MASTER THESIS

Towards a process-oriented ADL for specifying communication flows in BPMS application landscapes

Utrecht University

Student name: J.R. (Jeremy) Loppies, BSc.
Student number: 6028772
MSc program: Business Informatics (MBI)
Department of: Information & Computing Sciences

First supervisor: dr.ir. J.M.E.M. (Jan Martijn) van der Werf
Second supervisor: dr. L.M. (Marcela) Ruiz Carmona

BPM | COMPANY

Daily supervisor: drs. M.E. (Marco) Bussemaker

Date: April 4, 2019
Version: 1.0 [*PUBLIC*]
Status: Final

Acknowledgements

In September 2016, after I finished the bachelor of HBO-ICT (Business IT & Management) at Utrecht University of Applied Sciences, I started with the master of Business Informatics at Utrecht University. Right now, it is 2019. I am glad that I have taken this step to further increase my knowledge and develop my skills with scientific research within the field of Information Sciences.

First of all, I would like to thank Jan Martijn van der Werf for perceiving the scientific value of my research proposal when I contacted him for the first time. His input, feedback, and valuable discussions during the bi-weekly meetings helped me through my entire research. His suggestions helped me to find potential research directions when I was "stuck in the middle". I also want to thank Marcela Ruiz for her feedback and input that helped me to further elaborate my research.

I would like to thank Eelco Vissinga for giving me the opportunity to conduct my research at BPM Company. I especially want to thank Marco Bussemaker for his daily supervision, and providing me with relevant practical information from BPM Company. I really enjoyed the valuable discussions that we had during our weekly meetings. I think we both have learnt a lot from each other. A special thanks to all Pega architects who contributed to my research during the interviews and competence meetings. Without their input, I was not able to include the practitioners' perspectives to my research. I also want to thank all colleagues for the nice and pleasant working atmosphere at the office in Zeist.

Lastly, I want to thank my friends and family for their support and motivation during my research. As always, they guide me through my career with their valuable advices.

Jeremy Loppies

Culemborg, April 2019

Abstract

Business Process Management (BPM) can be supported by a BPMS. Especially, to foster the automation of end-to-end processes. Nowadays, a BPMS is often a low-code development platform. By means of the configuration of executable business process models, a process-driven application can be created. As part of this configuration, a BPMS is situated within an application landscape where it is integrated with other systems in order to collect and use the information that is required for the execution of the business processes. In this research, regarding the low-code development capabilities of a BPMS, we focus on the communication flows (data/information flows, message flows) through APIs and (web) services within the BPMS application landscape. This landscape can get quite complex when there are a lot of communication flows both within the BPMS and between the BPMS and the integrated systems. Therefore, we tend to answer the following main research question: "*What are the constituents of a process-oriented ADL for specifying communication flows in BPMS application landscapes?*" For this, we have designed an Architecture Description Language (ADL) which is tailored to the process-oriented functionality of a BPMS. Previous related research does not particularly focus on this topic. During the design process of the intended ADL, relevant literature has been combined with the perspectives of practitioners. We have acquired the practitioners' perspectives by means of both semi-structured interviews and focus groups. The design process has resulted in a process-oriented ADL that is in fact a coherent set of several models of BPMN, Architecture, and UML. The models are all related to each other in certain ways within the scope of the ADL. We have validated the practical applicability and added value of the ADL by means of a case study, including semi-structured interviews with practitioners. The case study validation results show that, regarding the specification of communication flows within a BPMS application landscape, the intended ADL is perceived as a useful and valuable means that will be easy to apply and understand within BPMS development projects.

Keywords: Architecture Description Language (ADL), Communication Flows, Business Process Management System (BPMS), Application Landscape, Traceability

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ADL	Architecture Description Language
API	Application Programming Interface
BPM	Business Process Management
BPMS	Business Process Management System

1. Introduction

In this chapter, this master thesis is introduced by means of describing the research context and scope, problem statement, relevance, and objectives. In addition, the remainder of this master thesis is given.

1.1 Research context

Nowadays, Business Process Management (BPM) is a mature discipline that is widely applied within organizations. Both practitioners and scientific researchers recognize the importance and relevance of BPM in the industry (van der Aalst, 2013). BPM can be defined as a way to map, construct and optimize business processes in a structured manner. In this way, the organizational objectives can be obtained in a better way (Weske, 2012). The focus and practical application of BPM has changed over the past decades. Though, since the construction of BPM as a concept, BPM focuses on the arrangement of work within an organization by means of the optimization of business processes. It has taken several decades before the core of BPM (process thinking) had been fully developed to today's principles of BPM. Currently, people within an organization, the process participants, are usually specialized in one particular business/discipline. As so-called specialists, they have knowledge on producing and delivering one specific product / one particular business, for example, sales and customer relationship. Before most people became specialists, they were generalists, focusing on multiple disciplines.

The concept of Business Process Reengineering (BPR) together with Workflow Management (WFM) were involved in the evolution of BPM. Moreover, emerging technologies such as Business Process Management Systems (BPMSs) have fostered the automation of end-to-end business processes. A BPMS can be defined as a software intensive system that supports the execution and monitoring of business processes by means of (partly) automating activities. In this way, the process participants have to carry out none or less activities manually. The business processes are executed by means of executable business process models, and can then be logged/monitored, analyzed, and optimized. For example, the efficiency of the business process can be analyzed based on the business performances. Usually, the business process models have been created by means of the well-known Business Process Model and Notation (BPMN) (Dumas, La Rosa, Mendling, & Reijers, 2018).

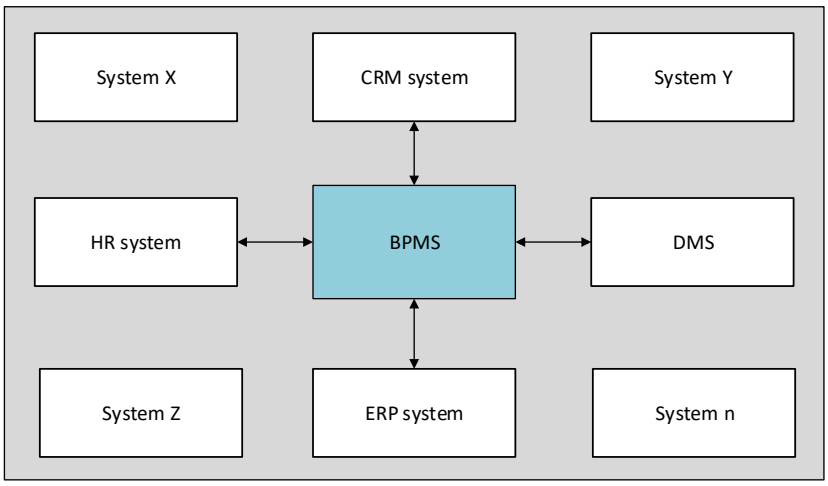


Figure 1: Simplified BPMS application landscape

A BPMS that is used within an organization belongs to the application landscape. An application landscape is "*the entirety of the business applications and their relationships to other elements, e.g. business processes in a company*" (Buckl, Matthes, & Schweda, 2009, p.1). In other words, an application landscape can be seen as a coherent set/overview of an organization's information systems and the corresponding interrelations with several business elements. So, basically, it visualizes the running environment of a certain system. In Figure 1, a simplified application landscape fragment, including a BPMS, is shown. At this level of abstraction, it is visualized that a BPMS communicates with other systems in order to share information/data and exchange messages. These are communication flows (data/information flows, message flows) which are handled by the interfaces of a BPMS. A typical

business process that can be executed and logged within a BPMS is processing a production order. During the execution of such a process, usually, information from other systems is acquired and used. Each system provides the services of a certain business function (or department). For example, the CRM system provides the services of the business function called Customer Service, and, therefore, contains the customer information that is required for the registration of the production order during the business process. The same goes for the other activities in order to completely receive, register, and process the production order. This results in cross-organizational (or cross-functional) business processes that are executed and orchestrated by a BPMS. A BPMS thereby continuously generates, stores, and adjusts information in a certain order during the execution of the business processes (Dumas et al., 2018). Usually, not only systems within a single organization, but also systems from other organizations are involved in a BPMS application landscape. (Rozanski & Woods, 2012).

For any type of system that is implemented in an organization, the application landscape is one of the many important attention points. In general, the right Business-IT alignment needs to be established, which involves both organizational and technical considerations. In the past decade, Business-IT alignment was, and still is, a difficult challenge to tackle (Lankhorst, 2017; Vares, Amiri, & Parsa, 2017). Obtaining the right Business-IT alignment is one of the main objectives of Enterprise Architecture, which is an architecture discipline that focuses on the interrelations between different architecture domains within an organization. This includes the business architecture, which aims at the structure of the business processes, and the application architecture, which focuses on the application landscape. Moreover, different viewpoints and abstraction levels can be consulted. For example, it is possible to aim at the business process level in order to derive what applications / data sources are used during the execution of a certain business process (Lankhorst, 2017; Rozanski & Woods, 2012).

Architecture documentations/descriptions serve as a means for the communication, reasoning, creation, analysis, validation, and refinement of a system's architecture. It is mainly used to communicate how a system's architecture satisfies the concerns of the stakeholders through different viewpoints. This fosters a successful implementation of a system, and the fact that the desired Business-IT alignment can be obtained (Bass, Clements, & Kazman, 2003). Regarding BPMSs, Ravesteyn and Versendaal (2009) proposed the critical success factors (CSFs) as part of an effective approach for implementing a BPMS. The CSFs were divided into five different clusters/phases. One CSF aims at architecture design and points out the importance of understanding and modeling the business processes, including the interrelations between the processes and systems (data sources) that are involved. Therefore, the right architecture descriptions need to be created. Another interesting cluster aims at adhering to the domain of service-oriented architecture (SOA) during the implementation, as well as the use of web services.

1.2 Problem statement

In this research, we focus on the fact that, nowadays, a BPMS is often a low-code development platform that can be used for developing process-driven applications. Simply put, the foundation of such applications are the configured executable business process models, and the corresponding data models and user interface. In contrast to systems such as ERP systems, a BPMS explicitly determines how information is collected and used by means of the configured business process models (Weske, 2012).

Pourmirza, Peters, Dijkman and Grefen (2017) investigated the architecture behind a BPMS. They have done a Systematic Literature Review (SLR) on scientific papers describing the architecture of existing Business Process Management Systems (BPMSs). Based on the results of the SLR, they suggest the need for the design of a revised BPMS reference architecture. In order words, a generic software architecture that can be used to design the architecture of a BPMS. For this reference architecture, they concluded that the information exchange between a BPMS and other systems through interfaces (APIs), including inter-organizational communication, can be researched in more detail. In other words, the acquirement and arrangement of information during the execution of business processes, both within the BPMS and between the BPMS and other systems within the corresponding application landscape. This deals with the so-called business logic within an organization, which entails the arrangement of the communication (information/data flows, business objects etc.). The business logic determines how data is created, stored, changed and shared within the system. Business logic is implemented in a certain

programming language, such as Java and C#, and can be managed by middleware, which could be a BPMS (Levina, Holschke, & Rake-Revelant, 2010).

When a lot of business processes are executed by a BPMS, many communication flows of the collection and use of information can be derived within the application landscape. Thus, from a process-oriented point of view, it can be difficult to visualize and describe the integration between the business processes, BPMS functionality, and information/data in an easy and unambiguous way. Especially, specifying in what way a BPMS relates to and communicates (transferring information) with other relevant systems from different business functions across the application landscape. This can be investigated from different architecture viewpoints, and levels of abstraction. From a technical perspective, usually, the communication flows are structurally bounded and managed at run-time by the concept of SOA (Muller & Bohm, 2011), mainly in conjunction with a so-called enterprise service bus (ESB) (Menge, 2007).

As mentioned before, architecture descriptions support the communication/reasoning on the architecture of a certain system. For this research, we assume that a so-called Architecture Description Language (ADL) would be a suitable solution for specifying communication flows in BPMS application landscapes. In short, an ADL can be defined as a formal modeling language for creating a textual and/or graphical description/documentation and analysis of a software system's architecture and its structure and behavior in general or within a specific domain. In this case, it is an ADL regarding communication flows within the application landscape of a BPMS. In the past decades, many ADLs have been developed (Clements, 1996; Malavolta, Lago, Muccini, Pelliccione, & Tang, 2013; Guessi, Cavalcante, & Oliveira, 2015). The majority has been created to meet certain domain-specific concerns, for example, the Architecture Analysis & Design Language (AADL) for the analysis of embedded systems, and ArchiMate for Enterprise Architecture (Butting, et al., 2017). Besides such ADLs, there are also more general ADLs, for example, the widely applied Unified Modeling Language (UML). However, at this point, it is unclear what ADL is suitable for specifying communication flows in BPMS application landscapes. In addition, we need to determine how this ADL can be applied by practitioners in a process-oriented way. Thus, we can formulate the following problem statement:

"Currently, it is unclear what is required from a process-oriented ADL in order to specify communication flows in BPMS application landscapes".

1.3 Research objective and scope

Based on the research context and problem statement, the main objective of this research is *to design a process-oriented ADL that supports application development on a BPMS in terms of specifying communication flows within the corresponding application landscape.*

For this, we need to understand the common software architecture of a BPMS, and how a BPMS is implemented and used within an organization's application landscape. For this research, communication flows entails both information/data flows and message flows (choreographies) at different levels of abstraction within a BPMS. Moreover, we need to understand how a BPMS communicates with other invoked systems through interfaces (API's). So, the exact ratio between the BPMS and other integrated/invoked systems. In this way, a clear overview of the interrelations (traceability) between the different architecture domains regarding communication flows (information flows, service message flows, data flows etc.) can be created, as well as the determination of whether the desired Business-IT alignment has been established or not. Furthermore, it can be clarified how information/data from all events that occur during a particular process are acquired from any involved business function / application.

The intended ADL will be process-oriented and domain-specific. Namely, the application landscape of BPMSs that are used in service-oriented environments. It will serve as an unambiguous means when creating architecture descriptions. The meta-model that specifies the syntax and semantics of the intended ADL will be its most important asset. Regarding the software life cycle (Langer, 2016), the research context involves both BPMS implementation and the development of a business application that runs on a BPMS. As mentioned before, the foundation of such an application are the business processes that are configured within the BPMS.

1.4 Relevance

This research will result in new scientific knowledge. This knowledge will be relevant for both future scientific research and practitioners within the field of the implementation of BPMSs.

1.4.1 Scientific relevance

We propose a new, domain-specific ADL that is added to the knowledge base (literature). This intended ADL will foster the documentation/description of communication flows within BPMS application landscapes at different levels of abstraction. By doing so, regarding business and application functions and services, several properties are linked between software architecture and enterprise architecture. In addition, the ADL will contribute to the creation of a revised BPMS reference architecture, as purposed by Pourmirza et al. (2017).

1.4.2 Social relevance

Practitioners within the research context (architects, developers etc.) can apply the intended ADL in practice during a BPMS application development/implementation project. Multiple types of stakeholders will benefit from the application of the ADL. Especially, the organizations that are the service-oriented environments of a BPMS project, Furthermore, the ADL will contribute to the automation and specification of end-to-end processes by means of a BPMS. Especially, for understanding how information from integrated systems is collected and used by BPMS for the execution of the business processes.

1.5 Document structure

In this chapter, we have introduced the research that is elaborated in this master thesis. Chapter 2 contains the research approach that elaborates on the research questions and research methods. Chapter 3 contains an overview of the phasing and corresponding milestones of this research. Then, in chapter 4, the theoretical background is presented. In chapter 5, the case study organization is introduced, as well as their BPMS. Based on chapter 4 and 5, the design process of the intended ADL is elaborated in chapter 6. Moreover, the specification/structure and guidelines of the ADL are given. Next, chapter 7 focuses on the case study validation of the ADL. Finally, the results of the research are concluded in chapter 8. In addition, the limitations and future work are discussed. The remaining parts of this master thesis are chapter 9, which contains an overview of the full references to the literature, and the additional information in the Appendices.

2. Research approach

The research is conducted in a structured way. Therefore, in this chapter, we elaborate the research approach by means of presenting the list of the research questions that are answered, as well as the outline and justification of the main research method.

2.1 Research questions

In this research, we tend to answer the following main research question (MRQ):

MRQ) *What are the constituents of a process-oriented ADL for specifying communication flows in BPMS application landscapes?*

To answer the MRQ, we have formulated the following sub research questions (SRQ):

SRQ1) *What is the role of a BPMS within an application landscape?*

Answering this question will lead to applicable knowledge on the definition and purposes of a BPMS within an application landscape, its relation to BPM, as well as the architecture (functionalities, interfaces etc.) of a BPMS.
To answer this question, we conduct a literature review on relevant literature. In addition, we investigate the architecture behind a BPMS in practice by means of a desk research.

SRQ2) *What needs to be considered when designing a process-oriented ADL for specifying communication flows in BPMS application landscapes?*

By answering this question, it is clarified what ADLs are, including examples of appropriate existing ADLs, and how our intended ADL could support modelling and describing communication flows in the application landscape of a BPMS that is applied in service-oriented environments. In this way, we can define the literature-based requirements of the intended ADL, partly based on contextual considerations. Furthermore, a collection of existing ADLs has been created and compared/analyzed that will be used to design the intended ADL.
To answer this question, we read relevant literature, including literature on the Method Association Approach. In addition, we conduct an explorative semi-structured interview with one or more practitioners to clarify how the current implementation of a BPMS is done in practice. In this way, we can indicate the contextual considerations for the design of the intended ADL.

SRQ3) *What are the characteristics of the ADL?*

When this question has been answered, the characteristics (syntax and semantics) of the ADL have been determined and described. This consists of specifying requirements and considering multiple viewpoints / architecture layers, based on the meta-model of the ADL. This all is done iteratively.
In order to answer this question, we use the answers to SRQ1 and SRQ2. In addition, opinions (requirements) from multiple practitioners (architects) are acquired for the design of the ADL. For this, we conduct both semi-structured interviews and focus groups. The Method Association Approach (Deneckère, Hug, Onderstal, & Brinkkemper, 2015) provides us a structured way for designing the ADL based on suitable existing ADLs.

SRQ4) *How can the ADL be applied by practitioners?*

By answering this question, it is described how the ADL can be applied by practitioners within the domain of BPMS application landscapes.
To answer this question, we obtain input from multiple practitioners by means of both semi-structured interviews and focus groups.

SRQ5) *Is the designed ADL valid and applicable in practice for the desired purposes?*

This question is answered to determine if the designed ADL both contributes new scientific insight to the knowledge base (literature), and gives the desired means regarding the representation of communication flows in BPMS application landscapes.
In order to iteratively validate the developed ADL and determine its practicable applicability based on certain variables, we conduct a case study on the implementation of a BPMS in practice. In general, this

entails that we analyze documentation on a running or previous project in order to create situational models (viewpoints) by means of the ADL. The case study includes several validation interviews with practitioners. The validation is carried out iteratively. This means that the ADL is revised X times during the research. So, after the first validation, SQ3 and SQ4 (and, if needed, SQ1 and/or SQ2) are answered again in order to create an improved version based on the previous results. If all validation variables have been target in a positive way, the ADL is valid, and has been completely designed.

2.2 Research methods

In order to answer the research questions, we apply several research methods.

2.2.1 Information Systems Research Framework

The main research method of this research is the *Information Systems Research Framework* from Hevner, March, Park, & Ram (2004). In short, this method entails that, based on the business needs / expertise from the environment (the practitioners), applicable knowledge from the knowledge base (literature, methodologies etc.) is gathered and used in order to develop and evaluate a certain artefact. In this case, a process-oriented Architecture Description Language (ADL).

Building and validating the ADL iteratively (= Assess and Refine) ensures that it sufficiently contributes new scientific knowledge to the knowledge base, and that it is applicable in practice for achieving the desired objectives. In Figure 2, the application of the Information Systems Research Framework to this research is shown.

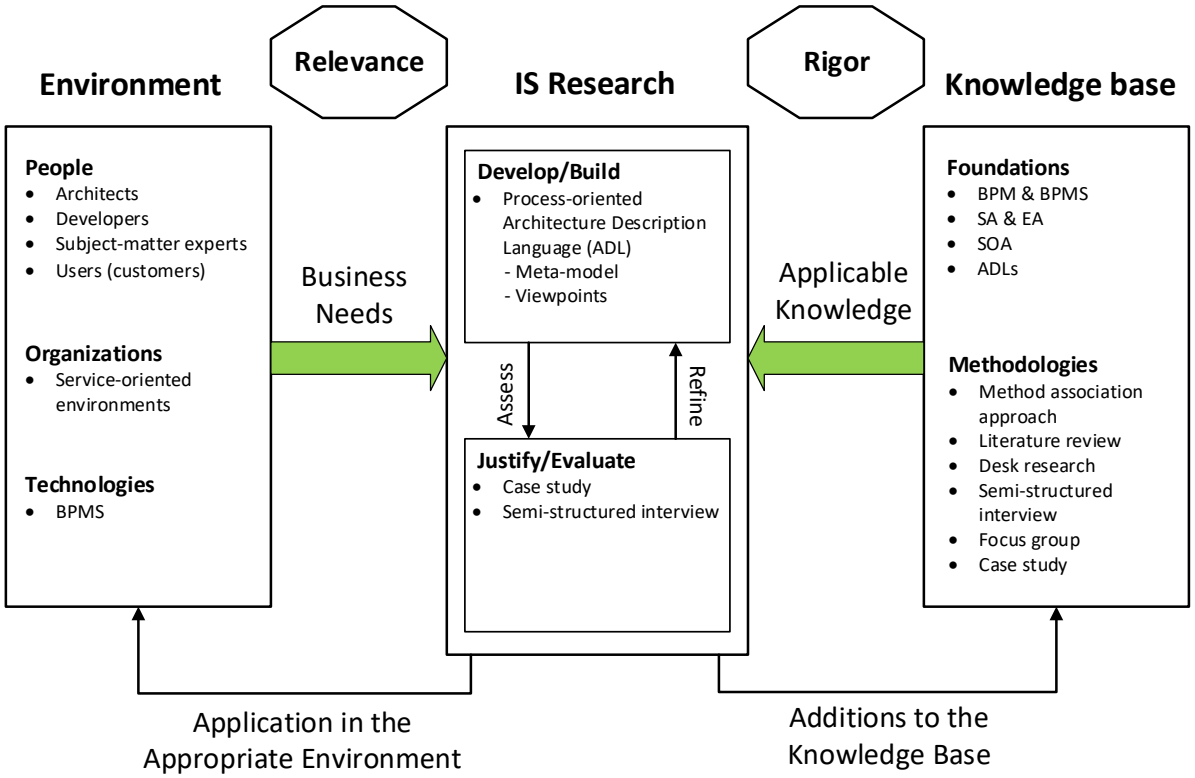


Figure 2: Information Systems Research Framework. Adopted from Hevner et al. (2004)

Hevner et al. (2004) define two paradigms regarding Information Systems Research. At the one hand, *Behavioral Science* is about the development and justification of knowledge for predicting and/or describing relevant phenomena within the context of the business need(s). On the other hand, *Design Science* focuses on the creation and evaluation of artifacts that have been designed to tackle a particular business need. Due to the fact that our objective is to design and evaluate an artifact (the ADL), *Design Science* is the most suitable paradigm.

The Design Science paradigm consist of three different cycles (Hevner, A Three Cycle View of Design Science Research, 2007). During this research, we follow these cycles. Below, the application of each cycle to this research is briefly described.

Rigor Cycle
This cycle is positioned between *Knowledge base* and *IS Research*. Following this cycle ensures that all relevant literature is gathered from the knowledge base. At the end of this research, new knowledge is added to the knowledge base. This can then be used for future research purposes. *The Rigor Cycle is followed by answering SRQ1, SRQ2 and SRQ5.*

Relevance Cycle
This cycle is positioned at both *Environment* and *IS Research*. Within this cycle, the business needs / requirements are acquired from the environment. The designed artifact is validated by means of a field study. This requires formulating acceptance criteria. The validation of the artifact might take place multiple times in order to meet all acceptance criteria. *The Relevance Cycle is followed by answering SRQ3, SQR4 and SRQ5.*

Design Cycle
This is the core cycle that involves designing and evaluating the artifact iteratively. The output of both the Relevance Cycle (the business needs / requirements) and Rigor Cycle (literature and methodologies) are used within the Design Cycle. Eventually, after several iterations, the output from the Design Cycle will be input for the other two cycles. *This cycle is followed by answering SRQ3 and SRQ4.*

2.2.2 Method Association Approach
The Method Association Approach (MAA) serves as a structured way for constructing the intended ADL based on suitable existing ADLs. The steps of the MAA are depicted below in Figure 3.

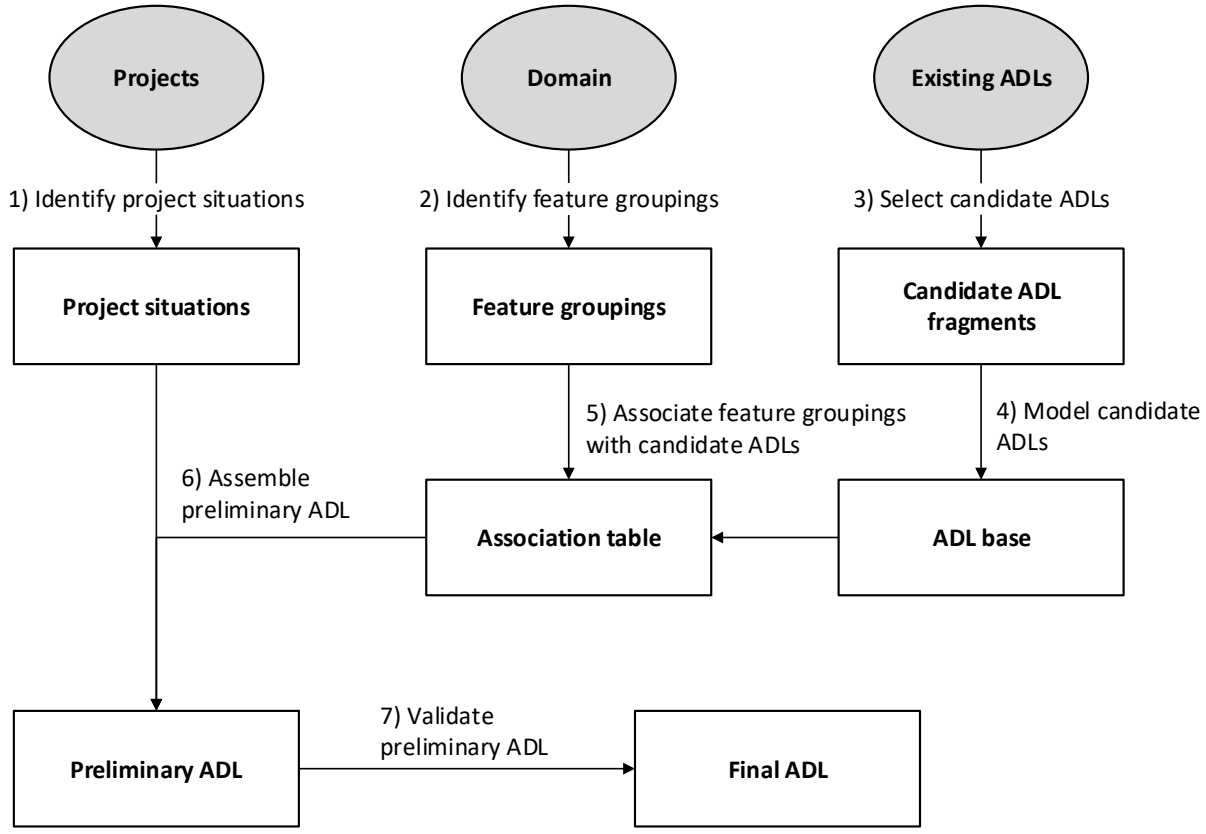


Figure 3: Method Association Approach. Adopted from Luinenburg, Jansen, Souer, Van De Weerd, & Brinkkemper (2008)

Regardless of the fact that the intended ADL is not a method to be designed, the MAA can serve as a structured design (and validation) approach. For this reason, we have partly adjusted the original MAA model to the context of this research as shown in Figure 3. However, we do not strictly follow the MAA. Below, the application of the MAA to this research is briefly described:

1) Firstly, the research context is mapped in order to formulate a proper problem statement. This entails investigating both the functional architecture and technical architecture of a BMPS in practice, and the implementation approach, including the creation of architecture descriptions) that is currently applied by a real organization through interviews and desk research.
2) The desired features (requirements) of the intended ADL are divided into multiple categories (feature groupings).
3) Relevant literature on existing candidate ADLs is reviewed in order to collect knowledge on several fragments of existing ADLs that can be used to design the intended ADL.
4) Relevant models of the candidate ADLs are selected and then stored at one place.
5) The properties of the candidate ADLs are compared and analyzed with the desired features. This results is the selection of the most suitable existing ADLs.
6) A preliminary version of the intended ADL is created and described based on the most suitable fragments of existing ADLs. This is done by means of both conducting interviews and focus groups with relevant stakeholders (architects, developers etc.). The validation is done by means of a case study, which entails applying the ADL to a running example.
7) The final version of the intended ADL is created and described based on the design and validation iterations of the previous draft versions of the intended ADL.

