WEON: Towards a softWare Ecosystem ONtology

Claudio Gutierrez
Department of Computer Science (DCC)
University of Chile
cgutierr@dcc.uchile.cl

Romain Robbes
PLEIAD Laboratory
Department of Computer Science (DCC)
University of Chile
rrobbes@dcc.uchile.cl

ABSTRACT

The natural distributed character of software ecosystems calls for a shared conceptualization and language to describe their architecture and their evolution. In this regards, ontologies play a central role. In this paper: we argue in favor of such an approach by showing that there is succesful experience applying ontologies to the fields of software engineering and software architecture; show the issues arising in ecosystem evolution and the needs for higher levels of formalization of the description of such processes; finally, we present a roadmap to develop an ontology for this area.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement; D.2.11 [**Software Engineering**]: Software Architectures; I.2.4 [**Artificial Intelligence**]: Semantic Networks

General Terms

Standardization

Keywords

Software Ecosystems, Software Architecture, Software Evolution, Ontologies

1. INTRODUCTION

Software ecosystems are defined by Lungu [25] as “a collection of software systems, which are developed and co-evolve in the same environment”. The environment can be organizational (a company), social (an open-source community), or technical (the Ruby ecosystem).

The natural scaling implicit in the notion of software ecosystems calls for a shared conceptualization to describe the architectures of such systems, which naturally are composed of smaller and distributed systems with their own architectural descriptions and languages).

As shown later in this paper, the literature and practice shows there are scattered efforts in developing ontologies for software specification, description, evolution and architecture. Thus, there is experience in the field, and second, there is need for a common understanding between them in order to transform these scattered efforts in a concerted one.

Hence, following the experiences of the W3C (World Wide Web Consortium), we argue that developing a Working Group to delineate a shared conceptualization of the basics of WEON, a softWare Ecosystems ONtology [1], would be a mean to transform these scattered efforts into a stable and standardized ontology.

In this position paper, we:

1. present the evidence that there is enough background and experiences in the community to address such collaborative effort;
2. present a brief and preliminary review of the main works on this topic to highlight the work done;
3. detail some of the issues affecting software ecosystems and their evolution;
4. and provide a roadmap that enumerates the main challenges behind such an effort.

2. ONTOLOGIES IN SOFTWARE ENGINEERING, ARCHITECTURE, AND EVOLUTION

The interplay between the fields of software engineering, software architecture and ontologies has been receiving increasing attention lately. Indeed, the applications of ontological conceptual frameworks and techniques to the area of software engineering and software architecture is not new. In this section, we will briefly review the most important developments of these mutual influences. (For background on ontologies and its engineering see the books of Goméz-Pérez *et al.* [10], and Staab and Studer [33]).

2.1 Ontologies in Software Engineering

There is good experience developing and using ontologies in the area of software engineering. The W3C established in 2001 a working group to work on the benefits of applying knowledge representation languages common to the Semantic Web, such as RDF and OWL, in Systems and Software Engineering practices. Their goal was to motivate software engineering practitioners, outline the benefits of these techniques, and to encourage collaboration between the Semantic Web and the Systems and Software Engineering communities [35].

WEA ’13, August 19, 2013, Saint Petersburg, Russia
Copyright 13 ACM 978-1-4503-2314-7/13/08 ...$15.00.

[1] so as to avoid name clashes with SEON, the Software Evolution ONtology pyramid by Würch *et al.* [38]

Wouters *et al.* [37] proposed a semi-formal approach based on the notion of ontolgogies to cope with the problem of managing large sets of use cases. Their ontology is based on three categories of information: labels, concepts and relations. It is not difficult to see that it could be specified in standard ontological languages such as RDFS and OWL.

Realizing that metamodels and ontologies have been developed in parallel and in isolation, Henderson-Sellers [14] investigates forms to establish bridges between these two worlds. Happel and Seedorf [13] provide a good overview of the applications of ontologies in the area of software engineering. They mention the relevance of ontologies in the following areas: (1) Analysis and Design (Requirements engineering Component reuse). (2) Implementation (Integration with Software modelling languages; ontologies as domain object model, Coding support, Code documentation). (3) Deployment and Runtime (Semantic Middleware, Bussiness rules, Semantic Web Services) and (4) Maintenance (Project Support, Updating, Testing).

