Identifying Reusable Services in Legacy Object-Oriented Systems: A Type-Sensitive Identification Approach

Manel Abdellatif, Rafik Tighilt, Naouel Moha, Yann-Gaël Guéhéneuc, Hafedh Mili, Ghizlane Elboussaidi, Jean Privat

Abstract—The migration of legacy systems to *service-oriented architecture* (SOA) is one of the main strategies for modernizing such systems. The success of modernizing legacy software systems to Service-Oriented Architecture (SOA) depends on Service Identification Approaches (SIAs), which identify reusable functionalities that could become services. In this paper, we perform a comparative analysis of service identification approaches in academia and industry. We extract from this comparative analysis several recommendations about the inputs, process and outputs that a service identification approach should have. Based on these recommendations, we propose *ServiceMiner*, a bottom-up service identification approach that relies on source-code analysis, because other sources of information may be unavailable or out of sync with the actual code. We rely on a categorization of service types and describe the code-level patterns characterizing types of services. We evaluate *ServiceMiner* on two case studies: an open-source, enterprise-scale legacy ERP system and an inventory management system. Then we compare our results to those of three state-of-the-art approaches. We show that ServiceMiner helps at identifying architecturally-significant services with 77.9% of precision, 66.4% of recall, and 71.7% of F-measure.

Index Terms—Service Identification, Legacy Systems, Migration, Service Types, Reuse, Re-engineering, Survey

1 Introduction

The maintenance and migration of legacy software systems are central IT activities in many organizations in which these systems are mission-critical. These systems embed hidden knowledge that is of significant values. They cannot be simply removed or replaced because they execute effectively and accurately critical and complex business logic. However, legacy software systems are difficult to maintain and scale because their software and hardware become obsolete [1]. They must be modernized to ease their maintenance and evolution.

A common strategy for modernizing such systems is their migration to a *service-oriented architecture* (SOA), which defines a software architectural style where systems are made of services that are reusable, distributed, relatively independent, and often heterogeneous [2].

The migration of a legacy system to a SOA usually follows typical steps [3] that includes (1) the legacy system understanding, this step aims at acquiring information about features and functionalities of the legacy system; (2) target system understanding, this step enables the design of major components of the target service oriented architecture, the standards to be used, the expected quality of services, etc.; (3) migration feasibility determination, in this step legacy-to-SOA migration feasibility is studied from technical, economical and organizational perspectives to mitigate the risk of migration failure; (4) service identification, this step consists in identifying reusable groupings—clusters of functionalities in the legacy system that qualify as *candidate services* in the target architecture to promote software reuse and avoid the development from scratch; (5) implementation and integration of services in the target SOA system; (6) service deployment and maintenance, this steps includes activities such as service publishing, service discovery, versioning, testing, etc.,

Service Identification (SI) is considered one of the most challenging steps of the migration process since it establishes the foundation for the later steps in the development of the target SOA based system [3], [4]. Several SI approaches have been proposed in the literature [5], [6], [7], [8], [9], [10], [11]. However most of them have limited identification accuracy and usually require several types of inputs (e.g., business process models, use cases, activity diagrams, etc.) that may not be available.

Also, several practitioners highlighted the importance of identifying service types when migrating legacy systems to SOA [12]. They claimed that type-aware SI provides important information on the nature and business capabilities of the identified services. Service types can be used to classify service candidates according to a hierarchical-layered schema and offers the possibility to prioritize the identification of specific types of services according to the business requirements of the migration process. Besides, existing source-code SI approaches use similar *functional-clustering criteria*—typically *cohesion* and *coupling*, which lead to candidate services that are often architecturally irrelevant for the new SOA-based system.

In previous work, we conducted separately a systematic literature review [13] and a survey with practitioners who migrated legacy systems to SOA [12]. In this paper, we combine these two studies and analyze the gaps between academia and industry in terms of the used inputs, processes and outputs of SIAs. We derive several recommen-

dations from this analysis and propose *ServiceMiner*, our type-aware SI approach to support the migration of legacy systems to SOA [14].

We propose a bottom-up approach relying on source code analysis, as other sources of information (e.g., business process models, use cases, activity diagrams, etc.) may be unavailable or out of sync with the actual code. We use a categorization of service types based on previous service taxonomies and describe the code-level patterns characterizing each type of service. We evaluate *ServiceMiner* on two open-source systems and compare its results to those of three state-of-the-art SI approaches [6], [8], [15]. We show that our approach highly automates the identification of specific types of candidate services, which are architecturally significant for the new SOA-based system.

The rest of this paper is organized as follows. In Section 2 we describe related works on service identification. In Section 3, we describe our global research methodology. In Section 4 we detail gap analysis of SIAs in academia and industry. In Section 5 we describe *ServiceMiner* and detail the detection results. In Section 6 we discuss the approach and derive some recommendations. Finally in Section 8 we conclude.

2 Related Work

Several approaches were presented to identify services from legacy software. However, only five approaches [16], [17], [18], [19], [20] considered the types of services.

Marchetto *et al.* [17] proposed a stepwise type-sensitive service identification approach that extracts reusable services from legacy systems based on dynamic analysis of java-based systems. They proposed guidelines to identify Utility, Entity and Task services from legacy systems. They executed several test scenarios and extracted reusable functional groupings that they qualified as candidate services. They identified Utility services by manually mining non-business-centric functionalities and looking at cross-cutting functionalities that can be grouped and exposed as candidate Utility services. They extracted candidate Entity services by analyzing the persistent objects and the classes using them. Finally, they considered each main functionality of the target application as a possible candidate Task service. Although the proposed SI approach is type-sensitive, the identification is still manual and based on executing several test scenarios that may not cover all the functionalities of the system. The approach was validated on small Java systems, limiting its application on real enterprise systems.

Huergo *et al.* [19] proposed a method to identify services based on their types. They rely on UML class diagrams of object-oriented based systems from which they derive state machine diagrams to identify the states of the objects in the system. They start by manually identifying *Master data* that they define as entity classes considered to play a key role in the operation of a business. Each Master data is considered as a candidate Entity service. Next, they derive state machine diagrams that are related to the identified Master data. They analyze the transitions on the state machine diagrams and identify Task and Process services. The identification process is also not fully automated and relies on the manual identification of master data in the system that qualify as Entity services.

Alahmari *et al.* [16] identified services based on analyzing business process models. These business process models are derived from questionnaires, interviews and available documentations that provide atomic business processes and entities on the one hand, and activity diagrams that provide primitive functionalities, on the other hand. Different service granularity are distinguished in relation to atomic business processes and entities. Dependent atomic processes as well as the related entities are grouped together at the same service to maximize the cohesion and minimize the coupling. However the implementation details of the approach is not fully described.

Fuhr *et al.* [18] considered three types of services: Business, Entity, and Utility services, which are identified from legacy code based on a dynamic analysis technique. The authors relied on a business process model to identify related classes. Each activity in the business process model is executed and classes called during the execution of an activity are considered related. The identification of services is based on a clustering technique where the similarity measurement is the number of classes used together in one activity. The identified clusters are then manually mapped into the different service types. A strong assumption of this approach is that business process models are available to execute activities.

Grieger *et al.* [20] presented an approach identifying three service types when analyzing legacy code. The first type refers to initial Design services that implement business values. These occurrences are identified based on refining the existing legacy code related to business values. The second type corresponds to coarse-grained services, e.g., business processes. These are identified based on orchestrating other services related to the same underlying business process (i.e., structural dependent services). The last service type is related to services that implement cross-cutting concerns and technical functionalities used across different services (i.e., Utility services). The identification of these services is based on partitioning the functionalities of multiple services to recover individual and common parts. The authors relied on a clone detection algorithm to extract cloned functionalities shared among different services. The identified cloned functionalities are given to software architects to decide if they should be moved into an existing service or merged into a new one. The proposed approach highly depends on the manual and iterative refinement of the identified candidate services with software architects. The approach also lacks of empirical evidence on its reliability to support software architects during the migration process as there is no information about the quality of the identified services.

We notice that there is a lack of SI approaches that are type-sensitive. These approaches focus on identifying Business, Entity, and Utility services. Most of these approaches are not fully-automated and need other inputs than the source code (e.g., business processes, execution traces, state machine diagrams, etc.) to identify services in legacy systems.

