

ASSIGNMENT OF MASTER'S THESIS

Title:	Messaging and Task Management Application Based on the PSI Theory
Student:	Bc. Roman Lanský
Supervisor:	Ing. Robert Pergl, Ph.D.
Study Programme:	Informatics
Study Branch:	Web and Software Engineering
Department:	Department of Software Engineering
Validity:	Until the end of summer semester 2016/17

Instructions

The Performance in Social Interaction (PSI) theory is one of the basic Enterprise Engineering theories that studies interaction and communication between people and helps with the distribution of responsibilities. The aim of the thesis is to apply the theory to increase productivity in daily professional and personal lives through a personal application for communication and task management based on the PSI Transaction Axiom.

1. Analyse the usability of the PSI theory for everyday communication and task management.
2. Create a design of the application.
3. Create a web-based prototype of the application as a proof-of-concept.
4. Test the application on real situations, evaluate the results.

This thesis is a subject of cooperation between the #ForMetis company and FIT CTU #CCMi.

References

Dietz, J.L.G., 2006. Enterprise ontology: theory and methodology. Springer, Berlin; New York.

L.S.

Ing. Michal Valenta, Ph.D.
Head of Department

prof. Ing. Pavel Tvrđík, CSc.
Dean

Prague February 9, 2016

CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF SOFTWARE ENGINEERING



Master's thesis

Messaging and Task Management Application Based on the PSI Theory

Bc. Roman Lanský

Supervisor: Ing. Robert Pergl, PhD.

17th February 2017

Acknowledgements

I want to thank my supervisor Ing. Robert Pergl, PhD. for introducing me to the subject, my colleague Ing. Marek Skotnica for giving me a technological insight, and Dr. ir. Steven van Kervel for filling me with enthusiasm about the topic.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on 17th February 2017

.....

Czech Technical University in Prague
Faculty of Information Technology

© 2017 Roman Lanský. All rights reserved.

This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).

Citation of this thesis

Lanský, Roman. *Messaging and Task Management Application Based on the PSI Theory*. Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2017.

Abstrakt

PSI teorie tvoří jednu ze stěžejních částí oblasti Enterprise Engineering. Pokrývá společenské a komunikační aspekty organizací, které jsou považovány za základní stavební kameny podniků. Tato teorie má kořeny v sociologii, především v Habermasově teorii komunikativního jednání. Cílem práce je představit uplatnění této teorie v praxi v podobě aplikace pro posílání zpráv a správu úkolů vhodné pro každodenní využívání. Jedním z hlavních záměrů práce je zprostředkování této teorie široké veřejnosti.

Tato práce zkoumá hlavní teorie a pod ně spadající axiomy, podle nichž jsou určeny funkční požadavky a na jejich základě je navržena webová aplikace vycházející z PSI teorie komunikace, je umožňuje správu požadavků a slibů mezi uživateli. Průběh vývoje prokázal, že tato teorie je pro podobné využití vhodná a vysoce užitečná, jelikož vytváří pevný základ pro funkční logiku aplikace.

Klíčová slova PSI, teorie PSI, ψ -teorie, DEMO, Enterprise Engineering, Enterprise Ontology, Ontologie, Aplikace pro posln zprv a sprvu kol

Abstract

PSI theory is one of the integral parts in the field of Enterprise Engineering. It covers the social and communication aspects of the organisation, which are considered the essential building blocks of enterprises. The theory has roots in the sociology, mainly in the Habermas's Theory of communicative action. The aim of the thesis is to present the application of the theory in the form of a messaging and task management application suited for everyday use. One of the main goals is to expose the theory to broad public.

The thesis explores core theories and underlying axioms to devise a set of functional requirements upon which designs a web-based application that utilises the PSI theory of communication to manage requests and promises between users. The theory proves useful and well suited for the task, laying a solid foundation for the application's logic.

Keywords PSI, PSI-theory, ψ -theory, DEMO, Enterprise Engineering, Enterprise Ontology, Ontology, Task and Management application

Contents

Introduction	1
Aim of the thesis	1
Key points	2
Structure and methodology outline	2
I Theoretical part	3
1 Theory	5
1.1 Chapter introduction	5
1.2 Ontology	5
1.3 Enterprise ontology	6
1.4 Enterprise engineering	7
1.5 PSI Theory	7
1.6 DEMO	23
2 State of the art	25
2.1 Chapter introduction	25
2.2 Trello	25
2.3 Google Inbox	26
2.4 Asana	26
2.5 Wunderlist	27
2.6 Chapter conclusion	27
II Practical part	29
3 Analysis of requirements	31
3.1 Chapter introduction	31
3.2 Operation axiom analysis	31

3.3	Transaction axiom analysis	33
3.4	Composition axiom analysis	35
3.5	Distinction axiom analysis	36
3.6	Functional requirements not tied to the theory	36
3.7	Non-functional requirements analysis	36
3.8	Requirements summary	37
3.9	Chapter conclusion	37
4	Used technology	41
4.1	Chapter introduction	41
4.2	.NET	41
4.3	C#	43
4.4	ASP.NET Core	43
4.5	LINQ	45
4.6	Entity Framework	46
4.7	Web-specific tools and libraries	47
4.8	Dependency management tools	48
5	Application design	49
5.1	Vocabulary specification	49
5.2	Use case diagrams	50
5.3	Conceptual model	56
5.4	Required views	57
5.5	State machine	58
5.6	Transaction chaining	61
5.7	Application prototype	61
6	Test scenarios	65
6.1	Chapter introduction	65
6.2	Chapter summary	68
	Conclusion	69
	Evaluation of functional requirements	69
	Evaluation of non-functional requirements	69
	Opportunities	69
	Thesis conclusion	71
	Bibliography	73
	A Acronyms	77
	B Contents of enclosed CD	79

List of Figures

1.1	EE theories	7
1.2	Coordination act	10
1.3	Standard notation	11
1.4	Actors	12
1.5	The basic transaction pattern	13
1.6	The basic transaction pattern	14
1.7	The standard pattern of a transaction	15
1.8	The complete transaction pattern	16
1.9	The composition pattern	16
1.10	The self-activation	18
1.11	The distinction axiom	18
1.12	The coordination act	19
1.13	The claims	20
1.14	Intelligent System Partitioning	22
1.15	Integrated System Perspectives	23
2.1	Trello	25
2.2	Google Inbox	26
2.3	Asana	26
2.4	Wunderlist	27
4.1	.Net Standard	42
4.2	MVC Pattern	44
4.3	Entity Framework	46
4.4	AdminLTE	47
5.1	Actor hierarchy	50
5.2	Anonymous user use cases	51
5.3	Application user use cases - promises	54
5.4	Application user use cases - contacts	54
5.5	Business user use cases	55

5.6	Basic transaction	55
5.7	Basic conceptual model	56
5.8	Basic state machine	58
5.9	Initiator revokes state machine	59
5.10	Executor revokes state machine	59
5.11	Complete state machine	60
5.12	State machine extension - save	60
5.13	Simple chaining	61
5.14	Onion architecture	62
5.15	Database schema	63

List of Tables

3.1	Functional requirements	38
3.2	Non-functional requirements	39
6.1	Evaluation of functional requirements	70
6.2	Non-functional requirements	71

Introduction

Aim of the thesis

The aim of the diploma thesis is to propose a conceptual application for messaging and task management, based on the Performance in Social Interaction (PSI) theory of enterprise engineering. The theory studies interaction and communication between people and helps with the distribution of responsibilities by proposing a transaction axiom, a universal model of communication patterns. This axiom is a foundation for finding and understanding basal acts occurring during interaction between people, where one agent requests something from the other participant. The theory is strongly leveraged in the Design & Engineering Methodology for Organizations (DEMO), an enterprise modelling methodology in which communication is the cornerstone of organisations. The DEMO is a robust tool for enterprise process analysis and representation based on several underlying scientific theories. As such, it has a potential downside of not being readily comprehensive for people unfamiliar with it, which makes the methodology difficult to approach and partially conceals the benefits it could bring.

A possible solution for providing people with knowledge of the positives is to take some core aspects of the axioms the theory builds on and incorporate them unobtrusively in common activities they perform daily. The thesis is intended as a complement to the DEMO by focusing on the everyday aspect of communication, specifically on creating a platform for keeping track of tasks required from and by its users. The platform is based on aforementioned PSI theory, providing users with a way to manage their responsibilities emerging from ordinary interactions with other people, from noting simple tasks through creating ad hoc process chains to enabling businesses to offer their services in the form of transparent and easy-to-reach entry point transactions.

From the conceptual point of view, the thesis aspires to explore the possibility of simplifying the PSI theory on the presentation front. It must be easily comprehensive to a wide range of users without any prior knowledge of the theory and its principles and terms.

Key points

The key points of the thesis are:

- Introduction of the ψ -theory and its axioms
- The design and implementation of an application prototype that employs the ψ -theory in an approachable manner
- Utilisation of modern technologies

Structure and methodology outline

In the first part of the thesis the theories are explored and presented. The theoretical chapter is a compilation of scientific resources on the topic of ψ -theory and Enterprise Ontology and provides an insight into the field of Enterprise Engineering. It also contains a short chapter about the state-of-the-art applications similar to the goal of the thesis.

Second part applies the theory in the form of an application prototype. Firstly the functional and non-functional requirements are stated. Then the application design is shown, followed by an introduction of used technologies.

In the second half of the practical part, the key features of the application prototype are shown and a proof of concept in the form of test scenarios is provided. The thesis then evaluates and summarises its results.

Part I

Theoretical part

Theory

1.1 Chapter introduction

The first chapter of the thesis presents underlying theories upon which the work stands. It is a compilation of researched source materials. The chapter defines basal and related concepts in order to set a foundation for a formulation of requirements, both functional and non-functional. The backbone of the chapter is the section 1.5 which introduces the ψ -theory and its major components. Special attention is given to the transaction axiom owing to the fact that an application of this axiom is the aim of the thesis.

The chapter starts with a general notion of what ontology is, as this term is crucial in the understanding of the enterprise ontology (and consequently the enterprise engineering). The core theory follows divided into multiple subsections to ensure it is covered thoroughly. The chapter ends with a section devoted to the DEMO methodology¹.

1.2 Ontology

Ontology is a philosophical and scientific discipline holding a prominent position in theoretical computer science. According to Lofaro [1], it can be defined as:

“Ontology:

1. In philosophy, ontology is the study of the nature of being, becoming, existence, or reality, as well as the basic categories of being and their relations. Traditionally listed as a part of the

¹The DEMO methodology is not in the scope of the thesis, but the author decided to include a brief overview as it is closely connected to the topic and will be addressed later in the chapter regarding future opportunities.

major branch of philosophy known as metaphysics, ontology deals with questions concerning what entities exist or can be said to exist, and how such entities can be grouped, related within a hierarchy, and subdivided according to similarities and differences.

2. In computer science and information science, an ontology formally represents knowledge as a set of concepts within a domain, and the relationships between pairs of concepts. It can be used to model a domain and support reasoning about concepts.” (Lofaro, 2015)

As one can see from the two definitions, there is a difference in the philosophical and engineering approach to ontology. The former revolves around more general questions and can be approximated as “[...] the study of what there is.” [2]. It deals with philosophical problems of (non-)existence of things or entities. The latter could be considered as a concrete instance of this metaphysical study. Gruber (1993) in [3] provides the following definition: “An ontology is an explicit specification of a conceptualization. The term is borrowed from philosophy, where an ontology is a systematic account of Existence. For knowledge-based systems, what *exists* is exactly that which can be represented.” Those two notions can be distinguished from each other grammatically as well² – the philosophical notion is a mass noun and thus will appear without an article, the one used in computer science is a regular noun and can have a plural form. [4] There is only one philosophical ontology, in contrast to many kinds of scientific ontologies, which are usually named by the domain they describe. The thesis makes use of the enterprise ontology, which will be addressed later in section 1.3.

In the field of the enterprise engineering (see section 1.4), ontology is a cornerstone – allowing engineers to conceptualise entities, their properties and connections between them, leading to better understanding of a given problem. Proper utilisation of the ontology discipline enables an abstraction of domains and in turn an emergence of patterns, a simplification of complex systems and provides a base for means of automation.

1.3 Enterprise ontology

Dietz in [5] declares that: “An enterprise ontology is a formal and explicit specification of a shared conceptualization among a community of people of an enterprise (or a part of it). It includes static, kinematic, and dynamic aspects.

²Although many authors do not use the terms correctly, as can be seen in the provided definitions.

<p>Ideological Theories <i>selecting the things to make</i> ethical, political, etc. ideas EE-theories: σ-theory</p>	<p>Technological Theories <i>designing and making things</i> analysis and synthesis EE-theories: β-theory, ν-theory</p>
<p>Ontological Theories <i>understanding the nature of things and their use by us</i> explanation and prediction EE-theories: ϕ-theory, τ-theory, δ-theory, π-theory, ψ-theory</p>	
<p>Philosophical Theories <i>philosophical foundations</i> epistemology, mathematics, phenomenology, logic EE-theories: ω-theory</p>	

Figure 1.1: Classification scheme for enterprise engineering theories [6]

In particular, an enterprise ontology satisfies the next five quality requirements (C₄E): Coherence, Comprehensiveness, Consistency, Conciseness, Essence.” The PSI theory underlies the notion of enterprise ontology, as presented by Dietz.

1.4 Enterprise engineering

Enterprise Engineering is an emerging discipline that consists of many theories (an overview can be seen on figure 1.1). The two main concepts used by Enterprise Engineering are the Enterprise Ontology (see section 1.3) and Enterprise Architecture (which, according to [5], “guides the (re)design and (re)engineering of an enterprise such that its operation is compliant with its mission and strategy, and with all other laws and regulations.”)

Enterprise engineering may appear abbreviated and referred to as EE in the following text.

1.5 PSI Theory

In this main section of the chapter the foundation of the thesis is presented: the ψ -theory. The section is divided into three parts - the first one generally describes the basic notion of what the theory is and why it exists. The second part is focused on the general ψ -theory and its axioms. This part is a staple of the thesis and an application of concepts depicted in it is the main aim of the whole work. The third and last part of this section is about the special ψ -theory which addresses the consequences of the general ψ -theory for the systemic ontology of organisations [7].