Würsch *et al.* [39] showed that ontologies provide a very good support for day-to-day tasks performed by developers. In particular, they showed that most queries that developers need on a daily basis, as identified by Sillito *et al.* [32], were implementable in a framework allowing natural language queries, using Semantic Web technologies and ontologies as their foundations.

Finally, let us record that Calero *et al.* [2] compile research done towards sharing knowledge of the problem domain and using a common terminology among all stakeholders.

2.2 Ontologies in Software Architecture

Additionally to the developments mentioned above, there are several experiences developing and using ontologies in the area of software architecture. One of the most explicit effort is that of SOA Ontology of the Open Group [11], whose goal is to develop common understanding of Service-Oriented Architecture in order to aid understanding, and potentially be a basis a for model-driven implementation. It is directed to Business people, Architects and System and software designers. There are, though, several other developments that implicitly use the notions behind the one of "ontology", that is, the standardization of language, of relationships between concepts, and logical formalization of informal knowledge. With respect to software architecture, there are four important directions for our purposes.

2.2.1 Documentation of Software Architectures

Jansen *et al.* [18] claim that current documentation approaches have severe shortcomings in capturing the knowledge of large and complex systems. The main challenges are: understandability; locating relevant architectural knowledge; traceability; change impact analysis; design maturity assesment; trust. They propose to enchance traditional software architecture using formal architecture knowledge. We also note that all these problems are amplified when one deals with software ecosystems, instead of single systems.

2.2.2 Representing design dependencies

In most design activities, the analysis of the different alternatives is not stored. Thus, after a system is developed, the design rationale is not anymore available to further developers. This is more relevant when the system is undergoing future changes not known at design time. As reported by Lubars, several approaches have been proposed to overcome this problem [24], and one thing that is transversal to all is the need to a standard common language which is able to capture the network of design dependencies.

2.2.3 Architectural knowledege

Different organizations maintain and store in different forms the information of their software architectures. The book by Boer *et al.* [5] defines a "core model", i.e. a minimal semantic networks (a basic ontology) based on four perspectives: sharing, compliance, discovery and traceability, to attempt to cover the architectural knowledge domain.

2.2.4 Ontologies and Software Architecture

Based on the previous three directions, work has been performed towards using ontologies to fulfill this purposes:

Kruchten [22] emphasizes the need to leverage architectural decision design to become a first class citizen in the process of developing complex software systems. He proposes a classification model for the organization, attributes and relationships (internal and with external artifacts) of design decisions. From here he develops a taxonomy (a simple ontology) from which networks of such interrelated decisions could help the reasoning about them.

Garzas and Piattini [9] address the problem of microarchitectural design knowledge. They present an ontology that organizes and formalizes this knowledge, which comprises in separately declarative and operative knowledge, and encompasses rules, patterns, and refactoring.

Based on the effective reusability of software architecture knowledge, Erfanian *et al.* [8] propose two ontologies to address the problem of evaluation of some types of architectures. The use of ontologies allows to express the vocabularies and the semantics of the domain in a formal and explicit way, thus allowing reuse and semi-automation of such knowledge.

2.3 Ontology Evolution

One of the main characteristic of the field of Software Engineering is that of Software Evolution, as encompassed by Lehman's laws of Software Evolution [23]. As we will discuss below, Software ecosystems are no exception to the rule. In this context, the research about ontology evolution becomes crucial. The dynamics and changes of the software gives rise to changes in the application requirements. In turns, if they are formally specified, this amounts to changes in the underlying ontologies.

Stojanovic [34] surveys methods and tools of this process. She defines the requirements for ontology evolution and presents a process model that fulfils them.

In this area an important issue to highlight are the notions of versioning and evolution. As Noy and Klein [26] point out, ontology evolution is not the same a schema evolution (e.g. in databases). They mention the difference between usage paradigms, the presence of explicit semantics and different knowledge models. These issues play an important role when considering the evolution of software.

