Discovering Microservices in Enterprise Systems Using a Business Object Containment Heuristic

Adambarage Anuruddha Chathuranga De Alwis[1], Alistair Barros[1], Colin Fidge[1], and Artem Polyvyanyy[2]

[1] Queensland University of Technology, Brisbane, Australia
{adambarage.dealwis,alistair.barros,c.fidge}@qut.edu.au
[2] The University of Melbourne, Parkville, VIC, 3010, Australia
artem.polyvyanyy@unimelb.edu.au

Abstract. The growing impact of IoT and Blockchain platforms on business applications has increased interest in leveraging large enterprise systems as Cloud-enabled microservices. However, large and monolithic enterprise systems are unsuitable for flexible integration with such platforms. This paper presents a technique to support the re-engineering of an enterprise system based on the fundamental mechanisms for structuring its architecture, i.e., business objects managed by software functions and their relationships which influence business object interactions via the functions. The technique relies on a heuristic for deriving business object exclusive containment relationships based on analysis of source code and system logs. Furthermore, the paper provides an analysis of distributing enterprise systems based on the business object containment relationships using the NSGA II software clustering and optimization technique. The heuristics and the software clustering and optimization techniques have been validated against two open-source enterprise systems: SugarCRM and ChurchCRM. The experiments demonstrate that the proposed approach can identify microservice designs which support multiple desired microservice characteristics, such as high cohesion, low coupling, high scalability, high availability, and processing efficiency.

Keywords: Microservice discovery, system reengineering, cloud migration

1 Introduction

Microservices have been introduced to the software industry as the latest form of service-based software, allowing systems to be distributed through the Cloud as fine-grained components, containing individual operations, in contrast to services under a Service-Oriented Architecture (SOA) which includes all logically related operations [1]. Such loosely coupled, highly cohesive composition of the microservices enables scaling up and replication of specific parts of systems and business processes through the Cloud, and allows them to be flexibly composed in Web, mobile computing, and Internet-of-Things (IoT) applications.

Such benefits have led Netflix™, and now Twitter™, eBay™ and Amazon™ to develop novel architectures for software solutions as microservices. Nonetheless, microservices have so far not been adopted for the dominant form of software in businesses, namely enterprise systems, limiting such systems' evolution and their exploitation of the

full benefits of cloud-enabled platforms such as Google Cloud and Amazon AWS. In particular, this limits the evolution of business applications involving IoT and blockchain [2].

Enterprise systems, such as enterprise resource planning (ERP), customer relationship management (CRM) and supply chain management (SCM), are large and complex and contain complex business processes encoded in application logic managing business objects, in typically many-to-many relationships [3]. Restructuring enterprise systems as microservices is technically cumbersome. It requires tedious search and identification of suitable parts of the system to restructure, program code rewrites, and the integration of the newly developed microservices with the remaining 'backend' enterprise systems. This is a costly and error-prone task for developers, mainly due to two reasons. Firstly, it is difficult to identify the functional dependencies between different functions and operations by examining millions of lines of code in the system. Secondly, it is challenging to figure out an optimal splitting of the system functionalities as fine-grained microservices while minimizing the communication cost (i.e. service calls) between them in order to provide better availability and processing efficiency.

Automated software re-engineering techniques have been proposed to improve the efficiency of transforming legacy applications, structures [4], specifically focusing on optimizing cohesion and coupling of software packages and components using static analysis techniques that focus on source code [5] and dynamic analysis techniques that focus on software execution patterns and system logs [6]. However, these techniques have, to date, not been applied to the re-engineering challenges of microservices, which are more fine-grained (i.e single operational functions) and distributed compared to Service-Oriented-Architecture (SOA) services in which multiple operations are combined within a single function.

Enterprise systems provide semantic insights, available through the business objects that they manage, which influence the software structure and the processes they support. For instance, an order-to-cash process in SAP ERP is supported through functions of software components: multiple sales orders, deliveries shared across different customers, shared containers in transportation carriers, and multiple invoices and payments. To support this process, multiple functions are invoked asynchronously, reflecting business object relationship types and cardinalities, and these can be seen through cross-service interactions, correlations, and data payloads [7]. Such insights provided by business object relationships are promising for improving the feasibility of automated system re-engineering, where modules correspond to single business objects. As examples, Pẽrez-Castillo *et al.* [8] used the transitive closure of strong business object dependencies derived from databases to recommend software function hierarchies, while Lu *et al.* [9] demonstrated process discovery using SAP ERP logs based on business objects.

This paper presents discovery techniques that support the identification of suitable consumer-oriented parts of enterprise systems which could be re-engineered as microservices based on the knowledge gained through business object relationships and their execution patterns. Specifically, the paper addresses two fundamental areas of microservice discovery, namely exclusive containment of business object relationships, while analysing the cost and complexity of functional calls between different microservices based on a heuristic rule by optimizing the Non-dominated Sorting Genetic Algorithm

(NSGA) II. Experiments were conducted on SugarCRM[3] and ChurchCRM[4] to validate the presented methodology and showed that it can identify microservice designs which support multiple microservice characteristics, such as high cohesion, low coupling, high scalability, high availability, and processing efficiency while preserving coherent features of enterprise systems and minimizing the overall communication overhead of the system.

The remainder of the paper is structured as follows. Section 2 presents a containment heuristic, while Section 3 describes a microservice discovery process based on the heuristic. Section 4 discusses an implementation and validation of the proposed technique. Related work is summarized in Section 5. The paper closes with a conclusion.

2 Containment Relationships and Heuristic

Enterprise systems (ESs) use different Business Objects (BOs) to store the information related to their functionality, and these BOs have different relationships between them. For example, the creation of a 'purchase order' will result in the invocation of functions involved in the creation of 'line items' reflecting an exclusive containment of business objects. It is important to identify such relationships between BOs, since microservices (MSs) are functionally isolated and loosely-coupled parts connected to each other, much like components of a distributed system, and typically focus on individual BOs, locally managed through a database [10]. A better understanding of such BO relationships can be obtained through the following formal characterization.

Given an ES s, by OP_s, A_s, T_s, and B_s, we refer to the set of its all *operations*, *attributes*, *database tables*, and *business objects*, respectively. We will omit the subscripts where the context is clear. We define domination and exclusive containment relationships over business objects as follows.

