Using Dynamic Analysis and Clustering for Implementing Services by Reusing Legacy Code

Andreas Fuhr, Tassilo Horn, Volker Riediger
Institute for Software Technology
University of Koblenz-Landau
Koblenz, Germany
Email: {afuhr,horn,riediger}@uni-koblenz.de

***Abstract*—Migrating legacy systems towards Service-Oriented Architectures requires the identification of legacy code that is able to implement the new services. This paper proposes an approach combining dynamic analysis and data mining techniques to map legacy code to business processes and to identify code for service implementations based on this mapping. Validating the clustering solution in a first case study resulted in values of 70,6% in precision and 83,5% in recall.**

I. Motivation

According to a study in 2010 [1], the greatest future challenge for companies is *managing complexity*. Markets are becoming more volatile, more uncertain and more complex, forcing companies to react to market changes rapidly. CEOs come up against these threats by enabling their companies to quickly react to these changes. In addition, the company's software systems must be enabled to catch up to the fast changes, too.

In recent years, Service-Oriented Architectures (SOAs) have been emerging as one possible software paradigm providing flexibility for fast software changes. SOAs are viewed as an abstract, business-driven approach decomposing software into loosely-coupled *services* enabling the reuse of existing software assets for rapidly changing business needs [2]. A service is an encapsulated, reusable and business-aligned asset with a well-defined *service specification* providing an interface description of the requested functionality. The service specification is implemented by a service component which is provided by a *service provider*. Its functionality is used by *service consumers* [3].

To keep costs low for switching to SOAs, companies aim at reusing their existing systems ("legacy systems") as much as possible. Migrating legacy systems – that is, transferring software systems to a new environment without changing the functionality [4] – enables already proven applications to stay on stream instead of passing away after some suspensive servicing [5]. Migrating to services enables both, the reuse of already established and proven software components and the integration with new services to support changing business needs.

One key issue in migrating legacy systems towards SOAs is the identification of legacy code that is able to implement the services. Functionality that should be provided by one service may be scattered across the legacy system. Hence, it is difficult to identify the pieces of code that belong to one service. Matters are complicated further by the fact that legacy code is often not documented and identifiers are often not named appropriately. Therefore, techniques are needed to identify legacy code that is able to provide the functionality of services, without relying on any naming convention or documentation.

A. Contribution of the Paper

This paper introduces an approach to identify legacy code for service implementation by using dynamic analysis and clustering techniques. We exploit a mapping of legacy code to business processes established during dynamic analysis, to identify clusters of classes able to implement services.

The remainder of this paper is organized as follows. Section II describes how we mapped legacy code to business processes by using dynamic analysis. Section III introduces clustering techniques leveraging this mapping to identify legacy code able to implement services. Section IV reports on the application of the approach to a first proof-of-concept case study. Section VI summarizes the paper and provides an outlook on open research issues.

II. Gathering Data: Dynamic Analysis

In our approach, we aimed at mapping legacy code to business processes. As precondition, we therefore need a business process model of the customer's business. This model might already exist from previous business engineering activities. However, in most cases, such a model does not exist and must be created at the beginning of the migration project.

For identifying legacy code that is executed during a business process, we used the dynamic analysis environment shown in Fig. 1 [6].

For mapping legacy code to business processes, the business process steps (called activities) were recorded (1) and simulated on the legacy system which had been instrumented to log all code execution (2). Both pieces of information were stored