Finally, the SEON pyramid of ontologies defined by Würch *et al.* [38] shows that ontologies have been applied in the context of software evolution, including concepts such as changes to a software system over time, defects affecting the system, and the history of the software system's artefacts. A subset of SEON was used in the previously mentioned work of Würsch *et al.* [39].

3. ISSUES IN ECOSYSTEM EVOLUTION

The problem of ecosystem evolution has attracted intense research interest in recent years, suggesting this is a practical problem. The main issues we detail below are the ones of the independent evolution of software systems a particular depends on, outside of the control of the developers of said system, and issues related to the porting of changes from one branch to another in case of parallel evolution of branches, which is made more complex by the large amount of duplication. In addition to this, all the problems encountered in the evolution of a software system and its architecture are compounded by the size of the software ecosystems, and logistical issues due to the potential distribution of the data accross multiple repositories.

3.1 Evolution of Frameworks and Librairies

When a client depends on a library or a framework, any change in the library or framework may break the client's code, who may need to be updated in reaction to the change. There have been several empirical studies of the phenomenon: Dig and Johnson studied 5 open-source frameworks, and found that 80% of API-breaking changes were refactorings [6]. Kapur *et al.* did a similar study on 3 additional systems, with a finer granularity between the successive versions of the source code [20]: they found that the average number of API changes between versions was between 13 and 28 for all the systems. Finally, Robbes *et al.* performed a large-scale study of how developers reacted to deprecation in an software ecosystem featuring several thousands of projects [29]; among other findings, they found that some of the deprecations had an impact on a large amount of developers, that clients took time to notice their code base was outdated, and that not all clients updated their code.

As a consequence, there has been several approaches to react to changes in evolving frameworks and libraries. Henkel and Diwan proposed to record refactorings performed in the library code to replay them in the client code [15], a functionality that is now present in Eclipse. Dig *et al.* proposed a refactoring-aware versioning system to achieve similar goals [7]. Dagenais and Robillard analyze how the framework adapts to its changes in order to provide recommendations to API changes [4], while Shaefer *et al.* analyze adaptions from previous clients to issue similar recommendations [30]. Rather than issuing recommendations on how to update the code base, Holmes and Walker recommend changes event that may need a reaction from the developer [17]. Finally, Cossette and Walker compared several change recommendation techniques on a set of evolving APIs, and found that each technique gave a correct recommendation in around 20% of the cases, and that the recommendations given by the techniques were complementary [3].

3.2 Porting of Changes and Code Duplication

Similar issues happen when a system or ecosystem is forked from another: Some changes done in one system (such as important bug fixes or new functionality) need to be ported to the other branches. The study by Ray and Kim of FreeBSD, NetBSD, and OpenBSD shows that 11 to 16% of the changes happening in one of the systems are changes ported from another branch, and that 13 to 33% of the active developers were involved in porting these changes [28]. This shows that code duplication is a significant issue in software ecosystems. In the same vein, the study of Schwartz *et al.* showed that the Squeaksource ecosystem had 15% of its methods being clones of other methods [31]. Also heterogeneity in languages and infrastructures, and raw size of the ecosystem are issues.

3.3 Size issues

Software Ecosystems, being composed of individual systems, are characterized by their massive size. For instance the dataset of the Ruby on Rails ecosystem provided by Wagstrom *et al.* weights more than 8 gigabytes [36]. The Squeaksource ecosystem analyzed in [29] and [31] weights 24 gigabytes. Other software ecosystems are larger, such as the Maven ecosystem of Java librairies dataset presented by Raemaekers *et al.*, that contains nearly 150,000 distinct applications and libraries, and is measured in hundreds of gigabytes [27].

To address these issues requires special care as performance becomes a concern. For instance, standard code clone detection tools do not scale well to large sizes of data, hence specialized techniques were designed to ensure such a task scales to large datasets, such as the work of Koschke [21]. We note that issues of scale are also addressed in ontology research, an example of this kind of work is the one by Hogan *et al.* [16] on making OWL scale to large amounts of data.

