Estimating Functional Reusability of Services

Felix Mohr

Department of Computer Science, University of Paderborn, Germany
felix.mohr@uni-paderborn.de

Abstract. Services are self-contained software components that can be used platform independent and that aim at maximizing software reuse. A basic concern in service oriented architectures is to measure the reusability of services. One of the most important qualities is the *functional* reusability, which indicates how relevant the task is that a service solves. Current metrics for functional reusability of software, however, either require source code analysis or have very little explanatory power. This paper gives a formally described vision statement for the estimation of functional reusability of services and sketches an exemplary reusability metric that is based on the service descriptions.

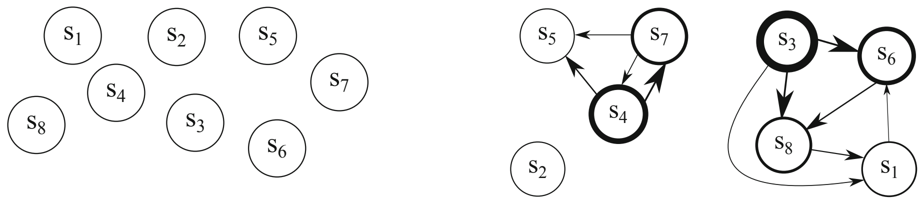
1 Introduction

During the last decade, the focus in software development has moved towards the service paradigm. Services are self contained software components that can be used platform independent and that aim at maximizing software reuse.

A basic concern in service oriented architectures is to measure the *functional reusability* of the services in general or for specific tasks. Such a metric would support the analysis of relations between services, allow to estimate the potential impact of new services, and indicate the suitability of automatization techniques like composition; Fig. 1 shows this information gain. Usually, we have no knowledge about how services in a network are related; they are merely members of a homogeneous set (Fig. 1a). Analyzing their specifications helps us recognize relations between them and identify reuse potential (Fig. 1b).

Surprisingly, there is no metric of which we could say that it is even close to be satisfactorial in regards of measuring reusability. The main problem is that most reusability metrics are based on code analysis [7], e.g. the Halstead metric and others. However, the idea of services is precisely that the implementation needs not to be available. Reusability metrics for black box components exist [4, 9, 10] but are notoriously inexpressive; that is, they do effectively not say anything about functional reusability even though that was the design goal.

This paper gives a vision statement for the service reusability question and hints to possible solutions for the problem. Intuitively, the reusability of a service s is based on the number of problems for which there is a solution that contains s. Instead of simply using this number directly as a metric for reuse, it should be somehow weighted, since the complexity and likelihood of occurrence of problems strongly varies. Unfortunately, it seems to be very hard, or even impossible,



(a) No knowledge about services relations **(b)** Insights in how service are related

Fig. 1. A metric for reusability helps us learn more about how services are related

to effectively compute such a metric in practice. As a consequence, I argue that the reusability must be *estimated* by another metric that is reasonably related to reusability. I give a sketch of an exemplary metric that measures the *contribution* of services in a service set on the basis of their description. Note that I use the terms service and component synonymously in this paper, because the distinguishing feature (way of execution) is irrelevant for us.

Summarizing, this paper contributes to the question of how reusability of services can be measured. I present a formal problem description for reusability estimation and give an example for a reusability metric.

The rest of this paper is organized as follows. Section 2 gives a deeper background for the motivation of this paper. Section 3 introduces the formal problem definition, and Section 4 sketches one approach to tackle the defined problem.

2 Background and Motivation

First of all, this paper is not about reuse but about *reusability*; more specifically, a *metric for functional reusability.* Reuse means that there is a software artifact s_1 that actually *is* employing another existing software artifact s_2. Reusability, in contrast, talks about the *potential* that, for a software artifact s, there is a set S of yet nonexistent software artifacts that *would* use s. Hence, reusability is dedicated to what *could be done* and not to what *is* done with a software component. Much has been said about *principles* of software reuse and reusability [2,3,6,8], but that debate is mostly about how to *increase* reusability through sophisticated design. In contrast, I assume a certain service architecture as given and (try to) *estimate* the reusability under these conditions.

While metrics for software *reuse* have been around for decades, most of them are unsuitable for services [7] and do not contribute to the *reusability* question.

In contrast to classical reuse metrics, metrics that estimate the functional reusability of black box components are alarmingly poorly studied. The only two metrics I am aware of are Washisaki [10] and Rotaru [9]. However, none of the two has an acceptable expressiveness power, since Washizaki only considers the *existence* of meta information and Rotaru only counts the *number of parameters* in order to express reusability. A comprehensive and exhaustive survey of these and other works is found in [1].