Definition 2.1 (Domination [11] and exclusive containment [12] relationships) Given two business objects $b, b' \in B$, of an ES we say that:

- b *dominates* b' iff for every create or delete operation $op \in OP$ that either uses some attribute of b' as one of its input parameters or writes its result into some attribute of b', it holds that op uses some attribute of b as one of its input parameters or writes its result into some attribute of b;
- b' is *exclusively contained* in b iff $b' \neq b$, and b dominates b', and there exists no $b'' \in B$, $b'' \neq b$, $b'' \neq b'$, such that b'' dominates b'. ⌟

By $\gamma : B \rightarrow \mathcal{P}(B)$, we denote the function that relates every business object to the corresponding set of business objects it exclusively contains.

The behaviour of an ES, or a MS system, is based on the invocation of operations which consist of well-defined processing sequences governed by BO relationships. Such sequences illustrate a particular execution pattern based on the structure and behaviour of an organization. Therefore, we argue that a proper analysis of such process sequences and BO relationships will help to derive prominent microserviceable components. This

[3] https://www.sugarcrm.com/

[4] http://churchcrm.io/

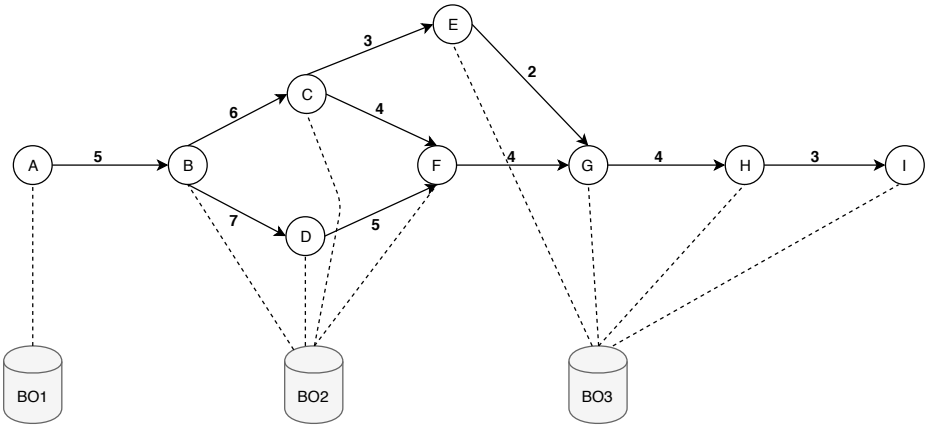

Fig. 1: A pattern of CRUD operations and BO relationships.

assumption leads us to derive an exclusive containment heuristic which assists in MS discovery that aligns with Definition 2.1. As an example, assume that an ES induces a pattern of CRUD operation dependencies depicted in Fig. 1; in the figure, each node represents a CRUD operation. These operations further relate to the BOs on which they are performed, see BO1, BO2, and BO3 in the figure. If BO2 is exclusively contained in BO3, every time an operation that involves BO2 gets executed, it also processes BO3, i.e. BO3 does not have an independent life-cycle and depends on BO2. If we decouple software corresponding to the CRUD management of different BOs the communication overhead between the software components will increase and will result in a lower level of the system availability and efficiency. This understanding leads us to Heuristic 2.2.

Heuristic 2.2 (Exclusive containment)
A MS composed of a business object $b \in B$, a non-empty set of business objects $C \subset \gamma(b)$, and a non-empty set of operations $D \subseteq OP$, each performed over at least one business object in $\{b\} \cup C$ (i.e., each operation in D either uses an attribute of some business object in $\{b\} \cup C$ as an input parameter or writes its result into an attribute of some business object in $\{b\} \cup C$), when used as part of an MS system, leads to higher levels of the system's availability and scalability, and processing efficiency. ⌟

An ES can consist of BOs which do not participate in the exclusive containment relationships with other BOs. A better understanding for a possible function splitting in the presence of such BOs can be achieved through the knowledge of execution calls and their frequencies, since the number of executions between different BOs plays a major role in achieving processing efficiency and system availability. In general, if the number of communications between microservices increases, the number of network hops between the microservices also increases, which inevitably results in lower availability of the corresponding MS system and, consequently, long waiting times [1]. Hence, a proper microservice discovery and recommendation technique should consider both

BO relationships, like exclusive containment, and execution frequencies of the operations over different BOs. Taking this understanding further, we developed an automated microservice discovery process, which is described in the next section.

3 Automated Microservice Discovery

To perform microservice discovery, we developed a five-step approach, which is illustrated in Fig. 2. Since MSs are focused on accessing operations of BOs, or partitions of BOs, in the system [13], we propose to start microservice discovery with the identification of the BOs used by a given ES, refer to step 1 in the figure. To derive the BOs, we evaluate all the SQL queries of the given ES and identify the relationships between database tables, which are then used to derive the BOs according to the approach described by Nooijen *et al.* [14]. In the second step, the operations performed by the system are analysed using static analysis techniques and classified into different categories based on their relationships and types, such as association create, association delete, create and delete. The BOs derived from step one and the operations extracted and categorised using step two are provided as the input to the third step, in which we identify the containment relationships between different BOs as described in Section 3.1.

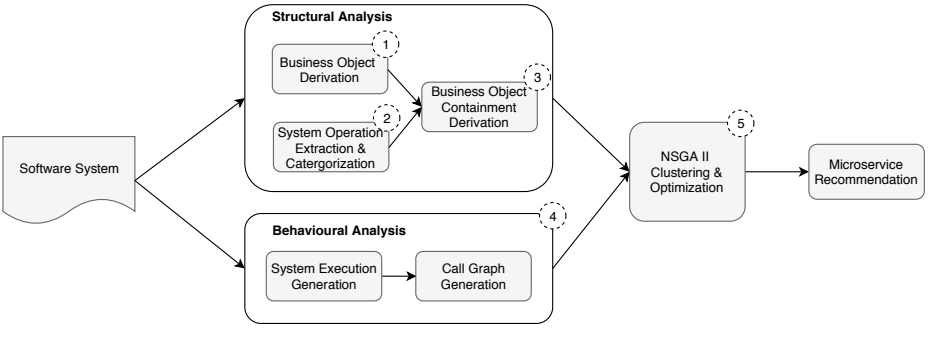

Fig. 2: Overview of the microservice discovery approach.