2.2.3 Literature review

In this sub paragraph, the literature review approach is described. This includes specifying suitable sources for finding the required literature, and describing the search approach. Eventually, an overview of the current knowledge on the field of BPM, BPMS and ADLs is created that serves as our theoretical foundation to identify the relevant literature gaps. This overview can be found in chapter 4.

We use Google Scholar as the main source to find relevant literature. Through this source, lots of (scientific) literature from different databases and websites can be found. To ensure no relevant literature is missed out, IEEE Xplore, WorldCat, and Scopus are also accessed. When searching for relevant literature, we make use of several search strings, including abbreviations and some synonyms. Basically, the search strings are the key words that identify the topic, context, and objectives of this research. The search strings have been categorized and are listed below.

Research approach

- *Information Systems Research Framework;*
- *Method Association Approach.*

BPM & BPMS

- *Business Process Management (BPM);*
- *Business Process Management system (BPMS / BPM system);*
- *Workflow Management (WFM);*
- *Process-driven;*
- *Event-driven;*
- *Communication flow;*
- *Information/data flow;*
- *Message flow;*
- *Choreography.*

Architecture

- *Architecture Description Language (ADL);*
- *Software Architecture (SA);*
- *Enterprise Architecture (EA);*
- *Service-Oriented Architecture (SOA);*
- *Application landscape.*

To collect other relevant literature from the literature that was found through the aforementioned sources, we make use of the Snowballing Method from Wohlin (2014). Basically, this method entails that, based on the reference list of each found paper, book etc., other potential literature is found. In

addition to this so-called backwards snowballing, we apply forward snowballing by means of checking in what other papers a particular interesting paper is cited. The more this paper has been cited by other writes, the better the quality and reliability of this paper.

Regarding literature on relevant existing ADLs, the main objective is to gather the specification documents. In addition, several papers on the practical applications are found. For each paper, we determine its relevance by reading the title, abstract, introduction, and conclusion. If applicable, other sections are read. Informative text is highlighted.

2.2.4 Desk research

The desk research entails the examination of documents, presentations and other material on a BPMS that is applied in practice. We apply this research method in order to collect practical facts about the current architecture behind this BPMS, the provided approach of application development, and the corresponding implementation within an organization.

2.2.5 Semi-structured interviews

We conduct semi-structured interviews. This means that interview protocols are used that do not need to be completely followed. In other words, there is room for asking additional ad-hoc questions to further clarify the interviewee's answers. Thus, it is possible that the questions are not asked in the stated order and/or that some questions could not be asked due to time constraints. The main objective of the semi-structured interviews is to obtain the opinions from the relevant practitioners on the design and practical application of the intended ADL as part of determining its constituents.

2.2.6 Focus groups

Next to the semi-structured interviews, we hold one or more focus groups. During a focus group, input from multiple person is collected by means of a presentation/discussion on several topics related to the design and validation of the intended ADL.

2.2.7 Case study

As stated before, we validate the intended ADL multiple times by means of a case study. This entails that the ADL is used within a project in the real world. This also includes that we conduct several semi-structured interviews with practitioners in order to collect their opinions on the outcomes of the case study validation.

2.3 Conceptual overview

In Table 1, it is briefly indicated how each sub research question (SRQ) will be answered. Basically, this table summarizes paragraph 2.1 and 2.2. In addition, for each SRQ, it is specified what practical input is required from the environment.

Table 1: Conceptual overview

SRQ	How?	Outcomes / required information	Practical input
1	• Literature review • Desk research	• The role of a BPMS within an application landscape: - Definitions of BPM - BPM life cycle - Definitions of a BPMS - Main functionalities of a BPMS - BPMS architectures - BPMS interfaces	• Documentation on the architecture behind a BPMS that is applied in practice
2	• Literature review • Semi-structured interviews	• Purpose and requirements of an ADL: - Examples of appropriate ADLs - Literature-based requirements of the ADL - Contextual considerations • Method Association Approach	• The current approach of implementing a BPMS that is applied in practice
3	• Semi-structured interviews • Focus groups • Method Asso-ciation Approach	• Requirements specification of the ADL • Applicable viewpoints and views, including a meta-model of the ADL • Design and description of the properties of the intended ADL - Syntax and semantics	• Opinions from relevant practitioners on the design of the ADL
4	• Semi-structured interviews • Focus groups	• Description of how the ADL can be applied by practitioners	• Opinions from relevant practitioners on the desired practical application of the ADL
5	• Case study • Semi-structured interviews	• Validation protocol: - Variables and/or acceptance criteria • Iterative validation of the ADL: - Situational models / viewpoints • Evaluation of the practical applicability and scientific contribution to the literature	• Information about current and/or previous implementations of a BPMS that is applied in practice • Opinions from relevant practitioners on the validity and practical applicability of the ADL

3. Phasing and milestones

This chapter elaborates on the phasing and milestones of this research. For this, we show the main steps and outcome of this research by means of a roadmap.

3.1 Roadmap and phasing

In Figure 4, a conceptual research roadmap that visualizes the steps (= sub phases) and outcomes of the proposed research is shown. Basically, this roadmap is a concise overview of chapter 2.

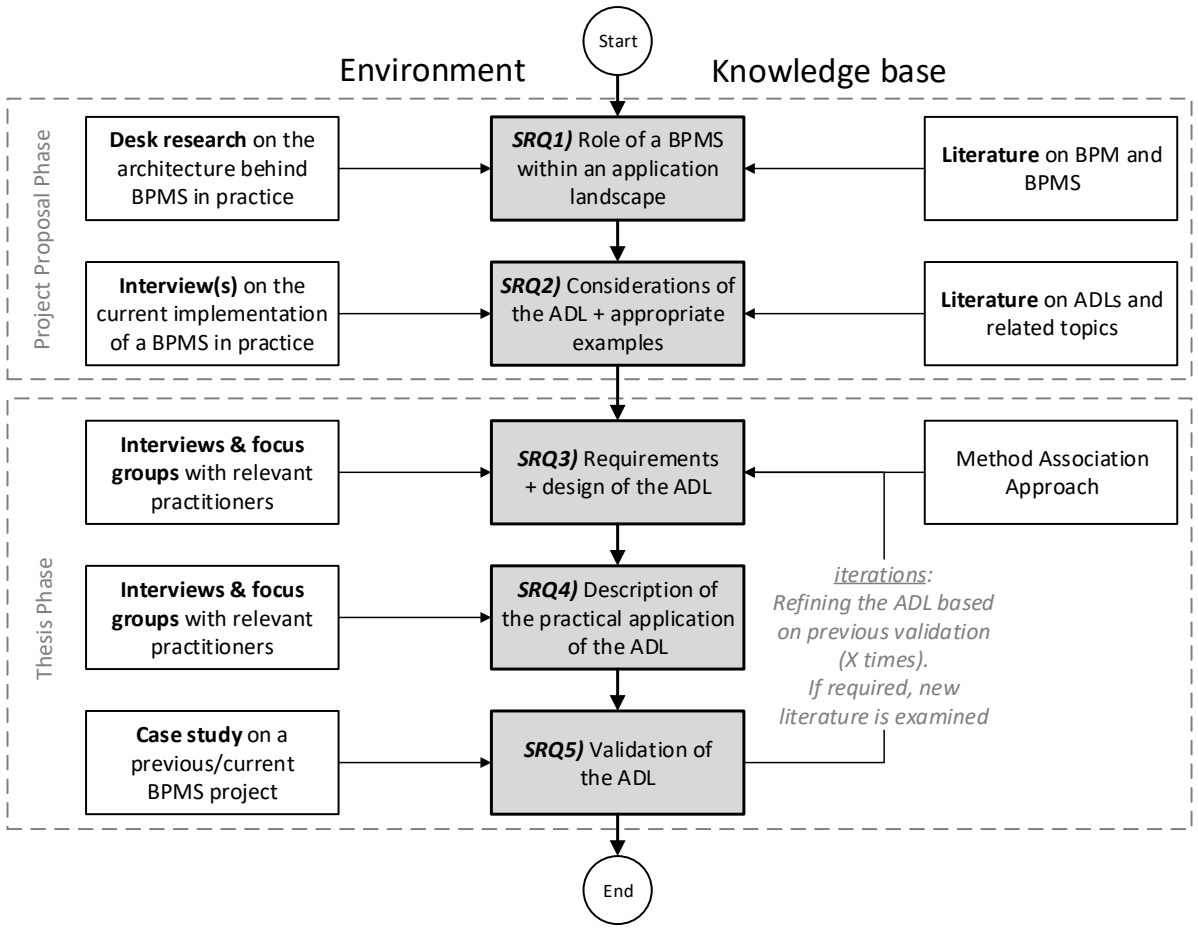


Figure 4: Conceptual research roadmap

SRQ1 and SRQ2 are answered during the **Project Proposal Phase**. This phase lasts around three or four months, since the beginning of this research on 15-Jun-2018. Then, during the **Thesis Phase**, SRQ3, SRQ4, and SRQ5 are answered. The expected duration of this phase is five months, since the end of the previous phase.

3.2 Milestones

During this research, there are several milestones, which are specified in Table 2.

Table 2: Milestones

Phase	Sub phase / Deliverable	Deadline
Project Proposal Phase	Project Proposal	15-Oct-18
	• Role of a BPMS within an application landscape	12-Oct-18
	• Considerations of the ADL + appropriate examples	12-Oct-18
Thesis Phase	Thesis report	15-Mar-19
	• Requirements + design of the ADL	15-Jan-19
	• Description of the practical application of the ADL	31-Jan-18
	• Validation of the ADL	22-Feb-19

4. Theoretical background

By means of conducting a literature review, we obtain applicable knowledge from relevant literature. This knowledge serves as the theoretical background for this research. Therefore, this chapter contains the results of the literature review. We describe the relevant theories and models that are needed to answer SRQ1 and SRQ2. Eventually, we have identified literature gaps that give us a clear view on the potential scientific contribution of this research.

4.1 Business Process Management

In the past decades, Business Process Management (BPM) has become a quite mature discipline that is of great interest in today's organizations, and scientific research field (van der Aalst, 2013; Recker & Mendling, 2015). As the term BPM already indicates, BPM is about managing business processes. A business process is a coherent set of activities that are carried out in a specific way in order to reach a particular goal (Weske, 2012). For example, processing (or rejecting) customer payment claims within an insurance company. For this example, a customer claim can only be processed if it meets some predefined requirements. In other words, it is determined whether the customer claim is correct or not. Incorrect claims are rejected. In Figure 5, this short example case is modelled by using the well-known Business Process Model and Notation (BPMN).

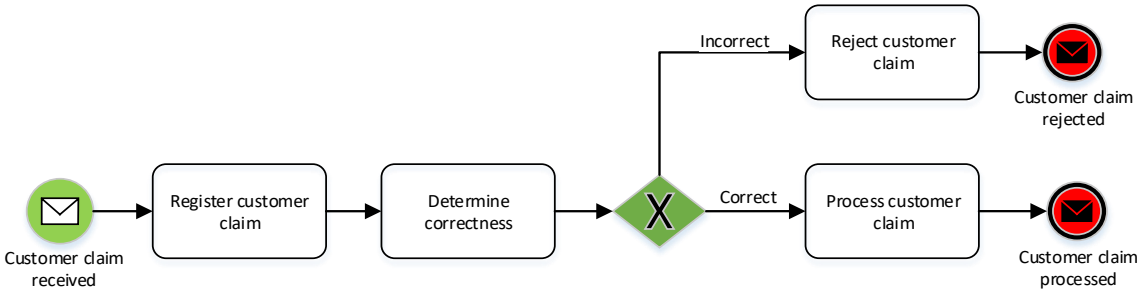


Figure 5: BPMN example

According to Dumas, La Rosa, Mendling, and Reijers (2018, p. 6), BPM is *"a body of methods, techniques and tools to discover, analyze, redesign, execute and monitor business processes in order to optimize their performance".* Quite similar to this, Weske (2012, p. 5) says that *"Business process management includes concepts, methods, and techniques to support the design, administration, configuration, enactment, and analysis of business processes".* Hence, based on these two definitions, BPM can be called a structured way of constructing, mapping and optimizing business processes. By means of applying BPM, the core business of an organization can be systematically improved.

When applying BPM, business processes are continuously mapped, monitored and optimized/adjusted in a structured way. This is divided into different steps that are repeatedly assessed in a specific order, the so-called BPM lifecycle, which visualizes the aforementioned definitions of BPM. In Figure 6, the BPM lifecycle is depicted (Dumas et al., (2018).

First, *Process identification* aims at the selection and identification of the business processes that are relevant to a certain business problem (issue). This results in a revised process architecture to be used in the next steps. Then, the current business processes are modelled and documented during the *Process discovery* step. This results in as-is process models. The *Process analysis* step results in an analysis, and, if possible, performance measures, of all issues that occur in the business process that is investigated. Based on the outcomes of the previous steps, potential changes to the business process are determined during the Process redesign step. This is done in order to tackle the identified issues. In this way, a to-be process model has been created. Eventually, *Process implementation* involves both automation and change management in order to bridge the gap between the as-is process model and to-be process model. After the implementation of the revised business process, data on the business performance is acquired and analyzed (*Process monitoring)*. This is done in order to determine the extent to which the to-be business process is improved in comparison to the previous version.

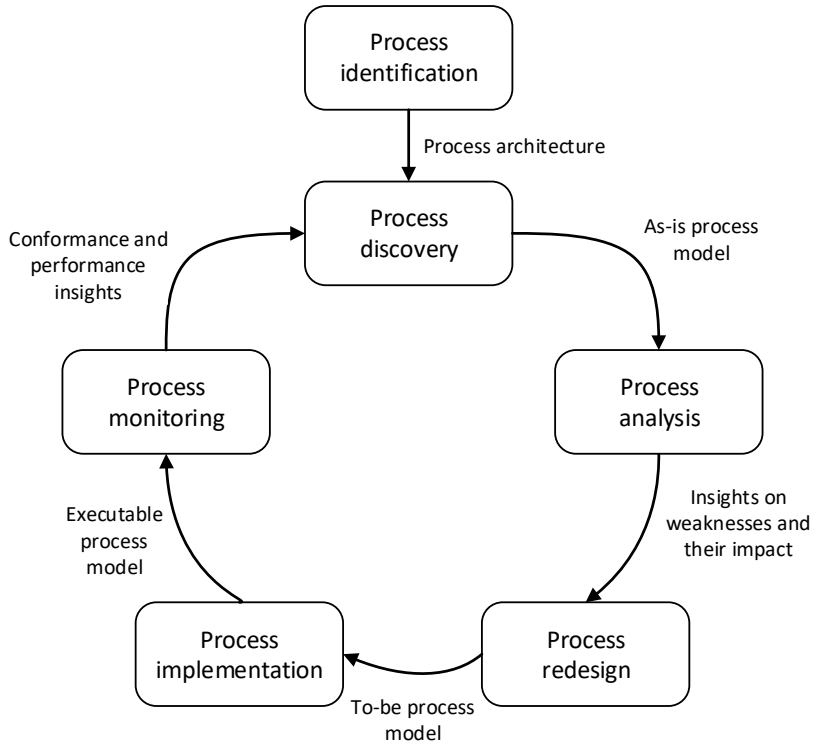


Figure 6: BPM lifecycle. Adopted from Dumas et al. (2018)

BPM is related to a discipline called Workflow Management (WFM). WFM focuses on the (partly) automation of business processes by means of ensuring correct information flows (documents, tasks etc.) between different persons within an organization in order to realize a certain business goal. The coordination of the information flows is done based on a predefined set of rules (Hollingsworth, 1995). BPM is quite similar to WFM. However, WFM focuses more on the management of the information flows (documents) between people, whereas BPM aims at the improvement/optimization of business processes and the corresponding interrelations within an organization as a whole (Dumas et al., 2018).

Case Management (CM) is a discipline that is similar to BPM. CM aims at the arrangement of information/data that is required to fulfill/finalize a certain business process. For this, a case represents the collection of the required data that is used during the life cycle of a case. This case life cycle is divided into multiple stages/states. The most important difference between BPM and CM is the fact that BPM focuses on single processes, whereas CM aims at the interrelation of a complete set of business processes (stages) regarding the workflow / life cycle of a certain case. This involves the arrangement of the input of multiple people during the workflow. For example, to fully process a customer payment claim within an insurance company, it would be necessary that people from different departments within the company need to perform certain tasks regarding the approval or rejection of the payment claim. Possible stages/states for the payment claim would be *creation*, *send*, *registration*, *check* and *approved/rejected* (Dumas et al., 2018; Marin, 2016).

4.2 BPMS

Given the notion of BPM and the BPM lifecycle, a Business Process Management System (BPMS) is one of the emerging technologies that supports the automation of end-to-end processes. A BPMS is a software intensive system that (partly) automates the steps of the BPM lifecycle. Besides process automation for workload reduction, a BPMS also provide insight into the performance (efficiency) of the business processes, and simplifies the evolution of business processes within the BPM lifecycle. A BPMS ensures that activities/events of the business processes are carried out at the right time and at the right place. Therefore, explicit executable (BPMN) process models need to be loaded into the BPMS (Dumas et al., 2018). The need for explicit process models is given by the definition of a BPMS from Weske (2012, p. 5): *"A generic software system that is driven by explicit process representations to coordinate the enactment of business processes"*.

A BPMS is classified as a so-called process-aware information system (PAIS). BPMSs are related to Workflow Management Systems (WFMSs), another PAIS which do not support all steps of the BPM Lifecycle. WFMSs mainly support modeling and executing/automating processes (Dumas et al., 2018). Besides WFMSs, there are several other business and IT systems/disciplines that have contributed to the creation of a BPMS as a new type of system. To visualize this, an evolution roadmap of BPMSs is shown in Figure 7.

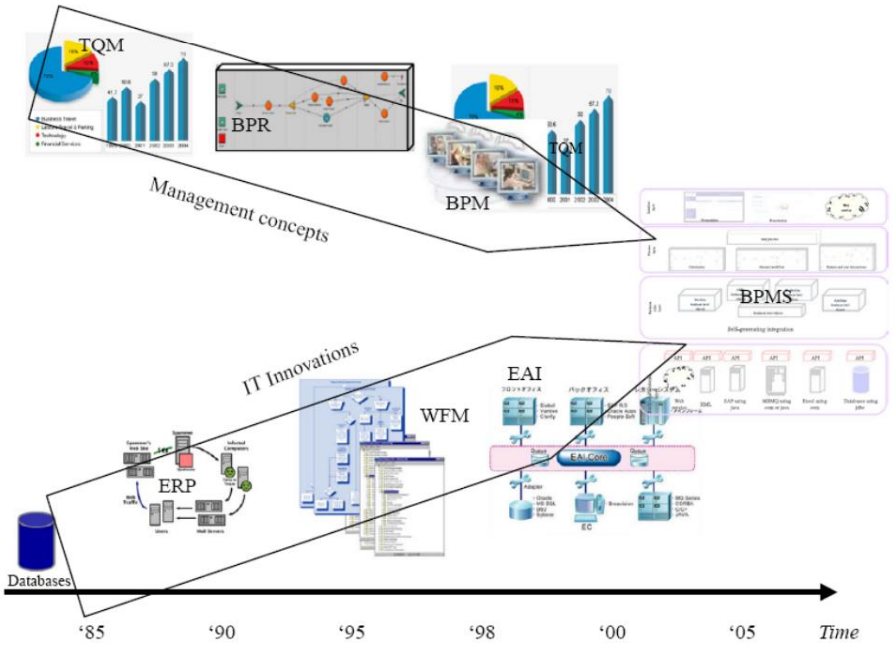


Figure 7: BPMS evolution roadmap. Adopted from Ravesteyn & Versendaal (2007)

There are different types of BPMSs. Not every BPMS offers the same features, and, therefore, do not support the BPM lifecycle in the same way. There are BPMSs that only provide the ability to model, automate, and analyze business processes, whereas other BPMSs also provide more advantaged capabilities, such as Business Intelligence, Robotic Process Automation, and Business Rules Management. Based on the way a BPMS structures the business processes, and the extent to which it is process-driven or data-driven (the orientation on process or data), four types can be defined. In Figure 8, these types are depicted.

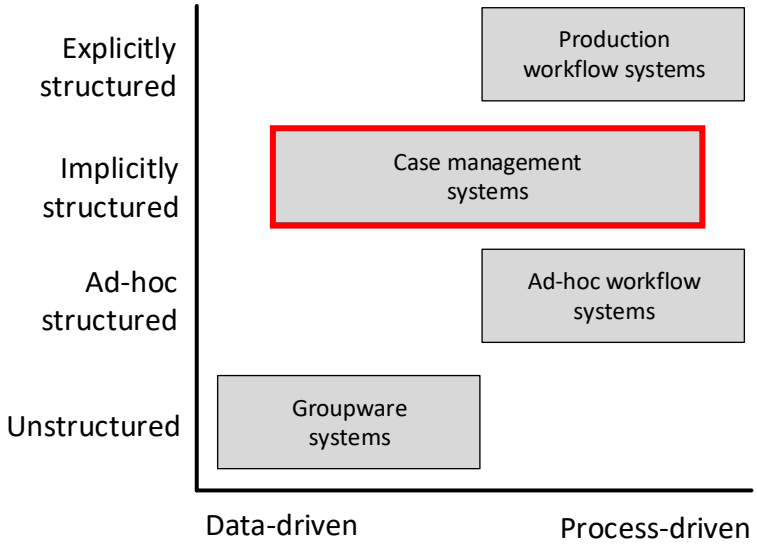


Figure 8: Types of BPMSs. Adopted from Dumas et al. (2018)

Below, each type of BPMS is briefly described (Dumas et al., 2018):

- *Production workflow systems.* These are the most used BPMSs. They provide the general features of a BPMS. The systems work with explicit process models. Sometimes, it collaborates with one or more separated database management systems;
- *Groupware systems*. The core of these BPMSs is the possibility of document sharing and communicating between different users. However, a groupware system hardly supports business process management purposes;
- *Case management systems*. These systems support the creation, execution and management of business processes (cases) that have been modelled implicitly. This entails that these business processes are partly modelled at a high abstraction level. Details, such as the history and current state of a certain case, can be easily monitored and provided to the users;
- *Ad-hoc workflow systems*. Within these systems, it is possible to instantly design and change business processes (cases), even when there are executed. Therefore, the users need to be familiar with the business processes that have been loaded into the system. Furthermore, suitable process modeling tools need to be at hand.

The Workflow Reference Model has originally served as a solid foundation for the design of both WFMS and BPMS architectures since its publication (Pourmirza et al., 2017). This reference model is depicted in Figure 9.

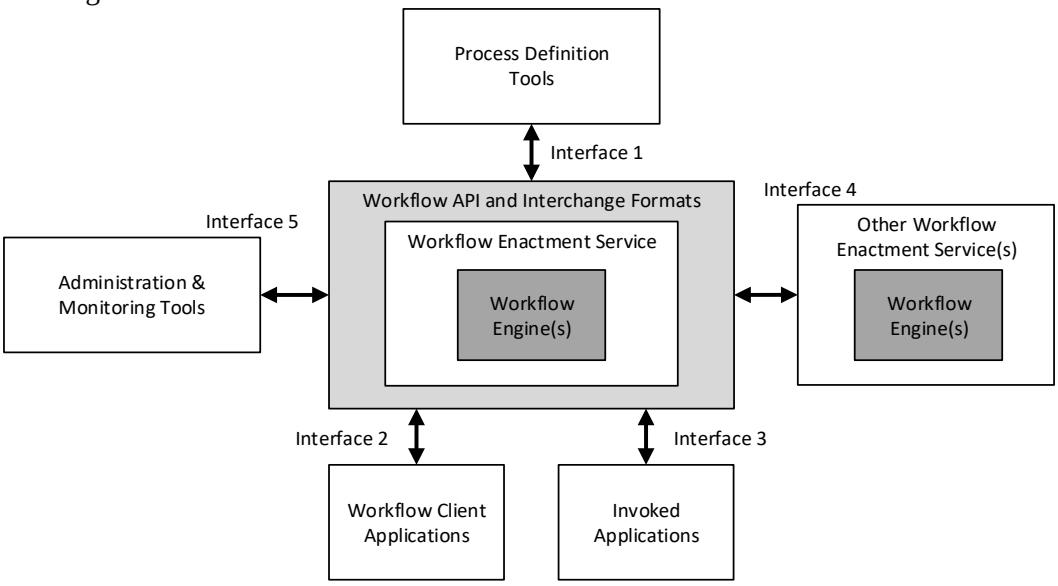


Figure 9: Workflow Reference Model. Adopted from Hollingsworth (1995)

As can be seen, the components and corresponding interfaces of a WFMS are shown. The *Workflow Enactment Service* is responsible for creating, maintaining, and executing workflow instances by means of one or multiple workflow engines. In this way, the run-time environment is provided with external data that is necessary for the execution of architectural activities. The *Workflow Engine* executes the architectural activities within the run-time environment of the corresponding service and workflow instance. In addition, *Other Workflow Enactment Service(s)* (= external services) are used for maintenance of workflow instances. Human input is registered through the *Workflow Client Applications.* The business process models are created and documented by means of the *Process Definition Tools.* The *Administration & Monitoring Tools* are used for monitoring and administration purposes. In case external sources are needed, *Invoked Applications* are involved.

The interaction between the different parts goes through several interfaces. *Interface 1 - The Workflow Definition Interchange* is an application programming interface (API) that handles the exchange of information about the business processes that have been loaded into the BPMS. *Interface 2 - The Workflow Client Application Interface* handles the communication (transferring workflows) between the Workflow Client Applications and the Workflow Enactment Service. *Interface 3 - The Invoked Applications Interface* transfers required process definition details from applications (local or external).

Interface 4 - The Workflow Application Program Interface (WAPI) handles the communication between separated workflow systems. *Interface 5 - The Administration and Monitoring Interface* is responsible for the exchange of relevant information regarding the administration, and the mapping of business processes.

The today's general architecture of a BPMS is quite similar to the aforementioned Workflow Reference Model. In Figure 10, a simplified architecture model of a BPMS is shown.

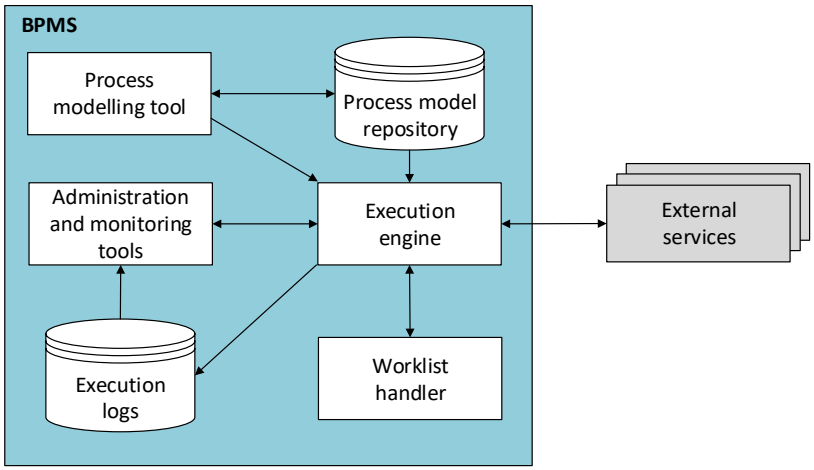


Figure 10: General architecture of a BPMS. Adopted from Dumas et al. (2018)

A BPMS consists of several tools/modules and repositories (the software components) and corresponding communication flows (information exchange) through interfaces. Basically, a BPMS can be seen as a system that is a coherent set of several tools (modules), repositories, and interfaces between them. Nowadays, most interfaces are configured in conjunction with / as web services in order to be able to access components of the BPMS via the internet. The *Process Modelling Tool* is used to design and change process models. These models are saved in and loaded from the *Process model repository*, and can be executed through the *Execution engine*. This engine is the central point of a BPMS and creates process instances / cases that can be executed. In most situations, *External services* are involved when a certain business process is executed. These services are provided by external applications within the application landscape where the BPMS has been implemented. The *Worklist handler* can be seen as the place where the status of work list items are maintained. These items are carried out by the process participants, which are the actors (internal or external) of the organization. The *Administration and monitoring tools* are needed for the administration of all events that occur within the BPMS, and monitor the performance of business processes that are carried out through the BPMS. Monitoring the execution of business processes results in *Execution logs* that are stored in a certain repository. The main difference between the Workflow Reference Model and the general BPMS architecture is the fact that the today's use of web service has changed the exact functionality of several interfaces (Dumas et al., 2018).