An important property of development environments that aim at increasing reuse is that software components are described *semantically*. We know that two key activities in component based software development are that components are *(i) stored for later reuse* and *(ii) identified as candidates for a solution of a current task* [2, 3]. The second step (look-up problem) requires a sufficiently abstract description of what a component does in order to avoid or reduce reading natural language documentation or even source code analysis. The latter may work in very small environments but not in large organizations with hundreds of components. Hence, reuse, which is the main goal of services, requires a formal specification that reasonably *abbreviates* their functional behavior.

Although software is currently rarely semantically described, current trends in software development suggest that this may change in near future. This can be mostly observed in a shift in the programming style that is less technical than before and that tends towards *workflows* [5]. Business workflow oriented software is much more suitable for business domain specific descriptions than former, rather technical component models.

Assuming that this trend continues, this paper presents one way to estimate software reusability by analyzing semantic descriptions.

3 Problem Description

This section describes the formal frame of this paper. First, I describe my idea of an ideal reusability metric in terms of composition problems to whose solution a service may contribute. Second, I explain the obstacles that occur with that metric and the necessity for alternatives. Third, I introduce the formal service model that underlies the rest of the paper.

3.1 An Ideal Metric for Functional Reusability

An ideal metric for the functional reusability of a service would be based on the number of *composition problems* to whose solution it contributes. This is because a service is (re)used if and only if there is a program that invokes it. In the service world, we would call such a program a service composition. A service composition, in turn, is only created if there is a problem that it solves. If $\mathcal{P}$ is the set of all imaginable (composition) problems, let $\mathcal{P}_s \subseteq \mathcal{P}$ be the set of composition problems $p \in \mathcal{P}$ for which there is a service composition that solves p and that invokes s.

Instead of just counting the number of problems, consider the probability that $p \in \mathcal{P}_s$ occurs at all, which yields the following ideal reusability metric:

$$r^*(s) = \sum_{p \in \mathcal{P}_s} probabilityOfOccurrence(p)$$

This metric only captures the functional reusability and ignores other factors that may affect the practical *suitability* of solutions. In fact, non-functional aspects may cause that a service s is never part of a *chosen* solution for problem p. However, we shall focus on the purely functional aspect here.

The problem set $\mathcal{P}$ is usually infinite, but we may expect the metric to converge towards a finite number for every service s. Of course, divergence is an issue, because there will be probably infinitely many "potential" problems where a service s could be part of a solution. However, I think that the number of actually occurring problems to whose solution a service s may contribute is quite limited. The probability of occurrence of other problems to whose solution s may contribute, hence, is either zero or converges to zero very fast. Concludingly, we may assume that $r^*(s)$ takes a fix value in $\mathbb{R}_+$ for every service s.

The reusability of a reference service may *normalize* the metric. Using service s^* as the reference service, the normalized reusability of a service s is:

$$||r^*||(s) = \frac{r^*(s)}{r^*(s^*)}$$

This allows us to say that a service is, for example, twice as reusable as the reference service; this makes the metric more intuitive.

3.2 The Need for Estimation

It is not possible to compute the metric r^* in practice. Computing $\mathcal{P}_s$ requires to actually *solve all composition problems* in $\mathcal{P}$ under the condition that s is part of the solution. Even if $\mathcal{P}$ is finite, it is probably very large and cannot efficiently be traversed, not to mention solving a complex composition task in every iteration. Apart from that, the probability of occurrence is usually not known and estimating it can be cumbersome.

Consequently, the task is to find a function that *estimates* the normalized ideal reusability. Formally, for a set S of services, the task is to find an efficiently computable function $r : S \to \mathbb{R}_+$ that comes reasonably close to $||r^*||$.

Due to the lack of benchmark possibilities, I argue that it would not be appropriate for this task to speak of approximation. Approximation usually is associated with an approximation error $\epsilon \in \mathbb{R}_+$ that describes the maximum discrepancy between the approximation and the optimum. For example, we would like to have $(r(s) - r^*(s))^2 < \epsilon$ for all $s \in S$. It is anything but clear how this kind of assertions about $||r^*||$ can be achieved, so, strictly spoken, we cannot claim to approximate reusability.

However, it is absolutely possible to define other metrics that, due to some semantic link to reusability, are *indicators* for reliability. As a consequence, approaches for estimating reusability must qualitatively explain why it is reasonable to *believe* that they are a reliable indicator of reusability.

3.3 The Formal Framework for the Service Environment

To obtain commonsense about the nature of problems in $\mathcal{P}$, we shall not rely on natural language descriptions but on formal descriptions. In this paper, I assume that services are described through inputs, outputs, preconditions, and effects (IOPE), and where problems are characterized by an initial state and a goal state that must be reached from the initial state through the application of services.