1.5.1 What is the ψ -theory

The abbreviation PSI in the PSI theory stands for the “Performance in Social Interaction”. As with all of the other major enterprise engineering theories, the abbreviation is commonly replaced by the Greek letter with the same pronunciation, in this case the letter ψ [”psi”]. Before proceeding to the definition of the scope of the theory, the clarification of the distinction between words *enterprise*, *organisation* and *business* is needed. Perinforma³ in [8] defines them as:

- **Enterprise:** the general term to refer to any kind of collaborative activity by human beings. Examples: companies, governmental agencies, health care institutions, sports clubs, and building projects. Every enterprise has a *business* and an *organisation*.
- **Organisation:** term to refer to the construction perspective on an enterprise. The organisation of an enterprise consists of a network of actor roles and transaction kinds⁴.
- **Business:** term to refer to the function perspective on an enterprise, in particular the function as perceived by the consumers of its services.

The theory focuses on construction and operation of organisations [6] and consists of two branches – a “general ψ -theory” and a “special ψ -theory”. The former defines four axioms revolving around the social aspect of organisations, claiming that “the human communication is the root of information and (social) action, and subsequently of organisation” [6]. It is the front side of the complete ψ -theory, while the latter represents the system side and the influences of the general ψ -theory on the conceptual modelling of systems.

“The ψ -theory is one of the currently eight theories that constitute the theoretical foundations of the discipline of enterprise engineering, as envisioned by the Ciao! Network.” [9] Proponents of the theory claim that by applying this theory along with a few supporting theories (which are together known as the enterprise ontology presented in section 1.3) we can achieve a goal of *intellectual manageability* when modelling, (re-)developing and implementing an enterprise. This claim is leveraged by the DEMO methodology (see section 1.6).

³Alicia P.C. Perinforma is the author of “The essence of organisation”, but is in fact a pseudonym of professor Jan L.G. Dietz, one of the most prominent people in the field of enterprise engineering.

⁴Actor roles and transaction kinds are explained later in the chapter in sections 1.5.2.1 and 1.5.2.2, respectively.

1.5.2 General ψ -theory

Allowing a simplification, the following can be stated: When devising a new theory, methodology, or ontology, one will usually start with discovering elements and patterns connecting them. After the initial discovery, those elements and patterns are matched against entities of a researched domain to uncover the set common among all the entities. This set is then used as a framework for the said domain and a basis for the theory, methodology, or ontology. From here it is implied that the knowledge of core elements and patterns is crucial in creating an intrinsic model of a domain and those core features must be presented when one aims to provide theoretical description of a domain.

The organisation is the domain of the general ψ -theory. Dietz and Hoogervorst in [6] establish the following: “The general ψ -theory explains that human communication is the root of information and (social) action, and subsequently of organisation. This basic understanding makes organisations primarily social systems, of which the elements are human beings, bestowed with appropriate authority and bearing the corresponding responsibility.” This quote summarises the idea behind the general ψ -theory – it identifies human beings as the core elements of enterprises and their communication as the base of operation and cooperation. The pattern of the communication and with it the whole general ψ -theory is primarily based on the Theory of Communicative Action by Jürgen Habermas ([10], [11]) and on Speech Act Theory ([12], [13]).

The general ψ -theory consists of four axioms:

1. **Operation axiom** defines the basic notions of an actor, a C-act/fact and a P-act/fact.
2. **Transaction axiom** presents a general pattern of communication between actors - a transaction.
3. **Composition axiom** regards relations between transactions.
4. **Distinction axiom** pertains the communication abilities of actors.

These axioms will be explained in more detail in their own subsections. Together they act as a framework upon which the ψ -theory stands and cover the essential social aspect of enterprises.

Following subsections are based on the primary bibliographical resource - the “Enterprise Ontology - Theory and Methodology” by professor Jan L.G. Dietz [14].

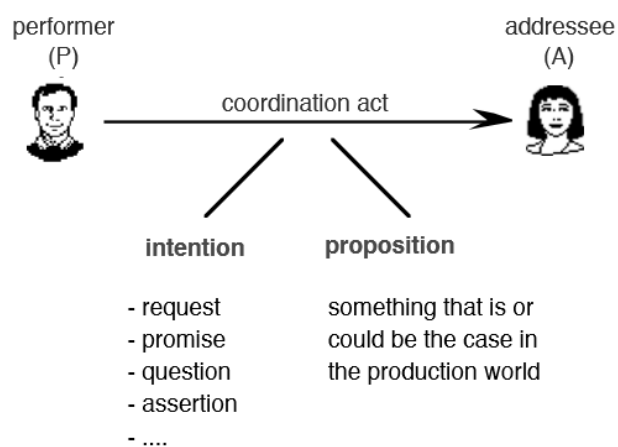


Figure 1.2: Representation of a coordination act [14]

1.5.2.1 Operation axiom

“The first axiom of the ψ -theory states that the operation of an enterprise is constituted by the activities of actor roles, which are elementary chunks of authority and responsibility, fulfilled by subjects. In doing so, these subjects perform two kinds of acts: production acts and coordination acts. These acts have definite results: production facts and coordination facts, respectively. The axiom is commonly referred to as the operation axiom.” [14]

The operation axiom defines two distinct areas (‘worlds’) in which the theory takes place. Those areas are the production world (‘P-world’) and the coordination world (‘C-world’). The two worlds have a clearly defined state in every point of time - the state is a set of every production (or coordination, respectively) fact created before the said point of time. It is worth noting that facts cannot be deleted once created, they can only be cancelled by creating a so-called ‘anti-fact’. This guarantees the consistence of the worlds and ensures that the knowledge of the current state is sufficient to have complete information of those two worlds.

Every enterprise operates by having its subjects (human beings) take part in activities based on actor roles, performing acts in both of these worlds. By successfully performing an act, a corresponding fact comes into existence.

Coordination acts (or ‘C-acts’) are acts taking place in the C-world. As a quote from Perinforma in [15]: “A coordination act is the focal point of every communication [...]”. A C-act is defined as: “An act performed by one actor,

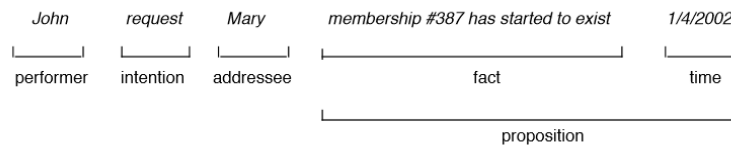


Figure 1.3: Standard notation of a coordination act [14]

called the performer, and directed to another actor, called the addressee.” [14] The C-act can be distinguished from general communication act by having two important parts - the intention act and the proposition act, both contained within the single C-act. The intention part is expressed by the performer using an adequate social construct, i.e. a request, promise, question, assertion. The proposition part always concerns a production fact (which ensues that every C-act must have a corresponding P-fact, although possibly non-existing) and a time specification. This time indicates when the P-act/fact should occur.

Figure 1.2 shows a representation of a generalised coordination act. To represent a C-act, one can use a ‘*standard notation*’, depicted on figure 1.3. This notation makes it simple to identify all the necessary parts of a C-act and is more suited for automated systems than the natural spoken form.

Production acts (or ‘P-acts’) are acts taking place in the P-world. The quote about C-acts from Perinforma in [15] in the previous section continues as: “[...] a production act is the crucial core of every transaction.” [15] It was stated that every C-act is bound to relate to a P-fact, which can be either existing or desired to exist. While the coordination acts are about discussion and specification of P-facts, the P-act’s purpose is to bring the P-fact itself into existence.

There are two kinds of P-facts, material and immaterial [14]. Although they are clearly different from each other in the real world, they behave similarly in the P-world, which is shown on the following two examples:

- **Example 1: An immaterial P-fact** A verdict in the case #123 has been delivered.
- **Example 2: A material P-fact** A yellow wooden chair has been sold to the customer⁵.

In the first example, the verdict is initially *stated* by the executor (i.e. a judge). If the initiating actor (which would be the state in this case) *accepts* the verdict (as lawful), the immaterial P-fact is created. In the second example, it may

⁵P-facts are usually more specific than this because they must be uniquely identified, but for the sake of the illustrative example the author of the thesis used a simplified form.

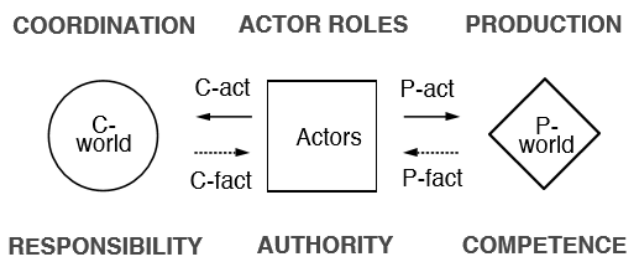


Figure 1.4: Actors and their responsibility, authority, and competence [14]

seem that the P-fact came into existence when the yellow chair was built. But in reality, the desired outcome for the initiator is not the fact of the chair being built, rather it is the change of ownership of that particular chair [13]. The craftsman *states* the chair is ready, the customer confirms it is the product he wanted (and not for example a red table) and *accepts* it. Only after the acceptance – which is a result of a cooperation act between the actors – is the material P-fact deemed existing.

Actors Every act is brought about by actor roles. These are “the essential units of authority and responsibility” [14]. An actor role is an abstraction from the subjects performing the role – thus the organisational model can declare authorisations and responsibilities without knowledge of the actual fulfilment of the roles, which is a matter of implementation. This brings clarity to the distribution of competences and flexibility during (re-)design of an enterprise’s structure.

When performing an act, the roles are filled by subjects who become actors. An actor has a number of activities on his *agendum* and commits to dealing with them. An actor has to satisfy three conditions to be able to justly carry out the acts, which are featured on figure 1.4:

- **Responsibility** means that during the performance of an act, the actor will uphold values and norms as is expected of him for being a member of both an enterprise and society.
- **Authority** means that the actor has been permitted by the necessary laws to be able to perform the act.
- **Competence** means that the actor has the knowledge and expertise to be able to successfully finish the P-act and corresponding C-acts.

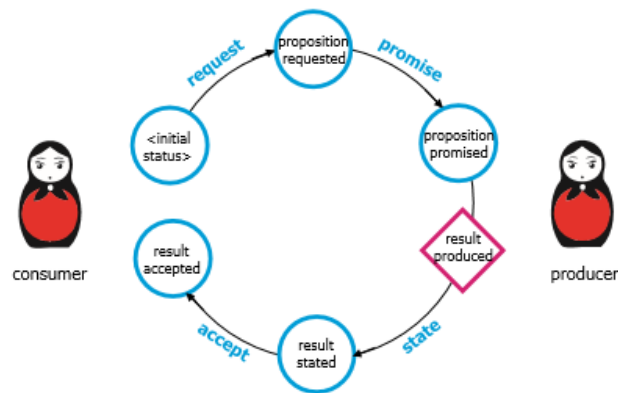


Figure 1.5: The basic transaction pattern [6]

1.5.2.2 Transaction axiom

Dietz and Hoogervorst in [6] claim that “C-acts and P-acts appear to occur in universal patterns, called *transactions*. A transaction involves two subjects, one in the role of consumer (*initiator*) and the other one in the role of producer (*executor*).” The transaction axiom postulates that the patterns are universal enough to encompass all the different processes, which can be “considered as paths through a generic coordination pattern” [14].

Transaction is a chain of coordination acts and at most one production act. It can be divided into three phases, the order phase (‘O-phase’), the execution phase (‘E-phase’) and the result phase (‘R-phase’). It is strictly between two actors – the actor who starts the transaction and expects a P-act/fact to be created is the *initiator*. The second actor who is supposed to produce the P-act/fact is the *executor*. Every actor role can be an initiator of multiple transactions, but is an executor of exactly one transaction kind. The transaction kind is a set of transactions creating the same product kind.

There are three transaction patterns distinguished from each other by complexity. The basic transaction pattern is contained within the standard transaction pattern, which is incorporated in the complete transaction pattern.

Basic transaction pattern is the simplest version of the three. It shows only the ‘happy path’ where the consumer (initiator) requests a proposition. The producer (executor) promises he will fulfil the proposition, produces a result and states the result. The transaction is concluded with the consumer accepting the stated result (see figure 1.5). The basic pattern presents the four elementary C-acts and facts:

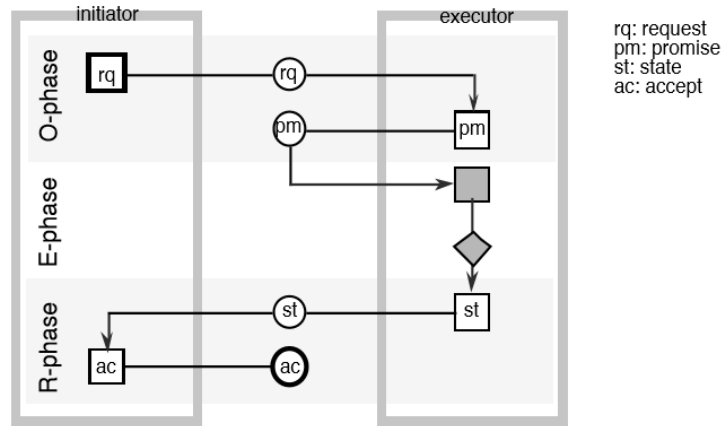


Figure 1.6: Formal notion of the basic transaction pattern [14]

1. *Request* (C-fact: P is Requested)
2. *Promise* (C-fact: P is Promised)
3. *State* (C-fact: P is Stated)
4. *Accept* (C-fact: P is Accepted)

The patterns are usually presented in formal notion, depicted in figure 1.6. This model shows scopes of interest for both actors, mapping of acts to aforementioned three phases, and relations between C-act/facts and P-act/fact. Acts are represented by squares, facts by circles. Gray fill is the P-world part of the transaction. Elements with special meaning have thick border: thick border on C-act marks an initiating act while thick border on C-fact signs a terminal fact for the transaction.

Standard transaction pattern is the common version of the three patterns. As shown on figure 1.7, it extends the ‘happy path’ with means of expressing dissent. Instead of promising, the executor can perform the C-act of *declining*. Likewise, instead of accepting, the initiator can perform the C-act of *rejecting* the stated product. Both those acts present a new type of a C-fact called a discussion state (the double disks). In these states, the actors can try to come to a mutual agreement on how to proceed with the transaction. The ultimate decision in the case of *decline* is on the initiator, who can either retry the request (possibly changed) or *quit* the transaction, moving the flow to a terminal state. Accordingly in the case of *reject* the executor decides whether to *state* the product again, or *stop* the transaction.