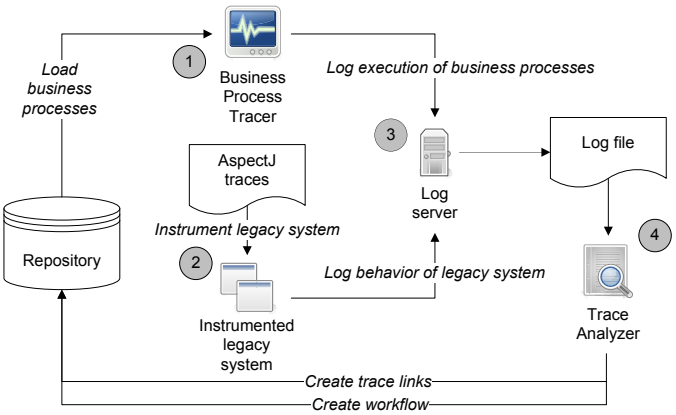

Fig. 1. Dynamic analysis set-up

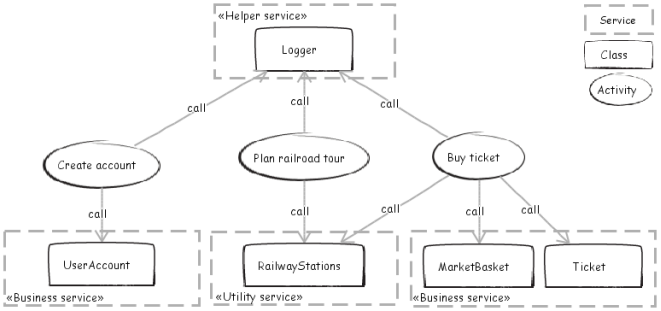

Fig. 2. Service types and usage of code in activities

synchronously to a log server (3). The information was post-processed (4) and was written back to an integrated repository storing models for business processes, code and the mapping.

a) The integrated repository: To connect business processes and code, both pieces of information are stored as models in an integrated repository. Entities of both models are connected via a tracing model. All models conform to specialized metamodels which together form the overall repository metamodel.

b) The Business Process Tracer: The Business Process Tracer is a graphic tool reading the modeled business processes from the repository and visualizing them as UML 2.0 activity diagrams.

During execution, a user operates the tracer in parallel to the legacy system. In the tracer, an activity is selected and activated and then the user requests the functionality from the legacy system by using its user interface.

Whenever an activity is activated or deactivated in the tracer, a message containing a timestamp and that activity's name is sent to the log server.

c) The instrumented legacy system: The legacy system is instrumented (for example using aspect-oriented approaches). Before and after each code execution (for example a method invocation or object instantiation), a `BEGIN` or `END` message is sent to the Log Server. This message contains a timestamp, the name of the current thread and an identifier of the code that has been invoked.

d) The log server: The log server receives the messages sent from the Business Process Tracer and the instrumented legacy system and stores them in log files. It provides a synchronization mechanism which guarantees the correct order of the logged messages, even though the legacy system and the tracer may be running on different machines with clocks ticking asynchronously.

e) The Trace Analyzer: So far, all components of the dynamic analysis framework dealt with recording the sequence of activity (de)activations in an executed business process and the corresponding low-level actions which were executed in the legacy system in response to that.

The Trace Analyzer's job is to identify the correct model entities corresponding to the identifier names in the log files and to connect the entities via trace links.

It iterates over the log file entries. From the `begin/end`-activity entries, it determines the activity in the business process model which was executed at that time. Between the `begin` and `end` entry for an activity, there are code calls, which were executed in the legacy system while performing this activity. Therefore, a mapping is created connecting the activity in the business process model to the code in the legacy source code model.

The following section describes how this mapping was processed by clustering techniques to identify legacy code able to implement services.

III. Identifying Legacy Code for Service Implementation: Clustering Techniques

Clustering techniques divide heterogeneous data into more homogeneous subgroups so that items in the same cluster are "similar". For computing how "similar" items are, similarity or dissimilarity measures like the Euclidean distance are used.

Section III-A introduces our core concept of clustering legacy code according to its usage in business processes. Section III-B briefly introduces which clustering algorithms we used and Section III-C describes how we identified the best fitting clustering solution. Section IV will then present the results of this approach applied to a first case study.

A. Core Concepts

By using clustering techniques, we aimed at grouping legacy code (at the class-level[1]) such that one cluster contains classes that implement one service. We distinguished three types of services [7]:

1) business services,
2) utility services and
3) helper services.

Fig. 2 sketches the relation between legacy code usage during activities and the type of services.

[1]In this first proof-of-concept, we grouped legacy code at the level of classes. However, other levels of granularity (like method-level) could be used, too.

