A technique to identify microservices on monolithic systems

Wen Zhenglin

2016-3-29

Overview

- Technique
- Evaluation

Idea

Monolithic systems get larger over the time, deviating from the intended architecture, and becoming tough, risky, and expensive to evolve

The idea behind microservices architecture is to develop a single large, complex, application as a suite of small, cohesive, independent services

Major challenge

Despite these problems, enterprise systems often adopt monolithic architectural styles

Therefore a major challenge nowadays on enterprise software development is to evolve monolithic system on tight business schedules, target budget, but keeping quality, availability, and reliability

The technique Successfully applied

Successfully applied on a 750 KLOC real-world monolithic banking system, which manages transactions from 3.5 million banking accounts and performs nearly 2 million authorizations per day

Here we demonstrate that our approach could identify good candidates to become microservices on a 750 KLOC banking system, which reduced the size of the original system and took the benefits of microservices architecture, such as services being developed and deployed independently, and technology independence

The Proposed Technique

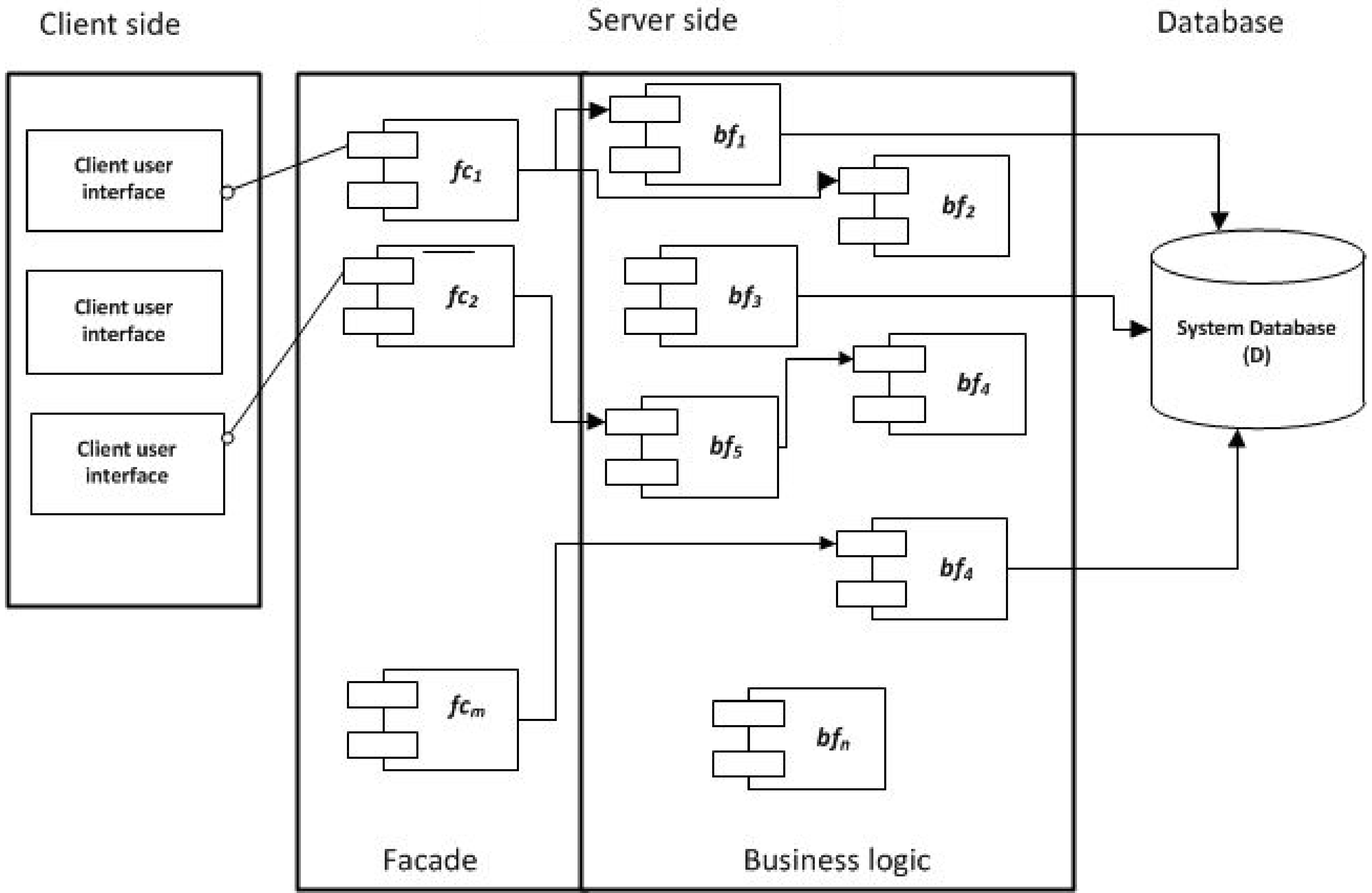
Three main parts

The proposed technique considers that monolithic enterprise applications have three main parts

- A client side user interface
- A server side application
- A database

It also considers that a large system is structured on smaller subsystems and each subsystem has a well-defined set of business responsibilities

We also assume that each subsystem has a separate data store


Client side
Server side
Database
Client user interface
Client user interface
Client user interface
fc_1
fc_2
fc_m
bf_1
bf_2
bf_3
bf_4
bf_5
bf_4
bf_n
System Database (D)
Facade
Business logic

In formal terms

We assume that a system S is represented by a triple (F, B, D)

F = {fc1 , fc2 , . . . , fcn } is a set of facades

B = {bf1 , bf2 , . . . , bfn } is a set of business functions

D = {tb1 , tb2 , . . . , tbn } is a set of database tables

Facades (fci) are the entry points of the system that call business functions (bfi). Business functions are methods that encode business rules and depend on database tables (tbi)

Define an enterprise organization

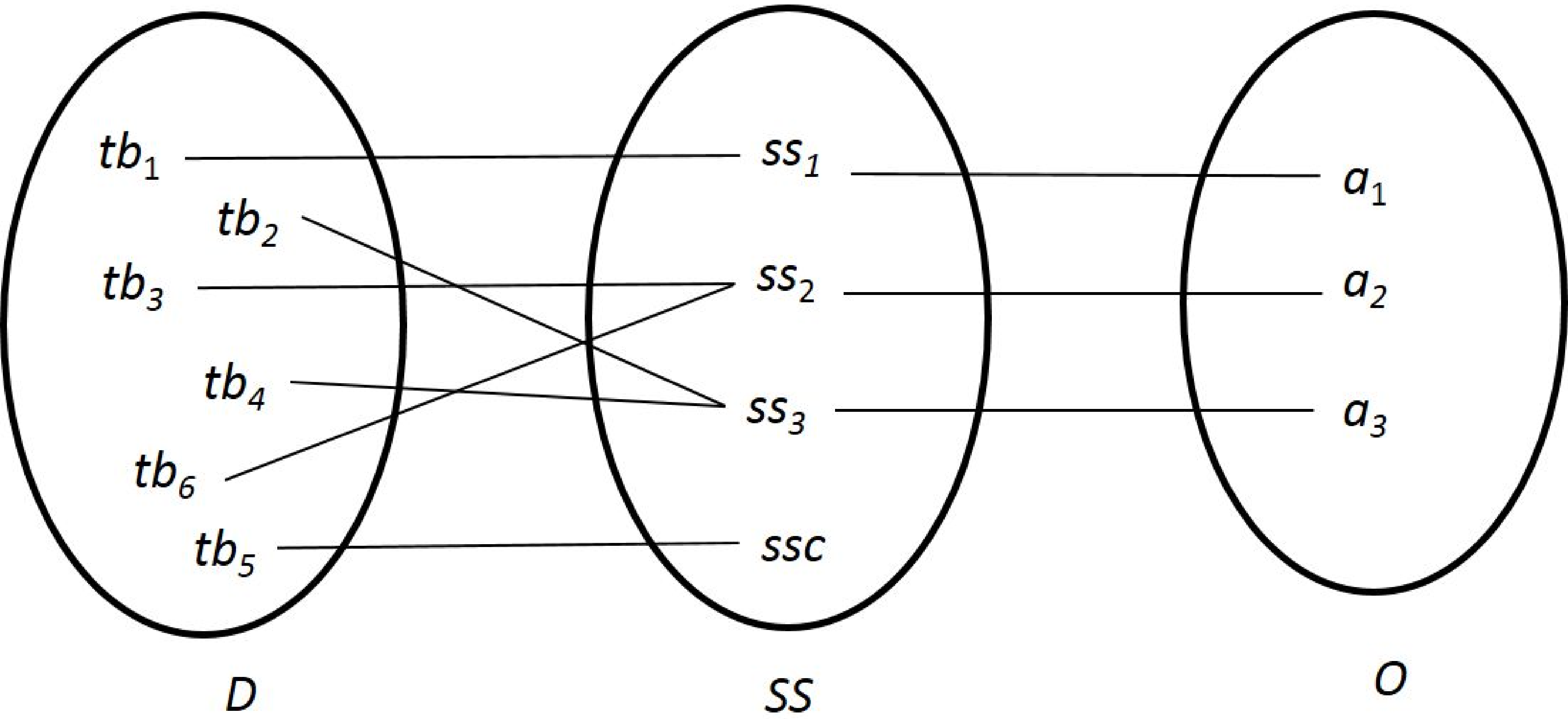
$O = \{a_1, a_2, \ldots, a_w\}$ is divided into business areas a_i, each responsible for a business process