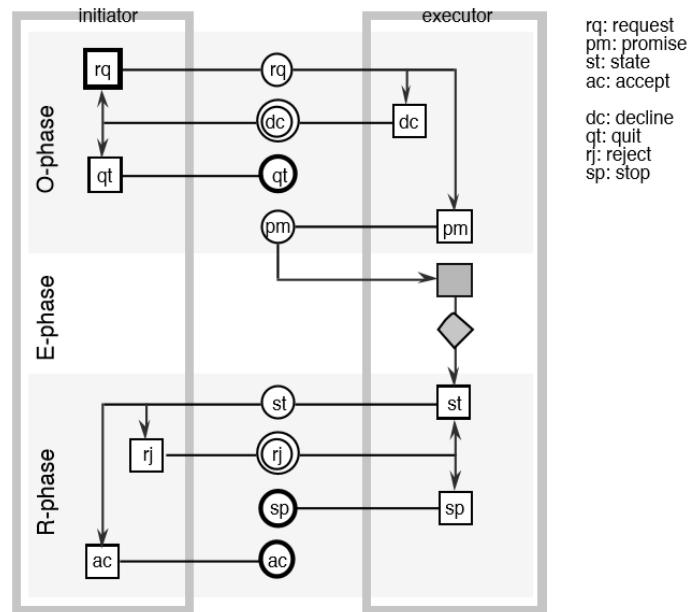


Figure 1.7: The standard pattern of a transaction [14]

Complete transaction pattern is considered the universal type of the patterns [6] and is represented by figure 1.8. This model adds four smaller subpatterns called the revocation (cancellation) patterns. Those side steps are present in case one of the actors regrets performing a step. The actor can use the revocation pattern and demand the transaction to move back before the revoked step. If the other actor allows it, the transaction continues from the new step. If he refuses, the transaction continues from the state it is currently in. Clearly, one can only revoke the steps that have been already performed.

Two things are worth mentioning regarding the transactions. The first is a tacit performance of C-acts. This enables some C-acts to be concluded automatically (*tacitly*) without the expressive consent of the actor. These acts may prove useful in case the explicit performance would only hinder the transaction flow and is of no significance. But even when performed tacitly, it is always there. The second thing is that not all of the transaction paths are always applicable in a particular process, it always depends on the context of a domain. But they are usually at least worth considering when modelling the domain as they can provide important insight on the workings and shortcomings of organization.

1. THEORY

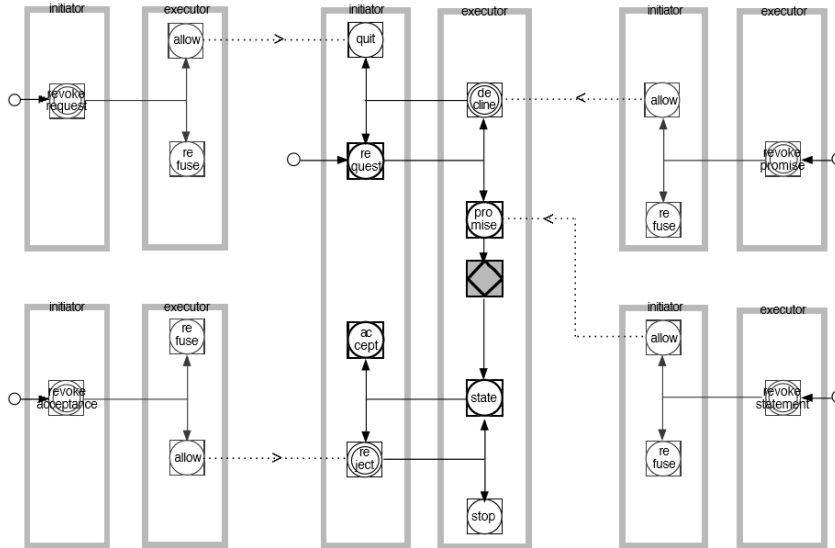


Figure 1.8: The complete transaction pattern [6]

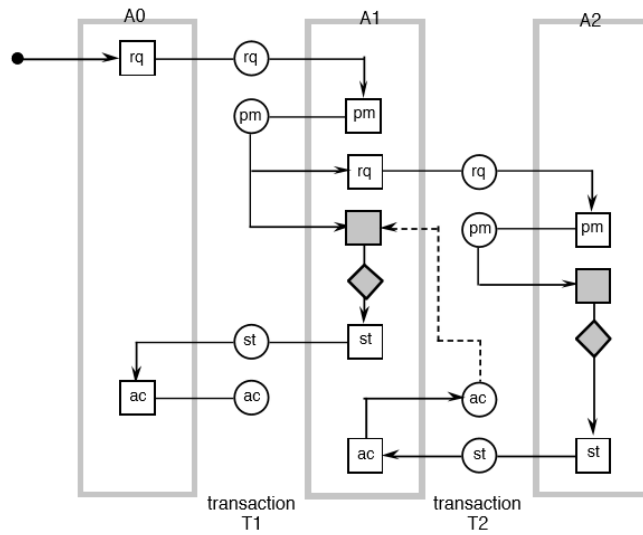


Figure 1.9: The composition pattern [14]

1.5.2.3 Composition axiom

The composition axiom addresses the fact that many products in the real world are not atomic and require more than one production step to be completed (for example, a material P-fact of creating a car needs the parts to be ready before the final assembly, or an immaterial P-fact of obtaining a building permit requires a number of approvals before it can be issued). The composition axiom enables connections between transactions. The connections are of two types: an initiation connection will issue one or more *requests* to other transactions after reaching some state in a transaction flow. The second type is a wait connection which blocks the flow of the transaction until the linked transaction reaches the desired state. Both are depicted on figure 1.9, where the actor A1 performs the request of T2 after the T1 is promised. He then waits with the P-act until the T2 is accepted.

The composition links are drawn from any of C-facts and either link to the *request* C-act of another transaction (in case of an initiation link) or any act (C or P) of another transaction (in case of a waiting link) [16]. Theoretically the source can be any of the C-facts, but commonly the terminating facts are used to avoid issues with consistency⁶.

The composition axiom also presents the only three ways how the transactions can be started:

1. **External initiation** (the black dot on figure 1.9) enables actors from outside of an enterprise to request transactions. This usually covers the services and products offered by an enterprise to external customers.
2. **Enclosed initiation** (the composite request of T2 on figure 1.9) happens inside an enterprise and is started as a reaction to another transaction.
3. **Self-activated initiation** (the backward arrow of T1 request on figure 1.9) covers the periodic and control activities [14]. After the transaction reaches the self-initiating state (the *request* in the provided example) it issues a request to itself.

The composition axiom allows the definition of the **business process** in [17]: “A *business process* is a collection of causally related transaction types, such that the starting step is either a request performed by an actor role in the environment (*external activation*) or a request by an internal actor role to itself (*self-activation*). Every transaction type is represented by the complete transaction pattern.” (as quoted from [14])

⁶Naturally, if one can find the reaction to, for example, the *state* fact instead of the usual *accept* fact desirable, he can do so.

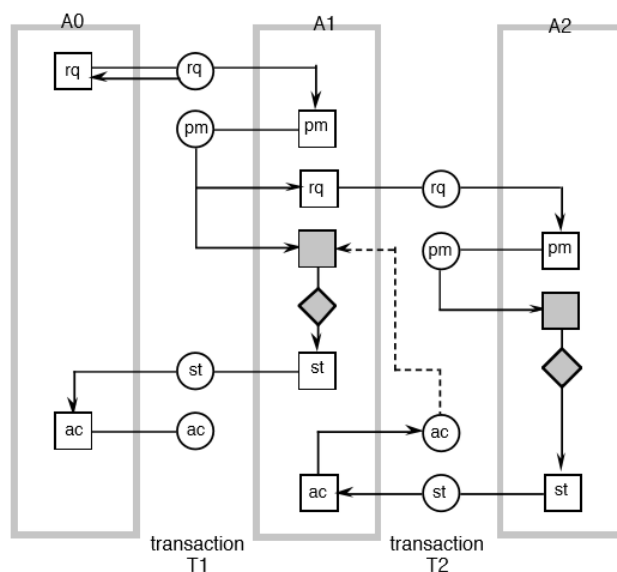


Figure 1.10: The self-activation [14]




COORDINATION	HUMAN ABILITY	PRODUCTION
<i>exposing commitment</i> (as performer) <i>evoking commitment</i> (as addressee)	 performa	<i>ontological action</i> (deciding, judging)
<i>expressing thought</i> (formulating) <i>educing thought</i> (interpreting)	 informa	<i>infological action</i> (reproducing, deducing, reasoning, computing, etc.)
<i>uttering information</i> (speaking, writing) <i>perceiving information</i> (listening, reading)	 forma	<i>datalogical action</i> (storing, transmitting, copying, destroying, etc.)

Figure 1.11: Summary of the distinction axiom [14]

1.5.2.4 Distinction axiom

“The fourth axiom of the ψ -theory states that there are three distinct *human abilities* playing a role in the operation of actors, called *performa*, *informa*, and *forma*. These abilities regard communicating, creating things, reasoning, and information processing. Because this axiom serves, in particular, in neatly separating our diverse concerns, it is called the distinction axiom.” [14]. It is summarised on figure 1.11.

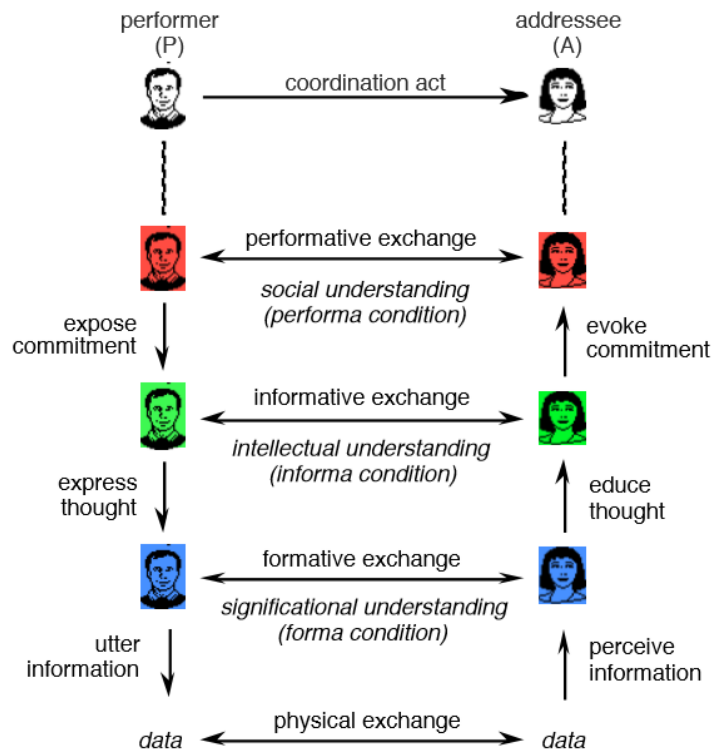


Figure 1.12: The process of performing a coordination act [14]

- **Forma** is a human ability to perceive/store/retrieve information. It encompasses formulation of sentences in languages and syntactical side of the communication. *Forma* means form.
- **Informa** is a human ability to communicate thoughts between people, to remember and recall knowledge. It concerns the content of information.
- **Performa** is a human ability to create new original things through the communication - commitments, decisions. It is considered the essential ability for doing business [14].

These three items are the basis for two important things - during coordination, they are all preconditions to a successfully performed act. During production, they can be used to create a distinction between three types of transactions.

The figure 1.12 shows the process of a coordination act. The performer and addressee must meet on four levels to be able to bring about the C-act: The physical layer requires the actors to (somehow) exchange the information. It may be in the form of direct speech, writing, or any other communication

	Constativa	Regulativa	Expressiva	
objective world				claim to truth
social world				claim to justice
subjective world				claim to sincerity
	<i>question assertion</i>	<i>request promise</i>	<i>praise apology</i>	

Figure 1.13: Summary of Habermas theory of communicative action [14]

technology. The *forma* condition is satisfied by the performer’s usage of such a form that is comprehensive for the addressee (i.e. using a language that both parties speak). The *informa* condition necessitates the addressee to understand the meaning of the commitment. And the final level of understanding, the *performa* condition, signifies that both subjects reach social understanding – the intent of the commitment.

The production part of transactions (and subsequently the whole transaction) can be categorised by the ability used to obtain the product. This creates the three types of transactions:

1. **Datological transactions** regard the storing, recalling, copying, transmitting and destroying data. They are representation of the *forma* ability.
2. **Infological transactions** concern the usage of knowledge – reasoning, computing, deriving and reproducing. They are representation of the *informa* ability.
3. **Ontological transactions** are the most essential transactions, because they cover the creation of original things. “[They], together with the involved (ontological) actor roles, constitute the ontology of an enterprise.” [18]. As such, they are representation of the *performa* ability.

Claims are a part of Habermas’s Theory of communicative action ([10], [11]). According to Habermas, “three spheres of human existence play a role in or are affected by communication. They are called the objective world, the subjective world, and the intersubjective or social world.” [14] When communicating, the speaker raises three validity claims towards the hearer. The communicative act is successful when and only when the claims are considered fulfilled.

If the hearer wants to challenge them, he has the right to do so. Each of the claims belongs to one of the aforementioned spheres and is considered a dominant one depending on the type of the communicative act, although all of them are always present. The figure 1.13 summarises the claims. They are described as follows:

- **Claim to truth:** Concerns the factual correctness of the communicative act. The acts where the claim to truth is dominant are called the *constantiva*.
- **Claim to justice:** Concerns the social authority and responsibility to fulfil the act. The acts where the claim to justice is dominant are called the *regulativa*.
- **Claim to sincerity:** Concerns the the subjective, inner meanings of the act. The acts where the claim to sincerity is dominant are called the *expressiva*.

In the enterprise ontology, the primary role is led by the claim to justice which means most of the acts are from the category of *regulativa*.

1.5.3 Special ψ -theory

While the general ψ -theory concerns the front (human) side, the special ψ -theory represents the back side. As described in [6]: “”PSI” is read as ”ISP”. It has two meanings: *Intelligent System Partitioning* and *Integrated System Perspectives*. [...] In addition, the formalisation of the special ψ -theory (is) the so-called CRISP model”. In this section, all the three meanings will be briefly described.

