
LEVERAGING SOFTWARE ENGINEERING FRAMEWORKS,
METHODS AND TOOLS TO AUTOMATE CRM PRE-SALES
IN REAL ESTATE APPLICATIONS

Submitted to Indian Statistical Institute
in partial fulfillment of the thesis requirements for the Degree of
DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE



Author: **Parthasarathi Ray**
Senior Research Fellow


Supervisor: **Pinakpani Pal**
Associate Professor

Applied Statistics Unit
Indian Statistical Institute
Kolkata - 700108, India

*Dedicated to
my wife, Sugata,
and
our son, Sambuddha.*

DECLARATION OF AUTHORSHIP

I, **Parthasarathi Ray**, a research scholar at *Applied Statistics Unit (ASU), Indian Statistical Institute (ISI), Kolkata*, declare that this thesis encompasses the research conducted by me under the guidance of **Pinakpani Pal (ECSU, ISI Kolkata)** with valuable inputs from **Prof. Subhamoy Maitra (ASU, ISI Kolkata)**. I affirm that this work is entirely original, both in terms of research content and narrative, and to the best of my knowledge, the materials contained in this thesis have not previously been published or written by any other person, nor has been submitted as a whole or as a part for any degree/diploma or any other academic award anywhere before.



Parthasarathi Ray

Applied Statistics Unit

Indian Statistical Institute, Kolkata

203, Barrackpore Trunk Road

Kolkata 700108, INDIA.

CERTIFICATE FROM SUPERVISOR

This is to certify that the work contained in the thesis entitled “**Leveraging software engineering frameworks, methods and tools to automate CRM pre-sales in real estate applications**”, submitted by **Parthasarathi Ray** for the award of the degree of Doctor of Philosophy in Computer Science to Indian Statistical Institute, Kolkata, is a record of the bonafide research works carried out by him under my direct supervision and guidance.

I consider that the thesis has reached the standards and fulfilling the requirements of the rules and regulations relating to the nature of the degree. The contents embodied in the thesis have not been submitted as a whole or as a part for the award of any degree or diploma or any other academic award anywhere given before.



Pinakpani Pal

Associate Professor

ECSU

Indian Statistical Institute, Kolkata

203, Barrackpore Trunk Road

Kolkata 700108, INDIA.

LIST OF PUBLICATIONS/MANUSCRIPTS

The list of publications/manuscripts that are included in this thesis is as follows. Chapter 3 is based on papers 1 and 3. Chapter 4 is based on paper 2. Chapter 5 is based on paper 4. Chapter 6 is based on paper 5. Chapter 7 is based on paper 6.

1. **Parthasarathi Ray** and Pinakpani Pal, “Extending the SEMAT Kernel for the Practice of Designing and Implementing Microservice-Based Applications using Domain Driven Design”, [IEEE 32nd Conference on Software Engineering Education and Training \(CSEE&T\), 2020](#), DOI: <https://doi.org/10.1109/CSEET49119.2020.9206200>.
2. **Parthasarathi Ray** and Pinakpani Pal, “An agile approach to automate Real Estate CRM (pre-sales) using Scrum and Essence”, [Conference on Software Engineering Research & Practice \(SERP\), 2021, \(In Press\)](#)
3. **Parthasarathi Ray** and Pinakpani Pal, “An Essence based framework using a Domain Driven Design approach to address Microservices lifecycle from identification to implementation”, [Conference on Scientific Computing \(CSC\), 2021, \(In Press\)](#)
4. **Parthasarathi Ray** and Pinakpani Pal, “Extending Essence to adopt User Story and Microservice practices leveraging Scrum to automate pre-sales function of real estate CRM”, [communicated to: Journal of Software: Evolution and Process, Manuscript ID: JSME-24-0292](#).
5. **Parthasarathi Ray** and Pinakpani Pal, “Petri net modelling of key CRM pre-sales functionalities for real estate”, [communicated to: IEEE Transactions on Software Engineering, Manuscript ID: TSE-2024-07-0377](#).
6. **Parthasarathi Ray** and Pinakpani Pal, “Petri net modelling of certain key operational aspects regarding leads and Sales Executives in real estate CRM”, [communicated to: IEEE Transactions on Software Engineering, Manuscript ID: TSE-2024-07-0375](#).

ACKNOWLEDGEMENT

I would like to express my heartfelt gratitude and appreciation to my research project guide, Pinakpani Pal, who spent countless hours to ideate and refine the ideas with his focused guidance and constructive criticism. It has been a wonderful and instructive journey for me. I extend my deep gratitude to the head of my department, Professor Subhamoy Maitra, for sharing his expertise and knowledge, timely facilitation and thought provoking insights. I also extend my thanks and gratitude to the Applied Statistics Unit (ASU) in particular and Indian Statistical Institute in general, and my heartfelt appreciation to the esteemed faculty members at ISI, Kolkata, for their guidance and support throughout my tenure at the institute. I am also grateful to the non-teaching staff at the ASU and the Dean's office (like Nayan, Arijit, Surajit and Amarjit) for their assistance.

My heartfelt gratitude goes to my parents for their love and encouragement, and a special shout out to my wife, Sugata, for her patient understanding and unwavering support throughout the highs and lows of this period. I am also thankful to the other family members and friends, and my colleagues and juniors in ISI like Animesh, Debasmita, Jyotirmoy, Subhra, Abhinav, Manmatha and Suman for their continuous support and help.

ABSTRACT

In this thesis, we leverage the learning from Essence, a “language and kernel“ of Software Engineering, which has resulted from the efforts of the SEMAT initiative founded to bring together industry, research and education to deal with the problem of immature practice in software engineering.

We have developed a framework to address the microservices lifecycle using Domain Driven Design (DDD), and introduced three alphas along with the Work Products and Activities associated with those alphas. This is industry agnostic and can be used anywhere.

Then onwards, we have focused our attention on automating real estate CRM (pre-sales) as our industry scenario, and have stated with creating an agile approach using Scrum and Essence, intended to address the issues that many Scrum implementations face by leveraging the strengths of Essence.

Our next endeavour has been to create a full-fledged software engineering method that adopts the User Story practice to address the requirements area and the microservices practice to develop and deploy the software system. This is intended to provide a comprehensive view of the software endeavor by creating a method adopting a set of practices using Essence as the common ground. This exercise, carried out for our industry scenario, can be similarly extended for other industries as well with Essence as the unifying framework.

Delving deeper into CRM pre-sales functionalities, we have decided to focus on the core area of the assignment process that deals with the dynamics of scheduling/rescheduling of site visit requests from leads in the backdrop of the availability/unavailability of Sales Executives on the said day of site visit along with applicable constraints. This is essentially a workflow process, and we have used Petri nets to model the same, given the widespread applicability of Petri net in modelling similar application domains and workflows. We have constructed generalized Petri net models of the assignment process and verified if they satisfy the desired properties of the systems being modelled by carrying out their behavioural analysis.

We would next consider those processes whose execution is necessary as prerequisites for the assignment process to function, and which are also instrumental in setting up the post-processing aspects of the assignment process. Key outcome of those processes would be the priority setting of leads (indicative of their maturity potential) and the queue adjustment of Sales Executives (indicative of their availability at that point of time). To ensure the smooth running of the automation process it is necessary to consider the operational aspects of these processes we are talking about. Accordingly we have detailed out the underlying functionalities of these processes with their operational characteristics during system go-live followed by the steady state execution. We have modelled those processes using Petri net and validated their conformance to certain verification criteria by doing behavioural analysis.

The value of the Petri net based modelling exercises in terms of clarifying and improving our design understanding can also apply to other industry domains where similar Petri net based modelling and analysis can be conducted.

Contents

1	Introduction	25
1.1	Problem statement	28
1.2	Research questions	28
1.3	Objectives	29
1.3.1	General objectives	29
1.3.2	Specific objectives	29
1.4	Scope	29
1.5	Structure of the thesis	30
2	Preliminaries and Background	31
2.1	SEMAT and Essence	31
2.1.1	The SEMAT initiative	32
2.1.2	SEMAT in the industry and academia	33
2.1.3	SEMAT Essence Kernel	33
2.1.4	SEMAT Essence Language	36
2.1.5	Integrating specific theories and practices using Essence	38
2.2	Microservices and Domain Driven Design (DDD) - An Architectural Overview and Activities involved	38
2.3	Petri net	42
2.4	Customer Relationship Management (CRM)	43

2.4.1	CRM functionalities prevalent in real estate industry	44
3	An Essence based framework using a Domain Driven Design approach to address Microservices lifecycle from identification to implementation	47
3.1	Alphas for Microservices practice using DDD	48
3.1.1	Alpha: Features	48
3.1.2	Alpha: Microservice	49
3.1.3	Alpha: Domain Model	52
3.2	Work Products and Activities for Microservices practice using DDD .	55
3.2.1	Work Products and Activities for the Feature alpha	55
3.2.2	Work Products and Activities for the Microservice alpha . . .	58
3.2.3	Work Products and Activities for the Domain Model alpha . .	60
3.3	Summary of chapter	63
4	An agile approach to automate Real Estate CRM (pre-sales) using Scrum and Essence	65
4.1	Business Process Modelling for in-scope CRM pre-sales functionalities	66
4.1.1	Background	66
4.1.2	Broad Scope	67
4.1.3	Functionalities to be Modelled	67
4.1.4	Process Modelling	68
4.2	Adoption of Essence and Scrum	69
4.2.1	The Kernel Alpha states of the development endeavor undertaken by the <i>Nirmanik</i> Team	70
4.2.2	Adoption of Scrum within the Essence framework	72
4.3	Summary of chapter	73
5	Extending Essence to adopt User Story and Microservice practices leveraging Scrum to automate pre-sales function of real estate CRM	75
5.1	Context	78

5.2	Considerations for composing practices into method using Essence . . .	79
5.3	Adoption of User Story practice	81
5.3.1	User Story Lite practice - an overview	82
5.3.2	How we adopted User Story Lite practice for <i>Nirmanik</i>	82
5.3.3	Alpha state cards for Requirements	86
5.3.4	Applicable Essence cards for User Story Lite	86
5.3.5	The Value of the Kernel to the User Story Lite practice	87
5.4	Adoption of Microservice practice	89
5.4.1	Microservices Lite practice - an overview	89
5.4.2	How we adopted Microservices Lite practice for <i>Nirmanik</i>	91
5.4.3	Applicable Essence cards for Microservices Lite	96
5.4.4	The Value of the Kernel to the Microservices Lite practice	96
5.5	Summary of chapter	98
6	Construct a generalized Petri net model of key CRM pre-sales real estate functionalities	101
6.1	Context	103
6.2	Functionalities for assigning leads to appropriate Sales Executives: life-cycle of leads	105
6.3	Functionalities for assigning leads to appropriate Sales Executives: assignment workflow process description	108
6.3.1	Provisional Assignment process for Prospective Leads	108
6.3.2	Rescheduling for Scheduled Leads	109
6.3.3	Assignment process for Visited Leads	111
6.3.4	Requesting Original Sales Executive change while requesting re-visit for Visited Leads	113
6.3.5	Rescheduling for Active Leads	114
6.3.6	Requesting Original Sales Executive change for Active Leads	115
6.3.7	Requesting Original Sales Executive as well as Date change for Active Leads	117

6.3.8	The rules of ordering of Sales Executives to be considered for the concerned lead for the date in question	118
6.4	Modelling considerations	119
6.4.1	Desired properties of the model being constructed	120
6.4.2	Suitability of Petri net as our modelling tool	121
6.5	Petri net modelling of the assignment process workflow	122
6.5.1	Creating generalized Petri net models for the process flows	122
6.5.2	Petri net modelling of <i>Block I</i> (flow for selection of dates)	125
6.5.3	Petri net modelling of <i>Block II</i> (flow for selection of slots)	131
6.6	Verification of our Petri net models	136
6.6.1	Verification of Petri net modelling - <i>Block I</i>	139
6.6.2	Verification of Petri net modelling - <i>Block II</i>	143
6.6.3	Additional behavioural properties exhibited by the Petri net models for <i>Block I</i> and <i>Block II</i>	147
6.6.4	Summary of the verification exercise conducted on the Petri net models for <i>Block I</i> and <i>Block II</i>	149
6.7	Summary of chapter	150
7	Petri net modelling of certain key operational aspects regarding leads and Sales Executives in real estate CRM	153
7.1	Background	155
7.2	System Initialization	158
7.2.1	Slot Initialization with queue setting (Go-Live)	158
7.3	Steady State Operation	158
7.3.1	Slot Initialization with queue setting (Steady State)	160
7.3.2	Priority setting of leads	161
7.3.3	Queue adjustment for SEs	163
7.4	Modelling considerations: Desired properties for the model to exhibit	170
7.5	Petri net modelling of functionalities in scope	171
7.5.1	Petri net modelling of slot initialization with queue setting	171

7.5.2	Petri net modelling of priority setting of leads	173
7.5.3	Petri net modelling of queue adjustment for SE	177
7.6	Verification of our Petri net models	179
7.6.1	Verification of Petri net modelling of slot initialization with queue setting	181
7.6.2	Verification of Petri net modelling of priority setting of leads .	182
7.6.3	Verification of Petri net modelling of queue adjustment of SEs	183
7.6.4	Summary of the verification exercise conducted on the Petri net models	184
7.7	Summary of chapter	185
8	Conclusion	187

List of Figures

2-1	Method architecture [1].	33
2-2	(a) User Story alpha card; (b) Prepare User Story activity card. Both showing linkage with Essence elements.	36
3-1	The Essentialized model showing the work products and the activities for the Feature alpha	56
3-2	The Essentialized model showing the work products and the activities for the Microservice alpha	58
3-3	The Essentialized model showing the work products and the activities for the Domain Model alpha	61
4-1	Communication Flow of real-estate pre-sales CRM	68
4-2	Functional Flowchart of real-estate pre-sales CRM	68
4-3	Alpha State Cards	69
5-1	(a) An example of a shared Practice Library. (b) Method: composition of practices on top of the Essence kernel and language (for <i>Nirmanik</i>)	81
5-2	The Essentialized model showing the User Story Lite practice [2].	83
5-3	Splitting the first user story: <i>Nirmanik::Track pre-sales leads</i>	85
5-4	Requirements alpha state card [2]: (a) Conceived. (b) Bounded. (c) Coherent.	86

5-5	(a) User Story alpha card; (b) Prepare User Story activity card. Both showing linkage with Essence elements.	87
5-6	(a) Story Card work product. (b) Find User Story activity card.	87
5-7	(a) The Splitting User Story pattern card [2]. (b) Accept User Story activity card.	88
5-8	User Story Lite coverage of kernel solution activity spaces.	88
5-9	The Essentialized model showing the Microservices Lite practice [2].	90
5-10	Design Model for <i>Nirmanik::Lead Tracking Subsystem</i>	92
5-11	Microservice Design of <i>Nirmanik:: Ingesting Leads from external sources</i>	95
5-12	Microservice alpha card.	96
5-13	Work product card (a) Design Model. (b) Microservice Design.	97
5-14	Microservices Lite coverage of kernel solution activity spaces.	97
5-15	Coverage of the composition of User Story Lite and Microservices Lite of kernel solution activity spaces.	99
6-1	Statechart Diagram for the life cycle of the leads.	120
6-2	Petri net diagram of <i>Block I</i> first variant - DateSelector	126
6-3	Petri net diagram of <i>Block I</i> second variant - DateIterator	126
6-4	Petri net diagram of <i>Block II</i> - SlotAllocator and SlotUpdater	132
6-5	<i>Reachability tree for Block I (DateSelector)</i>	140
6-6	<i>Reachability tree for Block II (SlotUpdater)</i>	145
7-1	Petri net modelling of Slot Initialization with Queue - for one SE.	172
7-2	Petri net diagram of priority setting for a lead.	174
7-3	Petri net diagram for queue adjustment for SEs.	177

List of Tables

2.1	Elements of Essence language	37
5.1	The states of the Essence Kernel alphas mapped with regard to the development endeavor undertaken by <i>Nirmanik</i> [3].	79
6.1	Categorization of flows described in sections 6.3.1 to 6.3.7 according to the composition of the respective variants of the date selection flow and the slot selection flow as applicable.	125
7.1	Reachability Tree for fig. 7-1.	182
7.2	Reachability Tree for fig. 7-2: decrement operation.	183
7.3	Reachability Tree for fig. 7-2: increment operation.	184

LIST OF ACRONYMS AND ABBREVIATIONS

Expansion	Acronyms/ Abbreviations
Customer Relationship Management	CRM
Sales Executive	SE
Software Engineering Method and Theory	SEMAT
Object Management Group	OMG
Unified modeling language	UML
Domain Driven Design	DDD
INVEST	Independent, Negotiable, Valuable, Estimable, Small, and Testable
That is	i.e.
Visit request	VR
Initial visit request	InitVR
Reschedule request	RR
Original Sales Executive	OS
OS change request	OSCR
Reschedule and OS change request	ROSCR
Callback	CB
No Queue	NQ

LIST OF SYMBOLS

L_{rd} - Lead requesting to schedule an appointment for Requested Date rd ;

SE^i - i^{th} Sales Executive;

EA_{dt} - Evaluate Allocation possibility of the given date from the date range dt ;

m_{dt} - Instances of dates in the range opened up / made available for allocation;

AR_{dt} - Accept or Reject date dt (evaluate whether the instance dt matches rd);

R_{dt} - Reject date dt since it doesn't match with rd ;

A_{dt} - Accept date dt since it matches with rd ;

RR_{dt} - Return Rejected date instances;

L_{cb} - Lead to be called back at a later date;

odL_{rd} - Lead requesting to schedule an appointment for date rd , with possible presence of an Original Date od (in case the request is for rescheduling);

SE_{od}^j - Possible un-allocation of the previous assignment of j^{th} Sales Executive for the said lead on Original Date od ;

ES^i - Evaluate Suitability of i^{th} Sales Executive;

SE_{sd}^i - Collection of assignable slots sd prepared for i^{th} Sales Executive (for the date considered dt);

m_{sd}^i - instances of slots (collection of sd) available to be assigned for i^{th} Sales Executive for date dt ;

AR_{sd} - Accept or Reject slot sd ;

R_{sd} - Reject slot sd ;

RR_{sd} - Return Rejected slot instances;

$L_{sd}^{i_p/o}$ (for *SlotAllocator*) - Matching of i^{th} Sales Executive (SE^i) for the lead for slot sd for the first time (with no previous allocation to be unassigned), indicating a provisional matching;

$L_{sd}^{i_p/o}$ (for *SlotUpdater*) - Matching of i^{th} Sales Executive (SE^i) for the lead for slot sd , which is either $L_{sd}^{i_p}$ indicating SE^i is an Alternate Sales Executive (a provisional matching) where $i \neq j$, or else it is $L_{sd}^{j_o}$ indicating the matching is for the Original Sales Executive (SE^j) where $i = j$;

Visited : previous day lead visited information;

Skipped : previous day lead skipped visit information;

p - System capacity in days to provision future site visit requests;

P_i^l - Priority value i assumed by lead l ;

ψ - Maximum priority value that can be assumed by a lead capped to a specific value (positive integer) by the system administrator.

Q_j^i - Queue value at priority group j for SE SE^i ;

h_i^g - The global queue value of SE_i

h_i^{dt} - The daily queue value of SE_i for date dt

NQ^i - No Queue state for i^{th} Sales Executive;

dt_{gl} - go-live date

dt_{sys} - system date

n_{pr} - the number of leads belonging to the priority group pr having SE_i as OS

t_{pr} - the threshold value for the priority group pr

f_{pr} - a function f mapping the number of leads n_{pr} belonging to the priority group pr having SE_i as the OS, with respect to the threshold value t_{pr} for that priority group pr .

Contents

1.1	Problem statement	28
1.2	Research questions	28
1.3	Objectives	29
1.3.1	General objectives	29
1.3.2	Specific objectives	29
1.4	Scope	29
1.5	Structure of the thesis	30

Software seems to be all-pervasive nowadays [4]. However, even though there have been numerous success stories, there have been significant failures with software implementation as well. The situation today is just as Brooks [5] stated many years ago: “In no other discipline is the gulf between best practice and typical practice so wide.” Lethbridge et al [6] noted that while most practitioners can quickly get up to speed when it comes to software technologies (such as web, mobile, cloud technologies, etc.), it is much harder to raise capabilities in software engineering, such as requirements, analysis, architecture, testing, and project management. Practitioners frequently rely solely on their, often limited, experience and proceed by trial-and-error without any real learning.

In the continuously evolving landscape of Software Engineering, a multitude of

approaches have made their way, lending structure and discipline to the creation of software products. While the number of these methods and practices has shown a remarkable increase over time, one fall-out of this has been the propensity of throwing out the methods and way of workings that the organizations had established thus far, and again starting over with new methods. Perhaps unsurprisingly, while there have been success stories, they are punctuated by way too many failed efforts, some of them turning out to be quite costly ones.

In this thesis, we have started off by leveraging the learning from Essence, a “language and kernel“ of Software Engineering, which has resulted from the efforts of the SEMAT initiative founded to bring together industry, research and education to deal with the problem of immature practice in software engineering and has evolved as a method- and practice-independent approach.

Our focus has been on creating reusable assets as we have created frameworks and built up methods consisting of practices by essentializing those.

We have developed a framework to address the microservices lifecycle using Domain Driven Design (DDD) and introduced three alphas to extend the SEMAT Kernel, and added the Work Products and Activities associated with those alphas. This is industry agnostic and can be used anywhere.

We have then taken up the industry area of our interest and created an agile approach for automating Real Estate CRM (pre-sales) aspects using Scrum and Essence, intended to address the issues that many Scrum implementations face by leveraging the strengths of Essence. Our next endeavour has been to create a full-fledged Software Engineering method that adopts the User Story practice to address the requirements area and the microservices practice to develop and deploy the software system being built by Essentializing them. Thus we have been able to provide a comprehensive view of the software endeavor by creating a method adopting a set of practices using Essence as the common ground. This exercise, carried out for a specific industry use case, can be similarly extended for other industry use cases as well with Essence as the unifying framework.

We have continued with the industry context of the automation of CRM pre-sales

for real estate as a running thread. Amongst the functionalities, we have identified a focus area of the assignment process dealing with the dynamics of scheduling/rescheduling of site visit requests from leads in the backdrop of the availability/unavailability of Sales Executives on the said day of the site visit along with applicable constraints. We have decided to use Petri nets to model the same which is essentially a workflow process, given the widespread applicability of Petri net in modelling related application domains and workflows.

After explaining the life-cycle of leads and allocation of Sales Executives, we have constructed generalized Petri net models of the assignment process. The resulting generalized Petri net models have system properties that can be directly linked to certain desirable performance criteria of systems as borne out by the behavioural analysis conducted on them to obtain a verification of the suitability of our modelling.

We also acknowledge that in order to ensure the smooth running of the automation process when it is eventually implemented, it is crucial to focus on the operational aspects of the same, with the execution of the assignment process being central to it. One such aspect would be the initiation of the automated system on the very first day of its operation (i.e. the go-live date), while the other aspect to follow logically in its wake would deal with how it runs in steady state. We have described the underlying functionalities, modelled those using Petri net, and verified if they satisfy the desired properties of the systems being modelled by carrying out behavioural analysis of the Petri nets.

However, the value of the Petri net based modelling exercise undertaken in terms of clarifying and improving our design understanding that is key to successful implementation applies not only to the business scenarios in scope for our current work (i.e. real estate CRM), but it can also extend to other industry domains where similar Petri net based modelling and analysis can be conducted.

1.1 Problem statement

Usage of software is ubiquitous across the globe, and stories of success and failure have proliferated over time. As Lethbridge et al. have mentioned, it might be possible to pick up speed quickly in software technologies but swift upskilling in Software Engineering is a much more difficult proposition [6]. Practitioners in Software Engineering are guided more by their own experience which is usually limited and hence the proceedings happen mostly by trial-and-error. The key question is how to enable practitioners or project teams with the necessary wherewithal to ensure success in project delivery.

1.2 Research questions

The following questions are defined:

1. There are several practices in use for different aspects of software development. How can they be adopted into a method to ensure successful project delivery?
2. While SEMAT speaks about bridging the gap between industry and academia using Essence, can we take up an industry agnostic scenario as well as an industry scenario and show how we can leverage Essence to deliver in both cases?
3. Amongst the functionalities involved in the industry scenario, can we focus on a core functionality and model it using a suitable framework, and validate that against certain desirable verification criteria?
4. Can we focus on the operational aspects of the said functionality and identify the operations necessary for it to function correctly, model those appropriately and verify those models' effectiveness?

1.3 Objectives

1.3.1 General objectives

For the industry scenario of automating pre-sales CRM for real estate, leverage software engineering frameworks, methods and tools like Essence to construct a method consisting of several practices for development purposes, and Petri net to model a core functionality and the associated processes that carry out the pre-processing and post-processing of that functionality from an operational aspect.

1.3.2 Specific objectives

1. Create an industry agnostic framework to adopt certain practices leveraging Essence.
2. Create an agile method leveraging Essence out of representative practices used in different areas of software development in the context of an industry scenario.
3. Focus on a core functionality in that industry scenario, model it using a suitable framework and validate against certain verification criteria.
4. Describe the operational aspects of system initialization followed by the steady state operation by focusing on a set of processes needed for the pre and post-processing of the said core functionality.

1.4 Scope

This thesis involves the automation of CRM in real estate leveraging various methods and tools of Software Engineering like Essence and Petri net, and select practices in use. The scope of this research is described as follows:

1. Provide a background of the key frameworks/methods and the industry scenario.
2. Create an industry agnostic framework to address the Microservices lifecycle using DDD, leveraging Essence.

3. Craft a method to deliver the functionalities of real estate CRM (pre-sales) by selecting the representative practices from the respective areas of software development, essentialize and adopt them into the unifying framework of Essence.
4. Focus on a core functionality from the industry scenario, assignment of leads to Sales Executives based on their site visit requests, and model that using Petri net, and validate that against certain desirable verification criteria.
5. Use Petri net to model the operation of processes responsible for pre-processing and post-processing of the assignment process, key outcomes being lead priority setting and SE queue adjustment, and verify those models.

1.5 Structure of the thesis

This Ph.D. Thesis is organized into eight chapters as follows:

In this Chapter (Chapter 1) we provide an overview of the research including an introduction, problem statement, research question, objectives and scope.

In Chapter 2 we discuss the preliminaries and provide a background of the key frameworks/methods utilized and the industry scenario.

In Chapter 3 we create an Essence based framework using a Domain Driven Design approach to address Microservices lifecycle from identification to implementation.

In Chapter 4 we show how to create an agile approach to automate Real Estate CRM (pre-sales) using Scrum and Essence.

In Chapter 5 we extend Essence to adopt User Story and Microservice practices leveraging Scrum to automate the pre-sales function of real estate CRM.

In Chapter 6 we construct a generalized Petri net model of a key functionality in CRM pre-sales, the assignment process.

In Chapter 7 we carry out the Petri net modelling of certain key operational aspects regarding leads and Sales Executives in real estate CRM.

In Chapter 8 we take a look back at the work we have covered in this thesis and conclude.

Preliminaries and Background

Contents

2.1 SEMAT and Essence	31
2.2 Microservices and Domain Driven Design (DDD) - An Architectural Overview and Activities involved	38
2.3 Petri net	42
2.4 Customer Relationship Management (CRM)	43

In this section, we take a look at the landscape of Software Engineering focusing on the methods and tools that we would be leveraging extensively (Essence, Microservices, Domain Driven Design and Petri net) in this thesis, as well as discuss the key concepts of the domain/industry (CRM in real estate) to be used as a common thread throughout our thesis.

2.1 SEMAT and Essence

Software usage is ubiquitous across the globe, and stories of success and failure have proliferated over time. The following quote by Brooks, while made long back, still seems to hold: “In no other discipline is the gulf between best practice and typical practice so wide” [5]. As Lethbridge et al. have mentioned, it might be possible to pick up speed quickly in software technologies but swift upskilling in software engineering is

a much more difficult proposition [6]. Practitioners in software engineering are guided more by their own experience which is usually limited and hence the proceedings happen mostly by trial-and-error.

As software engineering has continued to evolve, it has brought in its wake a host of methods over time that have been embraced to create software products. The number of these methods has been on the rise over time, and a consequence observed of their adoption has been the discarding of methods previously being used by the organizations while embracing the new. That inevitably takes a toll on the software practitioners as the established way of working gets upset, which may very well have a bearing on the forthcoming outcomes. This points to an inherent immaturity in software engineering which needs to be addressed.

2.1.1 The SEMAT initiative

“The recent SEMAT (Software Engineering Method and Theory) initiative was founded to bring together industry, research and education to deal with the problem of immature practice in software engineering” [7]. In that regard, SEMAT focuses its effort towards the following :

1. “a software engineering language and kernel”, and,
2. “a general and widely accepted theoretical framework.”

The outcome of the first element of the initiative is a “language and kernel” of software engineering - termed Essence. It allows us to extend according to practices adopted for a given project and facilitates analysis and comparison of software development methodologies [4, 2], and has been demonstrated to provide guidance to both small and large development [8, 9].

The second element of the initiative culminates in a framework that is pragmatic enough to be of use to software teams [7, 10].

2.1.2 SEMAT in the industry and academia

The kernel and language defined in the Essence are scalable, extensible, and easy to use, including the ability to compare, evaluate, adapt, simulate and measure methods [11, 1]. The kernel includes elements forming the basis of vocabulary and it is extensible for defining future technologies, practices, social working patterns, methods, and research [1]. Besides, the kernel is extensible to support different projects, allowing the possibility to add practices, such as use cases, architecture, and component-based development. Also, according to OMG, SEMAT reduces the gap between academic research and heuristic application in industry. The method architecture included in the Kernel is shown in Figure 2-1 [1].

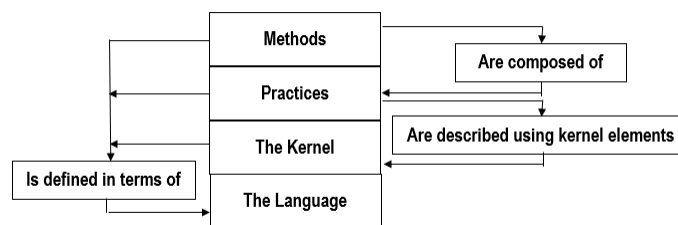


Figure 2-1: Method architecture [1].

The SEMAT Essence kernel addresses industry orientation through the incorporation of relevant features and tools. Also, it has been augmented with different approaches to arrive at solutions for different organizational challenges.

2.1.3 SEMAT Essence Kernel

Essence is the software engineering kernel resulting from the efforts of SEMAT which has become an OMG industry standard [11, 1]. Embodying “the essential rather than the accidental in software engineering” [2], it extracts the salient features of software engineering and “makes them practical and actionable” for real life software implementation utilizing a “multi-dimensional state-based method-independent model of software engineering endeavors” [8, 9].

The kernel includes elements forming the basis of vocabulary and it is extensible

for defining future technologies, practices, social working patterns, methods, and research [1]. It allows for “extension mechanisms to add project or practice specifics” which makes Essence versatile enough to reach out to and compare various development methodologies [2]. It is used to describe practices in a method-agnostic way, allowing us to select necessary practices and compose them to form the method as required for the engagement. Thus Essence is like a complementary framework that can be used with whatever framework or method an organization is using.

SEMAT Essence Kernel defines the most basic elements needed by all kinds of development. It also facilitates the understanding of developers regarding the state the project is in and the activities to be accomplished next [9].

Essence Kernel includes six elements:

- Alpha,
- State,
- Checklist,
- Activity,
- Activity Space, and
- Work Product [9].

Essence examines the prevailing software development lifecycles through several dimensions utilizing the principle of separation of concerns [12, 13]. Essence terms these dimensions as alphas. Tracking the progress of alphas is of critical importance in the development process. The alphas are characterized by states indicating their advancement through a lifecycle. Each state is accompanied by a checklist containing quantitative criteria that must be achieved, thus allowing for measurements. A state is said to be achieved on completion of all the items in the checklist and this is how it enables a development team to gain awareness of their project situation. The checklist equips a team to assess the current project situation and identify problems in the current step [9]. State transition without satisfying the checklists implies

assuming risk in that dimension [4]. Work products are tangible artifacts describing an alpha and they help verify achievement of alpha states. Activity is performed for the creation or updation of a work product [2]. Activity Space is another key aspect of Essence Kernel which talks about concrete development activities and tasks, whereas a Work Product captures the results [9]. This is how Essence Kernel facilitates the management and monitoring of projects utilizing all these elements [9].

The progress of an alpha might be achieved by progressing smaller parts of the alpha, called a sub-alpha. “Sub-alphas are alphas in their own right that help to move forward or slow the progress of the kernel alphas” [2]. We would be using these two terms alpha and sub-alpha interchangeably as we build the model for the practices in focus of this thesis subsequently.

The Essence specification identifies seven alphas that are commonly applicable for all software engineering endeavors:

- “Opportunity,
- Stakeholders,
- Requirements,
- Software System,
- Work,
- Team, and
- Way-of-Working” [4].

The above-mentioned alphas help categorize the factors identified in the existing literature. For the uncategorized ones, Essence allows for “extension mechanisms to add project or practice specifics” which makes Essence versatile enough to reach out to various development methodologies [9].

Essence organizes the elements of software engineering, e.g. the seven common alphas as above, into three areas of concern, the ambit of each of those covering a distinct dimension of software development [9].

Customer: covers everything related to the usage of the software system intended to be delivered. The alphas Opportunity and Stakeholders are mapped here.

Solution: contains everything to do with the specification and construction of the software system. The alphas Requirements and Software System are mapped here.

Endeavor: about the team undertaking development work and their way of approach to carry it out. The alphas Work, Team and Way of Working are mapped here.

Table 2.1 summarizes the concepts for the elements used in Essence language along with their corresponding icon symbols [2].

Out of the artifacts mentioned there, we have selected two key artifacts, an alpha card and an activity card, and shown the two corresponding cards for illustration purposes. Fig. 2-2(a) and Fig. 2-2(b) show the Essence alpha card “User Story” and the Essence activity card “Prepare a User Story” respectively, pointing out their linkage with the constituent Essence elements.

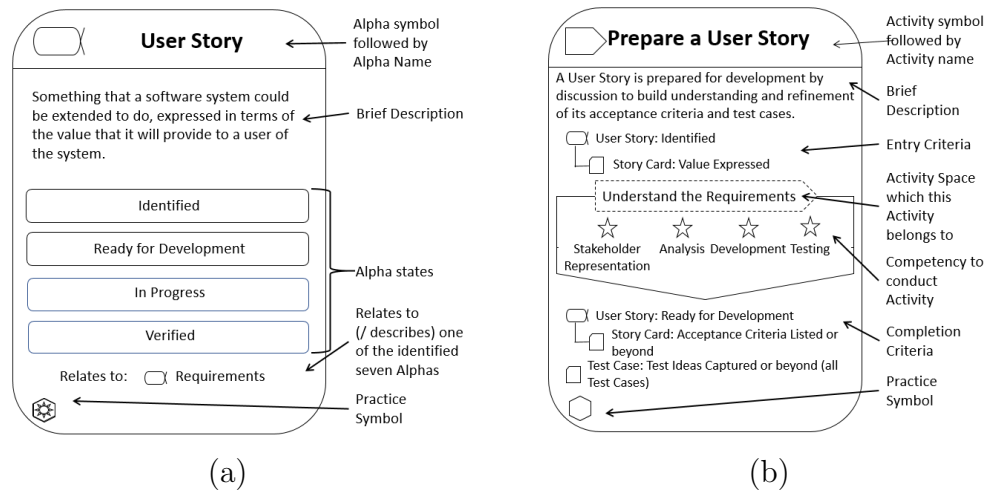
















Figure 2-2: (a) User Story alpha card; (b) Prepare User Story activity card. Both showing linkage with Essence elements.

