SOFAS: Software Analysis Services

Ghezzi, G

DOI: https://doi.org/10.1145/1810295.1810398

SOFAS - SOFtware Analysis Services

[Extended Abstract]

Giacomo Ghezzi
University of Zurich, Institute for Informatics
Binzmuhlestrasse 14
8050, Zurich, Switzerland
http://seal.ifi.uzh.ch/ghezzi/
ghezzi@ifi.uzh.ch

ABSTRACT

We propose a distributed and collaborative software analysis platform to enable seamless interoperability of software analysis tools across platform, geographical and organizational boundaries. In particular, we devise software analysis tools as services that can be accessed and composed over the Internet. These distributed services shall be widely accessible through a software analysis broker where organizations and research groups can register and share their tools. To enable (semi)-automatic use and composition of these tools, they will be classified and mapped into a software analysis taxonomy and adhere to specific meta-models and ontologies for their category of analysis. We claim that moving software analysis "outside the lab and into the Web" is highly beneficial from many point of views. Simple, common analyses can be effortlessly combined together into much meaningful, complex and novel ones. Analyses can be run everywhere and anytime without the need to install several tools and to cope with many output formats. Empirical studies can be easily replicated. At last, we claim that this will greatly help in the maturing of the field and boost its role in supporting software development practices

1. MOTIVATION

Successful software systems must continuously change or they become progressively less useful. However, these modifications lead to a slow but continuous increase in complexity and deterioration in quality and usability, known as Entropy or "software rot". The continuous changes, so important for the success of a software, are thus also the main causes of its dismissal. Therefore, as a software evolves, more and more resources are needed to preserve and simplify their structure. Having an always up to date and thorough view of a software system greatly helps in avoiding this problem—or at least keeping it under control. Historical data stored into repositories by systems such as version control, bug and issue tracking, mailing lists, etc. is crucial for that purpose. Studies using this data to analyze various aspects of software development (*e.g.* software design/architecture, development process and dynamics, etc.), have emerged and flourished only in the last decade. These studies have highlighted the value of collecting and analyzing this data. Yet, each of these studies has built its own methodologies and tools to extract, organize and utilize such data to perform their research. This means that, for every required analysis, a specialized tool, with its own explicit or implicit meta-model dictating how to represent the input and the output, has to be installed, configured and executed. Thus the sharing of information between tools is only possible by means of a cumbersome export towards files complying to a specified exchange format. Even if different analyses of the same kind exist (*e.g.* code duplication analysis), there is no way to compare their results or integrate them other than with manual investigation. Tool interoperability is hampered even more by their stand-alone nature, as well as their platform and language dependence.

Therefore, despite this richness, the field still lacks of ways to effectively and systematically share, integrate and study data coming from different analyses when we need to gain a deeper insight into a software system's evolution.

ICSE '10, May 2-8 2010, Cape Town, South Africa

2. THE SOFAS APPROACH

Our vision is to tackle the problem by devising a *distributed and collaborative software analysis platform to allow for interoperability of software analysis tools across platform, geographical and organizational boundaries*. Such tools will be mapped into a software analysis taxonomy and will adhere to specific meta-models and ontologies for their category of analysis and offer a common service interface that enables their composite use on the Internet. These distributed analysis services will be accessible through an incrementally augmented catalog, where organizations and research groups can register, share and compose their tools. To fulfill this vision, three main research topics need to be tackled.

1. **Analysis Web Services**
   These services need to be described, categorized and kept track of in a consistent way in order to be effectively used. Moreover it should be possible to (semi)-automatically composed them to provide higher-level services or complex analysis workflows.
2. **Knowledge sharing and usage**
   The different kinds of software analysis data need to be defined and structured in a homogeneous and consistent way (something which the field still lacks). In this way, information coming from different analyses can be easily composed and integrated. This is crucial

to allow more advanced and complex analysis workflows and thus substantially enrich the body of knowledge about a software system. At last, to really take advantage of all this wide-ranged body of knowledge, query languages and facilities are needed.

3. **Added Value**
   Tools and platforms aimed at supporting Software Evolution Analysis already exist. However, as we have seen, many issues are still open. We need to prove that our approach helps in solving them and that its advantages and the novelty set it apart from the existing solutions.