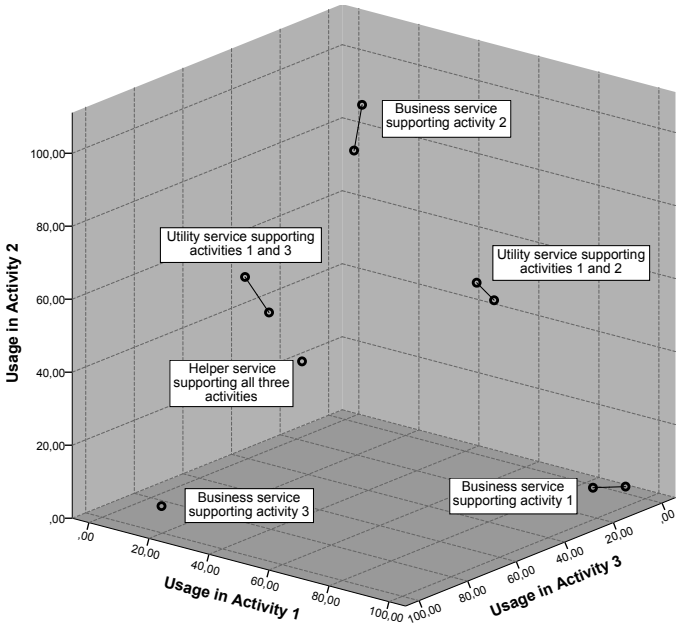

Fig. 3. Clustering of classes by usage in activities

Business services implement specialized business functionality of an activity. Therefore, they are used almost exclusively by one activity. As a consequence, legacy code that is used in only one activity may be suited to implement a business service supporting that activity.

Utility services provide specialized functionality used by more than one service. Legacy code that can be reused to implement utility services is characterized in a way that it is used during multiple, but not most of the activities.

Helper services provide general functionality that is used by most of the other services. Legacy code for implementing helper services is used during almost all activities.

In order to identify these three types of services, we looked at how often each legacy class was used during each activity of the business processes. Based on how often code was used during the activities, clustering techniques were used to identify groups of code forming one of the three service types.

Fig. 3 exemplifies, how clustering was used to identify the three types of services. The figure shows a clustering of classes that have been used in three activities. Classes that have been used almost exclusively in one of the activities form business services. Classes that have been used in two of the activities form utility services. And the class that has been used equally in all three activities forms a helper service.

B. Clustering Algorithms

In our approach, we used the TwoStep algorithm to approximate the number of clusters and the k-means algorithm to compute additional clustering solutions with given numbers of clusters.

The TwoStep algorithm [8] is a hierarchical clustering algorithm based on the Log-Likelihood distance measure. The algorithm provides capabilities to approximate the best number of clusters. The range to look for the optimal number of clusters was considered to be around the number of activities. The k-means algorithm [9] is a partitioning clustering algorithm using the Euclidean distance measure. It divides a dataset into a given number of clusters. The number of clusters was approximated by the TwoStep algorithm, first.

We used both algorithms to calculate multiple clustering solutions varying in number of clusters and training parameters. The most challenging part was then to identify the "right" clustering solution.

C. Identifying the Best Fitting Clustering Solution

To narrow down the clustering solutions, we filtered the solutions by their *Silhouette Coefficient*. The Silhouette Coefficient combines measures for the cohesion (intra-cluster distances) and the coupling (inter-cluster distances) of clusters. Ideal clustering solutions should provide high cohesion (all items in a cluster strongly belong together) and low coupling (items in other clusters do not belong to the given cluster). The *Silhouette Coefficient* of a clustering solution is near to 1 if all clusters have high cohesion and low coupling. According to [10], values below 0.51 potentially indicate an artificial solution. Therefore, clustering solutions with a silhouette coefficient below 0.51 were removed from the solution space.

The remaining clustering solutions were interpreted manually. Each cluster was analyzed to identify which activities where supported by the legacy classes belonging to the cluster. A meaning like "This cluster represents a business service supporting the *LoadGraph* activity" was given to each cluster. The most intuitive solution (a reasonable but subjective decision during the interpretation process) was selected as final clustering solution.

IV. Case Study: SoamigExtractor

The identification of legacy code for service implementation by dynamic analysis and clustering techniques has been evaluated on one business process of the SOAMIG[2] project: the manual extraction of legacy code in order to implement a service. The *CodeExtraction* business process (Fig. 4) was re-documented by interviewing project members and was modeled as BPMN activity diagram. The business process contains the five activities (that is, steps of the business process) (1) *Load graph*, (2) *Select elements*, (3) *Refactor elements*, (4) *Extract elements* and (5) *Save graph*.