1.5.3.1 Intelligent System Partitioning

In the section 1.5.2.4 three types of transactions were described. “Based on this distinction, the network of transaction kinds and related actor roles that constitutes the complete ontological model of an organisation can usefully be partitioned in three subnetworks: one containing the original transaction kinds and actor roles, one containing the informational ones, and one containing the documental ones.” [6]. Naturally, the original transaction kinds and actor roles are the ontological transactions from the axiom, informational ones are the infological and documental ones are the datalogical. From this networks three parts of organisation can be derived, as seen on figure 1.14: The B(usiness)-, I(nformational)- and D(ocumental)-organisation. The pyramid shape of the figure shows that the I-organisation supports the B-organisation, and in turn is supported by the D-organisation. While the D- and I-organisations contain only the appropriate transaction kinds, the B-organisation may sometimes comprise of not only the original transaction kinds, but some informational

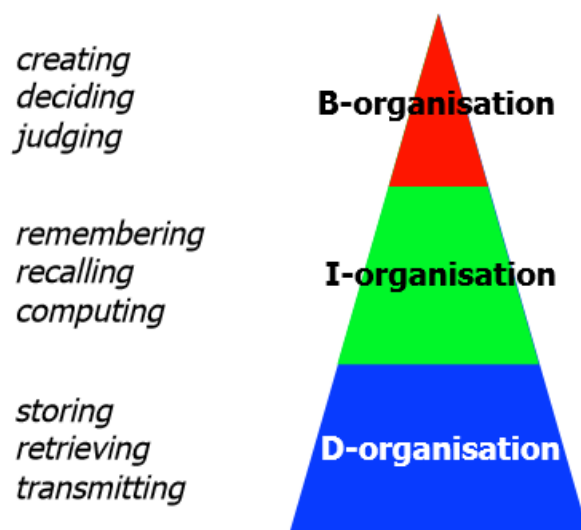


Figure 1.14: Intelligent System Partitioning [6]

and documental ones as well – “this is the case if the business of the enterprise is (also) to provide informational and/or documental services.” [6]

The so-called *essential model of the enterprise* is a notion used for the ontological model of the B-organisation in which the supporting services from the I-organisation are modelled as information links. Every enterprise has exactly one essential model and it represents its core functionality abstracted from implementation.

1.5.3.2 Integrated System Perspectives

“The complete ontological model of an organisation is divided into four sub models, each representing a particular perspective or view on the complete model: the construction model (CM), the process model (PM), the Fact Model (FM), and the Action Model (AM)” [6]. Those partial models are different views upon one main model in which all of them are entwined.

- **CM:** The construction model lists all the identified transaction kinds and the corresponding actor roles.
- **PM:** The process model maps the transactions using the composition axiom into the business processes.
- **FM:** The fact model contains business object classes, their properties, and business laws.

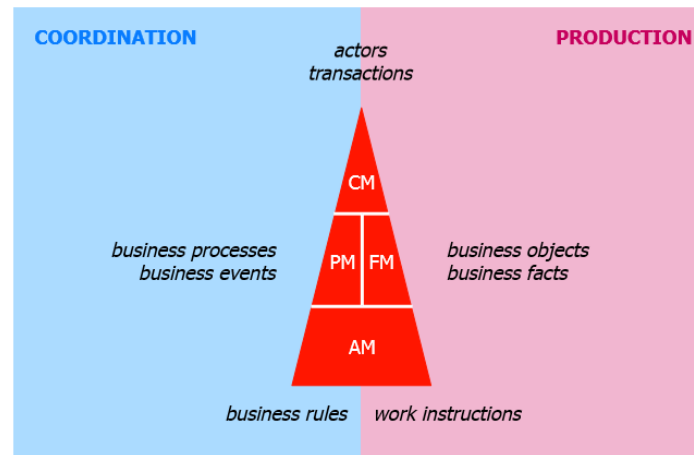


Figure 1.15: Integrated System Perspectives [6]

- **AM:** The action model is a collection of business rules and work instructions.

1.5.3.3 CRISP model

“The CRISP model is a formalisation of the special ψ -theory, in which the intrinsic connection of the four sub models, as presented [on figure 1.15], is clarified.” [8] *Crispies* are a ψ -theory based automata.

1.6 DEMO

DEMO is an acronym for ‘Design & Engineering Methodology for Organizations’. It is a methodology based on principles of Enterprise Engineering, especially the Enterprise Ontology and ψ -theory. DEMO gives an insight in organisations by mapping them using the four basic models, depicted on figure 1.15.

State of the art

2.1 Chapter introduction

This chapter briefly explores the state of the art in the field of messaging and task management applications. As there is a great number of such programs, the author chose four representatives of major approaches.

2.2 Trello

Trello (on the figure 2.1) is a lightweight dashboard application. It tracks tasks using a system of pins and dragging them from one notice board to another. Favoured by many programmers for its nice and clean interface.

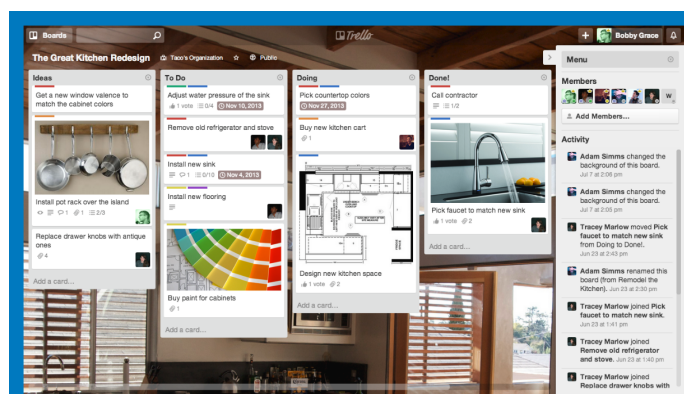


Figure 2.1: Trello [19]

2. STATE OF THE ART

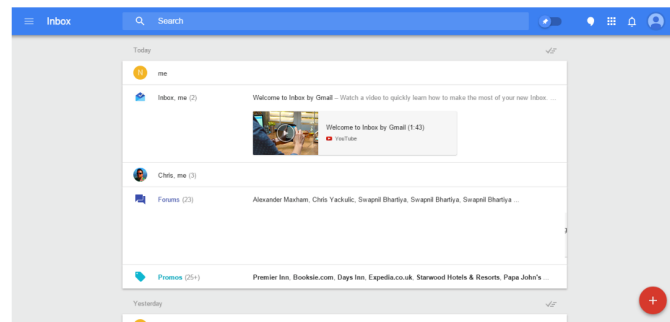


Figure 2.2: Google Inbox [20]

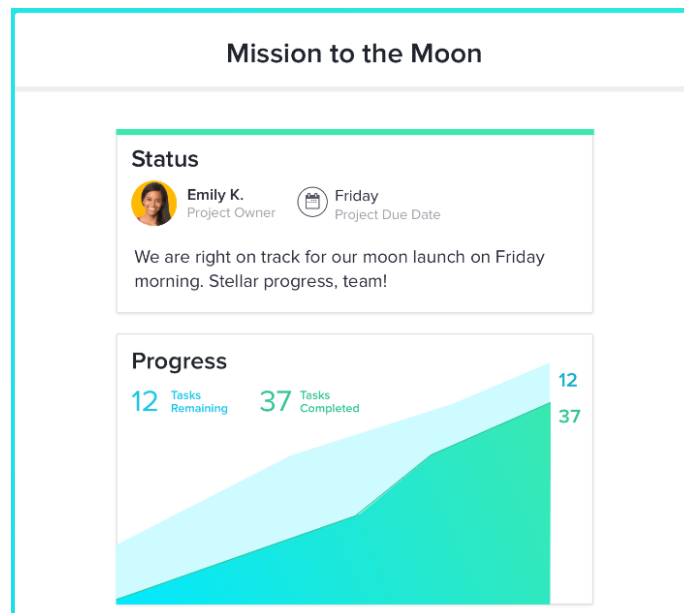


Figure 2.3: Asana [21]

2.3 Google Inbox

Google Inbox (on the figure 2.2) is an application built by Google on top of Gmail. It adds different view upon the user's emails and turns them into tasks with advanced sorting and grouping methods. Although the direct integration may seem promising, the whole application feels like a reskinned email client.

2.4 Asana

Asana (on the figure 2.3) states in its motto: "Responsibilities and next steps are clear, so you can shoot for the moon and get there." [21] In the field of task

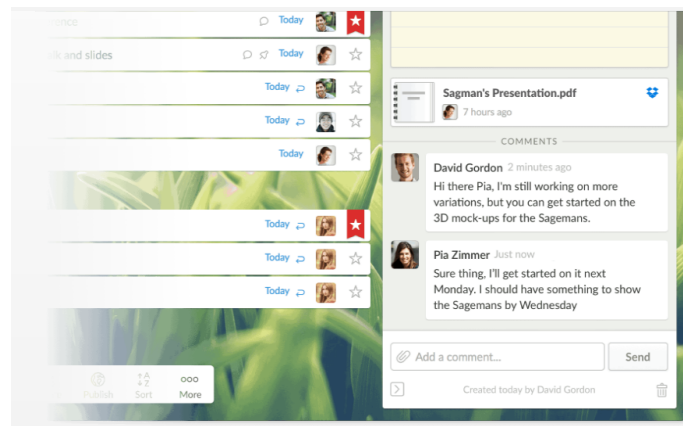


Figure 2.4: Wunderlist [22]

management and collaboration apps, Asana seems like one of the real state-of-the-art representatives. Rich in features, it leans towards being a workplace issue tracker rather than a personal application.

2.5 Wunderlist

Wunderlist (on the figure 2.4) started as a personal task tracking application. Now acquired by Microsoft, Wunderlist added support for collaboration, but lacks the clear distribution of responsibilities.

2.6 Chapter conclusion

The selected examples are just a tip of an iceberg and task management applications are plentiful. But after doing the research, there are two things to note:

- As the supply of both the free and commercial applications is high, one can assume the demand is matching.
- Although the ψ -theory directly describes the mechanics of commitment in the enterprise domain and is a great basis for the task management, there are no major applications utilising it.

This may present an opportunity for the development of such application.

Part II

Practical part

Analysis of requirements

3.1 Chapter introduction

This chapter analyses the presented theory and explores requirements needed for the application of the theory. The aim of the thesis is to apply the transaction axiom, but the remaining three axioms are an integral part of the ψ -theory and their impact is analysed as well. The analysis method is to focus on the important concepts of each axiom and extract their importance for the functionality of the application. The key thing to note is the scope of the analysis - it aligns with the intention of creating a simple task management application for everyday use with a low learning curve. The requirements are brought about to support this goal.

3.2 Operation axiom analysis

The first axiom declares basic entities used by the other axioms. Important concepts of this axiom are:

- Actors
- Coordination world, coordination acts, coordination facts
- Production world, production acts, production facts
- Responsibility, authority, competence

3.2.1 Actors

Actors are the core part of the ψ -theory. Every coordination act requires exactly two of them, the performer and the addressee. The theory states that actors are subjects taking on the actor role belonging to the currently performed transaction. By becoming the actor, the subject binds himself with

the responsibility of carrying out the actor role. All currently active roles of a subject represent the acts he needs to deal with – the *agendum*.

Requirements

- The user **can take an actor role**, thus commit himself to perform an act.
- The user **can see his agendum**. This allows user to carry out his acts.

3.2.2 C-world

The C-world contains all the C-acts and facts. They always concern a P-fact in the form of a proposition part and an intention regarding the proposition. The C-acts are carried out by two actors.

Requirements

- The user **is aware of the C-act he is performing**, having the required knowledge about the C-act and corresponding P-fact. He knows what his role in the act is.
- The user **can perform all the active acts on his agendum**. Active act is an act not waiting on another fact due to the composition axiom.

3.2.3 P-world

In the P-world, the performer of the P-act creates a new P-fact. Every C-act and fact is tied to a P-fact, making the P-world the core part of every transaction and transitively the core part of the application. One must realise that while the application is P-world centred, the P-world itself exists outside of the application and P-facts are only referenced in the transactions.

Requirements

- The user has the **ability to reference (describe) the P-fact of the transaction**. This enables the specification of every transaction.

3.2.4 Responsibility, authority, competence

These conditions are required of every actor during the performance of an act. The *responsibility* is a social construct rather than being something quantifiable and poses no functional requirement. The other two conditions may be theoretically transformed into functional constraints, but this inherently strays from the goal of simplicity. All of them should rest upon the user's judgement.

Requirements

- The user decides whether **the responsibility, authority or competence conditions are met.**
- The user can **challenge the responsibility, authority or competence of the other actor during a transaction.**

3.3 Transaction axiom analysis

The transaction axiom orchestrates communication between two actors, the initiator and the executor. It presents a pattern of this orchestration with three varying levels of complexity – the application should implement the most advanced of the patterns as it covers all of the interaction possibilities for the actors. Important concepts of this axiom are:

- External initiation
- Movement through the pattern
- Production
- Revocation
- Tacit acts

3.3.1 External initiation

This is the common form of the initiation in the scope of the application. The initiator desires something to become existent therefore he chooses an appropriate executor and creates a request. There is one major issue to be addressed: transactions classified by the author of the thesis as the ‘*ad hoc transactions*’. In the enterprise ontology, each transaction is of some transaction kind (depending on the corresponding product kind). This is manageable, because the EO maps the domain of one particular enterprise at a time, so the set of product and transaction kinds is finite. The application that is developed in this thesis behaves differently - for the sheer number of production kinds it can cover is infinite. It must be able to handle the ad hoc transactions by allowing the specification of product during runtime, but provide means to predefine product kinds as well.

Requirements

- The user can **create ad hoc transaction by requiring a newly described P-fact.** He is responsible for choosing the right executor for this transaction (but in the case of misjudging, the executor can decline the request).

- The user can **offer transactions based on predefined product kinds**.
- The user can **request a predefined transaction from the offering actor**.

3.3.2 Movement through the pattern

The most important part of the application is realising the transaction pattern. The theory provides rules on what can happen depending on the set of existing C-facts in the transaction.

Requirements

- The application **provides the actors with eligible actions based on the transaction state**.
- The application **handles the logic after the user performs actions**.

3.3.3 Production

As stated before, the production part of the pattern is out of the application scope.

Requirements

- The application **takes into account the external production of P-facts**.

3.3.4 Revocation

Revocation patterns are an important part of the complete transaction pattern.