Definition 1. *A **service description** is a tuple (I, O, P, E). I and O are disjoint sets of input and output variables. P and E describe the precondition and effect of the service in first-order logic formulas without quantifiers or functions. Variables in P must be in I; variables in E must be in $I \cup O$.*

As an example, consider a service *getAvailability* that determines the availability of a book. The service has one input b for the ISBN of a book and one output a for the availability info; we have $I=\{b\}$ and $O=\{a\}$. The precondition is $P = \mathit{Book}(b)$ and requires that the object passed to the input b is known to have the type *Book*. The effect is $E = \mathit{HasAvInfo}(b, a)$ and assures that the object where the output a is stored contains the info whether b is available.

I acknowledge that this formalization may not always be adequate, but it is the by far most established service description paradigm besides finite state machines (FSM), and even FSM service representations can often be efficiently transformed into an IOPE representation. The introduction of a(n even) more sophisticated formalism is beyond the scope.

Definition 2. *A **composition problem** is a tuple $(S, \mathit{pre}, \mathit{post})$ where pre is a precondition that must be transformed into the postcondition post by arranging services from a set S described as in Def. 1. pre and post are first-order logic formulas without quantifiers and functions.*

For simplicity, I leave a knowledge base out of the model. A knowledge base is usually used to express ontological information and logical implications in the model but is not needed to explain the idea of this paper.

4 Estimating Reusability Using Semantic Descriptions

This section gives a brief sketch about one possibility to use semantic service descriptions to estimate their reusability. A service s is most likely to be reused if there are many other services that can do something with the effect of s; then s *contributes* to those services. This section defines the relevance of a service based on its contribution to other services and the relevance of those services in turn. The higher the relevance of a service, the higher the number of problems that can be solved with it; this relevance is a good estimation for reusability.

4.1 The Service Contribution Graph

We can capture the direct relation of two services in a *service contribution graph*. Given a set of services S with descriptions as in Def. 1, a service contribution graph is a graph (V, E) with exactly one node in V for every service in S and with an edge $(s_i, s_j) \in E$ if and only if at least one literal in the effect of s_i and the preconditions s_j can be unified. Intuitively, there is a link between from

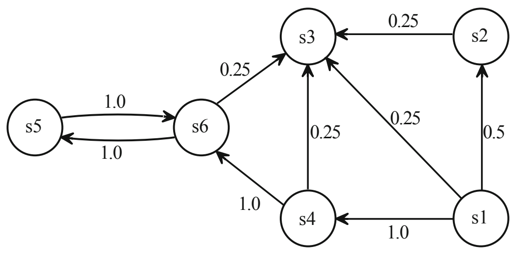


Fig. 2. An exemplary contribution graph for 6 services

s_i and s_j if the effect of s_i has something to do with the precondition of s_j. A service contribution graph is directed and usually cyclic.

The service contribution graph exploits the idea that a service composition contains a service only if it is necessary for a (possibly indirect) successor service or for the desired result of the entire composition. An edge between u and v indicates a chance that there may be compositions that invoke v after u. Thereby, the service contribution graph gives a rough insight into how control flows of service compositions in this environment could look like.

To give consideration to the fact that contributions of services to the preconditions of other services usually vary, the edges in the service contribution graph should carry a weight. A weight function $w : E \to [0, 1]$ indicates for an edge (u, v) to which *degree* service u contributes to v. That is, to which extent the effects of u cover the preconditions of v. An example is depicted in Fig. 2.

A good implementation for the weight function is a modified leveled matching. Usually, a leveled matching algorithm determines to which degree the preconditions of services s_1 and s_2 match and to which degree the effects of s_1 and s_2 match. In this case, however, we want to know to which degree the *effect* of s_1 matches the *precondition* of s_2; the precondition of s_1 and the effect of s_2 are not of interest here. If preconditions and effects are conjunctions, a basic matcher could return the percentage of literals in the precondition of s_2 that are contained in the effects of s_1. For other precondition structures, the matcher would need more sophisticated techniques.

The matcher should take into account the *explanatory power* of the different description elements. For example, a data type information is much less significant for the service description than an ontological concept or even a relation.

4.2 Basic Service Relevance

A basic reusability estimation could measure the *recursive* contribution of a service to the preconditions of other services. For example, if a service s_1 only contributes to a service s_2 that does not contribute to any other service, s_1 is probably less reusable than a service s_3 that contributes to a service s_4 that contributes to five services, each of which contributing to another five services.

Since the contribution graph is usually cyclic, we must be cautious to avoid an infinite recursion for the relevance metric. We can reasonably avoid infinite recursion by assuming that no service is called twice in one composition. A good argument to do this is that we want to measure the relevance of a service by its benefits to *other* services. Otherwise the relevance of a service value would be increased only due to the possibility to invoke itself, which does not make sense. There will be compositions that are excluded by this assumption, but this should not be the usual case. In addition and in order to reduce computational efforts, we reduce a bound $k \in \mathbb{N}$ for the considered recursion depth.