2.1.4 SEMAT Essence Language

A collection of elements is used to delineate and articulate the relationships among concepts in Essence in a visual manner. “The elements in Essence Language are the

Table 2.1: Elements of Essence language

Element Type	Symbol	Description
Alpha		An essential element of the software engineering endeavor that helps assessing the progress and health of the endeavor
Alpha State		The alpha states denote the lifecycle of an alpha
Activity		Things which practitioners carry out
Activity Space		A placeholder for things to be carried out in the software engineering endeavor with zero to many activities
Work Product		The tangible artifacts produced by practitioners while conducting software engineering activities
Level of Details		The extent of detail in a Work Product. An example would be a Story Card having its Value Expressed or additionally having its Acceptance Criteria Listed
Competency Level		Encompasses the abilities, capabilities, attainments, knowledge, and skills needed to accomplish a certain kind of work. A number is provided to indicate expertise level necessary
Pattern		An arrangement of other elements represented in the language
Resource		A source of information or content, such as a website or a book reference
Essentialized Method		A team's way of constructing its work in a System of Interest development endeavor
Essentialized Practice		A repeatable approach to carry out something with main areas of concern that a software engineering endeavor has to focus upon
Essence Kernel		A set of elements forming a common ground to describe a systems engineering endeavor
Essence Language		The Essence Language is the domain-specific language to define methods, practices and kernels
Essence		Essence comprises both a language and a kernel of software engineering to provide a common ground as a basis for presenting guidelines for all practices

same as the elements in Kernel: Alpha, Alpha State, Activity, Activity Space, Work Product and Competency” [1].

It is also possible for developers to use Essence Language to “define new Kernel, State, and practices based on SEMAT Essence” to address specific requirements [14].

2.1.5 Integrating specific theories and practices using Essence

We have to appreciate that each software engineering methodology is unique with different objectives and factors, hence the need for a specific theory that explores the relationship among them. Those factors of interest, however, still have to remain valid within “a more general software engineering context” which is “modelled in Essence through alphas and alpha states” [4].

Understanding of the above would prove useful in chapters 3, 4 and 5 where we would be leveraging Essence to essentialize practices and create our own methods.

2.2 Microservices and Domain Driven Design (DDD) - An Architectural Overview and Activities involved

Microservices architecture is becoming increasingly instrumental for organizations across the globe in their quest for digital transformation, as well as to help shorten the time to market for their new lines of products and services. Initiatives by organizations like Netflix, Amazon, Coca-Cola, Uber and so on to migrate legacy applications into a microservice architecture or build new applications altogether leveraging microservices bear testimony to that [15].

The microservices architecture is about developing an application as a collection of single-purpose services. Each of these services would be in charge of its individual data, logic, and behaviour. A microservice is expected to realize only a specific business capability that stems from distilling the functionalities around a common theme resulting in a bounded context [16]. Careful identification and implementation of the

constituent microservices and combining those together are key to developing a full-fledged application. The change and redeployment can be restricted to only a single microservice to accomplish the desired update needed for a domain [17]. However, this architecture can also bring in its wake fresh challenges since an organization's core business processes are likely to encompass several microservices, thus obscuring visibility into an end-to-end process. Here adoption of Domain Driven Design (DDD) can prove beneficial as it strives to map business domain concepts into software using the domain model as the central concept [17].

DDD is a set of tools that aid in designing and implementing software delivering high value, both strategically and tactically [18]. Any organization needs to excel at its core business and the DDD strategic development tools facilitate making optimum software design and integration choices for the business. DDD promotes continuous refinement of domain understanding and has a natural fit to agile software development processes [19].

Adopting a distributed software architecture for splitting the business domain can pose certain challenges, which is why functional decomposition of an application and decentralized governance become important factors to consider in this context [15]. A particularly relevant design challenge is about "identifying the right kind of partition of the system into microservices" [20], as it can have repercussions on system performance [21]. As mentioned before, ideally each service needs to cater to one single responsibility [22], which would have a bearing on "the application's quality of service" [23] and the microservices count [15].

The celebrated book by Evans [17] identifies "the essential principles, activities and patterns required when adopting DDD". This facilitates depicting "the real world in the architecture", e.g. by using "bounded contexts representing organizational units" [24], while also focusing on the core domain, thus bringing about an all-around qualitative improvement in the software architecture [25].

Classifying the principles and patterns according to architectural concepts like architecture perspectives and architecture requirements can be of help while designing microservice architectures, and so would the mapping of DDD activities against an

established software development life cycle [19].

Designing The Architecture

While the framework provided by Vogel et al. for software architecture [26] has relevance in classifying microservices and DDD, the hexagonal architectural pattern introduced by Cockburn [27] can be adapted in DDD's context to structure a microservice-based application by segregating the concerns of a microservice in multiple layers in the resultant architecture [28]. The hexagonal architecture comprises the domain model, application services and adapters with ports (facilitating information exchange between microservice and clients). A client utilizes an exposed port along with its adapter for consuming the microservice.

Evans suggests a “four-layered architecture for DDD, consisting of the user interface, application, domain and infrastructure layers, to separate the domain from other concerns” [17]. This layering concept's influence on the microservice architecture can be understood as we look into the adaptation of the hexagonal architecture pattern [27]. As Vernon suggests, the hexagonal architecture and the onion architecture are the same [18]. The onion architecture, building on the hexagonal architecture, provides good clarity in terms of the representation of the fine-grained building blocks of a microservice's underlying structure [28, 29].

Activity Overview

During analysis, a knowledge crunching activity takes place with domain experts for the discovery of the information model (part of the domain model) with the possible creation of a prototype [19]. These two would influence each other given domain knowledge obtained respectively and thus enrich the other stream.

The feedback obtained from the prototype demonstration would help accomplish the following activities: refinement of the information model, discussion and finalization of the design ideas and application of design patterns [19].

Domain design, a critical activity, is part of the “Strategic Design” in DDD which deals with matters of strategic importance to the business, by dividing work by im-

portance and devising the most optimum integration mechanisms. As we'll elaborate later, a strategic design pattern called bounded context would be important in segregating the domain models. A Ubiquitous Language would be developed as part of the domain model within each individual bounded context. Domain design is similar to the system design activity described by Bruegge et al. [30]. The system is divided into subsystems that can be realized by individual teams using bounded contexts according to Conway's Law [24].

Subdomain is another concept relevant while delving deeper into strategic design. While useful for bespoke/new development, they are also key in dealing with legacy systems. Legacy systems are likely to be designed in a way divergent from the fundamental tenets of DDD advocating design with bounded contexts, thus inviting the term "Big Ball of Mud" by Vernon. The legacy system effectively becomes a jumbled entity housing a number of logical domain models enmeshed together. Treating each of those logical domain models as a Subdomain helps manage the complexity involved by effectively translating the problem space into one equivalent to that developed using DDD and multiple bounded contexts. Multiple Ubiquitous Languages may also need to be considered.

Context mapping (about integrating multiple bounded contexts) would be a key activity. As elaborated later, context maps define both team relationships and technical mechanisms that exist between two integrating bounded contexts.

Each bounded context is implemented as a microservice. The web APIs related to the microservices' entry points and the corresponding application logic would correspondingly undergo implementation and testing [19].

Developing a "deep model" to facilitate software development requires "exploration and experimentation" [17]. This implies that domain insights can be obtained throughout the software development lifecycle, which might lead to modification of artifacts created earlier.

This would become relevant in chapter 3 while creating an Essence based framework using a Domain Driven Design approach to address the Microservices lifecycle.

2.3 Petri net

We'd come across the need to decide on a suitable approach to model the flow of certain key processes of the CRM pre-sales functionalities. The increasing complexity of modern systems has implications in system development, and it makes eminent sense to make room for due considerations during the planning phase itself as invariably different constraints tend to come into play due to scarcity of system resources. Hence the design and operation of such a system would demand appropriate modeling and analysis. As modelling deficiencies can have telling effects on development time/cost as well as operational efficiency, the correctness of the models deployed during the planning phase becomes a key imperative. With the workflow being a recurring concept in many application domains, and processes tending to be a key part of that, it follows that the adoption of a framework suitable for modeling and analyzing workflow processes would be necessary [31].

The classical Petri net was invented by Carl Adam Petri in the sixties and has been used since then to model and analyze an array of processes in various application domains ranging from communication protocols and embedded systems to flexible manufacturing systems and distributed information systems, and is found particularly suitable to model workflows [32, 31, 33]. Petri net would be our model of choice given its usefulness in modelling, formal analysis and design of systems like the one in the current scope. While Petri net would be used to model the flow of the assignment process in our application domain (CRM pre-sales), an important objective would be to carry out the validation of our modelled system with regard to a specific set of properties. We'd leverage Petri net based behavioural analysis techniques to verify the suitability of our modelling. This would enable us to determine from a system designer capacity whether the desired functional properties of the system are present or not [34].

This initial understanding would help us in chapters 6 and 7 where we would use Petri net to model specific functionalities of CRM pre-sales in real estate.

2.4 Customer Relationship Management (CRM)

CRM helps companies stay connected to customers, streamline processes, and improve profitability [35]. Client centricity/accessibility fuels the growth story of CRM as organizations strive for real-time access to customer data to drive higher sales and improve customer experience with increased personalization. The share of CRM software, “the largest and fastest growing enterprise application software” [36], continues to rise.

CRM manages the customer interactions for a company by analyzing the customer’s history with the company to improve business relationships, focusing on customer retention and driving sales growth [37]. The systems of CRM compile data from a variety of communication channels, including a company’s website, phone/mail/chat, campaigns (both digital and non-digital) and social media [38].

As with other industry segments, a number of software offerings are available in the market today in the CRM space, addressing a host of functionalities and business scenarios. “*To exploit the significant market opportunity, product managers in CRM application providers should double down on cloud deployments and consider adding functionality in the fast-growing marketing segment,*” said Julian Poulter, senior director analyst at Gartner [36]. While the pros and cons of those CRM applications need to be carefully considered in the given context, there’s also the option of developing bespoke solutions. To make the right decision to achieve the desired benefits, we should be leveraging software engineering.

The prevalent challenges in CRM business and the fluidity of the market resulting in newer opportunities and constraints make a strong case for using software engineering to realize the desired effectiveness and efficiency in a time-bound manner. Ever-increasing complexity is a given and software engineering needs to address that by applying proven practices and attaining consistency.

2.4.1 CRM functionalities prevalent in real estate industry

“Business of real estate is not merely an operating necessity; it’s a strategic resource” [39]. “It is increasingly apparent that real estate can help organizations change or can prove to be a hindrance” [40]. As technology is redefining the way of transactions and managing things, business process automation has become a crucial driver for real estate companies to achieve excellence. Factors like “communication, improved service delivery, time to market and new avenues for growth are driving the real estate businesses to adopt technology” [41].

The real estate business can be quite complex involving multiple, often interlinked, processes. Let’s take a look at the major business processes that characterize this segment.

1. *Property Management*: Property search is usually technology enabled (e.g. via internet search). Real estate companies manage property listings online through web applications allowing online search [41].
2. *Automated Workflows*: These promote efficiency by sending regular notifications to staff / management which are configurable per requirements. Data can be consolidated and reports automated for sending to higher authorities.
3. *Lead Generation and Marketing for Properties*: Generating high-quality leads from various sources or campaigns is crucial. Campaigning is a useful marketing strategy in the real estate industry utilizing social media/email advertising to target potential clients. Website property listings help generate inquiries and qualified leads.
4. *Projects, Inventories and Assets*: Integrated online software enables project management and inventory management at multiple locations.
5. *Legal Documentation*: The agreements, sale deeds, etc. which are important from a compliance perspective for projects can be stored by digital document management.

6. *Vendors and Bill of Material (BOM)*: Tracking stock, inviting quotes from vendors, etc. can be made possible by the software. The accessibility of consumption status and BOM allows higher transparency and more informed inventory usage.
7. *Valuation and Advisory Services*: This is utilized for determining the property value. This entire process can be automated, facilitating decisions to rent, lease or sell assets.
8. *Analytics*: Correlating different data points allows for comprehensive analysis and decision making for a real estate business [41].

This understanding would be of use in chapters 4, 5, 6 and 7 where the automation of CRM pre-sales in real estate runs as a common thread.

An Essence based framework using a Domain Driven Design approach to address Microservices lifecycle from identification to implementation

Contents

3.1 Alphas for Microservices practice using DDD	48
3.2 Work Products and Activities for Microservices practice using DDD	55
3.3 Summary of chapter	63

To counter the digital disruption in today's world, the organizations are opting for digital transformation. While there is a marked propensity to turn towards microservices architecture to re-imagine the application landscape, there are also challenges involved. This is where applying proven practices through software processes can help us, as well as the adoption of the Domain Driven Design (DDD) approach.

A microservice is expected to realize only a specific business capability which stems from distilling the functionalities around a common theme resulting into a bounded context [16]. Careful identification and implementation of the constituent microservices and combining those together are key to developing a full-fledged application. However, this architecture can also bring in its wake fresh challenges since an organization's core business processes are likely to encompass several microservices,

thus obscuring visibility into an end-to-end process. Here adoption of Domain Driven Design (DDD) can prove beneficial as it strives to map business domain concepts into software using the domain model as the central concept [17]. We have provided an architectural overview for the same and the activities involved in section 2.2 of this thesis.

Here we would endeavour to extend the standard SEMAT Kernel to deal with different aspects of Microservices practice foundation leveraging DDD. We would identify the alphas characterizing this practice, specify the work products involved, describe associated activities and establish inter-linkages among the artifacts.

The subsequent sections would address the following:

- Alphas proposed for Microservices practice using DDD: here we would identify and propose the alphas as we model the Microservices practice using DDD in Essence;
- Work Products and Activities for Microservices practice using DDD: this is where we identify and describe the work products and the associated activities for the above-mentioned alphas as well as the inter-relationship amongst those elements within the process flow as part of the model building;
- Summary of chapter.

3.1 Alphas for Microservices practice using DDD

3.1.1 Alpha: Features

Features are artifacts addressing user or system requirements, with a bearing on the usability and value of the software system in question. The way of addressing the features would have ramifications on system architecture, data models and so on, right up to implementation. The requirements are specified with behaviour-driven development (BDD) [42] in the form of features. As a method of agile software development, BDD should specify a software system by referencing its behaviour. Thus

a feature describes a functionality of the application. The use of natural language and predefined keywords allows the developer to create features directly with the customer [43]. Analyzing the features leads to the initial domain model by deriving domain objects and their relationships.

During design, DDD should be applied based on the features defined with BDD. DDD's main focus is the domain and the domain's functionality, rather than technical aspects [44]. The central design artifact is the domain model, which represents the target domain.

The states of the Features alpha are defined as follows.

Formulated

The needs for the proposed system are articulated via an initial feature set.

Coherent

The features are clarified to the level of specific details, allowing us to form a consistent view of the essential characteristics of the proposed system.

Addressed

The extent of the feature set that has been addressed thus far is good enough to justify the motivation for a new system in an acceptable manner to the stakeholders.

Fulfilled

The entire collection of the feature set has been addressed in depth such that the needs for a new system are completely satisfied.

3.1.2 Alpha: Microservice

The primary alpha in this practice is the Microservice alpha which will advance from its identification to its deployment. Since a software system consists of microservice

sub-alphas, the progress and health of the former are dependent on each and every microservice.

The states of the Microservice alpha are defined as follows.

Foundational

This is about establishing the scope of the microservice. We would leverage DDD to reflect the customer's business domain into the intended application by the use of a domain model wherein all relevant information about the domain or business would be stored [45]. Once the requirements have been specified with BDD in the form of features, an initial domain model would be created from the features through the derivation of a tactical diagram (consisting of the domain objects and their interrelationships). This initial domain model contains the application's business logic and provides the semantic foundation for all the specified features.

Stratified

A key artifact of DDD, the bounded context, is formed here which is a potential candidate for a microservice. Knowledge crunching from DDD [17] is applied to gather the domain knowledge and the structure of the business. As an organization's domain knowledge may remain scattered across the whole business, analysing the business is essential to understand the business processes and the interaction of different departments. Various sources like domain experts, documents and organizational aspects should be considered to extract the domain knowledge, which should be leveraged to structure the domain and form the bounded contexts. Application analysis and business analysis considerations from [28] would lead to the bounded contexts [45].

A bounded context (1) exhibits high cohesion and low coupling, (2) can be managed by one development team, (3) ideally has high autonomy to reduce communication/coordination effort between development teams, (4) has a unique language that is not (necessarily) shared and (5) represents a meaningful excerpt of the domain [45]. Each of the established bounded contexts can be considered a microservice and requires or provides a unique interface for communication. When developing a

microservice-based application, a bounded context may not be so fine-grained to start with, thus being large enough to contain two or more microservices. It should be re-considered as it is preferable to have a one-to-one relationship. This facilitates the maintenance of the architecture through a clean mapping between bounded contexts, microservices and the development teams responsible.

Choreographed

Here the context map gets created that establishes the communication path between the bounded contexts. In other words, the candidate microservices are choreographed to realize the intended application functionality. An application consists of multiple bounded contexts, each having its own domain model and its own ubiquitous language based on the domain knowledge and acting as a contract for communication between project members and stakeholders. For the development of microservice-based applications, the multiple bounded contexts support the idea of a microservice architecture. The bounded contexts and their interconnections altogether constitute the entire domain knowledge of the application. Context map models the bounded contexts and their relationships [45].

For bounded contexts which are related to each other, teams may require inter-communication. This would entail assessing the extent of communication effort required between the said teams and establishing clear communication paths between those. This would drive the specification of dependencies and communication channels between teams and accordingly choosing a communication pattern based on [17, 18]. Communication patterns like Anti-Corruption Layer (ACL), Open Host Service and Published Language serve to minimize the communication between different teams, as well as the impact on interface changes. Adding those bounded contexts and communication relationships is an essential part of the context map. For instance, when communication between teams is not possible (such as when foreign services are adopted), DDD patterns like ACL should be applied. Finally, the relationships (including the pattern) and the bounded contexts are added to the context map diagram [45]. Aligning the context map to the customer's domain leads to a natural-

looking architecture [46].

Complete

Following the iterative process of analysis/design to implementation, models are implemented and tested as specific parts of the application get developed and new features are addressed in successive cycles. These features require analysis and may impact the domain model, potentially leading to the formation of new bounded contexts. Thus the models, including the bounded contexts and the context map, are refined according to the features and the knowledge crunching process in the previous steps. Effectively the microservice gets evolved to fulfil all its required interfaces with possible refinement of its structure.

3.1.3 Alpha: Domain Model

The domain model is central to DDD and treated as an alpha for the practice we are considering. It contains everything necessary to understand the domain. As collaboration with customers is essential to explore and model the domain, the first and recurring step of DDD is Knowledge crunching [17]. In tandem with customer discussions, the development team carries out the modelling activity and creates the domain model iteratively. A ubiquitous language is established, acting as the cross-team language. The creation of the domain model is strongly influenced by exploration and experimentation [17]. A judicious approach would be to refine the domain model repeatedly, if needed, to minimize implementation risk. Domain model creation is an iterative activity, aligned with the principles from agile development processes.

The Model-Driven Engineering (MDE) [47], describing an approach used to effectively express domains in models, is relevant in this regard. The model-driven architecture (MDA) framework [48], which supports the implementation of MDE, suggests three steps that would be required to progress an application from abstract design to implementation. Three models are created accordingly which bear relevance

to our subsequent discussion. Those are as follows [19]:

- computation independent model (CIM): provides domain concepts without any technology consideration,
- platform independent model (PIM): enriches CIM with computational aspects; and
- platform specific model (PSM): enriches PIM with implementation specifics for the chosen technology platform.

The states of the domain model alpha are defined as follows.

Initiated

This is about creating an initial domain model. The goal of DDD is to align the customer's business domain with the intended application. The domain model is instrumental in achieving the said goal by storing all relevant information about the domain or business, aided by the usage of the ubiquitous language [45]. Once the requirements are specified with BDD in the form of features, an initial domain model would be created from the features through the derivation of a tactical diagram (consisting of the domain objects and their inter-relationships). This initial domain model contains the application's business logic and provides the semantic foundation for all the specified features. A domain model, as per Evans, contains everything necessary to understand the domain [17] and goes beyond what we traditionally understand by a domain model (connected to a formalized model using UML [49]). The connotation is similar to the term information model introduced by Fairbanks [50] to distinguish between the two concepts. The model here corresponds to the computation-independent model (CIM) in MDA.

Foundational

If the domain structure is not sufficiently clear even after addressing several features, more features need to be considered until the domain model appears to be meaningful.

Subsequently, that domain model is examined and structured into several bounded contexts [45].

The focus of DDD practice is on an intended application [17] to ensure the “perfect fit” of the gathered information, called “domain knowledge,” for the application. This domain knowledge is captured in domain models. An application consists of multiple bounded contexts, each with its own domain model. To effectively assimilate the collective business knowledge (which can be of significant proportions), the domain knowledge is split into multiple domain models. The validity of each domain model is limited through the bounded context. Of the two types of “design activities” introduced by DDD [17], the first is of relevance here. It is called the “strategic design”, with tasks in modelling and structuring the domain’s macro architecture (e.g., departments are used to define boundaries). The resultant domain model can be regarded as a platform-independent model (PIM) equivalent as per MDA parlance [19].

Implementable

Here a context map is developed as the domain model gets refined further. The second of the two types of DDD “design activities” [17], the “tactical design”, is of relevance here. It refines the macro architecture even further and enriches the bounded contexts with domain knowledge. This activity represents the micro-architecture of the domain and that of the microservice.

By now the development team exploring the customer’s domain has created the context map and domain model. Both the strategic and tactical designs are completed at this stage, whereby the context map should get integrated at this point [45]. To enable further downstream implementation, platform-specific implementation details need to be added and the ensuing domain model can be regarded as a platform-specific model (PSM) equivalent as per MDA parlance [19]. It enriches the PIM equivalent domain model created previously with implementation specifics of the chosen technology platform.

Evolved

In iterative development, implementation of the models takes place along with testing as specific parts of the application are developed. Following the iterative process, new features are implemented into successive cycles. These features would require analysis and may impact the domain model, potentially leading to new bounded contexts being created. Thus the models, including the bounded contexts and the context map, are refined according to the features and the knowledge crunching process in the previous steps.

3.2 Work Products and Activities for Microservices practice using DDD

This section deals with the work products and the associated activities for the three respective alphas that we have proposed in the previous section.

3.2.1 Work Products and Activities for the Feature alpha

The Microservices practice using DDD has the following work products for the Feature alpha:

- Scenarios,
- UI/UX (User Interface/User Experience) Design Prototype, and,
- Test Cases.

Figure 3-1 represents a model of the Microservices practice using DDD, showing the relationships between the elements (alpha, work products and activities) in the practice corresponding to the Feature alpha, the activity flow and the relationships with kernel elements (e.g. Features and Requirements).

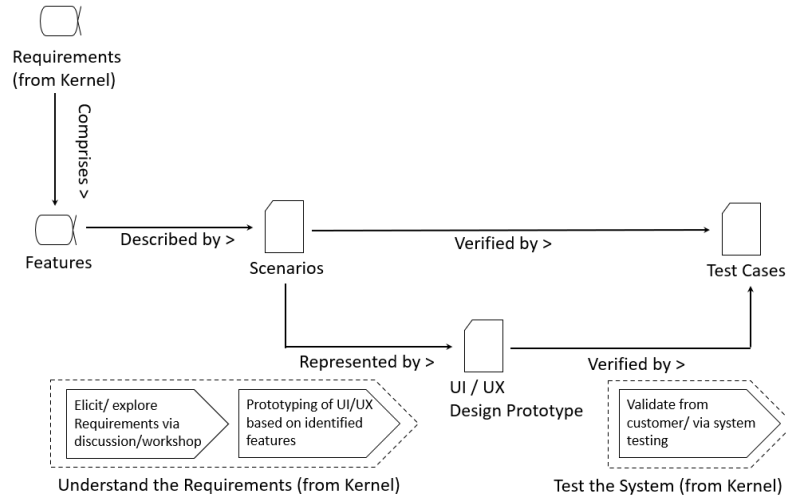


Figure 3-1: The Essentialized model showing the work products and the activities for the Feature alpha

Work Product - Scenarios (describing the Feature alpha):

Requirements in BDD are elicited by developers as well as application users. The shared understanding is captured in an executable requirement specification, leveraging the ubiquitous language known from DDD [43].

This requirement elicitation activity is carried out in an outside-in fashion [9], extracting the most visible behavior which is subsequently implemented. During implementation, new details are discovered, leading to unearthing further requirements or refining existing ones. Requirements, captured as features, are broken into granular artifacts called scenarios [9], described using the formal language Gherkin and having the following levels of detail:

- Each scenario consists of an initial context, event and expected outcome, referred to as steps.
- Each of the above three steps starts with a predefined Gherkin keyword, which is necessary to execute the tests in later stages.

Work Product - UI/UX Design Prototype (describing the Feature alpha):

Knowledge crunching is essential to obtain an understanding of the domain. Features/scenarios enable capturing the application requirements and provide a useful abstraction of the domain logic to be modelled in the domain model.

The design prototype, useful in assessing design concept feasibility, would be predicated on features identified. The design prototype being considered here would be specifically oriented to the User Interface/ User Experience (UI/UX) aspects of the application. This also opens up opportunities for domain model discovery/validation via client interaction with the UI.

The associated activity, “Prototyping of UI/UX based on identified features”, is iterative. Brainstorming sessions for the iterations deal with the design ideas and their feasibility given the constraints/boundary conditions. The accumulated design ideas are implemented to refine and build the design prototype [28]. Client inputs are assessed along with internal improvement recommendations to decide on the design changes forming part of the subsequent iteration’s scope. The iterations are concluded upon satisfactory closure of all client requirements.

Work Product - Test Cases (describing the Feature alpha):

BDD is conceptualized on test-driven development (TDD) and automated acceptance tests to verify the correctness of the application [43, 51]. The scenarios in the features are broken down into steps, starting with a predefined Gherkin keyword to help test execution in later stages as test cases are created.

The requirement specification becomes a “living documentation” [43] via the traceability of features to implementation and testing activities. The validation of the test cases happens via the activity “Validate from customer/via system testing” as the validation counterpoints happen to be the (internal) system testing and the customer/user acceptance testing.

3.2.2 Work Products and Activities for the Microservice alpha

The Microservices practice using DDD has the following work products for the Microservice alpha:

- Microservice Design,
- Microservice Build and Deployment Script, and
- Microservice Test Cases.

Figure 3-2 represents a model of the Microservices practice using DDD, showing the relationships between the elements (alpha, work products and activities) in the practice corresponding to the Microservice alpha, the activity flow and the relationships with kernel elements (e.g. Microservice and Software System).

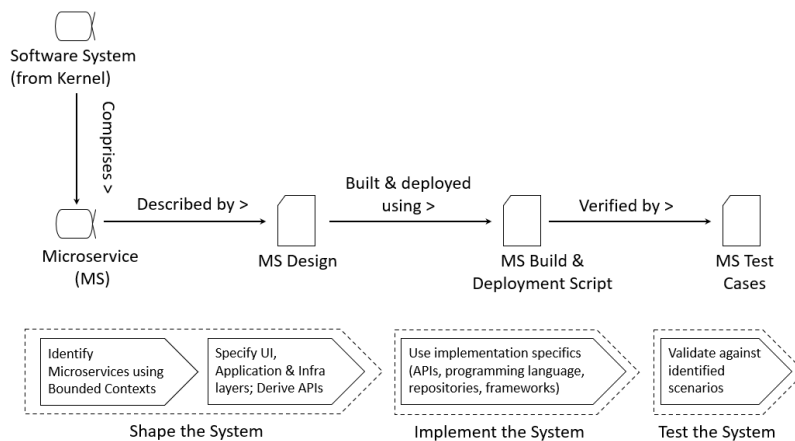


Figure 3-2: The Essentialized model showing the work products and the activities for the Microservice alpha

Work Product - Microservice Design (describing the Microservice alpha):

This captures microservice design aspects like behavior, internal design and interfaces. The activity to “Identify Microservices using Bounded Contexts” is key since bounded contexts with proper granularity, would be candidates for microservices on a 1:1 basis. Related activities to specify UI, Application and Infrastructure layers and subsequent

derivation of APIs would be carried out to complete the design. This work product comprises the following levels of detail:

- Scope of a microservice is articulated to specify the behavior.
- The elements within the microservice are next specified which would constitute the more detailed internal structure, facilitating coding activities.
- Scope of a microservice in terms of its interfaces is specified.

Work Product - Microservice Build and Deployment Script (describing the Microservice alpha):

This is about realizing the design by using the implementation specifics (like programming languages, APIs, databases or repositories, any framework that might be used, and so on). Automation would be leveraged to enable the production and deployment of each microservice in a repeatable fashion. This work product comprises the following levels of detail:

- Initially there would be a plan to attain the said objectives, breaking it down into steps without an actual implementation in terms of scripts.
- Then the actual script is constructed and its workability in the development/deployment instance is ensured.
- Subsequently the efficacy of the script in the eventuality of microservice upgrades (which may be a continuous process) is ensured so that other elements (like other microservices) are not affected.

Work Product - Microservice Test Cases (describing the Microservice alpha):

Testing of a microservice starts with identifying scenarios. Given a microservice's execution dependencies, it might necessitate stubbing out the associated dependencies. This work product comprises the following levels of detail:

- First the scenarios for the microservice usage are listed in priority order.
- Test case scope is established with stubbing of dependencies if needed.
- The scripting and automation of the test cases are accomplished, which are to be executed as part of the build and deployment process.

3.2.3 Work Products and Activities for the Domain Model alpha

The Microservices practice using DDD has the following work products for the Domain Model alpha:

- Domain Views,
- Bounded Contexts, and
- Context Map.

Figure 3-3 represents a model of the Microservices practice using DDD, showing the relationships between the elements (alpha, work products and activities) in the practice corresponding to the Domain Model alpha, the activity flow and the relationships with kernel elements (e.g. Domain Model and Software System).

Work Product - Domain Views (describing the Domain Model alpha):

To simplify the modeling and make it more structured, modelling the various domain aspects via different diagrams is encouraged, and “domain views” are utilized in that context [44, 28]. Accordingly, the domain model would comprise different domain views (containing domain objects) that are assigned to a specific domain view type. This domain view type is used to determine the domain objects and identify the possible representation (say with UML diagrams). A type of domain view is specified through one or more stakeholders.

A domain view may be of two types, having the following levels of detail:

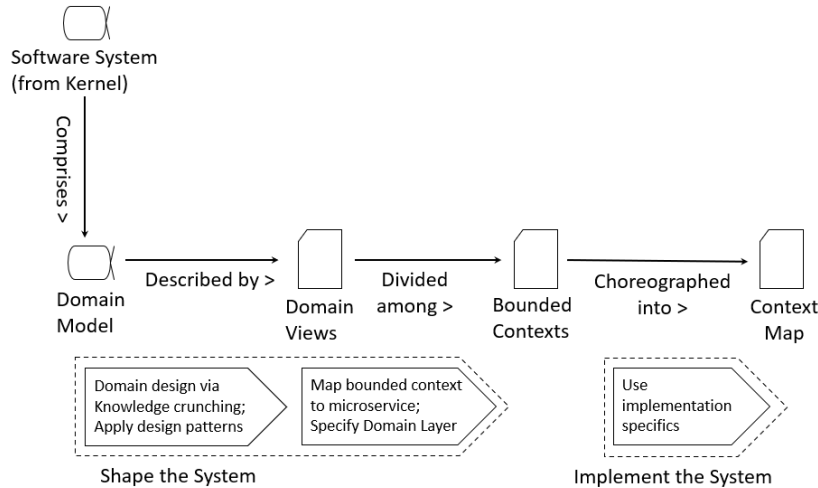


Figure 3-3: The Essentialized model showing the work products and the activities for the Domain Model alpha

- the “relation view” addressing the static behavior of the domain. This would be utilized for modelling domain objects with emphasis on their (real-life) inter-relationships [28].
- the “process view” addressing the dynamic behavior of the domain. This would be utilized for modelling the processes of the domain, as the focus would be on domain objects’ interactions [28].

The associated activities are “knowledge crunching” to create the domain design and apply the design patterns, which are essentially iterative in nature.

Work Product - Bounded Context (describing the Domain Model alpha):

As “knowledge crunching” [17] is applied to gather the domain knowledge and the structure of the business, such “application business analysis considerations would lead to the bounded contexts” [45]. ”A bounded context (1) exhibits high cohesion and low coupling, (2) can be managed by one development team, (3) ideally has high autonomy to reduce communication/coordination effort between development teams, (4) has a unique language that is not (necessarily) shared and (5) represents a meaningful excerpt of the domain” [45].

Each of the established bounded contexts is a potential microservice requiring or offering a communication interface. Efforts should be taken to make a bounded context fine-grained enough to contain one and only one microservice, such that it results in a clean mapping amongst the bounded contexts, microservices and the respective development teams in charge which can help maintain the architecture over time. The crucial activity here is to map the bounded contexts to microservices.

In a DDD project, there would be multiple bounded contexts in play - one of those being the Core Domain. There will also be various Subdomains in other bounded contexts. With DDD strategic design, the most optimal modeling composition would result in one Subdomain per bounded context and vice versa. Sometimes there might be multiple Subdomains in one bounded context, but it's a sub-optimal modeling outcome [52].

A Subdomain is a sub-part of the overall business domain. Subdomains help logically break up the whole business domain to facilitate understanding the problem space on a large, complex project. It is also a clear area of expertise.

Layers usually found in a bounded context would be “Input Adapters (UI controllers, REST endpoints and message listeners); Application Services orchestrating use cases and managing transactions; and Output Adapters such as persistence management and message senders” [52].

Work Product - Context Map (describing the Domain Model alpha):

The bounded contexts with their interconnections represent the application's domain knowledge, as context map models the bounded contexts' relationships [45]. Regarding the context mapping activity, understanding inter-team relationships and integration between bounded contexts are important. Clear separation and defined contracts between those make change management easier.