The research aimed to solve these challenges can be split into three main bodies of work, corresponding to major contributions:

1. **Distributed Software Analysis Architecture**
   The goal is to build services for several existing software analysis tools (starting with the ones developed in our research group in the past years) and all the supporting infrastructure to effectively track, use and manage them. The idea of exposing software analyses as web services and creating an online platform to share them to overcome the known language and platform barriers is novel and has not yet been proposed. Moreover we claim that it can be highly beneficial for the software analysis field as whole, as it offers a platform to facilitate collaboration between different research groups, it opens up analysis tools and data to outside researchers from other fields and it can lead, in the future, to an extensive online repository of data extracted from OSS projects.This work is split in three parts:
   (a) **Initial investigation:** Study the state of the art of software analysis and of the few existing similar approaches. Test early ideas and investigate their feasibility with early prototypes. Lay down the first version of the overall solution.
   (b) **First architecture prototype:** Based on the initial investigation, develop a first complete, working prototype. This includes, picking the appropriate technologies and tools, creating software analysis web services based on the analyses developed by our research group and create all the rest of the infrastructure to effectively share, use and manages these services.
   (c) **Stable architecture:** With the lessons learned, make the transition from a research prototype to a stable and flexible application which will then be released to the public. In this way external users can start using it and give feedback and, ideally, contribute by adding their own analyses. With this version, users will be able to fetch, register, use and combine services into predefined workflows.
2. **Ontologies and Semantic Web**
   The goal is to provide an extensible corpus of ontologies to describe the data produced by the different types of analyses and seamlessly integrate that into our architecture. To do that, we plan to use the most recent and up to date Semantic Web Technologies (e.g. we use OWL as the ontology authoring language of choice). Semantic Web is a rather new and emerging technology and only a very few works investigated using it outside its breeding ground, the Web. In particular, the use of ontologies in software engineering and software analysis has a big potential potential that has not been exploited to its fullest yet. In fact, by using OWL to describe and structure the data, we have both a clear, univoque semantics and syntax by just using one language. Moreover, query languages and reasoners already exist to easily extract and automatically infer new additional data from an existing knowledge base. This work is split in two parts:
   (a) **Development and refinement of ontologies** Develop an ontology to describe the data produced by each of the analysis service existing in the platform.
   (b) **Integration** Integration the created ontologies into the architecture.
3. **Analysis Services Description and Composition**
   In order for these services to be effectively used, a clear mechanism to describe what they offer, how they offer it and what they need as prerequisite information to function is needed. We plan to develop a service description language by which these analysis services can not only be described, but also (semi)-automatically composed in real-time into higher-level services based on their semantics and on what they offer. To do that, we are going to use the new W3C Semantic Annotations for WSDL (SAWSDL) recommendation[1], which allows to add semantic annotations to WSDL components and thus giving semantic meaning to the web services themselves, their inputs and their outputs. Apart from laying down the theoretical foundations of this description language and of the software analysis composition, we intend to build the necessary infrastructure to actually allow users to easily compose different services into workflows. Service composition in general has been addressed in countless works, but most of them dealt with the technicalities of it and focused just on small or abstract toy examples. On the other hand, our aim is to apply our service composition approach to software analysis, a real, existing case that has never been associated to this field of research. Moreover, we foresee that these services will produce and exchange huge amounts of data, knowing that it would be a real and challenging stress test for the whole approach. At last, semantics-based service composition is a new field of research and has not been concretely studied yet, as semantic web services are a totally new concept. This work can thus be divided in three parts:
   (a) **Foundations of software analysis services description** Formalization of the language used to describe software analysis services. This will not consist in developing a new web service definition language, but in the formalization of the use of new a technology, SAWSDL, to effectively describe web services.
   (b) **Foundations of semantics based service composition** Formalization of the principles of semantic based web service composition: how, based

[1] http://www.w3.org/TR/sawsdl/

on their semantically enhanced description, web services can be (semi)-automatically composed.

(c) **Support infrastructure and integration** Develop the support infrastructure for the semantic based service composition and integrate that into our Distributed Software Analysis Architecture.

As final, concrete outcome, we envision a web portal where people can share, compose and use each other's tools. In order to be effectively used both by human users and directly by computers and third party applications, this portal will be accessible through a series of human oriented rich web applications and more machine-oriented access points.

3. RELATED WORK

Software analysis is one of the key activities in software engineering as it allows to extract the most diverse and extensive information regarding a software system, *e.g.*for the purpose of evolution analysis, reengineering, etc. The classic analyses targeting models and source code have been around for years. Only in the last years many research groups have shifted their attention to software evolution and the whole established community of reverse engineering, reengineering, and program comprehension has actually acknowledged that evolution is indeed the umbrella of their research activities. There is a plethora of researches on these topics. However, a report on the state of the art is out of the scope of this paper. Approaches focusing on the software evolution either study its source code change history [18, 16], bug history [14], its underlying dynamics [1, 15] or a combination of them [3, 7]. However, all these approaches rely on their own ad-hoc developed tools and techniques and none targeted the issue of using and composing different, independent analyses.