Usually, a BPMS runs within a service-oriented architecture (SOA), which is a widely applied architecture style (Dumas, La Rosa, Mendling, & Reijers, 2018). Ko, Lee and Lee (2009) already indicated the raising importance of SOA for BPM within the industry. Basically, by means of SOA, application components provide their business functionalities as (web) services to other applications. These services can then be invoked through interfaces. SOA makes it easy to add, remove, and reuse application components. This results in a flexible architecture that is easy to manage (Lankhorst, 2017). As already mentioned before, usually, during the execution of a business process, multiple

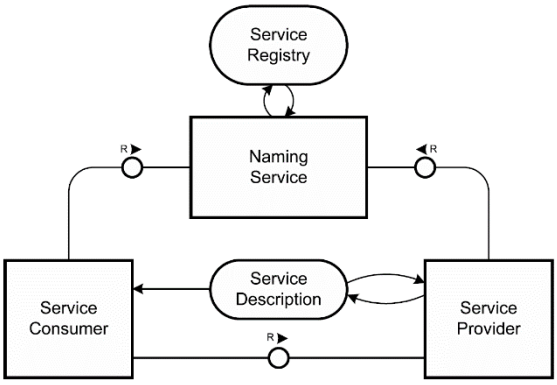


Figure 11: Typical service-oriented architecture. Adopted from Menge (2007, p. 2)

applications are involved. By means of service orchestration, the services of multiple applications can be integrated with each other in a certain sequence in order (partly) automate a (complex) business process. To illustrate this, a typical SOA is shown in Figure 11. Basically, a service that is provided by a service provider is stored in a so-called naming service. This 'repository' is accessed by service consumers that want to make use of a provided service, based on the corresponding service description. In this way, an entire business process can be configured based on assigning services to the activities of the business process. It is not a problem if the applications are programmed in different languages, not part of the same application landscape, and/or other interoperability factors. SOA is mainly combined with an Enterprise Service Bus (ESB). Basically, an ESB fosters Enterprise Application Integration (EAI) since it manages all communication flows between different applications within an application landscape. This means that the applications do not directly communicate with each other for exchanging information/data using different interfaces and protocols. Instead, they only communicate with the ESB which ensures that a communication flow from a certain application goes to the intended application in a standardized way. In case an organization has hundreds of different applications, an ESB prevents that an application landscape becomes an entire mess of communication flows between all applications (Menge, 2007). This is visualized in Figure 12. Nowadays, a BPMS can provide enterprise application integration capabilities that are related to the functionalities of an ESB.

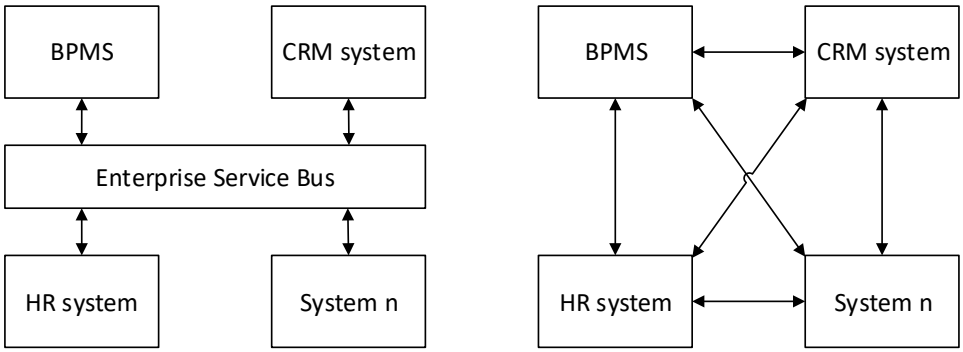


Figure 12: Communication flows with an ESB (left) and without an ESB (right)

4.3 Architecture

Within the context of this research, we can define *Architecture* as a coherent structure/foundation of a system's concepts, properties, and corresponding elements and interactions, within a specific environment that somehow has influences on the system's design and evolution (Lankhorst, 2017). There are different architecture disciplines that each focus on a particular architecture domains. To a certain extent, these disciplines are related to each other. In the following sub paragraphs, the relevant disciplines are described.

4.3.1 Software Architecture

According to Bass, Clements, and Kazman (2003, p. 45), Software Architecture (SA) is "*the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both*". More precisely, software architecture is about software design, the management of stakeholders and concerns, the arrangement of functional requirements, and characteristics of the system. A software architecture of a system has both a *static structure* and *dynamic structure*. The static structure entails the functional design time elements (components) and the corresponding arrangement that provide the system's desired features. The *dynamic structure* refers to the system's behavior of the run-time elements and corresponding interactions through interfaces (Rozanski & Woods, 2012). A software architecture is specified in an architecture description (AD), which is defined as *"a set of products that documents an architecture in a way its stakeholders can understand and demonstrates that the architecture has met their concerns"* (Rozanski & Woods, 2012, p. 207). In Figure 13, the conceptual contents of an AD are shown. This research focuses on the red marked parts. In short, the model shows that *Stakeholders* (users, architects, developers etc.) have one or more *Concerns* about a certain software *System* which is deployed and used in a particular environment to fulfill some objective(s). This system has an *Architecture* which can be described and visualized by using different Architecture *Viewpoints* that are included to the *Architecture Description (AD)*. The viewpoints are

defined to serve as the guidelines for creating a particular type of *Architecture View*, which is a specific representation of system by means of a certain notation/visualization, in order to address the concerns of one or more stakeholders. *Architecture Models* can be used to represent a view. The conventions of the models are specified by a *Model Kind*. Furthermore, *Architecture Rationale* and *Correspondence Rules* specify the architecture design reasoning and arrangement/relations between the contents of an AD.

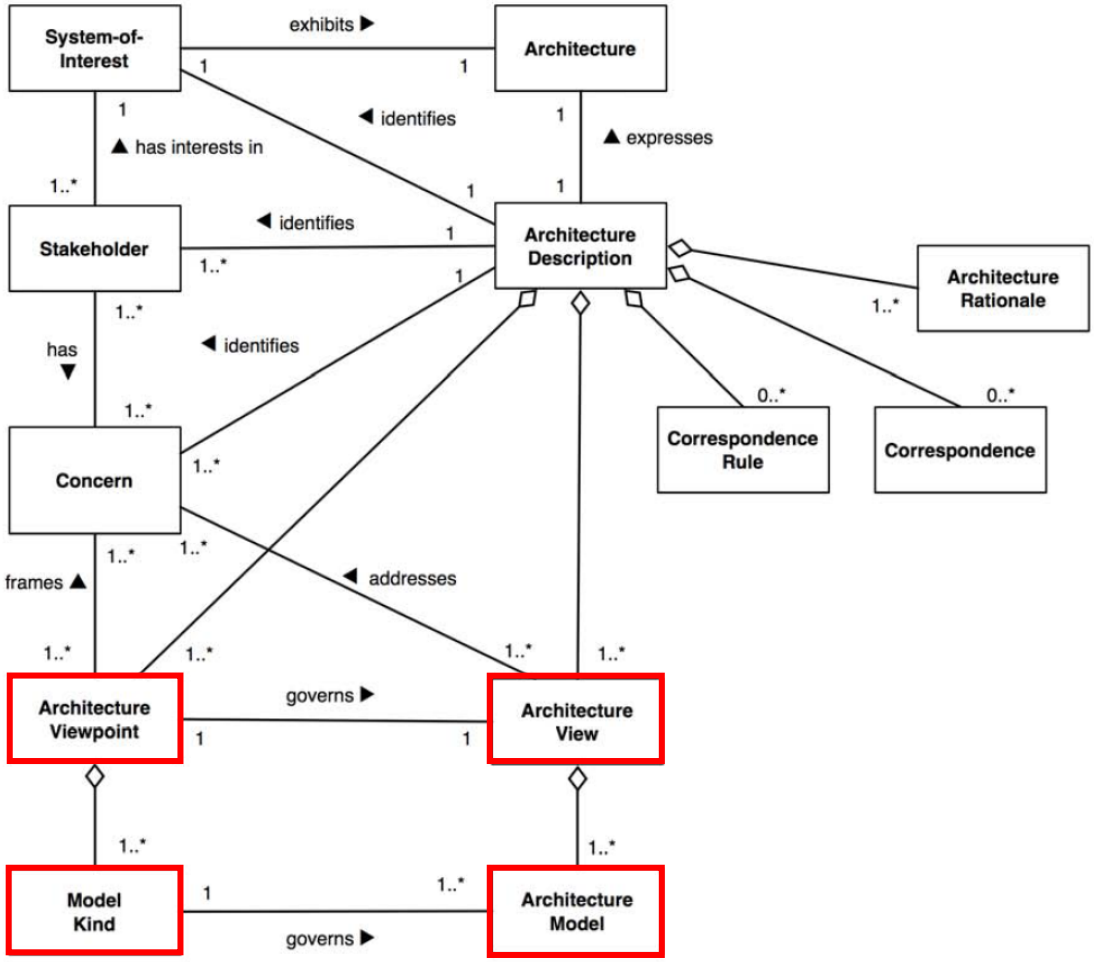


Figure 13: Conceptual model of an architecture description. Adopted from ISO/IEC/IEEE (2011, p. 5)

Both the static structure and dynamic structure of a software system can be visualized from different viewpoints, each consisting of multiple views. In Table 3, the software architecture viewpoints of Rozanski & Woods (2012) are briefly described.

Table 3: Software architecture viewpoints

Viewpoint	Definition + example views
Context viewpoint	This viewpoint specifies the contextual environment of the system. So, external elements such as people and other systems, and how these elements interact with the system. Relevant views are, e.g., the UML use case diagram and the UML context diagram.
Functional viewpoint	This viewpoint describes the functional software components (modules such as Sales and Production) and corresponding interactions through interfaces. So, the static structures. Relevant views are, e.g., the functional architecture model (FAM) and a feature diagram.
Information viewpoint	This viewpoint represents how information is stored and managed within the system and how the information is shared within the system's contextual environment. Relevant views are, e.g., an ERD, a BPMN process model, and a UML class diagram.

Concurrency viewpoint	This viewpoint focuses on the concurrency structure of a system. Therefore, it specifies the system's behavior and communication protocols between different components/modules. Relevant views are, e.g., petri nets, and a UML state diagram.
Development viewpoint	This viewpoint aims at the architecture (resource code structures, dependencies etc.) that is used during the development process. A relevant view is, e.g., a code line model.
Deployment viewpoint	This viewpoint describes the technical environment / infrastructure of the system. Relevant views are, e.g., network models, and a UML deployment diagram.
Operational viewpoint	This viewpoint specifies the use of the system in case it runs live within its running environment. Relevant views are e.g. installation models, and migration models.

For this research, the functional viewpoint and information viewpoint are most relevant when designing the intended ADL. Namely, these viewpoints focus on the specification of communication flows (information flows, message flows, data flows etc.) and the corresponding interfaces.

4.3.2 Enterprise Architecture

Lankhorst (2017, p. 3) defines Enterprise Architecture (EA) as *"a coherent whole of principles, methods, and models that are used in the design and realisation of an enterprise's organisational structure, business processes, information systems, and infrastructure"*.

The main objective of enterprise architecture is to structure/align and manage both the business and IT within an organization is such way that organizational goals are achieved in the most effective and efficient way. Therefore, in contrast to software architecture, enterprise architecture has a wider scope since it does not only focus on the architecture of a single software systems. In fact, enterprise architecture aims at multiple architectural domains and the interrelations between them at the organization/enterprise level. For this, enterprise architecture divides an organization into a business layer, application layer, technology layer, and how these layers are connected to each other. These layers are visualized in Figure 14, and are distinguished by ArchiMate, which the ADL for Enterprise Architecture Modelling (The Open Group, 2017).

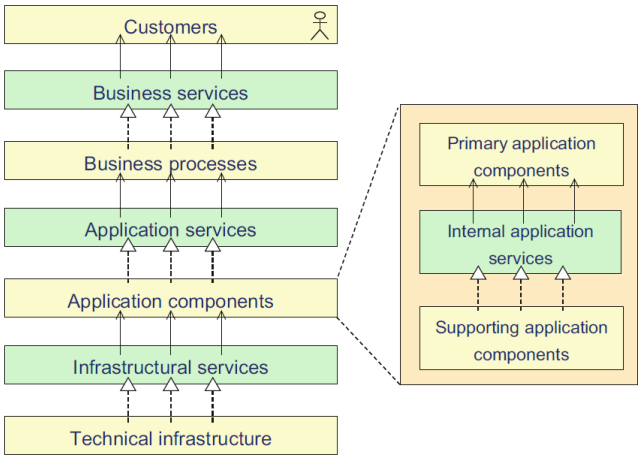


Figure 14: Enterprise Architecture layers. Adopted from Lankhorst (2017, p. 76)

The different layers/domains are interconnected by means of services. A service can be defined as a functionality of a certain entity that is provided to its environment. More precisely, the business processes provide the business services to the customers (external environment). This is specified in the business architecture that also aims at the business functions. A business function is "*a collection of business behaviour based on a chosen set of criteria (typically required business resources and/or competences), closely aligned to an organisation, but not necessarily explicitly governed by the organisation*" (Lankhorst, 2017, p. 91). BPM focuses on the business architecture layer. During the

execution of a business process, some activities might use a certain functionality of an application. This functionality is provided by means of application services from the application architecture. This research focuses on this layer, because this layer elaborates on the application landscape, as well as the individual application components. At the application layer, software architecture comes along since it focuses on the internal architecture and services of a single application. For this research, a BPMS is the system of interest. All applications run on certain infrastructure hardware (servers, databases etc.) from the technical infrastructure. This architecture layer provides the infrastructural services in order to use the applications at run-time. Hence, the link between EA and BPM is the fact that EA shows in what way business processes and corresponding business functions are interrelated with the application landscape and technical IT infrastructure. In other words, EA clarifies what applications support the execution of the business processes, and on what infrastructure hardware (servers, databases etc.) these applications are running (Lankhorst, 2017).

The specification and visualization of the communication flows deals with business logic which entails the way an organization operates, and thus how data within the organization needs to be managed. Many systems (could) have a so-called 3-tier architecture. Such an architecture is an extension of the well-known client/server (2-tier) architecture, and consists of three layers: *presentation / user interface layer* at the top, *business / application logic layer* in the middle, and a *data layer* at the bottom. Business logic entails both business rules and workflows. The business rules (IF-THEN) define the path of the workflow, including the decision points and communication flows that occurs within the process workflows. Thus, the business rules determine how information/data flow within a workflow (Levina, Holschke, & Rake-Revelant, 2010). Business logic can be seen as a part of the middleware tier that manages the communication between what is shown to the clients/users within the user interface at the presentation layer, and the data that is stored in a database at the data layer (Rozanski & Woods, 2012).

As mentioned before, for this research, we want to clarify how communication flows (information/data flows, message exchange, interfaces) between a BPMS and different external applications from multiple business functions within an organization (and optional applications of external organizations) can be described and modelled in an unambiguous and process-oriented way. The answer to this question suggests several links between the domain of software architecture and enterprise architecture.

4.3.3 Model-Driven Architecture

A BPMS can provide the capabilities for Model-Driven Engineering (MDE). For this, in Figure 15, the Model-Driven Architecture (MDA) is depicted. The MDA is an approach for MDE, and is defined and maintained by the Objected Management Group (OMG). In short, MDE uses models as the main artifacts during software development. In other words, MDE uses models that can be transformed into executable software code in order to simplify parts of the software development process.

MDA applies several OMG standards, including the Unified Modelling Language (UML), and the Meta-Object Facility (MOF). The MDA distinguishes three levels of abstraction that are interrelated regarding several aspects, including communication flows: (1) the CIM, which is a high-level representation of the business domain, (2) the PIM, which is a general specification of the system's structure and behavior regarding both the business and IT services and functionalities, and (3) the PSM, which is a technology specification, including code models, of the system, aimed at a certain implementation platform (Brambilla, Cabot, & Wimmer, 2017).

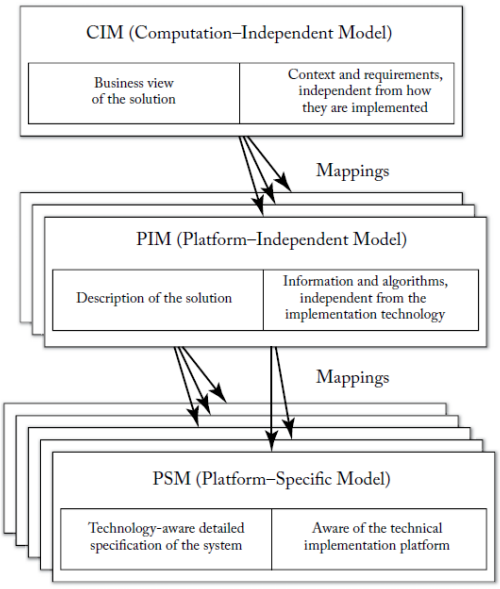


Figure 15: Model-Driven Architecture. Adopted from Brambilla et al. (2017, p. 45)

4.4 Architecture Description Languages

As part of the design of the intended ADL, we need to examine literature on the notion of ADLs. This includes an identification of the main building blocks and requirements of an ADL, and a brief comparison analysis of suitable existing ADLs. This analysis is elaborated in chapter 6, after chapter 5 elaborates on the BPMS of the case study organization.

4.4.1 Definition of an ADL

For several decades, research has been done on the notion of ADLs. An ADL is a language (textual and/or visual) that can be used to model and describe the conceptual architecture of a software system. It has a specific (formal) syntax and semantics that serve as a means for creating an explicit specification of an architecture (Clements, 1996; Medvidovic & Taylor, 1997).

From a syntactical point of view, there are general-purpose languages (GPLs) and domain-specific languages (DSLs) (Brambilla et al., 2017). According to Malavolta, Lago, Muccini, Pelliccione and Tang (2013), there are three types of architectural languages (ALs): (1) general box-and-line languages, (2) formal ADLs, and (3) UML and its subsets/profiles. To put this in perspective, Clements (1996) described at what points an ADL differs from requirements languages, programming languages and modelling languages. Regarding software architecture, requirements languages aim at the problem space, whereas ADLs are focused on the solution space. In contrast to modelling languages, ADLs focuses more on the representation of software components. Modelling languages emphasize a system's behavior, such as process modelling. Furthermore, the difference between ADLs and programming languages is the fact that ADLs are more explicit regarding the cohesion and interconnections of architectural abstractions. Next to these languages, a domain-specific language is a language that can be used to specify a domain-specific (part of a) system. Despite the fact that ADLs focus either on a general or particular domain, according to Lankhorst (2017), many ADLs lack of a clear, overall view on the interrelations across different architectural domains/layers within an organization. Moreover, when aiming at a lower abstraction level, it is difficult to interrelate different model elements, mainly regarding the dynamic structure of a software system (Rozanski & Woods, 2012).

Hence, due to the fact that ADLs have several overlaps with the other aforementioned languages, and the difference between them are not always completely clear, in this thesis report, we define an ADL as *any type of graphical / modelling language that can be used to visualize and specify the architecture of a system*. This definition is aligned with the definition of an ADL, according to ISO/IEC/IEEE (2011, p. 10): *"any form of expression for use in architecture descriptions"*. Thus, next to strictly called ADLs, we also consider UML-based languages and (general) modelling languages as ADLs during this research. General/informal box-and-line languages are out of scope in order to avoid ambiguity regarding the model shapes.

4.4.2 Common properties and requirements

Medvidovic & Taylor (1997) described the following main building blocks as part of their framework for classifying a certain language as an ADL:

- *Components* are collections of the functional run-time and behavior elements (calculations or data stores) of a system, for example, a module, ruleset, interface, and web service. When the components' semantics, including its constraints, can be modelled, it is easier to perform architectural analysis/mappings with respect to different abstraction levels. The interaction services that are provided by a component to other components are defined by its interface. The use of abstract component types supports reuse of components. Furthermore, by means of refinement and subtyping, an ADL can foster the evolution of its components;
- *Connectors* are used to model and specify the interaction between models, e.g., by means of different type of flows. The interface and constraints of a connector define how certain points are used to connect with components. Performing analyses on a connector is fostered by means of a clear communication protocol specification and transaction semantics. Similar to components, the evolution of the connectors can be fostered by means of refinement and subtyping;

- *Architectural configurations* specify the way components and connectors can interact with each other. This can be done implicitly across different specifications, explicitly in a separated specification, or through an in-lined manner within the models. Clear and correct configurations ensure that the specification, including the constraints, of the ADL is understandable in order to simplify the refinement of its properties, such as scalability and traceability;
- *Tool support* can be provided by an ADL in order to perform (automated) activities such as the modeling and specification of an architecture and corresponding views, analysis of certain properties (errors, consistency among views etc.), and incremental changes (refinement) of the architecture models.

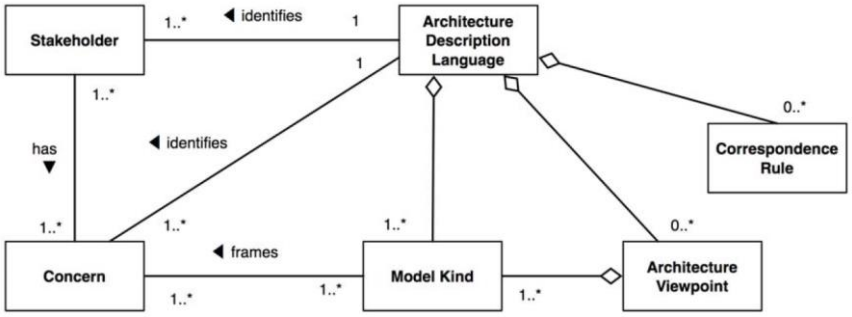


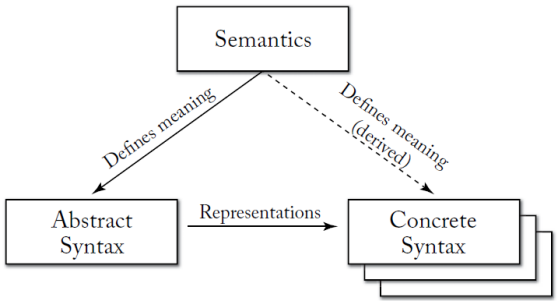
Figure 16: Conceptual meta-model of an ADL. Adopted from ISO/IEC/IEEE (2011, p. 11)

By means of an ADL, a structured description of a software system's architecture can be created. In Figure 16, a conceptual meta-model of the specification of an ADL is depicted. This model shows that, by means of an ADL, the *Concerns* of one or more *Stakeholders* can be described and visualized. Usually, this can be done through different *Viewpoints*. However, as indicated by [0..*], an ADL does not have to focus on a viewpoint. The conventions/interrelations between multiple viewpoints (*Model Kind*) can be specified as well. *Correspondence Rules* are needed to ensure consistency and traceability within the viewpoints and corresponding views that have been created by means of the ADL.

An ADL is meant for either a general or particular / domain-specific purpose in the field of software systems. Though, apart from the aforementioned main building blocks of an ADL, most ADLs share the properties that are specified in Table 4. We have written down the properties that are the minimal requirements of an ADL in red. These properties and requirements, which are still relevant nowadays, are derived from the results of the survey of Clements (1996) on the following early (academic) ADLs: ArTrek, Code, Demeter, Modechart, PSDL/CAPS, Resolve, Unicon, and Wright.

Table 4: Common properties and requirements of an ADL

Property	Description
Graphical and/or textual (formal) syntax and semantics	The syntax of an ADL specifies how it needs to be used, whereas the semantics is about the meaning/definition of the graphical and/or textual notation. In Figure 17, the relationships between the syntax and semantics are shown. The concrete syntax is the actual representation (textual and/or graphical) of the abstract syntax, which specifies what is allowed to be modelled in what way. The semantics define the meaning of concrete syntax elements. The syntax and semantics of a language can be formal, semi-formal or informal. Usually, an ADL is formal. This is characterized by the fact that a formal syntax and semantics is extensive and is often hard to understand. Though, this provides a precise/explicit notation that reduces unambiguity, and could be used within a tool for certain purposes, such as (automated) mathematical analysis. Informal ADLs are often created ad-hoc, and do not provide formal analysis capabilities. In between, a semi-formal ADL has a well-

	defined syntax. However, the corresponding semantics is often incomplete or quite implicit (Guessi, Cavalcante, & Oliveira, 2015). The syntax and semantics of most languages are specified by means of a meta-model, for example, the meta-model of an ADL in Figure 16. In fact, a meta-model specifies the rules/constraints of how the elements of a certain language can be used to create specific models (Deneckère et al., 2015).  **Figure 17**: Relationship between syntax and semantics. Adopted from Brambilla et al. (2017, p. 64)
Viewpoints and abstraction levels	Despite the fact that an ADL does not have to focus on a specific viewpoint, most ADLs do support the creation of views of one or more viewpoints. The same goes for the possibility of distinguishing and specifying different levels of details / abstraction levels within architecture models.
Architecture creation, refinement and validation	An ADL must support the creation, refinement and validation of architecture descriptions. Architecture creation is about the creation and specification of architecture models. Architecture refinement deals with managing/monitoring the incremental changes/ refinement of the architecture description. Architectural validation entails determining and evaluating whether or not the system's architecture meets certain requirements (Rozanski & Woods, 2012). The intended ADL will be process-oriented. According to Clements (1996), a process-oriented ADL mainly focuses on the creation, validation, analysis and refinements of architecture descriptions.
Analysis support	By means of architecture-level information, analytical purposes on non-functional properties need to be provided. Example of architectural analysis are analyzing/calculating a system's availability and reliability, and a consistency check between viewpoints.
Architecture styles	An architecture style can be seen as a coherent set of architectural elements and corresponding rules/guidelines for the relationships between the elements and the use in a given context (Rozanski & Woods, 2012). The two most common architecture styles must be represented by an ADL. The first one is the *component based style* that distinguishes functional and logical components and fosters the reusability of the components regarding software design. The second one is the *layered style* which divides an architecture into multiple layers that are interconnected with each other. Furthermore, due to the scope of this research, the *service-oriented architecture (SOA) style*, and the traditional *client/tier (n-tier) style*, which is basically how a client communicates with a server via the

	internet in order to receive data from a certain database, are also relevant.
Design decisions capturing	An ADL can provide design decisions capturing by means of general text annotations only and/or by more advanced specification mechanisms to derive design rationale.
Specifying distributed systems	An ADL can provide the ability to model the interrelations and communication between (components of) distributed systems through integrations and interfaces.
Machine readable / tool support	The possibility of using an ADL within a certain tool that could support/automate, for example, the analytical purposes of the ADL.
Common architecture links	The possibility of specifying architectures that adhere to / are based on a common reference architecture from a different (higher) abstraction level.

4.4.3 Related work on the development of ADLs

In the past decades, lots of research has been done on the development of ADLs, mainly to meet domain-specific needs. However, there are no studies that are particularly focused on an ADL for BPMSs. Some studies only performed a general survey on the properties of existing ADLs, for example, the aforementioned survey of Clements (1996), whereas other researchers have defined the characteristics of an ADL for a certain type of system. For example, Guessi, Cavalcante & Oliveira (2015) have determined characteristics for the development of an ADL that can be used to formally describe the architecture of software intensive systems-of-systems. A BPMS can be categorized as such a system due to the fact that a BPMS integrates small pieces of functionality of multiple systems in a certain order for the execution of the business processes. It was analyzed to what extent these characteristics are presented in the following existing ADLs: UML, CML, SysML and X-UNITY. It was concluded that none of these ADLs fully provided the required features. Faulkner & Kolp (2003) focused on the domain of information systems architectures that contain multiple agents (a situational system entity that is flexible to adhere to its design objective). For this, they have identified the requirements for an ADL called SKwyRL-ADL that can be used to specify such systems. This ADL was designed based on several existing ADLs, and a certain agent model.

