# Discovery and Description of Software Evolution Services

Jan Jelschen
Carl von Ossietzky Universität, Oldenburg, Germany
jelschen@se.uni-oldenburg.de

## Abstract

A catalog of software evolution services is a prerequisite for the creation of an integration framework to enhance software evolution tool interoperability. The catalog acts as the framework's inventory of basic blocks for toolchain building. There is no comprehensive survey of established techniques spanning all areas of the field of software evolution. Existing techniques and tools are described in scientific publications in varying form and detail, and are not expressed in terms of services. This paper presents an approach to discover services from literature, and extract information about them along a description model.

## 1 Introduction

Software evolution projects depend heavily on proper tool support. Each project requires its own, individual workbench supporting its tasks and activities. Existing tools usually only implement single techniques, and often lack sufficient means for interoperability. Software evolution practitioners therefore have to create the required integration logic manually, a repetitive and error-prone task, yielding inflexible and non-reusable toolchains.

Building a *component-based* integration framework to address these issues requires existing techniques and tools, documented in scientific publications, to be compiled. A *service-oriented* view is taken, to keep concrete implementations (tools wrapped into framework-compatible *components*) decoupled from provided functionality (*services*), maximizing flexibility and reusability. The discovery and description process described here is aimed at producing a catalog of software evolution services [3].

There are no surveys of techniques covering the entire field of software evolution, only specific sub-areas or groups of techniques. Naturally, neither are such surveys tailored towards service identification, nor do publications on specific techniques describe them in terms of services.

Services have to be described in a concise and consistent way, providing information to make it easy to find appropriate services given a specific problem, and to allow them to be combined with other services automatically in an integration framework.

A systematic literature review [8] will be conducted, to achieve the following four goals: 1) Find publications likely to describe techniques or tools to identify service candidates, 2) confirm candidates to be proper, relevant services, 3) extract all information necessary to completely describe confirmed services, and 4) classify services to ease finding those suitable for a given task.

This paper provides an overview over the review setup and process of service discovery (Section 2) and description (Section 3). A summary is given in Section 4, with a brief discussion of ongoing work.

## 2 Service Discovery

The publication database DBLP [4] serves as basis for the literature review. It currently contains over 2.1 million computing science publications. The following process has been followed to achieve *Goal 1* and reduce this huge data set to a more manageable-sized set of publications likely to contain descriptions of tools and techniques: 1. Only publications from conference proceedings of *CSMR*, *ICPC*, *ICSM*, *IWPSE*, *SCAM*, *WCRE*, and the *Journal of Software: Evolution and Process* were selected (3 366 in total). 2. Abstracts of those publications were fetched, to have more information to base decisions regarding their relevance on. 3. Publications were *clustered* using the *Lingo* [7] algorithm on TF-IDF statistics calculated from titles and abstracts. This produced 177 overlapping clusters of papers with common topics, identified by a phrase assigned as the cluster's name. One cluster of 448 papers likely describing techniques was selected. 4. Subclusters were created to group papers describing similar techniques. For a first manual sampling, the cluster *Refactoring Techniques* (20 papers) was selected. To find phrases identifying potential services, papers were skimmed for keywords hinting at techniques or tools (e.g. *using*, *applying*, *technique*, etc.).

As an ongoing example, *Refactoring Identification*, discovered this way from Prete et al. [6], will be used.

For candidate validation (*Goal 2*), the definition given in source publications is checked to actually describe a functionality useful to software evolution activities. To ensure relevance, services are required to have been the topic of at least five publications, by authors not directly affiliated with each other. Papers in the same cluster, cited related work, and publications citing the source are considered. The threshold of five has been set randomly, and will have to be adapted further along the identification process, to find the right balance between excluding obscure services, and including required ones. Industry relevance is recognized as another important factor, but might not be evident from scientific literature. Lacking a proper measurement, it is currently not evaluated.

*Example:* Prete et al. define refactoring identification as a technique for "Automatically identifying which refactorings happened between two program versions [. . . ]". They dedicate a large section to related work, easily exceeding the relevance threshold.

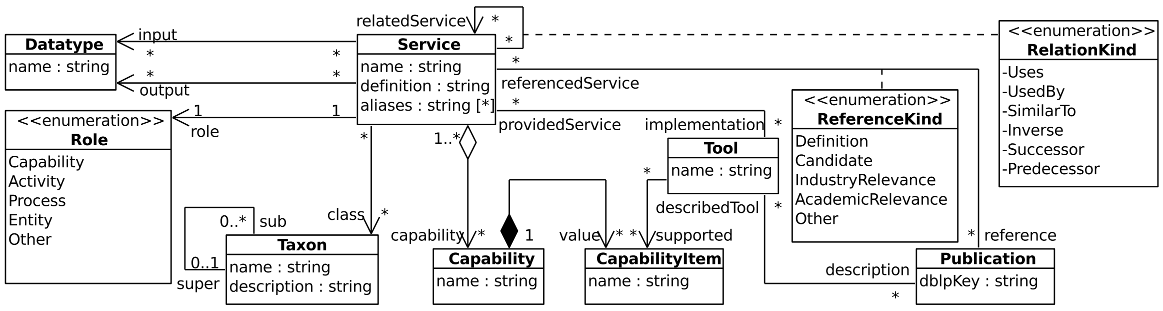

*Figure 1:* Class diagram of the service discovery and description model.