Teams in charge of interdependent bounded contexts need proper communication amongst themselves. The extent of communication efforts needed between them as well as clear inter-communication paths need to be established. This drives the specification of dependencies and communication channels between teams, eventually

helping to select a communication pattern based on [17, 18]. Communication patterns like Anti-Corruption Layer (ACL), Open Host Service and Published Language help minimize inter-team communication as well as the interface change repercussions. Adding those bounded contexts and communication relationships is an essential part of the context map. For instance, when communication possibilities between teams may not be very apparent (cases involving the adoption of foreign services), DDD patterns like ACL can be useful. Finally, the relationships (including the pattern) and the bounded contexts would be used to embellish the context map diagram [45]. The type of interface supplied to enable integration with a given bounded context is predicated on what the owning team of the said bounded context provides (e.g. RPC via SOAP or messaging interface using queues). If database integration is unavoidable, it is advisable to ensure consuming model isolation via ACL [52].

While carrying out the activity of using implementation specifics (like mapping the domain objects into classes for an object oriented programming context), the alignment of the context map to the customer's domain needs to be maintained; consequently, a natural-looking architecture is realized [46].

3.3 Summary of chapter

A Domain Driven Design (DDD) approach can help organizations reap the full benefits of adopting microservice architecture. We have leveraged Essence in our endeavour to develop a framework to address the microservices lifecycle leveraging DDD. We have identified and proposed three alphas [53] and the associated work products and activities, depicting their inter-relationships within the process flow [54]. We hope this can act as an aid in the successful adoption of related practices in the industry.

While this is completely industry agnostic, we would now focus our attention on a specific industry scenario, the automation of CRM pre-sales for the real estate industry, for the remaining parts of this thesis. We would endeavour to create an agile approach using Scrum and Essence to address project development in the said industry scenario in the next chapter.

An agile approach to automate Real Estate CRM (pre-sales) using Scrum and Essence

Contents

4.1 Business Process Modelling for in-scope CRM pre-sales functionalities	66
4.2 Adoption of Essence and Scrum	69
4.3 Summary of chapter	73

Agile transformation seems to be in use all around us but according to Jeff Sutherland, co-founder of Scrum, a significant number of Scrum implementations tend to run into issues. At least part of the solution, according to Sutherland, can be found in the Essence framework as he goes on to underline the value of Essence by pointing out Essence as “the key to success” in his blog ‘Better Scrum with Essence’ [55].

Here we talk about the automation of the pre-sales function of real-estate CRM in an agile way while leveraging Essence by presenting the journey of a representative real-estate organization named *Nirmanik* to address real-life industry problems.

To understand the key concepts of Essence one can refer back to section 2.1 of this thesis. The domain/industry understanding is provided in section 2.4 of this thesis.

We have the subsequent sections to address the following:

- Business Process Modelling for in-scope CRM pre-sales functionalities

- Adoption of Essence and Scrum
- Summary of chapter

Let us now discuss the imperatives facing a real-estate organization, *Nirmanik*, as we attempt to address the requirements to automate the pre-sales Management system and analyze the business processes using a modelling approach.

4.1 Business Process Modelling for in-scope CRM pre-sales functionalities

4.1.1 Background

A large conglomerate, as part of its diversification strategy, had ventured into real estate to take advantage of the favourable market conditions and synergy with its other affiliated operations and spun off the *Nirmanik* organization. It had recruited people with knowledge of real estate business in key positions to execute the plans to kickstart the real estate venture. While the operation started in a predominantly manual mode, a clear need was felt to automate the processes. The conglomerate had an in-house IT team with a track record of catering to many of the affiliated organizations' IT initiatives and hence a good knowledge of the prevailing software stacks and experience of collaboration. An empowered team was formed consisting of key people from the real estate business, key people from the IT team and specific representation of the top management stakeholders.

The designated team went about evaluating the existing CRM products in the market. Features of some license-based products mapped favourably to the required functionalities but operating expenditures like yearly license renewal were high as licenses were needed for all the telecallers, sales executives and the management team. Other products would require significant customization for extending the product features to realize desired functionalities, resulting in a sizeable one-time implementation cost on top of recurring licensing costs. None of these seemed to be an attractive

proposition.

After a lot of deliberation, the designated team decided to recommend leveraging the conglomerate's in-house IT team to custom build a solution to address the requirements. Based on the team's recommendations, the management approved an internal project to engage the in-house IT team along with the business experts to build a home-grown software solution on cloud using a standardized software stack to address the business requirements in an agreed time window. While there would be a one-time investment for implementation cost, the yearly operating expenditure should be reasonably low since there would be no recurring product licensing costs involved.

4.1.2 Broad Scope

As part of the pre-sales automation scope for *Nirmanik*, the focus would be on the following business processes:

- *Property Management*
- *Lead Generation / Marketing for Properties*
- *Automated Workflows*

Let us now model the above functionalities and process flow using appropriate methodologies.

4.1.3 Functionalities to be Modelled

Nirmanik had put up a company website with its property listings to solicit enquiries from prospective customers. A team of tele-callers would sift through those inquiries as well as scout the leads from online real estate portals or social media after those are de-duplicated (removing potential repetitions), and call up the promising prospects. The leads finding response, termed "opportunities", would next be allocated (following some algorithm) to a team of Sales Executives for subsequent site visits. The Sales

Executives would meet the prospective customers at the appointed time to carry out actual site visits. Subsequent interactions would lead to either a showing of intent from the prospect eventually culminating in a sale of property (booking) or lost opportunity. Handling of the sale of property would be done via the sales / post-sales process which would be beyond our scope.

4.1.4 Process Modelling

We have captured the workflow of real-estate pre-sales CRM functionalities in scope using the “high and low communication flow orientation diagrams” [56] in Fig. 4-1 and Fig. 4-2.

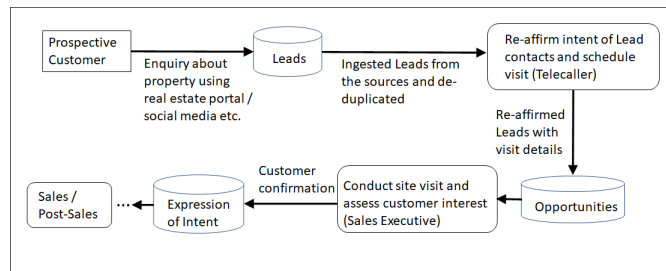


Figure 4-1: Communication Flow of real-estate pre-sales CRM

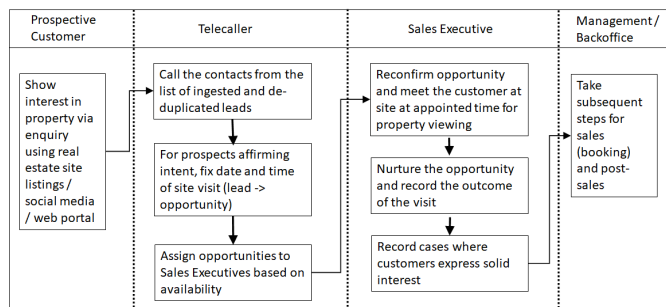


Figure 4-2: Functional Flowchart of real-estate pre-sales CRM

Let us now narrate how the real-estate organization, *Nirmanik*, adopted Essence and Scrum to address their business challenges.

4.2 Adoption of Essence and Scrum

To understand how Essence and Scrum come together to create a strong value proposition, it is important to note that Essence doesn't propose any fundamental change to the actual Scrum content but allows for an enriched presentation of the same to teams. As popularized in the experiment conducted by Sutherland [55], a key aspect of Essence is the usage of cards to represent the main concepts of any practice, such as the principles, roles, activities and work products.

Thus a pragmatic approach to achieve project success would be for the “Agile mindset” and Essence to go hand in hand facilitated by the “Alpha State Cards” [57] (shown in Fig. 4-3).

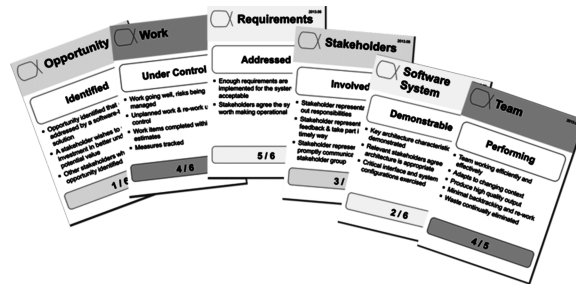


Figure 4-3: Alpha State Cards

Most teams using Scrum would track the state of fine-grained things like individual Product Backlog Items or Improvements. While Essence offers to help by clearly delineating the states they go through, the team may still miss the forest for the trees by focusing too much on the finer details and missing the big picture in the process until it's too late.

Essence comes to the rescue by pointing out “the essential things to track as Alphas, each with a defined life-cycle of states and a checklist for each state”. An alpha is “an essential element of the software engineering endeavor; one that is relevant to an assessment of its progress and health.” [9].

Essence prescribes certain key Alphas that are universally applicable irrespective of the practices considered, as they enable a team to figure out where they are in the big picture. Let's map out the states of the Essence Kernel Alphas with regard

to the development endeavour undertaken by the *Nirmanik* team, based on certain assumptions regarding the Essence kernel alpha states for the *Nirmanik* project.

4.2.1 The Kernel Alpha states of the development endeavor undertaken by the *Nirmanik* Team

We would use the convention of adding *Nirmanik::* prefix to the *generic artifact name*, with the resulting name put in *italics*, to distinguish artifacts used in the *Nirmanik* project context from the generic Essence artifacts.

Let's start with alphas in the *Nirmanik::Customer* area of concern and map their states.

- *Nirmanik::Opportunity* - Value Established

The *Nirmanik* management had already approved an internal project to address the business requirements. The in-house IT team would be engaged along with business experts to build a home-grown software solution on cloud using a standard software stack.

- *Nirmanik::Stakeholders* - Involved

External stakeholders would include the top management of the *Nirmanik* organization who had shown a willingness to fund the new software effort. Internal stakeholders would include the development team.

Next, we would be considering the alphas in the *Nirmanik::Solution* area of concern.

- *Nirmanik::Requirements* - Coherent

The requirements for the proposed *Nirmanik* software solution that had to be built were clarified and the internal algorithms were well-articulated.

- *Nirmanik::Software System* - Architecture Selected

Nirmanik had discussed and budgeted for the software solution to be built and the associated fees. The team had identified the sources for leads (company

website, real estate portals, social media feeds, etc.), obtained interface specifications and decided on the specifics for triggering the alerts in the workflow.

Finally, we talk about the alphas in the *Nirmanik::Endeavor* area of concern.

- *Nirmanik::Work* - Prepared

There was evident clarity regarding which all stakeholders were funding this work. The team had decided to utilize a backlog stemming from the requirement set drawn up for planning purposes. They also decided to keep a regular communication channel open with their stakeholders to ensure the relevance of their backlog.

- *Nirmanik::Team* - Performing

While the development team's IT personnel had different skill sets on an individual basis (like the front-end technologies, the backend technologies, and integration skills), collectively they possessed the know-how of the full software stack needed for development. The IT people used their previous shared experience of implementing IT initiatives of the parent conglomerate, while the wider team leveraged their experience of undertaking the joint exercise to recommend appropriate products.

- *Nirmanik::Way of Working* - Foundation Established

The team continually made sure to understand the scope of each requirement item by maintaining regular touchpoints with stakeholders. They routinely scrutinized the way of working and made changes if necessary. The team had established a development/test environment and agreed on the software stack to be used [2].

The *Nirmanik* team members had their own ideas of which industry-related practices they'd like to select. While the team would brainstorm later to select the key practices for their development process, they decided at the outset to utilize the kernel extensively to apply the essence of software engineering and adopt Scrum on top of it. The following subsection provides an elaboration of the same.

4.2.2 Adoption of Scrum within the Essence framework

“A practice expressed in the Essence language provides explicit guidance to teams by clarifying the things to work with, the things to do, the required competencies and patterns” [2].

If a team’s starting point is a full-scale prescriptive ‘framework’ and then it keeps on making decisions as to which elements need to be left out, there’s a likelihood of adopting more than what’s needed. An inexperienced team may become saddled with unnecessary techniques at the outset that way or burdened with unnecessary process overhead. On the contrary, starting with the essential and then adding one practice as required for a specific area is a better strategy, which is in line with what Essence recommends.

The *Nirmanik* development team had to decide on which specific practices to be adopted on top of the kernel. They held brainstorming sessions to select the key practices and decided to opt for iterative development in the agile way as the agreed approach of work, settling on Scrum.

Alphas are key to assessing progress and health in a development endeavor per Essence. “The kernel calls out universal alphas explicitly, while practices call out practice-specific alphas” [2]. The practice specific alpha-s for the Scrum practice are the sprint and the Product Backlog Item (PBI) alphas.

By definition, a sprint is a given interval of time whereby completion of some useful work is achieved. States of the sprint alpha would be: Scheduled, Planned and Reviewed. As teams address work items in a backlog, the work items, known as Product Backlog Items (PBIs), are considered to be alphas as well. States of the PBI alpha would be: To Do, Ready, Doing and Done.

The team would split up the whole endeavor into iterations to help build the software in increments, leveraging Scrum. This would entail the following activities: Sprint Planning, Daily Scrum, Sprint Review and Sprint Retrospective [2].

During sprint planning, the team along with the product owner (PO) would select the PBIs for the sprint and agree on the prioritized ones for inclusion in the upcoming

sprint. Each day, a brief team meeting (daily scrum or daily standup) is held to coordinate the work and plan for the upcoming day. Each team member explains the work accomplished since the last meeting, the current plan, and any obstacle in meeting the goal. This meeting does not discuss solutions to problems - a separate meeting for that may be convened. A sprint review activity is held at the end of the sprint with key stakeholders to review the product in its current state. The stakeholders might recommend certain product improvements that will get added to the product backlog. The sprint retrospective, conducted at the end of each sprint, provides an opportunity to figure out improvements in the way of working going forward.

There are three key roles: the PO, the Scrum master and developers [2]. These roles are represented as patterns in the Essence language; Sprint Planning, Daily Scrum, Sprint Review, and Sprint Retrospective are activities; while Product Backlog, Sprint Backlog, and Increment are work products [2].

Essence uses cards to represent the main concepts of a practice. There is a set of cards that describe Scrum. They facilitate a tactile way to get familiarized with the ideas, especially via game play with a team, and help drive the discussions within the team. Typically they have some annotation summarizing each concept. Making this information accessible to the team acts in setting up quick reminders and provides ground truths around the Scrum framework. The process thus is kept living and breathing within the context of the usual workflow.

4.3 Summary of chapter

We have shown how a representative real-estate organization has gone about automating their pre-sales CRM function in an agile way leveraging Scrum within the Essence framework after we have done the business process modelling for in-scope CRM pre-sales functionalities.

However, additional practices would be needed to carry out specific software development activities like requirement elicitation or deployment. For example, since

the solution area is not addressed by Scrum, the team would require explicit guidance there. We would be showing subsequently the adoption of the User Story practice to address the requirements area and the microservices practice to develop and deploy the software system being built with regard to this endeavour.

Extending Essence to adopt User Story and Microservice practices leveraging Scrum to automate pre-sales function of real estate CRM

Contents

5.1 Context	78
5.2 Considerations for composing practices into method using Essence	79
5.3 Adoption of User Story practice	81
5.4 Adoption of Microservice practice	89
5.5 Summary of chapter	98

Here we would be extending the exercise to automate the pre-sales CRM function of the real-estate organization, *Nirmanik*, leveraging Essence as we would adopt two practices, User Story and Microservices, into the method being constructed using Essence.

As far as the domain/industry is concerned, section 2.4 of this thesis provides an understanding of the background. For the key concepts of Essence, section 2.1 may be referred to.

The IT organization structure of a real estate company has a strong bearing on its technology adoption. Typically, a real estate company tends to fall into one of the

following three categories that are applicable to all non-IT companies in general:

C-I: While there is a clear IT strategy present for the organization, it may not have a support IT structure, or even if it does, that IT structure is not equipped to implement the IT strategy on its own. So it is necessary for that organization to hire external consultants and/or IT vendors to implement that IT strategy. If a support IT structure does exist for such an organization, it may not be capable of executing projects by itself, and would likely find it difficult to embrace the latest technology stack/programming paradigms. However, its personnel might possess the skills to interact with external consultants / IT vendors for successful project implementation and can possibly take over the support and maintenance post go-live. Joint application development might also be a possibility.

C-II: This organization is characterized by the presence of a mature in-house IT division capable of executing project(s) (which can involve greenfield development or package solution adoption) on its own. The IT division also has the expertise and maturity to embrace the latest technology stack/programming paradigms.

C-III: IT support is almost non-existent in this case, with the organization just being able to stay afloat with no significant investment in IT resources forthcoming.

Nirmanik belongs to category C-I above. It has an in-house IT team that would need prescriptive guidance to translate the IT vision into reality, necessitating the involvement of external IT consultants to provide the necessary expertise and facilitate methodology adoption.

As we made Essence central to *Nirmanik*'s endeavor as a framework while leveraging Scrum, it became clear that we would need to induct additional practices to address the four consecutive kernel solution activity spaces: "Understand the Requirements", "Shape the System", "Implement the System" and "Test the System",

which are key to the software development lifecycle spanning requirement analysis, design, development and testing. The primary aim here would be to articulate our accomplishment in leveraging Essence to help *Nirmanik* address the said kernel solution activity spaces by adopting the User Story and the Microservice practices, and detail those out along with the alphas and work products associated with them as part of their essentialization.

While the User Story and the Microservice practices are both quite well established in their individual capacities, the value addition here would be to weave them into a method leveraging Scrum after essentializing them as guided by the Essence framework. The resulting method can also act as a guidance for similar project implementation endeavors in a variety of industry/domain settings where these practices are deemed to be good choices, assuming the organization in question either falls in the aforementioned category C-I requiring help from external IT consultants / IT service providers or belongs to category C-II by virtue of having a robust internal IT organization. Organizations belonging to category C-III would not qualify for the reasons already stated.

There is an even wider context where it might be possible to extend our work, whereby the exercise carried out here can act as a blueprint for a variety of industry/domain settings where the practices selected (or found suitable) happen to be different from the ones used here. Leveraging the principles of our work here and keeping Essence as the central framework, those practices can be essentialized and composed into a method to realize the objectives for the industry/domain setting under consideration.

We have the subsequent sections to address the following:

- Context
- Considerations for composing practices into method using Essence
- Adoption of User Story practice
- Adoption of Microservice practice

- Conclusion and future direction

5.1 Context

The industry being considered is the real estate business where technology driven automation has been making a real difference.

The real-estate organization, *Nirmanik*, has put up a company website with its property listings to solicit enquiries from prospective customers apart from traditional advertisement channels like billboards and hoardings. The inquiries from these channels, online real estate portals, social media and call inquiries would be scouted after being de-duplicated (removing potential repetitions), and the list of these potential customers i.e. the *leads* (resorting to the customarily used term) would be called up.

The status of a lead who is getting ingested into the system for the first time (Fresh Lead) would be one of the following:

- Guest Lead: Any Fresh Lead who wants to visit but is yet to confirm a visit date. It is expected that a telecaller would try calling up the lead a certain number of times to fix the visit date to convert the lead to a Prospective Lead.
- Prospective Lead: Any Fresh Lead who shows a clear intent to visit right away by providing a visit date.

In response to the request by the Prospective Lead, an assignment process would be carried out to match the request to an available Sales Executive to arrive at a mutually acceptable time slot for the requested site visit date.

The Sales Executives would meet the prospective customers at the appointed time to carry out actual site visits. Subsequent interactions would lead to either a showing of intent from the prospect eventually culminating in a sale of property (booking) or lost opportunity. Handling of the sale of property would be done via the sales / post-sales process which is beyond our scope.

While the key practices for *Nirmanik's* development process were yet to be selected, it was decided at the outset to leverage Essence as the framework. The states

of the Essence Kernel alphas were mapped with regard to the development endeavor undertaken by *Nirmanik*, based on certain assumptions regarding the Essence kernel alpha states for the project, as shown in Table 5.1. [3].

Table 5.1: The states of the Essence Kernel alphas mapped with regard to the development endeavor undertaken by *Nirmanik* [3].

Areas of concern	Alpha	Alpha state
<i>Nirmanik::Customer</i>	<i>Nirmanik::Opportunity</i>	Value Established
	<i>Nirmanik::Stakeholders</i>	Involved
<i>Nirmanik::Solution</i>	<i>Nirmanik::Requirements</i>	Value Conceived
	<i>Nirmanik::Software System</i>	Architecture Selected
<i>Nirmanik::Endeavor</i>	<i>Nirmanik::Work</i>	Initiated
	<i>Nirmanik::Team</i>	Performing
	<i>Nirmanik::Way of Working</i>	Foundation Established

We would continue to follow the convention of adding *Nirmanik::* prefix to the *generic artifact name*, with the resulting name put in *italics*, to distinguish the *Nirmanik* artifacts from the generic Essence artifacts.

A successful progression of the *Nirmanik* development endeavor would be reflected in the advancement of the above-mentioned states of the Essence Kernel alphas. It clearly would become an imperative to decide on the best practices to adopt, culminating in the selection of key practices for the development process while continuing to have Essence as the framework while leveraging Scrum. The next section would talk about the considerations that would apply to compose practices to arrive at a method using Essence.

5.2 Considerations for composing practices into method using Essence

A method exists to facilitate the progress of software development teams by providing guidance on things to be done during the development process; in other words, providing access to the necessary practices. Methods are compositions of practices,

and as the number of practices increases, possible combinations of those can increase significantly. It thus follows that while methods and method variants abound in the world, the instances of reusable practices are much less.

A key value of Essence is in its ability to act as a common ground and to provide guidelines for all practices, thus providing clarity in understanding and subsequently enabling appropriate modification of practices. Essence provides a language and a kernel of software engineering, and facilitates the comparison of practices described using the same common ground [2].

While this allows the practices contributed by the software engineering community to be composed in different ways as necessary to form methods, those practices need to be essentialized first i.e. they need to be described using Essence (the kernel and the language) and as a result, the methods composed by using those practices would also be essentialized. Essentialization ensures that the descriptions of the method/practice are detailed to the basics. The value addition is realized by libraries of practices created out of many different methods. Thus we can mix and match practices selected from a shared Practice Library such as one illustrated in Fig. 5-1(a) to obtain a method that suits a project context for organizations like *Nirmanik* which belong to category C-I as well as those belonging to category C-II. While the starting point involved the Essence kernel as the base, we would go on to select a number of practices that would constitute *Nirmanik's* way of working. The set of practices thus selected, along with the kernel, would become the method for *Nirmanik*, as shown in Fig. 5-1(b) and described in detail later.

If a team's starting point is a full-scale prescriptive 'framework' and then it keeps on making decisions as to which elements need to be left out, there's a likelihood of adopting more than what's needed. An inexperienced team may become saddled with unnecessary techniques at the outset that way or burdened with unnecessary process overhead. On the contrary, starting with the essential and then adding one practice as required for a specific area is a better strategy, which is in line with what Essence recommends. This is very much applicable for organizations in category C-I, where the external IT consultants or the vendor IT organizations entrusted with the project

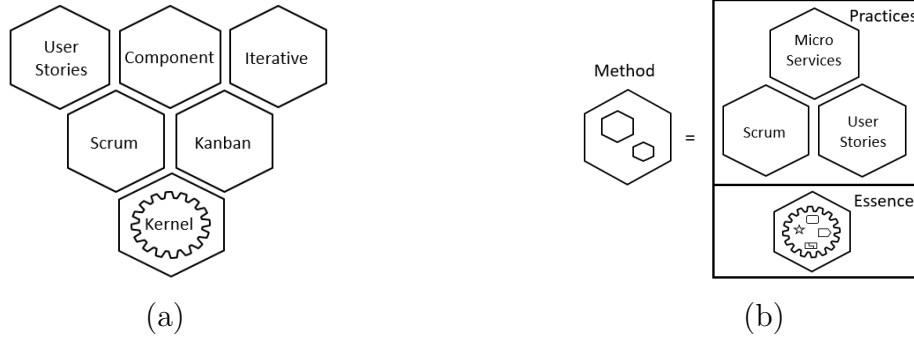


Figure 5-1: (a) An example of a shared Practice Library. (b) Method: composition of practices on top of the Essence kernel and language (for *Nirmanik*)

implementation should be well versed with this methodology and can facilitate the composition of the method based on the practices chosen after essentializing them. This can also be extended to the organizations in category C-II as they undertake project implementations on their own, because even though they have good experience/knowledge of the technical stack, their understanding of software engineering can always benefit from the stated guidance. The applicability to organizations belonging to category C-III is ruled out for reasons already stated before.

It was decided to opt for the iterative development in the agile way as the agreed approach of work for *Nirmanik*, building the software in increments leveraging Scrum. However, Scrum primarily addresses the activity spaces in the endeavor area of concern. We could discern gaps in the requirements area for instance, which is part of the solution area of concern, as we thought about the next steps. Acknowledging that the team would benefit from some explicit guidance regarding activities in the solution area of concern to attain greater effectiveness, we would adopt User Story practice as detailed in the following section.

5.3 Adoption of User Story practice

In the previous section where we mapped the states of the Essence Kernel Alphas for the project, the Requirements alpha [2] was deemed to be at the Conceived state, as shown in Fig. 5-4(a). To address the requirements and thus advance the Requirements

alpha along the subsequent states, we decided to use the User Story Lite practice, a simplified version of the User Story practice, in our endeavor. User stories enable a team to contemplate, question, and understand the value of their endeavors from a user-centric point of view.

5.3.1 User Story Lite practice - an overview

A user story is a depiction of the system functionalities of interest, articulated by an approach utilizing informal discussion between the system user and the developer [58]. User stories might be considered as sub alphas, like treating Requirement Items as sub-alphas of Requirements [2]. The User Story Lite practice enables us to break down Requirements into User Story sub-alphas. The corresponding alpha card is shown in Fig. 5-5(a).

The work products in the User Story Lite practice are the Story Card, and the Test Case for each user story. A story card captures the narrative of a user story (shown in Fig. 5-6(a)), and a test case helps verify the same [2].

Fig. 5-2 represents the Essentialized model showing the User Story Lite practice, showing relationships between the elements (alpha, work products and activities) in the practice, the activity flow and relationships with kernel elements. These will be elaborated upon in the subsequent description of how we had gone about adopting the User Story Lite practice for *Nirmanik*.

5.3.2 How we adopted User Story Lite practice for *Nirmanik*

Working with User Story Lite would entail several activities, such as to find user stories, prepare each user story for development, and then accept the implementation of each user story.

The following user story was created:

Nirmanik::Basic User Story

The top management of the Nirmanik real estate organization wants to have an automated pre-sales management system in place so that leads obtained from

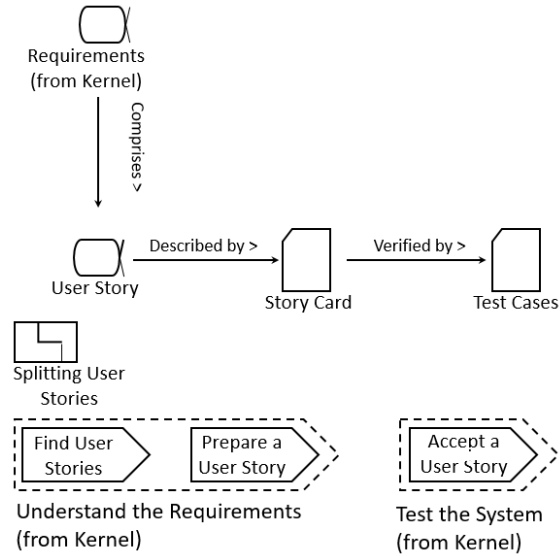


Figure 5-2: The Essentialized model showing the User Story Lite practice [2].

various sources can be acted upon and nurtured to closure quickly using well defined workflows with provision of appropriate and timely alerts before handing over to Sales/Post-Sales functions

Once we decided to adopt User Story Lite for *Nirmanik*, we had to undertake the following activities:

- “Find User Stories”,
- “Prepare each User Story for development”,
- “Apply the Splitting User Stories Pattern”, and
- “Accept the implementation of the User Story” [2].

Here is an elaboration on how these activities were undertaken.

Nirmanik::Find User Stories

The Find User Story activity card [2] is shown in Fig. 5-6(b).

The user stories to be taken up for development were identified, which would be as follows:

- i. *Nirmanik::Track pre-sales leads*

- ii. *Nirmanik::Assign prospective leads to sales executives*
- iii. *Nirmanik::Facilitate sales executive's activities to nurture the opportunity towards closure*
- iv. *Nirmanik::Build reports / dashboards to facilitate timely action by management*

Nirmanik::Prepare a User Story

The Prepare User Story activity card [2] is shown in Fig. 5-5(b).

While proceeding to prepare the user stories for development, our primary focus was on the first story in section 5.3.2, *Nirmanik::Track pre-sales leads*, as it was a key part of the CRM pre-sales activities for *Nirmanik* and provided the basis for the remaining three user stories as mentioned above. The user story developed for *Nirmanik::Track pre-sales leads* is as follows:

User Story *Nirmanik::Track pre-sales Leads*

The *Nirmanik* stakeholders would like to identify the pre-sales Leads coming from various sources and track them as they progress.

There should be a facility to capture Leads coming from various sources

Sources can be online portals, social/digital media, company website, calls etc.

Leads should be de-duplicated.

While the same Lead may come multiple times from different sources, only one instance of the same should be tracked over the effective lifecycle of the Lead.

There should be a provision to indicate the progress or maturity of Leads.

Applicable values for Status field for progress: Guest / Prospective / Inactive / Cancelled.

For the lead indicating site visit date (timeslot) preference, the lead status would be updated to *Prospective Lead*.

Acceptance criteria:

1. There should be provisions to ingest leads from multiple sources.

2. There shouldn't be duplicate instances of leads over a lead's effective lifecycle.
3. The Status field for the progress of leads should change appropriately as leads are tracked and followed up for visit confirmation.

Nirmanik::Applying the Splitting User Stories Pattern

This involved splitting the larger user stories into smaller stories in alignment with the INVEST criteria, especially the small and testable criteria [2]. The corresponding pattern card, Splitting User Story, is shown in Fig. 5-7(a).

Fig. 5-3 shows how the first user story, *Nirmanik::Track pre-sales leads*, was split into three smaller ones.

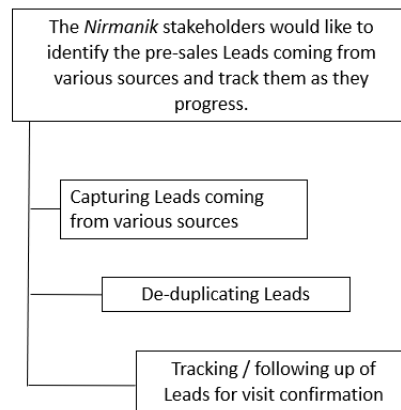


Figure 5-3: Splitting the first user story: *Nirmanik::Track pre-sales leads*.

Nirmanik::Accept a Story

This is about clearly depicting the acceptance criteria for the user stories. The frequent communication between the developers and the product owner over delivering the user story ensured better alignment and smoother acceptance. The Accept User Story activity [2] is shown in Fig. 5-7(b).

With the adoption of User Story Lite practice, the requirements for the proposed software solution to be built got clarified, a shared solution came to exist within the team, requirements format was agreed and the internal algorithms were

well-articulated. Accordingly, the Requirements alpha [2] would advance from the Conceived state, as shown in Fig. 5-4(a), to the Bounded and Coherent states, as shown in Fig. 5-4(b) and Fig. 5-4(c) respectively.

Also, there was clarity now regarding the stakeholders funding this work. It was decided to utilize a backlog stemming from the requirement set drawn up for planning purposes, and to keep a regular communication channel open with the stakeholders to ensure the relevance of the backlog, thus progressing the Work alpha [2] to the Prepared state.

5.3.3 Alpha state cards for Requirements

We have shown the Essence alpha state cards for Requirements: Conceived, Requirements: Bounded and Requirements: Coherent in Fig. 5-4(a), Fig. 5-4(b) and Fig. 5-4(c) respectively.

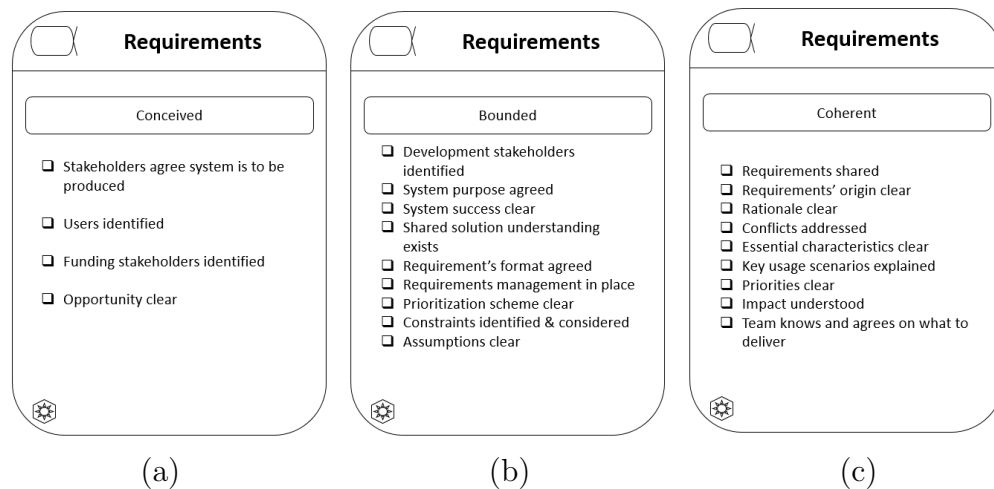


Figure 5-4: Requirements alpha state card [2]: (a) Conceived. (b) Bounded. (c) Coherent.

5.3.4 Applicable Essence cards for User Story Lite

We have shown the alpha card “User Story” in Fig. 5-5(a), the work product “Story Card” in Fig. 5-6(a), the activity cards “Prepare User Story”, “Find User Story” and

“Accept User Story” in Fig. 5-5(b), Fig. 5-6(b) and Fig. 5-7(b) respectively, and the pattern card “Splitting User Story” in Fig. 5-7(a).

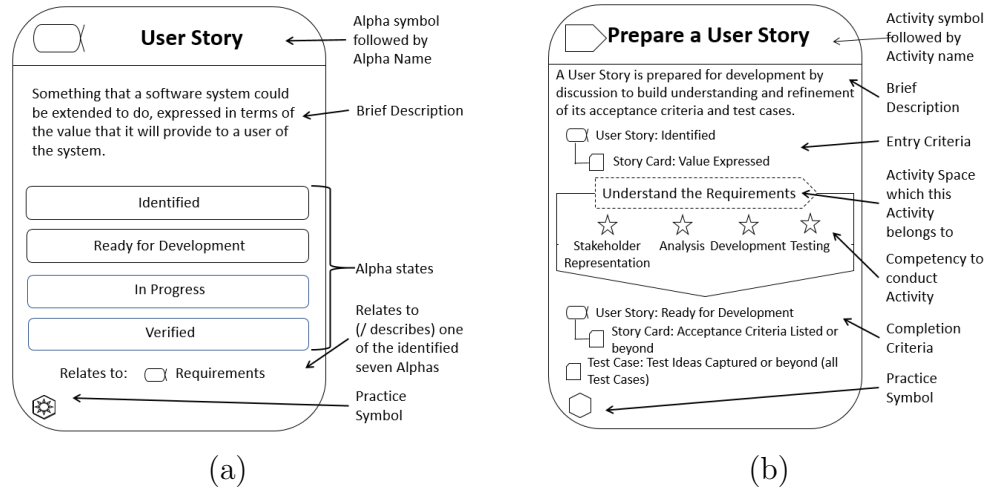


Figure 5-5: (a) User Story alpha card; (b) Prepare User Story activity card. Both showing linkage with Essence elements.