We describe our technique to identify microservices on a system S

Step #1

Map the database tables tbi ∈ D into subsystems ssi ∈ SS

Each subsystem represents a business area (ai) of organization O


tb_1
tb_2
tb_3
tb_4
tb_6
tb_5
ss_1
ss_2
ss_3
ssc
a_1
a_2
a_3
D
SS
O

Database decomposition

Subsystem SS2 , which represents business area a2 , depends on database tables tb3 and tb6

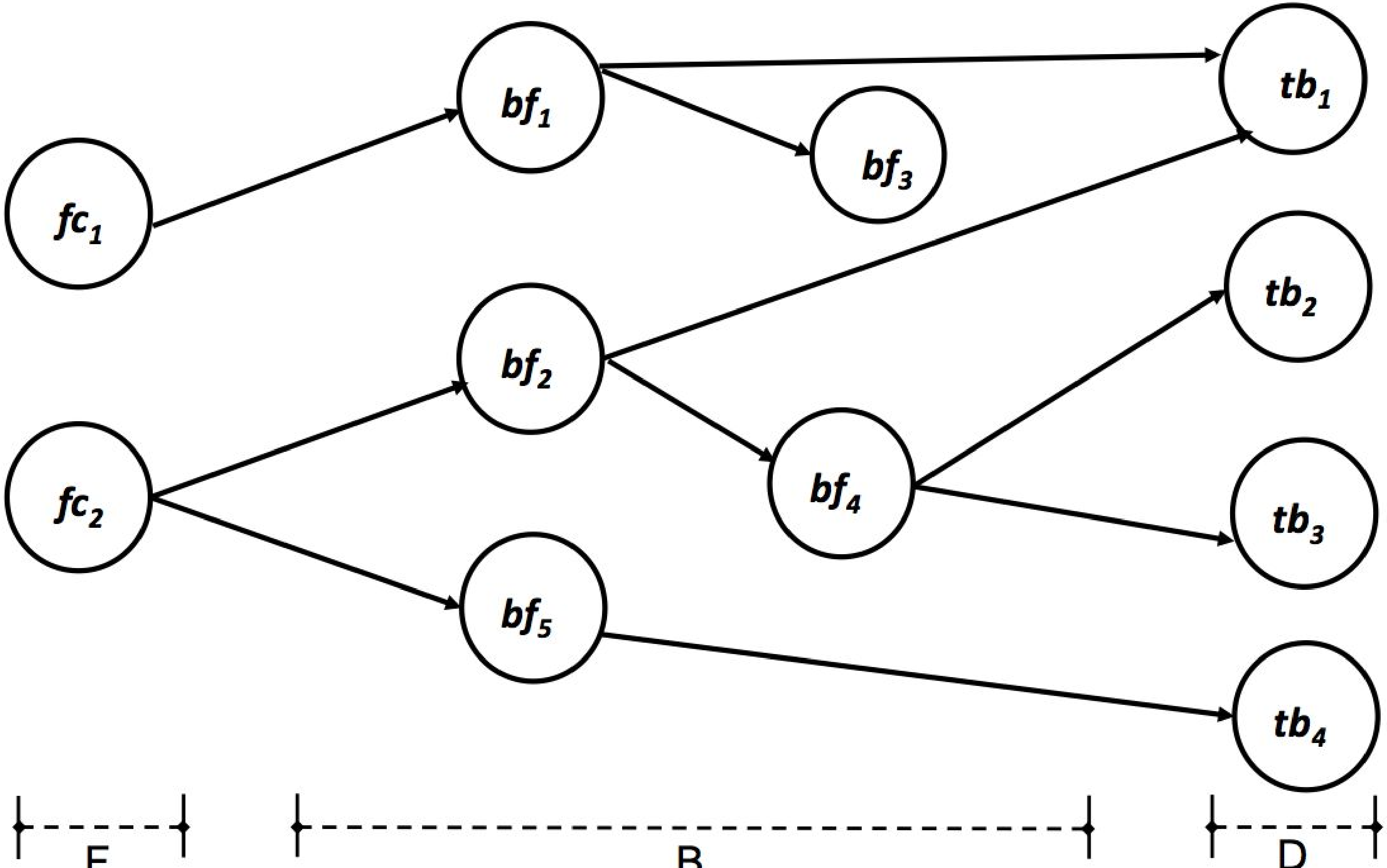
Tables unrelated to business process—e.g., error messages and log tables—are classified on a special subsystem called Control Subsystem (SSC)

Step #2

Create a dependency graph (V, E) where vertices represent facades (fci ∈ F)

Business functions (bfi ∈ B), or database tables (tbi ∈ D) and edges represent:

1. Calls from facades to business functions (case i)
2. Calls between business functions (case ii)
3. Accesses from business functions to database tables (case iii)


fc_1
fc_2
bf_1
bf_2
bf_3
bf_4
bf_5
tb_1
tb_2
tb_3
tb_4
F
B
D

Dependency Graph

Facade fc2 calls business function bf2 (case i)

Business function bf2 calls business function bf4 (case ii)

Business function bf4 accesses database tables tb2 and tb3 (case iii)

Step #3

Identify pairs (fci , tbj) where fci ∈ F and tbj ∈ D

there is a path from fci to tbj on the dependency graph

Eg.

We identify the pairs (fc1 , tb1), (fc2 , tb1), (fc2 , tb2),

(fc2 , tb3), and (fc2 , tb4)

Step #4

For each subsystem ssi previously defined in Step #1

Select pairs (fci , tbj) identified on the prior step where tbj $\in$ ssi

Eg.

ss3 = {tb2 , tb4 } then we select the pairs (fc2 , tb2) and (fc2 , tb4)

Step #5

Identify candidates to be transformed on microservices

For each distinct pair (fci , tbj) (a candidate microservice) obtained on the prior step, we inspect the code of the facade and business functions that are on the path from vertex fci to tbi in the dependency graph

Eg.

pair (fc2 , tb2), we inspect facade fc2 and business functions bf2 and bf4

The inspection aims to identify which business rules actually depend on database table tbj and such operations should be described in textual form as rules

A candidate microservice (M) is defined as follows

Name: the service name according to pattern [subsystemname].[processname]. For instance, Order.TrackOrderStatus

Purpose: one sentence that describes the main business purpose of the peration, which is directly associated to the accessed database entity domain. For instance, track the status of a customer order

Input/Output: the data the microservice requires as input and produces as output—when applied—or expected results for the operation. For instance, microservice Order.TrackOrderStatus requires as input the Order Id and produces as output a List of finished steps with conclusion date/time and a List of pending steps with expected conclusion date

M defined (cont.)