These activities are supported by the *SoamigExtractor* tool. The SoamigExtractor is a Java/Swing-based tool for the manual extraction of legacy code in order to form new modules. The tool supports the visualization of package and inheritance structure of a given legacy system, the selection of elements and their dependent code (using slicing techniques), some refactorings and the extraction of the selected elements.

[2]The SOAMIG project addressed the semi-automatic migration of legacy software systems to Service-Oriented Architectures, based on model-driven techniques and code transformation. See http://www.soamig.de for further information.

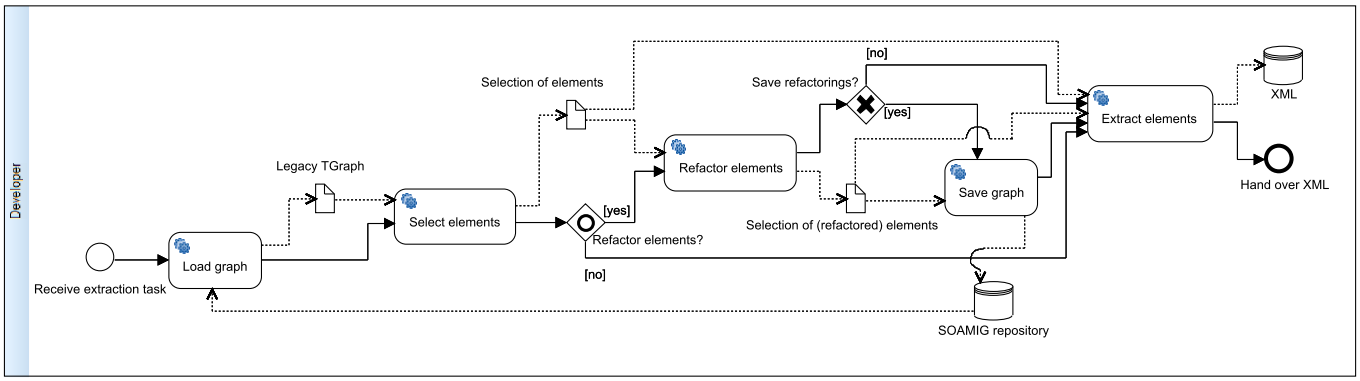

Fig. 4. BPMN model for the *CodeExtraction* business process

A. Rating the clustering solution

To rate the quality of the clustering solutions, an expert (the original developer of the system) allocated the classes of the SoamigExtractor to services manually. This "true allocation" was defined as the reference allocation that the clustering solution should match. The expert defined seven services for the SoamigExtractor: 1) *LoadGraph*, 2) *SaveGraph*, 3) *ExtractXml*, 4) *SelectCode*, 5) *RefactorCode*, 6) *DataPersistence* and 7) *Visualization*

Given the true allocation, the quality of a clustering solution was measured by computing a contingency table for clustering solutions: Let A be the true allocation of the data and C be the solution computed by a clustering technique [11]. Then, (i) N_{11} is the number of pairs of items which are in the same cluster in C and in the same cluster in A. (ii) N_{10} is the number of pairs of items which are in the same cluster in C but in different clusters in A. (iii) N_{01} is the number of pairs of items which are in different clusters in C but in the same cluster in A. (iv) N_{00} is the number of pairs of items which are in different clusters in C and in different clusters in A. The *precision* of a clustering solution is then defined as the ratio of items a clustering technique has computed right [12]:

$$P = \frac{N_{11}}{N_{11} + N_{10}} \tag{1}$$

The *recall* is defined as the ratio of items the clustering technique matched against the items expected by the true allocation:

$$R = \frac{N_{11}}{N_{11} + N_{01}} \tag{2}$$

In the following, two clustering solutions are evaluated and compared. Section IV-B describes a clustering solution based only on the usage of legacy classes during activities. Section IV-C presents an adapted solution based on a dataset including information about the classes' package.