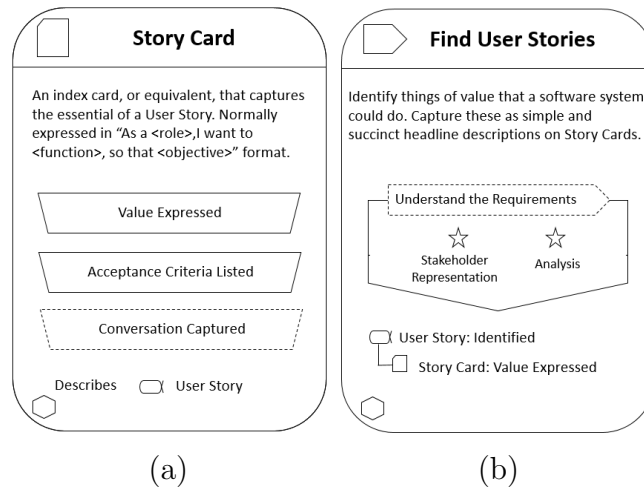


Figure 5-6: (a) Story Card work product. (b) Find User Story activity card.

5.3.5 The Value of the Kernel to the User Story Lite practice

A key accomplishment of essentializing the User Story Lite practice was to provide the *Nirmanik* team with clarity regarding which alphas were being progressed. We helped the *Nirmanik* team understand that the User Story Lite practice was instrumental in helping them to achieve the following Essence kernel alpha states.

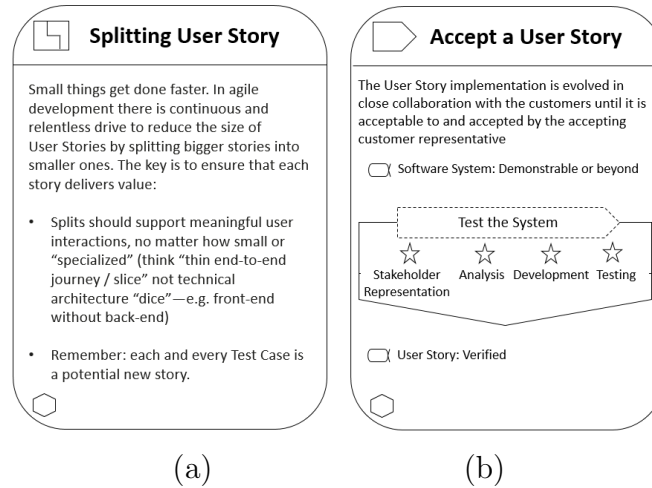


Figure 5-7: (a) The Splitting User Story pattern card [2]. (b) Accept User Story activity card.

- Requirements alpha: Bounded and Coherent state
- Work alpha: Prepared state

The three activities in User Story Lite [2] cover two kernel solution activity spaces, “Understand the Requirements” and “Test the System”, shown as shaded in Fig. 5-8.

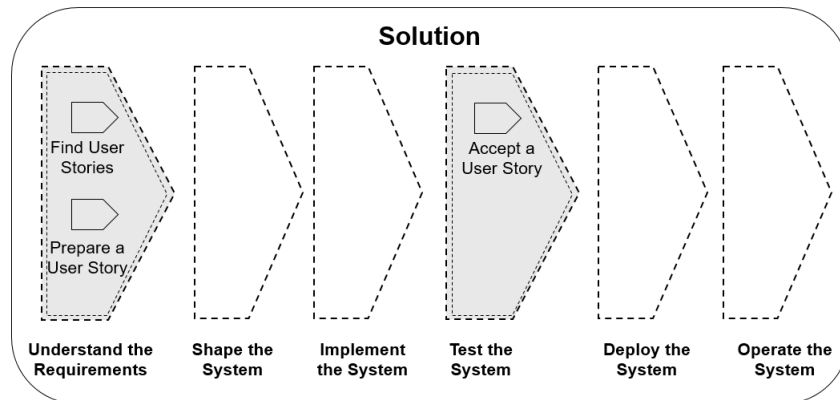


Figure 5-8: User Story Lite coverage of kernel solution activity spaces.

However, even with the adoption of the practices thus far, two kernel solution activity spaces: “Shape the System” and “Implement the System”, remained yet to be addressed, as signified by those two spaces positioned between the two aforementioned shaded ones in Fig. 5-8. The next section describes how we would go about adopting

the Microservice practice to address design and implementation, which would cover the two aforementioned kernel solution activity spaces, for *Nirmanik*.

5.4 Adoption of Microservice practice

The microservices architecture is about developing an application in terms of a collection of services demonstrating properties like cohesiveness and loose coupling, with each such service having complete ownership of its individual data, logic, and behaviour[53]. Related functionalities are combined into a specific business capability (termed as bounded context), with each microservice ideally implementing one such capability [16]. It is important to identify the right kind of partition of the system into microservices given the impact of the architecture on the system performance [20, 21].

In the Microservice practice, the microservices themselves can be viewed as sub-alphas of the Software System kernel alpha [2]. We decided to use Microservices Lite, a lighter version of the Microservice practice, for this endeavor. Individual components within the *Nirmanik::Lead Tracking subsystem* were built as separate microservices rather than as a monolith within *Nirmanik*. That approach ensured that there was no tight coupling to external systems, thus enabling the exploration and evolution of the new functionality in a de-risked manner.

Let us now understand the Microservices Lite (and Microservice) practice in greater detail and describe how we went about adopting this practice for *Nirmanik*.

5.4.1 Microservices Lite practice - an overview

Microservices Lite begins by identifying the requirements by applying a suitable practice, say user stories. Then the microservices to implement the requirements would need to be identified.

The primary alpha in the Microservices Lite practice is the Microservice alpha. All work products and activities in this practice are related to this alpha. The Microservice alpha can be treated as a sub-alpha of the Software System alpha [2]. The corresponding alpha card is shown in Fig. 5-12. As a software system consists

of microservice sub-alphas, each microservice has a bearing on the progress of the system [2].

The work products of Microservices Lite are as follows:

Design Model: Inspection of the software system is required to break it up into microservices leveraging criteria like low coupling and high cohesion. This work product focuses on the interfaces between and the collaboration among microservices.

Microservice Design: Work product describing a microservice design from interfaces to behaviour.

Microservice Build and Deployment Script: A work product, an automated script enabling each microservice to be deployed quickly, independent of others.

Microservice Test Case: A work product for measuring the behavior of a microservice.

Fig. 5-9 represents the Essentialized model showing the Microservices Lite practice [2], the relationships between the elements (alpha, work products and activities) in the practice, the activity flow and the relationships with kernel elements.

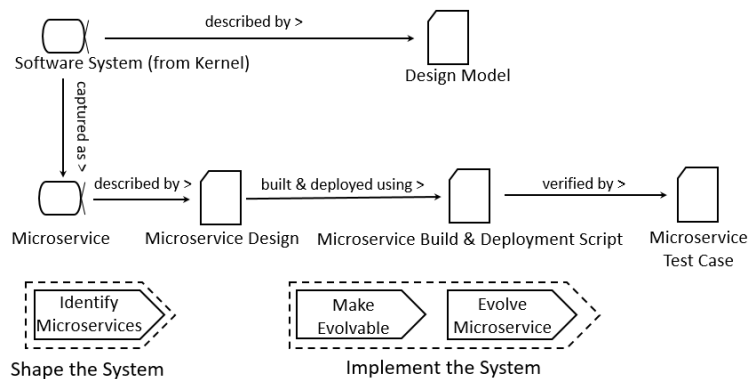


Figure 5-9: The Essentialized model showing the Microservices Lite practice [2].

Some select elements out of these will be covered in detail in the following section which describes our adoption of the Microservices Lite practice for *Nirmanik*.

5.4.2 How we adopted Microservices Lite practice for *Nirmanik*

Among the activities involved, a key one is about identifying Microservices, the Alphas in the Microservices Lite practice.

For *Nirmanik*, the ingestion/consumption of leads from external sources (company website, external property listing sites, social media, etc.) is independent of the identification of duplicates amongst those leads, while lead tracking and follow-up is an independent workflow altogether. These three demonstrate the essential attributes of low coupling and high cohesion, thus making a good case for potential microservices. Accordingly for *Nirmanik::Lead Tracking subsystem*, the microservices include one for Ingesting Leads from external sources, one for de-duplication of Leads, and one for Lead tracking and follow-up for visit confirmation.

As the initial efforts were focused on getting the design in place for the microservice oriented architecture, the Design Model, and the Microservice Design work products would be the most relevant artifacts resulting from the said exercise. We would accordingly consider the Design Model and the Microservice Design work products and elaborate on those two in the context of *Nirmanik*, specifically taking the *Nirmanik::Lead Tracking Subsystem* as an example.

Design Model

This describes the Software System alpha by listing the elements in the Software System and showing their interactions [2]. Fig. 5-13(a) shows the Design Model work product card [2].

Fig. 5-10 shows the design model created for the *Nirmanik::Lead Tracking Subsystem*. The work product in question has progressed to the level of Collaborations and Interfaces Defined, as it describes how the *Nirmanik::Lead Tracking subsystem* microservices collaborate among themselves and interact with the external lead generation sources (like real estate portals).

The left-hand side of Fig. 5-10 shows the external sources of Leads. Most of them

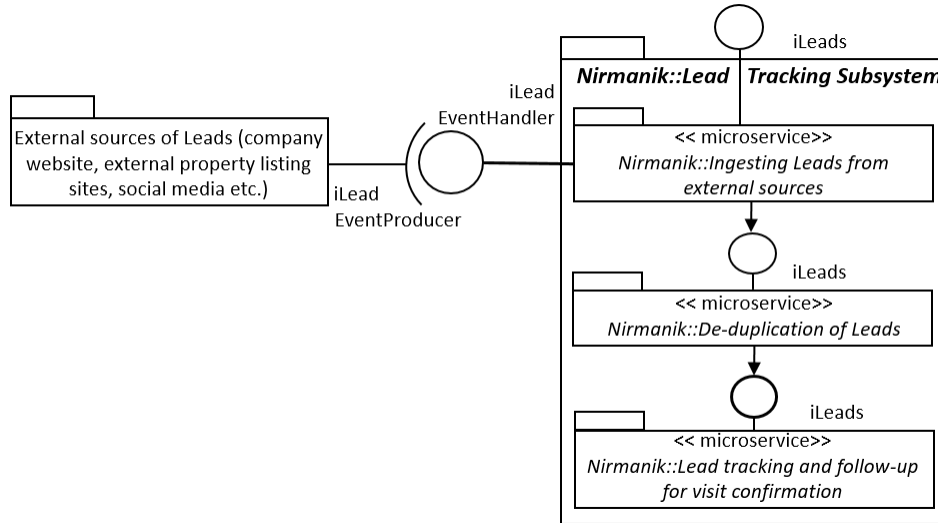


Figure 5-10: Design Model for *Nirmanik::Lead Tracking Subsystem*.

would provide API interfaces. For those which don't, suitable techniques need to be employed to build interfaces to provide the details of the leads as appropriate. These, collectively called *iLeadEventProducer*, are instrumental in emitting Lead events. When an incidence of note takes place (say an interested customer searching for a property listing) it would be transmitted via an event to the *iLeadEventHandler* interface. The *Nirmanik::Lead Tracking Subsystem* would process that information and leverage it to trigger the subsequent workflows.

The right-hand side of Fig. 5-10 shows that *Nirmanik::Lead Tracking Subsystem* is designed using three microservices:

- *Nirmanik::Ingesting Leads from External sources*,
- *Nirmanik::De-duplication of Leads*, and,
- *Nirmanik::Lead tracking and follow-up for visit confirmation*.

Each lead would be collected through the *iLeadEventHandler* interface. This lead ingestion process lends itself well to be treated as a microservice, shown as the *Nirmanik::Ingesting Leads from External sources* microservice in Fig. 5-10. This ingestion is accomplished via API, which can be push-based or pull-based. Pull API implies

some sort of batching mechanism. On the other hand, for leads that need to be treated on a near real-time basis, push API would be preferred.

De-duplication of the leads ingested by the system is an important activity. An individual interested in a given property might approach *Nirmanik* through multiple channels (multiple third-party property listing sites, social media or even *Nirmanik's* own portal). This might result in duplicate entries for essentially one single lead, resulting in potential redundancies and credit allocation problems later on. De-duplication of leads is critical for pre-sales to increase the effectiveness and efficiency of subsequent operations. Without it valuable resources might get wasted in pursuing the same leads, the same prospect might receive multiple calls resulting in irritation and possibly loss of sales, and reporting would become inaccurate.

A combination of key attributes of the prospect (e.g. mobile number and email id) can be used to identify duplicate leads. Ways to handle this might be to reject all duplicate leads outright or to treat the earliest lead to arrive as the parent lead and consider subsequent duplicate leads as its children. The second approach can help build intelligence on channel effectiveness in generating leads for *Nirmanik*.

The characteristics of this functionality also fulfill the key criteria for adopting a microservices approach. This is shown as the *Nirmanik::De-duplication of leads* microservice in Fig. 5-10.

Subsequently, the unique ingested leads are taken through a workflow as those leads are called up, entailing changes in the Status field for progress associated with leads based on the response (or lack of it) as detailed below.

The Status field for progress for the leads can assume, at this stage, one of the following values as described below.

- *Guest Lead*: Any fresh lead who wants to visit but is yet to confirm a visit date (time slot). It is expected that a telecaller would try calling the lead up a certain number of times to fix the visit date (time slot) in order to convert the lead to a *Prospective Lead*.
- *Prospective Lead*: Any fresh lead (or a *Guest Lead*) who shows a clear intent to

visit by providing a visit date (time slot).

- *Inactive Lead*: When the Lead does not respond to the telecaller. It is expected that telecallers would try calling up the *Inactive* leads for a certain number of times to fix up the visit date (timeslot) in order to convert them to *Prospective* leads.
- *Cancelled Lead*: An *Inactive Lead* that remains unresponsive over three attempts by the telecaller would have the status changed to *Cancelled*. A *Cancelled Lead* would no longer be considered by the system. In case he approaches at a later point in time he would be considered as a fresh lead and no past history would be taken into account. A lead that responds to the telecaller but indicates he is not interested in visiting in the foreseeable future is also categorized as *Cancelled*.

For *Prospective* leads, the site visit date (timeslot) needs to be captured. The next user story will be dealing with them.

This entire flow of activities has a good degree of cohesion and hence becomes a good candidate for a single microservice, shown as the *Nirmanik::Lead tracking and follow-up for visit confirmation* microservice in Fig. 5-10.

Each of the three microservices shows a high degree of cohesion internally but low coupling with each other, satisfying the essential characteristics of microservices. Each of these provides an iLeads interface to leads with a progressive amount of refinement (or processing) to subsequent stages.

Microservice Design

This is a work product describing a microservice's design, including the interfaces, behaviour and internal design [2]. Fig. 5-13(b) shows the Microservice Design work product card [2].

A key aspect of a microservice is its ability to communicate with other microservices, which is realized by its interface or API. It needs to be ensured that the APIs

(i.e. the component interfaces) are loosely coupled, which would be instrumental for independent deployment of microservices, again a key requirement in this paradigm.

While we have already covered a set of interfaces in the design model described in the previous sub-section, there might also be interfaces that a microservice would require for taking care of execution as explained below.

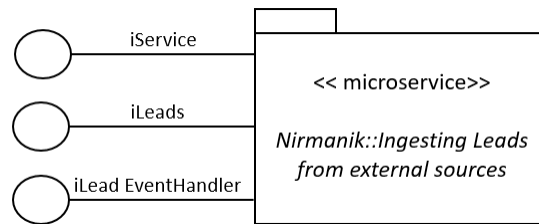


Figure 5-11: Microservice Design of *Nirmanik:: Ingesting Leads from external sources*.

Fig. 5-11 shows all the interfaces to the *Nirmanik::Ingesting Leads from External sources* microservice, which are:

- *iService* - takes care of the execution,
- *iLeads* - handles the processed Leads, and,
- *iLeadEventHandler* - manages external Lead generating events.

Fig. 5-11 is a work product for the Microservices Lite practice, which in the given context is drawn taking the *Nirmanik::Ingesting Leads from External sources* microservice as an example from the *Nirmanik::Lead Tracking Subsystem*, which has progressed to the level of Interfaces Specified.

As mentioned earlier, each of the remaining two microservices, *Nirmanik::De-duplication of Leads* and *Nirmanik::Lead tracking and follow-up for visit confirmation*, provides an *iLeads* interface to leads with a progressive amount of refinement (or processing) to subsequent stages. Also, both these microservices will have an *iService* interface to take care of the execution. We haven't created separate diagrams to show these two interfaces to those two microservices.

A key tenet to implementing distributed systems is to use business capabilities as the design element, moving away from the data-centric design [46]. We would subse-

quently consider the adoption of a publish/subscribe workflow, keeping in mind that adopting a message-oriented implementation is an effective way to keep the shared interfaces between components up-to-date (with appropriate refactoring) and asynchronous message-passing would aid in loose coupling, a desired trait of a microservice architecture.

5.4.3 Applicable Essence cards for Microservices Lite

We have shown the “Microservice” alpha card in Fig. 5-12, the Design Model work product card in Fig. 5-13(a) and the Microservice Design work product card in Fig. 5-13(b) [2].

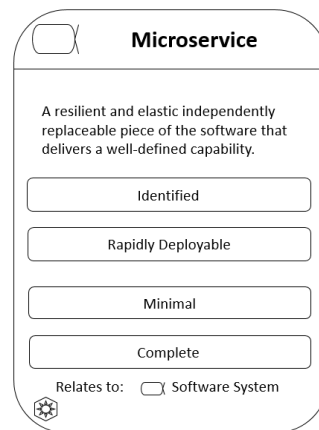


Figure 5-12: Microservice alpha card.

5.4.4 The Value of the Kernel to the Microservices Lite practice

Earlier in section 5.3.5 it was pointed out that two of the kernel solution activity spaces, “Shape the System” and “Implement the System”, were still not addressed after adopting the practices till that point.

We subsequently proceeded to adopt the Microservices Lite practice which addresses implementation guidance for *Nirmanik*, and is instrumental in covering two

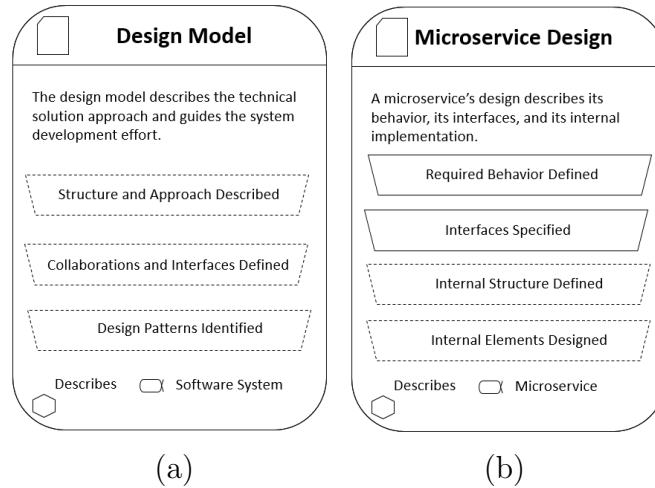


Figure 5-13: Work product card (a) Design Model. (b) Microservice Design.

kernel solution activity spaces, “Shape the System” and “Implement the System”, shown as shaded in Fig. 5-14.

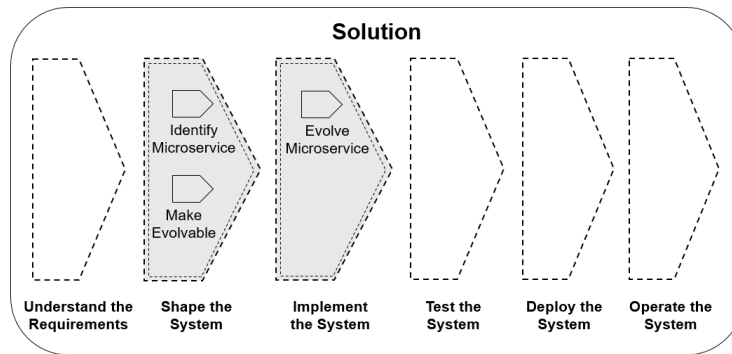


Figure 5-14: Microservices Lite coverage of kernel solution activity spaces.

The adoption of the Microservices Lite practice in addition to the User Story Lite practice already adopted earlier would result in covering all four consecutive kernel solution activity spaces: “Understand the Requirements”, “Shape the System”, “Implement the System” and “Test the System”, as explained in greater details in the next section.

5.5 Summary of chapter

We have crafted a view of the software project execution process in this chapter by showcasing how we adopted the practices User Story Lite and Microservices Lite on top of Essence kernel for the CRM automation journey of a real-estate organization, building upon our previous work[3] of adopting Essence as the foundation leveraging Scrum. Here we have identified four user stories for the applicable scope, out of which the first one, *tracking pre-sales lead*, has been prioritized and addressed. Then we have taken up the exercise of realizing the said user story via Microservice practice. Out of the work products associated with the Microservice practice, we have prioritized the Design Model and the Microservice Design work products and developed those two in the context of the user story being considered, *tracking pre-sales lead*, as part of our work here.

In this context, it might be worthwhile to touch upon some relevant considerations, such as how we can ensure the verification of functional correctness at the user story level and how the microservices identified in our solution are going to be implemented.

The microservice test cases work product deals with the verification of functional correctness at the user story level, while the Microservice Build and Deployment Script work product is about realizing the design by using the implementation specifics. We should consider these two work products and develop them if we decide to broaden our scope of work and address the implementation and functional correctness verification aspects in the future.

Another relevant question that may come up is how we can verify the performance aspects of the proposed solution.

This is a non-functional requirement (NFR). While this is certainly important, as far as the requirements aspect for the industry scenario in this thesis is concerned, our primary focus has been on functional requirements, and hence this is not part of the scope here. However, this can be a research possibility for us going forward.

Looking at Fig. 5-8 as well as Fig. 5-14, it can be observed that User Story Lite and Microservices Lite practices together cover the four consecutive kernel solution

activity spaces: “Understand the Requirements”, “Shape the System”, “Implement the System” and “Test the System”. This is depicted in Fig. 5-15 for ease of viewing.

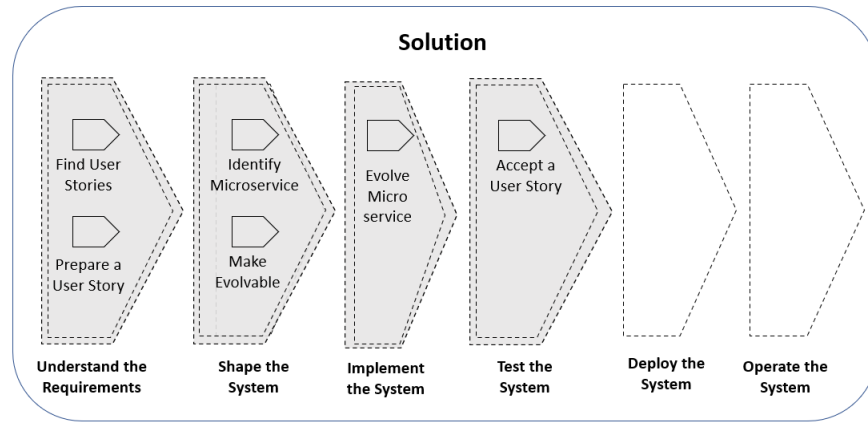


Figure 5-15: Coverage of the composition of User Story Lite and Microservices Lite of kernel solution activity spaces.

In other words, weaving the User Story Lite and the Microservices Lite practices in the method composition allows us to address the software development lifecycle spanning requirement analysis, design, development and testing.

Thus we have shown how established practices like Scrum, User Story and Microservices can be extended using Essence kernel as the unifying framework and how those essentialized practices can be composed into a method tailored for a given organization. A key user story has been prioritized and addressed by leveraging the User Story Lite practice. We have continued to focus on that user story by developing a set of work products in Microservices Lite practice to provide an informed view.

While the exercise as described here has been carried out for an organization belonging to category C-I as mentioned before, this would be applicable for organizations in category C-II as well.

The usefulness of the resulting method can be extended beyond the industry considered here as well as the domain, since that method would have wide applicability by acting as a guidance for similar project implementation endeavors in other industries and domains. While the User Story and the Microservice practices are almost ubiquitous in their own right, the exercise carried out here has enabled us to essentialize these practices and compose them into a method, as guided by the Essence

framework.

This has the potential to be extended to an even wider scope, whereby it can act as a blueprint for other industry/domain settings which would require a selection of practices different from the ones used here. The principles of our work here can be leveraged keeping Essence as the central framework, essentializing the new practices, and composing them into a method that would help meet the objectives for the industry/domain setting being considered.

Going forward, we would like to take up the second one out of the remaining user stories, as it would involve detailing an involved process logic for a workflow to carry out the assignment of leads intending to do site visits to the suitable Sales Executives, subject to different constraints and rescheduling/change in preferences. This would lend itself well to the usage of Petri Nets for modelling it, as we will show in the upcoming chapter.

Construct a generalized Petri net model of key CRM pre-sales real estate functionalities

Contents

6.1 Context	103
6.2 Functionalities for assigning leads to appropriate Sales Executives: life-cycle of leads	105
6.3 Functionalities for assigning leads to appropriate Sales Executives: assignment workflow process description . .	108
6.4 Modelling considerations	119
6.5 Petri net modelling of the assignment process workflow	122
6.6 Verification of our Petri net models	136
6.7 Summary of chapter	150

Continuing with the automation of the CRM pre-sales functionalities, out of the four user stories identified in section 5.3.2, the second one would be the focus of this chapter which deals with the assignment process. We would detail out the underlying functionalities of the allocation / de-allocation of Sales Executives in response to a broad spectrum of lead requests by addressing a variety of combinations arising out of scheduling/rescheduling of site visit requests from leads, the availability/unavailability of Sales Executives on the said day of site visit and the applicable constraints associated with the subsequent matching process. We would next decide

on a suitable approach to model the flow of the assignment process. With the workflow being a recurring concept in many application domains, and processes tending to be a key part of that, it follows that the adoption of a framework suitable for modeling and analyzing workflow processes would be necessary [31]. Petri net would be our model of choice given its usefulness in modelling, formal analysis and design of systems like the one in the current scope, and its suitability to model workflows [32, 31, 33]. Also, an important objective would be to carry out the validation of our modelled system with regard to a specific set of properties. We'd leverage Petri net based behavioural analysis techniques to verify the suitability of our modelling. This would enable us to determine from a system designer capacity whether the desired functional properties of the system are present or not [34].

For the key concepts of Petri net, section 2.3 may be referred to. As far as the domain/industry is concerned, section 2.4 of this thesis provides an understanding of the background.

This chapter would be organized into the following sections:

- Context
- Functionalities for assigning leads to appropriate Sales Executives: life-cycle of leads
- Functionalities for assigning leads to appropriate Sales Executives: assignment workflow process description
- Modelling considerations
- Petri net modelling of the assignment process workflow
- Verification of our Petri net models
- Summary of chapter

Let's now set up the required context for discussing the functionalities relevant to the assignment process.

6.1 Context

Let us look at the first of the four user stories identified in section 5.3.2 as follows to help set up the context and outline the entry condition of the core functionalities in scope for this chapter, which is the second of the four user stories.

The stakeholders would like to identify the pre-sales leads coming from various sources and track them as they progress. There should be a facility to capture leads coming from various digital sources like online real estate portals, social/digital media (possibly against campaigns), company website as well as traditional avenues like direct inquiries via phone calls and so on. All such leads should be de-duplicated, meaning that while the same lead may come multiple times from different sources, only one instance of the same should be tracked over the effective lifecycle of the lead.

Once the leads are aggregated over all incoming channels and then de-duplicated, to carry out a validation of the intent of those leads before they are passed on to a more intricate algorithmic layer to schedule site visits using the applicable roster of Sales Executives, taking various parameters into account that would be discussed later in detail.

Leads would have a status field for progress. The status of a lead who is getting ingested into the system for the first time (Fresh Lead) would be one of the following:

- *Guest Lead*: Any Fresh Lead who wants to visit but is yet to confirm a visit date. It is expected that a telecaller would try calling up the lead a certain number of times to fix the visit date in order to convert the lead to a Prospective Lead. Until that happens however this Guest Lead would continue to remain within the ambit of this (first) module only.
- *Prospective Lead*: Any Fresh Lead who shows a clear intent to visit right away by providing a visit date.

The status of a Guest Lead can change to one of the following given its response (or the lack of it):

- *Prospective Lead*: When the Guest Lead shows a clear intent to visit by providing a visit date.
- *Inactive Lead*: When the Guest Lead does not respond to the tele caller. This Inactive Lead would continue to remain within the ambit of this (first) module only till there is a status change. The previous state (lead status) would be retained so that it can be reverted to later on when the lead responds.
- *Cancelled Lead*: Guest Leads that respond to the tele callers but indicate they are not interested in visiting in the foreseeable future, as well as those Inactive Leads that remain unresponsive over three consecutive attempts by the tele callers would have the status changed to Cancelled. The Cancelled Leads would no longer be considered by the system. In case they approach at a later point in time they would be considered as Fresh Leads and no past history would be taken into account.

It needs to be noted that a Prospective Lead would no longer remain in the scope of the first module and would thus mark the entry condition into the second module, *Assigning leads to appropriate Sales Executives*.

To cater to the site visit date requested by the Prospective Lead, a (provisional) assignment process would be invoked for matching the request to an available Sales Executive and finding a mutually acceptable time slot for the Requested Date. This scenario will be covered in the next section as part of the workflow for the assignment of leads to Sales Executives.

6.2 Functionalities for assigning leads to appropriate Sales Executives: life-cycle of leads

We would now describe the assignment process for matching the site visit request from a lead to an available Sales Executive to arrive at a mutually acceptable time slot, constituting the key functionalities in scope for this chapter.

The invocation of the assignment process happens as the tele callers sift through the de-duplicated leads to call them up, and each of the responding leads subsequently gets allocated for the site visit to one out of a team of Sales Executives (depending on availability) following the said assignment process.

We would first document the terminology used in the life-cycle of leads in this section, and in the subsequent section detail the workflow for assigning the various types of leads to the Sales Executives based on the situational factors at hand, depicting the various scenarios for the assignment process and addressing each of those.

As described in section 6.1, the status of a lead who is getting ingested into the system for the first time (Fresh Lead) would be one of the following:

- *Guest Lead*
- *Prospective Lead*

The status of a Guest Lead can change to one of the following depending on whether it provides a response or not respectively (details in section 6.1):

- *Prospective Lead*
- *Inactive Lead*
- *Cancelled Lead*

It should be noted that whenever any lead changes to Inactive Lead, the previous state (lead status) would be retained so that the lead status can be reverted back to its previous state if the lead responds later on within three attempts. However, if

there is no response for three consecutive attempts, the lead is changed to Cancelled Lead.

A Prospective Lead would mark the intersection of the lead lifecycle with the entry into the second module, *Assigning leads to appropriate Sales Executives*, and the status of a Prospective Lead can change to one of the following depending on its response (or lack of it):

- *Scheduled Lead*: As the lead's site visit requirement by requesting an appointment for a later date (the Requested Date) has been validated, a Sales Executive has been assigned based on availability for a mutually agreed time slot on the Requested Date (of visit).
- *Inactive Lead*: When the lead does not respond to the telecaller or wants to reschedule but is unable to provide a visit date.
- *Cancelled Lead*

The status of a Scheduled Lead can change to one of the following depending on its response (or lack of it):

- *Visited Lead*: When a Scheduled Lead has made a site visit with an assigned Sales Executive. The said Sales Executive would now be deemed as the Original Sales Executive for the said Visited Lead and will continue to remain tagged to the said lead (unless the lead requests for a change of the Sales Executive or the Sales Executive is no longer available permanently/ for the foreseeable future).
- *Inactive Lead*: When the lead fails to turn up for the scheduled site visit (no-show) or wants to reschedule but can't provide a visit date.
- *Cancelled Lead*

The status of a Visited Lead can change to the following:

- *Active Lead*: When a Visited Lead who has made a site visit earlier with a particular Sales Executive, with the latter being deemed as the Original Sales

Executive for the said Visited Lead, asks for a revisit and a time slot is assigned based on the lead's required revisit date (which we'll refer to as the Requested Date). The Original Sales Executive continues to remain tagged with the lead.

- *Inactive Lead*: When the lead fails to respond.
- *Cancelled Lead*

The status of an Active Lead can change to one of the following:

- *Visited Lead*: When the Active Lead has made the site visit. The Original Sales Executive continues to remain tagged with the lead.
- *Inactive Lead*: If the Active Lead misses the scheduled site visit (no-show) or wants to reschedule but can't provide a visit date.
- *Cancelled Lead*

Thus the status of a lead can toggle between a Visited Lead and an Active Lead depending on whether it has expressed an intent to revisit the site or it has actually carried out a revisit of the site, whichever action is applicable. The Original Sales Executive continues to remain tagged with the lead unless the lead wants to change that or the said Sales Executive is no longer available.

For a Visited Lead or an Active Lead, sale closure is the desired end state (business outcome). However, for these two types of leads, it is also possible for the opportunity not to progress any further.

The status of an Inactive Lead can change to one of the following, depending on whether it does or doesn't respond to telecallers:

- *Guest Lead / Prospective Lead / Scheduled Lead / Visited Lead / Active Lead*: An Inactive Lead would revert to its retained previous state (lead status) of Guest Lead / Prospective Lead / Scheduled Lead / Visited Lead / Active Lead, if it responds within three attempts.
- *Cancelled Lead*: if it doesn't respond within three attempts.

The sale closure is a desired progression beyond this; however, that part is considered out of scope since here we are concerned with the pre-sales aspect only.

The Cancelled Leads are no longer taken into consideration by the system.

6.3 Functionalities for assigning leads to appropriate Sales Executives: assignment workflow process description

In this section, we will endeavor to describe the assignment of the various types of leads (as they progress through their individual lifecycles) to the Sales Executives (with or without constraints) taking into account the situational factors as applicable (like rescheduling requests from the leads, unavailability of Sales Executives and so on).

It is worth noticing that the assignment process is essentially a workflow. We tend to encounter workflows often in many application domains and processes form a key part of the same, hence it logically follows that we should aim to adopt a framework suitable for modeling and analyzing workflow processes [31].

6.3.1 Provisional Assignment process for Prospective Leads

Whenever a Prospective Lead's intent to schedule an appointment for a later date (the Requested Date) is registered, a corresponding provisional assignment would be attempted with an available Sales Executive for the Requested Date.

The Prospective Lead's request would be considered against the availability as per an ordered list of Sales Executives following the rules as described in section 6.3.8, with the lead being the Prospective Lead and the date being the Requested Date.