3.4 Logistical issues

Another issue is due to logistics. It is common that a software ecosystem is spread out in several different locations, making access to the data non-trivial. For instance, if one were to study the ecosystem of all open-source programs written in Java, one would need to access a very large number of different repositories in order to arrive to a reasonable approximation of said ecosystem. The studies performed so far have been usually limited to more centralized ecosystems, for which the data is confined to a small set of locations.

4. ROADMAP

Our literature reviews shows that there has been continuous, historical interest in using software ontologies in software engineering. We argue that the specific challenges with regard to software ecosystems, coupled with the challenges of software evolution magnified by the size of ecosystems, makes this natural fit of ontologies with software ecosystems all the more important. In addition, the distributed nature of software ecosystems makes this fit even more natural, as it closely mirrors the distribution of the data in the Semantic Web.

The development of an ontology to describe the architecture of a software ecosystem, and the description of the changes to such architectures, would provide important support for the comprehension of the architectures of large-scale software ecosystems, would increase awareness of the changes that are happening in the ecosystem, and would ensure the timely and adequate reaction to changes in systems a particular system is depending on. This is in line with challenges in software architecture of understandability, traceability, and change impact analysis, for which an ontology has already been proposed [18].

Another critical aspects we wish to underline is the need for concertation and cooperation. The multiplicity of works that were performed calls for a joint effort of standardization, instead of keeping the status quo where individual research groups lead their individual efforts. Traditionally, the way to do so has been in the Semantic Web community to define a Working Group.

We envision the following roadmap to arrive to such a vision. It is important to recall that this is not a linear, but an interative process.

1. *Survey and State of the Art.*

   Perform a systematic survey and evaluation of the work done in ontologies for software engineering and software architecture description languages. We presented in this paper a preliminary state of the art of these aspects that shows that there is enough ground from where to start such a task.

   In particular, regarding software ecosystem evolution abstractions, compare and contrast them with the extensive research done in ontology evolution in order to identify opportunities for reuse and adaptation of existing ideas.

2. *Scope and Goals.*

   One of the most relevant aspects when building an ontology is to delimit its scope and depth (size) and the level of interoperability with similar or related projects. On the basis of the experiences and use cases obtained in (1), it is important to define stages of coverage and development, hopefully starting with a simple core which could serve as basis to extensions in different directions. In this stage it is important to make committments to the degree of formalization and sophistication of tools that are expected to foster its use.

3. *Development.*

   We see three well delimited stages in the process of development of the ontology:

   a) Extract the relevant concepts that belong to an ontology of software ecosystems. Important in this regard are the advances in architectural description languages, on which work will need to be done to translate the concepts from the level of a single software system to the level of software ecosystems.

   b) Retrofit these notions to existing ecosystems in order to ensure the ontology correctly describes reality. Doing so requires the construction of a set of "benchmark" systems and use cases. Discuss and validate them with communities using different languages and practices.

   c) Formalize the evolution of ecosystems and the concepts therein, arriving to a first version of WEON, the softWare Ecosystem ONtology. Choose a formalism which balances user experience and current standards (RDFS, OWL, etc.) to describe WEON.

4. *Deployment/Instantiation.*

   Revise and formalize the ontology. Develop semi-automated techniques in order to apply the ontology to the large quantities of data characterizing software ecosystems.

   Develop tools to support developers when they face the issues described above. The work of Würsch *et al.* at the level of individual systems shows that this goal is achievable.

   Scale these tools to large specifications. This final step of our tentative roadmap could also benefit from the active research in scaling semantic web technologies to large-scale data.

5. CONCLUSION

We showed in this paper that the time is ripe to address the development of an ontology for software ecosystems: software ecosystems face challenges for which ontologies seem to be a natural fit. The large amount of research performed in developing and using ontologies in software engineering gives us further confidence that such a fit is possible and will give fruitful results.

We sketched a roadmap to achieve this goal of defining an ontology for software ecosystems, and to develop additional supporting tools. However, the fragmented nature of the work we surveyed made us aware that this roadmap needs cooperation beyond the boundaries of a single research group. Such an effort is not possible without the composition of a working group that will perform these duties.