Also, a number of primary studies have been proposed in the literature about SI without taking into account service

types. Many of the proposed techniques rely on Business Process Models (BPMs), to identify services within the context of legacy migration [21], [22], [23]. These techniques decompose processes into tasks and then map these tasks to legacy source code elements to identify candidate services. Other SI techniques use heuristics based on the *technical properties* of services, as reflected in various metrics [6], [8], [9], [24], [25]. Such techniques often use these metrics to drive clustering and machine learning algorithms that identify software artifact clusters as candidate services. Other AI-based techniques use ontologies and Formal Concept Analysis to identify services in legacy systems [10], [11], [26]. However these techniques are too complex and not ready for industrial applications. There are also wrapping-based SI techniques that put service interfaces around *existing* functional components and subsystems [5], [7], [27], which solve integration problems but do not solve maintenance issues.

3 Research Methodology

In this section, we will describe our research methodology. As depicted in Figure 1, we start by studying the literature and performing a survey with practitioners to analyze the gap between academia and industry in terms of SI approaches. Based on this study, we extract recommendations for inputs, process, and outputs that a service identification approach should have. Based on these recommendations, we propose our SI approach to extract reusable services through the analysis of legacy object-oriented software systems.

4 Gap Analysis of SI Approaches in academia and industry

In this section, we will describe the study design and the results of our systematic literature review and the survey to compare SI approaches in academia and industry.

4.1 Study Design

4.1.1 Taxonomy of SI approaches

In this section, we describe the design methodology of our systematic literature review as well as the mechanisms and data that we analyse to study the state of the practice of service identification in the literature. We follow the procedures proposed by Kitchenham *et al*. [28] for performing systematic reviews.

Figure 2 depicts our methodology. We first collected research papers based on search queries. We started by identifying relevant query terms based on our research questions and the context of our work: *service identification*, *SOA*, and *migration*. Then, for each keyword, we identified a set of related terms and synonyms using an online synonym finder tool[1]. Based on these terms, we defined the following search string:

> (service identification OR service mining OR service packaging) AND (migration OR modernization OR transformation OR re-engineering) AND (legacy OR existing systems OR Object-Oriented)

We executed this search query in different scientific search engines, such as Google Scholar, ACM Digital Library, and IEEE Xplore Digital Library, Engineering Village, etc.

Our search queries returned a total of 3,246 unique references. We then filtered these references, first, based on their titles, second, based on their abstracts, and finally, based on their contents. Two of the authors manually and independently at first analyzed all the papers and then reconciled any differences through discussions. We excluded from our review the papers that met one of the following criteria:

- Papers not written in English.
- Papers not related to service identification.
- Papers about *top-down* SIAs.
- Papers that did *not* propose a technique or a methodology for service identification.
- Papers published before 2004 and after 2019.

Based on these exclusion criteria, we reduced the number of references and retain 26 papers that focus on SIAs that analyze software artifacts. We believe that our search string may not cover all query terms related to service identification (e.g., *microservices*, *decomposition*, *restructuring*, etc.) and thus we risk to miss important studies. To minimize these threats, we (1) included in our search string the most important keywords related to *service identification*, and (2) applied forward and backward snowballing [29], [30] to minimize the risk of missing important papers. Forward snowballing refers to the use of the bibliographies of the papers to identify new papers that are referenced. Backward snowballing refers to the identification of new papers citing the papers being considered. We iterated the backward and forward snowballing and apply for each candidate paper our exclusion criteria. We stopped the iteration process when we have found no new candidate paper. We performed a total of nine iterations and added 15 papers. We thus obtained 41 papers that describe different SIAs.

4.1.2 Survey on legacy to SOA migration

The survey presented in this paper was conducted between October 2017 and March 2018 and aimed to investigate the state of the practice in SOA migration, in general, and service identification in particular. Our study consisted of four main phases:

A- Preparation of the online survey. We created a web-based survey[2] using Google forms. The survey was prepared based on our literature survey of the state-of-the art methods for SI and informal discussions with some subject matter experts. This helped identify the dimensions/aspects of the questionnaire, the individual questions, and the possible answers for each question. Before publishing the survey, we performed a pilot with six potential subjects, three from academia and three from industry, to validate the relevance of the questions, their wording, the coverage of the answers, etc. The six 'testers' went through the questions and suggested minor changes. The final survey contained six sections: 1) Participants' professional and demographic

1. https://www.synonym-finder.com/

2. https://goo.gl/forms/EE31KeA7R7pUeTYI2

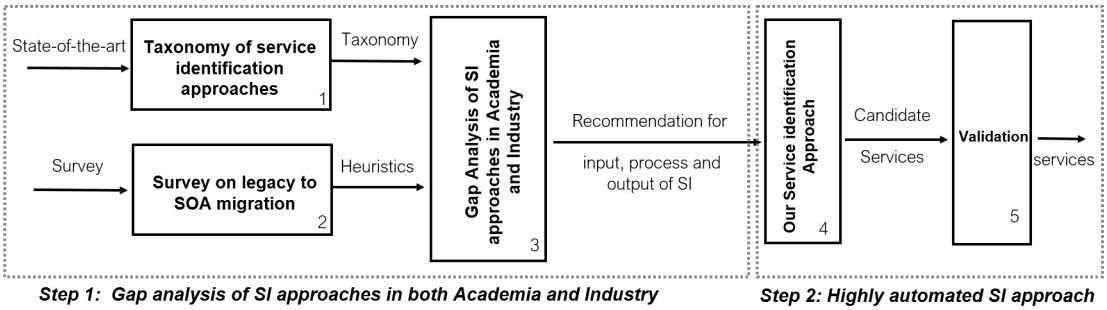

Fig. 1. Research Methodology

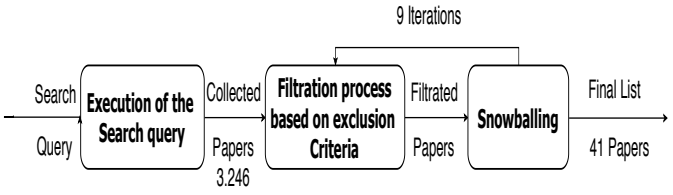

Fig. 2. Paper selection

data, 2) Type of migrated system, and reasons for the migration, 3) General information about SI methods (perception of importance, strategy, inputs, level of automation), 4) Detailed technical information about SI (technique/algorithm used, quality metrics considered), 5) Information on the types of services sought and targeted technologies, and 6) Information about the tools used, and the suggested best practices.

B- Selection of participants. We targeted developers with an industrial experience in SOA migration. Identifying and soliciting such developers was challenging. We relied on (1) information about companies that offer modernization services, (2) online presentations and webinars made by professionals that had the professional's contact information, and 3) search queries on LinkedIn profiles, such as "*legacy migration OR legacy modernization OR SOA architect OR SOA migration OR Cloud migration OR service migration OR service mining*". Once we identified potential participants, we sent them invitations via e-mail, LinkedIn, Facebook, and Twitter. We chose *not* to solicit more than three professionals from any given company to: 1) have an as wide representation as possible, and 2) to not overburden a single organization with our request.

C- Online survey. We invited 289 professionals to participate, and kindly asked them to forward our invitations to other people in their network who have experience in SOA migration and SI. The survey was completed by 47 people, two of which did not participate in SOA migration projects and whose responses were discarded, leaving us with 45 complete responses.

D- Validation. We assessed the reliability of the answers in the online survey by looking for spurious/facetious answers, contradictions between answers, etc. To be able to validate improbable answers, one question of the survey asked participants if they agreed to be contacted for a follow-up 30-minute interview, and 24 out of 45 agreed; however only eight could be interviewed. Information about the interviewed participants is detailed in Table 1. A two-pass method [31] was used to analyze our transcripts of the individual interviews[3]. The first pass of the analysis consists of *thematic coding* to identify broad issues related to legacy-to-SOA migration in general and SI in particular. The second pass of analysis was performed using *axial coding* to identify relationships among the identified issues. Major factors were identified using *meta-codes*. The *meta-codes* were then used to identify similar patterns across the data from the multiple interviewees. Overall, the answers were plausible, and the eight detailed interviews confirmed the questionnaire answers, although provided us with more in-depth information.

Participant	Profession	Years of experience	Country
P1	Technical Solution Architect	25 years	Germany
P2	Legacy modernization and enterprise IT architect	18 years	India
P3	Mainframe Modernization Specialist	33 years	USA
P4	Legacy and data Center senior consultant	30 years	Italy
P5	Software modernization expert	15 years	Canada
P6	IT Architect	20 years	Canada
P7	Director of technology	12 years	Canada
P8	Software Engineer	7 years	France

TABLE 1
Information about the participants in the interview sessions

4.2 Study results

In this section we will describe the results of our comparison study between academic and industrial SIAs in terms of the used inputs, the applied processes and the generated outputs.