Requirements

- The user **can ask for a revocation of any finished C-fact**. The ruling of the theory applies with all its implications (e.g. returning to a different transaction state in the case of successful revoke).

3.3.5 Tacit acts

The transaction axiom also contains the tacit acts. These are acts performed automatically, which means they are essentially skipped on the process level (but the actor is still deemed responsible for them and they are considered to be performed, resulting in the C-fact creation). The author of the thesis decided to omit them from the functional requirements for two main reasons:

1. They hold no value for the ad hoc transactions, because one must know the consequences of performing an act tacitly before it is skipped. This knowledge requires the understanding of the transaction kind and ad hoc transactions don't have specific kind before they are created.
2. The setting of tacit actions expects the user to be familiar with the axioms, but the application is meant for users with no prior knowledge of the ψ -theory.

Tacit acts may be added later in the future when the user experience with the theory and patterns grows.

3.4 Composition axiom analysis

The composition axiom allows the linking of transactions and forming more complex processes. The main concepts of the axiom are:

- The waiting link
- The initiation link
- Self-activated initiation

3.4.1 The waiting link

The waiting link enables blocking of C-acts before a particular fact is created in another transaction.

Requirements

- The user **can link a transaction to another one by adding an association**. The association is a non-blocking informational link.
- The user **can link a transaction to another one by adding a waiting link**. The waiting link is blocking.

3.4.2 The initiation link

The initiation link holds value for prepared transactions as it allows the creation of simple processes.

Requirements

- The user **can link a prepared transaction to another one by adding an initiation link**. This will start the linked transaction when the parent transaction reaches specified state.

3.4.3 Self-activated initiation

Self activated initiation adds the capacity to build periodic and control activities. While this certainly has a business value for enterprises, it may become a tool that clogs the application (the author wants every promise in the application to hold meaning and users tend to lose focus when overwhelmed with many recurring tasks), so it is omitted for now.

3.5 Distinction axiom analysis

While the distinction axiom proves useful for the enterprise ontology and the essential model of organisation, the application strives to cover all the types of transactions and not limit itself to the ontological ones. Thus the claims are the only relevant concept.

Requirements

- The user can **challenge the claim to truth, justice or sincerity of the other actor during a transaction**. The user has the responsibility to decide whether the claims were upheld and ultimately it is his decision on how to proceed with the transaction.

3.6 Functional requirements not tied to the theory

There are some functional requirements not directly tied to the theory, however, important for the operation of the application itself.

Requirements

- The user **can create virtual actors**. They are used as a substitute to the people outside of the application. The user can then track transactions without the need of the other party to join the application.
- The application **will manage user identity**. Every participating actor has an account which stores his information and agenda.

3.7 Non-functional requirements analysis

Apart from the functional requirements stemming from the theory, there are non-functional prerequisites that ensure good practices and usability of the application.

Requirements

- The application **is available to a wide range of platforms.**
- The application **is simple to use.**
- The application **can be easily extended or modified.**

3.7.1 Academic environment

The following non-functional requirements spring from the fact the thesis is created in an academic environment.

Requirements

- The application **is available as an open source.** The aim is to present the ψ -theory to a wide range of users so the base version should not be proprietary.
- The application **can be used and extended as a part of future academic works.** The scientific benefits of the thesis may be leveraged as well.

3.8 Requirements summary

3.8.1 Functional requirements

The table 3.1 contains the summary of the basic functional requirements gathered from the analysis of the theory.

3.8.2 Non-functional requirements

The table 3.2 lists the non-functional requirements based on good practices and academic environment.

3.9 Chapter conclusion

The requirements defined in this chapter cover the theory in regard of the thesis's goal. They serve as a basis for the application design and a benchmark of the application capabilities.

3. ANALYSIS OF REQUIREMENTS

Table 3.1: Functional requirements

<i>Code</i>	<i>Short description of the requirement</i>
FR-1	The user can take an actor role
FR-2	The user can see his agendum
FR-3	The user is aware of the C-act he is performing
FR-4	The user can perform all the active acts on his agendum
FR-5	The user has the ability to reference (describe) the P-fact of the transaction
FR-6	The user decides whether the responsibility, authority or competence conditions are met
FR-7	The user can challenge the responsibility, authority or competence of the other actor during a transaction
FR-8	The user can create ad hoc transaction by requiring a newly described P-fact
FR-9	The user can offer transactions based on predefined product kinds
FR-10	The user can request a predefined transaction from the offering actor
FR-11	The application provides the actors with eligible actions based on the transaction state
FR-12	The application handles the logic after the user performs actions
FR-13	The application takes into account the external production of P-facts
FR-14	The user can ask for a revocation of any finished C-fact
FR-15	The user can link a transaction to another one by adding an association
FR-16	The user can link a transaction to another one by adding a waiting link
FR-17	The user can link a prepared transaction to another one by adding an initiation link
FR-18	The user can challenge the claim to truth, justice or sincerity of the other actor during a transaction
FR-19	The user can create virtual actors
FR-20	The application will manage user identity

Table 3.2: Non-functional requirements

<i>Code</i>	<i>Short description of the requirement</i>
NFR-1	The application is available to a wide range of platforms.
NFR-2	The application is simple to use.
NFR-3	The application can be easily extended or modified.
NFR-4	The application is available as an open source.
NFR-5	The application can be used and extended as a part of future academic works.

Used technology

4.1 Chapter introduction

The chapter overviews all major technologies and external libraries used in development in the application prototype. Each technology is briefly introduced and described, with some non-obvious⁷ decisions explained. The importance of the chapter lies in the fact that the technologies shape the architecture of the app and directly or indirectly influence choices made during design and implementation time.

4.2 .NET

The Microsoft .NET platform [23] was created as a successor to the COM (*Component object model*) programming model in 2002. It was intended to provide a more powerful, flexible, and simpler programming mode than COM [24]. The .NET evolved from the Windows-only tooling to a platform capable of running on many operational system families, including the major Unix/Linux distributions and Mac OS X. It supports many programming languages, such as C#, Visual Basic, F#.

The .NET platform consists of three basic building blocks:

- **CLR:** Common Language Runtime is the runtime environment for all the .NET objects. It manages memory, hosting, threading, security, and serves various other purposes between the hosting system and .NET based application.
- **CTS:** Common Type System is a specification that describes all the data types and constructs supported by the runtime.

⁷The author chose the .NET based C# language implementation as advised by the thesis supervisor Ing. Robert Pergl, PhD., and the consultant Ing. Marek Skotnica.

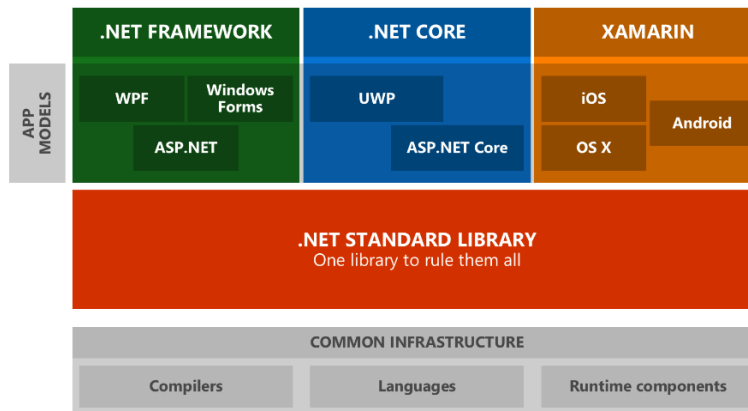


Figure 4.1: .NET Standard library [25]

- **CLS:** Common Language Specification is a set of features that all the .NET languages can agree upon. If an application is compliant with CLS, it can be consumed by other .NET languages.

4.2.1 .NET Core

.NET Core [26] is a new branch of .NET platform. Developed by the .NET Foundation (“An independent organization to foster open development and collaboration around the .NET ecosystem. It serves as a forum for community and commercial developers alike to broaden and strengthen the future of the .NET ecosystem by promoting openness and community participation to encourage innovation.” [27]), it is an open-source implementation of the .NET platform. As it conforms to the .NET Standard (illustrated on the figure 4.1), it is built upon the common infrastructure as the .NET framework. The main difference is in its openness and native support of cross-platform programming.

.NET Core version 1.0 was released on June 27, 2016. It is not a mature, rigid platform, but rather a constantly developing ecosystem – with a strong advantage of being based on the .NET platform and supported by major companies (for example Microsoft and Red Hat). The key reasons why the author chose the .NET Core as a main technology for the development of the application are:

- It is open-source which aligns with the idea of the app being available to broad public.
- It has extensive tooling for the development of standalone web applications.

- It is modular and new functionalities can be added quickly.
- It is new and emerging – as such it may not be ready yet for a corporate environment, but it is suited well for smaller apps exploring modern technologies.

4.3 C#

C# is a programming language, member of the C family (others being, for example, C, Objective C, C++, Java). The C# was created as a part of the first version of the .NET platform and has evolved substantially since then (current major version is 6.0, released in July 2015, version 7.0 is in development). It is a hybrid of other languages and incorporates concepts found in different programming paradigms - i.e. features found in functional languages (such as LISP or Haskell). It has a clean syntax, is simple, but yet powerful and flexible. According to [24], the most important part to understand about the language is that “it can produce code that can execute only within the .NET runtime [...]. Officially speaking, the term used to describe the code targeting the .NET runtime is *managed code*. The binary unit that contains the managed code is termed an *assembly*. Conversely, code that cannot be directly hosted by the .NET runtime is termed *unmanaged code*.”

4.4 ASP.NET Core

The description of the ASP.NET Core from the official announcement states the following: “ASP.NET Core is an open source web framework for building modern web applications that can be developed and run on Windows, Linux and the Mac. It includes the MVC framework, which now combines the features of MVC and Web API into a single web programming framework. ASP.NET Core is built on the .NET Core runtime, but it can also be run on the full .NET Framework for maximum compatibility.” [28] The framework provides middleware needed to build up web-based applications, such as hosting, configuration, identity management (both authentication and authorisation) and telemetry.

4.4.1 ASP.NET MVC

ASP.NET MVC, namely the most recent ASP.NET MVC 6, is a presentation layer framework based on .NET platform. As of the Core version, it was merged into the ASP.NET Core framework and is now an integral part of it. The MVC stands for the Model-View-Controller pattern (see the figure 4.2). This pattern separates the business logic (found in the Model part of the pattern) from the presentation logic (the View part). The Controller serves

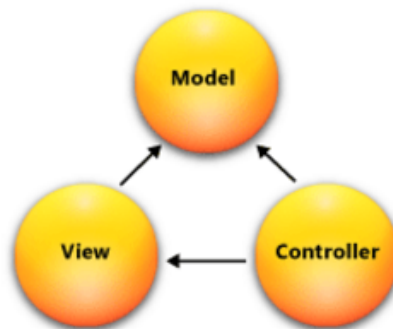


Figure 4.2: The MVC pattern [29]

as an entry point for application users, providing them with requested views of data and delegating inputs further into the application core. In ASP.NET Core, another level of abstraction is present in the form of a View models, resulting in the following four parts of the ASP.NET Core application:

1. **Model** is the domain logic of the application. It stays on the backend side and is not exposed to users.
2. **View model** represents one or more domain models, specifically their attributes that are of interest for users. The view model does not usually contain business logic apart from validation logic.
3. **Controller** handles user requests. Based on requested resource, it invokes application logic and maps results to a view model. It does not contain any business logic itself, with the exception of validation.
4. **View** is a web page based on one view model. It presents the view model to user and routes any interaction to controllers. Again, it does not contain any logic apart from the front-side validation (which is inherited from the view model)

The addition of view models allows proper separation of concerns – the domain model does not depend on the presentation layer of ASP.NET MVC. User interaction is handled through the views, view models and controllers and the logic is routed to the business layer of the application.

4.4.2 ASP.NET Identity

ASP.NET Identity provides an out-of-the-box solution for managing user access to web applications. It brings the necessary database model and basic views and controllers to enable user management (i.e. registrations, logins), authorisation environment (restricting access to parts of the application based

on roles) and usage of third party authentication services (e.g. the authentication from Google or Facebook).

4.4.3 Razor markup language

ASP.NET Core comes with its own markup language for the *Views* called Razor. From W3schools: “Razor is a markup syntax that lets you embed server-based code (Visual Basic and C#) into web pages.

Server-based code can create dynamic web content on the fly, while a web page is written to the browser. When a web page is called, the server executes the server-based code inside the page before it returns the page to the browser. By running on the server, the code can perform complex tasks, like accessing databases.” [30]

Razor is a powerful tool used to minimise code repetition when developing the *Views*.

4.5 LINQ

LINQ stands for ‘Language Integrated Query’. It is an extension of the C# language, natively available within the .NET platform. LINQ brings the advantages of functional programming to the objective-oriented C# easing many common tasks and reducing the ‘boilerplate’ code⁸. Mukherjee in [31] makes case for LINQ by presenting the following five benefits of using functional programming⁹:

- **“Composability** lets you create solutions for complex problems easily. It is based on the divide and rule principle, reducing complexity of the code.
- **Lazy evaluation** is a concept that provides the results of queries only when you need them. LINQ allows deferring execution, for example of SQL queries.
- **Immutability** lets you write code that is free of side effects.
- **Parallelizable** – functional programs are easier to parallelize than their imperative counterparts because most functional programs are side-effect free (immutable) by design.
- **Declarative** – declarative programming helps you write very expressive code, so that code readability improves. Declarative programming often also lets you get more done with less code.”

⁸The boilerplate is a part of code that needs to be repeated without many changes and is usually difficult or impossible to omit.

⁹The quote is rephrased.

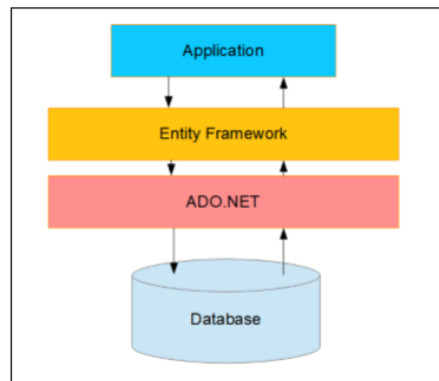


Figure 4.3: The Entity Framework architecture [32]

4.6 Entity Framework

Singh in [32] provides the following description for the Entity Framework (EF): “Entity Framework is an Object Relational Mapper (ORM) from Microsoft that lets the application’s developers work with relational data as business models. It eliminates the need for most of the plumbing code that developers write (while using ADO.NET) for data access. Entity Framework provides a comprehensive, model-based system that makes the creation of a data access layer very easy for the developers by freeing them from writing similar data access code for all the domain models.”