Many ADLs have been created by means of extending other existing ADLs. The majority of these ADLs are based on UML. These are the so-called UML profiles/subsets, for example, SoaML (Object Management Group, 2012), and SysML (Object Management Group, 2017b). In addition to this, by means of extending SoaML, Zúñiga-Prieto, Insfran & Abrahão (2016) have proposed an ADL for the specification of increment architectures that are integrated into the architecture of cloud services. For this, they adjusted the meta-model of SoaML at certain points.

Furthermore, there are ADLs that are in fact combinations of two or more ADLs. Behjati, Yue, Nejati, Briand, & Selic (2011) have combined SysML with several concepts of AADL in order to create the so-called Extended SysML for Architecture Analysis Modeling (ExSAM) profile for the specification of embedded systems. The ExSAM profile combines the system design and modelling capabilities of SysML with the analysis purposes of AADL. The design of the profile was done by means of a mapping and partly combining the syntax meta-models of both ADLs. More recently, Chen et al. (2018) have designed ArchME, which is a SysML-based ADL for modelling complex mechatronic system architectures. ArchME extends several system modelling capabilities of SysML.

4.4.4 Practical needs and application of ADLs

Less research has been done on the practical needs and application of ADLs. Regarding this topic, the most recent study was performed by Malavolta, Lago, Muccini, Pelliccione and Tang (2013). They have questioned 48 practitioners within the field of software architecture modeling by means of both interviews and questionnaires. The main purpose of their study was to collect and present data on the actual needs of the practitioners regarding ADLs, as well as their opinions (degree of satisfaction, usefulness, and limitations) on the features provided by existing ADLs. They have found that (early) academic ADLs do not fully fulfil the needs of today's practitioners. The practitioners rely more on the

ADLs that originate from the industry itself: the majority (86%) uses UML (or a UML subset/profile) for their architecture descriptions. Also, ArchiMate and AADL are commonly used. Moreover, most practitioners use multiple views within their architecture descriptions. Furthermore, the study has indicated that useful ADL features are related to the communication on the architecture between stakeholders by means of different views, a well-defined graphical syntax and semantics, and other more specific purposes such as analytical capabilities, traceability, cross-view consistency check, and versioning. Features such as interoperability checks and forward/reverse engineering tend to be less useful in practice.

4.5 Web services and APIs

Due to the scope of the intended ADL, we also investigate the today's common use of web services. A web service can be defined as software with certain functionalities / services (a module, a person etc.) that is accessible for a client (person, application, module of a BPMS etc.) via the internet (web browser). For this, a unique URL and HTTP protocol, usually, in conjunction with the so-called Simple Object Access Protocol (SOAP) for standardized component communication, is used. Next to SOAP, the Representational State Transfer (REST) is a similar protocol that is stateless and less extensive. Therefore, REST is most suitable for less extensive applications, such as ad-hoc based web-services. The interfaces and communication with other applications and all other specifications are written in XML. So, basically, a web service is a technical representation of a specific business function that is available via the internet. Basically, all types of web services are application programming interfaces (APIs), which specifies how a system can communicate with other system. However, not every API is a web service. Thus, there are APIs that cannot be used for the communication between system services via the internet (Sheng, et al., 2014).

Regarding web services and APIs / interfaces, next to ADLs, there are also other (technical) languages that are relevant for the design of the intended ADL. In the following sub paragraphs, these languages are briefly described.

4.5.1 Web Services Business Process Execution Language (WS-BPEL)

WS-BPEL is a so-called XML-based Web Services Description Language (WSDL) that can be used to specify events/actions that occur when a business process is executed by means of web services and corresponding provided functionalities. WS-BPEL is aligned with the Business Process Model and Notation (BPMN). WS-BPEL can be used to specify and execute two types of business processes that make use of interfaces of web services: executable processes (for behavior within the processes), and abstract processes (implicit processes). WS-BPEL distinguishes different types of executable activities, including the interaction (*invoke*, *reply*, *receive*) between applications, *wait* for a certain amount of time, and the indication of error conditions (*throw*). Most BPMN shapes and interfaces can be specified in WS-BPEL. For example, a Message Start Event that acts as the trigger for starting a process. In WS-BPEL, this event type uses a *receive* activity, and is specified as shown in Figure 18.

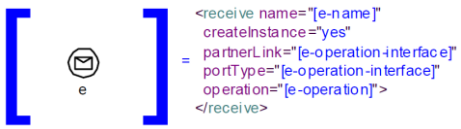


Figure 18: Message Start Event in WSBPEL. Adopted from Object Management Group (2013, p. 455)

Another example is a BPMN *Service Task*, including message flows. In WS-BPEL, this is called an *invoke* activity, and is specified as shown in Figure 19.

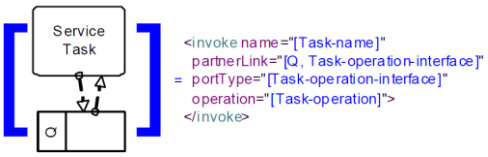


Figure 19: Service Task in WSBPEL. Adopted from Object Management Group (2013, p. 448)

Eventually, in this way, a complete business process (shapes, flows, interfaces etc.) can be made executable by means of specifying the whole process model in a large WS-BPEL script. However, WS-

BPEL has a few restrictions. It does not allow a deadlock, which is a state of a token that cannot move further through the process. Furthermore, the process needs to be synchronized which means at each sequence flow can have only one token (Object Management Group, 2013).

4.5.2 Web Services Choreography Description Language (WS-CDL)

Next to WS-BPEL, the Web Services Choreography Description Language (WS-CDL) is an XML-based language that is meant for describing the behavior of participants regarding their peer-to-peer collaborations / message exchange through web services. In other words, WS-CDL can be used to specify the communication protocols between participants within (inter-organizational) business processes. This is related to both the Choreography and Collaboration perspectives of BPMN. The main parts of a WS-CDL document are (1) *package information*, which act as the root/meta definition of a choreographic definition, and (2) *choreographic definition*, which is the main part of the specification of the participants' collaborations. Regarding activities, WS-CDL has work-unit activities and control-flow activities. The former specify the statuses regarding the execution of activities, whereas the latter entails three different types: *sequence*, *choice*, and *parallel* activities. Similar types of activities are also presented in WS-BPEL. In Figure 20, a part of a WS-CDL script is shown.

```
1  <sequence>
2   <interaction   name="AnnualStatementSubmission" .... >
3    <participate relationshipType="ClientTaxAdvisor"  .... />
4     <exchange name = "AnnualStatementSubmissionExchange" .... >
          ....
5     </exchange>
6   </interaction>
7   <choice>
```

= Start of a sequence activity, configuration of participant's relationship and the corresponding message exchange and decision point.

```
8     <workunit          name="CheckAnnualStatementRejection"
9                        guard="cdl:isVariableAvailable (cdl:getVariable(„isCompliant",
10                                „ServiceProviderRole") = false))"
11                       block="true">
12      <interaction     name="AnnualStatementRejection"
13                       channelVariable="RejectAnnualStatementChannel"
14                       operation="RejectAnnualStatement" initiate="false">
15      <participate     relationshipType="ClientTaxAdvisor"  .... />
          ...
16      </interaction>
17    </workunit>
```

= Work-unit activity regarding a message exchange on a rejected annual statement.

```
18    <workunit          name="CheckAnnualStatementAcceptance"
19                       guard="cdl:isVariableAvailable (cdl:getVariable(„isCompliant",
20                               „ServiceProviderRole") = true))"
21                       block="true">
22      <parallel>
23      <interaction     name="SendConfirmanceOfAceptance"
24                       channelVariable="ConfirmanceOfAcceptance"
25                       operation="SendConfirmanceOfAcceptance" initiate="false">
26      <participate     relationshipType="ClientTaxAdvisor"  .... />
          ...
27      </interaction>
28      <interaction     name="SendProcessedAnnualStatement"
29                       channelVariable="ProcessedAnnualStatemen"
30                       operation="SendProcessedAnnualStatemen" initiate="false">
31      <participate     relationshipType="TaxAdvisorMunicipality"  .... />
          ...
33      </interaction>
          ...
34      </parallel>
35    </workunit>
```

= Work-unit activity regarding a message exchange on an accepted annual statement.

```
36   </choice>
37  </sequence>
      ...
38 </choreography>
```

= End of the WS-CDL script

Figure 20: Decision point and interaction in WS-CDL. Adopted from Mendling & Hafner (2008, p. 9)

It contains a possible specification of a decision point within a process. This process involves a tax advisor that, in conjunction with the municipality, either accepted or rejects an annual statement from a client. This decision is made based on certain data variables and occurs along with both sequential and parallel activities. In contrast to WS-BPEL, WS-CDL is not executable. Hence, WS-CDL can only be used as a description language for the aforementioned purpose. Regardless of this fact, WS-CDL can be used along with WS-BPEL (Mendling & Hafner, 2008). Furthermore, for more comprehensive specifications of the communication different application components through application programming interfaces (APIs.), Interface Description Languages (IDLs) can be used. For example, the JavaScript Object Notation (JSON) web service protocol (Sheng, et al., 2014).

4.6 Summary

By means of a literature review, we have elaborated the theoretical background that partly provides the answers to SRQ1 and SRQ2. The answers to these questions also include a few indicated literature gaps. The main research method of this research serves as a structured way of new scientific knowledge that tend to close the gaps.

SRQ1) *What is the role of a BPMS within an application landscape?*

Business Process Management (BPM) can be defined as a structured approach for the design, mapping, and optimization of business processes in order to systematically improve an organization's core business. For this, the BPM lifecycle contains six steps regarding the evolution/lifecycle of business processes. A business process management system (BPMS) is a software intensive system-of system that partly automates the execution of the BPM life cycle steps. A BPMS provides the required features for modelling, executing, and analyzing/monitoring business processes by means of explicit executable models. Within an application landscape, a BPMS communicates with other systems as an orchestrator in order to exchange information/data that is required for the execution of the business processes. Nowadays, this communication is managed in a standardized and structured way by means of a service-oriented architecture (SOA) in conjunction with an enterprise service bus (ESB) and web services. Moreover, many BPMS have built-in features and different types of interfaces that simplify enterprise application integration (EAI).

SRQ2) *What needs to be considered when designing a process-oriented ADL for specifying communication flows in BPMS application landscapes?*

During this research, we define an ADL as "*any type of graphical / modelling language that can be used to visualize and specify the architecture of a system"*, which is similar to the definition of an ADL according to ISO/IEC/IEEE (2011, p. 10). The main building blocks of an ADL are its components, its connectors, the corresponding configurations, and additional tool support. An ADL must at least support the creation, analysis, refinement, and validation of architecture descriptions based on architecture level information, as well as the possibility to apply the common used architecture styles. Regarding the development of ADLs, lots of research have been done in the past decades. Many ADLs have been designed. Some ADLs are more general-oriented while others are aimed at domain-specific needs. However, limited scientific research has been done on a particular ADL for the domain of BPMSs, which can be seen as software intensive systems-of-systems. This includes the fact that many languages lack of an overall view on the interrelation across multiple architecture domains, as well as a clear traceability between different model elements regarding the dynamic structure of a software system. More precisely, the link between software architecture and enterprise architecture regarding business functions and modelling communication within inter-processes by means of message flows. In addition to this, the communication between a BPMS and invoked applications through API's and web services can be studied in more detail. Especially, regarding inter-organizational communication. This needs to be investigated at different levels of abstraction / granularity.

5. Case study organization

Regarding the context of this research and understanding the domain / practical context of the intended ADL, this chapter elaborates on the current architecture behind Pega Platform from Pegasystems, the BPMS that is implemented in practice by the case study organization: BPM Company. By means of this information, SRQ1 is answered completely. SRQ2 is answered completely in the next chapter. The required information is gathered by means of short explorative interviews / personal communications with relevant practitioners, following a few relevant online courses of the Pega Academy, and studying relevant documents and presentations from Pegasystems.

5.1 BPM Company

The case study organization of this research is *BPM Company*, a software/consultancy company that is specialized in both implementing the Pega Platform (the BPMS / low-code development platform sold by Pegasystems) and developing business applications on the platform at different types of organizations. Besides in The Netherlands, BPM Company also operates in Belgium and Romania. BPM Company has both Pega business architects and system architects working together on projects at the customers. BPM Company was founded in 2011. Since then, they have gained much experience regarding the development of business applications on the Pega Platform at different types of organizations.

We use the Pega Platform, including its application landscape, as the case subject during this research. The remaining paragraphs of this chapter elaborate more on the architecture of the Pega Platform, as well as the development/implementation approach that is applied.

5.2 Pega Platform

Since 2011, BPM Company is one of the Dutch partners of Pegasystems. Pegasystems is a software company from the United States and is specialized in developing and selling software for operational excellence and customer engagement purposes by means of business process management, customer relationship management, and digital process automation. Pega's software is suitable for multiple industries including the healthcare and financial organizations. The main product of Pegasystems is the BPMS called the Pega Platform, which is a so-called low-code development platform for rapid application development. The applications that run on the Pega Platform are driven by business process flows. The majority of the software code, including Java, HTML5 and SQL, is generated automatically, which also involves the configurations of components and connectors (communication flows). This ensures that applications are flexible and, thus, can be changed rapidly based upon situational/contextual changes. The application development is done in either the basic *Pega Express* or the more comprehensive environment *Pega Designer Studio*. The Pega Platform as a whole (a unified platform) can be called an extensive BPMS. Based on Figure 8, it is characterized as a case management system.

5.2.1 Functional architecture

In Appendix A, a comprehensive overview of the functional architecture[1] can be found. In addition, in Figure 21, a simplified model is shown.

As can be seen, the Pega Platform is a unified platform / BPMS that consists of several functional capabilities (modules), including Business Process Management, Business Rules Management, and Dynamic Case Management. All together, they (partly) support/automate the steps of the BPM lifecycle (see Figure 6). The functional capabilities must not be seen as separated products. Each of them can be accessed through browser-based models. The capabilities that are most relevant for this research are briefly described.

[1] *The material on the functional architecture of the Pega Platform was received from the daily supervisor and was acquired during a presentation from Pegasystems on the Pega Platform.*

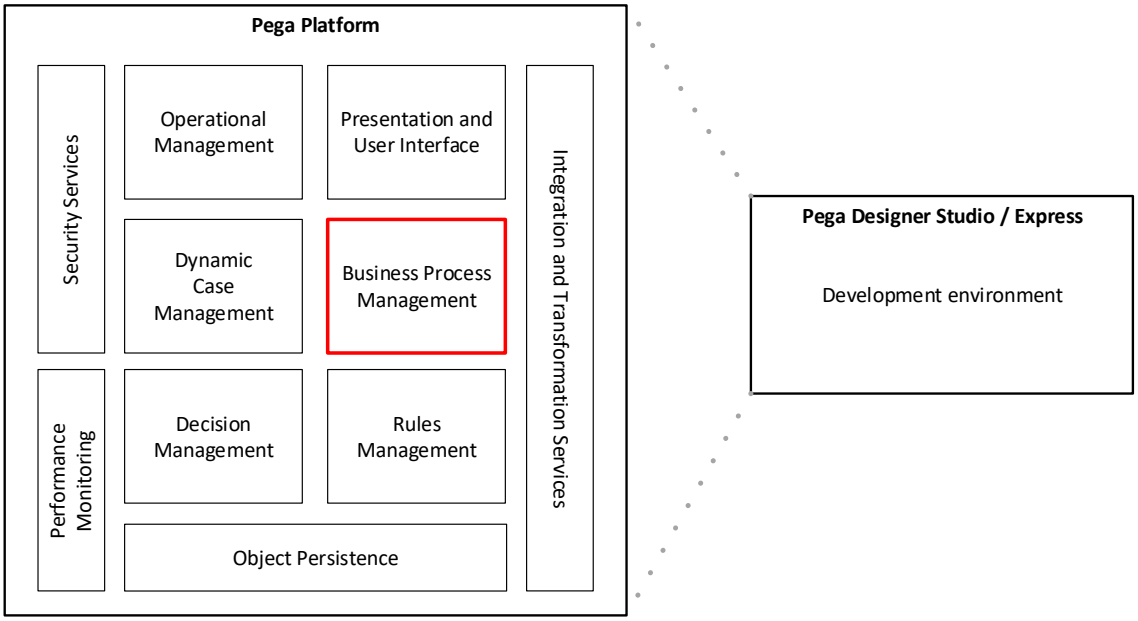


Figure 21: Pega Platform functional architecture (simplified)

Object Persistence Services. Within an application that is built on the Pega Platform, the cases are the core work objects. Namely, each case contains information that is needed for processing the business processes, including the corresponding rules and decisions. By means of a built-in framework and a relational database, complex relationships between objects can be modelled. Therefore, it is possible to easily reuse object libraries.

Business Process Management. This research focuses on this capability that manages process definitions. These definitions are used to model and specify each business process model at both high-level, by means of so-called Discovery Maps, and at low-level (implementation view), through a BPMN-based modelling language. It is quite easy to switch between these abstraction levels. The implicit business process models that are created and executed can contain manual activities, automated activities, decision points through business rules etc. Both happy flows and alternative flows (based on business rules) can be modelled and specified.

Business Rules Management. The rules engine handles different types of rules, including decision rules. The authorization of these rules are managed through the browser-based user interface (HTML forms), and can be specified in different ways within the business processes.

Dynamic Case Management. Within this functional capability, different subcase levels within business processes can be specified and managed. A *case* can be defined as a particular business transaction that is desired to be completed, for example, a restaurant reservation or processing an insurance claim. Each case goes through multiple stages and mainly consists of processes and several steps/tasks that can deal with multiple business functions within an organization. During the execution of a case, the status is changed after each stage/step. In addition, a *case type* consists of multiple instances that are called cases. Each case type follows a certain life cycle and can contain detailed information on the corresponding cases. From each case, the complete history can be saved. Furthermore, without necessarily changing data, cases can be reassessed multiple times within a business process. Next to structured process, also unstructured processes and ad-hoc processes are supported. Hence, different types of business processes (cases) can be handled.

Decision Management. By means of predictive models, business processes and customer experience can be improved. For this, relevant data is analyzed in order to find repeatable patterns. Both internal and external predictive models can be loaded into Pega's Decision Management.

Presentation and User Interface. The graphical user interface can be dynamically designed by means of a model-driven approach. It is possible but not required to write customized programming code, and

different interfaces, such as HTML 5 for internet browsers. The Pega Platform itself can be accessed across different platforms, including mobile devices.

Next to the functional capabilities above, Pega is compatible to work together with many types of systems *(Integration and Transformation)*, the use of the Pega Platform across different organizational levels can be managed *(Operational Management)*, user authorizations can be specified *(Security Services)*, and CPU, memory and related systems performance can be monitored *(Performance Monitoring)*.

5.2.2 Technical architecture

In Appendix A, a comprehensive overview of the technical architecture can be found. Though, the exact structure of the technical architecture depends on the implementation environment. Therefore, in Figure 22, a simplified overview of the most important elements of the technical architecture is depicted. This architecture model has been created by means of a conversation with the daily supervisor (M. Bussemaker, personal communication, July 4, 2018).

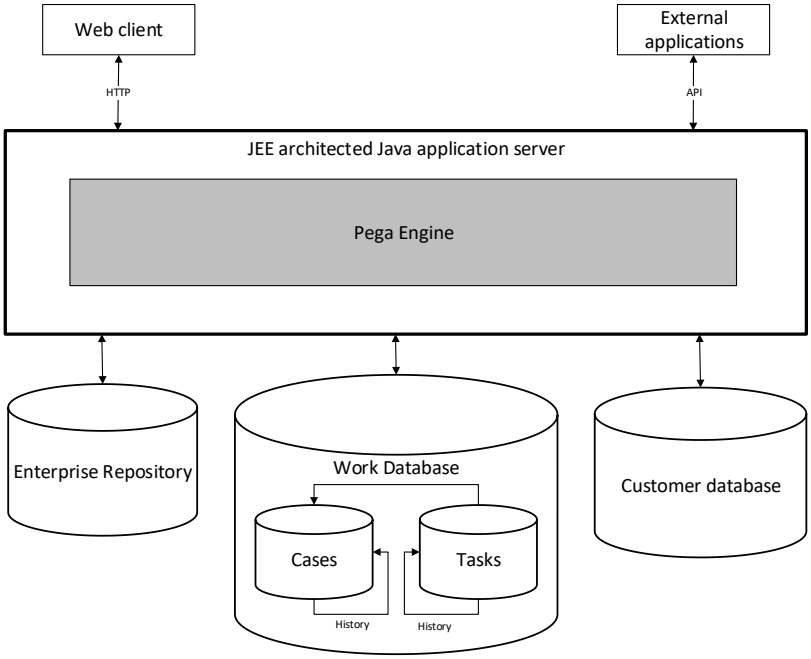


Figure 22: Pega Platform technical architecture (simplified)

Basically, the Pega Platform has a multi-tier architecture that consists of a Client Tier (= presentation/UI layer), App Tier (= business logic layer), and Data tier (= data layer), and can be scaled both horizontally and vertically. The core engine of the Pega Platform is the Pega Engine that runs on a JEE architecture Java-based application server on premise (locally) or in the Cloud. The Pega Engine runs the aforementioned functional capabilities. Regarding web services, a web client accesses the Pega platform through a Hypertext Transfer Protocol (HTTP) connection via an internet browser. In addition, Java configurations, XML scripts, SOAP, and other technical interfaces related to web services can be created and integrated automatically. For the communication and integration with external applications, APIs can be configured. By means of creating and executing models, XML scripts are generated.

Data can be stored in, for example, a SQL database. There is a so-called Enterprise Repository which contains the business rules, process definitions, execution logs, worklists, user interface setting and other required properties of a Pega application. Within the Work Database, there is a distinction between cases and tasks. Despite the fact that cases and tasks are quite similar, there is an important difference. A case must be seen as a dossier that contains structured information about a certain process, whereas executing a task effects the content of a case. In addition, tasks can be applicable to multiple cases. Historical data of both cases and tasks is kept up as well. Next to the Work Database and Enterprise Repository, it is also possible to integrate a Customer Database, which is a database that already exists within the organization where the Pega Platform is implemented.

5.2.3 Mapping with the Workflow Reference Model

When looking at the extent to which the architecture (both functional and technical) of the Pega Platform is structured based on the Workflow Reference Model (see Figure 9), it can be said that, apart from different names and merged and/or separated parts, the Pega Platform is compliant to the properties of the Workflow Reference Model. For example, the Workflow Reference Models shows separated Process Definition Tools and Administrated and Monitoring Tools. The functionalities of these tools are all integrated into the Pega Platform by means of the aforementioned functional capabilities. Therefore, the Pega Platform should be seen as an extensive BPMS (to be exact, a case management system) that provides more than just the regular BPMS functionalities.

5.2.4 Situational layer-cake structure

A specific property of the Pega Platform is the so-called *situational layer-cake* structure for organizing the case types, cases, data models, process definitions etc. of an application. They are organized in such hierarchic way (parent-child relationships) that it is easy to reuse application components and apply the corresponding configurations to multiple (sub) cases. This reduces the complexity of the application development. In addition, changes can be done rapidly in order to respond to changing situational factors. Furthermore, the situational layer-cake ensures that a user only gets information about a certain case (a piece of the whole cake) that are relevant for that particular situation (Pegasystems, 2018b).

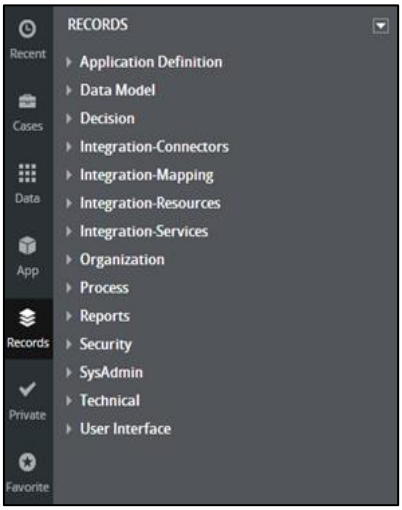


Figure 23: Application properties in Pega

In addition to the layer-cake, in Figure 23, a screenshot of the (automatically) standard configurable properties of an application built on the Pega Platform are shown. The main properties are:

- *User interface and Process*, which contain the workflows of the case life cycles within the application;
- *Decision*, which contains the business rules that are used within the process workflow as part of the decisions points.
- *Data model*, which contains the data types within the corresponding data models that are the required.

This main structure of a Pega application is related to the so-called 3-tier architecture, mentioned in paragraph 4.3.2. In this case, *process* and *user interface* are the presentation layer, *decision* deals with the business logic, and *data model* is aimed at the data layer. On the Pega Platform, the business logic of an application is automatically generated by means of the model-driven approach, including the manual configuration of the properties of the application.

5.3 Implementation approach

Implementing the Pega Platform, including the development of business applications on the platform, involves positioning the Pega Platform and its architecture within an organization's application landscape. Therefore, in this paragraph, it is described what implementation approach is applied by BPM Company. In addition, it is described how an example business process would be configured on the Pega Platform as part of the design and development of a business application.

5.3.1 High-level implementation approach

The development and implementation of applications on the Pega Platform is divided into multiple steps. It varies per project what is done during each step due to different contexts, requirements etc. However, in most cases, the following steps are taken (A. Wiegman, personal communication, August 16, 2018):

1) Identification of applicable business functions and corresponding business services. This step aims at the Enterprise Architecture level;
2) Definition of the applications functions, based on the applicable business functions. This step aims at the Domain Architecture level;

3) Creation of the Project Start Architecture / business analysis + technical application for the specification of the required information per business function. This step aims at the Solution Architecture level;
4) Pega design and development based on user stories and outcomes of the previous steps. For this, the so-called *Journey Centric Development Methodology* is applied;
5) Optional GAP analysis between the PSA and realized product.

During the steps above, multiple architecture description documents can be created. However, most of the times, architecture documents are not created (completely). To a certain extent, the intended ADL will foster the creation of the architecture documents regarding the communicating flows within the application landscape of the Pega Platform. An example of a document is the project start architecture (PSA) that is created during the first phase of a project.

Different viewpoints are created and described. This involves multiple abstraction levels, for example, both a high-level view on all application, including the data that that be gathering from these applications, and a view on the data attributes inside each database. Currently, mainly UML is used to create the applicable architecture models.

5.3.2 Journey Centric Development Methodology

The Journey Centric Development Methodology is a model-driven rapid delivery implementation methodology that is applied when developing applications on the Pega Platform. The methodology is based on the agile Scrum development methodology and adheres to an outcome-based approach. Case lifecycle management serves as the foundation of the application development of the Pega Platform. In practice, the four stages of the methodology are followed in a continuous cycle as depicted in Figure 24 (Pega Academy, 2018).

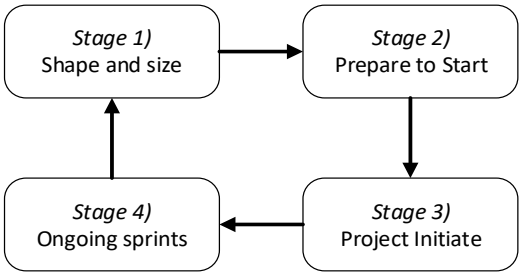


Figure 24: Stages of the Journey Centric Development Methodology

Within a development team, the main stakeholders are the product owner, Business architects (BAs), System Architects (SAs), and Subject Matter Experts (SMEs). The BAs are mainly responsible for the identification and description of service level agreements, business rules, use cases and features. The SAs are the developers. The SMEs provide the BAs and SAs with subject matter / domain-specific knowledge, for example, knowledge for the translation of the business processes to the required features within the application. The product owner represents the user of the application.

> First, during *Stage 1*, based on the identified case types and prioritized backlogs in the backlog, a roadmap for the future releases is created during the kick-off meeting. The case types are identified together with the creation of the user case backlog by means of the so-called Direct Capture of Objectives (DCO) that fosters the application delivery time. DCO captures the business requirements directly within the application based on a shared model. For the first release, the minimum required capabilities are determined and developed which results in the so-called Minimum Lovable Product (MLP), a Pega-specific term for the Minimum Viable Product (MVP). Ideally, the MLP is released within 90 days, divided into multiple sprints. For BPM Company, a sprint usually lasts two weeks.
>
> Then, in *Stage 2*, the development team is made before the project can start. This includes logistical and legal activities that are carried out.