4.2.1 Inputs of SIAs

We identified several types of inputs used for SI in academia and industry (e.g., source code, database, business process, user interface etc.). Figure 3 shows that both academic and industrial SIAs highly rely on source code, business process models and human expertise to identify services. This could be due to the availability of such artifacts in the context of legacy systems. We also noticed that both academic and industrial SIAs highly rely on the recovery of the business logic of legacy systems through the analyses of the source code to identify services with high business value. They also map the business processes with the legacy source code to extract reusable services through human expertise.

3. https://goo.gl/ZYv2Ut for sample transcripts

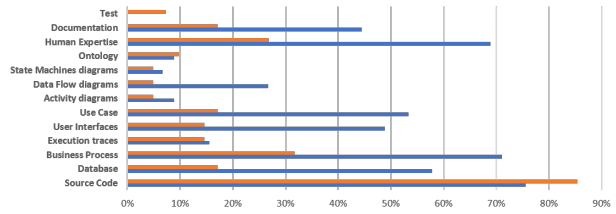

Fig. 3. Inputs of SIAs

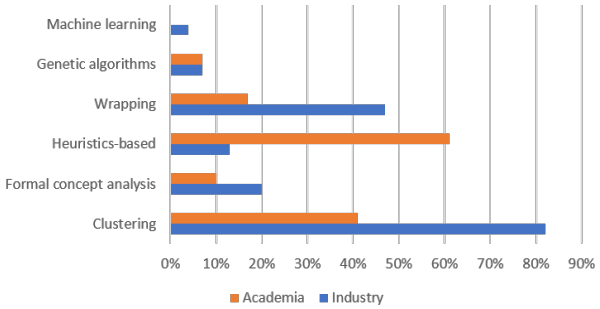

Fig. 4. Techniques of SIAs

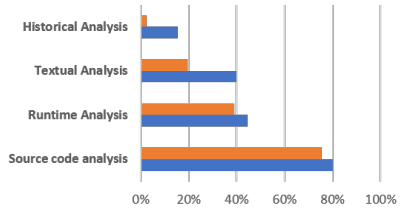

Fig. 5. Analysis types of SIAs

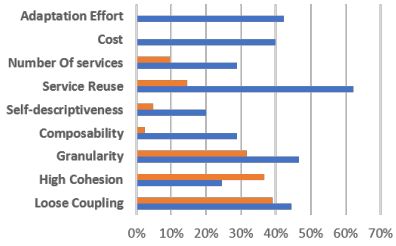

Fig. 6. Quality criteria of SIAs

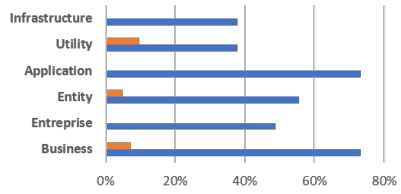

Fig. 7. Service Types of SIAs

These artifacts may help practitioners and researchers to have a better understanding of the legacy systems. On the other side, both academic and industrial SIAs rarely rely on ontologies, activity diagrams, state machine diagrams, and execution traces to identify services. This observation may be due to their unavailability or complexity to get such inputs especially when dealing with large systems. None of the participants mentioned the use of test cases to identify services. Only a few academic approaches relied on such inputs. However we found some discrepancies in the use of some inputs. In fact, although practitioners highly rely on the use of use cases, user interfaces, databases and data flow diagrams to identify services, such artifacts are less considered by researchers.

These differences could be explained by the the fact that practitioners generally migrate systems with very old technologies (COBOL, CICS, etc.) [12]. Consequently, they have/develop tools to generate documentations for these systems. They rely on these tools to automatically generate data flow diagrams for example of legacy systems and use cases that will be used to support (1) the understanding of the legacy system and (2) the identification of candidate services.

Finding 1: *Many software artifacts can be used for SI. The most widely used inputs by both industrial and academic SIAs are source code, business process models, and human expertise. There is a very low interest in relying on ontologies, activity diagrams, state machine diagrams, and execution traces to identify services in both academia and industry.*

4.2.2 Process of SIAs

In this section we will compare the processes applied in academic and industrial SIAs in terms of the used techniques and the types of analyses.

A- Techniques of SIAs

We identified several techniques that are used to identify services. We report the results in Figure 4.

- **Wrapping:** A black-box identification technique that encapsulates the legacy system with a service layer without changing its implementation. The wrapper provides access to the legacy system through a service encapsulation layer that exposes only the functionalities desired by the software architect [5], [32].
- **Genetic Algorithm:** A meta-heuristic for solving optimization problems that is based on "natural selection". It relies on the calculation of a fitness function to reach an optimal (or near-optimal) solution. By definition, an optimal solution is a feasible solution where the fitness function reaches its maximum (or minimum) value [33].
- **Formal concept analysis (FCA):** A method for data analysis where we derive implicit relationships between objects in a formal way. It is also considered as a principled way of grouping objects that have common properties [34]. To use FCA, we should first specify the context denoted by a triple $C=(E, P, R)$ where E is a set of finite elements, P is a set of finite properties and R is a binary relation based on E and P. Also a *formal concept* is defined as a grouping of all the elements that share a common set of properties. A partial order could be defined on the formal concepts with *concept lattices* [35], which also offer a structured visualization of the concepts hierarchy.

- **Clustering:** It consists of classifying and partitioning data into clusters (also called groups, categories or partitions) that share common properties. These clusters are built based on the internal homogeneity of their elements and the external separation between them. In fact, elements in the same cluster should be similar to each other while elements in different clusters should not [36].
- **Custom heuristics:** Some authors proposed their own heuristic algorithms, instead of using predefined algorithms, to decompose legacy software into SOA.
- **Feature location:** Feature location techniques aim at identifying software artifacts that implement specific functionalities in the system. These techniques are used to support software maintenance, aspect- or feature-oriented refactoring, and others [37].
- **Machine learning:** in our context, it consists of using machine learning classifiers to group and identify reusable components in legacy systems that qualify as candidate services.

We found that clustering functionalities of the legacy systems and exposing these clusters as services is the only most used technique in both academia and industry. We found several discrepancies in the used techniques to identify services in academia and industry. 47% of the participants highly relied on wrapping techniques because they considered the migration as an integration problem. They did not want to modify the core functionalities of the legacy systems because it provided useful and reliable business functionalities. This technique is however less considered by researchers who instead highly rely on custom heuristics. This technique is usually used/combined with clustering techniques to identify reusable services.

We observed a low interest by both academia and industry in using machine-learning techniques, formal-concept analysis, and genetic algorithms to identify services. Using these techniques may be challenging for practitioners because they are dealing with large systems to migrate and so the knowledge required to establish these techniques could be time consuming and may not lead to optimal results. More studies should be done by researchers to investigate the efficiency of such techniques in identifying services from legacy systems.

> ***Finding 2:*** *Although clustering is the most used technique in both academia and industry to identify services, there is a gap between the two fields in terms of the used identification techniques.*

B- Analyses Types of SIAs

SIAs may perform static, dynamic, lexical, historical analyses, or some combination thereof to identify services.

- **Static analysis** is performed without executing a software system. Dependencies between classes are potential relationships, like method calls and access attributes. These dependencies are analyzed to identify strongly connected classes, for example, to identify services. [7], [9], [22], [27], [38], [39], [40], [41], [42] are examples of identification methods based only on static analysis. The main advantage of static analysis is that it depends only on the source code. It does not address polymorphism and dynamic binding.
- **Dynamic analysis** is performed by examining the software system at runtime. Dependencies between software elements (e.g., class instantiations and accesses [43], function calls [42], [44], relationships between database tables [45], etc.) are collected during the program execution [46]. The execution is performed based on a set of cases that covers the system functionalities, called execution scenarios.
- **Textual analysis** techniques suppose that the similarity between the classes should be taken into account during service identification process. This analysis plays the main role in approaches that used features location and textual similarity techniques.
- **Historical analysis** techniques consist in considering different versions of the same system and analyzing its evolution.

Figure 5 shows that both researchers and practitioners follow the same analyses types to identify services. They highly rely on static analyses of the source code of legacy systems. They also use dynamic analyses to analyze the system's behaviour. Also, some practitioners (43%) and academic SIAs (20%) relied on textual analyses for the identification processes. Textual analyses include elements such as features identification techniques, natural language processing, legacy documentation analysis, etc. Only 18% of the participants and 2% of academic SIAs relied on historical analyses (analyses of different versions of the legacy system) to extract candidate services, which may be due to (1) the unavailability of several versions of the legacy system and (2) the difficulty to study the evolution of a legacy system to gather valuable information to identify reusable services.