The telecaller would verify with the Prospective Lead which of the time slots available at that time for the Sales Executives would suit the Prospective Lead, by traversing the list in an ordered fashion.

There would be the following possibilities:

I - An available time slot for a Sales Executive on the Requested Date is acceptable to the Prospective Lead.

The Prospective Lead has progressed to a Scheduled Lead and a provisional assignment for the same is recorded in the system, but the details of the Sales Executive is not intimated to the Scheduled Lead then and there; however the Scheduled Lead is made aware that a visit has been scheduled on the Requested Date.

II - None of the available time slots for any of the Sales Executives end up being acceptable to the Prospective Lead or there is no available time slot for any of the Sales Executives on the Requested Date.

A callback would be arranged for the Prospective Lead at a later date as no assignment could be made suiting the Prospective Lead's preference for the Requested Date.

6.3.2 Rescheduling for Scheduled Leads

When a Scheduled Lead asks for a reschedule of the date, it would be noted down as the Change Date. A new matching would now be attempted for for the Scheduled Lead against the availability as per an ordered list of Sales Executives, following the rules as described in section 6.3.8, with the lead being the Scheduled Lead and the date being the Change Date.

The telecaller would verify with the Scheduled Lead which of the slots available at that time for the Sales Executives would suit the Scheduled Lead, by traversing the list in an ordered fashion.

There would be the following possibilities:

I - An available time slot for a Sales Executive on the Change Date is acceptable to the Scheduled Lead.

A provisional assignment for the same is recorded in the system, but the details of the Sales Executive is not intimated to the Scheduled Lead then and there;

however, the Scheduled Lead is made aware that a visit has been scheduled on the Change Date.

- II - None of the available time slots for any of the Sales Executives end up being acceptable to the Scheduled Lead or there is no available time slot for any of the Sales Executives on the Change Date.

The telecaller would in this case (II) explore the availability of Sales Executives on an Alternate Date (different from the Change Date). This can give rise to the following possibilities:

- III - An available time slot for a Sales Executive on an Alternate Date is found that is acceptable to the lead and a new provisional assignment is carried out in the system.

- IV - None of the available time slots on the Alternate Dates for the Sales Executives are acceptable to the lead or the Sales Executives have no available time slots on any Alternate Date.

A callback would be arranged for the Scheduled Lead at a later date as no assignment could be made suiting the Scheduled Lead's preference. The Scheduled Lead would now become a Prospective Lead.

For the options above, the Scheduled Lead's request would be considered against the availability as per an ordered list of Sales Executives, following the rules as described in section 6.3.8, with the lead being the Scheduled Lead and the date being the Change Date or the Alternate Date as the case may be.

The previously assigned Sales Executive would be de-allocated for the originally assigned date corresponding to the prior (provisional) match existing in the system for the said Scheduled Lead, and would now become available for the corresponding time slot.

Post assignments made as described in sections 6.3.1 and 6.3.2, when the Scheduled Lead has made the site visit, its status is progressed to Visited Lead and the

Sales Executive who facilitated the site visit would be tagged as the Original Sales Executive for the said lead.

6.3.3 Assignment process for Visited Leads

Whenever the intent of a Visited Lead to re-visit the site on a future date (the Requested Date) is registered, the corresponding search for a Sales Executive would be attempted with a specific constraint, whereby the assignment would first try to find the availability of the Sales Executive who was assigned as the Original Sales Executive to the Visited Lead in order to come up with a mutually acceptable time-slot.

There would be the following possibilities:

- I - Availability of the Original Sales Executive for the Visited Lead can be found for a mutually acceptable time slot on the Requested Date.
- II - The Original Sales Executive is either unavailable on the Requested Date or his available slots on the Requested Date are not acceptable to the lead, and an Alternate Sales Executive is available for a time slot on the Requested Date which is acceptable to the lead. A provisional assignment is made for the Alternate Sales Executive accordingly.
- III - The Original Sales Executive is unavailable on the Requested Date or his available slots on the Requested Date are not acceptable to the lead, and either no Alternate Sales Executive is available on the Requested Date, or even if an Alternate Sales Executive is available on the Requested Date the corresponding time slot is not acceptable to the lead. Thus there is no possibility either way of allocating a Sales Executive on the Requested Date.

The telecaller would in this case (III) explore the availability of Sales Executives on an Alternate Date (different from the Requested Date). This can give rise to the following possibilities:

IV - Availability for the Original Sales Executive on an Alternate Date is found for a time slot that is acceptable to the lead and the new assignment is carried out in the system.

V - There is no availability of the Original Sales Executive on the Alternate Date or none of the time slots available on the Alternate Dates for the Original Sales Executive are acceptable to the lead.

The tele caller would in this case (V) explore the availability of Alternate Sales Executives on an Alternate Date which can give rise to the following two possibilities:

VI - One Alternate Date for an Alternate Sales Executive is found to have a time slot that is acceptable to the lead and a new (provisional) assignment is carried out in the system.

VII - None of the available time-slots on the Alternate Dates for the Alternate Sales Executives are acceptable to the lead or Alternate Sales Executives have no available time-slots on any Alternate Date.

In case none of the previous options work out, a callback will be arranged for the Visited Lead at a later date.

For the options above involving the Alternate Sales Executive, the Visited Lead's request would be considered against the availability as per an ordered list of Sales Executives excluding the Original Sales Executive, following the rules as described in section [6.3.8](#), with the lead being the Visited Lead and the date being the Requested Date or the Alternate Date as the case may be.

In case an assignment has successfully been worked out for the Visited Lead following the above steps, the lead status changes to Active Lead.

6.3.4 Requesting Original Sales Executive change while requesting re-visit for Visited Leads

Here the Visited Lead has requested change of the Original Sales Executive and also asked for a re-visit, the date of the re-visit being termed the Requested Date. This can result in the following possibilities:

- I - An available time-slot for an Alternate Sales Executive on the Requested Date is acceptable to the lead; the Alternate Sales Executive would be provisionally allocated.
- II - Either no Alternate Sales Executive is available on the Requested Date or the available time slots of Alternate Sales Executives on the Requested Date are not acceptable to the lead, thus resulting in no allocation of Alternate Sales Executive for the Requested Date either way.

The tele caller would in this case (case *II*) explore the availability of Alternate Sales Executives on an Alternate Date (for appropriate time slots). This now gives rise to 2 more possibilities as below:

- III - Availability of a time slot for an Alternate Sales Executive is found on an Alternate Date that is acceptable to the lead and a new provisional assignment is carried out in the system.
- IV - None of the available time slots for the Alternate Sales Executives on Alternate Date(s) are acceptable to the lead or Alternate Sales Executives don't have any availability on any Alternate Date.

For the options above, the Visited Lead's request would be considered against the availability as per an ordered list of Sales Executives excluding the Original Sales Executive, following the rules as described in section 6.3.8, with the lead being the Visited Lead and the date being the Requested Date or the Alternate Date as the case may be.

For option IV (the request can't be accommodated), a call back will be arranged with the Visited Lead for a later date.

6.3.5 Rescheduling for Active Leads

When the Active Lead asks for a reschedule of the visit date, the tele caller would note it down as the Change Date.

There would be the following possibilities:

- I - The Original Sales Executive is available for a time slot on the Change Date which is acceptable to the lead. The Original Sales Executive would be allocated accordingly.
- II - The Original Sales Executive is unavailable on the Change Date or his available slots on the Requested Date are not acceptable to the lead, and an Alternate Sales Executive is available for a time slot on the Change Date which is acceptable to the lead. The Alternate Sales Executive would be provisionally allocated.
- III - The Original Sales Executive is unavailable on the Change Date or his available slots on the Requested Date are not acceptable to the lead, and there is no availability of any Alternate Sales Executive on the Change Date, or the available time slots of the Alternate Sales Executives on the Change Date are not acceptable to the lead. Thus there is no allocation of the Sales Executive for the Change Date either way.

The tele caller would in this case (case *III*) explore the availability of Sales Executives on an Alternate Date (different from the Change Date). This can give rise to the following possibilities:

- IV - A time slot on an Alternate Date for the Original Sales Executive is found that is acceptable to the lead and the new assignment is carried out in the system.
- V - None of the time slots available on the Alternate Dates for the Original Sales Executive are acceptable to the lead or there is no availability of the Original Sales Executive on an Alternate Date.

The tele caller would in this case (case V) explore the availability of Alternate Sales Executives on an Alternate Date which can give rise to the following two possibilities:

VI - One Alternate Date for an Alternate Sales Executive is found to have a time slot that is acceptable to the lead and a new provisional assignment is carried out in the system.

VII - None of the available time slots on the Alternate Dates for the Alternate Sales Executives are acceptable to the lead or Alternate Sales Executives have no availability on any Alternate Date.

The Original Sales Executive would be de-allocated for the originally assigned date and would now become available for the corresponding time slot.

For the options above involving the Alternate Sales Executive, the Active Lead's request would be considered against the availability as per an ordered list of Sales Executives excluding the Original Sales Executive, following the rules as described in section 6.3.8, with the lead being the Active Lead and the date being the Change Date or the Alternate Date as the case may be.

If rescheduling can't be successfully carried out, a callback will be arranged for a later date for the lead (who will now become a Visited Lead from an Active Lead) since rescheduling is not possible as of now.

6.3.6 Requesting Original Sales Executive change for Active Leads

When the Active Lead requests the change of Original Sales Executive while keeping the original date of assignment (Original Date) intact, it can result in the following possibilities:

I - An Alternate Sales Executive is available for a time slot on the Original Date which is acceptable to the lead; the Alternate Sales Executive would be provisionally allocated for the Original Date.

II - Either no Alternate Sales Executive is available on the Original Date or the available time slots of Alternate Sales Executives on the Original Date are not acceptable to the lead, thus resulting in no allocation of Alternate Sales Executive for the Original Date either way.

The tele caller would in this case (case *II*) explore the availability of Alternate Sales Executives on an Alternate Date. This now gives rise to the following two possibilities:

III - One time slot on an Alternate Date for an Alternate Sales Executive is found that is acceptable to the lead and a new provisional assignment is carried out in the system.

IV - None of the time slots on the Alternate Dates for the Alternate Sales Executives are acceptable to the lead or Alternate Sales Executives don't have any availability on any Alternate Date.

The Original Sales Executive would be de-allocated for the Original Date and would now become available for the corresponding time slot.

For the options above, the Active Lead's request would be considered against the availability as per an ordered list of Sales Executives excluding the Original Sales Executive, following the rules as described in section [6.3.8](#), with the lead being the Active Lead and the date being the Original Date or the Alternate Date as the case may be.

If the Original Sales Executive change request can't be successfully carried out, a callback will be arranged at a later date for the lead (who will now become a Visited Lead from an Active Lead) since the Original Sales Executive change request is not possible as of now.

6.3.7 Requesting Original Sales Executive as well as Date change for Active Leads

Here the Active Lead has requested change of the Original Sales Executive and also asked for a date reschedule (from the Original Date), the new date being termed the Change Date. This can result in the following possibilities:

- I - An available time slot for an Alternate Sales Executive on the Change Date is acceptable to the lead; the Alternate Sales Executive would be provisionally allocated.
- II - Either no Alternate Sales Executive is available on the Change Date or the available time slots of Alternate Sales Executives on the Change Date are not acceptable to the lead, thus resulting in no allocation of Alternate Sales Executive for the Change Date either way.

The tele caller would in this case (case *II*) explore the availability of Alternate Sales Executives on an Alternate Date (for appropriate time slots). This now gives rise to 2 more possibilities as below:

- III - Availability of a time slot for an Alternate Sales Executive is found on an Alternate Date that is acceptable to the lead and a new provisional assignment is carried out in the system.
- IV - None of the available time slots for the Alternate Sales Executives on Alternate Date(s) are acceptable to the lead or Alternate Sales Executives don't have any availability on any Alternate Date.

The Original Sales Executive would be de-allocated for the Original Date and would now become available for the corresponding time slot as the request is successfully accommodated.

For the options above, the Active Lead's request would be considered against the availability as per an ordered list of Sales Executives excluding the Original Sales Executive, following the rules as described in section [6.3.8](#), with the lead being the

Active Lead and the date being the Change Date or the Alternate Date as the case may be.

If the request can't be accommodated, a callback will be arranged for the lead (who will now become a Visited Lead from an Active Lead) for a later date.

6.3.8 The rules of ordering of Sales Executives to be considered for the concerned lead for the date in question

The concerned lead's request need to be considered against the availability as per an ordered list of Sales Executives for the date in question, and the basis of ordering for the Sales Executives would be:

1. The Sales Executives who do not happen to be the Original Sales Executive for any lead (Visited Lead / Active Lead) having a site visit scheduled on the date in question would be higher in the pecking order, with the one having the highest number of time-slots available on the date in question at the top of the list.
2. The Sales Executive who is the Original Sales Executive for the highest number of leads having a site visit scheduled on the date in question would appear at the bottom of the pecking order.
3. Amongst the Sales Executives who happen to be the Original Sales Executive for the same number of leads having site visits scheduled on the date in question, those with a higher number of time slots available on the date in question would appear higher in the pecking order.

A key aspect of the functionalities described for this module has to do with the life-cycle of leads, with the entry condition to this module being marked by a Prospective Lead which can subsequently progress to an Active Lead or a Visited Lead. As mentioned, the status of a lead can toggle between a Visited Lead and an Active Lead depending on its expressed intent to revisit the site or actually carrying out the revisit of the site, whichever action is applicable. For a Visited Lead or an Active Lead,

sale closure is the desired end state (business outcome) which also signals an exit condition for this module. However, for these two types of leads, it is also possible for the opportunity not to progress any further, which would result in those becoming Cancelled Leads. The Cancelled Leads, as discussed before, are no longer taken into consideration by the system. This state can be reached from any other state of a lead as applicable in the scope of this module and marks the other exit condition for this module.

Now that we have detailed the functionalities for the module *Assigning leads to appropriate Sales Executives*, our next endeavour would be to carry out the modelling exercise for the assignment process and subsequently verify the suitability of the same.

6.4 Modelling considerations

Let's start our modelling endeavour by creating a Statechart diagram to model the life cycle of the leads that we have discussed previously in section 6.2. The Statechart diagram models the behavior of an object during its lifetime by specifying the sequence of states attained by that object as it responds to events, along with its responses themselves respectively, over its lifecycle [59, 60, 61]. Hence a Statechart diagram would be a good candidate to capture the lead life cycle. The corresponding model is shown in Fig. 6-1.

In the current context, the key modelling consideration is centred around addressing the flow of the assignment process that forms the core of the CRM pre-sales functionalities in scope. The modelling should be able to take into account the dynamic interplay involving the assignment of Sales Executives, the request from the leads (depending on their states) to change the date of visit or the Original Sales Executive (for a Visited / Active Lead) as well as the assignment conflict/unavailability of Sales Executives and so on. The inherent abstraction of the assignment process is essentially a workflow, which happens to be a recurring concept in many application domains. Since processes usually form a key part of that, it follows that we should aim to adopt a framework suitable for modeling and analyzing workflow pro-

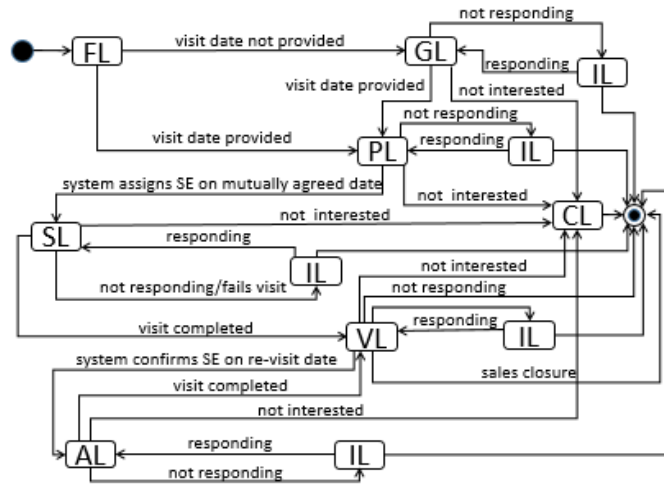


Figure 6-1: Statechart Diagram for the life cycle of the leads.

cesses [62, 63, 64].

While we'd decide on which framework to adopt in section 6.4.2 and subsequently carry out the modelling exercise utilizing that framework in section 6.5, let us first specify the verification criteria to determine the suitability of the model that we intend to construct by listing down the desired properties that the model should demonstrate.

6.4.1 Desired properties of the model being constructed

Our model should ideally be able to exhibit the following properties.

1. *Reach a specific state:* A key aspect of designing a system is ascertaining if it can reach a specific state, which essentially implies if it is capable of exhibiting a particular behavior. This would eventually lead us to affirm if the modelled system would be able to exhibit the desired properties per agreed specifications.
2. *Prevent overflow:* The modelled system should ensure that overflows don't exist. For example, if we consider an allocation node in a manpower allocation facility, given its capacity is finite, it is necessary to make sure that its representation by the corresponding property in the modelling adopted is such that this capacity

does not get exceeded by the operations [65].

Also, the nodes with overflow possibilities are potential bottlenecks, and to prevent any such possibility we need to ensure that there are no such instances in the models created.

3. *Prevent deadlock:* The modelled system should ensure that there is no deadlock in the system. If a sequence leading to a particular state can never be realized then it is redundant and should be eliminated. It follows that such an occurrence needs to be identified as it is likely to signify an error in the model or an inconsistency in the system being modelled.
4. *Fairness:* This property is related to being able to perform actions in the order of their announcements. In certain fairness considerations, the performability of a waiting action is not influenced by other actions, whereas in others a waiting action can either fall down the pecking order if other competing actions get the nod earlier or, at worst, fail to perform. The latter case is equivalent to the case of an announced action turning out to be non-performable when it should be executed with respect to its waiting period.

6.4.2 Suitability of Petri net as our modelling tool

Petri net has been a powerful construct with widespread usage across application domains, facilitating workflow modelling, with Petri net based analysis techniques being used for analysing those [66, 31, 67, 68]. The classical Petri net, invented by Carl Adam Petri in the sixties [32], has subsequently been utilized in a variety of application domains to model and analyze various processes. By analyzing a Petri net model, useful information about the underlying system can be obtained, for instance in terms of whether bottlenecks/deadlocks may exist. Crucially, it can be used to suggest changes without the need to run tests on the actual system [65]. The usage of Petri net to model workflows has been quite widespread, and a Petri net based framework would be suitable for workflow modeling in light of the following reasons:

1. *formal semantics*: the semantics of the Petri net are formally defined.
2. *expressiveness*: all the primitives needed to model a workflow process are supported.
3. *properties*: As allowed by the mathematical foundation.
4. *analysis*: Many analysis techniques are available thus facilitating the usage of Petri nets for workflow modeling and allowing us to prove properties like deadlock.
5. *vendor independent*: It's a tool-independent framework for modeling and analyzing processes [66, 67, 68, 69].

Taking all of the above into consideration, we have decided to use Petri net as the modelling approach for capturing the dynamics of the assignment process.

6.5 Petri net modelling of the assignment process workflow

In the preceding section (section 6.4) we have discussed the motivation behind the modelling approach that we'd like to adopt for modelling the functionalities in scope, especially the assignment process mentioned in section 6.3 In this section we'd be using Petri net, the framework of our choice, to model the flow of the assignment process that will allow us to formalize the main concepts, and in the following section carry out the verification of the suitability of the models created.

6.5.1 Creating generalized Petri net models for the process flows

Analysis of each of the flows described in sections 6.3.1 to 6.3.7 shows a pattern involving the interplay of two fundamental flows: one involving the selection of dates and the other dealing with slot selection. It therefore makes sense to model these two

fundamental flows in their nuanced details, as the combination of those two using their specific variants as applicable would embody the respective flows as described in sections [6.3.1](#) to [6.3.7](#).

The first of the two fundamental flows that we have identified deals with the selection of dates. We'd refer to this as *Block I* of the generalized flow. There are two variants of the flow for the selection of dates.

The first is the DateSelector, an example of which can be found as a constituent of the flow of section [6.3.1](#). The purpose of the flow for selection of dates here is to evaluate if the date being considered (Requested Date) falls within the range of dates opened up / made available for allocation. If it does, subsequently the flow for dealing with slots would be invoked for that particular date (Requested Date). Regardless of whether the execution of this flow with the Requested Date does or does not result in a Sales Executive assignment, there is no further iteration here with any other date.

The second and final variant of the flow for selection of dates is the DateIterator, an example of which can be found as a constituent of the flow of section [6.3.2](#). The flow for selection of dates here initially evaluates if the date being considered (Requested Date) falls within the range of dates opened up / made available for allocation, and if it does, invokes the flow for dealing with slots subsequently for that particular date (Requested Date). However, in case the flow for dealing with slots subsequently doesn't result in a Sales Executive assignment, there would be iterations of the flow for the selection of dates taking place whereby instances of dates (Alternate Dates) would get passed serially as input to the flow for the selection of dates with the subsequent invocation of the flow for dealing with slots, until either an assignment is achieved in the latter or all possible iterations are exhausted with no resultant assignment.

The second fundamental flow deals with the selection of slots. We'd refer to this as *Block II* of the generalized flow. There are two variants of this flow as well.

The first is the SlotAllocator, an example of which can be found as a constituent of the flow of section [6.3.1](#). It is utilized for the sole purpose of creating a new assignment of Sales Executive for the date being considered (Requested Date), if possible, and

there is no need for un-assigning a previous allocation since no such prior assignment exists in the system for the lead in question.

The second and final variant of the flow for dealing with slots is the SlotUpdater, an example of which can be found as a constituent of the flow of section 6.3.2. It consists of two elements - one to deal with the new assignment of Sales Executive for the date being considered (Requested Date) which is nothing but the SlotAllocator which we discussed earlier, and another needed for un-assigning a previous allocation since we do have a prior assignment existing in the system for the lead in question that needs to be un-assigned, which is effectively a SlotDeallocator.

Let's now summarize these variants of the two generalized flows more succinctly so that we can use them to categorize the flows in the previous section into a matrix.

- *Flow for selection of dates:* Here we'd come across two variants - DateSelector and DateIterator. The former i.e. DateSelector is utilized to evaluate if the date being considered falls within the range of dates opened up / made available for allocation. The latter i.e. DateIterator is utilized for similar purposes but in an iterative mode where instances of dates are passed serially as input to it with the subsequent invocation of the flow for dealing with slots, until either an assignment is achieved in the latter or all possible iterations are exhausted with no resultant assignment. Hence in order to model the flow for selection of dates we would model the DateSelector using Petri net modelling and point out the distinctions for the DateIterator.
- *Flow for dealing with slots:* There are two variants to be found here - SlotAllocator and SlotUpdater. The latter i.e. SlotUpdater essentially consists of a SlotAllocator (for a new assignment) and a SlotDeallocator (for un-assigning a previous allocation). Thus SlotAllocator is essentially a subset of SlotUpdater where the SlotDeallocator doesn't get triggered. Hence in order to model the flow for dealing with slots it would suffice to depict the SlotUpdater using Petri net modelling.

From our above understanding, we can now categorize the flows in sections 6.3.1

to 6.3.7 into a matrix using two dimensions for classification, one for Date Selection (indicating the DateSelector or the DateIterator as applicable) and the other for Slot Selection (indicating the SlotAllocator or the SlotUpdater as the case may be), as shown in Table 6.1.

Table 6.1: Categorization of flows described in sections 6.3.1 to 6.3.7 according to the composition of the respective variants of the date selection flow and the slot selection flow as applicable.

	SlotAllocator	SlotUpdater
DateSelector	6.3.1	
DateIterator	6.3.3, 6.3.4	6.3.2, 6.3.5, 6.3.6, 6.3.7

In line with our reasoning established in the preceding paragraphs, the generalized Petri net modelling of the overall flow would consist of modelling the following two building blocks:

1. *Block I*: here we will model the flow for the selection of dates by considering the DateSelector, and point out the distinctions of the DateIterator (the other variant) with the DateSelector to arrive at a generalized Petri net model of this block.
2. *Block II*: here we will model the flow for selection of slots by focusing on the SlotUpdater to arrive at a generalized Petri net model of this block. As already explained the other variant, SlotAllocator, is essentially a subset of SlotUpdater.

Let us now construct the Petri net models for these building blocks.

6.5.2 Petri net modelling of *Block I* (flow for selection of dates)

Here we'd construct a generalized Petri net model for *Block I*, the flow for finding out if a date requested by a lead is within the range of dates considered by the system for allocation. As mentioned, this block consists of two variations, DateSelector and DateIterator, as shown in Fig. 6-2 and Fig. 6-3 respectively.

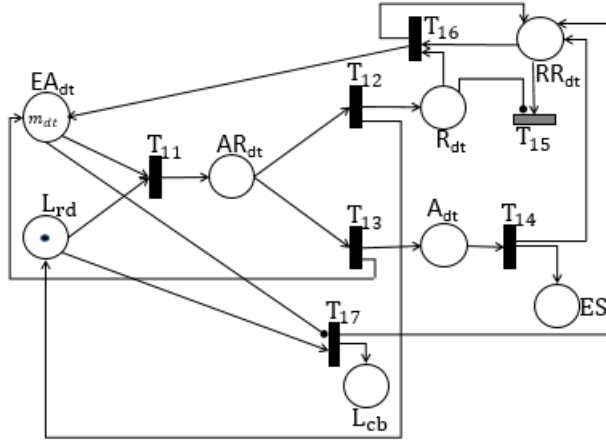


Figure 6-2: Petri net diagram of *Block I* first variant - DateSelector

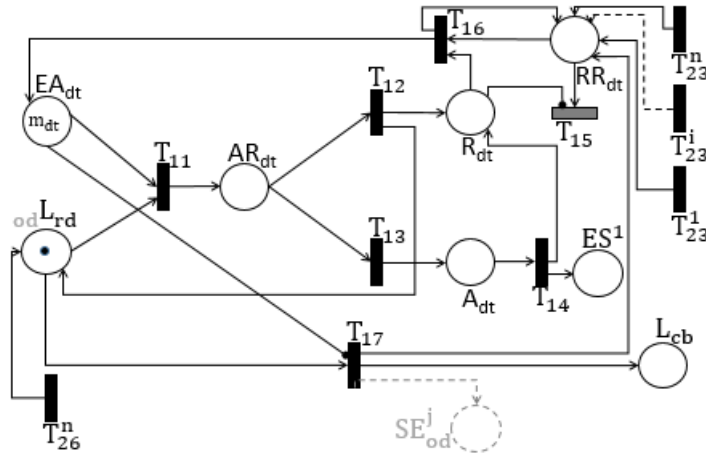


Figure 6-3: Petri net diagram of *Block I* second variant - DateIterator

Let us define the legends we'd be using to explain the modelling of this block.

Legends:

DateSelector:

L_{rd} - Lead requesting to schedule an appointment for Requested Date rd ;

EA_{dt} - Evaluate Allocation possibility of the given date from the date range dt ;

m_{dt} - Instances of dates in the range opened up / made available for allocation;

AR_{dt} - Accept or Reject date dt (evaluate whether the instance dt matches rd);

R_{dt} - Reject date dt since it doesn't match with rd ;

A_{dt} - Accept date dt since it matches with rd ;

RR_{dt} - Return Rejected date instances;

L_{cb} - Lead to be called back at a later date;

ES - Evaluate Suitability of Sales Executives;

DateIterator:

${}_{od}L_{rd}$ - Lead requesting to schedule an appointment for date rd , with possible presence of an Original Date od (in case the request is for rescheduling);

SE_{od}^j - Possible un-allocation of the previous assignment of j^{th} Sales Executive for the said lead on Original Date od ;

ES^1 - Evaluate Suitability of Sales Executives starting with the 1st Sales Executive in the allocation list;

Let's first consider the first variant, the DateSelector, as shown in Fig. 6-2.

Transition T_{11} Rule (Synchronized):

Input Place: The place EA_{dt} provides a collection of m_{dt} tokens corresponding to the m_{dt} instances of dates in the range opened up / made available for allocation. Presence of a token in the place L_{rd} indicates the request of the lead L to schedule an appointment for the Requested Date rd . When tokens are present, these input places of T_{11} are synchronized for the firing of transition T_{11} .

Result: Out of m_{dt} tokens available in the place EA_{dt} , one token is taken, leaving $(m_{dt}-1)$ tokens available in the place EA_{dt} . As the token from L_{rd} is taken, L_{rd} no longer has any token present. One token is deposited in the place AR_{dt} .

Transitions T_{12} and T_{13} Rules (Conflict):

Input Place: Presence of a token in the place AR_{dt} enables the two transitions T_{12} and T_{13} but only one can fire at a time.

Result (Transition T_{12} Rule): If the date token taken from EA_{dt} does not match with the date rd requested by the lead L , the transition T_{12} gets fired, depositing a token each in the place R_{dt} (Rejected Dates) as well as the place L_{rd} , which can potentially re-enable the firing of the transition T_{11} . The circular firing sequence $T_{12} \rightarrow L_{rd} \rightarrow T_{11} \rightarrow AR_{dt} \rightarrow T_{12}$ can iterate consecutively for n times (say) (with $0 \leq n \leq m_{dt}$), resulting in T_{12} being fired for n times, leaving $(m_{dt}-n)$ tokens available in the place EA_{dt} , n tokens being deposited in the place R_{dt} and one token in L_{rd} .

Result (Transition T_{13} Rule): If the date token taken from EA_{dt} matches with the one requested by the lead L , the transition T_{13} gets fired and a token is deposited in the place A_{dt} (Accepted Dates) and a token is deposited in the place EA_{dt} . After the firing sequence $T_{12} \rightarrow L_{rd} \rightarrow T_{11} \rightarrow AR_{dt} \rightarrow T_{12}$ iterating consecutively for n times, ($n < m_{dt}$), resulting in T_{12} being fired for n times, T_{11} is fired leaving $(m_{dt}-n-1)$ tokens available in the place EA_{dt} and no token in L_{rd} , followed by T_{13} , leaving $((m_{dt}-n-1)+1)$ i.e. $(m_{dt}-n)$ tokens available in the place EA_{dt} (as a token is returned to EA_{dt}) and one token in A_{dt} .

Transition T_{14} Rule:

Input Place: Presence of a token in the place A_{dt} , indicating the date requested by the lead L matches with one of the dates in the range opened up so far, thus enabling the transition T_{14} .

Result: As the token from A_{dt} is taken, A_{dt} no longer has any token present. A token each is concurrently deposited in two places, ES and RR_{dt} .

The presence of a token in the place ES would initiate slot selection which is in the scope of *Block II*.

Transition T_{16} Rule (Synchronized):

Input Places: The place R_{dt} has n tokens ($0 \leq n \leq m_{dt}$). Presence of a token in the place RR_{dt} would enable the transition T_{16} when $n > 0$ for R_{dt} . With tokens present, these input places of T_{16} are synchronized for the firing of transition T_{16} .

Result: The token from RR_{dt} is taken leaving no token there.

With n tokens available in the place R_{dt} , $1 \leq n \leq m_{dt}$, one is taken leaving $(n - 1)$ tokens remaining in R_{dt} . A token is deposited in the place EA_{dt} which now has a collection of $(m_{dt} - n + 1)$ tokens. The place RR_{dt} will have the token deposited back while the place R_{dt} has $(n - 1)$ tokens, enabling the transition T_{16} to be fired once again (provided $n > 1$).

This circular firing sequence will continue for n times till there is no token left in R_{dt} i.e. all n tokens are removed from it. The place EA_{dt} will now have a collection of $((m_{dt} - n) + n)$ tokens i.e. m_{dt} tokens. The place RR_{dt} will have one token.

Sink Transition T_{15} Rule (Synchronized):

Input Places: The place RR_{dt} , populated with one token, and the absence of a token (inhibitor) in R_{dt} will enable the sink transition T_{15} .

Result: It will take the token from RR_{dt} , making it empty. This will also mark the termination/completion of the flow for date selection.

Transition T_{17} Rule (Synchronized):

Input Places: When none of the m_{dt} dates match the date for the lead, it implies that the transition T_{12} has fired for m_{dt} times ($n = m_{dt}$), leaving the place EA_{dt} without any token (and the place R_{dt} with m_{dt} tokens). The inhibitor edge from this input place and the presence of a token in the input place L_{rd} would enable the transition T_{17} to fire.

Result: The token from the input place L_{rd} is removed. A token is deposited in the place indicating that a callback needs to be arranged for the lead L at a later date. A token is deposited in the place RR_{dt} .

Now let us consider the second variant, the DateIterator, as shown in Fig. 6-3.

While there is a significant similarity with the first variation (the DateSelector), the key distinction here is that different dates (Alternate Dates) are considered iteratively as dt by the DateIterator when the previous iteration with the Requested Date or an Alternate Date hasn't resulted in an assignment for a Sales Executive.

The specific differences that the DateIterator has with the DateSelector are as follows.

Transition T_{11} Rule (Synchronized):

Input Place: As is the case for Transition T_{11} with the DateSelector, the place EA_{dt} would be populated with a collection of m_{dt} tokens corresponding to the m_{dt} instances of dates in the range opened up / made available for allocation.

However, the other input place is now indicated by odL_{rd} with the prefix od added to L_{rd} and shown as greyed out (indicating it's optional) in Fig. 6-3. As is the case with the DateSelector, the suffix rd indicates the request of the lead L to schedule an appointment for date rd . However, the distinction here is the presence of the optional (greyed out) prefix od , indicating the possibility of an Original Date od being present in case this happens to be a reschedule of date.

Result: Similar to the *Result* for Transition T_{11} for the DateSelector.

Transition T_{17} Rule (Synchronized):

Input Place: Similar to the *Input Place* for Transition T_{17} for the DateSelector.

Result: While other things remain similar with the *Result* for the Transition T_{17} for the DateSelector, the distinction here is the addition of the optional (greyed out) presence of the place SE_{od}^j . A token may be deposited there in case there exists a prior assignment involving Original Sales Executive SE^j on

the Original Date od which now needs to be un-assigned based on the current request. Since this may or may not happen depending on the case, the place SE_{od}^j is shown as greyed out.

In addition the DateIterator has inputs from transitions $T_{23}^1 \dots T_{23}^i \dots T_{23}^n$ from *Block II* into the place RR_{dt} , one of which would fire in case one of the Sales Executives $SE^1 \dots SE^i \dots SE^n$ has been finally assigned. It is also possible that none of those eventually fires, in case no assignment is possible for any Sales Executive.

A key feature of the DateIterator is its execution in an iterative manner, initiated by the Transition T_{26}^n from *Block II* since it deposits a token in the place odL_{rd} indicating the request of the lead L to iterate the process of scheduling an appointment as the preceding iteration with the Requested Date or the Alternate Date hasn't resulted in an assignment and a new iteration should be attempted now with a new Alternate Date - each such date being passed as rd .

6.5.3 Petri net modelling of *Block II* (flow for selection of slots)

Here we'd construct a generalized Petri net model for *Block II*, the flow for finding out a suitable (date-time) slot amongst all the Sales Executives available to fulfill the lead's requirement, once it is validated that the date requested by the lead is within the range of dates considered by the system for allocation (which is handled by *Block I*). Thus an underlying assumption in this section is that a particular date dt (which can either be the Requested Date or Change Date or Alternate Date as the case may be, but not the Original Date) is being considered.

