early 2006, we have proposed and illustrated free/open source perspectives of licensing for services [14] and extended the concept further by [15] in early 2007. And, our perspective on F/O-Services are not all futuristic! It is the need to enrich the service-oriented community by itself and to enable services to proliferate without limits. We have seen the Honest Public License (Version 1 as on August 2006) having a clause to handle services. As a milestone in free software licensing, in November 2007, Free Software Foundation has released GNU Affero General Public License. According to AGPL, the modifications to software that power publicly available services should be contributed to the free software community.

However, till now, the concept of Free/Open Services is still in its nascent stage. The service-oriented community is afraid as there is no tradition of freeing/opening services and the fear of going against encapsulation. Let us all join together for proclamation and promotion of this F/O-Services community and to make a new brave world of F/O-Services.

REFERENCES

1. J. Feller, B. Fitzgerald, S. Hissam, and K. Lakhani, Perspectives on Free and Open Source Software, MIT Press, 2007.

2. T. Erl, Service-Oriented Architecture: Concepts, Technology & Design, Prentice Hall/Pearson PTR, 2005.

3. F. Casati, and M.C. Shan, Dynamic and Adaptive Composition of E-Services, Information Systems, 6(3), 2001.

4. C. Szyperski, Component Software: Beyond Object Oriented Programming, ACM Press, New York, 1998.

5. C. Shapiro, and H. Varian, Information Rules: A Strategic Guide to the Network Economy, Harvard Business School Press, 1999.

6. J. Hagel, Out of the Box: Strategies for Achieving Profits Today and Growth Tomorrow Through Web Services, Harvard Business School Press, 2002.

7. E. Raymond, The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary, O’Reilly Press, 2000.

8. P. Weill, and M. Vitale, Place to Space: Migrating to E-business Models, Harvard Business School Press, 2001.

9. D. Ferrante, Software Licensing Models: What’s Out There?, IEEE IT Professional, 8(6), 2006.

10. T. O'Reilly, What Is Web 2.0: Design Patterns and Business Models for the Next Generation of Software, Communications & Strategies, 65(1), 17-37, 2007.

11. Web Services and Open Source at OSCON. http://developers.slashdot.org/article.pl?sid=06/07/26/1537213 (Posted on July 26, 2006)

12. log.ometer.com: Log for July, 2006. http://log.ometer.com/2006-07.html (Posted on July 29, 2006)

13. Evaluating a Free/Open Service Definition (rough draft). http://tieguy.org/blog/2007/07/22/evaluating-a-freeopen-service-definition-rough-draft/ (Posted on July 22, 2007)

14. V. D'Andrea, and G. R. Gangadharan, Licensing Services: An "Open" Perspective, In: Proceedings of the International Conference on Open Source Systems, 143-154, Springer, 2006.

15. G.R. Gangadharan, V. D’Andrea, and M. Weiss, Free/Open Services: Conceptualization, Classification, and Commercialization, In: Proceedings of the International Conference on Open Source Systems, 253-258, Springer, 2007.

ABSTRACT

Service-oriented computing (SOC) represents the convergence of technology with an understanding of cross-organizational business processes. In general service consumers have no access to the implementation details of a service, including whether or not a service uses other services, and what are these other services. An approach similar to Free/Open Source Software (FOSS) that opens the accessibility of the source code of services and use/distribution of services would significantly enhance the understandability of the service composition process (including data and control flow) and allow the creation of derivative services. A novel concept of Free/Open services (F/O-Services) adopts and adapts the principles of FOSS approach to SOC, to enhance the widespread use of services.

KEY WORDS

Free/Open Services (F/O-Services)
Service Oriented Computing
Free/Open Source Software
Service Dependency
Service Licensing

AUTHOR BIOS

G. R. Gangadharan is a researcher at the Politecnico di Milano, Milan, Italy. His research interests are mainly located on the interface between technological and business perspectives. His research interests include green information systems, service oriented computing, and free/open source software. He has received Ph.D. degree in Information and Communication Technology from the University of Trento, Trento, Italy and European University Association. He is a member of IEEE. Contact him at geeyaar@gmail.com.

Vincenzo D'Andrea is an associate professor at the University of Trento, Trento, Italy. His research interests include service-oriented computing, free and open source licensing, and socio-technical systems. He received his PhD in Information Technology from the University of Parma. He is a member of the IEEE Computer Society and the ACM. Contact him at dandrea@disi.unitn.it.

Michael Weiss holds a faculty appointment in the Department of Systems and Computer Engineering at Carleton University, Ottawa, Canada. His research interests include open source ecosystems, service-oriented architectures, mashups/Web 2.0, business process modeling, product architecture and design, and pattern languages. Contact him at weiss@sce.carleton.ca.