The Entity Framework’s architecture is depicted on figure 4.3. It shows it as an abstraction layer upon the ADO.NET database access. Using EF enables programmers to employ one of three development styles when creating the data layer and modelling the database:

- **The Database First approach** – used when developing a database-centric application, or already having a legacy database, or the database design is done by another team.
- **The Code First approach** – helpful when the database is used just as a persistence mechanism to a domain and contains no logic itself.
- **The Model First approach** – the least used approach utilising Visual Entity Designer tool to model and create the database.

The EF that is a part of the .NET Core platform is called Entity Framework Core. It is still under development and lacks some functionalities the version for standard .NET has.

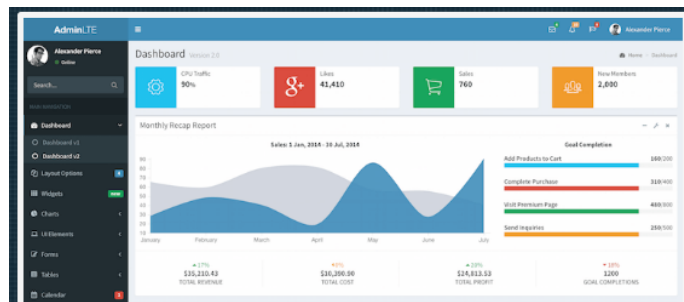


Figure 4.4: The AdminLTE Dashboard [34]

4.7 Web-specific tools and libraries

The application makes use of third-party javascript and CSS libraries mentioned below to support the presentation layer.

4.7.1 jQuery

“jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. With a combination of versatility and extensibility, jQuery has changed the way that millions of people write JavaScript.” [33] jQuery enables to use plugins to further enhance its functionalities.

jQuery validation is a plugin that adds client-side validation of user inputs.

jQuery autocomplete is a plugin that enriches text inputs by adding the option of asynchronous API calling and presenting the results to the user.

4.7.2 Bootstrap

Bootstrap¹⁰ is a library containing HTML, CSS and javascript code. It focuses the responsiveness of the web application – scaling, ordering and appearance of elements based on the device the web page is viewed on. Responsive web pages bring comfort to users and allow developers to have a single code base for wide range of possible display devices.

4.7.3 AdminLTE

AdminLTE¹¹ is a theme built upon Bootstrap. It adds new responsive widgets and tools to build a dashboard-styled responsive web pages. Figure 4.4 is an

¹⁰Bootstrap is licensed under MIT license.

¹¹AdminLTE is licensed under MIT license.

illustrative example of the styling and elements provided by the AdminLTE library.

4.8 Dependency management tools

These tools enable the distribution of applications without the external dependency libraries. Those dependencies are restored upon migrating the project to a new place.

4.8.1 Npm

The Node Package Manager (npm) is a part of the Visual Studio IDE. It provides the majority of package dependencies for projects.

4.8.2 Bower

“Bower is a package manager designed for web libraries. If multiple packages depend on a package - jQuery for example - Bower will download jQuery just once. This is known as a flat dependency graph and it helps reduce page load.” [35]

Application design

5.1 Vocabulary specification

The terminology of the ψ -theory can be slightly overwhelming for the general population. The main weakness as perceived by the author of the thesis is the word transaction. While it is a common term for people with knowledge and background in information technology, it may be too abstract for the target audience of the application. The author proposes usage of three expressions depending on what state the transaction is in:

1. **A promise** is a default naming convention used for the transactions in the application. It was chosen because it implies certain ownership of responsibility. Also the transaction becomes ‘official’ only after the executor promises to deliver the P-fact. A *promise* is always created from a *request*.
2. **A request** is a transaction before the promise state. The initiator has specified the desired P-fact and requested it of the executor, but may still be declined. A *request* can be created as an ad hoc transaction or from an *offer*.
3. **An offer** is a prepared transaction with specified product kind. An offer can be turned into a *request* (and then into a *promise*).

For the *state* C-act, the author uses the term ‘Done’¹² as in ‘Mark the *promise* as done’. It is a bit more natural to non-English speakers and aligns with the aim of the application to be easily accessible for everyone.

¹²On the front-end. In the specification the term *state* is still used.

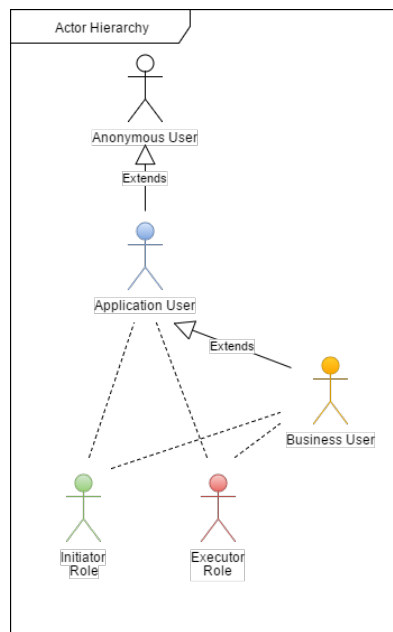


Figure 5.1: Actor hierarchy

5.2 Use case diagrams

The application use cases are defined in this section. It starts with presenting the actors (users) eligible for the application usage. Then the use case diagrams per every type of the actor are shown and described. Use case diagrams are based on the functional requirements declared in the chapter 3.

5.2.1 Actor hierarchy

On the figure 5.1, the following user roles are defined:

- **Anonymous user** – a basic user who has not yet registered or logged in. This user's use cases are the most limited.
- **Application user** – a common user. One becomes an application user by registering and logging in.
- **Business user** – a special sort of user with access to the predefined transaction creation. Extends the application user.
- **Initiator role** – application user interacting with a transaction where he assumed the *initiator* role.
- **Executor role** – application user interacting with a transaction where he assumed the *executor* role.

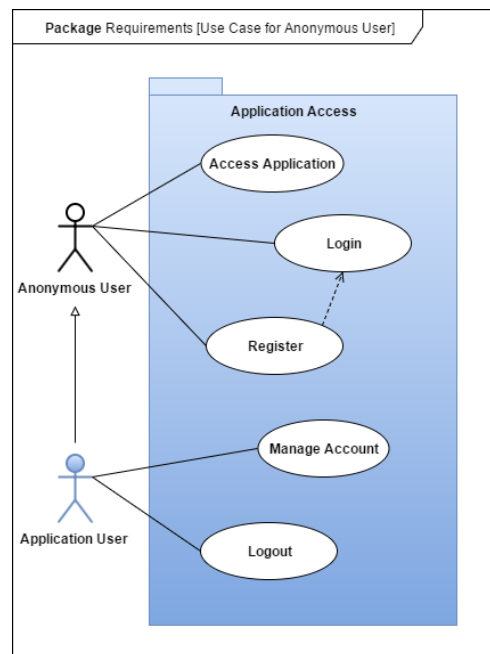


Figure 5.2: Anonymous user use cases

5.2.2 Anonymous user use cases

The figure 5.2 depicts the use cases for the anonymous and freshly logged in users.

- **Access application** – any user can reach the front page of the application. The front page enables register of new users and logging in for existing users.
- **Register as a new user** – an anonymous user can fill the registration form, which creates an account for him. He is then automatically logged in on the newly created account.
- **Login** – registered, unauthenticated anonymous users can use their account's credential to authenticate and gain access to the application and their data.
- **Logout** – a logged in application user can use the logout option to leave the application.
- **Manage account** – an application user can change settings on his account.

5.2.3 Application user use cases

Application user uses cases are illustrated on the figures 5.3 and 5.4. The first figure shows use cases regarding the promises, the second contact handling. The promise use cases are divided depending on the role required (common tasks are performed by an application user, role-specific are marked by the corresponding role).

5.2.3.1 Common promise use cases

- **View promises** – lists all the promises currently available to the actor. This use case represents the agendum part of the ψ -theory.
- **Filter promises** – allows the filtering of promises. For example filters out the promises in terminal states, so the user can focus on the present agendum.
- **View details of a promise** – displays the promise. If the current status of the promise allows an action to be performed by the viewing actor, it is presented to the actor and he can take the action.
- **Browse the promise hierarchy** – displays promises related to the currently viewed one.
- **Fill in for the other user** – if the other actor of the promise is marked as virtual (created by the current user), the user can perform his actions as well.
- **Link promises** – the user can add links between promises such as association links and waiting links.
- **Revoke previous step** – the user can revoke performed steps as allowed by the complete transaction pattern.

5.2.3.2 Initiator role use cases

- **Request something** – the initiator creates a new request.
- **Provide clarification** – the initiator can react in a discussion regarding promise, generally to clarify his intentions and desires.
- **Accept finished promise** – the initiator can accept a P-fact stated by the executor.

5.2.3.3 Executor role use cases

- **Promise something** – the executor can react to a request by promising it, thus creating a promise.
- **Request clarification** – the executor can react by further discussing the promise, generally to require clarification of initiator’s intentions and desires.
- **State the promise is done** – the executor states the P-act is completed and P-fact is created.

Common contact use cases

- **View contacts** – user can view his contacts and contact groups.
- **Manage contacts** – user can add or remove contacts from contact groups.
- **Search for user** – user can add a new contact by searching for an existing user in the application.
- **Create virtual contact** – user can add a virtual contact for whom he will manage actions. Virtual contacts are private and can be seen only by their creator.

5.2.4 Business user use cases

User classified as a business user can create *offers*, which are transactions with predefined product kind. The use case diagram is depicted on figure 5.5

- **Create predefined promise** – business user can create an offer by declaring its production kind.
- **View predefined promises** – business user can list all his offers.
- **Manage predefined promises** – business user can manage his offers, changing their definitions.
- **Link with other predefined promises** – business user can add links between offers such as association links and initiation links.

5.2.5 Basic transaction use case

The last use case illustrated on 5.6 displays the interaction between the two actor roles. It is a basic use case for the complete transaction pattern, listing the action possibilities without showing any constraints.

The figure 5.6 shows the basic transaction use case. It is derived from the basic transaction pattern

5. APPLICATION DESIGN

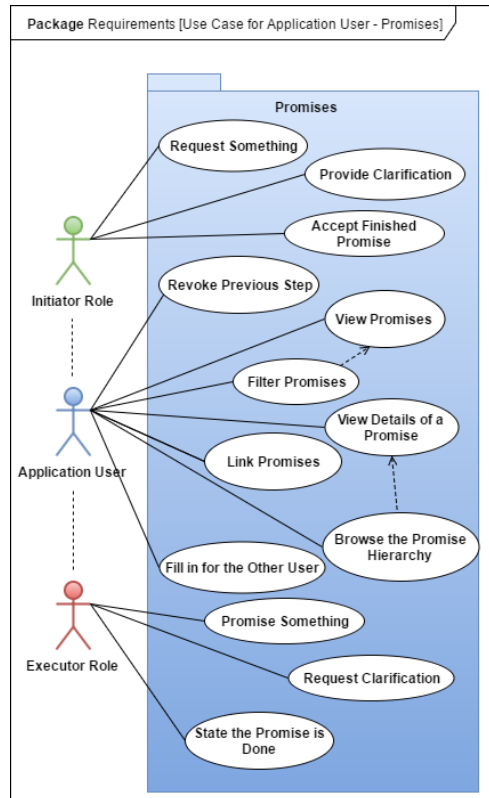


Figure 5.3: Application user use cases - promises

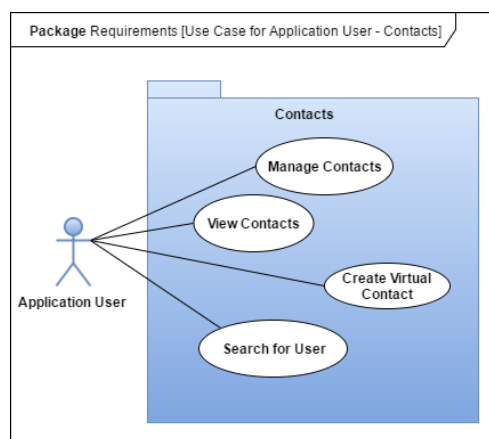


Figure 5.4: Application user use cases - contacts

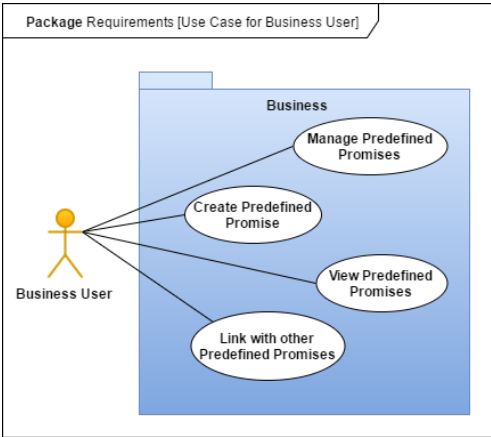


Figure 5.5: Business user use cases

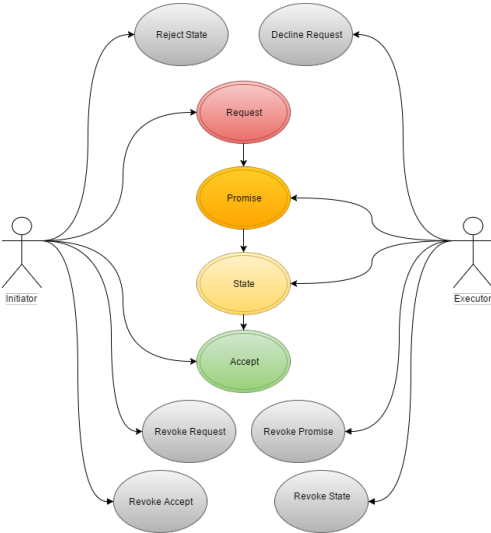


Figure 5.6: Basic transaction use case

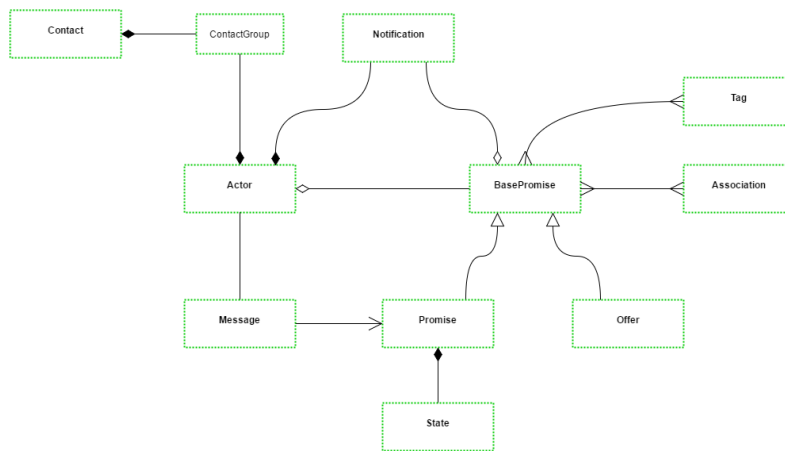


Figure 5.7: Basic conceptual model

5.3 Conceptual model

The bare conceptual model on figure 5.7 shows the basic classes needed for the operation of the application. Their brief descriptions follow:

- *BasePromise* is an abstract class from which promises and offers inherit.
- *Offer* is a class representing offers - prepared transactions.
- *Promise* is a class representing instantiated transaction.
- *State* is a status (corresponding to the last C-fact) of a transaction.
- *Association* represents links - initialisation, waiting, and non-blocking association.
- *Tag* A nice-to-have feature allowing the classification of promises based on added labels.
- *Message* A message, a part of a discussion about a promise.
- *Notification* A notification class adding various informative messages for users.
- *Actor* The application user.
- *ContactGroup* A class aggregating contacts.
- *Contact* A class representing a contact.