After performing the steps related to structural (i.e. static) analysis, behavioural (i.e. dynamic) analysis is performed as the fourth step to generate the call graphs related to the system execution. In this step, we use scripts to automate the system to generate system execution logs based on user behaviours and then use those execution logs to generate the call graphs. Finally, as the fifth step, all the structural and behavioural details generated are provided to an optimization algorithm in order to discover an optimal system splitting for MSs. The optimization algorithm analyses the BO relationships and operation calls to provide a microservice recommendation, as described in Section 3.2.

3.1 Containment Derivation

As depicted in Fig. 2, in order to derive a satisfactory splitting of system operations and BOs into MSs, the clustering and optimization algorithm needs two inputs, one from structural analysis and the other from the behavioural analysis. To perform the structural analysis to derive the BOs and the exclusive containment relationships among the BOs, we have developed Algorithm 1 which is composed of five steps.

In an ES, the information related to a BO is often stored in several database tables. Thus, one can define a BO $b \in B$ as a collection of database tables which it is scattered across, i.e. $b \subseteq T$. To identify the BOs, in the first step, the *BOS* function is performed by Algorithm 1, which derives the BOs, B, of the system through the analysis of the database table relationships and their data similarities, as described by Nooijen *et al.* [14]. In the second step of the algorithm, function *OPS* performs static analysis of the system and extracts all the operations *OP* of the system.

In the third step of Algorithm 1, the extracted operations get classified into those performed on individual database tables and those performed over several tables linked via foreign key relationships. Concretely, operations are classified into 4 categories, as association create (OP_a^c), association delete (OP_a^d), create (OP_c), and delete (OP_d) operations, refer to lines 4–17 of the algorithm; other operation types are not of interest here. At the end of the loop of lines 4–17, variables OP_c and OP_d contain the create and delete operations, respectively, that run over single database tables, while variables OP_a^c and OP_a^d contain the operations that perform create and, respectively, delete operations on multiple database tables related to multiple BOs.

In the general case, in a database, there can be foreign key relationships among the tables that relate to the same BO $b \in B$. Such relationships are not important for the exclusive containment derivation. Therefore, in Algorithm 1, filtration is done in order to identify the create and delete association operations related to multiple BOs, in lines 5–13. To achieve this filtration, first, the *TBLS* function extracts the tables T_k related to each association operation op_k. Next, after confirming that the tables in T_k do not relate to the same BO b, the algorithm adds the association operation to the respective set, either OP_a^c or OP_a^d.

In the fourth step, Algorithm 1 identifies the BOs which are related by a create (OP_a^c) or delete (OP_a^d) association operation, and stores them for further processing, in lines 18–22. Since association operations which occur within the same BO are not considered, at this step the algorithm analyses the operations in OP_a^c and OP_a^d to identify related BOs and stores them into pairs of BOs, $B_r \subseteq B \times B$.

Finally, in the fifth step, the algorithm evaluates whether any of the BOs related through an association operation and stored in B_r has its own create OP_c or delete OP_d operation (i.e. an operation performed over all the tables of the BOs), and verifies that at least one of the associated BOs is not associated with any other BOs of the system, in lines 25–26. Given two BOs $b, b' \in B$, if b' does not have its own create OP_c or delete OP_d operation, and if b' does not participate in further exclusive containment relationships with other BOs, except for the one it has with b, then we identify that b' is exclusively contained in b, as b' does not have its own independent life-cycle apart from

Algorithm 1: Computing exclusive containment relationship over BOs

Input: Source code SC and database schema DB of an ES s.

Output: A function γ that captures exclusive containment relationships over the business objects of s, and the set of all business objects B of s.

$B = \{b_1, \ldots, b_n\} := BOS(SC, DB)$; // Identify BOs

$OP = \langle op_1, \ldots, op_m \rangle := OPS(SC)$; // Identify operations

$OP_a^c := OP_a^d := OP_c := OP_d := B_r := \emptyset$;

/* Classify operations in OP */

for *each* $k \in [1 \,..\, m]$ **do**

  if op_k *is an association create operation* **then**

    $T_k := TBLS(op_k)$;

    if $\nexists\, b \in B \,.\, T_k \subseteq b$ **then**

      $OP_a^c := OP_a^c \cup \{op_k\}$; // Identify an association create operation

  else if op_k *is an association delete operation* **then**

    $T_k := TBLS(op_k)$;

    if $\nexists\, b \in B \,.\, T_k \subseteq b$ **then**

      $OP_a^d := OP_a^d \cup \{op_k\}$; // Identify an association delete operation

  else if op_k *is a create operation* **then**

    $OP_c := OP_c \cup \{op_k\}$; // Identify a create operation

  else if op_k *is a delete operation* **then**

    $OP_d := OP_d \cup \{op_k\}$; // Identify a delete operation

end

for *each* $b_i \in B$ **do**

  for *each* $b_j \in B$, $b_i \neq b_j$, **do**

    if $\exists\, op \in OP_a^c \cup OP_a^d \,.\, (b_i \cap TBLS(op) \neq \emptyset) \wedge (b_j \cap TBLS(op) \neq \emptyset)$ **then**

      $B_r := B_r \cup \{(b_i, b_j)\}$;

  end

end

for *each* $(b, b') \in B_r$ **do**

  if $(\nexists\, op \in OP_c \cup OP_d \,.\, b \cup b' \subseteq TBLS(op)) \wedge$
  $(\{b'' \in B \setminus \{b\} \mid (b', b'') \in B_r\} = \emptyset)$ **then**

    Record in γ that b' is exclusively contained in b, i.e., $b' \in \gamma(b)$;

end

return γ, B

b (i.e. b' is created and destroyed together with b). In the last line of the algorithm, the identified exclusive containment relationships γ and the BOs of the system get returned.

In addition to the identification of exclusive containment relationships over the BOs of the system through structural analysis, behavioural analysis should be performed to generate the call graphs to extract the details such as the graph nodes, execution calls between the nodes and the number of calls between the nodes. These structural data and behavioural analysis details are provided to the multi-objective optimization algorithm for further processing, as described in Section 3.2.