As mentioned, there are two variants, SlotAllocator and SlotUpdater, to model the flow of this functionality, and Fig. 6-4 combines the essential features of those two which we'd describe subsequently, keeping in mind that a particular date dt is to be considered.

It is to be noted here that we have not explicitly shown ES^{i+1} or L_{cb} as the output of transition T_{26}^i in Fig. 6-4, since a token would get inserted into ES^{i+1} or L_{cb} only

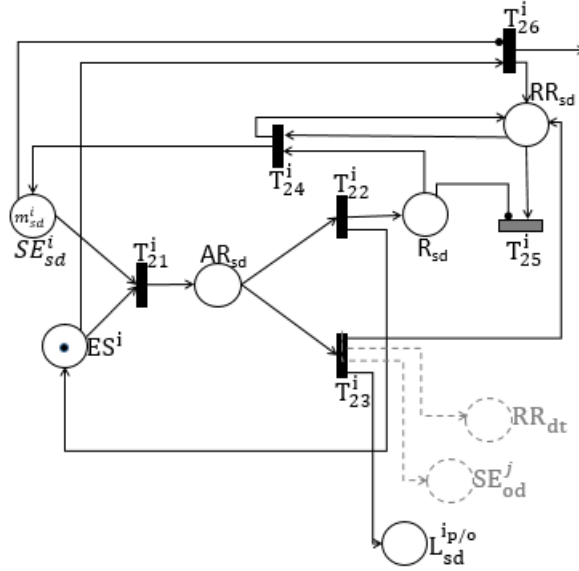


Figure 6-4: Petri net diagram of *Block II - SlotAllocator and SlotUpdater*

for certain specific scenarios as described under *Transition T_{26}^i Rule (Synchronized)* later in this section. However when we would construct the reachability tree, we introduced those (i.e. ES^{i+1} or L_{cb}) as the output of transition T_{26}^i , for the sake of completeness of the markings and the traversal in the reachability tree.

Let us define the legends we'd be using to explain the modelling of this block.

Legends:

SlotAllocator and SlotUpdater:

ES^i - Evaluate Suitability of i^{th} Sales Executive;

SE_{sd}^i - Collection of assignable slots sd prepared for i^{th} Sales Executive (for the date considered dt);

m_{sd}^i - instances of slots (collection of sd) available to be assigned for i^{th} Sales Executive for date dt ;

AR_{sd} - Accept or Reject slot sd ;

R_{sd} - Reject slot sd ;

RR_{sd} - Return Rejected slot instances;

RR_{dt} - Return Rejected date instances;

$L_{sd}^{i_p/o}$ (for *SlotAllocator*) - Matching of i^{th} Sales Executive (SE^i) for the lead for slot sd for the first time (with no previous allocation to be unassigned), indicating a provisional matching;

$L_{sd}^{i_p/o}$ (for *SlotUpdater*) - Matching of i^{th} Sales Executive (SE^i) for the lead for slot sd , which is either $L_{sd}^{i_p}$ indicating SE^i is an Alternate Sales Executive (a provisional matching) where $i \neq j$, or else it is $L_{sd}^{j_o}$ indicating the matching is for the Original Sales Executive (SE^j) where $i = j$;

N.B. The following, as mentioned in the previous para, are not shown in Fig. 6-4 but indicated in the reachability tree:

ES^{i+1} - Evaluate Suitability of $(i + 1)^{th}$ Sales Executive;

L_{cb} - The lead needs to be called back later since the assignment of a Sales Executive hasn't been possible;

SlotUpdater only:

SE_{od}^j - Possible un-allocation of the previous assignment of j^{th} Sales Executive for the said lead on Original Date od ;

The allocation process for a lead is initiated by trying out the assignment process with the 1st Sales Executive in the pecking order of Sales Executives, indicated by SE^1 , and in case it doesn't work out then the assignment process would be tried with the next Sales Executive, and so on. However, for elaboration purposes, we would generalize by considering the assignment process with the i^{th} Sales Executive, indicated by SE^i , where $i = 1 \dots n$ (n being the maximum count of Sales Executives possible).

Transition T_{21}^i Rule (Synchronized):

Input Places: The place SE_{sd}^i is populated with a collection of m_{sd}^i tokens, representing the respective instances of available slots for the accepted date dt for the i^{th} Sales Executive. When a token is present in the place ES^i , the input

places of transition T_{21}^i are synchronized to initiate the firing of the transition T_{21}^i .

Result: The place SE_{dt}^i has one token removed and has $(m_{sd}^i - 1)$ tokens now, and the place ES^i no longer has a token. A token is inserted in the place AR_{sd} .

Transitions T_{22}^i and T_{23}^i Rules (Conflict):

Input Place: Presence of a token in the place AR_{sd} enables the two transitions T_{22}^i and T_{23}^i but only one can fire at a time.

Result (Transition T_{22}^i) Rule: In case the slot is rejected (when the slot doesn't match with the one requested by the lead), the transition T_{22}^i gets fired. The token is removed from the place AR_{sd} . A token each is deposited concurrently in the place R_{sd} (Rejected slots) and the place ES^i .

The circular firing sequence $T_{21}^i \rightarrow AR_{sd} \rightarrow T_{22}^i \rightarrow ES^i \rightarrow T_{21}^i$ can iterate consecutively (say n times; $n \leq m_{sd}$) resulting in $(m_{sd}-n)$ tokens remaining available in the place SE_{sd}^i and n tokens deposited in the place R_{sd} . The place ES^i would have a token.

Result (Transition T_{23}^i) Rule: If the slot is accepted (the slot now matches with the one requested by the lead), the transition T_{23}^i gets fired. A token is inserted in the place RR_{sd} . This would initiate the return of the Rejected slots. Concurrently a token is deposited in the place $L_{sd}^{ip/o}$, implying that a slot sd for the lead L has now been assigned to the i^{th} Sales Executive, and it is either assigned to the Original Sales Executive which would be indicated by the matching L_{sd}^{io} or an Alternate Sales Executive which would be indicated by a provisional matching L_{sd}^{ip} .

A token may be deposited in the place SE_{od}^j in case there exists a prior assignment involving Original Sales Executive SE^j on the Original Date od which now needs to be un-assigned based on the current request. Since this may or may not happen depending on the case, the place SE_{od}^j is shown as greyed out.

A token may or may not be deposited in the place RR_{dt} . RR_{dt} is greyed out for that reason.

When SlotAllocator is involved, the place RR_{dt} won't feature in the generalized PN model; however, in the case of SlotUpdater, it will. SlotAllocator is applicable for sections 6.3.1, 6.3.3 and 6.3.4; and for those cases the transition T_{23}^i has no connection to RR_{dt} .

Transition T_{24}^i Rule (Synchronized):

Input Place: The place R_{sd} has n tokens ($0 \leq n \leq m_{sd}$). The presence of a token in the place RR_{sd} would enable the transition T_{24}^i when n is non-zero for R_{sd} . With tokens present, these input places are synchronized for the firing of transition T_{24}^i .

Result: The token from the place RR_{sd} is removed. The place R_{sd} has one token removed, and now has $(n - 1)$ tokens. A token is inserted in the place SE_{sd}^i which now has a collection of $(m_{sd}^i - n + 1)$ tokens. A token is deposited in the place RR_{sd} , which along with the presence of token(s) in R_{sd} would enable the firing of the transition T_{24}^i repetitively till there is no token left in R_{sd} i.e. all n tokens are removed from it. The place SE_{sd}^i will now have a collection of $(m_{sd}^i - n + n)$ tokens i.e. m_{sd}^i tokens. The place RR_{sd} will have one token.

Transition T_{26}^i Rule (Synchronized):

Input Place: As long as there is a token in SE_{sd}^i , the transition T_{26}^i can not fire. When none of the m_{sd}^i slots match with the one requested by the lead, the place SE_{sd}^i is without any token. The presence of a token in the place ES^i , along with the inhibitor edge from SE_{sd}^i in the above mentioned condition, would synchronize the input places of T_{26}^i to be fired.

Result: The token from the place ES^i is removed. A token is deposited in the place RR_{sd} .

One output of T_{26}^i has an open-ended arrow, to denote the following scenario. A token would get deposited in the place ES^{i+1} , signifying that the i^{th} Sales

Executive's slots are not matching the lead's requirement and the $i + 1^{th}$ Sales Executive's slots are to be explored next, as long as $i \leq n - 1$, where n is the maximum of Sales Executives available. However, if SE^i happens to be the last Sales Executive in the pecking order (i.e. $i = n$, where n is the maximum of Sales Executives available), then

either it leads to the next iteration of *Block I* if this *Block II* Petri net represents a SlotUpdater, by having a token deposited in the input place for the lead in *Block I*

or

it leads to a callback for the lead, if this *Block II* Petri net represents a SlotAllocator.

Sink Transition T_{25}^i Rule (Synchronized):

Input Place: The place RR_{sd} , populated with one token, and the absence of a token in R_{sd} (inhibitor) will enable the sink transition T_{25}^i .

Result: It will take the token from RR_{sd} , making it empty.

6.6 Verification of our Petri net models

Let us now carry out the verification exercise for our Petri net models by evaluating whether they exhibit the desired properties listed in section 6.4.1. We would be adopting the behavioural approach for our Petri net analysis for this verification exercise. Specifically, we'd focus on the following behavioural properties:

1. *Reachability:*

This property directly corresponds to the criterion *Reach a specific state* in section 6.4.1.

Reachability is a fundamental basis for studying the dynamic properties of a system. We need to exhaustively enumerate all the possible reachable markings

by firing the enabled transitions one by one, starting with an initial marking and reach a new state (marking) after each firing, resulting in a tree representation of the markings in the process. The firing of an enabled transition will change the token distribution (marking) in the net according to the transition rules associated with it. A sequence of firings will result in a sequence of markings thus allowing us to arrive at the reachability tree.

The analysis of the reachability trees can help build up a repository of useful information (e.g. specification of the set of reachable markings) [65]. If we are to ascertain whether the modelled system can reach a specific state M_i , which essentially signifies the demonstration of a functional behavior that is desired, we need to figure out a sequence of transitions that will get fired to transform an initial marking M_0 to M_i . The sequence of firings represents the required functional behavior.

It is worth noting that systems in real life can attain a given state by exhibiting different variants of functional behavior. The equivalence of this in a Petri net model is manifested by the possibility of having different transition firing sequences (which in essence represents the required functional behavior), each of which can transform a marking M_0 to the required marking M_i [34].

We would be creating the reachability tree for the Petri net models for both *Block I* and *Block II* which would allow us to find the sequence of firings of transitions which would result in transforming the marking M_0 to the marking M_i .

2. *Boundedness:*

This Petri net property aids in identifying the overflow possibilities in the modelled system [34], and it directly corresponds to the criterion *Prevent overflow* in section 6.4.1.

Considering the corresponding analogy of the allocation node in a manpower allocation facility in section 6.4.1, its representation as a bounded place will guarantee that this capacity will not be exceeded by associated operations [65].

This needs to be ensured in the models that we will create by having them exhibit the Boundedness property.

Also, unbounded places are potential bottlenecks. We need to verify that there are no such instances in the models created, thus preventing the possibility of any potential bottleneck.

3. *Liveness*:

Liveness, a Petri net property tied to the concept of deadlocks and deadlock-freeness [65], directly corresponds to the criterion *Prevent deadlock* in section 6.4.1.

A transition that cannot fire is a redundant transition and should such a transition exist in a net model, it warrants identification as it is likely to be indicative of a modelling error or an inconsistency in the system being modelled. A transition is dead in a marking if no sequence of transition firings exists to enable it [65], and we have to make sure that there is no redundant transition or dead transition in our model.

A transition is potentially fireable if there exists some firing sequence that enables it, and we have to ensure that every transition in our model is potentially fireable.

4. *Fairness*:

Fairness is a Petri net property that directly corresponds to the criterion *Fairness* in section 6.4.1.

Related problems are studied in the Petri net model where actions correspond to the firings of transitions. An action is announced if a transition becomes fireable, and it is performed if this transition fires. Fairness in Petri nets means firing of transitions in the order of their enabling. Therefore, fairness is a property of firing sequences: some of them satisfy fairness conditions while others do not and have to be excluded.

6.6.1 Verification of Petri net modelling - *Block I*

Here are the properties exhibited by adopting the behavioural approach for our Petri net analysis of *Block I*.

- **Reachability:**

We construct the reachability tree in Fig. 6-5 as we consider the DateSelector for *Block I*. As we have a truly finite tree in this case, it also follows that here the terms reachability tree and coverability tree are synonymous. Also, the Petri net in Fig. 6-5 is a finite capacity net.

Let us now highlight the following markings shown in Fig. 6-5 which depict significant events attained (by firing of certain sequential events as described) in the Petri net that we have modelled.

The marking M_2'' , indicating the acceptance at the first iteration itself, is reached by the sequential firing of T_{11} and T_{13} from M_0 , represented by $\sigma_2 = T_{11}T_{12}$.

The marking M_4'''' , indicating the eventual acceptance after at least one cycle of rejection, is reached from M_0 by the sequential firing of T_{11} , T_{12} , followed by $T_{11} \rightarrow T_{12}$ iterating consecutively for n times, $n = 0..(m_{dt} - 2)$, and then T_{13} .

The marking M_5''' , indicating eventual rejection, is reached from M_0 by the sequential firing of $T_{11} \rightarrow T_{12}$ iterating consecutively for m_{dt} times, followed by T_{17} . As the reachability tree has to show the intervening conflicts, this consecutive iteration of $T_{11} \rightarrow T_{12}$ for m_{dt} times is depicted in Fig. 6-5 by the sequential firing of $T_{11} \rightarrow T_{12}$ once, followed by $T_{11} \rightarrow T_{12}$ iterating consecutively for n times, $n = 0..(m_{dt} - 2)$, followed by $T_{11} \rightarrow T_{12}$ firing once, and then $T_{11} \rightarrow T_{12}$ iterating consecutively for $(m_{dt} - n - 2)$ times.

While Fig. 6-5 shows the reachability tree for *Block I* by considering the DateSelector, the reachability tree for the DateIterator would be very similar.

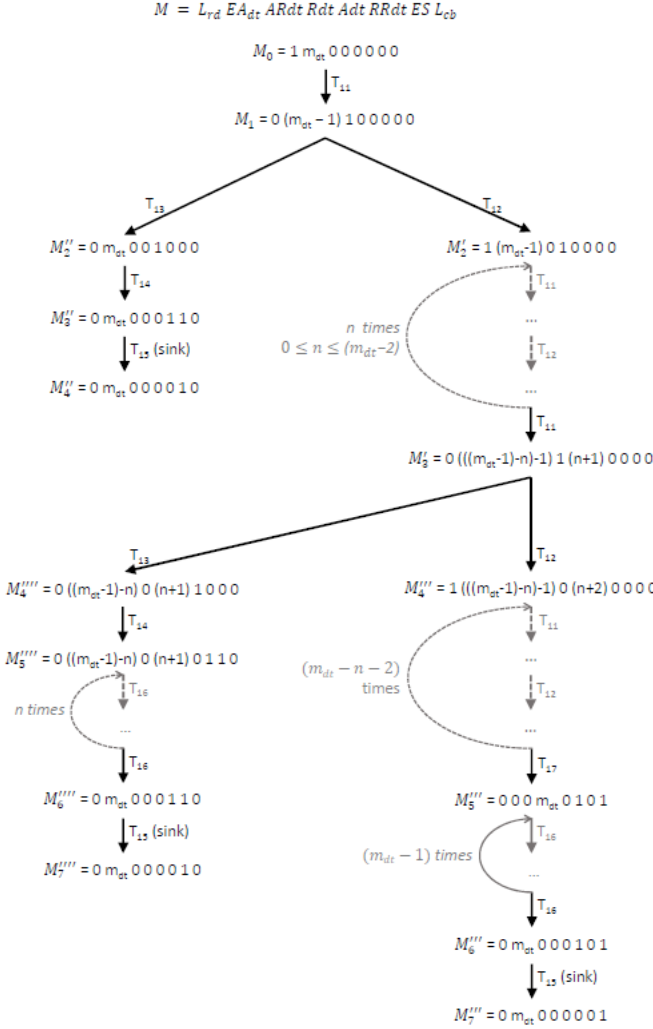


Figure 6-5: *Reachability tree for Block I (DateSelector)*

- **Boundedness:**

A PN is called k -bounded with respect to an initial marking M_0 , if each place in the net gets at most k tokens for all markings belonging to the reachability set $R(M_0)$, where k is a finite positive integer.

The DateSelector is m_{dt} -bounded, and so is the DateIterator.

- **Liveness:**

As far as liveness is concerned, we would start by defining the degrees of liveness of individual transitions first and then subsequently the degrees of liveness of the entire Petri net.

For the DateSelector associated with *Block I*, we have the following observations.

Transition T_{11} is *L1-Live*, since it will fire once at a minimum and up to m_{dt} times at a maximum (which is a finite number).

Transition T_{12} can have multiple firing possibilities:

1. it can never execute when the first token itself taken from AR_{dt} is accepted and hence transition T_{13} is fired.
2. It can fire for a maximum of m_{dt} times, which however is still a finite number.

Transition T_{12} is *L1-Live*.

Transition T_{12} is not *L2-Live* since its firing count has an upper bound of m_{dt} which is a finite number.

Transition T_{13} can have two firing possibilities:

1. it may never execute, when all the tokens taken from AR_{dt} are rejected, with transition T_{12} being fired m_{dt} times.
2. It can be fired once.

Transition T_{13} is *L1-Live*.

Transition T_{14} , like transition T_{13} , can have two firing possibilities:

1. it may never execute.
2. It can be fired once.

Transition T_{14} is *L1-Live*.

Transition T_{16} has a similar case to T_{12} . It may never execute, or can fire for a maximum of m_{dt} times, which however is a finite number.

Transition T_{16} is *L1-Live*.

Transition T_{16} is not $L2$ -Live since its firing count has an upper bound of m_{dt} which is a finite number.

Transition T_{15} will be fired once.

Transition T_{15} is $L1$ -Live.

Transition T_{17} can have two firing possibilities:

1. it may never execute.
2. It can be fired once.

Transition T_{17} is $L1$ -Live.

Firing of transition T_{13} before T_{12} is fired even once will rule out the firing of transition T_{16} .

Hence the Petri net represented by *Block I* is not Live.

However, the Petri net represented by *Block I* is strictly $L1$ -Live since it is $L1$ -Live but not $L2$ -Live (all the transitions are $L1$ -Live but none are $L2$ -Live).

The Liveness considerations for the DateIterator would be very similar.

- **Fairness:**

We'd consider two basic fairness concepts:

1. *Bounded fairness (B-fairness)*
2. *Unconditional (global) fairness*

Bounded fairness (B-fairness):

Two transitions t_1 and t_2 are said to be in a bounded-fair (B-fair) relation if the maximum number of times that either one can fire while the other is not firing is bounded.

A PN, (N, m_0) is said to be a B-fair net if every pair of transitions in the PN are in a B-fair relation.

For both DateSelector and DateIterator, any two transitions are in a B-fair relation since the maximum number of times that either one can fire while the other is not firing is always bounded.

Both DateSelector and DateIterator would be a B-fair net since every pair of transitions in each of them is in a B-fair relation.

Unconditional (global) fairness:

A firing sequence σ is said to be unconditionally (globally) fair if it is finite or every transition in the net appears infinitely often in σ .

Each firing sequence σ in both DateSelector and DateIterator is unconditionally (globally) fair since it is finite.

A PN, (N, m_0) is said to be an unconditionally (globally) fair net if every firing sequence σ from $m \in R(m_0)$ is unconditionally fair.

The DateSelector and DateIterator both would qualify to be an unconditionally (globally) fair net since every firing sequence σ for each of them is unconditionally fair.

6.6.2 Verification of Petri net modelling - *Block II*

Here are the properties exhibited by adopting the behavioural approach for our Petri net analysis of *Block II*.

- **Reachability:**

We construct the reachability tree in Fig. 6-6 focusing on the SlotUpdater for *Block II*. Since we have a truly finite tree in this case, it also follows that here the terms reachability tree and coverability tree are synonymous. Also, the Petri net in Fig. 6-6 is a finite capacity net.

As mentioned earlier, we would be introducing ES^{i+1} or L_{cb} as the output of transition T_{26}^i in the reachability tree we are about to construct even though it's not shown in Fig. 6-4, for the sake of completeness of the reachability tree.

Let us now highlight the following markings shown in Fig. 6-6 which depict significant events attained (by firing of certain sequential events as described) in the Petri net that we have modelled.

The marking M'_2 , indicating the acceptance at the first iteration itself, is reached by the sequential firing of T_{21}^i and T_{23}^i from M_0 , represented by $\sigma_2 = T_{21}^i T_{23}^i$.

The marking M_4'''' , indicating the eventual acceptance after at least one cycle of rejection, is reached from M_0 by the sequential firing of $T_{21}^i \rightarrow T_{22}^i$ once, followed by $T_{21}^i \rightarrow T_{22}^i$ iterating consecutively for n times, $n = 0..(m_{sd} - 2)$, and then T_{21}^i followed by T_{23}^i .

The marking M_5''' , indicating eventual rejection, is reached from M_0 by the sequential firing of $T_{21}^i \rightarrow T_{22}^i$ iterating consecutively for m_{sd} times followed by T_{26}^i . As the reachability tree has to show the intervening conflicts, this consecutive iteration of $T_{21}^i \rightarrow T_{22}^i$ for m_{sd} times is depicted in Fig. 6-6 by the sequential firing of $T_{21}^i \rightarrow T_{22}^i$ once, followed by $T_{21}^i \rightarrow T_{22}^i$ iterating consecutively for n times, $n = 0..(m_{sd} - 2)$, followed by $T_{11} \rightarrow T_{12}$ firing once, and then $T_{21}^i \rightarrow T_{22}^i$ iterating consecutively for $(m_{sd} - n - 2)$ times.

While Fig. 6-6 shows the reachability tree for *Block II* by considering the SlotUpdater, the reachability tree for the SlotAllocator would be very similar, with the places SE_{od}^j and RR_{dt} being omitted from the markings.

- **Boundedness:**

A PN is called k -bounded with respect to an initial marking M_0 , if each place in the net gets at most k tokens for all markings belonging to the reachability set $R(M_0)$, where k is a finite positive integer.

The Petri net in *Block II* is m_{sd} -bounded, where m_{sd} is the maximum value (upper bound) that can be assumed by m_{sd}^i . When a Sales Executive's slots for a fresh day are initialized, then there are m_{sd} slots available assuming the Sales Executive is fully available on that day.

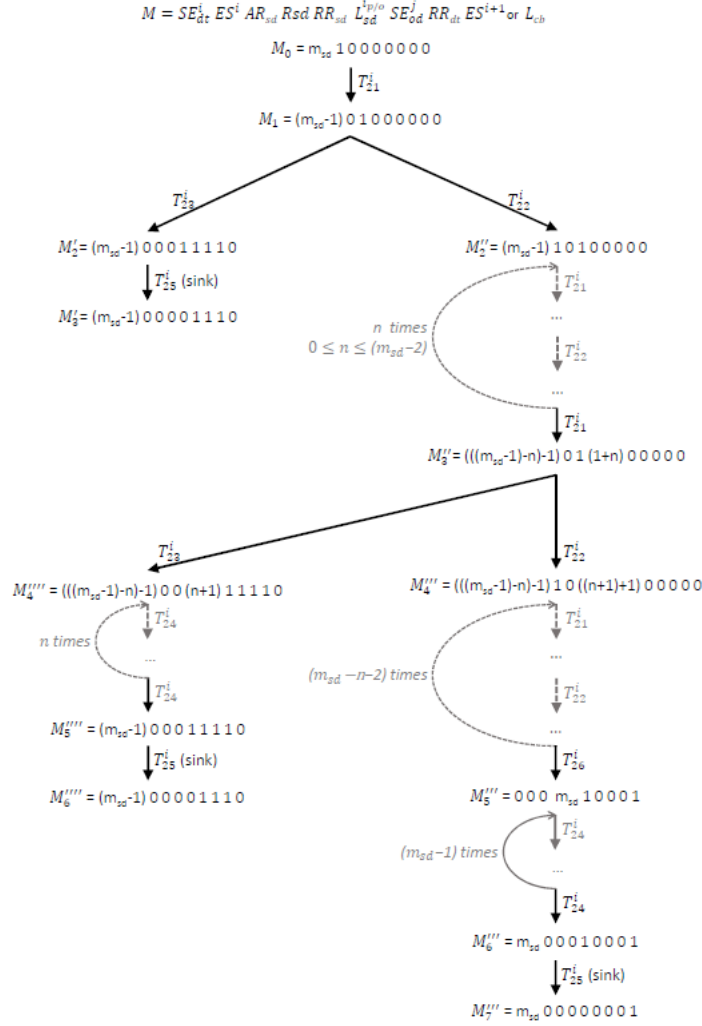


Figure 6-6: Reachability tree for Block II (SlotUpdater)

- **Liveness:**

As before, we would first define the degrees of liveness of individual transitions followed by the degrees of liveness of the entire Petri net.

For the Petri net in *Block II*, we have the following observations.

Transition T_{21}^i will fire at least once, and it can fire for a maximum of m_{sd}^i times, which however is a finite number.

Transition T_{21}^i is *L1-Live*.

Transition T_{22}^i can have multiple firing possibilities:

1. it may never execute, when the first token itself taken from AR_{sd} is accepted and hence transition T_{23}^i is fired.
2. It can fire for a maximum of m_{sd}^i times, which however is a finite number.

Transition T_{22}^i is *L1-Live*.

Transition T_{23}^i can have multiple firing possibilities:

1. it may never execute.
2. It can fire once.

Transition T_{23}^i is *L1-Live*.

Transition T_{24}^i will fire at least once, and it can fire for a maximum of m_{sd}^i times, which however is a finite number.

Transition T_{24}^i is *L1-Live*.

Transition T_{25}^i is *L1-Live*, since it will fire only once.

Transition T_{26}^i can have multiple firing possibilities:

1. it may never execute.
2. It can fire once.

Transition T_{26}^i is *L1-Live*.

The firing of transition T_{23}^i before T_{22}^i is fired even once will rule out the firing of transition T_{22}^i as well as transition T_{24}^i .

Hence the Petri net represented by *Block II* is not Live.

However, the Petri net represented by *Block II* is strictly *L1-Live* since it is *L1-Live* but not *L2-Live* (all the transitions are *L1-Live* but none are *L2-Live*).

- **Fairness:**

We'd consider two basic fairness concepts:

1. *Bounded fairness (B-fairness)*
2. *Unconditional (global) fairness*

Bounded fairness (B-fairness):

Two transitions t_1 and t_2 are said to be in a bounded-fair (B-fair) relation if the maximum number of times that either one can fire while the other is not firing is bounded.

A PN, (N, m_0) is said to be a B-fair net if every pair of transitions in the PN are in a B-fair relation.

For the Petri net in *Block II*, any two transitions are in a B-fair relation since the maximum number of times that either one can fire while the other is not firing is always bounded.

The Petri net in *Block II* would be a B-fair net since every pair of transitions in each of them is in a B-fair relation.

Unconditional (global) fairness:

A firing sequence σ is said to be unconditionally (globally) fair if it is finite or every transition in the net appears infinitely often in σ .

Each firing sequence σ in the Petri net in *Block II* is unconditionally (globally) fair since it is finite.

A PN, (N, m_0) is said to be an unconditionally (globally) fair net if every firing sequence σ from $m \in R(m_0)$ is unconditionally fair.

The Petri net in *Block II* would be an unconditionally (globally) fair net since every firing sequence σ therein is unconditionally fair.

6.6.3 Additional behavioural properties exhibited by the Petri net models for *Block I* and *Block II*

The following properties are exhibited by the Petri net models based on the behavioural analysis in addition to the ones described in the preceding sections.

- **Synchronic Distance:**

Synchronic Distance measures the degree of mutual dependence between two transitions in a PN.

For both DateSelector and DateIterator in *Block I*, the Synchronic Distance can at most be m_{dt} i.e. the max number of days opened up for allocation.

For *Block II*, the Synchronic Distance can be m_{sd} , where m_{sd} is the maximum value (upper bound) that can be assumed by m_{sd}^i . When a Sales Executive's slots for a fresh day are initialized, then there are m_{sd} slots available assuming the Sales Executive is fully available on that day.

- **Consistency:**

A PN is Consistent with respect to an initial marking m_0 if and only if the coverability tree has a directed circuit (not necessarily elementary) containing all the transitions at least once. It is Partially Consistent if such a directed circuit contains only some of the transitions.

The DateSelector in *Block I* is Partially Consistent since the coverability tree for that contains a directed circuit with only some of the transitions. When it is accepted, the token doesn't come back to L_{rd} , implying the circuit is not completed. However, when the token does come back to L_{rd} , it implies that acceptance never occurred, and hence the circuit contains only some of the transitions (since the transitions for acceptance like T_{13} and T_{14} will not execute).

The same considerations apply to the DateIterator in *Block I*.

Block II is Partially Consistent since the coverability tree for that contains a directed circuit with only some of the transitions. When it is accepted, the token doesn't come back to ES^i , implying the circuit is not completed. However, when the token does come back to ES^i , it implies that acceptance never occurred, and hence the circuit contains only some of the transitions (with the transition for acceptance excluded).

6.6.4 Summary of the verification exercise conducted on the Petri net models for *Block I* and *Block II*

Carrying out the behavioural analysis for the Petri net models for *Block I* as well as *Block II* earlier in this section has helped us verify that the set of properties stated in section 6.4.1 are indeed exhibited by the generalized Petri net models that we have constructed.

The following observations summarize the verification of our Petri net models with respect to the said set of properties by conducting the behavioural analysis:

1. *Reachability:*

We have created the reachability trees for the Petri net models for both *Block I* and *Block II*, allowing us to find the sequence of firings of transitions which would result in transforming a marking M_0 to M_i , where M_i represents a desired specific state and M_0 the initial state, in either of those two nets.

2. *Boundedness:*

Since there are finite markings and at each marking the number of tokens in each place can assume a value from the range $0 \dots m_{dt}$, the Petri net model for *Block I* is m_{dt} -bounded.

Similarly, the Petri net in *Block II* is m_{sd} -bounded, where m_{sd} is the maximum value (upper bound) that can be assumed by m_{sd}^i . When a Sales Executive's slots for a fresh day are initialized, then there are m_{sd} slots available assuming the Sales Executive is fully available on that day.

In a real-life environment, the boundedness or safeness of a Petri net indicates the absence of overflows in the modelled system. This has been ensured in the models that we have created.

Also, unbounded places are potential bottlenecks. There are no such instances in the models created and hence we don't have a potential bottleneck.

3. *Liveness*: A transition that cannot fire needs to be identified since it may represent an error in the model or an inconsistency in the system being modelled [65]. We have seen that there is no redundant transition or dead transition in our model.

A transition is potentially fireable if there exists some firing sequence that enables it. Every transition in our model is potentially fireable.

The Petri net models for both *Block I* and *Block II* are strictly L1-Live, and hence would be deadlock-free.

4. *Fairness*:

The Petri net models for both *Block I* and *Block II* qualify to be B-fair nets as well as unconditionally (globally) fair nets as established earlier. The fairness gives the said Petri nets more computational power as there is no need to modify those nets by excluding non-conforming firing sequences since no such sequence exists for those nets [70].

6.7 Summary of chapter

The focus in our current chapter has been on *Assigning leads to appropriate Sales Executives*. This is a workflow process dealing with a number of situational variables arising out of scheduling/rescheduling of site visit requests from leads who themselves would belong to different categories, and the availability/unavailability of Sales Executives on the said day of the site visit along with applicable constraints (like retaining the Sales Executive as far as practicable when the lead has already made a prior site visit or avoiding that situation altogether when the prospect has made an explicit request to assign an alternate Sales Executive).

We have decided to use Petri nets to model the same given the widespread applicability of Petri net in modelling related application domains and workflows. The assignment process is first detailed out and subsequently the building blocks associated with the constituent flows are modelled using Petri net. The resulting generalized

Petri net models have system properties that can be directly linked to certain desirable performance criteria of systems such as reachability, liveness, boundedness and fairness, as borne out by the behavioural analysis conducted on them to obtain a verification of the suitability of our modelling.

A key objective has been to arrive at generalized Petri net models for the overall assignment process thereby facilitating the design process. However, the value of the Petri net based modelling exercise undertaken in terms of clarifying and improving our design understanding that is key to successful implementation applies not only to the business scenarios in scope for this chapter (i.e. real estate CRM), but it can also extend to other industry domains where similar Petri net based modelling and analysis can be conducted.

We can build upon this even further in our subsequent endeavours by considering each individual scenario of the assignment and creating a specific Petri net model by leveraging the generalized models already built. The process would entail the selection of the relevant constituent flows from the overall generalized Petri net models as applicable for the scenario being considered, followed by the composition of the selected flows to realize the intended Petri net model which is specific to the said scenario. This can prove to be quite valuable in lending greater clarity during implementation.

What we would like to do subsequently is to focus on those processes which would ensure the prerequisites of the assignment process have been met, and which would also carry out the post-processing requirements of the assignment process. It is important to plan out the operational aspects when the automation would be put into action, which is why we would be considering the go-live as well as steady state aspects of those processes as we would look into the operational aspects related to leads and Sales Executives as we delve into those processes into the next chapter. We would detail the functionalities and model those using Petri net, complete with the verification of those models with respect to a set of desired properties that such models should exhibit.

Petri net modelling of certain key operational aspects regarding leads and Sales Executives in real estate CRM

Contents

7.1 Background	155
7.2 System Initialization	158
7.3 Steady State Operation	158
7.4 Modelling considerations: Desired properties for the model to exhibit	170
7.5 Petri net modelling of functionalities in scope	171
7.6 Verification of our Petri net models	179
7.7 Summary of chapter	185

The focus of this chapter will be on those processes which carry out the pre-processing as well as the post-processing activities of the assignment process which we have described in chapter 6. The key outcome of those processes would be the priority setting of leads (indicative of their maturity potential) and the queue adjustment of SEs (indicative of their availability at that point in time).

One entity of utmost importance to any real estate company would be the leads. Priority is an important attribute to be considered for every lead. It is a measure

of lead's seriousness in proceeding with the deal, impacted by factors like turning up duly for planned visits or failing to do so (skipping visits). A lead expressing a clear intent of a site visit by providing a date for the first time would be accorded a default priority.

As part of system initialization (go-live), the availability of Sales Executives (SE) over a predefined date range needs to be provisioned via the initiation of slots. A slot is a time slice of a day indicating the availability of an SE to facilitate a site visit. The allocation availability of an SE is indicated by his queue value, with a higher value indicating greater availability. When a request from a lead is run through the assignment process for the first time, those SEs are considered first for allocation whose queue values are the highest. The queue adjustment for SEs would happen based on various events taking place.