5.4 Required views

The application is web based and needs a specification of views that it consists of. From the theory, requirements and the conceptual model, the following views are considered core:

5.4.1 User management

This view contains all the interaction logic needed to manage user accounts. The operations are delegated to following subviews:

- **Log in view** – allows the users to log in
- **Register view** – allows the registering of new users
- **Manage account view** – for settings
- **Log out** – Ends the current user's session

5.4.2 Promises

This view contains all the transactions of the current actor - his agendum.

- **View all** – shows all the promises.
- **View specific** – shows the detail of a promise along with actions, should the actor be eligible for any
- **Create new** – creates a new request

5.4.3 Offers

This view contains all the prepared transaction kinds of the business user.

- **View all** – shows all the offers current user has
- **Manage** – allows the management of offers
- **Create new** – creates a new offer
- **View offers for current user** – when accessed through the user's contact list, shows all the offers selected contact has for the current user

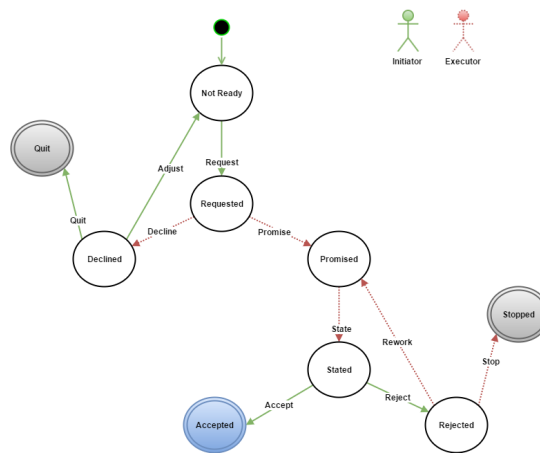


Figure 5.8: Basic state machine

5.4.4 Contacts

This view contains all the user’s contacts and groups.

- **View all** – shows all contacts and groups
- **Add new contact** – adds a new contact
- **Add new group** – adds a new group
- **View offers of the selected contact** – navigates to the *View offers for current user* subview

5.5 State machine

The state machine is the core application logic. It is derived from the transaction pattern and handles the flow of C-acts and facts. The basic state machine on figure 5.8 shows the paths reachable via normal execution (= without the revocations).

The figures 5.9 and 5.10 present the revocation patterns from the scope of the initiator, respectively executor role. The non-standard notation in the diagrams adds the “return to previous state” element. This element keeps the diagram compact and consolidates multiple revoke states.

On figure 5.11 the complete state machine is viewed. This is the state machine that needs to be implemented for the application to work properly.

The last figure 5.12 depicts the extension possibility for the state machine. It adds saving, which is not something derived from the theory but naturally a useful part of client applications.

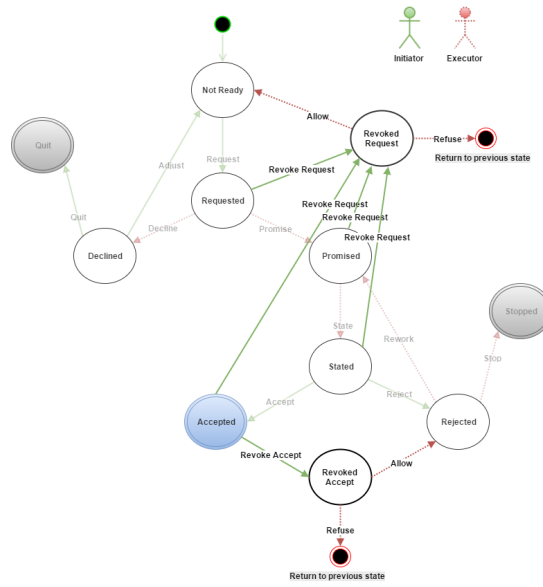


Figure 5.9: Initiator revokes state machine

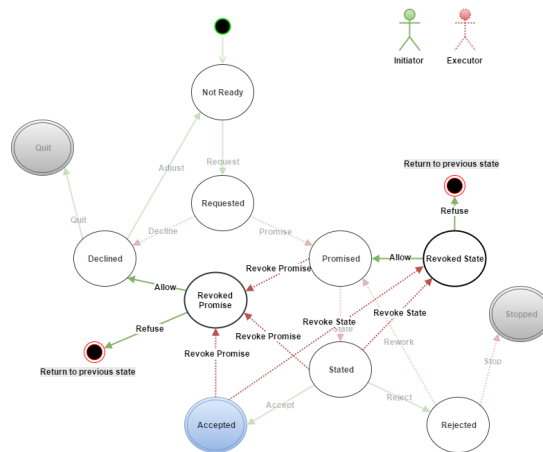


Figure 5.10: Executor revokes state machine

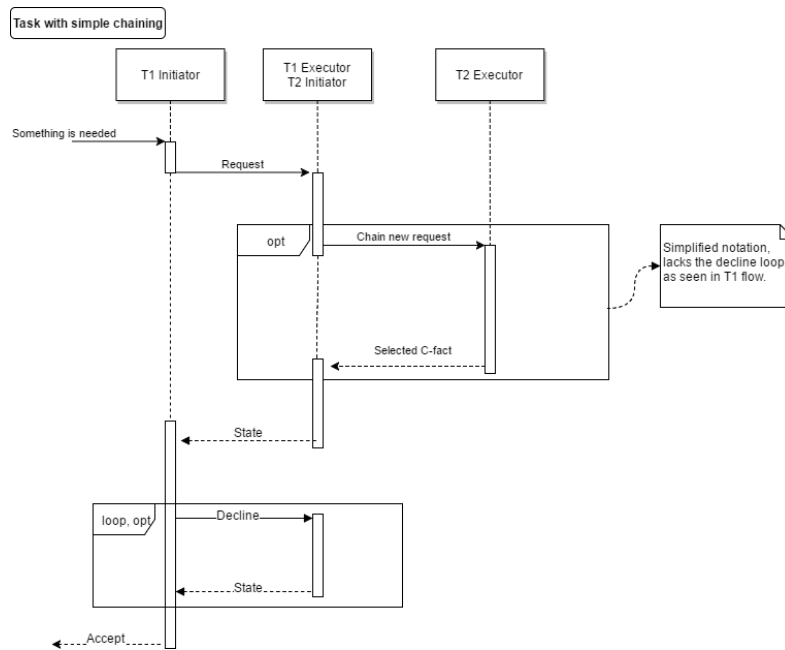


Figure 5.13: Simple chaining of transactions

5.6 Transaction chaining

The second concept apart from the base state machine is the requirement of transaction chaining as part of the composition axiom. The chaining is modelled by links, which modify the process flow - a conceptual example is shown on figure 5.13.

5.7 Application prototype

This section outlines the architecture of the application prototype. Because the application is under constant development even out of the thesis's scope, only key features are shown.

5.7.1 Architecture overview

The author intended to use the onion architecture (see the figure 5.14) for the application to neatly follow the separation of concerns principle. Currently the majority of the application conforms to the onion architecture (the components fit in layers), but the new .NET Core identity proved to be difficult to separate from the Entity framework and data layer. For this reason the prototype is nested in one project.

Onion Architecture

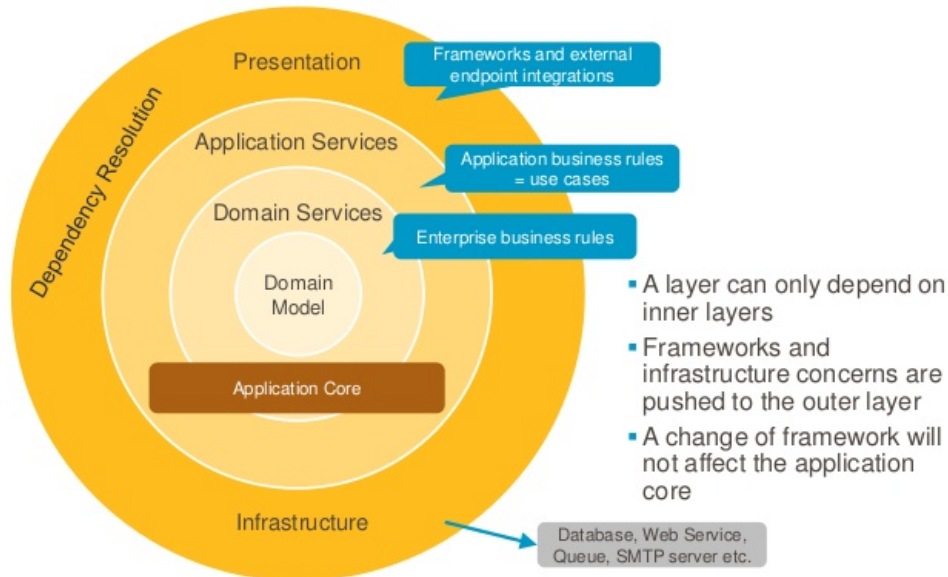


Figure 5.14: Onion architecture [36]

5.7.2 Dependency injection

The integral part of the architecture is the usage of dependency injection. It provides an abstraction of the data layer - the domain exposes interfaces of the persistence services that the data layer must implement. Those services are then registered from the assembly during runtime (with the option of controlling their lifespan) and injected into constructors of other classes.

5.7.3 Database

Database schema (shown on the figure 5.15) was created using the Entity Framework Core code-first approach. This allows the application designer to focus his mind on the domain object model and then let the persistence layer to generate itself. Although this may not result in the optimal schema, the speed of this method proves invaluable for a fast application prototyping. The schema becomes basically a 1:1 model, with a few exceptions – from which the most notable is the implementation of inheritance. As can be seen on the schema, the abstract class of `BasePromise` and its children `Promise` and `Offer` are merged into one table with the addition of a discriminator column. Currently, this is the only supported way of modelling inheritance (older Entity Framework versions allowed two more methods, EFCore is limited to just this one)

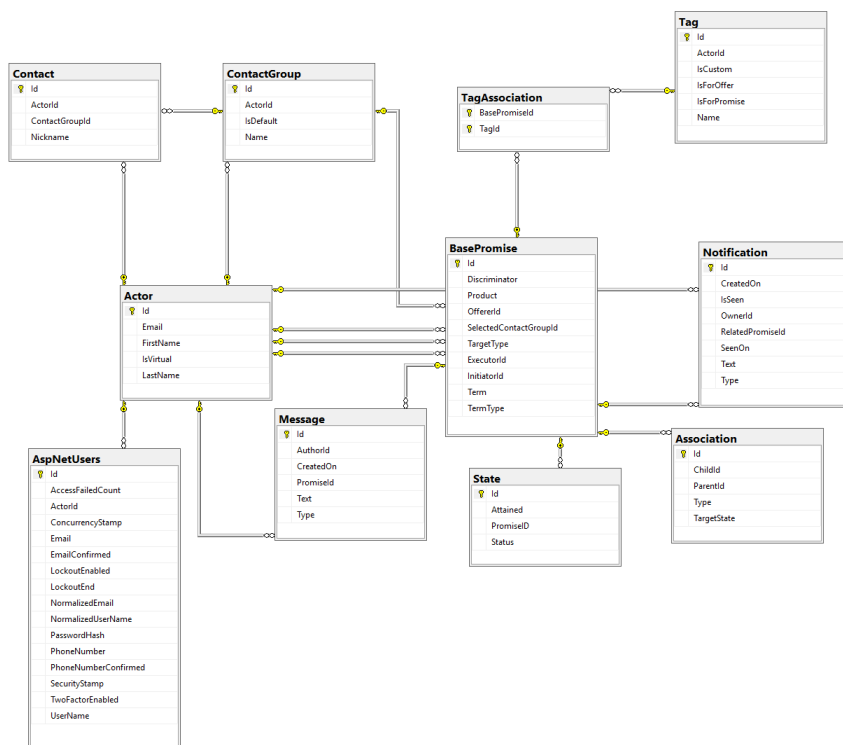


Figure 5.15: Database schema (with ASP.NET Identity tables omitted)

Test scenarios

6.1 Chapter introduction

This chapter presents four test scenarios based on real life situations. Those scenarios were picked as a showcase of the intended scope covered by the application. The scenarios are described and then walked through from the application's point of view.

1. **Scenario 1 – the Book:** Adam is writing a diploma thesis. He lacks a certain theory book that Eve has. Adam uses the app to track his tasks, Eve is not a user.
2. **Scenario 2 – the Teacher:** Mark is teaching a course at university. His students want to create appointments with him to consult his subject.
3. **Scenario 3 – the Middleman:** Hansel needs to insure a car, so he asks Derek the lawyer for legal advice. Derek does not have time, so he delegates the request on his articulated clerks.
4. **Scenario 4 – the Business:** George needs to borrow money. Harry owns a fair money lending company.