Jin and Cordy [12] were so far the only researchers to study software analysis integration. They propose an ontology based software analysis tool integration system that employs a domain ontology and specifically constructed external tool adapters. They use a service-sharing methodology that employs a common domain ontology defining the conceptual space shared by the different tools and specially constructed external tool adapters, that wrap the tools into services. They also implemented a proof of concept with three reverse engineering tools that allowed them to explore service-sharing as a viable means for facilitating interoperability among tools. We share with them the overall concept, but at the same time, the two approaches have many differences due to their partially distinct goals. In fact, the objective of their integration effort was to be able to apply a functionality/analysis available in one tool to the fact-base of another one in a very simple way. For this reason, they used a domain ontology just to describe the set of representational concepts that the different tools to be integrated require and support. On the other hand, our goal is to offer a much broader and versatile solution. In fact, we intend to exploit ontologies on a much broader scale: to catalog and describe the services, to represent and standardize their input and output accordingly to the type of analysis offered, to semantically link different results and to perform (semi)-automatic reasoning on them.

The use of web services and semantic web technologies for software analysis, and software engineering in general, has only just recently been addressed in research by just a few works. These works all have focused on providing ontologies to representing software analysis data and concepts to foster software reuse and maintenance. For example, generic software engineering concepts (classes, tests, metrics, requirements, etc.) [11], higher level meta-data about software components (e.g. the programming language, licensing models, ownership and authorship data) [10]. More related to our approach, Kiefer *et al.* [13], developed a software repository data ontology including software, release and bug related information based on based on Evolizer's [7] data models. However none of these models, are then used for concrete software engineering tasks other than a small proof of concept.

4. EVALUATION

We intend to incrementally validate the proposed approach as major milestones are reached. More precisely, when each of the three main work packages listed above is completed. We claim that the most appropriate way to evaluate our approach is to use a series of comparative case study-based validations. In fact, the nature of the project makes other possible evaluation techniques, such as empirical studies or controlled user studies, unfeasible. On the other hand, with comparative case studies we can test the usefulness of our approach, how it performs against existing, state of the art solutions and its novelty.

4.1 Distributed Software Analysis

The goal of this evaluation is to use case studies to compare our approach with the related existing state of the art solutions, show the novelty of it and how and why it helps solving many of the problems of software analysis and software evolution that motivated our work. As use case we intend to compare how some given specific software analysis tasks can be carried out with our approach, with OASIS [12], the only similar existing approach, and with the use of separate, independent, stand alone tools. The software analysis tasks to be used could be manifold: they could be just the "replay" of existing known software analysis empirical studies or totally novel studies.

4.2 Ontologies for Software Analysis

The goal of this evaluation is to show how and why ontologies are valuable in software engineering and in particular in software analysis. The main focus of this validation will be to show how using ontologies to represent and structure software artifacts data brings a totally new perspective to the field and helps overcome the problems of the existing approaches in software artifacts representation languages and the associated query languages. This evaluation will show how, combining the use of ontologies to represent software data with querying tools like GINSENG [2], it is possible to formulate, and have answers, to questions programmers ask during software evolution tasks just by using natural language. In particular we will use the questions that were collected by Silito *et al.* [17]. Moreover we will compare our approach with the most notable existing ones and in particular Ferret [5], the one closest to ours.

4.3 Software Analysis Services Description and Composition

The goal of this last evaluation is to show that, with our approach, it is possible to (semi)-automatically compose simple, specific analyses into workflows representing more high-level, complex ones. As for the previous evaluations, the goal is to compare our approach with the currently existing solutions. The main focus will be to show

the advantages of our solution when, given a specific complex software analysis task, several specific analyses (e.g. revision history, issue tracking history, code metrics, etc.) need to be combined. As for the Distributed Software Analysis evaluation, the complex software analysis tasks to be used could be manifold. In this case study, our approach, its performance and its results will be compared with the closest existing approaches (such as Ferret [5], Evolizer [7], etc.) and, again, with the use of separate, independent and stand alone tools. This validation should be considered as a natural follow up of the previous two.

5. PROGRESS

SOFAS' basic conceptual framework has been introduced in "Towards Software Analysis as a Service" [8]. In that paper we introduced the concept of "Software Analysis as a Service" and the need for it in the field. Moreover, we sketched the overall architecture and the technologies to use. All those ideas we further refined in the book chapter titled "Distributed and Collaborative Software Analysis" [9]. In that work, we focused more on how our approach can foster collaboration from the point of view of users and of the analyses themselves. SEON [4], a corpus of software engineering ontologies comprising a generic project versioning history ontology (fitting the most common SCMs), Bugzilla and Trac issue tracking history ontologies and a static source code model ontology, largely based on FAMIX [6], has been developed and published. All the services of the Distributed Software Analysis Architecture previously reported were also modified so that they structure the data produced accordingly to the associated ontologies. At last, a work to support developers with natural language queries during program comprehension and maintenance tasks is under way. This work will be used for the evaluation proposed in Section 4.2.