After the start of the project, in *Stage 3*, the user stories on the backlog are groomed for the definition of ready by means of DCO. This includes defining the acceptance criteria per user story. In addition, the goal and approach of the development project are discussed within the team.

After this, at the end of *Stage 4*, a complete incremental case type as a working software component has been tested and delivered at the end of each sprint. In other words, a new release is delivered. On each day, there is a stand-up meeting with all relevant stakeholders. Before the start of a new sprint, the backlog is refined based on the previous sprint. Eventually, when all sprints of a release have been finished, a new release is developed by starting again at *Stage 1*.

On the Pega Platform, both happy flows and alternative flows (based on business rules) of processes are modelled and specified. The process flows are visualized by means of an implicit BPMN-based workflow modelling language. By means of the so-called Agile Workbench of Pegasystems, agile Scrum related tasks can be performed, such as progress tracking and managing the product backlog. Furthermore, built-in reusable best practices and guardrails foster the speed, efficiency and successfulness of each development process (Pega Academy, 2018).

5.4 Short Pega application example

In Figure 25, a small screenshot shows how the life cycle / workflow of the payment claim approval process of a fictional running example case described below would be configured as a so-called *case type* within the Pega Designer Studio environment. In this way, we briefly describe and visualize the basics of the application development on the Pega Platform through case lifecycle management. In addition, several key Pega terms such as case and case type are described in more detail.

Consider a car insurance company that regularly receives online payment claims from their customers. The claims are processed by means of a BPMS and other integrated systems that all run on an application server.

When a customer needs to be paid due to car damage caused by another car driver, a payment claim is created and sent via the company's website. After the claim has been received, it is fully registered by an employee. For this, certain information from a separated CRM system is required. Then, another employee checks whether or not the claim is correct based on predefined requirements. For this, a separated DMS is accessed. After the assessment, a message of the decision about the claim is sent to the customer. If the claim is approved, the claim is fully processed in order to let the customer receive the desired payment. If the claim is rejected by the company, only a message about the rejection is sent to the customer.

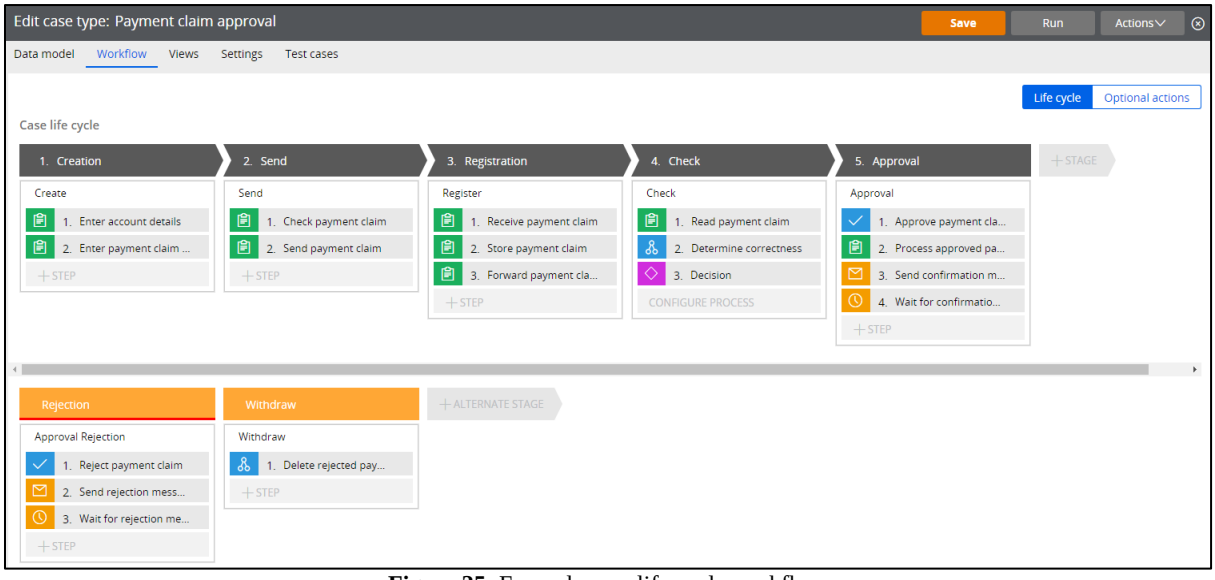


Figure 25: Example case life cycle workflow

In this example, the payment claim is called a *case* that is either approved, or rejected and withdrawn at the end of the *life cycle* (workflow). The case type / workflow has been divided into five normal *stages*. During each stage, one or more *processes* and *steps*, for example, *Send payment claim* in the *Send* stage, are carried out in order the reach the next stage. The life cycle of the payment claim contains one decision point at stage 4. Eventually, instead of an approved payment claim, it is also possible that the payment claim goes to the alternative stages based on certain business rules. This means that the payment claim has been rejected and withdrawn.

For each stage, the underlying processes and different types of steps (automated, manual etc.) of the workflows, including decision points, can be modelled and specified by means of the BPMN-based shapes. As an example, in Figure 26, the workflow model of the Approval stage is shown.

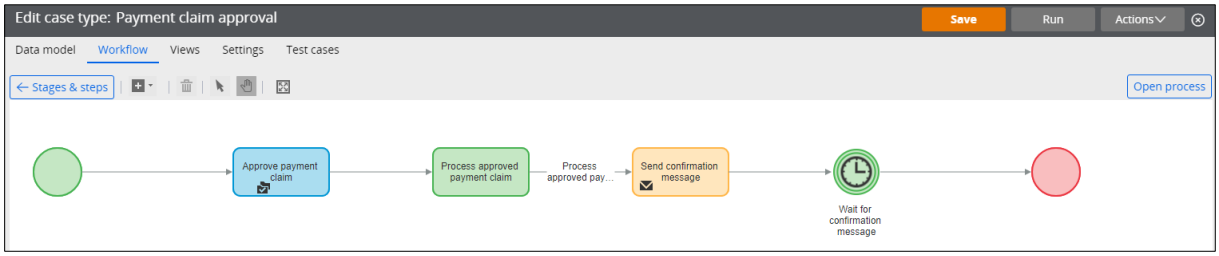


Figure 26: Pega's BPMN-based workflows - Approval stage

For each step, it can be specified whether it is carried out automatically or if human input is required, and what information is required. Next to this, business rules for the decision points, underlying data models, API's for the communication with other systems, user authorizations, the user interface and many other properties of the application can be configured and specified in the desired ways without extensive manual coding (low-code).

5.5 Summary

To describe the role of a BPMS within an application landscape (SRQ1) in the real world, and in order to clarify what has to be taking into account (considerations) when the requirements for the design and practical application of the ADL are determined (SRQ2), we have examined the Pega Platform. This was done by means of a desk research on relevant Pega material, several courses from the Pega Academy, and a few preliminary/explorative interviews.

SRQ1) *What is the role of a BPMS within an application landscape?*

BPM Company uses the Pega Platform of Pegasystems for low-code application development. More precisely, for their customers, they develop business applications that run on this BPMS. The Pega Platform must be seen as a comprehensive BPMS that contains multiple functional capabilities, including business process management, and a development environment / compiler. By means of the model-based configurations, the software code is generated automatically. This also involves the configuration of communication between the Pega Platform and external invoked applications, and the use of relevant interfaces and web services. From a technical perspective, the Pega Platform runs on a Java-based application server (locally or Cloud), and is accessed via an internet browser. The Pega Platform applies the so-called *situational layer-cake structure* for structuring and reusing all components of a business application. Furthermore, we have indicated that the architecture of the Pega Platform is compliant to the elements of the Workflow Reference Model (see Figure 9).

SRQ2) *What needs to be considered when designing a process-oriented ADL for specifying communication flows in BPMS application landscapes?*

Besides the architecture, we have also described the development/implementation approach of the Pega Platform. This approach is called the Journey Centric Development Methodology and is based on Agile Scrum. The intended ADL should support the different steps of the Journey Centric Development Methodology. Especially, in terms of the creation of architecture descriptions, such as the project start architecture (PSA). It is important to consider different level of abstractions (architecture layers) and granularity regarding the specification of communication flows. Furthermore, we used a small running example to show how the workflow of a case type would be configured on the Pega Platform.

6. Design and specification of the ADL

This chapter elaborates on the process of designing the main artefact of this research: the process-oriented ADL for specifying communication flows in BPMS application landscapes. We describe how the intended ADL is designed based on both the literature and practitioners. Furthermore, we elaborate the main properties and purposes of the ADL, as well as the guidelines for applying the ADL in practice. Eventually, the following SRQs have been answered completely:

- ***SRQ2)*** *What needs to be considered when designing a process-oriented ADL for specifying communication flows in BPMS application landscapes?*
- ***SRQ3)*** *What are the characteristics of the ADL?*
- ***SRQ4)*** *How can the ADL be applied by practitioners?*

6.1 Selection criteria and requirements

We create the intended ADL based on suitable existing ADLs (the so-called candidate ADLs) by means of utilizing the Method Association Approach (MAA). Regarding the MAA steps, up to this point, we have identified project situations (step 1) by means of several preliminary/explorative interviews, and desk research on the architecture and development/implementation approach of a BPMS that is applied in practice (the Pega Platform). Hence, we can formulate feature groupings (step 2), in other words, the criteria and corresponding requirements of the intended ADL that we use to select and analyze/compare suitable existing ADLs. The majority of the criteria and corresponding requirements is formulated based on the literature. So, based on the main building blocks, common ADL properties, and requirements of an ADL that we described in paragraph 4.4. In addition, our desk research on the Pega Platform also supported step 1 and 2 of the MAA.

In Table 5, we specify the formulated criteria and the corresponding properties/values (the requirements). We briefly explain why the criteria and their properties have been chosen, as well as which properties are most applicable to the design of the intended ADL.

Table 5: Overview and description of the selection criteria

<table>
<tr><th>Selection criteria</th><th colspan="2">Properties/values (=requirements)</th></tr>
<tr><td rowspan="4">Syntax and semantics</td><td colspan="2">Graphical</td></tr>
<tr><td colspan="2">Textual</td></tr>
<tr><td rowspan="3">Formality</td><td>Formal</td></tr>
<tr><td>Semi-formal</td></tr>
<tr><td>Informal</td></tr>
<tr><td colspan="3">An ADL has a syntax and semantics. It is preferred that the syntax and semantics of the intended ADL are both graphical and textual, because architecture models need to be created and specified. Depending on the extent to which advanced analytical/mathematical capabilities are relevant, it might not be necessary that the intended ADL has a formal syntax and semantics. However, to reduce ambiguity, the candidate ADLs must be at least semi-formal.</td></tr>
<tr><td rowspan="5">Viewpoints</td><td colspan="2">Context viewpoint</td></tr>
<tr><td colspan="2">Functional viewpoint</td></tr>
<tr><td colspan="2">Information viewpoint</td></tr>
<tr><td colspan="2">Concurrency viewpoint</td></tr>
<tr><td colspan="2">Development viewpoint</td></tr>
<tr><td colspan="3">The intended ADL must support at least the five viewpoints that are listed above, which were described in the chapter 4. Each viewpoint focuses on certain architectural properties/elements:
• Context viewpoint: business functions, organization structure, use cases;
• Functional viewpoint: application components, interfaces, (web) services;
• Information viewpoint: business processes, information/data structure, choreographies, scenarios;
• Concurrency viewpoint: concurrencies, system states;
• Development viewpoint: standardization, code lines.</td></tr>
</table>

The deployment viewpoint and operational viewpoint are less relevant because they focus on a system that is running live in a certain environment. We mainly focus on the actual application development on a BPMS in design time.		
Abstraction levels	**System-Aggregation**	*(0): no view on the system (= blackbox)*
		(1): high-level view of the functional components (e.g. an application server)
		(2): view inside the functional components (e.g. modules and interfaces of an application server)
		(3): low-level view of the functional sub components (e.g. components of an server engine)
		(4): very low-level view on functional detailed sub-components (e.g. more details on the components of an server engine)
	This dimension aims at the level of detail (granularity) regarding functional components that can be modelled and specified. Level (0) contains the least details, whereas level (4) focuses on the most details.	
	Data-Aggregation	*(0): no data-related components*
		(1): high level view of the data-related components (e.g. a database)
		(2): general view inside the data components (e.g. smaller data stores within a big database)
		(3): low-level architectural view on the entities (e.g. an ERD of particular data stored in a database)
	This dimension aims at the level of detail (granularity) regarding data-related components that can be modelled and specified. Level (0) contains the least details, whereas level (3) has the most details.	
As described by Poumirza et al. (2017), different levels of elaboration / abstraction levels regarding different dimension can be considered when specifying an architecture. Therefore, it is relevant to know at what level of detail an architecture can be specified by means of each candidate ADL.		
Architecture styles	*Component-based*	
	Layered style	
	Client/server (n-tier)	
	Service-oriented architecture (SOA)	
The most common architecture style are the component-based, layered style, and the client/server (n-tier). In addition, due to the scope of this research, the service-oriented architecture (SOA) style is also relevant.		
Architectural purposes	*Creation*	
	Analysis	
	Refinement	
	Validation	
	Modelling and describing distributed systems	
	Common architecture links	
	Design decisions capturing	
	Tool support	
These are several tasks that can/must be supported by an ADL. For the intended ADL, the most relevant purposes are: creation, refinement, validation, modelling and describing distributed systems, and common architecture links. Analytical capabilities are less important due to the fact that the ADL is not going to be used for advanced formal analytical purposes.		
Components	**Business process related**	*Inter-processes*
		Intra-processes
	Software architecture related	*Static structure*
		Dynamic structure

<table>
<tr><td colspan="3">Due to the desired scope of the ADL, both business related components for inter-processes and/or intra-processes, and software architecture related components (static and/or dynamic structures) need to be supported. Therefore, the most suitable existing ADLs are analyzed regarding the presence of both types of components. It is preferred that the ADL can be used to model and specify both inter-processes and intra-processes.</td></tr>
<tr><td rowspan="3">Connectors</td><td colspan="2">Information/data flows</td></tr>
<tr><td colspan="2">Message flows</td></tr>
<tr><td colspan="2">Interfaces (API)</td></tr>
<tr><td colspan="3">Due to the desired scope of the ADL, it must be possible to model and specify at least interfaces (API), information/data flows and message flows. Other connector types (flows) are less relevant.</td></tr>
<tr><td rowspan="3">Configurations</td><td rowspan="3">Category</td><td>Implicit configuration (based on interconnection information that is spread over definitions over separated components and connectors)</td></tr>
<tr><td>Explicit configuration (components and connectors are modelled separately from the configurations)</td></tr>
<tr><td>In-line configuration (explicit configurations, including specifications of component interaction protocol)</td></tr>
<tr><td colspan="3">According to Medvidovic and Taylor (1997), a language that apply implicit configurations cannot be strictly called ADLs. Regardless of this fact, those language can be suitable as well for the design of the intended ADL.</td></tr>
</table>

6.2 Selection and comparison analysis of candidate ADLs

By means of the specification of the selection criteria and corresponding requirements above, we can examine relevant literature in order to select and compare candidate ADLs.

For certain reasons, we have excluded many ADLs from the comparison analysis. One of the reasons is the fact that most ADLs mainly focus on technical software aspects regarding system engineering and hardware components (buses, processors, ports etc.) of a system, whereas the intended ADL is mainly focused on the business processes and business functions (process-oriented), and partly on software engineering. Secondly, several ADLs are more or less not applicable in practice and/or not updated/supported (anymore), such as SADL (Malatova, Lago, Muccini, Pelliccione, & Tang, 2018). This results in a lack of clear reliable insights into the practical applicable of these outdated ADLs.

Another reason why we have excluded ADLs is their purpose/scope. Most ADLs are meant for describing certain types of (technical) software systems and, therefore, consists of elements that are too domain-specific. Thus, the majority of the notation of these ADLs is not applicable within the scope of the intended ADL. A good example is AADL for real-time performance critical systems (Feiler, Gluch, & Hudak, 2006). In addition to this, there are many ADLs which are basically extensions of UML (so-called UML profiles. Most of these UML profiles are not relevant for the intended ADL. This includes SoaML for service-oriented architectures (Object Management Group, 2012), because, in comparison to UML, SoaML does not provide any relevant additional diagrams or notations, or provide less elements in comparison to UML.

Furthermore, some ADLs are quite similar / are duplicates. For instance, both the event-driven process chain (EPC) and BPMN can be used to model business processes at a similar abstraction level. However, in comparison to BPMN, EPC is less comprehensive and, therefore, it is not relevant to be included to the analysis (Dumas et al., 2018).

Regarding the concurrency viewpoint, Petri Nets are quite useful for the specification, visualization, and mathematical analysis of the concurrency, the states of run-time elements and communication transitions of a system. This is aligned with the corresponding process flows and business logic that also specify the process communication structures (Rozanski & Woods, 2012). However, due to the fact that the intended ADL does not focus on the transitions of run-time elements, which is more specific than communication flows, we have also excluded Petri Nets from the comparison analysis.

Eventually, we have selected the following candidate ADLs:

- *Business Process Model and Notation (BPMN);*
- *ArchiMate;*
- *Unified Modelling Language (UML).*

In the following sub paragraphs, these candidate ADLs are described and analyzed in such way that the values in Table 7 of the requirements of the criteria have been reasoned. Based on the comparison analysis, we determine to what extent each candidate ADL could support the creation of the intended ADL. In other words, what properties of the candidate ADL can be added to the properties of the intended ADL. In this way, we clarify why we have selected the three aforementioned candidate ADLs.

To visualize the properties of the candidate ADLs, we use again the fictional running example case below about the car insurance company, which we already used in paragraph 5.4. In this way, we can identify both the benefits and limitations of the candidate ADLs. At the same time, several components and connectors of each ADL are visualized. Basically, in this running example, there are two organizations/actors, which are the customer and car insurance company. They communicate with each other regarding the payment claims. There are two employees that carry out the activities that involve multiple information flows, and decision points. Besides the BPMS, at some points, also other systems are assessed. By means of each selected ADL, the described process is modelled. There might be some parts that cannot be modelled in a single view. Moreover, it is also possible that some parts cannot be modelled (completely) in the desired way. Such constraints are indicated as well.

Consider a car insurance company that regularly receives online payment claims from their customers. The claims are processed by means of a BPMS and other integrated systems that all run on an application server.

When a customer needs to be paid due to car damage caused by another car driver, a payment claim is created and sent via the company's website. After the claim has been received, it is fully registered by an employee. For this, certain information from a separated CRM system is required. Then, another employee checks whether or not the claim is correct based on predefined requirements. For this, a separated DMS is accessed. After the assessment, a message of the decision about the claim is sent to the customer.

If the claim is approved, the claim is fully processed in order to let the customer receive the desired payment. If the claim is rejected by the company, only a message about the rejection is sent to the customer.

6.2.1 Candidate ADL 1: Business Process Model and Notation (BPMN)

BPMN is one of the standard notations for business process modelling. The most recent version (2.0.1) was released in 2013 (Object Management Group, 2013).

Syntax and semantics

BPMN has a semi-formal syntax and semantics that is both graphical and textual. The graphical notation of BPMN consists of different shapes/symbol with specific meanings, not just general box and lines. BPMN is designed in such standardized way that it can be easily understood by both business stakeholders and technical stakeholders, especially business analysts and developers.

Viewpoints and abstraction levels

The core of BPMN is process modelling, which is related to the UML activity diagrams. By means of BPMN, only concepts that are related to business processes can be modelled and specified. The creation of organizational models, business rules models, data flow models, and functional models is excluded from BPMN. Below, the types of BPMN sub-models (the viewpoints) are briefly explained:

- *Processes (public and private)*: the well-known and frequently used BPMN business process diagrams (BPD). Private processes represent intra-processes that are carried within one particular organization. Public processes entail interaction between two processes of two different organizations, so, inter-processes;
- *Choreographies*: represent the interactions between two or more process participants within a process. They explicitly show message exchanges as activities;
- *Collaboration:* shows the detailed communication between two or more business entities, for example, a doctor and a patient;
- *Conversation* diagram: can be used to informally visualize interactions between process participants within a single process.

Based on the description of the BPMN sub-models above, it has been indicated that BPMN mainly focuses on the *information viewpoint* due to the fact that the models show how information is stored, processed, and within processes shared across (an) organization(s). Since this partly deals with software behavior, BPMN partly supports the *concurrency viewpoint* to a certain extent. The specification of technologies (applications) is out of scope, because BPMN does not provide the ability to model software components, behavior, and technical infrastructure components, such as a software module and an application server. Regarding business process abstraction levels, BPMN is focused on both the M1 and M0 level (Smirnov, Reijers, Weske, & Nugteren, 2012).

Components, connectors and configurations

BPMN consist of the following types of elements that together are the components and connectors for creating a BPMN model:

- *Flow objects*: these are the core elements of BPMN that are used to model a business process's behavior by means of *activities*, *gateways* (decision points) and *events (trigger)*.
- *Connecting objects*: two flow objects can be linked by means of a *sequence flow* (show the order two activities are carried out), *message flow* (for modelling the communication links between different processes), *association* (represents an information flow*)*, or *data associations.* Data flows cannot be modelled.
- *Swim lanes*: a single organization is modelled as a *pool* that consists of one or more *lanes* representing process participants that are involved in the business process.
- *Artifacts*: represent some additions to a BPMN model, and can be a *group* or *text annotation*.
- *Data:* a data element can be a *data input*, *data output*, *data object*, or *data store*.

Based on the elements above, it is indicated that by means of the business process related components, both inter-processes and intra-processes can be modelled. Regarding software architecture related components, only data-related components (= static structure) can be modelled. Application components cannot be modelled and specified. Though, BPMN provides process interfaces that are in fact collections of operations provided by services needed so that a certain process can be used by other processes and corresponding services. Furthermore, information flows, message flows for inter-processes, and other connecting objects can be modelled.

Regarding configurations, BPMN is an implicit configuration language due to the fact that the configurations cannot be explicitly derived from a single model. Instead, the configurations can be acquired from the interconnections between the components and connectors and their meanings. This means that BPMN cannot be called a strict ADL (Medvidovic & Taylor, 1997).

Architecture styles

Since BPMN cannot be used to model software components and the interrelation between different architecture layers by means of services, the component-based style is not support. Furthermore, neither the layered style, nor the client/server (n-tier) and service-oriented architecture (SOA) are supported.

Architectural purposes

Regarding the business process level of an architecture, BPMN can be used to create business process models. This can be done within several tools that can validate the syntax applied of the created models, for example, Eclipse[2]. Also, stepwise architecture refinement is partly supported. Apart from analysis business process performance and related metrics, analytical purposes based on architecture-level information on completeness, interoperability etc. are not provided. Modelling distributed systems is partly supported in terms of the possibility to model inter-processes. Regarding common architecture links, it is possible to model and link a sub process separately from a larger corresponding process model. Furthermore, only general annotations can be used for design decisions capturing.

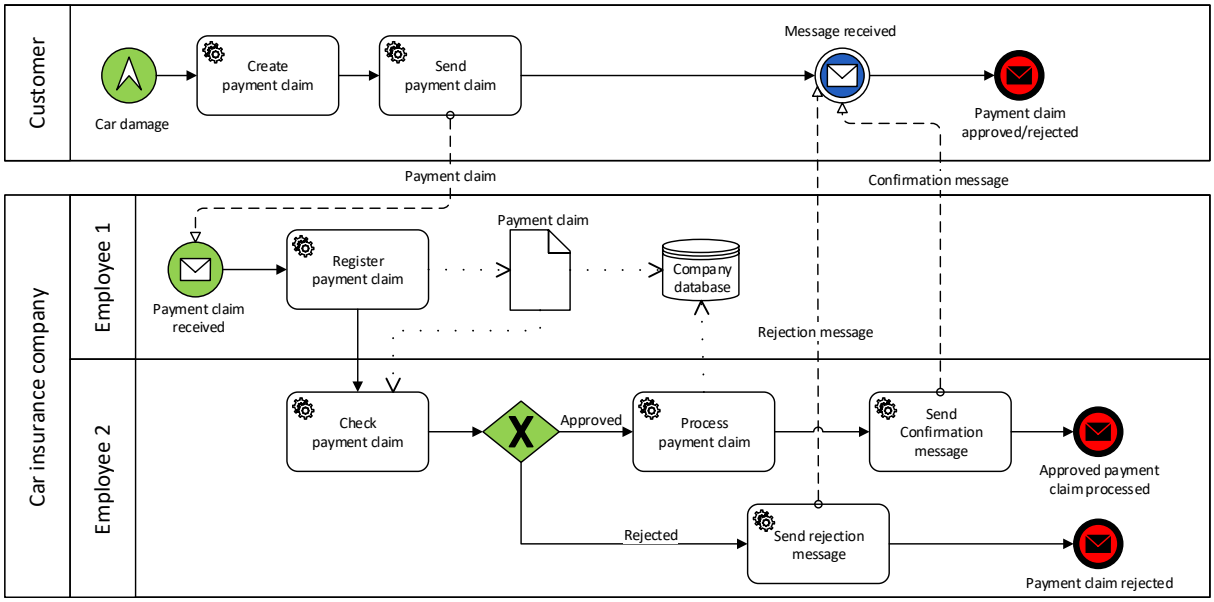


Figure 27: Example case - BPMN business process diagram (BPD)

EXAMPLE

In the BPMN business process diagram (BPD) in Figure 27, the fictional running example case has been modelled. The car insurance company, including the employees, and customer have been modelled separately by means of pools and swim lanes. The green circled shapes represent start events, whereas the red shapes are end events. The rectangles represent the process activities that are supported by the BMPS, DMS and CRM system. However, these systems and corresponding interfaces could not be modelled. Instead, the gear symbols indicate that the activities are carried out by means of a system. In addition, a data object (payment claim) is depicted, as well as a data store representing the company's database and several information flows have been modelled. Furthermore, there is one decision point that is represented by the green diamond shape, and three message flows between the customer and the company are shown.

[2] https://www.eclipse.org/bpmn2-modeler/

CONCLUSION

Despite the fact that BPMN is more a notation for process modelling rather than a strict ADL, BPMN is a widely applied standard that is a suitable language for the design of the intended ADL due to the explicit process-oriented focus regarding the information viewpoint. BPMN has many properties that can serve as a solid base for the creation of the intended ADL. Next to the well-known BPMN process models, also the BPMN choreographies, collaborations and conversations are useful. However, due to the fact that BPMN is explicitly meant for business process modelling, it is not explicitly focused on software architecture modelling. For example, it does not support the creation of explicit software architecture models representing software components, connectors, and interfaces. Moreover, there is not a certain BPMN shape which explicitly represents an application component.

The two ADLs below are complementary to BPMN. They can be used to elaborate certain BPMN elements in a different way as an extension of a single BPMN model. In other words, these languages can be considered as a specialization / additional viewpoints of BPMN. For this reason, these languages were not analyzed separately. Instead, they are briefly described below.

Decision Model and Notation (DMN)

Regarding business logic / business rules, a BPMN process model can be extended in separated models and/or simplified by means of the Decision Model and Notation (DMN). DMN can be used to create decision tables and decision trees for specifying a certain decision point. For the fictional example, *Check payment claim* is a decision point which involves whether or not a new payment claim meets certain business rules. In other words, the requirements of an approvable claim. Modelling this decision point explicitly might result in lots of gateway and possible flows, and thus a complex BPMN process model (Object Management Group, 2016a).

Case Management Model and Notation (CMMN)

In addition to this, Case Management Model and Notation (CMMN) is a graphical language for representing the life cycle of a case. The shapes of CMMN are complementary to BPMN shapes. The most important difference is the fact that a CMMN model is more implicit, whereas a BPMN process diagram is a more explicit representation of a business process. The reason for this is the fact that CMMN is mainly suitable for modelling business processes of a case that are mainly influenced and changed due to changing situational events. Usually, such processes do not need a predefined execution sequence and/or are not continuously repeated. The most important unique CMMN elements are a *stage* of a case, *case file* (similar to a BPMN data object), and a *milestone*. Similar to BPMN, CMMN distinguishes different types of tasks (Object Management Group, 2016b).

6.2.2 Candidate ADL 2: ArchiMate

ArchiMate is a modelling language for enterprise architecture. It can be used to design/document and analyze an organization's enterprise architecture. In 2017, version 3.0.1 was released by The Open Group (The Open Group, 2017).