3.2 NSGA II Clustering and Optimization

The Non-dominated Sorting Genetic Algorithm II (NSGA II) is a multi-objective optimization algorithm which provides an optimal set of solutions while achieving global optima, when there are multiple conflicting objectives to be considered [16]. It has been evaluated as one of the best algorithms to cluster software packages and classes while achieving multiple objectives such as high cohesion and low coupling [5].

Since MS discovery should evaluate two major objectives to extract an optimal set of MSs, NSGA II was chosen to guide the extraction. The first objective is to minimize the number of communications between different nodes in the execution call graphs (i.e. the cost of calls between different MSs). The second objective is to minimize the clustering distance of operations and the BOs (i.e. the cost of clustering BOs). A better understanding of the two objectives can be achieved by referring to Fig. 1. When considering the nodes in the figure, one can decide to split the operations related to BO2 and BO3 into two groups in several ways. For example, a possible pair of groups can be 'B, C, D, F, E, G' and 'H, I', whereas 'B, C, D, F' and 'E, G, H, I' would be another such pair of groups. If two MSs get implemented based on the first splitting of the operations, then BO3 gets shared between the two MSs, which one may argue is not optimal. However, one can also argue that the splitting which ensures that each of the two BOs, i.e., BO2 and BO3, gets managed by a dedicated MS leads to a higher communication cost. In the figure, edge weights denote the numbers of calls between operations. Thus, the communication cost of the MSs according to the first splitting is equal to four, while according to the second splitting it is equal to seven.

For real system executions with thousands and millions of nodes and calls, deriving an optimal function splitting of the system with related BOs is a complicated task. Therefore, the NSGA II algorithm was modified to provide a set of optimal solutions as described in Algorithm 2.

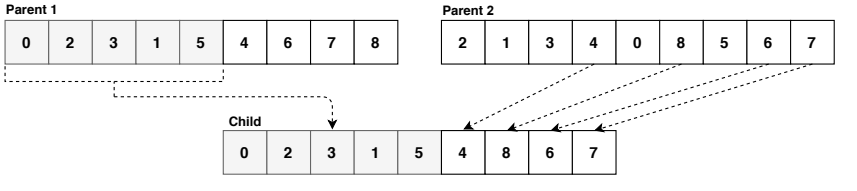

Fig. 3: Crossover to generate child population.

In order to derive optimal solutions, Algorithm 2 follows 3 steps and requires the population size (n), number of generations (Gen), chromosome length (C_Len), crossover probability (Cr_Prob) and mutation probability (Mut_Prob) as input data. Apart from the above standard parameters, our algorithm requires further input, such as the exclusively contained BO relationships (γ), BOs of the system (B) which were derived from Algorithm 1, and execution graph nodes (N) and their relationships (R) extracted from the execution graphs. In the algorithm, the population size (n) defines how many chromosomes are populated in a single generation, while the number of generations

(*Gen*) defines the number of times the algorithm generates different populations before it stops. The crossover probability (*Cr_Prob*) and mutation probability (*Mut_Prob*) define the probability for performing crossover and mutation on chromosomes. Interested readers can find further details about NSGA II in A fast and elitist multiobjective genetic algorithm: NSGA-II by Deb *et al.* [16].

In the first step of Algorithm 2, the *SYNTHESIZEPOP* function synthesizes a parent population of the given size n (see line 1). *SYNTHESIZEPOP* uses a random number generator to generate chromosomes of length *C_Len*, where a chromosome is a sequence of numbers each representing a node in the execution graph. A chromosome generated for the execution graph in Fig. 1 can be represented as a sequence of numbers '0, 1, 2, 3, 4, 5, 6, 7, 8', as depicted in Fig. 3, in which the numbers refer to the nodes in 'A, B, C, D, E, F, G, H, I', such that 0 refers to *A*, 1 refers to *B*, etc. Apart from generating the parent population, *SYNTHESIZEPOP* calculates and stores the fitness for each parent. The fitness calculation is preformed in two steps. First, the algorithm calculates the maximum cost for a chromosome as $\sum_{i=0}^{C\_Len} 2^i$. Then, the algorithm calculates the cost of clustering as the cost of BO splittings and calls between the clusters. The node clustering cost represents to which extent the nodes related to the same BO have been grouped together. The information about the relationships between the nodes and BOs can be obtained through *B* (BOs) and *N* (operation nodes), as depicted in Fig. 4. For example, using the information provided in Fig. 4, it is clear that node '4' (or node 'E' in Fig. 1) is associated with BO3, and node '5' (or node 'F' in Fig. 1) is associated with BO2; refer to the directed arcs in Fig. 4. If there is an exclusive containment relationship between two BOs, they are combined and supplied as a single BO to the algorithm. For example, if BO1 and BO2 are in the exclusive containment relationship, then *B* in Fig. 4 must be replaced with $\langle 1, 1, 1, 1, 2, 1, 2, 2, 2 \rangle$, where 1 stands for the combination of BO1 and BO2, and 2 represents BO3.

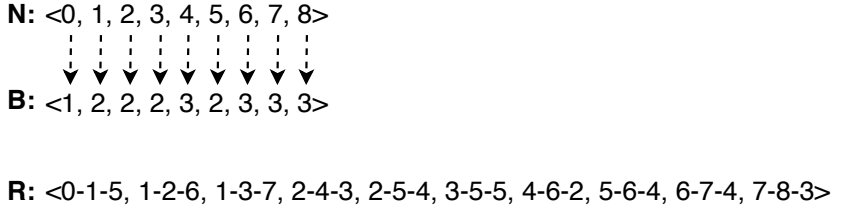

Fig. 4: Example data provided to NSGA II for the execution graph in Fig. 1.