> ***Finding 3:*** *Both researchers and practitioners mostly relied on static analyses of the source code to identify services.*

4.2.3 Outputs of SIAs

A number of service type taxonomies exists [2], [16], [17], [18], [19], [20], [47], [48] that consider different aspects (e.g., domain specificity, granularity, governance) to distinguish service types. We study and combine these taxonomies and limit ourselves to those services types that are *distinguishable at the code level*. We distinguish between *domain-specific services*, and *domain-neutral services*. *Domain-specific Services* fall into four major types:

1) **Business services:** They correspond to business processes or use cases. These are services used by users. These services generally compose/use the Enterprise-task, Application-task, and Entity services described in the following.
2) **Enterprise services:** (Also called capabilities [47]) they are of finer granularity than business process services. They implement generic business functionalities reused across different applications.
3) **Application services:** These services provide functionalities specific to one application. They exist to

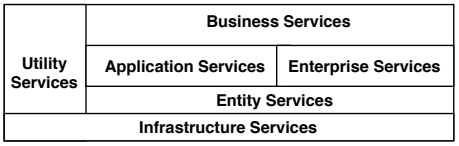

Fig. 8. Taxonomy of service types

support reuse within one application or to enable business process services [47].

4) **Entity services:** (Also called information or data services) they provide access to and management of the persistent data of legacy software systems. They support actions on data (CRUD) and may have side-effects (i.e., they modify shared data).

Domain-neutral Services are services that provide functionalities to develop, use, and compose domain-specific services:

1) **Utility services:** They provide some cross-cutting functionalities required by domain-specific services. Logging and authentication services are examples of Utility services.
2) **Infrastructure services:** They allow users deploying and running SOA systems. They include services for communication routing, protocol conversion, message processing and transformation. They are sometimes provided by an Enterprise Service Bus (ESB).

We asked the participants what are the types of services that they identify and mined in the literature type-sensitive SIAs. We report the results in Figure 7. We found that identifying services according to their types is highly adopted in industry. On the other side, we found that there is a lack of academic SIAs that are type-sensitive. Additionally, during the interview sessions several practitioners highlighted the importance of identifying service types when migrating legacy systems to SOA. They claimed that type-aware SI provides important information on the nature and business capabilities of the identified services [12].

Because domain-specific services represent the business core functionalities of SOA, they were the most targeted services (i.e., business and application services) during the service identification processes compared to technical services. Utility and infrastructure services were the less targeted services because they are SOA-specific services and utility services are relatively easy to implement.

> ***Finding 4:*** *Service identification is a business-driven process that prioritizes the identification of domain-specific services rather than technical services. There is a lack of type-sensitive SIAs in the literature while it is highly considered by practitioners.*

4.3 Recommendations

According to the old military adage, if the terrain differs from the map, trust the terrain. With legacy systems, documentation (the map) may be absent or awfully out of date. The source code (the terrain) is the only reliable source of information about what the current system does. Therefore, a typical service identification approach should consider the analysis of the source code of legacy systems (when available) to accurately identify reusable services. Source code analysis could also be complemented by other inputs such as business processes and human expertise.

Regardless of the service identification approach the output must be high-value, coarse grained domain services. While the research literature has been concerned by the implementation (and quality thereof) of services, it should seek to defined, assess, and optimize the business values of the identified services. Service identification should depend on service types to improve the identification accuracy by narrowing the search space through the types and their associated code-patterns. Service types can be used to classify service candidates according to a hierarchical-layered schema and offers the possibility to prioritize the identification of specific types of services according to their business values and requirements of the migration process.

None of the interviewed practitioners mentioned the use of research papers or academic resources for their migration projects. From the point of view of practitioners, *"academics do not see the larger picture of the real industrial problems and challenges they are facing"* as stated by one of the interviewed practitioners. The lack of cost-effective academic service identification technique and the lack of validation on real enterprise-scale systems is the most problem that hinder knowledge transfer between academia and industry in the context of legacy-to-SOA migration.

5 Service Identification by Type: Our Approach

Based on the recommendations derived from our literature review and the industrial survey, we propose *ServiceMiner*, a type-sensitive bottom-up approach that relies on static analysis and clustering techniques to identify services from legacy systems.

5.1 Study Design

Figure 9 summarizes our SI approach, *ServiceMiner*, which consists of two phases: (1) a *pre-processing phase* in which we build the call graph of the system based on source code analyses, perform an initial clustering of highly connected classes, and compute the code metrics used in the services detection rules and (2) a *processing phase* in which we apply rules on the generated clusters to filtrate, reorganize, and classify them to identify candidate services and their types.

5.1.1 Pre-processing Phase

A- Call Graph Generation: Our SI rules in Table 2 use code metrics, such as fanin and fanout, computed on the *call graph* of the legacy system. Thus, in a first step, we parse the source code of the legacy system and build its call graph. Legacy systems come in different languages and may combine several technologies. The OMG *Knowledge Discovery Metamodel* (KDM) [49] was defined to represent (legacy) systems at different levels of abstraction, regardless of languages and technologies. Thus, we use MoDISCO [50], an Eclipse-based open-source implementation of the KDM

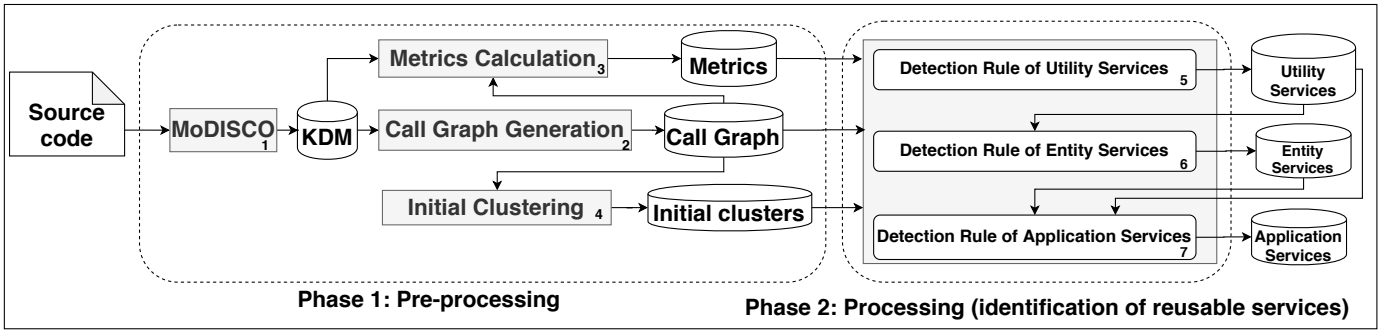

Fig. 9. Overview of our SI approach

that provides (1) an extensible parsing framework to obtain KDM models from files in different languages and (2) a framework to navigate the KDM models, which we extend to generate call graphs of legacy systems.

B- Metrics Calculation: Our rules use class-level and method-level metrics. We use the call graphs obtained in the previous step to compute class-level metrics. For the sake of simplifying the implementation of our SI approach, we use *Understand*[4] to compute method-level metrics. We also analyze the static relationships between the modules of the systems and assign a weight to each of them according to their relative importance. A module may be a procedure, an object, or any other piece of software depending on the programming language. A relationship may be *generalization*, *aggregation*, or *association*, between classes in object-oriented systems for example. The total relationship strength between a pair of related modules is:

$$Weight(C_i, C_j) = \sum_{t=1}^{T} W_t \times NR_t$$

where C_i and C_j are modules, T is the number of relationships, W_t the weight assigned to a relationship type t, and NR_t the number of type t relationships between C_i and C_j. We study the relationships to ensure that only related modules are grouped together in the following steps.

C- Initial Clustering : The SI rules in Table 2 apply to candidate clusters that group functionalities in a legacy system. Finding such groupings within a call graph is akin to a call-graph clustering problem. We rely on Kruskal's maximum spanning-tree algorithm [51] to generate our initial set of clusters because (1) it is an efficient polynomial-time algorithm for generating clusters based on a graph structure, (2) it was used by several state-of-the-art SI approaches [8], [15], and (3) a free implementation of the algorithm is provided in the open-source Java library *Jgrapht*, which can be easily integrated into our implementation. To enhance the clustering results of the spanning-tree algorithm, we put the modules that are reachable only from certain other modules in the same cluster. For example in case of object-oriented systems, if a class `A` is only accessible from a class `B` and the two classes are not in the same cluster then they must be grouped in same one.

4. http://www.scitools.com

5.1.2 Processing Phase: Identification of Reusable Services

Not all the clusters qualify as candidate services. In this step, we select the functional groupings/clusters to migrate while considering the different types of services. We first discuss service types and their code patterns qualitatively and then express them as rules (see Table 2).