6.1.1 Scenario 1 – the Book

In the most basic scenario, the user Adam wants to use the app to track his responsibilities, but the other person of the transaction is not a registered user. Adam can perform the following steps to deal with the situation:

1. *Adam creates an ad hoc request to himself with the P-fact described as 'My diploma thesis is written', with the time part of the proposition set to the deadline.* This will help him track the associated promises, should there be any.

6. TEST SCENARIOS

2. Adam realises he lacks a theory book. He knows Eve has it, but she is not willing to join the application. Adam asks for the book and Eve promises she will lend him the book next week. *Adam navigates to his contacts in the app and creates a new virtual contact named 'Eve'*. This contact will track his interactions with Eve.
3. *Adam issues a new request to the virtual Eve with P-fact of 'The book X is lent'*.
4. *Adam opens the detail of the new request and presses the promise action.* He can do so, because Eve is his virtual contact.
5. *Adam opens the main request of 'My diploma thesis is written' and adds a blocking association with the new request.*
6. Adam now has the two transactions tracked. He waits until Eve lends him the book. He has the promise displayed on his agenda, so he knows she has promised him the book.
7. When Eve lends him the book, *Adam states and accepts the second transaction.* He can now continue with the first transaction. The scenario is resolved.

6.1.2 Scenario 2 – the Teacher

In the second scenario, the common app usage is shown. This one makes use of the predefined transactions – offers.

1. Mark and his students use the app to track their promises. He wants to simplify the process of students requesting consultations.
2. *Mark navigates to the 'Contacts' part of the app.*
3. *Here, he creates a new group called 'My students' and adds his students to the group.*
4. *Then he navigates to the 'Offers' part of the app.*
5. *Mark creates a new offer. He sets the offer target to the newly created group 'My students' and the P-fact kind to 'A consultation is booked'.*
6. After saving, students whom he put in the group can display his offer and create a new request, which copies the P-fact.

6.1.3 Scenario 3 – the Middleman

The middleman scenario uses ad hoc requests to delegate production, while maintaining the responsibility.

1. *Hansel navigates to the ‘Promises’ part of the application and issues a new request to Derek, described as ‘Legal advice regarding my car insurance is given’.*
2. Derek does not have time for producing the P-fact himself. But he knows that one of his clerks could produce the P-fact.
3. *Derek promises Hansel.*
4. *Derek then creates two new requests to his articulated clerks. He copies the P-fact.*
5. *Derek associates the new requests with a promise to Hansel. He marks those associations as blocking.*
6. Lets assume both the clerks promise. Derek waits until one of them produces the legal advice and states the fact.
7. *Derek accepts the fact. He then navigates to the other request he made and revokes it.*
8. *After the revocation is allowed, Derek states the legal advice back to Hansel. For Hansel, this means that Derek is still responsible for the stated fact.*
9. *Hansel accepts, concluding the transaction.*

6.1.4 Scenario 4 – the Business

The seemingly most complicated scenario presents the application’s capability of simulating a simple process and serving as a front for businesses.

1. Because Harry is a business owner and wants to be forthcoming towards his customers, *he registers as a new user called ‘Harry’s money lending company’, HMLC for short.*
2. *He then logs in as his new user and navigates to ‘Offers’.*
3. *In ‘Offers’, he sets the entry point of the composite transaction - an offer with P-fact kind ‘Money is lent’ offered to Public.*
4. *He then creates a number of hidden offers. In them he describes what he needs before he can lend someone money (i.e. ID card, account balance sheet...).*

5. *Harry then links the entry point offer to those hidden offers using an initiation link that triggers on 'promised' state.*
6. His business is now set.
7. *George adds the HMLC user to his contacts and displays his offers. He sees the entry point offer only.*
8. *George requests the offer.*
9. *Harry promises.* This triggers the hidden offers and George receives multiple requests that he needs to fulfil.
10. George now knows what he has to do to conclude business with Harry.

6.2 Chapter summary

In the chapter the test scenarios have been shown and walked through. The scenarios cover basic usability of the app and were chosen to show the most of the features the prototype contains. The important part is that the actual wording of the scenarios does not matter – the application supports the definition of any P-fact, so the only thing that will stay the same is the pattern used by all these transactions.

Conclusion

Evaluation of functional requirements

The table 6.1 contains the fulfilment status of functional requirements. All the functional requirements have been fulfilled by the application prototype, although an extensive user testing is required to confirm the coverage is good enough to deliver pleasant experience for the users.

Evaluation of non-functional requirements

The table 6.2 contains the fulfilment status of non-functional requirements. The more complex answers are below:

1. YES. It is hosted in the cloud and working on a wide range of devices thanks to the usage of responsive design.
2. This can't be answered because no extensive user tests were performed. They are the next step.
3. YES, at "<https://github.com/r-l/ciaoapp>"

Opportunities

This section presents the opportunities for the application in the near future.

- **User tests** The application is now ready for release to a wider range of users. The user tests will tell whether the concept is well devised and understandable for people from non-IT areas.
- **Presentation to CIAO! Community** The application can be presented in the CIAO! community. The author recommends that the presentation is done after the user tests are over to gather more data and debug potential issues.

Table 6.1: Evaluation of functional requirements

<i>Code</i>	<i>Short description of the requirement</i>	<i>Fulfilled</i>
FR-1	The user can take an actor role	YES
FR-2	The user can see his agendum	YES
FR-3	The user is aware of the C-act he is performing	YES
FR-4	The user can perform all the active acts on his agendum	YES
FR-5	The user has the ability to reference (describe) the P-fact of the transaction	YES
FR-6	The user decides whether the responsibility, authority or competence conditions are met	YES
FR-7	The user can challenge the responsibility, authority or competence of the other actor during a transaction	YES
FR-8	The user can create ad hoc transaction by requiring a newly described P-fact	YES
FR-9	The user can offer transactions based on predefined product kinds	YES
FR-10	The user can request a predefined transaction from the offering actor	
FR-11	The application provides the actors with eligible actions based on the transaction state	YES
FR-12	The application handles the logic after the user performs actions	YES
FR-13	The application takes into account the external production of P-facts	YES
FR-14	The user can ask for a revocation of any finished C-fact	YES
FR-15	The user can link a transaction to another one by adding an association	YES
FR-16	The user can link a transaction to another one by adding a waiting link	YES
FR-17	The user can link a prepared transaction to another one by adding an initiation link	YES
FR-18	The user can challenge the claim to truth, justice or sincerity of the other actor during a transaction	YES
FR-19	The user can create virtual actors	YES
FR-20	The application will manage user identity	YES

Table 6.2: Non-functional requirements

<i>Code</i>	<i>Short description of the requirement</i>	<i>Fulfilled</i>
NFR-1	The application is available to a wide range of platforms.	See 1.
NFR-2	The application is simple to use.	See 2.
NFR-3	The application can be easily extended or modified.	YES
NFR-4	The application is available as an open source.	See 3.
NFR-5	The application can be used and extended as a part of future academic works.	YES

- **Implementing a consumer to the API** The API is currently consumed only by the application's frontend services. To reach maturity, a different project should use the API.
- **Exploring the business opportunities** The app is meant to be open-source and free, but an extended, more feature-rich business version could be developed.

Thesis conclusion

The aim of the thesis was to analyse the ψ -theory and create a prototype of a messaging and task management application based on the said theory. The author of the thesis thinks that although with some minor imperfections, the goal of the thesis was sufficiently reached. The weaker point of the work is the documentation of the application itself. This is because the development was strongly based on the extreme programming and other agile methods and the application changed very fast. This is at least partially balanced by the application being open-source and still in development outside of the thesis' scope, so the documentation can be added directly to the repository and kept up-to-date.

The core value of the thesis lies the theoretical part and the exploration of requirements based on the ψ -theory, together with the technological overview. It firmly sets the application foundations and can be leveraged during further development.

The application itself proves useful and it seems worth to take on the next step – presenting the application to the public. With some enhancement and extensions it could fare well against similar applications in its category.

The author thus regards the thesis and the assignment fulfilled.

Bibliography

- [1] Lofaro, R. J. Knowledge Engineering Methodology with Examples. In *Encyclopedia of Information Science and Technology, Third Edition*, 2015, pp. 4600–4607, doi:10.4018/978-1-4666-5888-2.ch451.
- [2] Hofweber, T. Logic and Ontology. In *The Stanford Encyclopedia of Philosophy*, edited by E. N. Zalta, Metaphysics Research Lab, Stanford University, fall 2014 edition, 2014.
- [3] Gruber, T. R. A Translation Approach to Portable Ontology Specifications. In *Knowledge Acquisition*, Computer Science Department, Stanford University, 1993.
- [4] Oxford Dictionaries. <https://en.oxforddictionaries.com>.
- [5] Dietz, J. L. G. Enterprise Engineering. 2016, [Cited 2017-02-12].
- [6] Dietz, J. L. G.; Hoogervorst, J. The ψ -theory v2 [online]. [Cited 2017-02-07]. Available from: <http://www.ciaonetwork.org/uploads/eewc2014/EE-theories/TEEM-5%20PSI%20v2.pdf>
- [7] Bunge, M. A. *Treatise on Basic Philosophy, vol.4*. D. Reidel Publishing Company, Dordrecht, The Netherlands, 1979, ISBN 978-3-540-33149-0.
- [8] Perinforma, A. P. *The essence of organisation*. Sapio Enterprise Engineering (www.sapio.nl), second edition, 2015, ISBN 978-90-815449-4-8.
- [9] Dietz, J. L. G.; Hoogervorst, J.; al. The Discipline of Enterprise Engineering. In *International Journal of Organisational Design and Engineering*, volume 1, 2013.
- [10] Habermas, J. *The theory of communicative action*. Beacon Press, 25 Beacon Street, Boston, Massachusetts 02108, third edition, 1981, ISBN 0-8070-1506-7.

- [11] Habermas, J. *The theory of communicative action*. Beacon Press Boston, third edition, 1981, ISBN 0-8070-1400-1.
- [12] Austin, J. L. *How to do things with words*. Oxford University Press, 1962.
- [13] Searle, J. R. *Speech Acts*. Cambridge University Press, 1969.
- [14] Dietz, J. L. G. *Enterprise Ontology - Theory and Methodology*. Springer-Verlag Berlin Heidelberg, first edition, 2006, ISBN 978-3-540-33149-0.
- [15] Dietz, J. L. G. *Red garden gnomes dont exist*. Sapio Enterprise Engineering (www.sapio.nl), third edition, 2015, ISBN 978-90-815449-2-4.
- [16] Dietz, J. L. G. DEMOSL-3. 2015, [Cited 2017-02-12]. Available from: <http://www.ee-institute.org/download.php?id=165&type=doc>
- [17] Dietz, J. L. G. Generic recurrent patterns in business processes. In *Business Process Management*, edited by W. van der Aalst; A. ter Hofstede; M. Weske, Springer-Verlag, 2003.
- [18] Dietz, J. L. G. System Ontology and its role in Software Development. In *CAiSE05 workshops, Lecture Notes in Computer Science*, Springer, 2005.
- [19] Inc., T. Trello [online]. [Cited 2017-02-15]. Available from: <https://trello.com/>
- [20] Inc., G. Google Inbox [online]. [Cited 2017-02-15]. Available from: <https://www.google.com/inbox/>
- [21] Asana. Asana [online]. [Cited 2017-02-15]. Available from: <https://asana.com/product>
- [22] Corporation, M. Wunderlist [online]. [Cited 2017-02-15]. Available from: <https://www.wunderlist.com/>
- [23] Corporation, M. .NET [online]. [Cited 2017-02-15]. Available from: <https://www.microsoft.com/net/>
- [24] Troelsen, A. *Pro C# 5.0 and the .NET 4.5 Framework*. Apress, sixth edition, 2012, ISBN 978-1-4302-4234-5.
- [25] Corporation, M. Introducing .NET standard [online]. [Cited 2017-02-15]. Available from: <https://blogs.msdn.microsoft.com/dotnet/2016/09/26/introducing-net-standard/>
- [26] Corporation, M. .NET Core [online]. [Cited 2017-02-15]. Available from: <https://www.microsoft.com/net/core/platform>
- [27] Foundation, N. About the .NET Foundation [online]. [Cited 2017-02-15]. Available from: <https://dotnetfoundation.org/about>

- [28] Corporation, M. Announcing ASP.NET Core 1.0 [online]. [Cited 2017-02-15]. Available from: <https://blogs.msdn.microsoft.com/webdev/2016/06/27/announcing-asp-net-core-1-0/>
- [29] Corporation, M. MVC Pattern [online]. [Cited 2017-02-15]. Available from: [https://msdn.microsoft.com/en-us/library/dd381412\(v=vs.108\).aspx](https://msdn.microsoft.com/en-us/library/dd381412(v=vs.108).aspx)
- [30] w3schools.com. Razor markup [online]. [Cited 2017-02-15]. Available from: https://www.w3schools.com/asp/razor_intro.asp
- [31] Mukherjee, S. *Thinking in LINQ*. Apress, 2014, ISBN 978-1-4302-6844-4.
- [32] Singh, R. R. *Mastering Entity Framework*. Packt Publishing Ltd., 2015, ISBN 978-1-78439-100-3.
- [33] jQuery Foundation, T. jQuery [online]. [Cited 2017-02-15]. Available from: <https://jquery.com/>
- [34] Studio, A. AdminLTE Dashboard [online]. [Cited 2017-02-15]. Available from: <https://almsaeedstudio.com/>
- [35] Inc., T. Bower [online]. [Cited 2017-02-15]. Available from: <https://bower.io/>
- [36] slideshare.net. Onion architecture [online]. [Cited 2017-02-15]. Available from: <https://image.slidesharecdn.com/2014march-applicationarchitecture-140322071318-phpapp02/95/application-architecture-58-638.jpg>

Acronyms

DEMO Design & Engineering Methodology for Organizations

PSI Performance in Social Interaction

EE Enterprise Engineering

EO Enterprise Ontology

C-act Coordination act

C-fact Coordination fact

P-act Production act

P-fact Production fact

rq request

pm promise

st state

ac accept

API Application interface

Contents of enclosed CD

	readme.txt	the file with CD contents description
	src	the directory of source codes
	application.....	implementation sources
	thesis.....	the directory of L ^A T _E X source codes of the thesis
	text	the thesis text directory
	DP_Lansky_Roman_2017.pdf	the thesis text in PDF format