Syntax and semantics

ArchiMate has a semi-formal syntax and semantics that is both graphical and textual. Different types of shapes, symbols and colors are provided to model and distinguish different architecture layers and domains.

Viewpoints and abstraction levels

ArchiMate links concepts between different architecture domains. For this, ArchiMate distinguishes three architectural layers: *Business*, *Application*, and *Technology*. Each layer corresponds to several viewpoints and views. This means that, besides business processes and corresponding actors, services, organization models etc., also the application landscape, interfaces and the technical infrastructure of an organization can be modelled. All software architecture viewpoints, specified in Table 3, are supported, except the concurrency viewpoint and development viewpoint. When looking at the exact viewpoints of ArchiMate, taking into account the scope of this research, the most relevant viewpoints are: the layered viewpoint, business process cooperation viewpoint, application viewpoint, application collaboration viewpoint, and application usage viewpoint.

Regarding abstraction levels, modules and sub modules can be modelled with several viewpoints (level 2). This also means that high-level concreteness regarding the names of components can be applied. Databases, data objects etc. can also be modelled (level 1). However, representing more detailed data-related components, for example, by means of an entity relationship diagram (ERD), are not supported.

Components, connectors and configurations

An ArchiMate *model* consists of multiple *concepts*, which can be a component (*behavior, structure, motivation, composite*) or a *connector (different types of relationships/arrows)* between them. The ArchiMate Framework represents the building blocks of ArchiMate. This framework shows the three aspects of ArchiMate across the three architectural layers:

- *Active structure:* entails the architectural elements that somehow have some behavior within the architecture, for example, the actors and applications that carry out activities.
- *Passive structure:* information objects and data objects that are used during some behavior of an active structure element. For example, an invoice document that is used by an employee from an insurance company.
- *Behavior aspects:* the business processes, services, events etc. that are performed by active structure elements, like an actor that is assigned to a particular business process.
- *Motivation aspects:* by means of these aspects, reasoning on architecture design (rationale) can be modified and specified.

Hence, both the static structure and dynamic structure of a system can be modelled in ArchiMate. Regarding the business process related components, only intra-processes can be modelled and specified due to the absence of message flows and other related connectors for inter-processes. Though, ArchiMate contains different types of relationships/connectors that represent a certain dependency between two modelled elements. Furthermore, there are ArchiMate shapes for modelling interfaces in a general way. ArchiMate distinguishes (1) business interfaces for the use of a business service by the environment (actors), (2) application interfaces for making application services available for other application components, and (3) technology interfaces that provide the technology services of nodes. These interfaces cannot be further modelled and specified in more details.

Regarding the configurations, ArchiMate is an implicit configuration language. The reason for this is the fact that the configurations are neither modelled separately from the components and connectors or by means of interaction protocols. Instead, the interconnections between the components and connectors and their meanings can be used to derive the configurations. Hence, similar to BPMN, ArchiMate cannot strictly be called an ADL.

Architecture styles

The component-based style, layered style (= ArchiMate layered viewpoint), and the client-service (n-tier) architecture style are provided by ArchiMate. Also, service-oriented architecture is supported due to the fact that services and applications can be modelled regarding different architecture layers.

Architectural purposes

There are several tools, for example Archi[3], which can be used to create ArchiMate models. The creation, analysis, refinement and validation of an architecture description is supported by ArchiMate. Different kinds of analytical capabilities (calculations) can be done by means of architecture-level information to a certain extent, such as the workload and availability of servers within a network. For design decisions capturing, only general annotations can be used. Regarding common architecture links, it is possible to model and link more detailed architectures that adhere to a larger/general architecture. Despite the fact that only intra-processes can be modelled, it is possible to specify the architecture of distributed systems to a certain extent. This is provided by means of a viewpoint that aims at the collaboration and interactions between separated applications.

EXAMPLE

In Figure 28, the ArchiMate application usage viewpoint regarding the example case is shown.

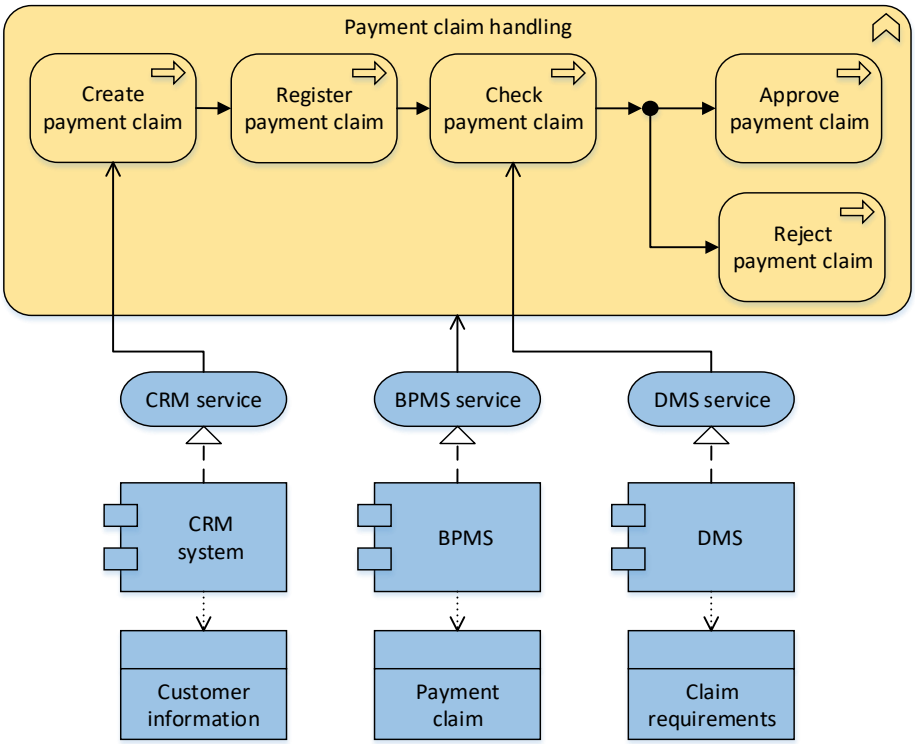


Figure 28: Example case - ArchiMate application usage viewpoint

In this viewpoint, it is modelled which systems and corresponding information objects are used. In the application collaboration viewpoint, it could be further specified how these systems interact with each other via interfaces. Similar to the pools and swim lanes of BPMN, a different ArchiMate viewpoint can be used to assign the actors to the activities they carry out. The message flows / interaction between the customer and company that occur during the business process could not be modelled.

By means of the so-called Layered Viewpoint, multiple architectural layers are shown in one model. Regarding the example case, it could be shown that the three systems run on an application server which is then modelled at the Technology layer.

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

CONCLUSION

Similar to BPMN, ArchiMate cannot be called a strict ADL. However, it does provide several capabilities that are suitable to involve in the design of the intended ADL. ArchiMate does support architecture creation, analysis, refinement, and validation to a certain extent. Business processes and all other corresponding architectural layers can be modelled and specified. In contrast to BPMN, ArchiMate also focuses on modelling and specifying applications, interfaces and technical infrastructure elements. However, regarding software architecture models, ArchiMate models are quite abstract. The exact software architecture of a particular application cannot be modelled. Also, in contrast to BPMN, inter-processes, including message flows, cannot be created with ArchiMate. Furthermore, there is no strict guideline regarding the required order of modelling the ArchiMate elements.

Penicina (2013) has mapped and linked corresponding concepts/elements of BPMN and ArchiMate with each other. This has resulted in the overview in Table 6.

Table 6: BPMN and ArchiMate element comparison (Penicina, 2013)

ArchiMate – business layer element	**BPMN element**
Business Process	Business Process Diagram, Pools, Lanes
Function	Task, Sub-Process
Business Interaction	Collaboration Diagram
Business Event	Event
Business Object	Data Object
Business Role	Lane
ArchiMate – application layer element	**BPMN element**
Application Function	Service Task, Script Task
Data Object	Data Object
ArchiMate – technology layer element	**BPMN element**
Device	Data Store
Artefact	Data Objects

This comparison between BPMN and ArchiMate shows that both languages have many common elements. Some are represented by similar shapes, for example, a data object, and some elements are only represented differently, for example, a business role (ArchiMate) and a lane (BPMN.) However, there are a few differences. Next to the absence of message flows in ArchiMate, which we indicated by means of the fictional example case, ArchiMate also does not provide the ability to model and specify the activities of a business process in much details similar to BPMN. Though, by means of ArchiMate, application components, servers and other elements related to software architecture can be explicitly modelled which is not provided by BPMN.

6.2.3 Candidate ADL 3: Unified Modelling Language (UML)

UML is a general modelling language that can be used to model a software system's design for software engineering purposes. In 2017, version 2.5.1 was released by the Object Management Group (Object Management Group, 2017a).

Syntax and semantics

UML has a semi-formal syntax and semantics which is both graphical and textual. Many types of software engineering related diagrams can be created by means of UML. For this reason, different aspects of a system can be modelled.

Viewpoints and abstraction levels

There are two types of UML diagrams:

- *Structural UML diagram*, like a component diagram and class diagram;
- *Behavioral UML diagram*, for example, an activity diagram and a use case diagram.

Each single diagram represents a particular viewpoint. For example, for the information viewpoint, a UML activity diagram can be used to model a business process similar to a BPMN process model. Another example is the UML deployment diagram for representing technical components as part of the deployment viewpoint, such as servers, databases and network links to represent the technical infrastructure of an organization.

Eventually, all viewpoints from Table 3 are supported by UML. In contrast to BPMN and ArchiMate, UML does support the development viewpoint by means of the UML package diagram. Regarding the abstraction levels of the viewpoints, both general and more detailed models can be created due to the many types of diagrams. In other words, all abstraction levels can be applied. Besides the UML activity diagram, also the component diagram, class diagram, object diagram, sequence diagram, and communication diagram are most for the design of the intended ADL.

Components, connectors and configurations

Basically, any type of diagram that can be designed by means UML consists of:

- *Elements/components,* for example, a use case diagram that consists of use cases and actors;
- *Connectors/relationships* between the elements, for example, the arrows between the actors and the corresponding use cases within use case diagram.

In addition, it is possible to add *notes/annotations* in order to clarify points that cannot be modelled and/or to foster design reason capturing. Instead of using annotations, it is also possible to make use of the so-called UML stereotypes which makes it possible to introduce new model elements to a certain diagram. In this way, for example, an UML profile can be created for specific domain-specific elements.

Regarding business process related components, both inter-processes and intra-processes can be specified to a certain extent. Namely, by means of a UML activity diagram, which is partly similar to a BPMN process model, it is not allowed to use message flows as part of the workflow of an inter-process within an activity diagram. For message flows, other diagrams are provided which have not a process-oriented visualization such as the sequence diagram. Due to both structural and behavioral UML diagrams, UML supports the specification of both the static structure and dynamic structure of a system. Furthermore, different types of flows can be used, including information/data flows, alongside with the specification of interfaces. For this, UML has a so-called InformationFlows package that provides the means for modelling information flows between systems at a general/high abstraction level.

Based on the configurations, UML is an implicit configuration language. This is indicated by the fact that configurations are derived from components and connectors, which means that they are not modelled explicitly and/or separately.

Architecture styles

Due to the variety of UML, many other architecture styles are supported. Application components can be modelled both component-based and in a layered style. Moreover, the client/server (n-tier) style can be applied, as well as the service-oriented architecture style.

Architectural purposes

The creation, analysis, validation and refinement of architectures are possible based on architecture-level information. Links between separated architecture that adhere to a certain reference architecture can be indicated. However, for this, explicit guidelines are not provided. Furthermore, distributed systems can be modelled, and only general annotations can be used for design decisions capturing. Many tools for creating UML diagram are available, such as Astah UML[4].

EXAMPLE

In Figure 29, a UML activity diagram of the example case is shown.

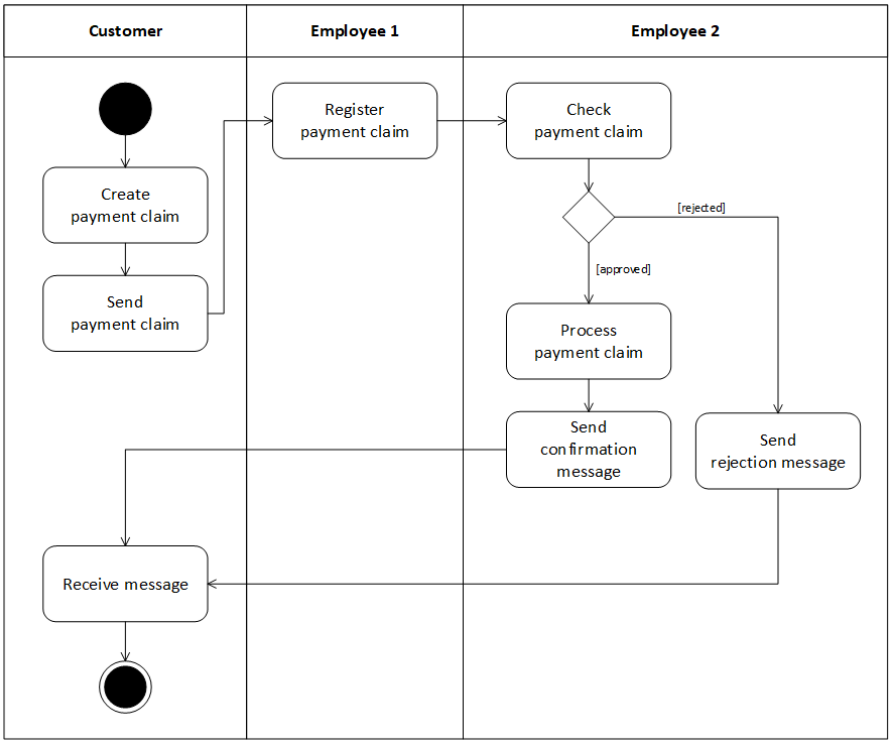


Figure 29: Example case - UML activity diagram

The UML activity diagram only shows the actors (by means of swim lanes), activities and the design point of the process. It cannot be modelled which systems are involved. Also, the information flows and message flows are not shown explicitly. For these purposes, more suitable UML diagrams are available, including the UML sequence diagram, and the UML component diagram.

CONCLUSION

UML has many properties that could be part of the intended ADL. There are multiple ADLs that have been designed based on / are extensions from UML, such as SysML. All properties of the criteria are provided by UML to a certain extent. Since both structural and behavioral diagrams can be created, both the static and dynamic structure of a system can be modelled and specified. This means that, as mentioned before, each software architecture viewpoint can be specified at multiple abstraction levels by means of the different types of diagrams. Regarding process modelling, BPMN and the UML activity diagram use similar elements at the same level of abstraction. This has been analyzed in more details by Geambaşu (2012). In addition, it is also possible to use the so-called UML stereotypes for adding new elements in order to create a UML profile for a certain domain.

[4] http://astah.net/editions/uml-new

Table 7: ADL comparison analysis results

Criteria	Requirement	BPMN	ArchiMate	UML	Interviews (n=5)	Focus groups (n=2)
Syntax and semantics	Graphical	✓	✓	✓	[5/5]	[2/2]
	Textual	✓	✓	✓	[5/5]	[2/2]
	Formality	Semi-formal	Semi-formal	Semi-formal	Semi-formal	Semi-formal
Viewpoints	Context	X	✓	✓	[4/5]	[2/2]
	Functional	X	✓	✓	[5/5]	[2/2]
	Information	✓	✓	✓	[5/5]	[2/2]
	Concurrency	O	X	✓	-	-
	Development	X	X	✓	-	-
Abstraction levels	System-Aggregation	X	2	4	Level 3	Level 3
	Data-Aggregation	1	1	4	Level 3	Level 3
Architecture styles	Component-based	X	✓	✓	[3/5]	[2/2]
	Layered style	X	✓	✓	[3/5]	[1/2]
	Client/server (n-tier)	X	✓	✓	[4/5]	[2/2]
	Service-oriented architecture	X	✓	✓	[5/5]	[2/2]
Architectural purposes	Creation	✓	✓	✓	[5/5]	[2/2]
	Analysis	O	O	✓	(2/5)	-
	Refinement	O	✓	✓	[5/5]	[2/2]
	Validation	✓	✓	✓	[5/5]	[2/2]
	Distributed systems	O	O	✓	[5/5]	[2/2]
	Common architecture links	O	O	O	[5/5]	[2/2]
	Design decisions capturing	Annotations	Annotations	Annotations	-	-
	Tool support	✓	✓	✓	[5/5]	[2/2]
Components	Business process related	Inter/intra	Intra	Inter/intra	Inter/intra	Inter/intra
	Software architecture related	Static	Static/dynamic	Static/dynamic	Static	Static
Connectors	Information/data flows	✓	✓	✓	[5/5]	[2/2]
	Message flows	✓	X	O	[5/5]	[2/2]
	Interfaces (APIs)	O	O	✓	[5/5]	[2/2]
Configuration	Category	Implicit	Implicit	Implicit	Implicit	Implicit

✓: provided, O: partly provided, X: not provided, -: mentioned as not relevant, [1/5] & [1/2]: mentioned as relevant 1x etc.

In Table 7, a summarized overview of the results of the comparison analysis of candidate ADLs is shown. ☑ indicates a requirement that is fully provided in a process-oriented way by the corresponding candidate ADL. Requirements marked with O are partly provided in a process-oriented way. X indicates a requirement that is not provided at all. Next to these literature review results, in the right two columns of the table, the results of both the semi-structured interviews and the focus groups have been included. These results are described in the next two sub paragraphs.

6.2.4 Semi-structured interviews

To acquire the practitioners' perspective on the requirements of the intended ADL, we conducted five semi-structured interviews with both Pega business architects and Pega system architects that each were involved in three different projects of BPM Company. In Appendix B, the interview protocol can be found which was used to assess the criteria and corresponding requirements of the ADL. All interviews were recorded. In addition, notes of the answers of the interviewees were taken. An important part of the interview protocol were the examples of architecture models of the candidate ADLs that were shown and discussed in order to determine the relevant parts of each candidate ADL, the preferred formality, level of details / abstraction levels etc. Moreover, in this way, the interviewees could explicitly determine, for example, what kind of model they prefer for creating a business process model: the BPMN process diagram, ArchiMate business process viewpoint, and/or the UML activity diagram. In Table 7, for each requirements, it is indicated how many interviewees (five in total) mentioned that it is relevant for the intended ADL. Thus, [1/5] means that a certain requirement was mentioned as relevant for the ADL by only one interviewee, [2/5] by two interviewees and so on up to [5/5]. In addition, [-] means that a certain requirement is not relevant for the ADL. Below, we discuss the results per criteria.

Syntax and semantics

All five interviewees prefer the use of visualization, and thus both a [graphical] and [textual] syntax for the creation of the models. Regarding the formality, most interviewees mentioned that the intended ADL needs to be designed in such way that there is still some extent of freedom due to less constrained syntax rules. Moreover, in most cases, not a particular ADL with specific symbols is used. Instead, just basic shapes (circles, rectangles etc.) are used which are sufficient to cover the most important aspects of the architecture of a Pega application. Based on this observation, it has been indicated that the intended ADL needs to be [semi-formal]. Though, each architecture model needs to be extended by a clear description that prevents ambiguity, and fosters its understandability.

Viewpoints

For each viewpoint, suitable models of BPMN, ArchiMate and UML were discussed. Most architects [4/5] mentioned that it is relevant to elaborate the context viewpoint. Mainly, in terms of business functions. Regarding the functional viewpoint, all interviewees [5/5] agreed on the relevance of architecture models that belong to this viewpoint. Especially, regarding the interaction between a BPMS and the integrated systems, the role of each system, and the system components that are involved. It is also important to clarify what system(s) is/are the system of record. The information viewpoint has [5/5] due to the fact that all five interviewees think that business process models are relevant to be created when specifying the communication flows in a BPMS application landscape. More precisely, the BPMN process diagram is perceived to be the most suitable way for creating business process models as part of the information viewpoint. Furthermore, data models are relevant to be created in order to determine data structures which are one of the most important properties of a BPMS application. Next to these relevant viewpoints, models regarding the concurrency viewpoint were not mentioned as relevant due to the quite technical and comprehensive properties of these models. This then means that we have not included the models of UML that are part of the scope of the concurrency viewpoint, including the UML state diagram, (and partly BPMN), to the design of the intended ADL. The development viewpoint also seemed to be irrelevant because, usually, the technical infrastructure (type of databases, servers etc.) is fixed for most BPMSs.

Abstraction levels

Regarding abstraction levels, the interviewed architects create quite detailed architecture models [Level 3], because, sometimes, also sub components are specified, as well as data models similar to the level of detail of Entity Relationship Diagrams (ERDs) in order to specify detailed data objects.

Architecture styles
Most architects create models that are likely service-oriented in terms of functionalities/components, interfaces and services [5/5]. Also, a client-server style is often applied. Both the component-based style and layered style seemed to be applied less often.

Architecture purposes
Architecture creation, validation, and refinements are the most important tasks that need to be provided and fostered in certain ways by the intended ADL [5/5]. Analysis purposes are less relevant [2/5]. Currently, in most cases, most models are not created to clarify what to build and how to build it. In fact, the models are mainly used as reference work to clarify certain aspects of the architecture in the future. Due to the fact that lots of contextual changes occur during a project, it needs to be easy to refine the architecture models of the intended ADL, and the corresponding descriptions at different moments during a project. Regarding the validation of the correctness of architecture descriptions, mainly peer-reviews or other related semi-formal techniques are applied. For this, the ADL can serve as a clear and solid communication means. Next to the three aforementioned purpose, the common architecture links are also relevant in terms of the traceability between architecture models [5/5], as well as the specification of distributed systems. Furthermore, it is preferred that the ADL is supported by a tool [5/5]. Apparently, design decisions capturing is not relevant. The most likely reason for this is the fact that design decisions are usually included to the description/specification of each model. It is therefore less relevant to use, for example, explicit annotations within the models.

Components
When creating process models, the architects include both [inter-processes] and [intra-processes] to these models. Though, it depends on the project context whether both types of processes are involved or not. Regarding software architecture related components, only the [static] structure is modelled. In other words, only the design time elements are specified.

Connectors
Based on the results of the criteria above, it is indicated that information/data flows, message flows, and interfaces (APIs) are all relevant to be included to the ADL [5/5].

Configurations
Based on the desired syntax and semantics, and the relevant viewpoints, the configuration of the intended ADL needs to be [implicit]. This means that the rules and meanings of the connection between the components and connectors of the ADL are not explicitly added to the models. Instead, clear guidelines and rules need to be defined.

6.2.5 Focus groups
Next to the semi-structured interviews, we also conducted two focus groups in order to gather the opinions from multiple practitioners at once on the design of the intended ADL. One focus group was held with 12 Pega business architects, and another focus group was conducted with 10 Pega system architects. Both focus groups were not recorded. Instead, only notes of the most important answers and/or points have been taken during the discussions. These were then consulted to target the selection criteria in Table 7.

First, a general explanation of ADLs, and a specific explanation of the perceived properties and purposes of the intended ADL was given. After that, small discussions were conducted by means of example models of the candidate ADLs. The main objective was to confirm the results of the semi-structured interviews. However, during the focus groups, the selection criteria were not explicitly target. Instead, the questions below were discussed. Below, in Table 8, the main summarized answers can be found.

Table 8: Focus groups results

1) What types of architecture models are relevant within your project life cycle? 2) As a minimum, what aspects of a Pega application do you want to describe by means of an ADL?
• Overall view on the context of different disciplines and stakeholders; • Organogram, including roles; • Mission / vision model;

• Business functions. Not fully required, but good for the context scope; • High-level process description; • Integration model / interface model; • Component / service diagram; • Roadmap of relations between work packages and architecture components; • Class diagram / data objects / CRUD.
3) What are your experiences with applying ADLs? For example, can you mention any strong and/or weak points of an ADL you are familiar with? What are the benefits and drawbacks of creating and describing architecture models? etc.
• Most architects did not have much experience with applying ADLs in practice. The ones who have experience, are familiar with the candidate ADLs of this research; • Most ADLs are meant for certain types of stakeholders. The degree of seniority influences the extent to which an employee can work with models; • Sometimes, a lack of knowledge is not necessarily a need for more details, but also for more context; • How do you ensure the quality of the models, especially when the size of the projects increases?; • It is difficult to enforce the right ownership of the usage of an ADL; • Due to time pressure, it is not always possible to correctly create all relevant models.
4) What are possible ways to minimize the so-called BA-SA communication gap, if any? For example, what are the best architecture models to bridge this gap?
• Making a minimal viable product (MVP) makes the BPMS architecture tangible; • Make the business responsible for the business-IT alignment; • The gap also deals with other target audiences, next to the BAs and SAs; • Traceability of different viewpoints for different target audiences; • A good PSA is important to realize a successful project. The contents depends on the context of the project; • Validating and updating architecture models.

In Table 7, similar to the semi-structured interviews, for most requirements, the results of the focus groups are shown. Due to the fact that two focus groups were held, [1/2] means a certain requirement is relevant for the intended ADL according to only one focus group, whereas [2/2] means that both focus groups think it is a relevant requirement. Based on the results of the focus groups, we can conclude that the focus groups both confirmed most results of the semi-structured interview, and added context to the results of the semi-structured interviews.

6.2.6 Summary

The remaining paragraphs of chapter 6 will provide the remaining answers to both SRQ3 and SRQ4. In this paragraph, ***SRQ2*** has been answered completely:

- *What needs to be considered when designing a process-oriented ADL for specifying communication flows in BPMS application landscapes?*

Currently, there is no ADL particularly aimed at (software built on) a BPMS. More precisely, there is no suitable ADL that fully meets all requirements of the intended ADL. Many ADLs have been excluded from the selection. A lot of ADLs are too-domain specific and/or too technical-oriented, outdated, or not applied in practice. In addition, there are also a lot of UML-profiles, and duplicates of the three candidate ADLs. We have analyzed and compared three candidate ADLs through a literature review, semi-structured interviews, and focus groups: BPMN, ArchiMate, and UML. Only UML nearly meets all defined requirements. For most criteria and corresponding requirements of the ADL, the results of the semi-structured interviews, and focus groups are aligned with each other. Regardless of minor differences, the individual results between the business architects and system architects were quite similar regarding both the interviews and focus groups. Based on all results, the following models of the candidate ADLs have been selected to be included to the specification of the intended ADL:

BPMN:	**ArchiMate:**	**UML:**
• process diagram; • choreography diagram.	• organization structure viewpoint; • business function viewpoint; • business process viewpoint; • application usage viewpoint; • application cooperation viewpoint.	• class diagram; • component diagram.

6.3 High-level architecture decomposition model

Based on the comparison analysis of the candidate ADLs in the previous paragraphs, we have designed the intended ADL utilizing the Method Association Approach. In this paragraph, we describe the high-level architecture decomposition model that visualizes the scope of the intended ADL.

In Figure 30, the high-level architecture decomposition model of the intended ADL is shown.