Finally, one should note that there are several definitions of ecosystems besides Lungu's. The work by Hanssen and Dybå [12] references three additional definitions:

- "A networked community of organizations or actors, which base their relations to each other on a common interest in the development and use of a central software technology" [12].
- "A set of businesses functioning as a unit and interacting with a shared market for software and services, together with the relationships among them" [19].
- "The set of software solutions that enable, support and automate the activities and transactions by the actors in the associated social or business ecosystem and the organizations that provide these solutions" [1].

If these definitions focus more on the organization and business aspects than on the software artifacts and their architecture, they certainly should be considered in the development of the ontology. This multiciplicity of definitions further stresses out the need for a working group as a discussion medium to arrive to a common understanding.

6. REFERENCES

[1] J. Bosch. From software product lines to software ecosystems. In *SPLC 2009: Proceedings of the 13th International Conference on Software Product Lines*, pages 111–119, 2009.

[2] C. Calero, F. Ruiz, and M. Piattini. *Ontologies in Software Engineering and Software Technology*. Springer-Verlag New York, Inc., 2006.

[3] B. Cossette and R. J. Walker. Seeking the ground truth: a retroactive study on the evolution and migration of software libraries. In *SIGSOFT FSE 2012: Proceedings of the 20th ACM SIGSOFT Symposium on the Foundations of Software Engineering*, page 55, 2012.

[4] B. Dagenais and M. P. Robillard. Recommending adaptive changes for framework evolution. *ACM Trans. Softw. Eng. Methodol.*, 20(4):19, 2011.

[5] R. C. de Boer, R. Farenhorst, P. Lago, H. van Vliet, V. Clerc, and A. Jansen. Architectural knowledge: Getting to the core. In *QoSA*, pages 197–214, 2007.

[6] D. Dig and R. E. Johnson. How do apis evolve? a story of refactoring. *Journal of Software Maintenance*, 18(2):83–107, 2006.

[7] D. Dig, K. Manzoor, R. E. Johnson, and T. N. Nguyen. Refactoring-aware configuration management for object-oriented programs. In *ICSE 2007: Proceedings of the 29th International Conference on Software Engineering*, pages 427–436, 2007.

[8] A. Erfanian and F. Shams Aliee. An ontology-driven software architecture evaluation method. In *Proceedings of the 3rd international workshop on Sharing and reusing architectural knowledge*, SHARK '08, pages 79–86, New York, NY, USA, 2008. ACM.

[9] J. Garzas and M. Piattini. An ontology for microarchitectural design knowledge. *IEEE Softw.*, 22(2):28–33, Mar. 2005.

[10] A. Gómez-Pérez, M. Fernández-López, and O. Corcho. *Ontological Engineering: with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web. (Advanced Information and Knowledge Processing).* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[11] T. O. Group. Service-Oriented Architecture Ontology—Technical Standard, 2010.

[12] G. Hanssen and T. Dybå. Theoretical foundations of software ecosystems. In *IWSECO 2012: Proceedings of the 4th Software Ecosystem Workshop*, 2012.

[13] H.-J. Happel and S. Seedorf. Applications of ontologies in software engineering. In *International Workshop on Semantic Web Enabled Software Engineering (SWESE'06)*, Athens, USA, November 2006.

[14] B. Henderson-Sellers. Bridging metamodels and ontologies in software engineering. *Journal of Systems and Software*, 84(2):301–313, 2011.

[15] J. Henkel and A. Diwan. Catchup!: capturing and replaying refactorings to support api evolution. In *ICSE 2005: Proceedings of the 27th International Conference on Software Engineering*, pages 274–283, 2005.

[16] A. Hogan, A. Harth, and A. Polleres. Scalable authoritative owl reasoning for the web. *Int. J. Semantic Web Inf. Syst.*, 5(2):49–90, 2009.

[17] R. Holmes and R. J. Walker. Customized awareness: recommending relevant external change events. In *ICSE 2010: Proceedings of the 32nd International Conference on Software Engineering*, pages 465–474, 2010.

[18] A. Jansen, P. Avgeriou, and J. S. van der Ven. Enriching software architecture documentation. *J. Syst. Softw.*, 82(8):1232–1248, Aug. 2009.