Given the *B* and *N* information for the graph in Fig. 1, the best BO clustering would be 'A', 'B, C, D, F', 'E, G, H, I', which leads to the chromosome '0, 1, 2, 3, 5, 4, 6, 7, 8'. For each chromosome, the clustering cost is calculated as $\sum_{i=0}^{C\_Len} 2^d$, where d is the distance from the first occurrence of a node related to a particular BO to the next occurrence of the node related to the same BO within the chromosome. For example, if we calculate the clustering cost for the three clusters '0', '1, 2, 3, 5', and '4, 6, 7, 8', the first cluster contains only one node and, thus, contributes 2^0 to the clustering cost. The second cluster contains four nodes, i.e. '1, 2, 3, 5'. Node '1' is the first node that relates to BO2, which contributes the cost of 2^0, as the first node of the second cluster. However,

subsequent nodes '2', '3', and '5', which are also associated with BO2, contribute the costs of 2^1, 2^2, 2^3, respectively. Hence, if we generate the costs associated with each node '0, 1, 2, 3, 5, 4, 6, 7, 8', i.e. including the nodes that relate to BO3, these will be $2^0, 2^0, 2^1, 2^2, 2^3, 2^0, 2^1, 2^2, 2^3$, which leads the total clustering cost of '31'.

Algorithm 2: NSGA II Algorithm

Input: $n, Gen, C\_Len, Cr\_Prob, Mut\_Prob, \gamma, B, N, R$
Output: A list of optimized clustering of BOs and OPs for MSs

$Pop^p = \langle pop_1, \ldots, pop_n \rangle := SYNTHESIZEPOP(n, C\_Len, \gamma, B, N, R)$;
$Pop^c := Rank^f := \langle\rangle$;
/* Perform crossover and mutation to generate child population */
while $Pop^c.length() < n$ **do**
  if $RANDOM(0, 1) < Cr\_Prob$ **then**
    $Pop^c := CROSSOVER(Pop^p, Pop^c)$;
  if $RANDOM(0, 1) < Mut\_Prob$ **then**
    $Pop^c := MUTATION(Pop^p, Pop^c)$;
end
for *each* $i \in [1 .. Gen]$ **do**
  $Pop^t := Pop^p + Pop^c$;
  $Rank^f = \langle rank^f_1, \ldots, rank^f_m \rangle := FASTNONDOMINATEDSORT(Pop^t)$;
  if $i == Gen$ **then**
    break;
  /* Identify the Pareto front of the generated population and rank them */
  $Pop^c := \langle\rangle$;
  for $k \in [1 .. m]$ **do**
    if $rank^f_k.length() < (n - Pop^c.length())$ **then**
      $Pop^c := Pop^c + rank^f_k$;
    else
      $Pop^c := Pop^c + CROWDCOMPARISONSORT(rank^f_k)$;
  end
  $Pop^p := Pop^c$; // Initialize new parent population
  $Pop^c = \langle pop^c_1, \ldots, pop^c_n \rangle := SYNTHESIZECHILD(Pop^p)$;
end
return $(Rank^f)$

The cost of execution calls between clusters is computed as the sum of inter-cluster calls between the different clusters. For the running example, i.e. the chromosome '0, 1, 2, 3, 5, 4, 6, 7, 8', this sum would be 5 + 3 + 4 = 12, because the costs of calls between the pairs of clusters in '0', '1, 2, 3, 5', '4, 6, 7, 8', are '5', '3' and '4'. The information on the costs of the calls between the nodes is extracted from the R relation, depicted in Fig. 4. For example, the number of calls between node 'A' (i.e. node '0') and node 'B' (i.e. node '1') is 5; this is given as 0-1-5 in R, refer to Fig. 4. Note that R is constructed based on the edge weights in Fig. 1. The cost of execution calls is often by far smaller than the cost of clustering. As this may create a bias in the optimization results, in order to minimize such effects, the total cost of calls is multiplied by a fixed number y.

Afterwards, the *fitness* of the chromosome is calculated by subtracting the sum of all the cluster costs and the sum of the cost of inter-cluster calls from the maximum possible cost of the chromosome $\sum_{i=0}^{C\_Len} 2^i$.

The second step of the algorithm generates the child population by performing crossover operations and mutation operations on the parent chromosomes (see lines 3–8). In order to perform the crossover operation, the algorithm selects two parents using binary tournament selection [16]. This is performed by randomly identifying two parent chromosomes and extracting the chromosome with the highest fitness value out of them. After identifying two parent chromosomes for crossover, the algorithm splits the first parent chromosome from a predefined position (normally half of the chromosome length) and includes it as the first part of the child chromosome. In general, as the next step, a regular genetic algorithm would extract the other part from the second parent chromosome. However, in this situation, the child chromosome should not contain repeating node values. As such, the rest of the child chromosome is generated by extracting values from the other parent which are not in the first part of the child chromosome as depicted in Fig. 3. Mutation is achieved by changing the position of two genes (i.e. positions of two nodes) in the parent chromosome.

After generating the first child population, the algorithm generates *Gen* new populations, refer to lines 9–23 in Algorithm 2; which constitutes the third (and last) step of the algorithm. First, the current total population Pop^t is computed at line 10 by concatenating the parent population Pop^p and the child population Pop^c. Next, the algorithm calculates the non-dominated fronts, or the Pareto fronts, of the total population. A non-dominated front contains the chromosomes which have the optimal values for the two objectives that were defined above, namely the cost of node clustering and the cost of calls. The chromosome's optimization of node clustering is calculated as the difference between the maximum possible cost of the chromosome and cost of its node clustering. Similarly, the chromosome's optimization of execution calls is calculated as the difference between the maximum possible cost of the chromosome and cost between its cluster calls. The non-dominated chromosomes in Pop^t are extracted as the first front using function *FASTNONDOMINATEDSORT* (see line 11). After extracting the first non-dominated front, the algorithm evaluates the other chromosomes in Pop^t and identifies the second non-dominated front. This process is repeated until all the chromosomes are categorised into different fronts (2, . . ., m), where each generated front may contain multiple non-dominated chromosomes.

Once the Pareto fronts are obtained, a new child population is created by concatenating the ranked fronts in several steps (see lines 14–20). First, the algorithm verifies that there is enough space in the child population to add all the chromosomes in each ranked front $rank_k^f$ by comparing the remaining space in the child population ($n - Pop^c.length()$) with the rank front size $rank_k^f.length()$ (see line 16). If there is enough space, the rank front is directly added to the child population (see line 17). If there is no space, then the algorithm identifies the most prominent chromosomes in the front using a crowd comparison sort [16] and assigns them to the child population (see line 19). Through the loop of lines 15–20, the algorithm filters out the chromosomes in the total population Pop^t with the highest objective fitness values. The new population is used as the next parent population and again synthesizes a new child population by performing

crossover and mutation (see lines 21–22). Finally, the non-dominated front (the Pareto optimal solution) $Rank^f$ is returned to the user which constitutes a suggestion for the best clustering of BOs and operation nodes in the system to develop MSs, line 24.