Features: the business rules the microservice implements, which are described as a verb (action), an object (related to the database table), and a complement. For instance, Identify the order number, Get the steps for delivery for type of order, Obtain finished steps with date/time, and Estimate the date for pending steps

Data: the database tables the microservice relies on

Step #6

Create API gateways to turn the migration to microservices transparent to clients

API gateway consists of an intermediate layer between client side and server side application

It is a new component that handles requests from client side—in the same technology and interface as fci —and synchronizes calls to the new microservice M and to fci’ —a new version of fci without the code that was extracted and implemented on microservice M

An API gateway should be defined for each facade

Three cases of synchronization

- (i) When the input of fci’ is the output of M or the input of M is the output of fci’
- (ii) When the input of M and fci’ are the same as API gateway input and the instantiation order is irrelevant
- (iii) When we have to split fci into two functions fci’ and fci’’ and microservice M must be called after fci’ and before fci’’

If we can synchronize the calls as described on case (i) or (ii),we identify the proposed microservice M as a “strong candidate”

If we can only synchronize the calls as in case (iii), we identify the proposed icroservice as a “candidate with additional effort”

Non candidate

Particularly in our technique, assuming a microservice of a subsystem ssx

If we identify a business rule in the microservice definition that needs to update data in tbi $\in$ ssx and tbj $\in$ ssx in the same transaction scope

We identify the proposed microservice as a “non candidate”

Microservice candidate

When every evaluated pair (fci , tbj) of a subsystem is classified as microservice candidate, we recommend to migrate the entire subsystem to the new architecture

In this case, we have to implement the identified microservices, create an independent database with subsystem tables

Develop API gateways, and eliminate the subsystem implementation (source code and tables) from system S

Although API gateways must be deployed in the same server as system S to avoid impacts on client side layer, the microservices can be developed using any technology and deployed wherever is more suitable

Evaluation

This technique applied our proposed technique on a large system from a Brazilian bank

The system handles transactions performed by clients on multiple banking channels (Internet Banking, Call Center, ATMs, POS, etc.)

It has 750 KLOC in the C language and runs on Linux multicore servers

The system relies on a DBMS with 198 tables that performs, on average, 2 million transactions a day

Step #1

We identified 24 subsystems including subsystem SSC

Table shows a fragment of the result obtained after this initial step

Headers represents subsystems and their content represent the tables they rely on

One problem we identified is that certain tables—highlighted in grey—are associated to more than one subsystem

Table 1. Mapping of Subsystems and Tables

Business Actions	Service Charges	Checks	Clients	Current Accounts	Saving Accounts	Social Bennefits	Pre-approved credit	Debit and Credit cards	SMS Channel
ACO	AGT	CHS	CLT	*CNT*	*CNT*	BEN	LPA	CMG	CTS
ACB	ISE	TCE		CCT	CPO	DPB		INP	STS
RCA	PTC	ECH		*RCC*	*LAN*	DBC		NPP	CMS
	PTF	HET		*LAN*	*RCC*	IBS		BIN	RCS
	RTE	CCF		CCO	MPO	LBE		CCM	RLS
	RTT			CCE	PPO			PBE	RTS
	TPT			CHE	SPA				
	TTE								
	UTM								
	DUT								

Step #2

We created a dependency graph composed by 1,942 vertices (613 facades, 1,131 business functions, and 198 database tables)

5,178 edges representing function calls and 2,030 edges corresponding to database table accesses

Due to the size of our evaluated system, we proceed our evaluation to the following five subsystems:

Business Actions,Service Charges, Checks, Clients and SMS channel

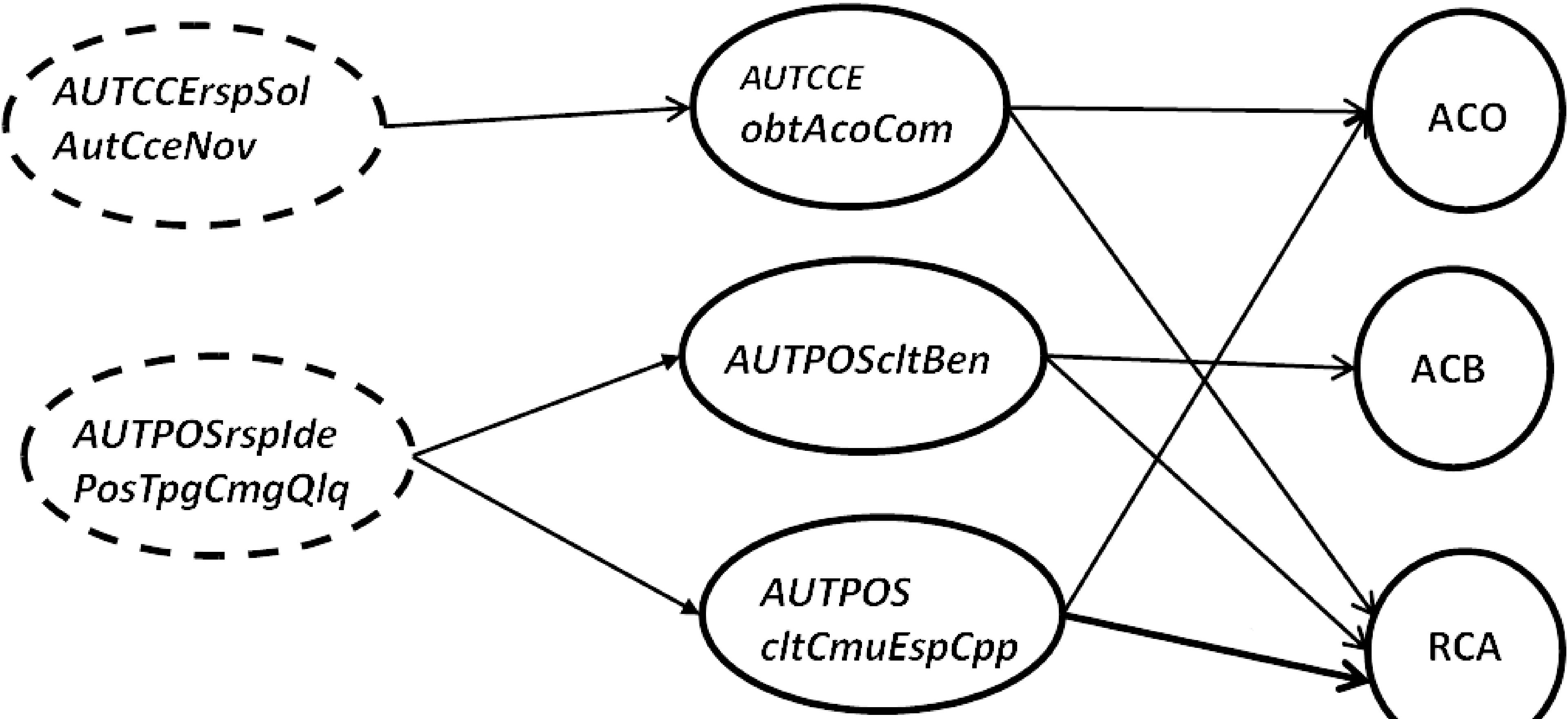
Table 2. Evaluated Subsystems

Subsystem	Business actions	Service Charges	Checks	SMS channel	Client
Tables (vertices)	3	10	5	6	1
Functions (vertices)	5	62	29	138	>150
Function calls (edges)	3	79	14	133	>150
Database accesses (edges)	6	14	22	140	26
Microservices candidates	1	3	8	4	4