The formal basis for the metric is a *composition tree*. Every node in that tree corresponds to a sequence of services, and the root corresponds to the sequence of length 1 that is just the service s itself. There is an edge from node $(s_1, \ldots, s_n)$ to node $(s_1, \ldots, s_{n+1})$ if (s_n, s_{n+1}) is an edge in the contribution graph and if s_{n+1} is not already in $\{s_1, \ldots, s_n\}$. The relevance of the service s sums up a default value of each node (here 1), weighted with the multiplied path weights from that node to the root.

The basic relevance of a service s for depth k is then:

$$r(s,k) = \begin{cases} 1 + \sum\limits_{s' \in c(s)} w(s,s') \cdot r(s', k-1) & \text{if } k > 0 \\ 1 & \text{else} \end{cases}$$

where $c(s)$ are the child nodes of service s in the composition tree[1] and $w(s, s')$ is the weight of the edge in the contribution graph. Since the maximal recursion depth k will usually be a parameter that is chosen once and then remains unchanged, let the relevance value of a service be denoted as $r_k(s) := r(s, k)$.

4.3 Discussion

The above metric is obviously very rudimentary, but it gives a clue of what a reusability estimating metric may look like. For example, the constant factor of a node could be substituted by an expert's estimation. Also, it would be a good idea to not only consider the outgoing edges of a service, but also its incoming edges in the contribution graph. However, compared to the absence of information as depicted on the left of Fig. 1, this metric already gives very useful insights. We can argue even for this simple measure that it is an estimator for reusability: The more compositions that start with a service are imaginable, the more problems will exist for which that service may be part of a solution.

Note that the alleged redundancy of execution paths that are merely permutations of each other is intended. For example, a service s_0 may contribute to both s_1 and s_2 while in turn s_1 contributes to s_2 and vice versa. Thereby, the relevance of s_0 is increased twice, once by s_1, s_2 and once for s_2, s_1. This may seem unreasonable at first, but it is actually quite what we want. The edge (s_1, s_2) only has a high weight if s_1 contributes for s_2 and vice versa. If one of the paths does not make sense, it will have a low weight anyway and will only marginally affect the relevance value of s_0.

[1] More precisely, the last service in the child, since nodes are service sequences.

5 Conclusion

This paper gives a vision statement for metrics of functional reusability of services and sketches service relevance as one possible such metric. It defines an ideal reusability metric and explains why such a metric is usually not computable. The sketched metric tackles this problem by estimating service reusability through service relevance, a recursive metric based on the contribution of services to the preconditions of other services in the network. Its explanatory power is limited by the quality of the service descriptions.

This paper is merely a first step into the direction of analyzing functional reusability, so there is great potential for future work; I just mention some options. First, it would be interesting to estimate the reusability of services in a completely different way; for example, we could use a simplification of the service model that makes the number of composition problems tractable. Second, the presented metric only works with forward edges in the contribution graph, yet we could take into account the provision of required service inputs. Third, the weights in the composition tree could be discounted depending on the depth in order to consider possible noise between the model and the real services that may affect composition. Fourth, the proposed metric could be integrated with a learning approach that collects information about how services are used together.

Acknowledgments. This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Center "On-The-Fly Computing" (SFB 901).

References

1. Fazal-e Amin, A., Oxley, A.: A review of software component reusability assessment approaches. Research Journal of Information Technology 3(1), 1–11 (2011)
2. Caldiera, G., Basili, V.R.: Identifying and qualifying reusable software components. Computer 24(2), 61–70 (1991)
3. Cheesman, J., Daniels, J.: UML components. Addison-Wesley, Reading (2001)
4. Choi, S.W., Kim, S.D.: A quality model for evaluating reusability of services in soa. In: Proceedings of the 10th IEEE Conference on E-Commerce Technology, pp. 293–298. IEEE (2008)
5. Frakes, W.: Software reuse research: status and future. IEEE Transactions on Software Engineering 31(7), 529–536 (2005)
6. Frakes, W., Terry, C.: Software reuse: metrics and models. ACM Computing Surveys (CSUR) 28(2), 415–435 (1996)
7. Gill, N.S., Grover, P.: Component-based measurement: few useful guidelines. ACM SIGSOFT Software Engineering Notes 28(6), 4 (2003)
8. Krueger, C.W.: Software reuse. ACM Computing Surveys 24(2), 131–183 (1992)
9. Rotaru, O.P., Dobre, M.: Reusability metrics for software components. In: Proceedings of the 3rd ACS/IEEE International Conference on Computer Systems and Applications, p. 24. IEEE (2005)
10. Washizaki, H., Yamamoto, H., Fukazawa, Y.: A metrics suite for measuring reusability of software components. In: Proceedings of 5th Workshop on Enterprise Networking and Computing in Healthcare Industry, pp. 211–223. IEEE (2003)