4 Implementation and Validation

A proper MS system should provide high execution efficiency with desirable levels of scalability and availability. Furthermore the packages and components related to each MS should be highly cohesive and loosely coupled [1, 17]. In order to validate that our MS discovery and recommendation process provides MSs with these desirable characteristics, while optimizing BO relationships and operation call costs, we developed a prototype[5] based on the algorithms presented in Section 3 and experimented with it on the SugarCRM and ChurchCRM Systems.

This section presents the details of the experiments that we conducted on both the SugarCRM and ChurchCRM customer relationship management systems. SugarCRM contains 8116 source files and 600 attributes divided between 101 tables, while ChurchCRM contains 8039 source files and 350 attributes divided between 55 tables. Execution sequences were generated for both systems covering all the major functionalities and user cases related to them[6]. The logs related to both systems were captured using the log generation functionality available in the systems. They were then analyzed using the process mining tool Disco[7] and call graphs were generated for SugarCRM with 178 unique nodes and for ChurchCRM with 58 unique nodes. Each of the nodes in call graph represents unique operation performed on database tables in the systems and the edges between each node represent the number of calls between the nodes.

Discovered MSs: Based on the provided data, the prototype managed to identify 13 different business objects related to SugarCRM such as 'action control lists', 'calls', 'contacts', 'campaigns', 'meetings', 'users', 'prospects', 'accounts', 'documents', 'leads', 'emails', 'projects' and 'email management'. The 'calls' and 'meetings' BOs are exclusively contained in 'contacts' BO and 'documents' BO exclusively contained in 'users' BO. The system identified 12 BOs such as 'calendar', 'locations', 'deposits', 'emails', 'events', 'family', 'group', 'property', 'query', 'users' and 'kiosk' for ChurchCRM. The BOs identified and the call graphs with execution details were given to the optimization algorithm and it derived 8 MSs for ChurchCRM and 11 MSs for SugarCRM.

Validation Process: The validation process was conducted by implementing two MSs for each system in Google Cloud. Each MS was hosted in Google Cloud using a cluster of size 2 which has two virtual CPUs and a total memory of 7.5 GB. The hosted MSs were exposed through the Google Cloud kubernetes API[8], allowing third party computers to access them via API calls. In order to simulate the legacy systems we created services which contain all the functionalities related to both MSs. The read and write operations to the local database were simulated by reading and writing data to a file resides in the MSs, themselves.

[5] https://github.com/AnuruddhaDeAlwis/NSGAII.git

[6] http://support.sugarcrm.com/Documentation/Sugar_Versions/8.0/Pro/Application_Guide/

[7] https://fluxicon.com/disco/

[8] https://kubernetes.io/

In order to validate the performance of the ChurchCRM MS system, evaluation was conducted under two criteria. First we tested each MS system against a load of 150,000 and 300,000 requests generated by 10 machines simultaneously, simulating the customer requests, while recording their total execution time, average memory consumption and average disk consumption. Then, we tested the same load against the legacy ChurchCRM system. In this execution we directly call the functions related to each MS without creating any interactions between the MSs. The results obtain are recorded in Table 3. However, in the real environment MSs interact with each other through executions and service calls. As such, to evaluate the performance of the MS system when there are interactions between MSs in the system, we generated service calls between them and tested against a similar load as described previously. The results obtained for the interactive execution are recorded in Table 5. We followed a similar approach to test the MS system related to SugarCRM with a load of 50,000 requests and 100,000 requests generated by 10 machines simultaneously. The results for the SugarCRM MS system without interaction between them are recorded in Table 7, while Table 9 contains the results obtained when there are interactions between MSs in the MS system. Since we have identified several exclusive containment relationships in SugarCRM, it was important to evaluate the validity of our Heuristic. As such, we implemented two MS systems. The first MS system contained two MSs which are exclusively related, but deployed in separate containers. The second MS system contained operations of both exclusively related BOs and deployed in a single container. Both systems were tested against a load of 50,000 requests and 100,000 requests generated by 10 machines and the obtained results are summarized in Table 11.

Based on the results detailed in Tables 3, 5, 7, 9 and 11 we calculated the scalability, availability and execution efficiency of different combinations and the results obtained are summarized in Tables 4, 6, 8, 10 and 12. Scalability was calculated according to the resources usage over time as described by Tsai *et al.* [19]. In order to determine availability, first we calculated the time taken to process 100 requests if a particular MS is not available. Then, we used the difference between the total up-time and total down-time as described by Bauer *et al.* [20]. Efficiency gain was calculated by dividing the total time taken by the legacy system to process all requests by the total time taken by the corresponding MS system. Furthermore, we calculated the structural cohesion and coupling of the packages in the legacy system and the new MS systems, as described by Candela *et al.* [5]. The results related to cohesion and coupling for the MSs and the legacy systems are summarized in Table 1 and 2.

Experimental Results: According to Tsai *et al.* [19], the lower the number the better the scalability. Thus, it is evident that the ChurchCRM legacy system has better scalability, availability and efficiency than MS system when there are no interaction between modules (refer Table 4). However, when there are interactions, MS system perform better in all three criteria of scalability, availability and efficiency than the legacy system as depicted in Table 6. When there are no interactions between modules, the SugarCRM MS system achieved less scalability than the legacy system (refer Table 8). However, the SugarCRM MS system managed to attain higher availability and eight times better execution efficiency than the legacy system when there are no interactions between MSs in the MS system as depicted in Table 8. When there are interactions

between modules in SugarCRM, MS system performed better in all criteria than the legacy system as depicted by Table 10. Apart from legacy system comparisons, it is evident from Table 12 that exclusively contained MSs attained a higher level of scalability, availability and efficiency gain when they were developed and deployed as single MSs without separating them into different containers.