Steps #3–#4

Considering subsystem Business Actions, we find the following pairs:

1. (AUTCCErspSolAutCceNov, ACO),
2. (AUTPOSrspIdePosTpgCmgQlq, ACO),
3. (AUTPOSrspIdePosTpgCmgQlq, ACB),
4. (AUTCCErspSolAutCceNov, RCA),
5. (AUTPOSrspIdePosTpgCmgQlq, RCA)


AUTCCErspSol
AutCceNov
AUTCCE
obtAcoCom
ACO
AUTPOScltBen
ACB
AUTPOSrspIde
PosTpgCmgQlq
AUTPOS
cltCmuEspCpp
RCA

Step #5

For pairs (AUTCCErspSolAutCceNov, ACO), (AUTCCErspSolAutCceNov, RCA) obtained on prior step

We inspect the code of facade AUTCCErspSolAutCceNov and the business functions it calls (AUTCCEobtAcoCom)

Identified microservice

Name: BusinessActions.ListBusinessActionsForAccount

Purpose: List business actions related to an account on the relationship channel

Input/Output: Account number and channel id as input, and a list of business actions as output

Features: Retrieve business actions assigned to the account; retrieve business actions enabled for the relationship channel; and retrieve the list of business actions assigned to the account and enabled for the channel

Data: Database tables ACO and RCA

Evaluated the other three pairs

We also evaluated the other three pairs (AUTPOSrspIdePosTpgCmgQlq, ACO), (AUTPOSrspIdePosTpgCmgQlq, RCA), (AUTPOSrspIdePosTpgCmgQlq, ACB)

The business functions they call AUTPOScltBen and AUTPOScltCmuEspCpp, and we identified the same microservices as the one described above

In fact, table ACB could be merged with ACO

Step #6

For facades AUTCCErspSolAutCceNov and AUTPOSrspIdePosTpgCmgQlq

We identified an API gateway that suits case (i) described in the proposed technique

We also evaluated steps #3 to #6 for subsystems:

Service Charges, Checks, Clients and SMS channel

Only three microservices

Although subsystem Service Charge has 10 tables and 51 facades, we only identified and defined the following three microservices:

- ServiceCharge.CalculateServiceCharge
- ServiceCharge.IncrementServiceChargeUsage
- ServiceCharge.DecrementServiceChargeUsage

Use a message queue manager (MQM) for communication

We identified 14 API gateways that also suit case (i)

We identified other 37 API that suit case (ii)

However, we can avoid the development of the last 37 API gateways since the called microservices have only input data and can be implemented with an asynchronous request

Thus, particularly in this case, we suggest to use a message queue manager (MQM) for communication and substitute the C code that performs an update on database table for a “put” operation on a queue

For subsystems Checks and SMS channel

We identified microservices and APIs with the same characteristics of subsystem Service Charge, which indicates that both subsystems are good candidates to be migrated to microservices

Nevertheless, for subsystem Client—which accesses only one table—we identified one microservice that must be called by more than 50 API gateways that suits case (iii)

Therefore, we did not recommend its migration to microservices, since the effort to split and remodularize more than 50 functions, create and maintain more than 50 API gateway are probably greater than the benefits of microservice implementation

Last but not least

We disregard subsystems that have one or more tables that appear in more than one subsystem list, such as table CNT of subsystem Current Account

Because our technique to identify microservices considers that only one subsystem handles operation on each table

Brief discussion

We classify our study as well-succeeded because we could identify and classify all subsystems

Create and analyze the dependency graph that helped considerably to identify microservices candidates

As our practical result, we recommended to migrate 4 out of the 5 evaluated subsystems to a microservice architecture

Reference

1. Alessandra Levcovitz , Ricardo Terra and Marco Tulio Valente. Towards a Technique for Extracting Microservices from Monolithic Enterprise Systems, UFMG,Belo Horizonte, Brazil, UFLA,Lavras, Brazil

Thank you !