The current main effort is to make SOFAS' architecture stable and flexible and integrate all the developed services and ontologies. Moreover, the creation of a rich web interface were users can compose the existing services into predefined analysis workflows is under way. Once these goals are reached, we will then focus on laying down the foundations of semantics based software analysis services description and composition.

6. REFERENCES

[1] J. Anvik, L. Hiew, and G. C. Murphy. Who should fix this bug? *Proceedings of the 28th International Conference on Software Engineering (ICSE 2006)*, pages 361– 370, 2006.

[2] A. Bernstein, E. Kaufmann, C. Kaiser, and C. Kiefer. Ginseng: A Guided Input Natural Language Search Engine for Querying Ontologies. In *2006 Jena User Conference*, May 2006.

[3] J. Bevan, E. J. Whitehead, Jr., S. Kim, and M. Godfrey. Facilitating software evolution research with kenyon. In *ESEC/SIGSOFT FSE*, pages 177–186, New York, NY, USA, 2005. ACM.

[4] J. Bielik. Seon: Designing software engineering ontologies. Master's thesis, University of Zurich, Department of Informatics, 2009.

[5] B. de Alwis and G. C. Murphy. Answering conceptual queries with ferret. In *ICSE '08: Proceedings of the 30th international conference on Software engineering*, pages 21–30, New York, NY, USA, 2008. ACM.

[6] S. Demeyer, S. Tichelaar, and S. Ducasse. FAMIX 2.1 - The FAMOOS Information Exchange Model. *Technical Report*, 2001.

[7] H. C. Gall, B. Fluri, and M. Pinzger. Change Analysis with Evolizer and ChangeDistiller. *IEEE Software*, 26(1):26–33, January/February 2009.

[8] G. Ghezzi and H. C. Gall. Towards software analysis as a service. In *Proceedings of the 4th International ERCIM Workshop on Software Evolution and Evolvability (Evol'08)*, L'Aquila, Italy, Sept 2008. IEEE/ACM DL.

[9] G. Ghezzi and H. C. Gall. Distributed and collaborative software analysis. In *Collaborative Software Engineering*, chapter 12. Springer, 2010.

[10] H. Happel, A. Korthaus, S. Seedorf, and P. Tomczyk. Kontor: An ontology-enabled approach to software reuse. *Proceedings of the 18th Int. Conf. on Software Engineering and Knowledge Engineering (SEKE 2006)*, 2006.

[11] D. Hyland-Wood, D. Carrington, and S. Kaplan. Toward a software maintenance methodology using semantic web techniques. *Proceedings of the 2nd International IEEE Workshop on Software Evolvability at IEEE International Conference on Software Maintenance (ICSM 2006)*, pages 23–30, 2006.

[12] D. Jin and J. R. Cordy. Ontology-based software analysis and reengineering tool integration: the oasis service-sharing methodology. *Proceedings of the 21st IEEE International Conference on Software Maintenance (ICSM 2005)*, pages 613–616, 2005.

[13] C. Kiefer, A. Bernstein, and J. Tappolet. Mining software repositories with isparql and a software evolution ontology. *Proceedings of the 4th International Workshop on Mining Software Repositories (MSR 2007)*, 2007.

[14] S. Kim, T. Zimmermann, E. W. Jr., and A. Zeller. Predicting faults from cached history. *Proceedings of the 29th international conference on Software Engineering (ICSE 2007)*, pages 489–498, 2007.

[15] A. Mockus and J. Herbsleb. Expertise browser: a quantitative approach to identifying expertise. *Proceedings of the 24th International Conference on Software Engineering (ICSE 2002)*, pages 503–512, 2002.

[16] N. Nagappan and T. Ball. Use of relative code churn measures to predict system defect density. *Proceedings of the 27th International Conference on Software Engineering (ICSE 2005)*, pages 284–292, 2005.

[17] J. Sillito, G. C. Murphy, and K. D. Volder. Questions programmers ask during software evolution tasks. In *SIGSOFT '06/FSE-14: Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 23–34, New York, NY, USA, 2006. ACM.

[18] T. Zimmermann, P. Weissgerber, S. Diehl, and A. Zeller. Mining version history to guide software changes. *Proceedings of the 26th International Conference on Software Engineering (ICSE 2004)*, pages 563–572, 2004.