There would be two variants of processes running every day. One would be a real-time variant that can be triggered any time in the day to carry out the post-processing requirements of the assignment process executed real-time to cater to first time site visit requests as mentioned above as well as subsequent requests (like date reschedule) from the leads. There would also be a batch variant that would run at the beginning of the day, processing the events that happened offline and got subsequently ingested the day before related to the site visits. Execution of both would result in priority setting of leads and corresponding queue adjustment of SEs, described in detail in the upcoming sections.

From an operational aspect, it requires a clear understanding of how these processes as part of the overall automated system should be initiated during go-live and executed during steady state in order to facilitate the assignment process. This constitutes our focus area in this chapter.

We would subsequently carry out the modelling of those functionalities using Petri net and then verify the suitability of those models by leveraging Petri net based behavioural analysis techniques. This would allow us to assess from a system designer's standpoint the designed system's ability to demonstrate the desired functional properties [34].

For the key concepts of Petri net, section 2.3 may be referred to. As far as the domain/industry is concerned, section 2.4 of this thesis provides an understanding of the background.

The remaining part of this chapter would consist of the following sections:

- Background
- System Initialization
- Steady State Operation
- Modelling considerations
- Petri net modelling of functionalities in scope
- Verification of our Petri net models
- Summary of chapter

7.1 Background

As *leads* i.e. potential customers interested in the properties of a real estate company reach out to the company using various means, telecallers would sift through the list of those leads and call up the promising ones.

A lead getting ingested into the system for the first time would provide a desired date of visit either right away or after one or more interactions with telecallers and considered a fresh lead with lead status "FRESH" and priority 1 (default).

The lead would next be allocated, following the assignment procedure, to one out of a team of SEs for site visit depending on availability. The SE would meet the said lead at the appointed time to conduct site visit. When the visit actually occurs or gets skipped (both offline events), the corresponding information would be ingested by the system later that day and processed the next day in batch mode.

The SE facilitating the visit would become the Original Sales Executive (OS) for the said lead. This connection between OS and the lead would be useful given the

usual comfort experienced by a person from the continued association with a service provider. Hence the system will try to ensure the OS retention for the lead for future visit requests unless the lead explicitly asks for OS change. The priority of the lead would be incremented upon visit completion. When this lead asks for a revisit, the priority would be incremented.

It is of course a possibility that the lead fails to turn up or skips the said visit (again an offline event which would be processed similarly to the visited event). He might also ask for a reschedule (visit at a later date) and/or ask for a change of the OS (provided he has made at least one visit previously), all of which would be real-time events and processed in a real-time manner as well. In all such cases the priority of the lead would be decremented by 1.

In case the system fails to accommodate the lead's request due to lack of capacity and arranges for a call-back at a later date, the priority of the lead would be incremented by 1. However, this effect needs to be considered along with that of the originating request. To illustrate this, when a lead asks for a reschedule that can't be accommodated by the system and a call-back is arranged, the net effect on the lead priority is nil as the incrementing and decrementing effects would cancel each other out.

Priority count can have a minimum value of 1 while it can go up to a maximum value of ψ , ψ being a positive integer capped to a specific value by the system administrator.

The load distribution for SEs via adjustment of their queue values would play an important role in tandem with the priority setting of leads that we described just now, as these two processes together would be responsible for setting up the assignment process to run in the intended manner. As we will see, the assignment process will also in turn trigger the priority setting of the leads as well as the queue adjustment of the SEs as part of its post-processing.

The queue setting of SEs would be done via initialization of the global queue value for each SE with the highest queue value possible (h), along with the initialization of all slots per day (m_{sl}) for each SE for the next p days (p , a system parameter,

signifying the system's capacity for accepting future visit requests) and setting the daily queue value for each day for each SE over the said date range to the respective global queue value, as part of go-live.

Subsequently, those daily queue values for the SE will get adjusted as fresh leads are assigned to that SE for the first time, and also based on subsequent real-time requests from those leads as already described. Queue value adjustment will also happen when the corresponding visit related events are processed in a batch mode as described earlier. Those events which are related to leads for whom the SE happens to be the OS will also result in the modification of the global queue value for the SE.

While the global queue value for an SE can get updated at any time depending on the events happening as described, it can never be negative and can't exceed the maximum queue value. Also, the daily queue value for an SE would be less than or equal to the global queue value for that SE, the difference attributable to cases involving allocation to fresh lead(s) for slot(s) on the concerned day, but it can never be negative.

Post go-live, at the start of each subsequent day dt , the slots would be initialized in batch mode for each SE SE_i for the p -th day from the run date dt i.e. $(dt + p)$. The corresponding daily queue value for that day $(dt + p)$ would be set to the SE's applicable global queue value. As mentioned, the global queue value for the SE would continue to be adjusted depending upon the specific set of events that happen to leads who have the SE as their OS.

Post visit, subsequent interactions would lead to either a progressive showing of intent from the lead eventually culminating in a sale of property (booking) or decline of interest leading to a lost opportunity. Both bookings and declines would be offline events which would take the concerned leads out of the purview of our current system, since handling of the sale of property will be done via the sales/post-sales process which is beyond our current scope.

In the subsequent sections, we will detail how the aforementioned processes should operate during system initiation (go-live) and the steady state to ensure the assignment process runs smoothly, taking into account the different events taking place over

the applicable day range.

7.2 System Initialization

In this section, we are going to describe the process execution during the system initialization, to make sure the assignment process is properly set up.

7.2.1 Slot Initialization with queue setting (Go-Live)

This would be run as a batch process.

When the system is made operational for the first time i.e. on its go-live date (dt_{gl}), it will initialize all the slots for all the SEs for p days, starting from the day after the go-live date (i.e. $dt_{gl} + 1$), since that is the earliest date any SE can carry out a site visit keeping a day's gap between the first date a booking can be made (dt_{gl}).

Each SE_i 's global queue value h_i^g for the said period would be initialized with h , and the daily queue value h_i^{dt} for each day dt over the said period would be set to the respective global queue value h_i^g , which is equal to h .

The corresponding routine `InitSlotSetQGL` (Algorithm 1) takes as input parameters dt_{gl} , p , h and m_{sl} , initializes the global queue (h_i^g) for SE_i to h and invokes the routine, `InitSlotSetQDaily`, for that SE's slot initialization and daily queue setting for each day from $(dt_{gl} + 1)$ to $(dt_{gl} + p)$, passing each available SE id SE_i and h .

The routine `InitSlotSetQDaily`, described in Algorithm 2, takes as input SE_i , date dt , queue value h^g and m_{sl} , initializes all m_{sl} slots for SE_i on dt and sets daily queue value for SE_i for date dt (h_i^{dt}) to h^g .

7.3 Steady State Operation

In this section, we will endeavour to describe how the system is operated at steady state, in terms of carrying out the operational aspects of those processes that are key to ensuring the proper execution of the assignment process.

Algorithm 1 InitSlotSetQGL: Algorithm for SE's slot initialization with queue setting during go live

Require: dt_{gl}, p, h, m_{sl}

```

1: for ( $dt = (dt_{gl} + p)$  to  $(dt_{gl} + 1)$ ) do
2:    $NumAvlblSE \leftarrow FetchSECount(dt)$ 
3:   for ( $i = NumAvlblSE$  to 1) do
4:      $SE_i \leftarrow FetchSEID(dt, i)$ 
5:      $SetGlobalQValSE(SE_i, h)$ 
6:      $InitSlotSetQDaily(SE_i, dt, h, m_{sl})$ 
7:   end for
8: end for

```

Algorithm 2 InitSlotSetQDaily: Algorithm for a specific SE's slot initialization with queue setting for a given day

Require: SE_i, dt, h^g, m_{sl}

```

1:  $SetDailyQValSE(SE_i, dt, h^g)$ 
2: for ( $sl = m_{sl}$  to 1) do
3:    $SetSlot(SE_i, dt, sl)$ 
4: end for

```

On the day the system has gone live (dt_{gl}), post slot initialization for the next p days as part of the go-live operation discussed in section 7.2.1, the steady state operation taking place that day (dt_{gl}) would be the processing of lead requests in real-time and assignments to be created in the system for visits scheduled from date ($dt_{gl} + 1$) onwards till ($dt_{gl} + p$) or call-backs (in case assignment couldn't be made). These would result in priority setting of the leads and the corresponding adjustment of queues (daily and possibly global) of the SE-s involved in real-time, which would be explained in sections 7.3.2 and 7.3.3.

On the morning of the next day ($dt_{gl} + 1$) the initialization of all slots for each SE_i for the p -th day from the run-date ($dt_{gl} + p + 1$) will take place with the corresponding daily queue for that day ($h_i^{dt_{gl} + p + 1}$) being set to the applicable global queue value (h_i^g). This process, described in section 7.3.1, will take place every day going forward.

Subsequently, the steady state operation taking place that day ($dt_{gl} + 1$) would be the processing of lead requests in real-time as already described in a preceding para.

That day, site visits take place for the first time based on the assignments created on the previous day (dt_{gl}), and accordingly the visit records as well as the visit skipped

records would be ingested by the system that day at a later point of time. However, the processing of those records would happen only the next day, $(dt_{gl} + 2)$, in a batch mode.

It would be followed by a batch process which would start executing from day $(dt_{gl} + 2)$ onwards on a daily basis. It processes the ingested visit records for the previous day, resulting in priority setting of the leads in batch mode (explained in section 7.3.2) and the subsequent adjustment of queues of the SE-s in batch mode (explained in section 7.3.3).

The same set of operations that took place on day $(dt_{gl} + 2)$ would get repeated every day onwards.

Let us now discuss these in greater detail.

7.3.1 Slot Initialization with queue setting (Steady State)

This runs as a batch process at the start of any given day later than dt_{gl} , say the system date dt_{sys} , where $dt_{sys} > dt_{gl}$.

It should be noted that on dt_{sys} the system would already have captured the availability of all SEs per day per slot from dt_{sys} till date $(dt_{sys} + p - 1)$ by virtue of the operations carried out till the previous day i.e. $(dt_{sys} - 1)$.

As part of the daily run executed as a batch process in the day beginning for dt_{sys} , the system would additionally capture the availability of each SE_i for all slots for date $(dt_{sys} + p)$ setting the corresponding daily queue value $h_i^{dt_{sys}+p}$ to the respective global queue value h_i^g applicable at that time.

The corresponding routine, InitSlotSetQSS, shown in Algorithm 3, takes as input system date dt_{sys} , p and m_{sl} , calculates date dt as $(dt_{sys} + p)$, and for each SE_i available for date dt fetch the corresponding global queue value h_i^g and invoke the InitSlotSetQDaily routine within a loop.

Algorithm 3 InitSlotSetQSS: Algorithm for SE's slot initialization with queue setting during steady state

Require: dt_{sys}, p, m_{sl}

- 1: $dt \leftarrow dt_{sys} + p$
 - 2: $NumAvlblSE \leftarrow FetchSECount(dt)$
 - 3: **for** ($i = NumAvlblSE$ to 1) **do**
 - 4: $SE_i \leftarrow FetchSEID(dt, i)$
 - 5: $h_i^g \leftarrow FetchGlobalQValSE(SE_i)$
 - 6: $InitSlotSetQDaily(SE_i, dt, h_i^g, m_{sl})$
 - 7: **end for**
-

7.3.2 Priority setting of leads

The priority setting of leads is accomplished via a mix of real-time and batch modes of operations, as explained below.

Priority setting of leads on a real-time basis

When a lead has expressed his intent to visit by providing a date and is being considered by the system for the first time (a fresh lead), a date of visit is assigned and its priority count is set to 1, its initial state. This is the priority initialization for a lead. The corresponding event would be *InitVR*.

A fresh lead will continue to retain its status until a visit happens. However, before undertaking the first visit, if he asks for a reschedule and a callback is generated, he will be treated like a lead who intends to visit for the first time but is yet to provide a visit date, and eventually if he does provide a visit date he will again be regarded as a fresh lead and start to be considered by the system for assignment.

Subsequently, the lead's priority setting would continue to happen based on real-time events as explained below.

If a lead makes a reschedule request and the system can accommodate it, its priority count will be decreased by 1. The corresponding event is *RR*.

If a lead makes an OS change request and the system can accommodate it, its priority count will be decreased by 1. The corresponding event is *OSCR*.

If a lead makes a reschedule as well as OS change request and the system can accommodate, its priority count will be decreased by 1. The corresponding event is

ROSCR.

The priority can never go below 1 while it is in the scope of this system.

For any of the above requests, if due to lack of capacity the system fails to accommodate and thus assigns a call-back at a later date to work out a visit time slot for the lead, the corresponding event (*RR* / *OSCR* / *ROSCR*) would be accompanied by the event *CB*, in that particular order. The effect of these two events needs to be considered together and the priority count of the lead would remain unchanged.

For this reason, we use an event array to capture the real-time lead events. The count of elements in the event array $event_l[]$ for each lead $Lead_l$ can be either 1 when the system can accommodate the request, or 2 when call-back is involved.

When a lead who has visited earlier (not a fresh lead) makes another visit request, the corresponding event is *VR* and the priority is incremented by 1.

The priority setting of leads in response to these real-time events is carried out in the routine for processing lead requests in real-time, *ProcLeadReqRealTime*, shown in Algorithm 4, which also does the queue adjustment of *SE*. Carried out as a post-processing routine of the assignment process, it takes $Lead_l$, $event_l[]$, SE_c , dt_c , sl_c , dt_{sys} , p as input parameters, where $event_l[]$ is the event array for $Lead_l$, and SE_c , dt_c , sl_c being the current *SE*, date and slot being assigned respectively in response to the event(s). The latter 3 would be null in case of callback.

Let us now consider the operational aspects of how the site visit requests of the leads are captured. Capturing the subsequent requests related to those site visit requests, like the reschedule request and/or the OS change request, will follow a similar pattern in operational terms.

Starting from the day it becomes operational (i.e. the go-live date dt_{gl}), the system would begin to capture site visit requests for the leads on a real-time basis. As a rule, the earliest day a visit request can be accommodated is the day after the date of request. The latest day a visit request can be accommodated is p days after the request date.

Let us illustrate the above mechanism by taking two examples below.

The visit dates being captured on the go-live date (dt_{gl}) can range from the next

day ($dt_{gl} + 1$) till date ($dt_{gl} + p$).

On any later day, say system date dt_{sys} ($dt_{sys} > dt_{gl}$), the system would keep on capturing throughout the day the intended site visit requirements for leads, which can range from ($dt_{sys} + 1$) till ($dt_{sys} + p$).

Priority setting of leads via batch mode

When a lead visits as planned, the event is *Visited* and its priority increases by 1. The SE facilitating the visit becomes the OS for the lead. If a lead fails to visit as planned, the event is *Skipped* and its priority decreases by 1. These visit records are ingested at a later point that day by the system, and get processed the next day beginning in batch mode.

This would be accomplished by the ProcVisitRecBatch routine shown in Algorithm 5 that takes dt_{sys} as input and processes the ingested visit records for the previous day and sets the priority for each lead associated with each of those records according to the event that took place.

7.3.3 Queue adjustment for SEs

We have previously explained in sections 7.2.1 and 7.3.1 the slot initialization process for SEs with queue initialization during go-live and steady state respectively, the corresponding routines being Algorithm 1 and Algorithm 3.

Those queue values will get adjusted when leads are assigned to the said SE in real-time, when other real-time events take place, and when the corresponding visit related events (*Visited* and *Skipped*) are processed in batch mode.

This queue adjustment process facilitating the load balancing of SEs would run both in real-time and batch mode, as explained below. It should be noted that daily queue value h_i^{dt} for each dt for SE_i should satisfy: $0 \leq h_i^{dt} \leq h_i^g$, h_i^g denoting the global queue value of SE_i .

When the global queue value for an SE changes, the corresponding incremental change needs to be applied to all the daily queue values for that SE for the next p

Algorithm 4 ProcLeadReqRealTime: Algorithm for processing lead requests in real-time, carrying out priority setting of lead and Q adjustment of SE

Require: $Lead_l, event_l[], SE_c, dt_c, sl_c, dt_{sys}, p$

- 1: $sts_l \leftarrow FetchStatusLead(Lead_l)$
- 2: **if** ($event_l[1] \neq InitVR$) **then**
- 3: $AssignRec_l^e \leftarrow FetchAssignRec(Lead_l)$
- 4: $SE_e \leftarrow AssignRec_l^e.SE$
- 5: $dt_e \leftarrow AssignRec_l^e.dt$
- 6: $pr_e \leftarrow AssignRec_l^e.pr$
- 7: $sl_e \leftarrow AssignRec_l^e.sl$
- 8: $h_l^g \leftarrow AssignRec_l^e.h^g$
- 9: **if** ($sts_l = FRESH$) **then**
- 10: $AdjQSEProc(SE_e, dt_e, 1, pr_e, ++)$
- 11: **else if** ($event_l[1] = VR$) **then**
- 12: $AdjQSEProc(SE_e, dt_{sys} + 1, p, pr_e, ++)$
- 13: $pr_c \leftarrow \min(pr_e + 1, h_l^g)$
- 14: **if** ($event_l[].count = 2$) **then**
- 15: $pr_c \leftarrow \min(pr_e + 1, h_l^g)$
- 16: **end if**
- 17: **else**
- 18: $AdjQSEProc(SE_e, dt_e, p, pr_e, ++)$
- 19: **end if**
- 20: **end if**
- 21: **if** ($event_l[].count = 1$) **then**
- 22: **if** ($sts_l = FRESH$) **then**
- 23: $pr_c \leftarrow 1$
- 24: $AdjQSEProc(SE_c, dt_c, 1, pr_c, --)$
- 25: **else**
- 26: **if** ($event_l[1] \neq VR$) **then**
- 27: $pr_c \leftarrow \max(pr_e - 1, 1)$
- 28: **end if**
- 29: **if** ($event_l[1] \in \{RR, VR\}$) **then**
- 30: $AdjQSEProc(SE_c, dt_{sys} + 1, p, pr_c, --)$
- 31: **else**
- 32: $AdjQSEProc(SE_c, dt_c, 1, pr_c, --)$
- 33: **end if**
- 34: **end if**
- 35: $SetAssignRec(Lead_l, pr_c, SE_c, dt_c, sl_c)$
- 36: **else**
- 37: **if** ($((sts_l \neq FRESH) \& (event_l[1] \in \{RR, VR\}))$) **then**
- 38: $SetAssignRec(Lead_l, pr_c, SE_c, null, null)$
- 39: **else**
- 40: $SetAssignRec(Lead_l, pr_c, null, null, null)$
- 41: **end if**
- 42: **end if**

days.

When the daily queue value for an SE changes for a day, the effect applies to that day only, lessening or increasing the ability of the SE to attend to fresh leads on that day.

Queue adjustment for SE in batch mode

We have explained previously in sections 7.2.1 and 7.3.1 how the slots for SEs are initialized during go-live and steady state respectively, with corresponding queue settings. Let us now describe how those queues get adjusted in batch mode.

The global queue value h_i^g of SE_i would get adjusted according to the number of leads for whom the SE happens to be the OS. Points to be considered are the relative impact of each such lead on the capacity to be provisioned and if any distinction is necessary between such leads.

The relative impact of each such lead would be considered by clustering all leads of the same priority in one group. Thus SE_i can have up to ψ different priority groups for each day dt . The reason for creating such different groups is to acknowledge the distinction between two leads with different priorities, as a higher priority indicates increasing engagement on the lead's part and hence a higher probability of a revisit. The leads within the same priority group would form a homogeneous group demonstrating the same characteristics as far as provisioning capacity to cater to them is concerned, which will change when a different priority group is considered.

When the number of such leads reaches a certain threshold it results in a queue shift by 1 for SE_i . This threshold value needs tuning to arrive at an optimum value.

The following equation shows the representation of f_{pr} , a function f mapping the number of leads n_{pr} belonging to the priority group pr having SE_i as the OS, with respect to the threshold value t_{pr} for that priority group pr .

$$\begin{aligned}
 f_{pr} &= 1, \text{ if } \frac{n_{pr}}{\log(\psi + 1 - pr) + 1} \geq t_{pr}, \\
 &= 0, \text{ otherwise.}
 \end{aligned}
 \tag{7.1}$$

Thus each specific priority group pr would have an associated t (threshold) value t_{pr} which would be a tunable parameter. Within each such priority group with value pr , when the corresponding lead count n_{pr} accumulates up to the value t_{pr} , the SE's queue value would be impacted (incremented or decremented as the case may be) by 1 and the corresponding lead count n_{pr} would be reset to 0.

When the visit records are processed and the queue adjustment happens for all the leads for whom the SE is the OS, there could be the possibility of shifts happening within the priority groups.

Let us consider the following scenario for illustration. A lead $Lead_l$ has SE_i as his OS. If the lead's priority shifts from pr_e (earlier priority) to pr_c (current priority) while SE_i continues to remain the OS, the effective capacity for SE_i should increase for priority group pr_e since some capacity for that priority group has been released now with $Lead_l$ leaving that group, while the effective capacity for priority group pr_c should decrease as some capacity for that priority group would need to be allocated now with $Lead_l$ entering that group. This is indicated by decrementing the value of $pr_c h_i^{dt}$ for priority group pr_c by 1 while incrementing the value of $pr_e h_i^{dt}$ for priority group pr_e by 1.

If in the process the $pr_e h_i^{dt}$ value for the priority group pr_e of SE_i on date dt reaches t_{pr_e} (i.e. the t threshold value for priority group pr_e), the corresponding h_i^{dt} value would get incremented and $pr_e h_i^{dt}$ is reset to 0.

An increment and a decrement within the same priority group pr_c of SE_i on date dt will cancel each other out and $pr_c h_i^{dt}$ will remain unchanged.

When the global queue value h_i^g of SE_i changes, the corresponding incremental effect would also apply on h_i^{dt} i.e. the daily queue values for SE_i for all dt in the date range starting from the system date till p days later.

When the SE's queue value h_i^{dt} for date dt becomes 0, it is taken off the queue (No-Queue i.e. NQ), implying it has no further allocation capacity.

All the above would be accomplished by the routine for processing visit records batch process, ProcVisitRecBatch, as shown in Algorithm 5 that takes dt_{sys} as input and processes the ingested visit records for the previous day and invokes the SE queue

adjustment process, AdjQSEProc.

The ProcVisitRecBatch routine wraps it up by invoking the routine SetAssignRec setting the recent most assignment for *Lead* with his current priority pr_c . For skipped event for a fresh lead this will pass nulls for SE, date and slot, whereas for all other cases, SE_c (the OS) would be passed along with null-s for date and slot.

Algorithm 5 ProcVisitRecBatch : Algorithm for processing visit records batch process, carrying out lead priority setting and SE queue adjustment

Require: dt_{sys}

```

1:  $dt \leftarrow dt_{sys} - 1$ 
2:  $NumRec \leftarrow FetchVisitCount(dt)$ 
3: for ( $i = NumRec$  to 1) do
4:    $VisitRec \leftarrow FetchVisitRecords(dt, i)$ 
5:    $Lead \leftarrow VisitRec.Lead$ 
6:    $SE \leftarrow VisitRec.SE$ 
7:    $pr_e \leftarrow FetchPriorityLead(Lead)$ 
8:    $sts \leftarrow FetchStatusLead(Lead)$ 
9:   if ( $VisitRec.event = Visited$ ) then
10:     $pr_c \leftarrow min(pr_e + 1, \psi)$ 
11:    if ( $sts \neq FRESH$ ) then
12:       $AdjQSEProc(SE, dt + 2, p, pr_e, ++)$ 
13:    end if
14:     $AdjQSEProc(SE, dt + 2, p, pr_c, --)$ 
15:     $SetAssignRec(Lead, pr_c, SE, null, null)$ 
16:  else
17:     $pr_c \leftarrow max(pr_e - 1, 1)$ 
18:    if ( $sts \neq FRESH$ ) then
19:       $AdjQSEProc(SE, dt + 2, p, pr_c, --)$ 
20:       $AdjQSEProc(SE, dt + 2, p, pr_e, ++)$ 
21:       $SetAssignRec(Lead, pr_c, SE, null, null)$ 
22:    else
23:       $SetAssignRec(Lead, pr_c, null, null, null)$ 
24:    end if
25:  end if
26: end for

```

Queue adjustment for SE on a real-time basis

When a lead is freshly ingested into the system requesting a visit date, it has a priority of 1. In order to allocate it to an available SE, the assignment process is run which

will match it against the SEs available. As this routine is invoked as part of the post-processing of assignment process, the SE's corresponding daily queue value h_i^{dt} is reduced by 1 on a real-time basis for dt when the requested visit is to take place, the corresponding event being *InitVR*.

When there is a reschedule request ("RR" event) for a lead $Lead_l$, there would be different outcomes depending on whether $Lead_l$ is a fresh lead or not.

If $Lead_l$ is a fresh lead, then it has its earlier priority value pr_e set to 1. For the earlier SE, SE_e , the queue value for the earlier date dt_e , $h_i^{dt_e}$, would get increased by 1.

If the RR request can't be accommodated by the system, indicated by the presence of *CB*, then the priority remains unchanged. There is no current SE allocation.

If the RR request can be accommodated by the system, then ideally the priority should be decremented by 1. But since it would become 0 in that case, which would not be a fair reflection on the load bearing capacity for the SE to whom it would be assigned currently i.e. SE_c , the priority would be kept unchanged at pr_e (which is 1) only. Also for SE_c , the queue value for the current date dt_c , $h_i^{dt_c}$, would get decreased by 1.

Let us now consider the case of a lead that is not a fresh lead. If $Lead_l$ is not a fresh lead, it implies that it already has an OS, the earlier SE SE_e .

In case the request can be accommodated by the system, the priority of the lead would change from pr_e to pr_c (it would be decremented by 1). The earlier date dt_e and slot sl_e would now be changed to the current date dt_c and slot sl_c . The queue adjustment would happen for the SE SE_c who would continue to remain the current SE (OS) for the lead in the following manner. Its effective capacity would decrease for the priority group pr_c where $Lead_l$ would belong now, and the effective capacity would increase for the priority group pr_e where $Lead_l$ belonged earlier.

In case the request can't be accommodated by the system, indicated by the presence of *CB*, the priority of the lead would remain unchanged at pr_e , and there would be no queue adjustment since the SE SE_e would continue to remain the OS for the lead.

When there is an "OSCR" event for a lead $Lead_l$, implying a request for the OS change, the queue adjustment would happen for the SE SE_e who was the earlier OS for the lead, whose effective capacity would increase for the priority group pr_e where $Lead_l$ belonged earlier. In case the request can be accommodated by the system, the priority of the lead would change from pr_e to pr_c (it would be decremented by 1). The queue adjustment would now happen for the SE SE_c who has become the current OS for the lead, whose effective capacity would decrease for the priority group pr_c where $Lead_l$ would belong now. This is an instance of shifts happening across priority groups for multiple SE-s.

In case the system is unable to accommodate the "OSCR" request, indicated by the presence of CB , $Lead_l$'s priority remains unchanged (pr_e). Since no SE could be assigned here, the net effect here would be the queue adjustment (increment) for SE_e .

The implications would be quite similar for the "ROSCR" event, where the additional factor to consider would be the change in date request (reschedule request) in addition to the OS change request.

When a lead who has visited earlier (not a fresh lead) makes another visit request, the corresponding event would be VR . If the system can accommodate then the priority of the lead would change from pr_e to pr_c (incremented by 1), but cannot exceed ψ . The earlier date dt_e and slot sl_e (which were null) would now be changed to the current date dt_c and slot sl_c .

However, if the system is unable to accommodate that (event CB in addition), the priority of the lead would change from pr_e to pr_c (incremented by 2), but cannot exceed ψ . The earlier date dt_e and slot sl_e (which were null) would now be changed to the current date dt_c and slot sl_c which would remain null.

For both cases with VR , the queue adjustment would happen for the SE SE_c who would continue to remain the current SE (OS) for the lead. Its effective capacity would decrease for the priority group pr_c where $Lead_l$ would belong now, while that would increase for the priority group pr_e where $Lead_l$ belonged to earlier.

These effects are provided by the AdjQSEProc routine which is invoked from

within the ProcLeadReqRealTime routine as shown in Algorithm 4.

The ProcLeadReqRealTime routine is carried out as a post-processing requirement of the assignment process and takes $Lead_l$, $event_l[]$, SE_c , dt_c , sl_c , dt_{sys} , p as input parameters, where $event_l[]$ is the array of events containing 1 or 2 elements for $Lead_l$ as already described, with SE_c , dt_c and sl_c being the current SE, date and slot being assigned in response to the event(s). The latter 3 would be null in case of call-back.

The ProcLeadReqRealTime routine wraps it up by invoking the routine SetAssignRec, setting the recent most assignment for $Lead_l$ with his current priority pr_c and the assigned SE SE_c along with the date dt_c and slot sl_c of the assignment when the system is able to accommodate it. In case of a call-back, when the lead is a fresh lead then this will pass nulls for SE, date and slot, whereas for the non-fresh leads, SE_c (the OS) would be passed along with null-s for date and slot.

Our next course of action would be to decide on a suitable approach to model the above functionalities as discussed.

7.4 Modelling considerations: Desired properties for the model to exhibit

Modern system development considers modelling and analysis as essential to manage the complexity and mitigate risks upfront in designing and operating such a system. Also, given the criticality of ascertaining the correctness of the models, we need to specify the desired properties that the model should demonstrate.

Before carrying out the modelling of the flows as previously described, let us specify the properties that our model should be able to exhibit as follows.

1. *Reach a specific state:* This signifies if the system exhibits a specific behavior, thereby affirming the modelled system's adherence to agreed properties.
2. *Prevent overflow:* Overflow should be prevented by avoiding nodes with overflow possibilities in the model as they imply bottlenecks.

3. *Prevent deadlock*: Deadlock prevention is clearly important.
4. *Fairness*: This relates to the ability to perform actions according to the order by which they are announced.

We would now carry out the modelling of certain key functionalities namely the processes dealing with slot initialization with queue, the priority setting of leads and the queue adjustment of SEs in the next section.

7.5 Petri net modelling of functionalities in scope

In sections 7.2 and 7.3 we have discussed the functionalities in scope of this chapter, and we intend to model the focus areas of slot initialization with queue, priority setting of leads and queue adjustment of SEs, in this section. Petri net would be our model of choice given its usefulness in modelling, formal analysis and design of systems like the one in scope.

7.5.1 Petri net modelling of slot initialization with queue setting

The Petri net modelling of the flow for slot initialization with queue setting, that aspect of the Steady State Operation which initiates allocation of fresh slots for all SEs for date dt along with highest priority queue formation during the daily run, is shown in fig. 7-1 for any SE SE_i .

Let us define the legends we'd be using to explain the modelling of the slot initialization process using fig. 7-1.

Legends:

IS_{dt} : Initiate Slot allocation for accepted date dt ;

SP^i : Slot Preparation for i^{th} SE;

C_{sl} : Collection of slots;

m_{sl} : Instances of slots which can be assigned to an SE for a given day (daily capacity);

SE_{sd}^i : Collection of assignable slots sd prepared for i^{th} SE (for date dt);

KT_{sl} : Keep Track of available slots;

IR_{sl} : Initiate Return of available slots;

RA_{sl} : Return Available slots

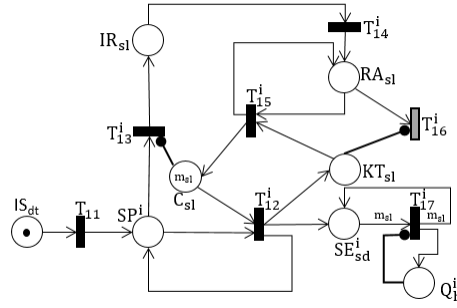


Figure 7-1: Petri net modelling of Slot Initialization with Queue - for one SE.

When allocation for slots for date dt needs to be initiated, a token would be there in IS_{dt} . A token in IS_{dt} results in the transition T_{11} getting fired. The token from the input place IS_{dt} is taken, leaving it with no token. A token is deposited in the place SP^i . This sets the stage for the slots to be prepared for the i^{th} SE.

The place C_{sl} provides a collection of m_{sl} tokens, where each token represents one specific instance out of the m_{sl} slots that can be allocated for the given SE SE^i on date dt . When there's a token in SP^i as well, these input places of T_{12}^i are synchronized for the firing of transition T_{12}^i .

A token is added to SE_{sd}^i and KT_{sl} . The token which was taken from the place SP^i is inserted back, enabling the firing of transition T_{12}^i once again.

This continues iteratively till there is no token left in C_{sl} . Both the places SE_{sd}^i and KT_{sl} are populated with m_{sl} tokens.

The place SE_{sd}^i eventually shows a transition to Q_h^i , implying the initiation of slots for SE^i on date dt using the slot initialization process happens with the highest queue allocation.

When there is no token left in C_{sl} (inhibitor), it will initiate the firing of transition T_{13}^i as there is a token present in SP^i .

The place IR_{sl} is populated with a token. The place SP^i has no token left.

The presence of a token in the place IR_{sl} initiates the transition T_{14}^i .

The place RA_{sl} is populated with a token. The place IR_{sl} has no token left.

There is a token in RA_{sl} and there are m_{sl} tokens in KT_{sl} . With presence of tokens, these input places of T_{15}^i are synchronized for the firing of transition T_{15}^i .

The token from RA_{sl} is taken leaving no token there. Out of m_{sl} tokens available in the place KT_{sl} , one is taken leaving $(m_{sl} - 1)$ tokens remaining. A token is deposited in the place C_{sl} . The place RA_{sl} will have the token deposited back while the place KT_{sl} has $(m_{sl} - 1)$ tokens, enabling the transition T_{15}^i to be fired once again.

This circular firing sequence will continue for m_{sl} times till there is no token left in KT_{sl} i.e. all m_{sl} tokens are removed from it. The place C_{sl} will now have a collection of m_{sl} tokens deposited in it.

The place RA_{sl} will have one token.

The presence of a token in RA_{sl} and the absence of any token in KT_{sl} results in the transition T_{16}^i getting fired.

The token from the input place RA_{sl} is taken, leaving it with no token.

The presence of m_{sl} tokens in the place SE_{sd}^i and absence of a token in the place Q_h^i initiates the transition T_{17}^i .

The place Q_h^i is populated with a token. The place SE_{sd}^i is re-populated with m_{sl} tokens.

7.5.2 Petri net modelling of priority setting of leads

The Petri net modelling of the priority setting for a lead is shown in fig. 7-2.

Any reschedule request (RR)/OS change request ($OSCR$)/reschedule and OS change request ($ROSCR$) would result in a token to be deposited on a real-time basis to the place $--$. A not fresh lead's repeat visit request (VR), as well as the system's inability to accommodate any of the aforementioned requests resulting in a

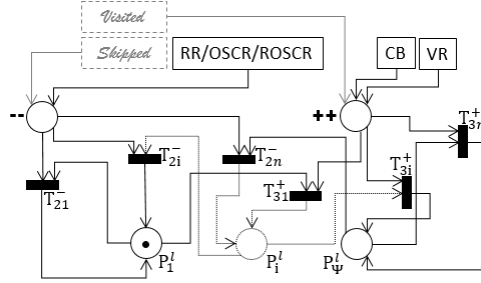


Figure 7-2: Petri net diagram of priority setting for a lead.

call-back (CB) at a later date, would result in a token being deposited on a real-time basis to the place ++.