[19] S. Jansen, A. Finkelstein, and S. Brinkkemper. A sense of community: A research agenda for software ecosystems. In *ICSE 2009: Proceedings of the 31st International Conference on Software Engineering*, pages 187–190, 2009.

[20] P. Kapur, B. Cossette, and R. J. Walker. Refactoring references for library migration. In *OOPSLA 2010: Proceedings of the 25th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pages 726–738, 2010.

[21] R. Koschke. Large-scale inter-system clone detection using suffix trees. In *CSMR 2012: Proceedings of the 16th European Conference on Software Maintenance and Reengineering*, pages 309–318, 2012.

[22] P. Kruchten. An ontology of architectural design decisions in software intensive systems. In *2nd Groningen Workshop on Software Variability*, pages 54–61, Dec 2004.

[23] M. M. Lehman and F. N. Parr. Program evolution and its impact on software engineering. In *ICSE 1976: Proceedings of the 2nd International Conference on Software Engineering*, pages 350–357, 1976.

[24] M. D. Lubars. Representing design dependencies in an issue-based style. *IEEE Softw.*, 8(4):81–89, July 1991.

[25] M. Lungu. *Reverse Engineering Software Ecosystems*. PhD thesis, University of Lugano, 2009.

[26] N. F. Noy and M. Klein. Ontology evolution: Not the same as schema evolution. *Knowl. Inf. Syst.*, 6(4):428–440, July 2004.

[27] S. Raemaekers, A. van Deursen, and J. Visser. The maven repository dataset of metrics, changes, and dependencies. In *MSR 2013: Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 221–224, 2013.

[28] B. Ray and M. Kim. A case study of cross-system porting in forked projects. In *SIGSOFT FSE 2012: Proceedings of the 20th ACM SIGSOFT Symposium on the Foundations of Software Engineering*, page 53, 2012.

[29] R. Robbes, M. Lungu, and D. Röthlisberger. How do developers react to api deprecation?: the case of a smalltalk ecosystem. In *SIGSOFT FSE 2012: Proceedings of the 20th ACM SIGSOFT Symposium on the Foundations of Software Engineering*, page 56, 2012.

[30] T. Schäfer, J. Jonas, and M. Mezini. Mining framework usage changes from instantiation code. In *ICSE 2008: Proceedings of the 30th International Conference on Software Engineering*, pages 471–480, 2008.

[31] N. Schwarz, M. Lungu, and R. Robbes. On how often code is cloned across repositories. In *ICSE 2012: Proceedings of the 34th International Conference on Software Engineering*, pages 1289–1292, 2012.

[32] J. Sillito, G. C. Murphy, and K. D. Volder. Asking and answering questions during a programming change task. *IEEE Trans. Software Eng.*, 34(4):434–451, 2008.

[33] S. Staab and R. Studer. *Handbook on Ontologies*. Springer Publishing Company, Incorporated, 2nd edition, 2009.

[34] L. Stojanovic. *Methods and tools for ontology evolution.* PhD thesis, Karlsruhe Institute of Technology, 2004. http://d-nb.info/1001606787.

[35] P. Tetlow, J. Pan, D. Oberle, U. Wallace E., and E. M., Kendall. Ontology driven architectures and potential uses of the semantic web in software engineering. In *W3C Working Draft, 2006*, 2006.

[36] P. Wagstrom, C. Jergensen, and A. Sarma. A network of rails: a graph dataset of ruby on rails and associated projects. In *MSR 2013: Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 229–232, 2013.

[37] B. Wouters, D. De Ridder, and E. Van Paesschen. The use of ontologies as a backbone for use case management. In *Workshop Objects and Classifications, A Natural Convergence, ECOOP 2000*, 2000.

[38] M. Würsch, G. Ghezzi, M. Hert, G. Reif, and H. C. Gall. Seon: a pyramid of ontologies for software evolution and its applications. *Computing*, 94(11):857–885, 2012.

[39] M. Würsch, G. Ghezzi, G. Reif, and H. Gall. Supporting developers with natural language queries. In *ICSE 2010: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, pages 165–174, 2010.