## 3 Service Description

Figure 1 shows a class diagram of the data model used to document services. For the process itself, it is important to be able to trace back *services* and *tools* to the *publications* which provided the relevant information. For services, the model allows to further specify the *kind of reference*, e.g. whether the publication serves as basic *definition*, or a case for *academic* relevance. The model's main goals are service *description* (*Goal 3*) and *classification* (*Goal 4*).

**Description.** Input and output *datatypes* are central information for an integration framework. They are identified by a name, only, to be defined rigorously in a separate step. Related work is considered to identify alternative datatypes; all possibilities are recorded for later consolidation, at which point further pre- or post-processing services might be identified. Since services only represent an abstract description of a technique, concrete *tools* implementing them have to be documented as well. Services define *capabilities* consisting of *capability items*, of which implementations may provide only a subset.

*Example:* Refactoring identification reads ASTs or source code, and might accept a similarity threshold. The output consists of code position pointers paired with refactoring names, and possibly confidence values. *REF-FINDER* [6] and *Refactoring Crawler* [2] are examples of implementing tools. Capabilities are the *programming language* implementations can handle, and the *refactorings* which can be identified.

**Classification.** Next to a primary *name*, a service can have *aliases*. A service has a textual (non-formal) *definition* describing what the service does and can be used for. Classification is possible along two lines: according to a *taxonomy* being built alongside the discovery and description process, e.g. from clustering information. Mens et al. [5] propose a comprehensive, multi-dimensional taxonomy scheme, which might be applied in the future, but for these first experiments this has not been realized. Services further have *roles*, a subset of a general-purpose service taxonomy by Cohen [1]. It distinguishes services according to their usage, e.g. services only used by other services (*Capability*), or services providing results with inherent software evolution use (*Activity*). Services can be *related* to each other, e.g. providing *similar* functionality.

*Example:* Refactoring identification is an *activity* service, and was classified using clustering information (e.g. *refactoring*, *program comprehension*), and as *code pattern detection*, a class representing commonalities with the *similar* service *clone detection*.

## 4 Summary and Next Steps

This paper presented an approach to discover, describe, and catalog *software evolution services*. Its application to the body of scientific publications in software evolution will yield a comprehensive survey of the entire field. Its contained service descriptions will serve as basis and guidance for the creation of an integration framework, and the catalog as a reference for tool implementors and framework users.

Execution of the discovery and description process is currently ongoing. To find undiscovered services within sets of yet unscreened papers, they will be classified bit by bit into the created taxonomy, using automatic classification. Publications with low-confidence classifications will be analyzed for new services and classes. This will be iterated until all relevant literature is classified with sufficient confidence. Furthermore, data structures are currently only identified by name. These will have to be described in rigorous detail in a separate work step, as this information is essential for an integration framework.

## References

[1] S. Cohen. Ontology and Taxonomy of Services in a Service-Oriented Architecture. *The Architecture Journal (Online Publication)*, 11, 2007.

[2] D. Dig, C. Comertoglu, D. Marinov, R. E. Johnson. Automated Detection of Refactorings in Evolving Components. In *ECOOP*, pp. 404–428, 2006. Springer.

[3] J. Jelschen, A. Winter. Towards a Catalogue of Software Evolution Services. *ST-Trends*, 31(2), 2011.

[4] M. Ley. DBLP - Some Lessons Learned. *PVLDB*, 2(2):1493–1500, 2009.

[5] T. Mens, J. Buckley, M. Zenger, A. Rashid. Towards a taxonomy of software evolution. *Proc. Workshop on Unanticipated Software Evolution*, 2003.

[6] K. Prete, N. Rachatasumrit, N. Sudan, M. Kim. Template-based reconstruction of complex refactorings. In *International Conference on Software Maintenance*, pp. 1–10. IEEE, 2010.

[7] O. Stanislaw, J. Stefanowski, D. Weiss. Lingo: Search results clustering algorithm based on singular value decomposition. In*Intelligent Information Processing and Web Mining: Proceedings of the International IIS: IIPWM 04 Conference*, pp. 359–368. Springer, 2004.

[8] C. Wohlin, P. Runeson, M. Höst, M. C. Ohisson, B. Regnell, A. Wesslen. *Experimentation in software engineering.* Springer, 2012.