B. First Results: Clustering on Usage Only

For the first approach, the TwoStep algorithm approximated 6 clusters (services) as the optimal number, matching the expert's intuition quite well. To find the best clustering solution, the parameters of the TwoStep algorithm have been varied. Various k-Means solutions have been computed, too. The solutions differed in the number of clusters (6 ± 2). This first approach resulted in three clustering solutions meeting the threshold criterion of a Silhouette Coefficient higher than 0.51. The three solutions were interpreted manually and the most intuitive solution was chosen.

The final clustering solution (k-Means with 6 clusters) reached a Silhouette coefficient of 0.883, attesting a strong, non-artificial, clustering solution. The six clusters were interpreted as five business services supporting the five activities and one utility service supporting the activities *Select elements*, *Refactor elements* and *Load graph* .

Comparing this solution to the true allocation defined by the expert, the clustering by usage values only resulted in a precision of 62,9% and a recall of 80,3%. Further analysis revealed, that most of the classes of the *DataPersistence* service were used almost exclusively in one of the activities. Therefore, they are put correctly into the service supporting the respective activity.

To force the clustering algorithms to put classes of the *DataPersistence* service into an own cluster, an additional parameter was added to the input: the qualified name of the package of the classes.

C. Adapted Results: Including Package Information

A clustering with additional information about the package of a legacy class resulted in 70,6% precision and 83,5% recall. Fig. 5 visualizes the distribution of the classes to the clusters. The classes implementing the service *DataPersistence* and *Visualization* were put into one cluster and three classes of the *DataPersistence* classes were associated to the *SelectCode* cluster. All other classes are allocated to clusters perfectly matching the expert's true allocation.

V. Related Work

Identification of legacy code able to implement services is still an open research issue and not yet explored very well [13]. However, some inspiring work has been done in disciplines relating to our approach.

In the context of research on feature location, some approaches identifying legacy code implementing features

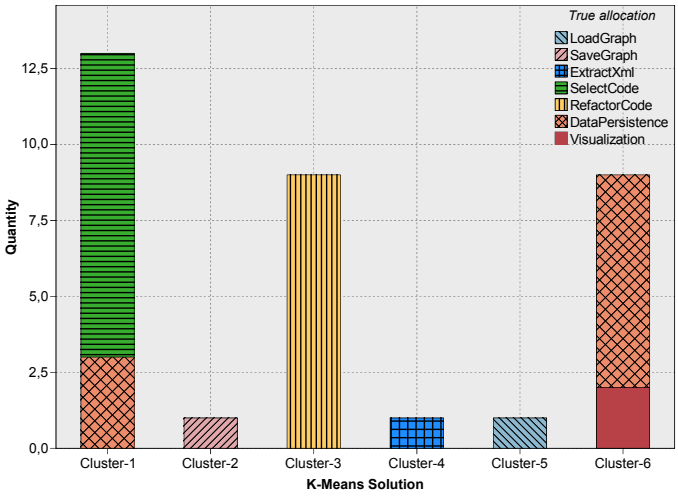

Fig. 5. Distribiution of classes to the clusters

(which are similar to services) have been developed [14], [15]. These approaches were quite inspiring for our work. However, they are missing the focus on business processes as the basis for dynamic analysis scenarios.

In research on software clustering, much work has been done in developing various clustering techniques and tools supporting the extraction of system parts (see for example [16]). However, the approaches do not account for the importance of business processes in the implementation of services, too.

In the context of identifying legacy code able to implement services, some work has been done, too. IBM analyzes names of operations and comment lines in order to find services [17]. They use information retrieval techniques to extract information from names and comments. Therefore, the approach fails as soon as naming conventions are not met or code is not documented well. Zhang et al. [18] identify legacy code able to implement services based on the recovered architecture of a legacy system. Marchetto and Ricca [19] use dynamic analysis for identifying legacy code able to implement services, too. However, they use test cases instead of business processes for their dynamic analysis scenario and they do not use clustering techniques to post-process the dynamic trace logs.

VI. Conclusion

In this paper, we presented an approach exploiting the mapping of legacy code to activities of a business process. The mapping was derived by dynamic analysis. We clustered legacy classes according to their usage during activities, forming clusters that gave hints which classes can be reused to implement services.