In the following, we will consider the identification of only Utility, Entity and Application services and detail how we can identify such types of services through the analysis of the source code of legacy software systems. In fact, non service-oriented legacy systems are likely not to contain *infrastructure services* and, thus, we do not consider this type of services in our analysis. Second, we do not distinguish between *Application services* and *Enterprise services* because they only differ in terms of scope of reuse: within a single system vs. across systems. Also, we do not consider *Business services* because (1) they orchestrate other services, such as *Enterprise* and *Application services*, and (2) other sources of information, e.g., business process models, are required to detect them.

Each type of service is mutually exclusive and their detection is hierarchical: first we detect candidate *Utility services*. Within the remaining groupings, we detect *Entity* and *Application services* as follows:

- **Utility services:** They provide highly generic functionalities that are separate from domain-specific business processes and reusable across a range of business functionalities [2]. We detect Utility services by identifying groupings that satisfy the rule described in Table 2: high fanin (groupings that are highly solicited/called) and low fanout (groupings that do not call many other clusters). Utility services are *domain-neutral* so the identified groupings should not be persistent or contain database queries.
- **Entity services:** They represent and manage *domain-specific* business entities, such as products, invoices. They are *data-centric* and reusable by other *domain-specific* services, such as Application services. We classify a grouping as an Entity service with (1) high fanin, (2) low fanout, (3) persistent modules, (4) access to the infrastructure (e.g., database), and (5) fine grained.
- **Application services:** They are domain-specific and provide business functionalities specific to one sys-

tem. They have low fanin compared to Entity and Utility services. They can also compose functionalities provided by Entity services. They generally perform complex computation. We use McCabe complexity metric as well as error handling capabilities to measure complexity computation. We classify a grouping as an Application service if it has (1) a call to at least one Entity service, (2) a high McCabe complexity, and–or (3) an error handling.

Service Type	Detection rules
Utility Services	Very High Fanin **AND** Very Low Fanout **AND Not** persistent
Entity Services	**Not** Utility service **AND** High Fanin **AND** Low Fanout **AND** Persistent **AND** Access to infrastructure **AND** Fine grained
Application Services	**Not** Utility **AND Not** Entity **AND** Low Fanin **AND** (Call to Entity $\geq$1 **OR** High McCabe Complexity **OR** Error Handling)

TABLE 2
Detection rules of services according to their types

5.2 Experimental Validation

Our validation divides into (1) a quantitative validation of *ServiceMiner* on a case study in comparison to a ground-truth, (2) a qualitative validation of the identified services that are related to a particular feature of our case study, and (3) a comparison of our identification results with those of two other state-of-the-art approaches [6], [8].

5.2.1 Case Studies

In this section we will describe our two case studies that we used to validate *ServiceMiner*

A- Compiere: As case study, we choose *Compiere* because it is one of the few available, large, and open-source *legacy* system available on the Internet. Compiere is a *legacy system* because it is a large ERP system with a long history, first introduced by *Aptean* in 2003[5]. It provides businesses, government agencies, and non-profit organizations with flexible and low-cost ERP features[6], such as business partners management, monitoring and analysis of business performance, control of manufacturing operations, warehouse management (automating logistics), purchasing (automating procurement to payment), materials management (inventory receipts, shipments, etc.), and sales order management (quotes, book orders, etc.). It supports different databases, such as Oracle and PostgreSQL. We use Compiere v3.3 because (1) it is the first stable release of the system, (2) it was released more than 15 years ago, (3) it includes 2,716 classes for more than 530 KLOC, and (4) it is not service-oriented.

5. http://www.aptean.com
6. http://www.compiere.com/products/capabilities/

B- FXML-POS: it is an open-source[7] inventory management system based on Java. It includes MANEL ►*XXX*◄ classes and MANEL ►*XXX*◄ LOC. It is a model view controller (MVC) based system that embeds several features such as purchase management, sales management, supplier management, etc. We use this system for our validation because (1) it is not service-oriented; and (2) it is a small system (we want to generalize the validation of *ServiceMiner* on both a large and a small system).

5.2.2 Ground-Truth Architecture

We need a *ground-truth* service-oriented architecture of our case studies to assess our approach. We asked two independent Ph.D. and Master's students to identify services in our case studies. They relied on several artifacts to build manually the ground-truth architecture by (1) analyzing the system, (2) understanding it, and (3) extracting its reusable parts that could become services. They used *Understand* to recover its design and to visualize class dependencies. They also generated views of its call graph that we make available online[8]. They also reviewed extensively the system documentation as well as its source code to have the best possible understanding and accurately identify services that can be integrated in the targeted SOA-based system. They found 473 services in Compiere and 18 services in FXML-POS, which they annotated manually according to their type.

5.2.3 Evaluation metrics

We assess our approach with respect to a ground-truth architecture and in comparison to other tools using measures of clustering and information retrieval: MojoFM, A2A, precision, and recall.

A. MojoFM

measures the similarity distance between two architectures A and B [52] by counting the numbers of changes to apply to A to obtain B. Changes include moves (Move) of classes from one cluster to another cluster and merges (Join) of clusters. It is defined as:

$$MojoFM(A,B) = \left(1 - \frac{mno(A,B)}{max(mno(\forall A,B))}\right) \times 100\%$$

with $mno(A,B)$ the minimum number of Move and Join operations needed to transform A into B. In our paper, A represents the set of services identified by one approach while B is either those provided by another tool or the ground-truth services. $MojoFM(A,B)$ reflects the amount of effort needed to transform one architecture into another: the greater the value, the less the effort.

B. Architecture2Architecture (A2A)

is also a measure of similarity that includes five operations to transform one architecture into another [53], which overcomes some of MojoFM limitations [54]. It is defined as:

$$A2A(A,B) = \left(1 - \frac{mto(A,B)}{aco(A) + aco(B)}\right) \times 100\%$$

7. https://github.com/sadatrafsanjani/JavaFX-Point-of-Sales
8. http://si-serviceminer.com

	# Services		Precision		Recall		F-measure	
	Compiere	POS	Compiere	POS	Compiere	POS	Compiere	POS
Application	24	6	(18/24) 75.0%	(5/6) 83.3%	(18/30) 60.0%	(5/10) 50%	66.7%	62.4%
Entity	278	8	(223/278) 80.2%	(6/8) 75%	(223/358) 62.3%	(6/8) 75%	70.1%	75%
Utility	101	4	(73/101) 72.3%	(3/4) 75%	(73/85) 86.0%	(3/5) 60%	78.6%	73.6%
Total	403	18	(314/403) 77.9%	(14/18) 77.8%	(314/473) 66.4%	(14/23) 60.9%	71.7%	70.3%
Average	n/a	n/a	77.9%		63.7%		71%	

TABLE 3
Overview of Service Identification Accuracy with *ServiceMiner*

with $mto(A,B) = remC(A,B) + addC(A,B) + remE(A,B) + addE(A,B) + movE(A,B)$ and $aco(A) = addC(A_0,A) + addE(A_0,A) + movE(A_0,A)$: $mto(A,B)$ is the minimum number of operations needed to transform architecture A into B and $aco(A)$ is the number of operations needed to construct architecture A from an "empty" architecture A_0. These two functions, mto and aco, use additions $addE$, removals $remE$, and moves $movE$ of classes from one service to another and additions $addC$ and removals $remC$ of services.

C. Precision and Recall

are defined commonly as:

$$Precision = \frac{TP}{TP+FP},\ Recall = \frac{TP}{TP+FN}$$

with TP the number of correctly identified services, FP the number of wrong services, and FN the number of missing services. We also use the F-measure, which is the harmonic average of the precision and recall

$$F-measure = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

5.2.4 Quantitative Validation

We applied *ServiceMiner* on our case studies to show its practical accuracy in identifying services in an existing system by considering, for each service type, several combinations of the criteria related to each rule. We measured precision, recall, and F-measure for each rule and report the results in Figures 10, 11, and 12.

For example, for Utility services, we considered several possible combinations of the criteria, such as *"very high fanin"*, *"very low fanout"*, and *"Not persistent"*. We tested all possible combinations of these criteria and measured precision, recall, and F-measure for each rule. As shown by Figure 11, the best F-measure is obtained when considering the three criteria to identify such type of services: considering clusters with only very high fanin or very low fanout or clusters that are only not persistent leads to poor precision values.

We did the same evaluation process to study the effectiveness of the detection rules for Entity and Application services. As shown by Figures 10, 11, and 12, the best F-measure values were obtained when applying all the detection rules detailed in Section 5.1.2, which we used in *ServiceMiner* to identify in Compiere 403 services: 24 Application services, 278 Entity services, and 101 Utility services. We report in Table 3 the accuracy of *ServiceMiner*: a precision of 77.9%, a recall of 66.4%, and a F-measure of 71.7%.