The information of visits happening or being skipped in the previous day will be ingested by the system at a later point, which will be processed in batch mode at the beginning of the day after the slot initialization process execution, each of which results in a token being inserted in the place ++ or the place -- respectively.

Let us define the legends we'd be using to explain the modelling of priority setting of leads using fig. 7-2.

Legends:

++ / -- : Increment / decrement priority of the lead;

P_1^l / P_i^l / P_ψ^l : Priority value 1 / i / ψ assumed by lead l ;

RR/OSCR/ROSCR : reschedule request /OS change request /reschedule and OS change request;

VR : visit request;

CB : Callback arranged for a later date for the lead;

Visited : previous day lead visited information;

Skipped : previous day lead skipped visit information;

At the outset, we need to keep in mind that for the lead l being considered here, only one of the places from P_1^l to P_ψ^l can be occupied at a given time since that lead would have a unique priority at any point in time.

If we consider the lead as a fresh lead who has indicated intent to visit a site for the first time by providing a date, the priority of the lead would be 1. This is the default case and hence a token is shown as present in the place P_1^l in fig. 7-2.

Priority setting of leads - increment

Let us first address the priority increment part.

Here a token would be present in the place $++$, which can be due to the batch feed of the *Visited* event of lead l on the previous day ($dt - 1$), or a callback (*CB*) (for a later date) arranged for the lead on that day (date dt), or a Visit Request (*VR*) event for a not fresh lead on that day (date dt).

Assuming the priority of the lead is 1, which is the default case when a fresh lead indicating intent to visit the site by providing a date is considered for the first time by the system, a token would be present in the place P_1^l .

With the presence of a token each, these two input places would be synchronized to fire the transition T_{31}^+ to deposit a token into the place P_2^l . The places $++$ and P_1^l both would now become token-less.

This implies that the priority of the lead l has been incremented from 1 to 2.

Let us now consider the priority increment for a lead with priority 2. Going by our assumption that the value of ψ is capped to 3 for this particular implementation, we can treat the generalized variable i to be 2 for the specific system being considered here. This implies that a token is already present in the place P_i^l , which is equivalent to P_2^l .

When a token is inserted in the place $++$, both these input places $++$ and P_i^l would be synchronized to fire the transition T_{32}^+ to deposit a token into the place P_ψ^l , since $\psi = 3$ for the system we are considering. The places $++$ and P_i^l both would now become token-less.

This implies that the priority of the lead l has been incremented from 1 to 2.

There is a special case with priority increment when the priority of the lead is ψ , whereby a token is present in the place P_ψ^l . With the presence of a token in the place $++$, these two input places would be synchronized to fire the transition $T_{3\psi}^+$, with the

places $++$ and P_ψ^l both becoming token-less. The firing would result in depositing a token into the place P_ψ^l . So effectively the place $++$ becomes token-less, while the place P_ψ^l would retain its token.

This implies that the priority of the lead l can not increase beyond ψ , which is the maximum value that a lead's priority can assume. As already stated, our assumption is that the value of ψ is capped at 3 here.

Priority setting of leads - decrement

Let us now address the priority decrement part.

Here a token would be present in the place $--$, which can be due to the batch processing of the *Skipped* event of lead l on the previous day ($dt - 1$), or RR / OSCR / ROSCR request coming from the lead on date dt in real-time.

As already stated, the value of ψ is capped to 3 for this particular implementation, and we can thus treat the generalized variable i to be 2 for the specific system being considered here.

Assuming the priority of the lead is 2, a token would be present in the place P_i^l .

With presence of tokens, these two input places $--$ and P_i^l would be synchronized to fire the transition T_{2i}^- to deposit a token into the place P_1^l . The places $--$ and P_i^l both would now become token-less. This implies that the priority of the lead l has been decremented from 2 to 1.

The decrementing of priority of another lead from ψ (effectively 3) to i (effectively 2) can be represented similarly.

There is a special case with priority decrement when the priority of the lead is 1, whereby a token is present in the place P_1^l . With presence of a token in the place $--$, these two input places would be synchronized to fire the transition T_{21}^- , with the places $--$ and P_1^l both becoming token-less. The firing would result in depositing a token into the place P_1^l . So effectively the place $--$ becomes token-less, while the place P_1^l would retain its token.

This implies that the priority of the lead l can not decrease below 1, which is the minimum value that a lead's priority can assume.

7.5.3 Petri net modelling of queue adjustment for SE

The Petri net modelling of queue adjustment for SEs is shown in fig. 7-3.

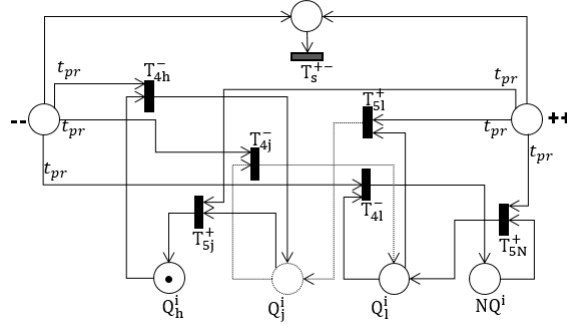


Figure 7-3: Petri net diagram for queue adjustment for SEs.

Let us define the legends we'd be using to explain the modelling of queue adjustment for SEs using fig. 7-3.

Legends:

$++ / --$: Increment / decrement queue value for SE;

$Q_1^i / Q_h^i / Q_j^i$: Queue value at 1 / h / j for SE SE^i ;

NQ^i : No queue for SE SE^i ;

At the outset, we need to keep in mind that for the SE SE^i being considered here, only one of the places from Q_1^i to Q_h^i can have a token at a given time since that SE would have a unique queue value at any point of time.

If we consider the SE as a new hire who has been inducted into the system for the first time, or we are considering the slot initialization of the SE for a given day, the queue value of the SE would be h for the day being considered. Accordingly a token is shown as present in the place Q_h^i in fig. 7-3.

Subsequently, there would be leads assigned to the SE, leading to adjustment (decrementing) of his queue value, since his capacity would now be reduced. The corresponding events of *Visited* (triggered via batch ingestion) will result in a further adjustment of his queue value, as the SE will now become the OS for these leads who themselves would have different priority values.

There would also be instances of the SE's queue value getting incremented, triggered via events like *Skipped* leading to a reduction of priority for concerned leads for which the SE continues to remain the OS.

For ease of understanding, we'll consider leads for a given priority pr here, with the corresponding t -value being considered as t_{pr} .

Queue adjustment for SEs - decrement

Let us first address the decrement part of queue adjustment for SEs.

Assuming the queue value of the lead is h , the default case as explained, a token would be present in the place Q_h^i .

A token would be deposited in the place $--$ for each lead with the specific priority being considered, based on applicable events like a *Visited* event of a lead happening on the previous day ($dt - 1$) which is obtained via the batch feed on the morning of day dt , or a *VR* coming from a fresh lead on date dt based on assignment process happening in real-time.

When the token count for the place $--$ reaches t_{pr} (implying the count of applicable leads with the specific priority reaches t_{pr}), these two input places, Q_h^i and $--$, would be synchronized to fire the transition T_{4h}^- to deposit a token into the place $Q_j^i, j = h - 1$. The places $--$ and Q_h^i both would now become token-less.

This implies that the queue value of SE^i has been decremented from h to $j, j = h - 1$.

We can continue representing in this way, till we consider the special case where a token is present in the place Q_1^i and t_{pr} tokens in the place $--$. These two input places would be synchronized to fire the transition T_{41}^- , which would result in depositing a token into the place NQ^i , with the places $--$ and Q_1^i both becoming token-less.

This implies that the SE SE^i has now been taken off the queue (No Queue - NQ).

Queue adjustment for SEs - increment

The increment part of queue adjustment for SEs would be similar to section 7.5.3 but in a reverse way.

Let us consider the SE SE^i who is off the queue. This would imply presence of a token in NQ^i . The queue increment will happen when there are events leading to t_{pr} tokens being deposited in the place $++$. These two input places would be synchronized to fire the transition T_{5N}^+ , which would result in depositing a token into the place Q_1^i , with the places $++$ and NQ^i both becoming token-less. Thus SE^i will now move to a state where it will be able to accept requests, with its queue value becoming 1.

There's a special case involving tokens from both the places $--$ and $++$. A decrement or increment operation only gets enabled when the token count for the respective place ($--$ or $++$) reaches t_{pr} , and nothing happens as long as the respective count remains less than t_{pr} . Since the decrement and increment operations by their nature cancel each other out, it is represented by the sink transition T_s^{+-} which can be triggered by presence of a token in both the places $--$ and $++$. The firing would result in the two places $++$ and $--$ both becoming token-less.

7.6 Verification of our Petri net models

We would be adopting the behavioural approach for our Petri net analysis here to determine if the set of properties stated in section 7.4 are exhibited with respect to the following behavioural properties:

1. **Reachability:** This property directly corresponds to *Reach a specific state* in section 7.4.

To determine if the modelled system can reach a specific state M_i , in essence exhibiting a desired functional behavior, a sequence of transitions needs to be produced that upon firing will transform an initial marking M_0 to M_i . The sequence of firings represents the required functional behavior.

We would be creating the reachability tree for the Petri net models which would allow us to find the sequence of firings of transitions which would result in transforming the marking M_0 to the marking M_i .

2. **Boundedness:** This Petri net property aids in identifying the overflow possibilities in the modelled system [34], and it directly corresponds to *Prevent overflow* in section 7.4. A bounded place will guarantee that the capacity associated will not get exceeded by related operations [65], which needs to be ensured in the models that we will create by having them exhibit the Boundedness property.

A Petri net is called k -bounded with respect to an initial marking M_0 , if each place in the net gets at most k tokens for all markings belonging to the reachability set $R(M_0)$, where k is a finite positive integer.

Also unbounded places are potential bottlenecks. We need to verify that there are no such instances in the models created.

3. **Liveness:** Liveness, a Petri net property tied to the concept of deadlocks and deadlock-freeness [65], directly corresponds to *Prevent deadlock* in section 7.4.

A transition is dead in a marking if no sequence of transition firings exist to enable it [65], and we have to make sure that there is no redundant transition or dead transition in our model since it possibly indicates a modelling error or an inconsistency in the system being modelled.

A transition is potentially fireable if there exists some firing sequence that enables it, and we have to ensure that every transition in our model is potentially fireable.

For each Petri net model, we would first define the degrees of liveness of individual transitions followed by the degrees of liveness of the entire Petri net.

4. **Fairness:** Fairness is a Petri net property that directly corresponds to *Fairness* in section 7.4.

Related problems are studied in the Petri net model where actions correspond to the firings of transitions. An action is announced if a transition becomes fireable, and it is performed if this transition fires. Fairness in Petri nets means firing of transitions in the order of their enabling. Therefore, fairness is a property of firing sequences.

We'd consider two basic fairness concepts:

Bounded Fairness (B- Fairness):

Two transitions t_1 and t_2 are said to be in a Bounded-fair (B-Fair) relation if the maximum number of times that either one can fire while the other is not firing is bounded.

A Petri net (N, m_0) is said to be a B-Fair net if every pair of transitions in it is in a B-Fair relation.

Unconditional (Global) Fairness:

A firing sequence σ is said to be unconditionally (Globally) Fair if it is finite or every transition in the net appears infinitely often in σ .

A Petri net (N, m_0) is said to be an unconditionally (Globally) Fair net if every firing sequence σ from $m \in R(m_0)$ is unconditionally fair.

7.6.1 Verification of Petri net modelling of slot initialization with queue setting

Here are the properties exhibited by adopting the behavioural approach for analyzing our Petri net models.

1. **Reachability:** We have constructed the Reachability Tree for the Petri net model for the slot initialization with queue setting process as shown in Table 7.1.
2. **Boundedness:** Our Petri net model is m_{sl} -bounded.
3. **Liveness:** Transitions T_{11} , T_{13}^i , T_{14}^i , T_{16}^i and T_{17}^i are $L1$ -Live, since each of them will fire only once. Transition T_{12}^i and T_{15}^i will each fire for m_{sl} times, which however is a finite number. Hence both are $L1$ -Live.

Our model is strictly $L1$ -Live since it is $L1$ -Live but not $L2$ -Live (as all transitions are $L1$ -Live but none are $L2$ -Live).

4. **Fairness:**

Bounded Fairness (B- Fairness):

In our model, any two transitions are in a Bounded-fair (B-Fair) relation since the maximum number of times that either one can fire while the other is not firing is always bounded. Hence our model would be a B-Fair net since every pair of transitions in each of them is in a B-Fair relation.

Unconditional (Global) Fairness:

Each firing sequence σ here is unconditionally (Globally) Fair since it is finite. Our Petri net model is an unconditionally (Globally) Fair net since every firing sequence σ therein is unconditionally fair.

Table 7.1: Reachability Tree for fig. 7-1.

T	M	IS_{dt}	C_{sl}	SP^i	SE_{sd}^i	KT_{sl}	IR_{sl}	RA_{sl}	Q_h^i
	M_0	1	m_{sl}	0	0	0	0	0	0
T_{11}	M_1	0	m_{sl}	1	0	0	0	0	0
$T_{12}^i \#$	M_2	0	0	1	m_{sl}	m_{sl}	0	0	0
T_{13}^i	M_3	0	0	0	m_{sl}	m_{sl}	1	0	0
T_{14}^i	M_4	0	0	0	m_{sl}	m_{sl}	0	1	0
$T_{15}^i \#$	M_5	0	m_{sl}	0	m_{sl}	0	0	1	0
T_{16}^i	M_6	0	m_{sl}	0	m_{sl}	0	0	0	0
T_{17}^i	M_7	0	m_{sl}	0	m_{sl}	0	0	0	1

indicates repetition of m_{sl} times

7.6.2 Verification of Petri net modelling of priority setting of leads

Here are the properties exhibited by adopting the behavioural approach for analyzing our Petri net models.

1. **Reachability:** The Reachability Trees for the Petri net model for priority setting of leads for the decrement and increment modes are shown in Tables 7.2 and 7.3 respectively.

Assumption: Given ψ is capped to 3 here, we can treat the generalized variable i to be 2 in this case.

2. **Boundedness:** The Petri net model is 1-bounded for both decrement and increment modes.
3. **Liveness:** In our model each transition can fire only once, if at all, thus being $L1$ -Live. Hence the Petri net model is strictly $L1$ -Live since it is $L1$ -Live but not $L2$ -Live (as all the transitions are $L1$ -Live but none are $L2$ -Live).
4. **Fairness:**

Bounded Fairness (B- Fairness):

Our model is a B-Fair net since each pair of transitions in it is in a B-Fair relation.

Unconditional (Global) Fairness:

Our model is an unconditionally (Globally) Fair net since every firing sequence σ therein is unconditionally fair.

Table 7.2: Reachability Tree for fig. 7-2: decrement operation.

T	M	--	PL_1^l	PL_i^l	PL_ψ^l
T_{2i}^-	M_1	1	0	1	0
	M_2	0	1	0	0
T_{21}^-	M_3	1	1	0	0
	M_4	0	1	0	0
$T_{2\psi}^-$	M_5	1	0	0	1
	M_6	0	0	1	0

7.6.3 Verification of Petri net modelling of queue adjustment of SEs

Here are the properties exhibited by adopting the behavioural approach for analyzing our Petri net models.

Table 7.3: Reachability Tree for fig. 7-2: increment operation.

T	M	++	PL_1^l	PL_i^l	PL_ψ^l
T_{31}^+	M_1	1	1	0	0
	M_2	0	0	1	0
T_{3i}^+	M_3	1	0	1	0
	M_4	0	0	0	1
$T_{3\psi}^+$	M_5	1	0	0	1
	M_6	0	0	0	1

1. **Reachability:** The Reachability Tree for our model for decrement and increment modes can be constructed like the previous section.
2. **Boundedness:** The Petri net model is t_{pr} -bounded.
3. **Liveness:** In our model each transition can fire only once, if at all, thus being $L1$ -Live. Hence the model is strictly $L1$ -Live since they are $L1$ -Live but not $L2$ -Live (as all the transitions are $L1$ -Live but none are $L2$ -Live).
4. **Fairness:**

Bounded Fairness (B- Fairness):

Our model is a B-Fair net since every pair of transitions therein is in a B-Fair relation.

Unconditional (Global) Fairness:

Our model is an unconditionally (Globally) Fair net since every firing sequence σ therein is unconditionally fair.

7.6.4 Summary of the verification exercise conducted on the Petri net models

The behavioural analysis for the Petri net models earlier in this section has helped us to verify that the properties stated in section 7.4 are indeed exhibited by our Petri net models.

The following observations summarize the verification of our Petri net models with respect to the said set of properties by conducting the behavioural analysis:

1. **Reachability:** We have created the reachability trees for the Petri net models which allow us to find the sequence of firings of transitions that would result in transforming a marking M_0 to M_i , M_i representing the specific state, for the model.

As we have a truly finite tree for each of the models considered, it also follows that the terms reachability tree and coverability tree are synonymous. Also, each of the Petri nets is a finite capacity net.

2. **Boundedness:** In a real-life environment, the boundedness or safeness of a Petri net indicates the absence of overflow in the modelled system. This has been ensured in the models that we have created.

Also unbounded places are potential bottlenecks. There are no such instances in the models created and hence we don't have a potential bottleneck.

3. **Liveness:** We have seen that there is no redundant transition or dead transition in our models, thus ruling out related errors in the model or inconsistency in the system being modelled [65].

Every transition in our models is potentially fireable.

The Petri net models we have constructed are all strictly L1-Live, and hence would be deadlock-free.

4. **Fairness:** We have shown that each of our Petri net models qualifies to be a B-Fair net as well as an unconditionally (Globally) Fair net.

7.7 Summary of chapter

The area of focus in this chapter has been on those processes which act as prerequisites for the core CRM pre-sales assignment process of leads based on their requests to

appropriate SEs to function, and which are also necessary for the post-processing activities of the assignment process. The processes in scope are the slot initialization process with queue setting, priority setting of leads, and adjustment of queues for SEs. There are two aspects to consider, go-live and steady state operations. Accordingly, we have described the relevant functionalities and modelled the key areas using Petri net. We have verified the correctness of the modelling by carrying out the behavioural analysis of models to show their adherence to the desired set of properties.

We believe this will help inform our design understanding and also provide greater clarity during the implementation and run-time of the functionalities for the business domain described in this chapter. It can also be extended to similar areas in other industry domains by carrying out Petri net based modelling exercises similar to ours.

In terms of future directions, we would like to incorporate at a later point in time the functionality whereby the availability of SEs per day per slot till a future date would be refreshed at the start of every day, accounting for any leave/leave cancellation/termination as well as new appointment of SEs based on the information available as of that day.

Conclusion

In this concluding chapter, we take a look back at the previous chapters and summarize the various aspects explored in the thesis as we conclude. The primary focus of this research has been on leveraging frameworks like Essence and Petri net to automate CRM pre-sales in real estate, exploiting various software engineering methods, practices and tools. Lastly, we talk about the potential directions for future research.

Usage of software is ubiquitous across the globe, and while stories of success abound, they have been dotted with so many instances of failure too. Practitioners in Software Engineering are guided more by their own experience which is usually limited and hence the proceedings happen mostly by trial-and-error. The world of Software Engineering has seen continuous churns, with so many different approaches making their mark over time. An inevitable fallout of this has been the inclination to discard the methods hitherto in use entirely in an attempt to embrace the new, and then keep on repeating this cycle. This is where Essence, a “language and kernel” of Software Engineering resulting from the efforts of the SEMAT initiative to deal with the problem of immature practice in software engineering is so appealing. It has evolved as a method- and practice-independent approach and we have leveraged that to create reusable assets by building up methods consisting of practices that we have essentialized in the process.

Chapter 1 has served as the thesis introduction.

Chapter 2 has provided a foundational understanding of the relevant areas in software engineering frameworks and methods like Essence, Scrum and Petri net, and

the industry domains like CRM and real estate, thus laying the groundwork for the readers to have adequate context for the later chapters of the thesis.

In chapter 3 we have developed a framework to address the microservices lifecycle using Domain Driven Design (DDD), introduced alphas to extend the SEMAT Kernel and added the Work Products and Activities associated with those alphas. This is industry agnostic and can be used anywhere.

We have subsequently taken up the industry area of our interest, real estate CRM (pre-sales), from chapter 4 onwards. We have created an agile approach for carrying out automation of functionalities in the said industry segment using Scrum and Essence in chapter 4.

Next in chapter 5 we have created a full-fledged Software Engineering method with the adoption of the User Story and the microservices practice within the Essence framework leveraging Scrum for our industry scenario. This provides a comprehensive view of the software endeavor by creating a method adopting a set of practices using Essence as the common ground. While this is carried out for a specific industry, it can be extended to other industries as well, utilizing Essence as the unifying framework.

We have continued with the industry context of the automation of CRM pre-sales for real estate in chapter 6 and focused on the assignment process, a workflow that deals with the dynamics of scheduling/rescheduling of site visit requests from leads in the backdrop of the availability/unavailability of Sales Executives on the said day of the site visit along with applicable constraints. We have constructed generalized Petri net models of the assignment process and verified that they conform to the desired set of criteria to be demonstrated by such systems.

Chapter 7 wraps this up by focusing on the operational aspects of the automated system with emphasis on the processes necessary for the pre-processing and post-processing of the assignment process. We would consider the initiation of the automated system on its go-live date as well as its execution in steady state as we describe the underlying functionalities, model those using Petri net, and carry out their verification using behavioural analysis.

It is to be noted that the value of the Petri net based modelling exercise undertaken

in terms of clarifying and improving our design understanding that is key to successful implementation applies not only to the business scenarios in scope here (i.e. real estate CRM), but it can also extend to other industry domains where similar Petri net based modelling and analysis can be conducted.

It might be a worthwhile exercise to construct additional variations of our Essence based exercise to arrive at a new method. We can select practices other than what we have taken for the relevant areas (like use case for requirements), essentialize them, and compose them under the unifying framework of Essence to create a new method. We may choose a different industry segment to carry out this exercise.

As far as the Petri net based modelling that we have carried out, we can build upon this even further by considering each individual scenario of the assignment and creating a specific Petri net model by leveraging the generalized models already built. This can prove to be quite valuable in lending greater clarity during implementation. We might also consider looking into the functionalities of other industry scenarios involving workflows for example and extend our Petri net based exercise in those areas.

Bibliography

- [1] “Essence – kernel and language for software engineering methods version 1.2,” OMG specifications, Object Management Group, October 2018. [Online]. Available: <https://www.omg.org/spec/Essence/1.2/PDF>
- [2] I. Jacobson, H. B. Lawson, P.-W. Ng, P. E. MacMahon, and M. Goedicke, *The Essentials of Modern Software Engineering: Free the Practices from the Method Prisons!* in collaboration with Morgan & Claypool Publishers: ACM Books series #25, 2019.
- [3] P. Ray and P. Pal, “An agile approach to automate real estate crm (pre-sales) using scrum and essence,” in *Conference on Software Engineering Research & Practice (SERP), 2021*, (In Press).
- [4] P.-W. Ng, “Integrating software engineering theory and practice using essence: A case study,” *Science of Computer Programming*, vol. 101, pp. 66–78, April 2015.
- [5] F. P. Brooks (Jr.), *The Mythical Man-Month: Essays on Software Engineering*, 2nd ed. Addison-Wesley, 1995.
- [6] T. C. Lethbridge, J. Díaz-Herrera, R. J. L. Jr, and J. B. Thompson, “Improving software practice through education: Challenges and future trends,” in *Future of Software Engineering (FOSE 2007), International Conference on Software Engineering. 2007*, USA, May 2007, pp. 12–28.
- [7] I. Jacobson, B. Meyer, and R. Soley, “The semat initiative: A call for action,” Dr. Dobb’s, InformationWeek, December 2009. [Online]. Available: <https://www.drdoobs.com/architecture-and-design/the-semat-initiative-a-call-for-action/222001342>
- [8] I. Jacobson, P.-W. Ng, P. E. McMahon, I. Spence, and S. Lidman, “The essence of software engineering: The semat kernel,” *Communications of the ACM*, vol. 55, pp. 42–49, 2012.

- [9] —, *The Essence of Software Engineering: Applying the SEMAT Kernel*. Addison-Wesley Professional, 2013.
- [10] P. Johnson, M. Ekstedt, and I. Jacobson, “Where’s the theory for software engineering?” *IEEE Software*, vol. 29, pp. 94–96, 2012.
- [11] “Essence – kernel and language for software engineering methods ver 1.1 with change bars,” OMG specifications, Object Management Group, October 2018. [Online]. Available: <http://semat.org/documents/20181/57862/formal-15-12-02.pdf/e7ba1188-c477-4585-b18a-06937f0e62f3>
- [12] E. W. Dijkstra, *A discipline of programming*. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1976.
- [13] —, “On the role of scientific thought,” in *Selected Writings on Computing: A personal Perspective*, D. Derickson, Ed. New York: Springer-Verlag, 1982, pp. 60–62.
- [14] S. Liu, “A systematic mapping study on semat and its methods,” Master’s thesis, Waseda University, Japan, 2018.
- [15] S. Tyszberowicz, R. Heinrich, B. Liu, and Z. Liu, “Identifying microservices using functional decomposition.” in *Dependable Software Engineering. Theories, Tools, and Applications. SETTA 2018.*, ser. Lecture Notes in Computer Science, X. Feng, M. Müller-Olm, and Z. Yang, Eds., vol. 10998. Springer, Cham, 2018, pp. 50–65.
- [16] J. Lewis and M. Fowler. (2014) Microservices. [Online]. Available: <https://martinfowler.com/articles/microservices.html>
- [17] E. Evans, *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley Professional, 2004.
- [18] V. Vernon, *Implementing Domain-Driven Design*. Addison-Wesley Professional, 2013.
- [19] R. H. Steinegger, P. Giessler, B. Hippchen, and S. Abeck, “Overview of a domain-driven design approach to build microservice-based applications,” in *SOFTENG 2017*, M. Kajko-Mattsson and P. Ellingsen, Eds., Venice, Italy, April 2017, pp. 79–87.
- [20] D. Namiot and M. Sneps-Sneppe, “On micro-services architecture,” *International Journal of Open Information Technologies*, vol. 2, no. 9, pp. 24–27, 2014.
- [21] R. Heinrich, A. van Hoorn, H. Knoche, F. Li, L. E. Lwakatare, C. Pahl, and S. Schulte, “Performance engineering for microservices: Research challenges and directions,” in *ICPE ’17 Companion: Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion*, April 2017, p. 223 – 226.

- [22] R. C. Martin, *Agile Software Development, Principles, Patterns, and Practices*. Pearson, 2003.
- [23] S. Hassan, N. Ali, and R. Bahsoon, “Microservice ambients: An architectural meta-modelling approach for microservice granularity,” in *2017 IEEE International Conference on Software Architecture (ICSA)*, 2017, pp. 1–10.
- [24] M. E. Conway, “How do committees invent,” *Datamation*, vol. April, pp. 28–31, 1968.
- [25] E. Landre, H. Wesenberg, and H. Rønneberg, “Architectural improvement by use of strategic level domain-driven design,” in *OOPSLA ’06*. New York, United States: Association for Computing Machinery, 01 2006, pp. 809–814.
- [26] O. Vogel, I. Arnold, A. Chughtai, and T. Kehrer, *Software Architecture: A Comprehensive Framework and Guide for Practitioners*. Springer-Verlag Berlin Heidelberg, 2011.
- [27] A. Cockburn. (2005) The pattern: Ports and adapters (alternatively hexagonal architecture). Last Accessed: September, 2020. [Online]. Available: <https://alistair.cockburn.us/hexagonal-architecture/>
- [28] B. Hippchen, P. Giessler, R. Steinegger, M. Schneider, and S. Abeck, “Designing microservice-based applications by using a domain-driven design approach,” *International Journal on Advances in Software (1942-2628)*, vol. 10, pp. 432 – 445, 12 2017.
- [29] J. Palermo. (2008) The onion architecture. Last Accessed: September, 2020. [Online]. Available: <https://jeffreypalermo.com/2008/07/the-onion-architecture-part-1/>
- [30] B. Bruegge and A. H. Dutoitt, *Object-Oriented Software Engineering Using UML, Patterns, and Java*. Prentice Hall Pressl, August 2009.
- [31] W. M. P. van der Aalst, K. Van Hee, and G. Houben, “Modelling and analysing workflow using a petri-net based approach,” in *Proceedings of the second Workshop on Computer-Supported Cooperative Work, Petri nets and related formalisms*. of, 1994, pp. 31–50.
- [32] C. A. Petri, “Kommunikation mit Automaten,” *PhD, University of Bonn, West Germany*, 1962.
- [33] T. Murata, “Petri nets: Properties, analysis and applications,” *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989.
- [34] R. Zurawski and M. Zhou, “Petri net and industrial application: A tutorial,” *Industrial Electronics, IEEE Transactions on*, vol. 41, pp. 567 – 583, 01 1995.

- [35] “CRM (Customer Relationship Management),” Website, Salesforce. [Online]. Available: <https://www.salesforce.com/in/crm/>
- [36] “Analysts discuss technology trends and their impact on customer experience at the gartner customer experience & technologies summit in sydney, june 17-18.” [Online]. Available: <https://www.gartner.com/en/newsroom/press-releases/2019-06-17>
- [37] “Customer relationship management,” Insights, Bain and Company, April 2018. [Online]. Available: www.bain.com/insights/management-tools-customer-relationship-management
- [38] R. Shaw, *Computer Aided Marketing and Selling*. 313 Washington Street Newton, MA, United States: Butterworth-Heinemann, 1991.
- [39] M. Apgar, “What every leader should know about real estate,” Article, Harvard Business Review, November 2009. [Online]. Available: <https://hbr.org/2009/11/what-every-leader-should-know-about-real-estate>
- [40] —, “Managing real estate to build value,” Article, Harvard Business Review, November-December 1995. [Online]. Available: <https://hbr.org/1995/11/managing-real-estate-to-build-value>
- [41] “10 ways business process automation is changing real estate,” Blog, Kreyon Systems, August 2016. [Online]. Available: <https://www.kreyonsystems.com/Blog/10-ways-business-process-automation-is-changing-real-estate/>
- [42] J. F. Smart, *BDD in Action: Behavior-driven development for the whole software lifecycle*. Manning Publications, May 2014.
- [43] M. Wynne, A. Hellesoy, and S. Tooke, *The Cucumber Book: Behaviour-Driven Development for Testers and Developers (Pragmatic Programmers)*, 2nd ed. Pragmatic Bookshelf, 2017.
- [44] S. Millett and N. Tune, *Patterns, Principles, and Practices of Domain-Driven Design*. Wiley, May 2015.
- [45] B. Hippchen, M. Schneider, I. Landerer, P. Giessler, and S. Abeck, “Methodology for splitting business capabilities into a microservice architecture: Design and maintenance using a domain-driven approach,” in *SOFTENG 2019*, L. Lavazza, Ed. Valencia, Spain: Pearson Curran Associates, Inc., 24-28 March 2019, p. 53–61.
- [46] S. Newman, *Building Microservices*. O’Reilly Media, Inc., February 2015.
- [47] D. C. Schmidt, “Guest editor’s introduction: Model-driven engineering,” *Computer*, vol. 39, 2006.

- [48] A. G. Kleppe, J. Warmer, and W. Bast, *MDA Explained: The Model Driven Architecture: Practice and Promise*, ser. Object Technology Series. The Addison-Wesley, April 2003.
- [49] I. Jacobson, G. Booch, and J. Rumbaugh, *Unified Software Development Process*, 1st ed., ser. Object Technology Series. The Addison-Wesley, 1999.
- [50] G. Fairbanks, *Just Enough Software Architecture: A risk-driven approach*, 1st ed. Marshall & Brainerd, August 2010.
- [51] K. Beck, *Test Driven Development: By Example*, 1st ed. Addison-Wesley Professional., November 2002.
- [52] V. Vernon, *Domain-Driven Design Distilled*. Addison-Wesley Professional, 2016.
- [53] P. Ray and P. Pal, “Extending the semat kernel for the practice of designing and implementing microservice-based applications using domain driven design,” in *IEEE 32nd Conference on Software Engineering Education and Training (CSEE&T), 2020*, Munich, Germany, November 2020.
- [54] —, “An essence based framework using a domain driven design approach to address microservices lifecycle from identification to implementation,” in *Conference on Scientific Computing (CSC), 2021*, (In Press).
- [55] J. Sutherland. (2020, October 7) Better scrum with essence. Blog. scruminc. [Online]. Available: <https://www.scruminc.com/better-scrum-with-essence/>
- [56] N. Kock, J. Verville, A. Danesh-Pajou, and D. DeLuca, “Communication flow orientation in business process modeling and its effect on redesign success: Results from a field study,” *Decision Support Systems*, vol. 46, pp. 562–575, 2009.
- [57] I. Jacobson. (2018) Alpha State Card Games — Agile Software Development. Ivar Jacobson International. [Online]. Available: <https://www.ivarjacobson.com/alphastatecards>
- [58] M. Cohn, *User Stories Applied for Agile Software Development*. Addison-Wesley, 2009.
- [59] D. Harel, “Statecharts: a visual formalism for complex systems,” *Science of Computer Programming*, vol. 8, no. 3, pp. 231–274, 1987.
- [60] G. Booch, J. Rumbaugh, and I. Jacobson, “Unified modeling language user guide, the (2nd edition) (addison-wesley object technology series),” *J. Database Manag.*, vol. 10, 01 1999.
- [61] J. A. Saldhana, “Uml diagrams to object petri net models: An approach for modeling and analysis,” 2000.
- [62] T. M. Koulopoulos, *The workflow imperative: Building real world business solutions*. John Wiley & Sons, Inc., 1997.

- [63] K. Hales and M. Lavery, *Workflow management software: the business opportunity*. Ovum, 1991.
- [64] J. Lee, M. Gruninger, Y. Jin, T. Malone, A. Tate, G. Yost, P. W. Group *et al.*, “The process interchange format and framework,” *The knowledge engineering review*, vol. 13, no. 1, pp. 91–120, 1998.
- [65] M. Zhou, K. McDermott, P. Patel, and T. Tang, “Construction of petri net based mathematical models of an fms cell,” in *Conference Proceedings 1991 IEEE International Conference on Systems, Man, and Cybernetics*, 1991, pp. 367–372 vol.1.
- [66] W. Aalst, “The application of petri nets to workflow management,” *Journal of Circuits, Systems, and Computers*, vol. 8, pp. 21–66, 02 1998.
- [67] G. Bracchi and B. Pernici, “Trends in office modeling,” in *Proceedings of the IFIP TC 8 working conference on office systems on Office Systems*, 1986, pp. 77–97.
- [68] M. D. Zisman, *Representation, specification and automation of office procedures*. University of Pennsylvania, 1977.
- [69] C. A. Ellis and G. J. Nutt, “Modeling and enactment of workflow systems,” in *Application and Theory of Petri Nets 1993*, M. Ajmone Marsan, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 1–16.
- [70] H.-D. Burkhard, “On fairness in petri nets,” *Rostocker Mathematisches Kolloquium*, pp. 85–96, 1982.