The clustering without including information about packages already lead to satisfying results. As advantage, this approach is independent of any naming conventions or semantics that have been tried to extract from the legacy code. Including package information in the clustering process lead to results better matching the expert's intuition. However, this approach relies on well-defined package structures.

In future research, this approach must be verified on enterprise-scale systems. In more complex systems, it will be much harder to identify well-interpretable clusters. In addition, estimating the right number of clusters was lead by subjective intuition; a well-defined technique to identify this number must be invented in future projects. Moreover, the approach must be verified on other legacy languages and on a different level of granularity (that is, for example, clustering on method level instead of class level).

However, the results of this first proof-of-concept shows a promising approach for identifying legacy code that is able to implement services.

References

[1] IBM, "Capitalizing on complexity: Insights from the global chief executive officer study," 2010.

[2] N. Gold, C. Knight, A. Mohan, and M. Munro, "Understanding service-oriented software," *IEEE Softw.*, vol. 21, no. 2, pp. 71–77, 2004.

[3] A. Arsanjani, S. Ghosh, A. Allam, T. Abdollah, S. Ganapathy, and K. Holley, "Soma: A method for developing service-oriented solutions," *IBM Syst J*, vol. 47, no. 3, pp. 377–396, 2008.

[4] H. M. Sneed, E. Wolf, and H. Heilmann, *Softwaremigration in der Praxis: Übertragung alter Softwaresysteme in eine moderne Umgebung*, 1st ed. Heidelberg: dpunkt.Verl., 2010.

[5] V. T. Rajlich and K. H. Bennett, "A staged model for the software life cycle," *IEEE Comput*, vol. 33, no. 7, pp. 66–71, 2000.

[6] A. Fuhr, T. Horn, and V. Riediger, "Dynamic analysis for model integration (extended abstract)," *ST-Trends*, vol. 30, no. 2, pp. 70–71, 2010.

[7] S. Alahmari, E. Zaluska, and D. De Roure, "A service identification framework for legacy system migration into soa," in *IEEE Seventh International Conference on Services Computing*. IEEE Computer Society, 2010, pp. 614–617.

[8] T. Chiu, D. Fang, J. Chen, Y. Wang, and C. Jeris, "A robust and scalable clustering algorithm for mixed type attributes in large database environment," in *KDD*, 2001, pp. 263–268.

[9] J. A. Hartigan and M. A. Wong, "A k-means clustering algorithm," *Appl Stat-J Roy St C*, vol. 28, pp. 100–108, 1979.

[10] L. Kaufman and P. J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley-Interscience, 2009.

[11] G. Gan, C. Ma, and J. Wu, *Data clustering: Theory, algorithms, and applications*. Philadelphia: SIAM, 2007.

[12] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to data mining*, 1st ed. Boston: Pearson Addison Wesley, 2005.

[13] K. Kontogiannis, G. A. Lewis, D. B. Smith, M. Litoiu, H. Müller, S. Schuster, and E. Stroulia, "The landscape of service-oriented systems: A research perspective," in *Proceedings of the International Workshop on Systems Development in SOA Environments*. IEEE Computer Society, 2007.

[14] O. Greevy, S. Ducasse, and T. Gîrba, "Analyzing software evolution through feature views," *J Softw Maint Evol-R*, vol. 18, no. 6, pp. 425–456, 2006.

[15] T. Eisenbarth, R. Koschke, and D. Simon, "Locating features in source code," *IEEE Trans. Software Eng.*, vol. 29, no. 3, pp. 210–224, 2003.

[16] B. Mitchell and S. Mancoridis, "On the automatic modularization of software systems using the bunch tool," *IEEE Trans. Software Eng.*, vol. 32, no. 3, pp. 193–208, 2006.

[17] I. Ronen, N. Aizenbud, and K. Kveler, "Service identification in legacy code using structured and unstructured analysis," presented at the IBM Programming Languages and Development Environments Seminar, Haifa, 2007.

[18] Z. Zhang, R. Liu, and H. Yang, "Service identification and packaging in service oriented reengineering," in *Proceedings of the 7th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, 2005, pp. 241–249.

[19] A. Marchetto and F. Ricca, "From objects to services: toward a stepwise migration approach for java applications," *Int J Softw Tools Technol Transfer*, vol. 11, no. 6, pp. 427–440, 2009.