Table 3 shows that the best accuracy of *ServiceMiner* pertained to Utility services with a precision of 77.9% and a recall of 86%. The identified Utility services relate to logging, Web uploading, printing, etc. For Entity services, we obtained a precision of 80.2%, a recall of 62.3%, and a F-measure of 70.1% with services for products, orders, invoices, etc. We missed some Entity services that have a low fanin/a high fanout because of our choice of metrics and thresholds, which could be refined by the developers when applied to their own systems. We observed a precision of 75%, a recall of 60%, and a F-measure of 66.7% for Application services, which relate to payment processing, tax calculation, and inventory management. The identification of Application services depends on the previous identifications of Entity and Utility services and, thus, false-positive Application services were mainly due to some Entity and Utility services being incorrectly labeled as Application services, such as caching-related services and Web-project deployment services.

Although we missed the identification of some services, we reduced the developers' effort needed to identify services by avoiding the manual identification of at least 66% of the candidate reusable services. Our recall could be improved by setting different thresholds and iterating through the identification process.

5.2.5 Qualitative Validation

We apply *ServiceMiner* on Compiere and FXML-POS to identify relevant services in these systems. We take the example of the sales orders management in Compiere and detail how *ServiceMiner* helps practitioners identify services related to this feature.

Sales orders management entails quotations, sales orders, and invoicing, linked to the shipment of goods to customers. The initial clustering step of our approach builds a set of candidate clusters that we then filtrate with our detection rules to identify candidate services. First, we identify Utility services by applying the first rule in Table 2, which yields Utility services about logging and printing. These services provide cross-cutting functionalities called by multiple other services (e.g., sales) with very low fanout and no persistence.

Second, we identify Entity services, i.e., clusters representing business entities. They refer to the business entities of the system, such as products, invoices, business partners, warehouses, and bank statements. These entities are persistent, have access to the database, and are invoked by other domain-specific services (e.g., Application and Business services).

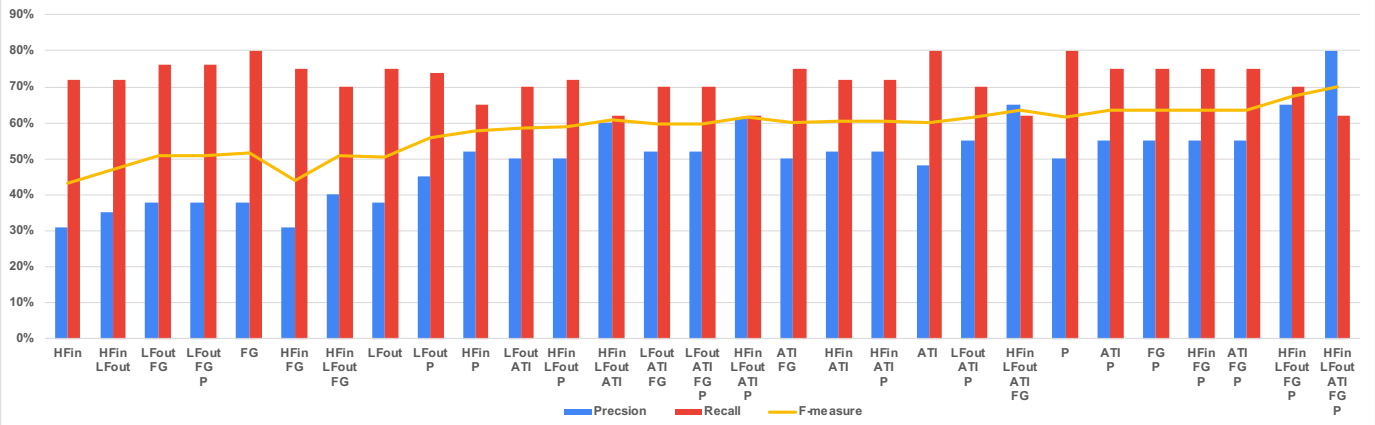

Fig. 10. Evaluation of the detection rules of Entity Services

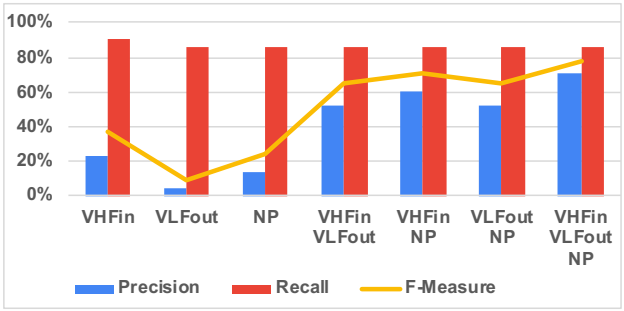

Fig. 11. Evaluation of the detection rules of Utility Services

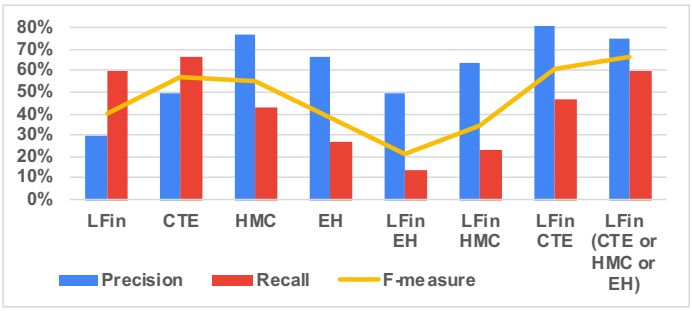

Fig. 12. Evaluation of the detection rules of Application Services

Third, we apply our last detection rule in Table 2 to identify Application services among the remaining clusters. An example of Application service related to sales orders processing is the payment service responsible for generating payments of the orders based on the information provided by the invoice, business partner, and bank statement Entity services. It is also responsible for handling errors when the payment is unsuccessful.

We obtained architecturally significant candidate services thanks to the application of our type-aware service identification approach. We believe that it can assist practitioners in the identification of candidate services because it automates the SI process of Utility, Entity, and Application services with acceptable precision and recall.

5.2.6 Comparison with State-of-the-art Approaches

We chose three existing approaches to compare their results against those of *ServiceMiner*: *MOGA-WSI* [8], *Service Cutter* [6] and *MicroserviceExtraction* [15]. These three were the only available approaches. *MOGA-WSI* uses spanning trees and provides candidate services with different levels of inter-service coupling. It relies on genetic and multi-objective optimization algorithms to refine an initial set of services. It also considers a set of managerial goals, such as cost effectiveness, ease of assembly, customization, reusability, and maintainability. *Service Cutter* is a graph-based approach considering 16 coupling criteria and two kinds of clustering algorithms, *Girvan-Newman* (GN) [55] and *Epidemic Label Propagation* (ELP) [56], which differ in terms of their (non-)deterministic behaviour. *MicroserviceExtraction* is also a graph-based approach that relies on both semantic and static coupling criteria. It relies on the historical analysis of the system by analyzing its evolution on Github to identify semantically related classes. Coupling criteria are used to identify clusters on the system's call graph which refer to candidate services.

We assess our approach with respect to a ground-truth architecture and in comparison to the three tools using measures of clustering and information retrieval: MojoFM [52], *Architecture2Architecture* (A2A) [54], precision, and recall. We rely on these metrics to study the identification results of each approach regardless of the types of services.

Table 4 lists the identification results and shows that our approach outperforms *MOGA-WSI*, *Service Cutter* and *MicroserviceExtraction* for all the reported metrics. We tried several configurations for both tools but all showed poor results in comparison to our approach. We observed that these approaches generate very unbalanced services. For example, *MOGA-WSI* identified a service of 253 classes and 143 services of one to six classes. Similarly, *Service Cutter* (EPL) identified two coarse-grain services and 393 fine-grained ones. Although service identification using *Service Cutter* with *Girvan-Newman* is deterministic, we were limited to a maximum number of 30 services, which lead to poor identification results.

We argue that our SI approach outperforms the three other approaches because: (1) *ServiceMiner* follows a stepwise process, which identifies Utility services then Entity services and finally Application services, (2) it uses simple,

straightforward metrics instead of complex goals, like maintainability, which are subjective and more difficult to define and measure, and (3) the studied state-of-the-art tools are proof of concepts, which may limit their applicability on enterprise-scale systems, such as Compiere.