According to Candela *et al.* [5], lack of cohesion and structural coupling are inversely related to the cohesion and coupling of the system. The lower the number the better the cohesion and coupling of the system [5]. When considering the cohesion and coupling values of ChurchCRM MSs in Table 1, it is evident that they have obtained better cohesion and coupling values than the legacy system. Even though Table 2, related to SugarCRM MSs, does not show a prominent gain in cohesion and coupling values related to all MSs, most of them manage to achieve either better cohesion or coupling values when compared with the legacy system.

The obtained results have affirmed that the MSs extracted based on the optimization algorithm and the recommendation of the prototype can provide the same services to users while preserving overall system behaviour and achieving higher scalability, availability, efficiency, high cohesion and low coupling while aligning with BO relationships and containment heuristics.

5 Related Work

Microservices have emerged as the latest style of service-based software allowing systems to be distributed through the cloud as fine-grained components, typically with individual operations, in contrast to services under SOA which include all logically related operations [1]. Even though microservices can support the evolution of ERP systems by providing exploitation in cloud-enabled platforms such as the IoT [2], the research conducted in this particular area is limited. To the best of our knowledge there is no research related to the automation of MS discovery in legacy systems, apart from the manual migrations achieved by Balalaie *et al.* [21]. They have described the complexity associated with the system reengineering process while pointing out the importance of considering BOs and their relationships in the system migration process. Martin Fowler emphasizes the importance of adapting BO relationships in microservices [17] by mentioning the Domain Driven Design (DDD) principles [18].

However, existing software re-engineering techniques do not consider the complex relationship of BOs along with their behaviours in the re-engineering process. Furthermore, studies show that the success rate of existing software re-modularisation techniques, especially for large systems, remains low [5]. A key stumbling block is the limited insights available from syntactic structures of software code for profiling software dependencies and evaluating their measurements for coupling and cohesion metrics [4]. As such, to derive successful re-engineering techniques, a methodology should consider the enriched semantic insights available through the BOs and functions in an ES.

In such a process the first challenge would be identifying the BOs which are distributed among several database tables in an ES system, while identifying the relationships among them. Nooijen *et al.* [14] and Lu *et al.* [9] have proposed methodologies

Table 1: Comparison of lack of cohesion and structural coupling in ChurchCRM.

Measure	Legacy	MS1	MS2	MS3	MS4	MS5	MS6	M7S	MS8
Lack of Cohesion	122	32.5	0.0	0.5	0.5	0.0	0.0	0.5	1.0
Structural Coupling	58	45.5	9.5	11.0	11.5	9.5	10.0	12.0	13.0

Table 2: Comparison of lack of cohesion and structural coupling in SugarCRM.

Measure	Legacy	MS1	MS2	MS3	MS4	MS5	MS6	M7S	MS8	MS9	MS10	MS11
Lack of Cohesion	0.428	0.0	2.0	0.25	0.0	0.5	0.2	0.0	0.0	0.0	1.0	0.11
Structural Coupling	7.857	3.0	17.33	12.0	9.0	20.0	11.8	15.7	5.0	24.0	14.67	8.1

Table 3: Legacy vs MS results without interactions between modules for ChurchCRM.

System Type	No of Requests	Ex. Time (ms)	Avg Mem (GB)	Avg Disk (MB)
Legacy	150000	280800	3.0075	25.49
Legacy	300000	540000	3.084	45.719
MS System	150000	306000	2.915	25.094
MS System	300000	626400	3.018	46.369

Table 4: Legacy vs MS System characteristics comparison for ChurchCRM.

Campaign Type	Scalability [Mem]	Scalability [Disk]	Availability [150000]	Availability [300000]	Efficiency [150000]	Efficiency [300000]
Legacy	1.896	3.3166	99.8754	99.940	1.00	1.00
MS System	2.169	3.8716	99.8642	99.930	0.92	0.86

Table 5: Legacy vs MS results with interactions between modules for ChurchCRM.

System Type	No of Requests	Ex. Time (ms)	Avg Mem (GB)	Avg Disk (MB)
Legacy	150000	756000	3.043	25.225
Legacy	300000	1188000	3.1025	45.192
MS System	150000	709200	2.89	13.502
MS System	300000	954000	3.0043	24.239

Table 6: Legacy vs MS System characteristics comparison with interactions between modules for ChurchCRM.

Campaign Type	Scalability [Mem]	Scalability [Disk]	Availability [150000]	Availability [300000]	Efficiency [150000]	Efficiency [300000]
Legacy	1.259	2.212	99.665	99.685	1.00	1.00
MS System	0.941	1.624	99.868	99.894	1.07	1.26

Table 7: Legacy vs MS results without interactions between modules for SugarCRM.

System Type	No of Requests	Ex. Time (ms)	Avg Mem (GB)	Avg Disk (MB)
Legacy	50000	1101600	2.9325	25.36
Legacy	50000	2394000	2.94	25.48
MS System	100000	122400	2.953	25.20
MS System	100000	298800	3.0605	45.99

Table 8: Legacy vs MS System characteristics comparison for SugarCRM.

Campaign Type	Scalability [Mem]	Scalability [Disk]	Availability [50000]	Availability [100000]	Efficiency [50000]	Efficiency [100000]
Legacy	2.367	2.372	95.779	97.662	1.000	1.000
MS System	3.088	5.438	99.512	99.702	9.000	8.012

Table 9: Legacy vs MS results with interactions between modules for SugarCRM.

System Type	No of Requests	Ex. Time (ms)	Avg Mem (GB)	Avg Disk (MB)
Legacy	50000	72000	2.941	24.789
Legacy	100000	169200	3.119	48.866
MS System	50000	72000	2.898	19.712
MS System	100000	140400	4.088	43.174

Table 10: Legacy vs MS System characteristics comparison with interactions between modules for SugarCRM.

Campaign Type	Scalability [Mem]	Scalability [Disk]	Availability [50000]	Availability [100000]	Efficiency [50000]	Efficiency [100000]
Legacy	2.929	5.443	99.713	99.831	1.000	1.000
MS System	2.682	4.164	99.713	99.859	1.000	1.21

Table 11: Exclusive MS vs separated MSs execution results for SugarCRM.