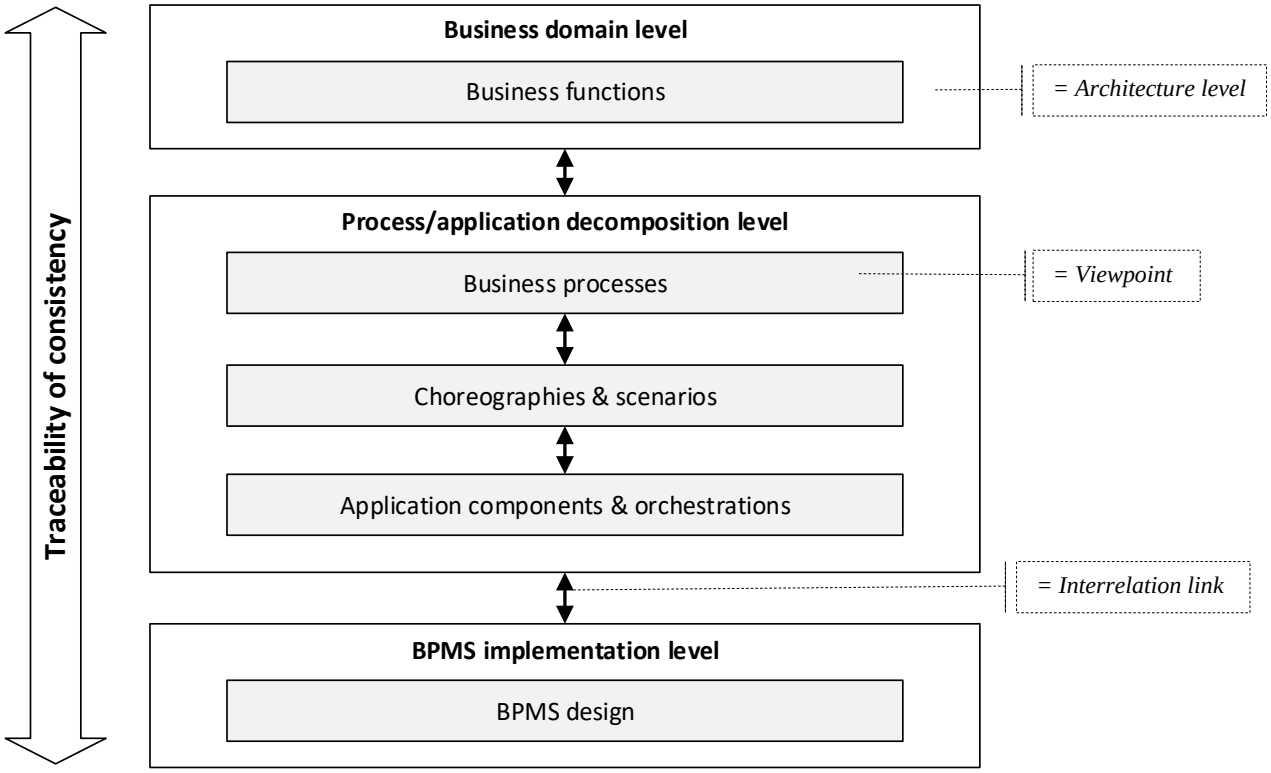


Figure 30: High-level architecture decomposition model of the intended ADL

As visualized, the intended ADL focuses on three architecture levels regarding the development of an application that runs on a BPM platform / BPMS: *business domain level*, *process/application decomposition level*, and *BPMS implementation level*. These levels aim at both business and IT related viewpoints, and the interrelations (translation) between them. Each level contains one or more viewpoints. For example, the *Business process* on the *Process/application decomposition level*.

The names of the architecture levels have been determined based on the model-driven architecture (MDA) that was described in paragraph 4.3.3. For each viewpoint, one or more suitable models of ArchiMate, BPMN, and UML are applied. Hence, basically, the ADL is in fact a coherent set of existing types of diagrams that are interrelated and complementary to each other in certain ways.

The main property of the ADL is the traceability of the consistency of the specification of the communication flows across the different architecture levels and viewpoints. In other words, in terms of the communication flows within the BPMS application landscape, it can be specified how the business functions can be translated to the business process, applications, and, eventually, the actual deployment of the BPMS at the lowest architecture level of the ADL. This is also possible in the opposite way, and between single viewpoints. Thus, it does not entail solely a top-down or bottom-up approach. Eventually, a coherent translation of the business related specification till the implementation related specification (and vice versa) regarding the communication flows within the BPMS application landscape can be created.

6.4 High-level ADL model structure

In this paragraph, we specify the high-level ADL model structure that clarifies the overall syntax of the ADL. In Figure 31, the high-level ADL model structure is shown. This model is a high-level visualization of the syntax of the ADL, and shows the interrelations between the selected architecture models of the candidate ADLs. Therefore, it can be considered as the meta-model of the intended ADL.

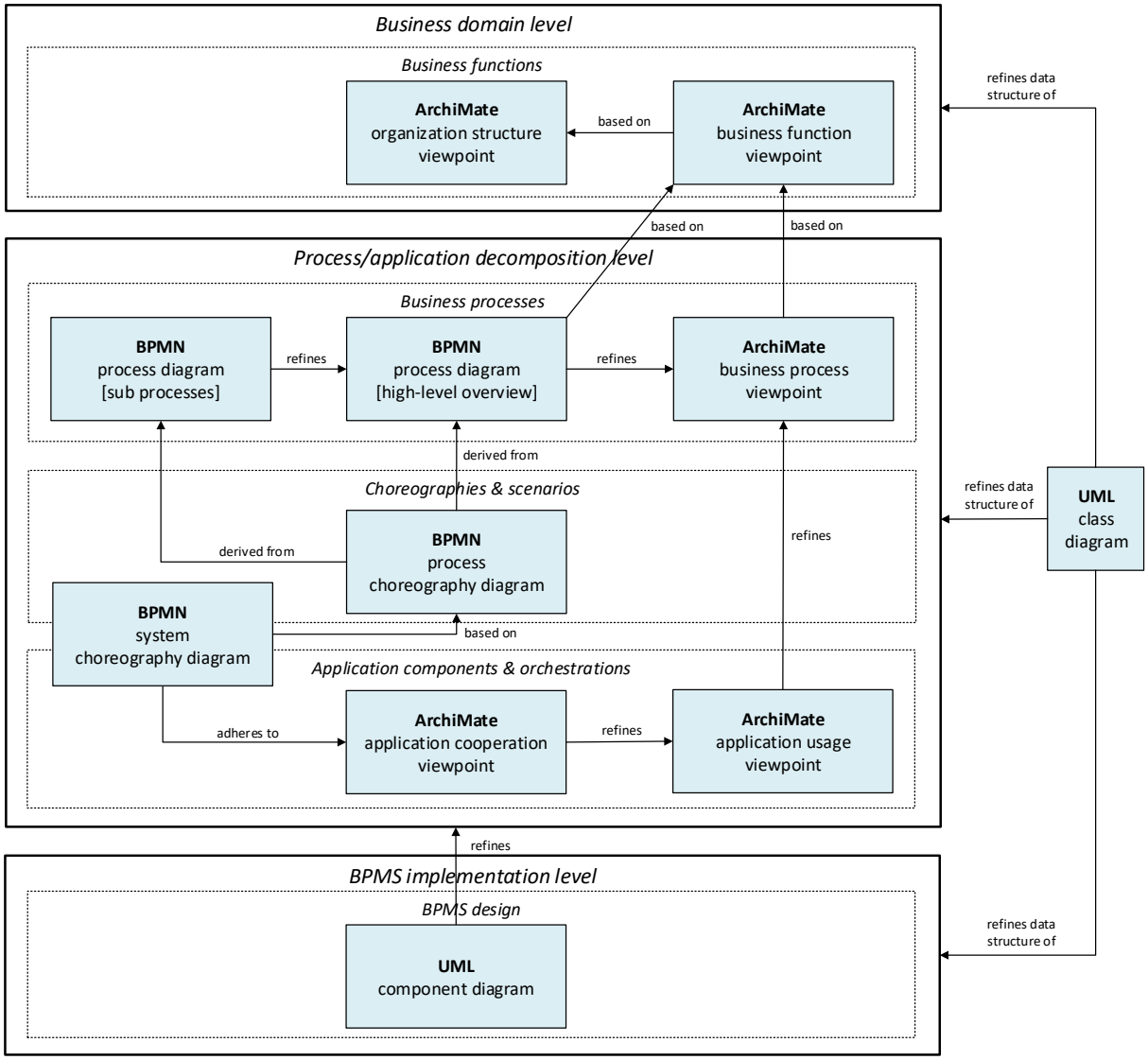


Figure 31: High-level ADL model structure

As shown in Figure 31, each selected model of the candidate ADLs have been assigned to a certain architecture level and the corresponding viewpoint. The arrows between each model indicate the interrelations / mappings between them. For example, the relation "*refines*" from the BPMN business process diagram [high-level overview] to the ArchiMate business process viewpoint means that the BPMN business process diagram [high-level overview] is a detailed/extended version of the ArchiMate business process viewpoint. The BPMN business process diagram [high-level overview] itself is then used to derive the contents of the BPMN process choreography diagram. The same goes for the other relations that are visualized in Figure 31. Due to the fact that the BPMN system choreography diagram is applicable to both the *choreographies & scenarios*, and *application components & orchestrations*, we have partly placed it on both the aforementioned viewpoints.

In the next paragraphs, we further describe the high-level ADL model structure in more details. In this way, for each architecture model, the syntax and semantics are specified, as well as the guidelines for practitioners.

6.5 General specifications & guidelines

Practitioners should apply the ADL in a certain way. Basically, the syntax of the ADL already specifies what is allowed and what is not allowed to be done in what way. Though, in general, there is a certain approach for applying the ADL in practice. This is described below.

6.5.1 Twin Peaks model

By means of the Twin Peak model (Cleland-Huang, Hanmer, Supakkul, & Mirakhorli, 2013), we visualize the essence of applying the ADL in practice. This is shown in Figure 32.

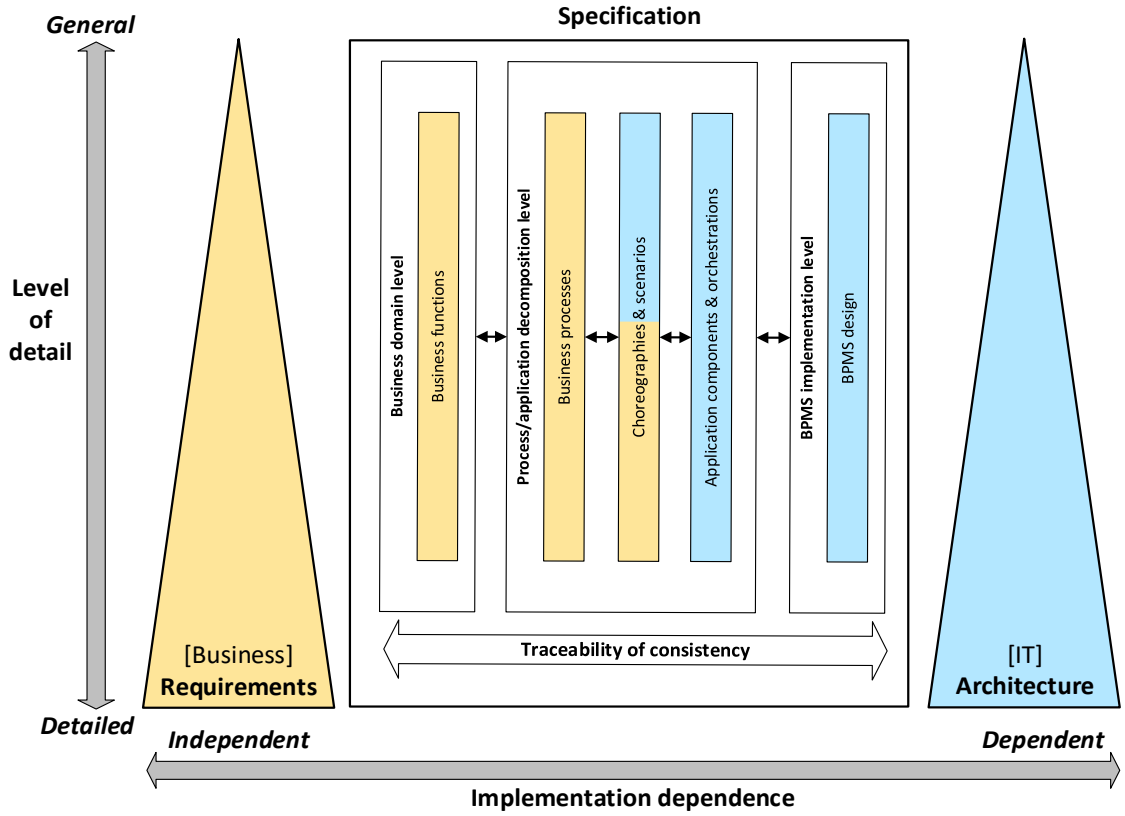


Figure 32: The ADL within the Twin Peaks model

The Twin Peaks model provides an iterative way for refining the architecture of a software system based on the requirements / user stories. Therefore, it contains two large triangles (= peaks) that represent the requirements (left peak), and the architecture (right peak). The x-axis indicates the implementation dependence, whereas the y-axis indicated the level of detail regarding the specification. The essence of the Twin Peaks model is the fact that both the requirements and the architecture design need to be aligned/consistent with each other. This means that they are both specified in correspondence with each other, so, not separately. Therefore, there are continuously refined in parallel through multiple iterations. This is aligned with the fact that, nowadays, software development is usually done in an Agile way, such as Scrum.

As shown in Figure 32, we have extended the original Twin Peaks model (Cleland-Huang, Hanmer, Supakkul, & Mirakhorli, 2013) with the *high-level architecture decomposition model* of the intended ADL that we have placed in the middle. We have turned this high-level model to the side in order to position it in the desired way between the two peaks. Namely, we have colored the requirements peak yellow, which represents the business. On the right side, we have colored the architecture peak, which represents the IT. Thus, initially, the requirements are defined by the business, and are then iteratively specified in collaboration with the IT. As shown in Figure 32, it is indicated what viewpoints are assigned to the business stakeholders and what viewpoints are meant for the IT stakeholders. We have done this by assigning the corresponding color of the peak to the applicable viewpoint of the ADL. In this way, a high-level division of tasks has been created. However, in practice, it will be the case that all

viewpoints require some input from both the business and IT during the iterations. Though, the owner of each viewpoint is either the business or the IT, which entails at least the following persons / roles:

- *Business*: including enterprise architects, business architects, subject matter experts;
- *IT*: including solution architects, system architects / developers.

6.5.2 General guidelines

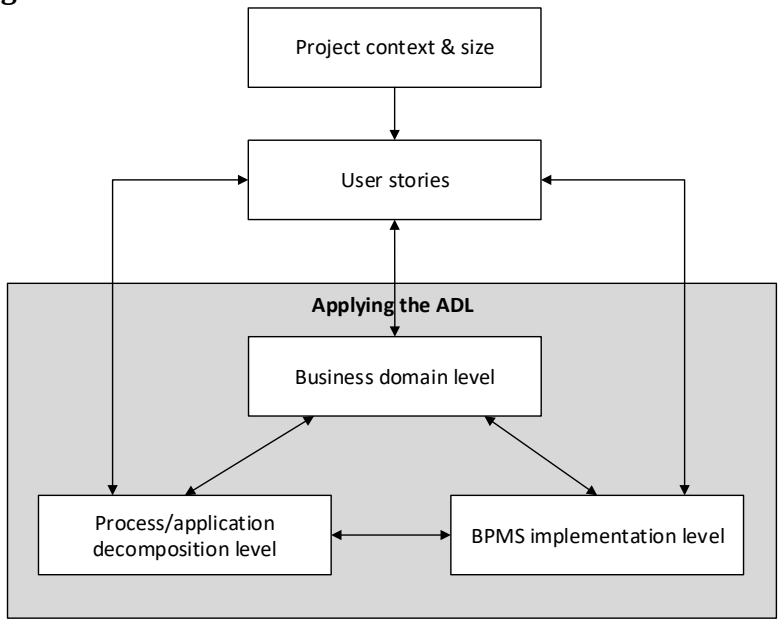


Figure 33: General process of applying the ADL

Figure 33 visualizes the general process of applying the ADL. First, the project context needs to be examined in order to determine properties such as the organization structure, the complexity of the application landscape, and the number of actors / process participants that are involved in the corresponding business processes. Next to the project context, the most important properties of projects is the project size / duration. The project size / duration might influence the extent to which it is necessary and/or possible to correctly create and maintain create architecture models.

To put the Twin Peaks model in context, based on the project context & size, and the user stories, the first iteration of creating the models figuratively occurs at the top of both peaks at a high abstraction level. Here, general descriptions of the requirements in terms of business functions, business processes, and partly the choreographies and scenarios for designing the architecture are specified. More precisely, after high-level requirements / user stories have been formulated, in general, based on the interrelation links, the architecture models of the ADL are created in the following order:

Business domain level

1. *ArchiMate organization structure viewpoint;*
2. *ArchiMate business function viewpoint;*

Process/application decomposition level

3. *ArchiMate business process viewpoint;*
4. *BPMN business process diagram [high-level overview];*
5. *BPMN business process diagram [sub processes];*
6. *BPMN process choreography diagram;*
7. *ArchiMate application usage viewpoint;*
8. *ArchiMate application cooperation viewpoint;*
9. *BPMN system choreography diagram;*
10. *UML class diagram;*

BPMS implementation level

11. *UML component diagram;*

After the first iteration, it depends on certain situational factors what the next steps will be. Usually, this is caused by new and/or changing user stories (requirements). Thus, after several iterations have been done, a different order of creating/maintaining the models might be more applicable, and/or a certain model might not be refined anymore. For example, if it is not relevant anymore to further refine the requirements in terms of business functions at the business domain level due to the fact that the specifications become more concrete. Moreover, eventually, it may also be relevant to start a new iteration at the BPMS design level. For example, to determine the specification of the other viewpoints based on the available (reusable) components within the BPMS design.

Hence, during the next iterations, in general, the architecture specification becomes more specific when reaching the bottom of the peaks. Moreover, during each iteration, the requirements also become closer to / more dependent on the actual implementation of the architecture design of the application (Cleland-Huang, Hanmer, Supakkul, & Mirakhorli, 2013). Therefore, we prefer to use the term *Project Architecture (PA)* instead of *Project Start Architecture (PSA)* to refer to the created architecture models, because they are refined/updated multiple times during the entire project.

6.5.3 Next paragraphs: detailed specifications & guidelines

In the next remaining paragraphs of this chapter, the viewpoint(s) of each architecture level of the ADL is/are described in terms of the syntax and corresponding semantics. As mentioned before, for each viewpoint, we make us of one or more models from the candidate ADLs (BPMN, ArchiMate, and UML). In several situations, we have partly adjusted the syntax of a candidate ADL in order to meet the requirements of the ADL. These so-called violations are mentioned explicitly.

For each model, we provide a detailed guideline that can be followed to create the model. Basically, these guidelines are step-by-step walkthroughs that explain how the model is created, and whether the business stakeholders or IT stakeholders are responsible for the creation and maintenance of the corresponding architecture model. The traceability of the interrelations and consistency between the models can be mapped, and are included to the corresponding guidelines. For this, as an example, we show a certain part of each model in order to highlight the essence of the mappings between them. These mappings are visualized by means of red dotted lines between two similar / complementary shapes. Eventually, it could be specified how each communication flow is decomposed across the different architecture levels.

As a practical example of the models that can be created by means of the ADL, we use the fictional case on the car insurance company again as a running example. At this point, we have extended the running example in several ways in order to demonstrate, specify and validate all properties (model elements) of the ADL.

The description of the extended case can be found in Appendix C. In Appendix D, an architecture document template can be found which can be used when the ADL is applied in practice. The fully elaborated ADL models and the corresponding descriptions of the running example are presented by means of this architecture document template in Appendix D.

6.6 Business domain level – specifications & guidelines

At this architecture level, the context of a BPMS application is specified. This is done in terms of the business functions and the corresponding business roles, both internal and external, that are involved. In addition, to put the business functions in context, the organization structure is also elaborated.

6.6.1 Business functions

A business function is a certain organizational part (department), for example Sales and Production, which can be considered as an umbrella term for a collection of multiple business processes. It is possible that a single business process is part of / makes use of multiple business functions.

ArchiMate organization structure viewpoint

The ArchiMate organization structure viewpoint is used to create a general organizational view that helps to visualize the context / business domain of a BPMS application. For this, only one particular ArchiMate shape is used, which is shown in Figure 34.

Figure 34: ArchiMate organization structure viewpoint shape

The *business actor* shape can be used to model departments / business units of an organization. Decompositions can be modelled by placing a business actor shape within another business actor shape.

GUIDELINES [main stakeholder: Business]

1. If available, consult the organogram / organization structure model of the organization.
2. Create a large *business actor* shape that represents the organization.
3. Model each department / business unit as a smaller *business actor* shape within the large *business actor* shape. If applicable, it can be modelled what departments / business units are part of the front office, back office etc. by means of placing the corresponding departments / business units within a *business actor* shape called front office.
4. To visualize a certain organizational hierarchy, the shapes can be modelled in a top-down way. For example, by placing the *business actor* shape called *Managing Board* at the top (instead of at the bottom) within the large business actor shape that represents the organization.
5. Briefly describe the meaning of each *business actor* (department), including the relations with other *business actors*.
6. Eventually, by means of the ArchiMate organization structure viewpoint, the business actors that are involved in the context of the project can be indicated.

Running example [Appendix D – Figure 1]
The car insurance company has been modelled as a large *business actor*. It is divided into the front office, back office, the managing board, and operations. Both the front office and back office consist of multiple departments / business units, including Finance, and HR. The ArchiMate organization structure viewpoint does not show a certain hierarchy regarding the organization structure. Though, a certain organization hierarchy has been considered while creating the model. Therefore, the managing board has been positioned at the top. Then, below of the managing board, the remaining departments are shown.

ArchiMate business function viewpoint

To model the business functions, the ArchiMate business function viewpoint is used. The corresponding shapes are shown in Figure 35.

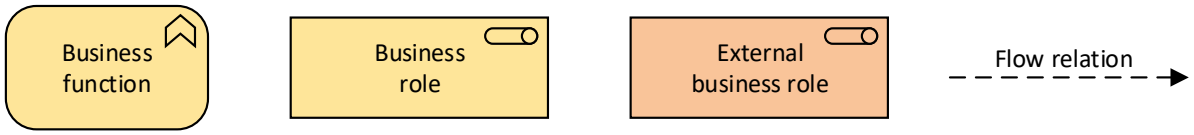


Figure 35: ArchiMate business function viewpoint shapes

Business functions are visualized by means of yellow rounded rectangles with a triangle/arrow symbol at the right top of the shape. *Business roles* are visualized as a rectangle with a cylinder symbol at the right top of the shape. A business role represents the organization which is addresses by the business function viewpoint. An organization can be divided into multiple business functions. Therefore, the business functions are modelled within the shape of a business role. In addition, also an *external business role* can be modelled that communicates with the organization. The external business roles are visualized as an orange business role, and can be, for example, a customer or a supplier. *Business functions* are connected to each other by means of black dotted arrows that represent a *flow relation*. This can represent, for example, information flows. These flows also include flows of physical goods that are exchanged. *External business roles* can only be connected to a *business function*.

Violations:
- ArchiMate does not apply a different color to explicitly visualize an external business role as an additional shape. We have added an external business role shape to emphasize a clear distinction between internal communication flows and external communication flows in terms of the business functions.

GUIDELINES [main stakeholder: Business]

1) By means of the ArchiMate organization structure viewpoint, identify the departments / business units that can be considered as business functions. For example, usually, the Finance department is also seen as a business function. To derive the remaining business functions, consult the contextual information that was collected beforehand and/or the business processes that is optimized by means of the BPMS. Eventually, for this first step, it is sufficient to identify only the business functions that are involved within the scope of the project. So, if, for example, Finance is not involved, this does not need to be included to the viewpoint.
2) Create a large *business role* shape that represents the organization. If applicable, the customer/client, suppliers and other external entities are modelled by means of the *external business role* shape.
3) For each business function, create a business function shape and place them within the large business role shape that represents the organization.
4) Determine what information (messages) are exchanges between the business functions.
5) Use the *flow relation* to connect two business functions that involve a certain information exchange. In addition, use the *flow relation* to connect the business functions that exchange information with an external business role.
6) Briefly describe the meaning of each *business function*, including the relations with other *business functions*, and *external business roles*.

Running example [Appendix D – Figure 2]
The car insurance company has been modelled as a *business role*. Several *business functions* are shown, including *Payment claim handling*, and *Finance*, that are involved in the process of handling a payment claim. Namely, the *Finance* business function is involved due to the fact that it provides the required financial details for the payment purposes. Both the insurant and the bank have been modelled as an *external business role*. There are several *information flows* from an external business role to the company. For example, an insurance payment claim that is exchanged from an Insurant to the Payment Claim Handling business function of the company.

Traceability: consistency with ArchiMate organization structure viewpoint

The ArchiMate business function viewpoint can partly be created ***based on*** the ArchiMate organization structure viewpoint. For example, in Figure 36, the business actor (= department) called *Finance* can be assigned to a business function with the same name.

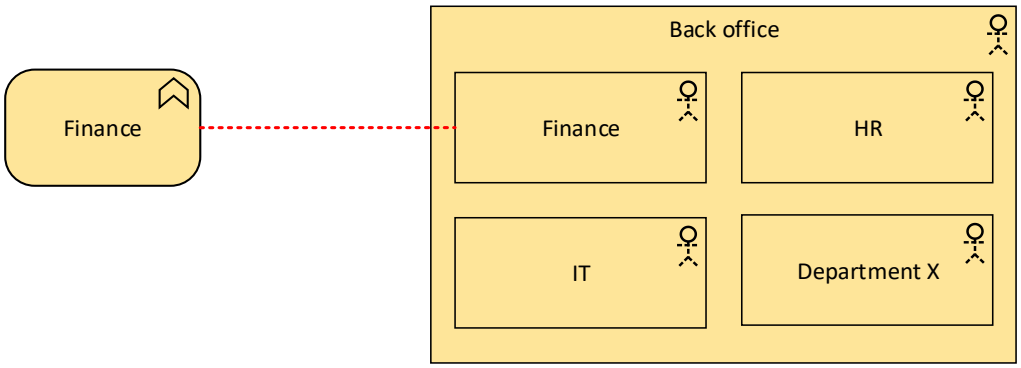


Figure 36: [Mapping] ArchiMate business function viewpoint <=> ArchiMate organization structure viewpoint

6.7 Process/application decomposition level – specifications & guidelines

This architecture level aims at the decomposition of the business processes that are (partly) automated by means of the BPMS and the other integrated systems. The functionalities of all systems are orchestrated in certain ways for the execution of the business processes that are configured in the BPMS.

6.7.1 Business processes

A business process is a coherent set of interrelated process activities that are carried out in a certain order to reach a certain goal, e.g., processing an insurance payment claim. A business process model specifies the orderliness of the activities. In other words, the order in which the activities are carried out, including possible decision points. For this, both an ArchiMate viewpoint and a BPMN diagram are used.

ArchiMate business process viewpoint

First, the ArchiMate business process viewpoint is described, which can be used to create a high-level business process model. The corresponding shapes are shown below in Figure 37.

Figure 37: ArchiMate business process viewpoint shapes

A *business process (or process activity)* is represented by a yellow rounded rectangle that contains an arrow symbol at the right top. A *business event,* which is represented by a yellow rounded rectangle with a rounded arrow symbol, triggers the execution of a business process. The business processes are carried out in a certain order. This is visualized by means of using the *triggering relation* to connect the business process based on the order of execution. Thus, the directions of the triggering relations show the corresponding order. The *assignment relation* is used to model what *business roles* are assigned to the execution of a business processes. A decision point is modelled by means of a *junction,* which is visualized as a black dot. Within a business process, multiple sub business processes could be modelled. This can be done by placing multiple business process shapes within one large business process shape.

GUIDELINES [main stakeholder: Business]

1) Based on the business functions and the flows between them within the ArchiMate business function viewpoint, indicate the main *business function* that is applicable to the business process that is optimized by means of the BPMS.
2) Identify the *business roles* that are involved in the selected business functions. For example, when the business function called Finance is involved, this implicates that a financial controller would be an applicable business role, next to a claim handler.
3) Identify the *business event* that serves as the trigger for starting the business process.
4) Identify the main tasks (= stages) within the business process.
5) Create a large *business process* shape, and then, add smaller business process shapes of the main tasks in the order of execution. Use the *triggering relation* to connect the main tasks business process shapes. In applicable, add a *junction* to visualize a decision point.
6) For each process participants, create a *business role* shape, and connect them to the main tasks they are involved in by means of the *assignment relation*.

Running example [Appendix D – Figure 3]
The business process can be called *handle payment claim*. The *business event* is a new payment claim that needs to be handled. The business process is divided into multiple smaller sub business processes that are carried out in a certain order. These processes are in fact different stages / phases. There are two decision points that have been modelled by means of a *junction*. By means of the assignment relation, each business role is assigned to one or more business processes.