Approach	#Services	MojoFM	A2A	Precision	Recall	F-measure
MOGA-WSI	396	11.0%	42.0%	14.0%	13.0%	13.5%
Service Cutter (EPL)	395	15.7%	51.0%	12.2%	10.3%	11.2%
Service Cutter (GN)	30	21.6%	41.0%	15.6%	9.7%	14.1%
ServiceMiner	**403**	**65.0%**	**73.0%**	**77.9%**	**66.4%**	**71.7%**

TABLE 4
Comparison results of service identification approaches MANEL ►*A mettre à jour*◄

6 Threats to Validity

Internal validity. Social desirability is a bias that leads any respondent to deny undesirable traits and report traits that are socially desirable. To minimize this threat, we did not put any incentives for the participants to participate in the interviews. We also guaranteed the interviewees their anonymity and emphasized that all the reported information will be only for research purposes.

In our SLR, we may have missed some important paper because of our search string that may not cover all query terms related to service identification, and because of our filtering process to keep only relevant studies. To minimize these threats, we tried to include in our search string the most important keywords related to service identification, and applied forward and backward snowballing to minimize the risk of missing important papers.Also, two of the authors manually and independently analyzed all the papers and then reconciled any differences through discussions.

Our SI approach as well as its validation depend on several algorithms and thresholds that threaten the internal validity of our results. To mitigate these threats, (1) we implemented *MOGA-WSI* based on their original papers to the best of our understanding and shared it for investigation and replication[9]; (2) we used the best identification results of these tools to compare our approach; (3) we explored the use of different metrics and threshold values; and, (4) we chose the spanning-tree algorithm for its ease of use and available, open-source implementation. Future work should consider comparing with other algorithms to vet further the reliability of our approach in comparison to other SI approaches.

We know that service detection rules may slightly differ from one system to another. However, our detection rules are easily customized, being flexible and extensible. We recommend to consider the same processing steps in the same order than in our approach to identify the services in existing systems according to their types. Also, legacy systems most likely embed poor design and coding practices, e.g., code smells, that reduce the separation of concerns within/between classes, which reduces the precision/recall of static-based SI approaches.

Construct validity. In our validation process, we should have relied on a real post-migration SOA-based system. However, we could not find any open-source enterprise-scaled system that was migrated to SOA. Thus, we relied on a *ground-truth* architecture to validate quantitatively the services identified by our approach. We are aware that there is no single "correct" SOA for a given legacy system. However, we relied on several artifacts (e.g., official documentations, source code analysis, etc.) and studied in-depth the system to identify reusable sets of classes that could be packaged into services. Also, we share our *ground-truth* architecture to allow others to confirm/infirm our claims. We put the original source code as well as the identified services online[10], which also reduces threats to reliability validity.

Conclusion validity. The information from our interviews also show some threats to the validity of our conclusion because some interviewees contradict each other or, for one interviewee, change their answers to the survey. However, this threat is acceptable because we use these interviews with the purpose to mitigate and discuss the answers to the survey.

External validity. Information from our interviews is not generalizable as the number of the interviewees is a bit on the low end for software engineering studies. However, it is acceptable given that it is unquestionably difficult to find interviewees in legacy-to-SOA migration domain. We only sought to obtain a better understanding of the results of the online survey. Also, Table 1 shows that our interviewees are experts in legacy-to-SOA migration and, thus, that our sample is still reliable because we are dealing with subject-matter experts.

Our case study may not be representative of all legacy software systems, which limits the generalizability of our results but *Compiere* is large and complex enough to validate our approach while we continue our search for other large, open-source systems. Finally, the identified services may not be representative of all service types, which may also limit the generalizabiltiy of our results. Our approach is extensible to new service types. It can also be extended to identify microservices [57] by mapping each Utility and Entity service identified by *ServiceMiner* to a microservice and decomposing each identified Application service into smaller microservices that each have a single responsibility.

7 Recommendations

We believe that our approach is beneficial for both researchers and practitioners interested in migrating legacy systems to SOA because (1) we highly automate the SI process, which is one of the most labour-intensive step for migrating such systems to SOA; (2) our SI approach yields to architecturally significant candidate services that can be packaged and integrated in the targeted SOA-based system while identifying their types; (3) our approach offers the possibility to prioritize the identification of specific service types based on their importance and the architectural/business needs of the migration process; and, (4) our approach is extensible to new technologies/languages, thanks to its use of the KDM metamodel as intermediate representation for C, C++, Python, etc.

9. https://github.com/MPoly2018/MOGA-WSI

10. http://si-serviceminer.com/ICSOC-2020-Replication

Finally, we recommend to consider service types to identify services in existing systems to improve accuracy. We also recommend to order the services to be migrated. We suggest to start first with Utility services because they are highly reusable and invoked by other services in the system (e.g., Entity, Application, Business process services, etc.); second, to continue with Entity services because they manage and represent the business entities of the system and are used by the other services; third, to identify Application services as they compose functionalities provided by Entity services; finally, to identify Business services that manage and compose/use the previous types of services.

8 Conclusion

In this paper, we performed a comparative analysis of service identification approaches in academia and industry. We derived several recommendations for the inputs, process and outputs that a service identification approach should have. Based on these recommendations we proposed *ServiceMiner*, a type-sensitive service identification approach for the migration of legacy software systems to SOA. We evaluated *ServiceMiner* on a real-world legacy ERP system and an inventory management system and compared its results with those of three state-of-the-art SI approaches. We showed that, in general, *ServiceMiner* identified relevant, architecturally-significant services with 77.9% of precision, 63.7% of recall, and 71% of F-measure. Also, we showed that it outperformed the state-of-the-art SI approaches by providing more architecturally-significant services. We believe that *ServiceMiner* can thus be used to assist practitioners in the identification of candidate services in their systems.

As future work, we will consider the identification of other types of services, such as Enterprise and Business services. We will also perform a qualitative validation of the significance and relevance of the identified services with developers. We will use an optimization algorithm to improve the clustering process and compare other algorithms to further study the reliability of our approach. Finally, we will complete our SOA migration road-map by exploring automatic service packaging techniques to efficiently package and integrate the identified services into the targeted SOA platform.

Acknowledgements

References

[1] G. Lewis, E. Morris, L. O'Brien, D. Smith, and L. Wrage, "Smart: The service-oriented migration and reuse technique," DTIC Document, Tech. Rep., 2005.

[2] T. Erl, *SOA Principles of Service Design*. NJ, USA: Prentice Hall PTR, 2007.

[3] R. Khadka, A. Saeidi, S. Jansen, and J. Hage, "A structured legacy to soa migration process and its evaluation in practice," in *MESOCA*, 2013, pp. 2–11.

[4] B. Bani-Ismail and Y. Baghdadi, "A literature review on service identification challenges in service oriented architecture," in *International Conference on Knowledge Management in Organizations*. Springer, 2018, pp. 203–214.

[5] G. Canfora, A. R. Fasolino, G. Frattolillo, and P. Tramontana, "Migrating interactive legacy systems to web services," in *CSMR*, 2006, p. 10.

[6] M. Gysel, L. Kölbener, W. Giersche, and O. Zimmermann, "Service cutter: A systematic approach to service decomposition," in *ESOCC*, 2016, pp. 185–200.

[7] R. Rodríguez-Echeverría, F. Maclas, V. M. Pavón, J. M. Conejero, and F. Sánchez-Figueroa, "Generating a rest service layer from a legacy system," in *Information System Development*. Springer, 2014, pp. 433–444.

[8] H. Jain, H. Zhao, and N. R. Chinta, "A spanning tree based approach to identifying web services," *International Journal of Web Services Research*, vol. 1, no. 1, p. 1, 2004.

[9] S. Adjoyan, A. Seriai, and A. Shatnawi, "Service identification based on quality metrics object-oriented legacy system migration towards SOA," in *SEKE*, 2014, pp. 1–6.

[10] M. J. Amiri, S. Parsa, and A. M. Lajevardi, "Multifaceted service identification: Process, requirement and data," *ComSIS*, pp. 335–358, 2016.

[11] Z. Zhang, H. Yang, and W. C. Chu, "Extracting reusable object-oriented legacy code segments with combined formal concept analysis and slicing techniques for service integration," in *QRS*, 2006, pp. 385–392.

[12] M. Abdellatif, G. Hecht, H. Mili, G. Elboussaidi, N. Moha, A. Shatnawi, J. Privat, and Y.-G. Guéhéneuc, "State of the practice in service identification for soa migration in industry," in *ICSOC*. Springer, 2018, pp. 634–650.