System Type	No of Requests	Ex. Time (ms)	Avg Mem (GB)	Avg Disk (MB)
Exclusive MS System	50000	64800	2.9965	25.116
Exclusive MS System	100000	136800	3.0700	25.471
Separated MS System	50000	237600	3.0025	25.321
Separated MS System	100000	1252800	3.0218	36.009

Table 12: Exclusive MS vs separated MSs characteristics comparison for SugarCRM.

Campaign Type	Scalability [Mem]	Scalability [Disk]	Availability [50000]	Availability [100000]	Efficiency [50000]	Efficiency [100000]
Exclusive MS System	2.283	2.260	99.971	99.984	3.67	5.27
Separated MS System	13.99	19.77	99.895	99.861	1.00	1.00

and heuristics to identify BOs based on the database schema and information in database tables. However, according to Lu *et al.* these derived BOs might not be perfect and they have to be re-clustered with the help of human expertise. Furthermore, Fuguo *et al.* have derived BO relationships using the information available in WSDL files and have classified the BO relationships into exclusive containment, inclusive containment, subtype and etc [12]. Apart from BO relationships, the number of execution calls between different microservices plays a major role in defining optimal MSs, because an excessive number of network calls can increase the response time while decreasing the availability of the service [1].

A proper identification of factors which fall under the behaviour of the system can be evaluated thorough dynamic analysis of the system [13]. However, there is still a gap in the area of correlating structural and behavioural analysis while considering the underlying BO semantics. As such, it is important to consider optimization techniques such as genetic algorithms to incorporate multiple objectives (i.e. system structure and behaviour) [5] in the software re-engineering process while focusing on BO relationships and the execution calls between different operations.

6 Conclusion

This paper presented a technique that can support the re-engineering of enterprise systems into microservices based on business objects and their relationships and associated operations with the coherent features and minimum communication overhead. A prototype was developed based on the proposed heuristic and the NSGA II optimization algorithm, and a validation was conducted using the implemented MS systems recommended by the prototype for SugarCRM and ChurchCRM systems. The paper demonstrated that the analysis of functions, CRUD operations, and BO relationships of an ES supports the effective identification of a solution to migrate the system into the corresponding MS system that has high cohesion, low coupling, and achieves higher scalability, higher availability, and processing efficiency.

References

1. Newman, S.: Building MSs NGINX. 1st edn. O'Reilly, Sebastopol (2015)
2. 2017 Internet Of Things (IoT) Intelligence Update, https://www.forbes.com/sites/louiscolumbus/2017/11/12/2017-internet-of-things-iot-intelligence-update/#43aa6f4c7f31. Last accessed 5 May 2018

3. Magal, S.R. and Word, J.: Integrated business processes with ERP systems. 1st edn. Wiley Publishing, (2011)
4. Anquetil, N. and Laval, J.: Legacy software restructuring: Analyzing a concrete case. In Software Maintenance and Reengineering (CSMR). In: Software Maintenance and Reengineering (CSMR) 15th European Conference, pp. 279–286 (2011)
5. Candela, I., Bavota, G., Russo, B. and Oliveto, R.: Using cohesion and coupling for software remodularization: Is it enough?. In: ACM Transactions on Software Engineering and Methodology (TOSEM), pp. 24. (2016)
6. Shatnawi, A., Seriai, A.D., Sahraoui, H. and Alshara, Z.: Reverse engineering reusable software components from object-oriented APIs. In: Journal of Systems and Software, pp. 442–460. (2017)
7. Barros, A., Decker, G., Dumas, M., and Weber, F.: Correlation patterns in service-oriented architectures. In: Proceedings of the 10^{th} International Conference on Fundamental Approaches to Software Engineering (FASE), pp. 245–259. Springer, (2007)
8. Pĕrez Castillo, R., Garcĩa Rodrĩguez de Guzmãn, I., Caballero, I. and Piattini, M.: Software modernization by recovering Web services from legacy databases. In: Journal of Software: Evolution and Process, pp. 507–533. (2013)
9. Lu, X., Nagelkerke, M., van de Wiel, D. and Fahland, D.: Discovering interacting artifacts from ERP systems. In: IEEE Transactions on Services Computing, pp. 861–873. (2015)
10. De Alwis, A., Barros, A., Polyvyanyy, A. and Fidge, C.: Function-Splitting Heuristics for Discovery of Microservices in Enterprise Systems. In: In International Conference on Service-Oriented Computing, (2018) (accepted on July 25, 2018)
11. Kumaran, S., Liu, R. and Wu, F.Y.: On the duality of information-centric and activity-centric models of business processes. In: In International Conference on Advanced Information Systems Engineering, pp. 32–47. Springer, Berlin, Heidelberg (2008)
12. Wei, F., Ouyang, C. and Barros, A.: Discovering behavioural interfaces for overloaded web services. In: Services (SERVICES), 2015 IEEE World Congress, pp. 286–293 (2015)
13. Hull, R.: Artifact-centric business process models: Brief survey of research results and challenges. In : On the Move to Meaningful Internet Systems, pp. 1152–1163. Springer (2008)
14. Nooijen, E.H.J, van Dongen, B. F. and Fahland, D.: Automatic discovery of data-centric and artifact-centric processes. In : In International Conference on Business Process Management, pp. 316–327. Springer (2012)
15. Hayes, I.: Specification case studies, 2nd edn. Prentice Hall International Ltd. UK (1987)
16. Deb, K., Pratap, A., Agarwal, S. and Meyarivan, T.A.M.T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. In: IEEE transactions on evolutionary computation, pp. 182-197. (2002)
17. Microservices a definition of this new architectural term, https://martinfowler.com/articles/microservices.html. Last accessed 3 May 2018
18. Evans, E.: Domain-driven design: tackling complexity in the heart of software, 1st edn. Addison-Wesley Professional (2003)
19. Tsai, W.T., Huang, Y. and Shao, Q.: Testing the scalability of SaaS applications. In: Service-Oriented Computing and Applications (SOCA), IEEE International Conference, pp. 1–4. (2011)
20. Bauer, E. and Adams, R.: Reliability and availability of cloud computing, 1st edn. John Wiley & Sons (2012)
21. Balalaie, A., Heydarnoori, A. and Jamshidi, P.: Migrating to cloud-native architectures using MSs: an experience report, In European Conference on Service-Oriented and Cloud Computing, pp. 201–215. Springer (2015)