[13] M. Abdellatif, A. Shatnawi, H. Mili, N. Moha, G. El-Boussaidi, G. Hecht, J. Privat, and Y. Guéhéneuc, "A taxonomy of service identification approaches for legacy software systems modernization," *J. Syst. Softw.*, vol. 173, p. 110868, 2021. [Online]. Available: https://doi.org/10.1016/j.jss.2020.110868

[14] M. Abdellatif, R. Tighilt, N. Moha, H. Mili, G. El-Boussaidi, J. Privat, and Y. Guéhéneuc, "A type-sensitive service identification approach for legacy-to-soa migration," in *Service-Oriented Computing - 18th International Conference, ICSOC 2020, Dubai, United Arab Emirates, December 14-17, 2020, Proceedings*, ser. Lecture Notes in Computer Science, E. Kafeza, B. Benatallah, F. Martinelli, H. Hacid, A. Bouguettaya, and H. Motahari, Eds., vol. 12571. Springer, 2020, pp. 476–491. [Online]. Available: https://doi.org/10.1007/978-3-030-65310-1_34

[15] G. Mazlami, J. Cito, and P. Leitner, "Extraction of microservices from monolithic software architectures," in *2017 IEEE International Conference on Web Services (ICWS)*. IEEE, 2017, pp. 524–531.

[16] S. Alahmari, E. Zaluska, and D. De Roure, "A service identification framework for legacy system migration into soa," in *SCC*. IEEE, 2010, pp. 614–617.

[17] A. Marchetto and F. Ricca, "From objects to services: toward a stepwise migration approach for java applications," *STTT*, vol. 11, no. 6, p. 427, 2009.

[18] A. Fuhr, T. Horn, and V. Riediger, "Using dynamic analysis and clustering for implementing services by reusing legacy code," in *WCRE*. IEEE, 2011, pp. 275–279.

[19] R. S. Huergo, P. F. Pires, and F. C. Delicato, "Mdcsim: A method and a tool to identify services," *IT Convergence Practice*, vol. 2, no. 4, pp. 1–27, 2014.

[20] M. Grieger, S. Sauer, and M. Klenke, "Architectural restructuring by semi-automatic clustering to facilitate migration towards a service-oriented architecture," *Softwaretechnik-Trends*, vol. 34, no. 2, 2014.

[21] E. Souza, A. Moreira, and C. De Faveri, "An approach to align business and it perspectives during the soa services identification," in *ICCSA*, 2017, pp. 1–7.

[22] H. M. Sneed, C. Verhoef, and S. H. Sneed, "Reusing existing object-oriented code as web services in a soa," in *MESOCA*. IEEE, 2013, pp. 31–39.

[23] R. S. Huergo, P. F. Pires, and F. C. Delicato, "A method to identify services using master data and artifact-centric modeling approach," in *ACM SAC*, 2014, pp. 1225–1230.

[24] A. Selmadji, A.-D. Seriai, H. L. Bouziane, R. O. Mahamane, P. Zaragoza, and C. Dony, "From monolithic architecture style to microservice one based on a semi-automatic approach," in *ICSA*. IEEE, 2020, pp. 157–168.

[25] I. Saidani, A. Ouni, M. W. Mkaouer, and A. Saied, "Towards automated microservices extraction using muti-objective evolutionary search," in *International Conference on Service-Oriented Computing*. Springer, 2019, pp. 58–63.

[26] M. Djeloul, "Locating services in legacy software:information retrieval techniques, ontology and fca based approach," *WSEAS Trans. Comput. (Greece)*, 2012.

[27] G. Chenghao, W. Min, and Z. Xiaoming, "A wrapping approach and tool for migrating legacy components to web services," in *ICNDC*, 2010, pp. 94–98.

[28] B. Kitchenham, "Procedures for performing systematic reviews," *Keele, UK, Keele University*, vol. 33, no. 2004, pp. 1–26, 2004.
[29] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *Proceedings of the 18th international conference on evaluation and assessment in software engineering*. ACM, 2014, p. 38.
[30] K. R. Felizardo, E. Mendes, M. Kalinowski, É. F. Souza, and N. L. Vijaykumar, "Using forward snowballing to update systematic reviews in software engineering," in *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. ACM, 2016, p. 53.
[31] K. Charmaz and L. Belgrave, "Qualitative interviewing and grounded theory analysis," *The SAGE handbook of interview research*, pp. 347–365, 2012.
[32] H. M. Sneed, "Integrating legacy software into a service oriented architecture," in *Software Maintenance and Reengineering, 2006. CSMR 2006. Proceedings of the 10th European Conference on*. IEEE, 2006, pp. 11–pp.
[33] M. Balabanović and Y. Shoham, "Fab: content-based, collaborative recommendation," *Communications of the ACM*, vol. 40, no. 3, pp. 66–72, 1997.
[34] G. Birkhoff, *Lattice theory*. American Mathematical Soc., 1940, vol. 25.
[35] G. Gratzer, *Lattice theory: First concepts and distributive lattices*. Courier Corporation, 2009.
[36] Rui Xu and D. Wunsch, "Survey of clustering algorithms," *IEEE Transactions on Neural Networks*, vol. 16, no. 3, pp. 645–678, May 2005.
[37] J. Rubin and M. Chechik, "A survey of feature location techniques," in *Domain Engineering*. Springer, 2013, pp. 29–58.
[38] L. Aversano, L. Cerulo, and C. Palumbo, "Mining candidate web services from legacy code," in *10th International Symposium on Web Site Evolution*. IEEE, 2008, pp. 37–40.
[39] Y. Baghdadi, "Reverse engineering relational databases to identify and specify basic web services with respect to service oriented computing," *Information systems frontiers*, vol. 8, no. 5, pp. 395–410, 2006.
[40] Z. Zhang and H. Yang, "Incubating services in legacy systems for architectural migration," in *11th Asia-Pacific Software Engineering Conference, 2004.* IEEE, 2004, pp. 196–203.
[41] H. Sneed, "Migrating to web services: A research framework," in *Proceedings of the International*, 2007.
[42] Z. Zhang, R. Liu, and H. Yang, "Service identification and packaging in service oriented reengineering." in *SEKE*, vol. 5, 2005, pp. 620–625.
[43] L. Bao, C. Yin, W. He, J. Ge, and P. Chen, "Extracting reusable services from legacy object-oriented systems," in *software maintenance (ICSM), 2010 IEEE International Conference on*. IEEE, 2010, pp. 1–5.
[44] A. A. C. De Alwis, A. Barros, A. Polyvyanyy, and C. Fidge, "Function-splitting heuristics for discovery of microservices in enterprise systems," in *International Conference on Service-Oriented Computing*. Springer, 2018, pp. 37–53.
[45] A. A. C. De Alwis, A. Barros, C. Fidge, and A. Polyvyanyy, "Discovering microservices in enterprise systems using a business object containment heuristic," in *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer, 2018, pp. 60–79.
[46] A. Shatnawi, H. Shatnawi, M. A. Saied, Z. A. Shara, H. Sahraoui, and A. Seriai, "Identifying software components from object-oriented apis based on dynamic analysis," in *Proceedings of the 26th Conference on Program Comprehension*. ACM, 2018, pp. 189–199.
[47] S. Cohen, "Ontology and taxonomy of services in a service-oriented architecture," *The Architecture Journal*, vol. 11, no. 11, pp. 30–35, 2007.
[48] OpenGroup, "The open group soa reference architecture," [Online; accessed 1-June-2020].
[49] G. E. Boussaidi, A. B. Belle, S. Vaucher, and H. Mili, "Reconstructing architectural views from legacy systems," in *WCRE*, 2012.
[50] H. Bruneliere, J. Cabot, G. Dupé, and F. Madiot, "Modisco: A model driven reverse engineering framework," *IST*, vol. 56, no. 8, pp. 1012–1032, 2014.
[51] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematical society*, vol. 7, no. 1, pp. 48–50, 1956.
[52] Z. Wen and V. Tzerpos, "An effectiveness measure for software clustering algorithms," in *Proceedings. 12th IEEE International Workshop on Program Comprehension, 2004.* IEEE, 2004, pp. 194–203.
[53] O. Maqbool and H. Babri, "Hierarchical clustering for software architecture recovery," *IEEE Transactions on Software Engineering*, vol. 33, no. 11, pp. 759–780, 2007.
[54] J. Garcia, I. Ivkovic, and N. Medvidovic, "A comparative analysis of software architecture recovery techniques," in *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering*. IEEE Press, 2013, pp. 486–496.
[55] M. E. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Physical review E*, vol. 69, no. 2, p. 026113, 2004.
[56] U. N. Raghavan, R. Albert, and S. Kumara, "Near linear time algorithm to detect community structures in large-scale networks," *Physical review E*, vol. 76, no. 3, p. 036106, 2007.
[57] S. Newman, *Building microservices: designing fine-grained systems*. " O'Reilly Media, Inc.", 2015.

aaaa