Traceability: consistency with ArchiMate business function viewpoint

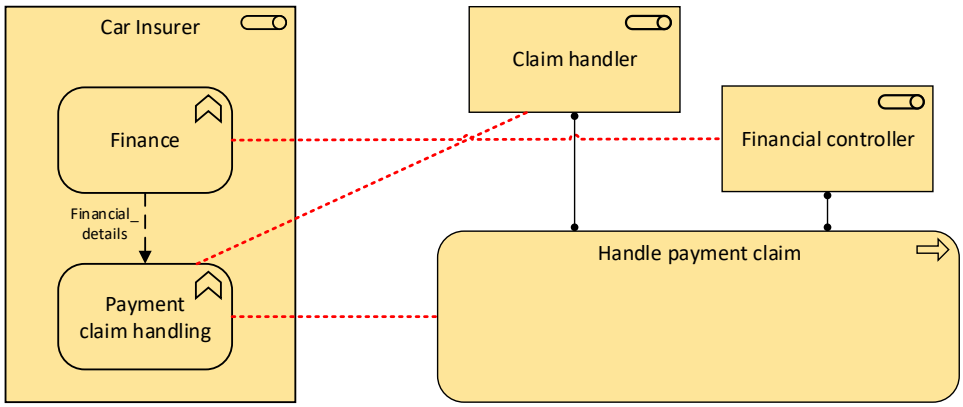


Figure 38: [Mapping] ArchiMate business function viewpoint <=> ArchiMate business process viewpoint

The elements/shapes within the ArchiMate business process viewpoint can be determined ***based on*** the ArchiMate business function viewpoint. For example, when the business process entails a payment claim that is handled (Handle payment claim), the corresponding business function might be called Payment claim handling. This entails that an applicable business role would be, for example, a claim handler. Within the ArchiMate business process viewpoint, the flow relations visualize what other business functions exchange information with the main business function. Another business function that would be involved is, for example, Finance that is responsible for the actual payment of a certain payment claim. This implies another business role that can be involved in the business process, for example, a financial controller.

BPMN process diagram [high-level overview] + [sub processes]

To create a more detailed process model, the BPMN process diagram is used. A limited set of the corresponding shapes are shown below in Figure 39.

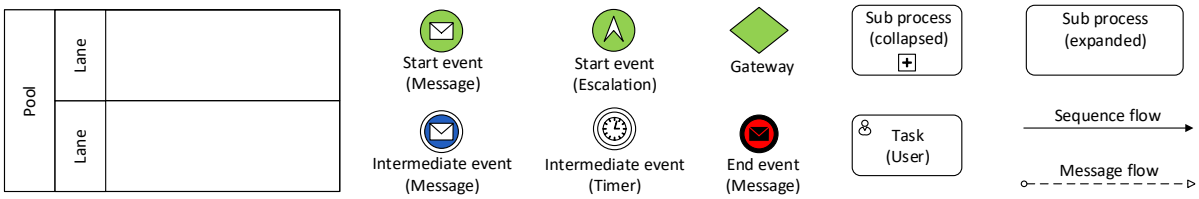


Figure 39: BPMN process diagram shapes (limited set)

There are many types of shapes and corresponding variants for creating process diagrams that are provided by BPMN. The set of shapes above is solely a small selection of possible shapes that can be used to create a detailed business process model. Though, the shapes in Figure 39 can be considered as the basic shapes that are presented in most BPMN process diagrams.

A single organization is modelled as a large horizontal rectangle: a *pool*. This is divided into multiple *lanes*, each representing a certain actor (process participant) within the corresponding organization. The other shapes are then modelled inside the lanes. Each business process starts based on a certain *start event* which is basically a trigger. Each type of start event has a specific symbol. For example, a *message start event*, or an *escalation start event* as shown above. The same goes for the different types of *end events*. *Process activities / tasks* are visualized by means of a rounded rectangle that contains a certain symbol at the top left. This symbol indicates how the tasks is carried out, for example, by a user or fully automatically by an application component. The tasks are linked by means of *sequence flows*. This then shows the order in which the tasks are carried out. Within a single process diagrams, *sub processes* can be modelled as well. These processes are then expanded / further elaborated in separated diagrams. For the intended ADL, *message flows* are most relevant. This type of flow is used to model the message exchanges between process participants from different pools. Next to the start events and end events, also different types of *intermediate events* can be added. This can be, for example, a message that needs to be received or a certain amount time of waiting before the next task is carried out. Furthermore, different types of *gateways* can be used to model decision points, based on a certain (Boolean) value / business rule.

Violations:
- BPMN does not allow event types and message flows to be used within a sub process diagram. We have done this to create clear distinctions between high-level views, and more detailed views on each sub process of the high-level views.

GUIDELINES [main stakeholder: Business]

1) Create a *pool* that represents the organization, and, for each process participant (= business role within the ArchiMate business process viewpoint), create a *lane* within the pool. In addition, create a *pool* for each external business role.
2) ***In case there is no direct communication between the process participants via the BPMS, and/or if many tasks are carried outside the BPMS, create a pool for the BPMS which will then contain the (automated) tasks that are carried out by the BPMS.***
3) For each *pool*, determine the type of *start event*, and place the corresponding shape within the corresponding lane.
4) For each main task from the ArchiMate business process viewpoint, add a *sub process* shape with the same name within the lane of the process participants that is assigned to the task. Connect the sub processes to each other by means of the *sequence flow*, according to the order within the ArchiMate business process viewpoint. In addition, identify the decision points, and add the corresponding type of *gateway*.
5) Identify the sub processes that are involved in a message exchange. Connect *message flows* between these sub processes and the *intermediate message events*.

6) Add the right type of *end events* to each pool.
7) For each sub process from the BPMN process diagram [high-level overview], determine the *tasks*, *decision points* and other details.
8) For each sub process, create a *separated process diagram*.

Running example [Appendix D – Figure 4 + 5]
Three separated *pools* have been created: the car insurance company, insurant, and bank. The pool of the bank is considered as a black box. The car insurance company has two *lanes* that represent the process participants/actors within the company: claim handler, and financial controller. The lanes visualize the executor of each *task*. Two abstraction levels has been created. One large diagram visualizes multiple *sub processes* that are carried out in a certain order. Each sub process is further elaborated within a separated sub process diagram. This then contains the actual task are executed. Most tasks are carried out by means of a user action within the application. Several tasks are carried out automatically. Within the process, there are four exclusive *gateways* / decision points regarding the approval or rejection of a new payment claim. Furthermore, there are several *intermediate message events* that are triggered by a *message flow*. For example, the reparation invoice that in sent by the insurant to the claim handler.

Traceability: consistency with ArchiMate business function viewpoint

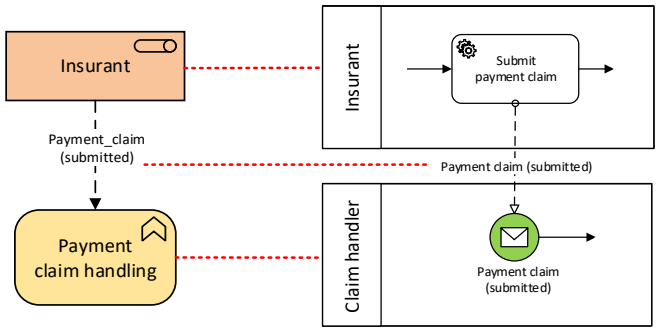


Figure 40: [Mapping] ArchiMate business function viewpoint <=> BPMN process diagram

The ArchiMate business function viewpoint is a high-level view of both the sequence flows and message flows within the corresponding BPMN process diagram. In other words, it is partly created ***based on*** the ArchiMate business function viewpoint. For example, as shown in Figure 40, in both models, there is a flow called *Payment_claim (submitted)*. In the ArchiMate business function viewpoint, the insurant is modelled as a business role, whereas in the BPMN process diagram it is visualized as a pool. The claim handler is modelled as a lane within the pool of the car insurance company. Due to the fact that the claim handler is involved in the business function called *Payment claim handling*, within the ArchiMate business function viewpoint, *Payment_claim (submitted)* goes to the aforementioned business function. Within the BPMN process diagram, *Payment_claim (submitted)* is the trigger for starting the first task that is executed by the claim handler.

Traceability: consistency with ArchiMate business process viewpoint

Both ArchiMate business process viewpoint and BPMN process diagram [high-level overview] can be used to create a business process model. The most important difference is the fact that the BPMN process diagram ***refines*** the ArchiMate business process viewpoint, by providing more shapes for creating more detailed process models. However, all business processes within the ArchiMate business process viewpoint need to be modelled in the same order within the BPMN process diagram. In this example, it is shown that a decision point / gateway has been modelled after *Check* within both models. In Figure 41, a fragment of both models shows the tasks regarding the check and approval of a payment claim. The sequence flows are identical, as well as the decision point. In contrast to the junction within the ArchiMate business process viewpoint, the BPMN process diagram shows the type of decision point. In this case, it is an exclusive gateway. In addition, the output sequence flows are indicated as *Approved* and *Rejected* within the BPMN process diagram. Though, within the BPMN process diagram, the tasks have been modelled as sub processes. These diagrams of these sub processes could be elaborated in

more detail in a separated diagram. Furthermore, both models show the process participants / roles that carry out the modelled tasks in a different way. This is also aligned with each other.

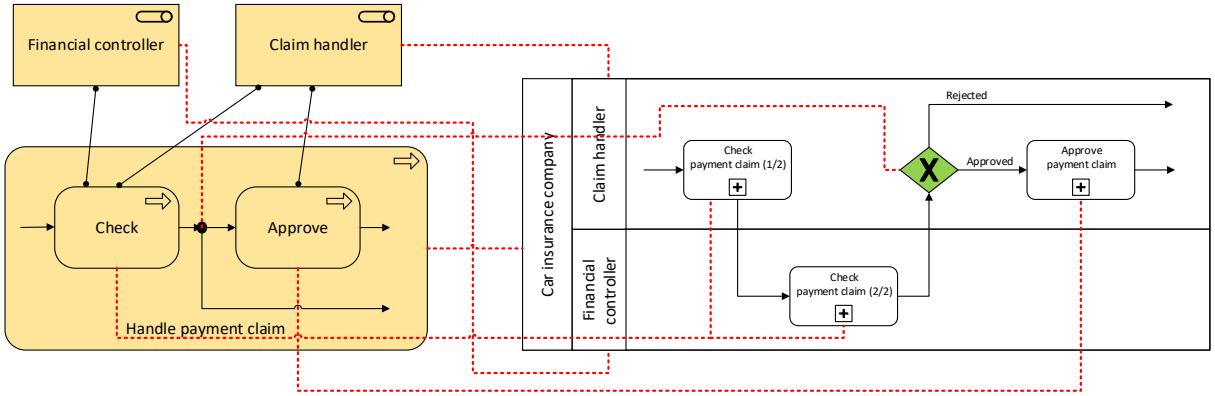


Figure 41: [Mapping] ArchiMate business process viewpoint <=> BPMN process diagram [high-level overview]

Traceability: consistency with BPMN process diagram [sub processes]

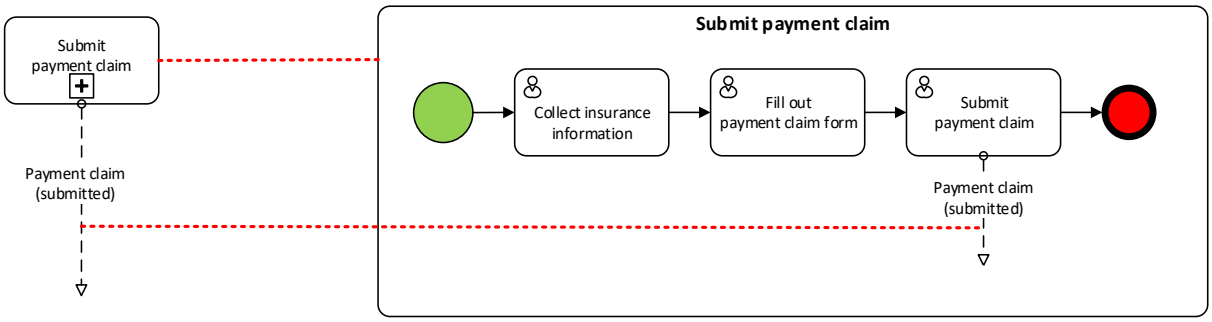


Figure 42: BPMN process diagram [high-level overview] <=> BPM process diagram [sub processes]

The high-level overview BPMN process diagram contains sub processes. Each sub process is ***refined*** in more details in a separated diagram. In Figure 42, this is done for *Submit payment claim*.

6.7.2 Choreographies & scenarios

By means of a business process model, choreographies and scenarios can be specified. Choreographies are the interactions / message exchange between the process participants from two different organizations.

BPMN process choreography diagram

For modelling choreographies and scenarios regarding the interaction between process participants, the BPMN choreography diagram is used. The corresponding shapes are shown below in Figure 43.

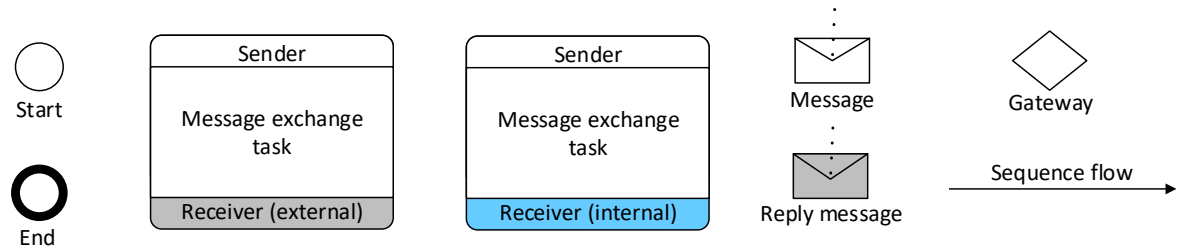


Figure 43: BPMN process choreography diagram shapes

The BPMN process choreography diagram is a specialization of the BPMN process diagram. It solely focuses on the interaction between process participants. Therefore, it specifies the orderliness of the message exchanges. In other words, it shows in what order messages are exchanged between process participants during the business process. The shapes are almost similar to the BPMN process diagram shapes. Both the *start* state and *end* state are modelled as a circle. The *message exchange tasks* are modelled as rounded rectangles, and are solely the tasks that involve any message exchange. Both the sender and receiver of a message exchange task are a person / role, and are modelled within the shape of a tasks. The sender and the initial message are colored white, whereas the receiver, and, if applicable, the reply message, are colored grey. When both the sender and receiver are part of the same organization (pool), the receiver is colored blue. This is called an internal message exchange. The message shapes are connected to the tasks by means of black dotted lines. A *reply message* only occurs in synchronous communication. Usually, the communications are asynchronous. Furthermore, similar the BPMN process diagram, the tasks are connected by means of sequence flows, and it is possible to use different types of *gateways* to visualize decision points.

An important property of a choreography is the realizability, which deals with the possibility of developing system that conforms to the choreography. For this, it is important that no deadlocks occurs within the choreography. In case of synchronous communication, it is important that the initiator/sender of a certain message exchange needs be involved in the previous message exchange (if it is not the first task that entails a message exchange) in order to let it be realizable. Namely, it needs to be clear to the sender if the previous message exchange has been executed. If this is not ensured, the choreography is not realizable.

Based on both the BPMN process diagram, and the BPMN choreography diagram, the scenarios can be derived. Basically, these scenarios are all possible paths through the different stages within a process. These paths are determined by means of decision points / gateways. This path could be visualized explicitly by means of overlays.

Violations:

- To distinguish this process-oriented BPMN choreography diagram from the system-oriented BPMN choreography diagram that is described in the next sub paragraph, we have named it BPMN process choreography diagram.
- BPMS does not consider two tasks within the same pool that are connected by means of the sequence flow are not seen as an internal message exchange. Moreover, BPMN does not apply the term internal message exchange. We do this to create a complete view on all communication flows.
- BPMN does apply the term “message exchange task”. We have added this term to explicitly distinguish them from “normal” BPMN process tasks.

GUIDELINES [main stakeholder: Business]

1) Identify the *tasks* that entail a message exchange. These are the tasks that have a *message flow* connected to them.
2) If applicable, identify the tasks that entail an *internal message exchange*. These are the tasks that are connected to a tasks within another lane from the same pool by means of a sequence flow instead of a message flow.
3) Model the *message exchange tasks*, according to the order they occur within the BPMN process diagram. In case a certain message exchange task entails synchronous communication, also add a *reply message* shape.
4) If applicable, also add *gateways* to visualize decision points. Based on these decision points, indicate all possible *scenarios*.

Running example [Appendix D – Figure 6]
During the entire process of handling the payment claim, 11 messages are exchanged. Most of them occur in an asynchronous manner. Most messages are exchanges between the claim handler or financial controller, and the insurant. Some messages are sent between the claim handler and financial controller. Thus, these are internal message exchanges. There is also one message that is sent to the bank, which also sends a reply message back to the claim handler about the processed payment. In total, there are three possible scenarios: 1) the payment claim is completely approved, 2) the payment claim is rejected after the first check, and 3) the payment claim is rejected after the calculation of the actual financial compensation.

Traceability: consistency with BPMN process diagram

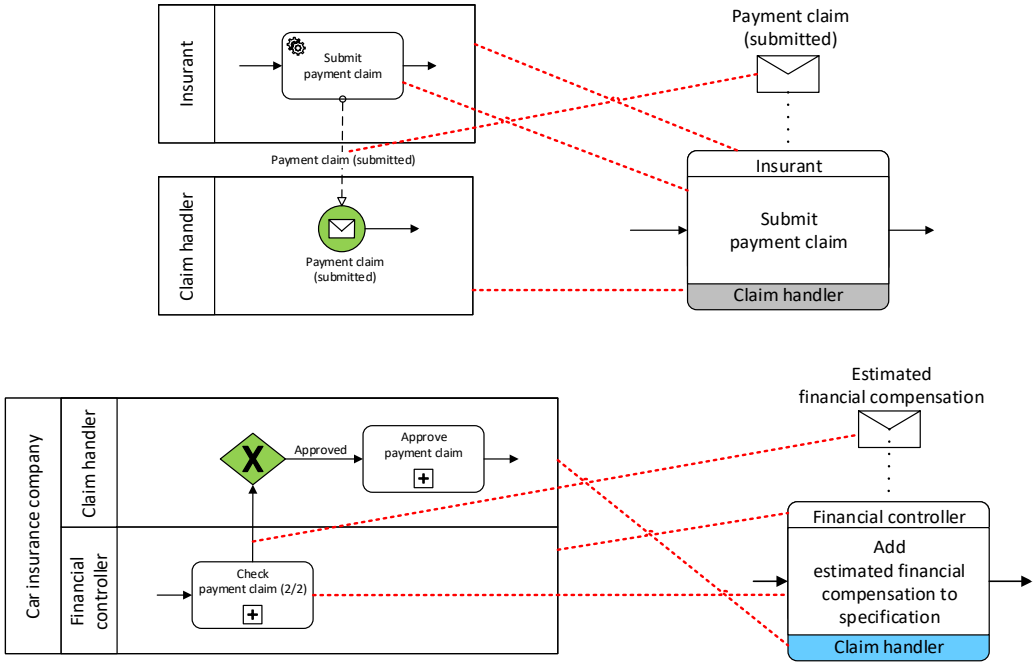


Figure 44: [Mapping] BPMN process diagram <=> BPMN process choreography diagram

The *BPMN process choreography diagram* is a specialization that is ***derived from*** of the *BPMN process diagram*. The choreography model focuses on the message exchange between both different pools and within a single pool between two process participants (a swim lane). In a BPMN process diagram, the former is modelled by means of message flows, while the latter is modelled as a control flow. Though, for the intended ADL, the latter is indeed seen as a so-called internal message exchange within a single pool/organization. In this example, the communication between the claim handler and the financial controller. The task called *Add estimated financial compensation to specification* is a task of the sub process *Check payment claim (2/2)* that results in the estimated financial compensation that is communicated to the claim handler.

6.7.3 Application components & orchestrations

Usually, a BPMS is integrated with other system in order to, for example, request and use customer data from a CRM system that is needed to execute the business process. The application components & orchestrations are defined as the role, behavior and structure of all systems that are used/invoked through interaction services and interfaces (API/web services) in the order of executing the business processes.

ArchiMate application usage viewpoint

To model at what point a certain system is used during the business process, the ArchiMate application usage viewpoint is used. The shapes for creating this viewpoint are shown below in Figure 45.

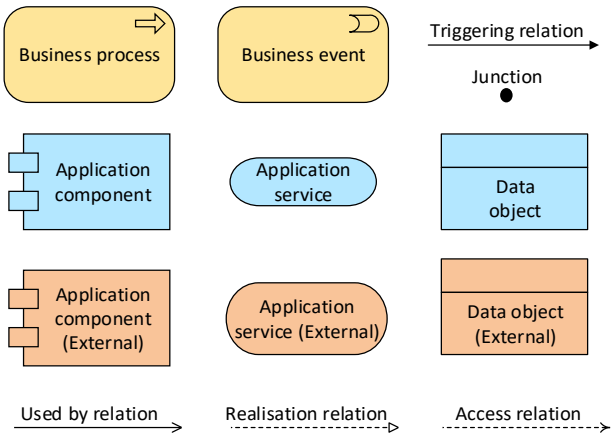


Figure 45: ArchiMate application usage viewpoint shapes

The ArchiMate application usage viewpoint is partly similar to the ArchiMate business process viewpoint. The difference is the fact that the ArchiMate application usage viewpoint visualized what systems are used for the execution of the business processes. Therefore, the *used by relation* is applied to connect an *(external) application component*, via an *(external) application service* that is realized by the application component, to a *business process* Such an application service is in fact a certain part of functionality that is realized by the corresponding application component. For this, the *realisation relation* is used. In some cases, an application service can also represent a web service that is required for the execution of the business processes. Furthermore, the application components could have access to a *data object*. This is visualized by means of the *access relation*. An external application component, application service, and data object is colored orange.

Violations:
- We apply a different color to explicitly model the aforementioned external elements.
- Data objects are not part of this viewpoint. Though, we have added data objects to this viewpoint in order to avoid an extra architecture model which would be quite similar.

GUIDELINES [main stakeholder: IT]

1) For each application component that is integrated with the BPMS, including the BPMS itself, create an *application component* shape.
2) Determine the application services that are provided by each application component, and create the *application service* shapes.
3) Connect each *application component* with the corresponding *application service(*s) by means of the *realization relation*.
4) Connect each *application service* to the *business processes* that makes use of it. For this, use the *used by relation*. The BPMS is used by every business process. Hence, this BPMS service is connected to the large business process shape.
5) Determine the data objects that can be accessed via each application component.
6) Connect each *application component* to the corresponding *data objects* by means of the *access relation*.

Running example [Appendix D – Figure 7]
The BPMS is used for all business processes. Next to this, a CRM system, a financial system, and a DMS are used, as well as an external bank system. These systems have access to the required data objects, and provides certain services to the business processes.

Traceability: consistency with ArchiMate business process viewpoint

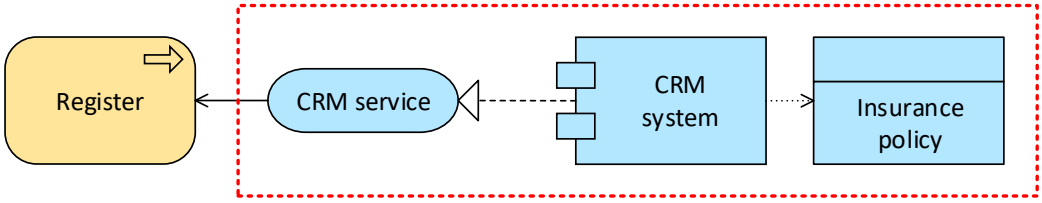


Figure 46: [Mapping] ArchiMate business process viewpoint <=> ArchiMate application usage viewpoint

The ArchiMate business process viewpoint is also included in the model of the ArchiMate application usage viewpoint. The only difference between these two viewpoints is the fact that the ArchiMate application usage viewpoint ***refines*** the ArchiMate business process viewpoint. Namely, all application (components) that are used within the business process are shown. In addition, it is shown what data objects can be accessed via the integrated systems. In Figure 46, this is highlighted by means of the red dotted transparent rectangle.

ArchiMate application cooperation viewpoint

For the execution of the business processes, the BPMS and the integrated systems collaborate in certain ways by means of interfaces (APIs). Thus, this viewpoint can be used to model the interactions that are possible/allowed between the all systems. The shapes are shown below in Figure 47.

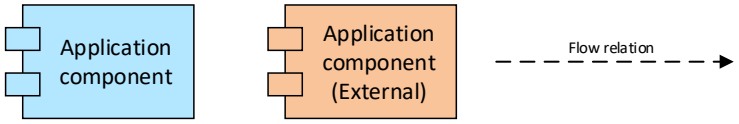


Figure 47: ArchiMate cooperation viewpoint shapes

The collaboration between *application components* is modelled by means of the *flow relation*. These type of relation is used to connect two application components that exchange information (messages). Thereby, there is an implication that an interface is used to realize the communication between two components. The interfaces could be modelled explicitly. A sub component can be modelled by modelling an application component shape within a larger application component shape. Sub components can also be connected to each other by means of the flow relation, which then also visualizes a collaboration. An external application component is colored orange.

Violations:
- We apply a different color to explicitly model an external application component.

GUIDELINES [main stakeholder: IT]

1) For each application component that is integrated with the BPMS, including the BPMS itself, create an *application component shape*. In addition, smaller *application component* shapes can be modelled within the larger BPMS application shape to visualize its internal components.
2) Determine the data/information (messages) that are exchanged between each application component.
3) Use the *flow relation* to visualize the corresponding exchanges.

Running example [Appendix D – Figure 8]
The BPMS is modelled as a large application component that consist of three sub components. The information (messages) that is/are exchanged between the BPMS and the integrated systems is visualized by means of the flow relations.

Traceability: consistency with ArchiMate usage viewpoint

Both viewpoints show the same application components / systems that are all involved in the execution of the business processes. The difference is the fact that the ArchiMate application cooperation viewpoint ***refines*** the former. Namely, it specifies the communication between the BPMS and the other systems. So, the information/data (messages) flows that are exchanged between them. In addition, it is possible to zoom in to a single system to get a high-level view on its internal components.

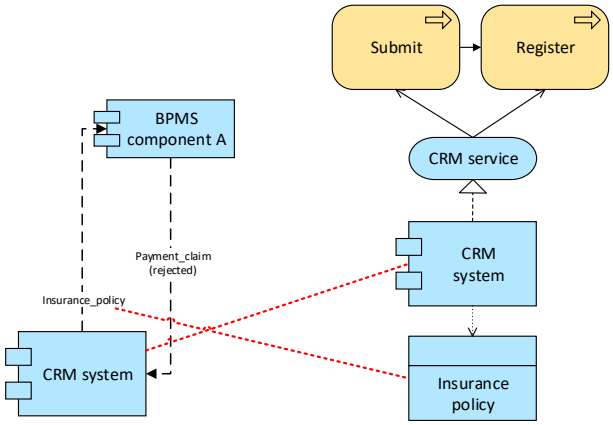


Figure 48: [Mapping] ArchiMate application cooperation viewpoint <=> ArchiMate application usage viewpoint

BPMN system choreography diagram

Both the ArchiMate application usage viewpoint and the ArchiMate application cooperation viewpoint can be used to derive the choreography regarding the interaction between the systems. In other words, the roles and capabilities of the BPMS and the integrated systems that are used in a certain order for the information and message exchanges. For this, the BPMN choreography diagram is used. This variant is then aimed at the order/path in which the systems interact with each other in order to exchange information / messages. The corresponding shapes are shown in Figure 49.

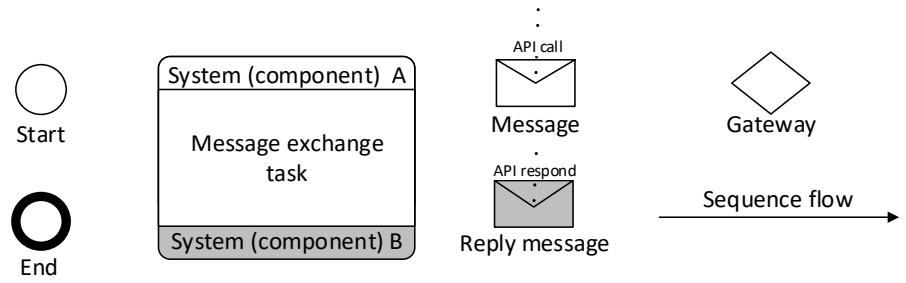


Figure 49: BPMN system choreography diagram

This diagram only shows the tasks of the BPMN process diagram that involve an interaction (message exchange) between two systems. These are then modelled in the order of execution by means of the *sequence flows*. The system that is the initiator / sender is colored white, while the system receiving the message is colored grey. The message could be a certain data object that is sent between two systems, or a service call regarding the request for a required data object. In case of a service call, the interface / service type is annotated at the dotted line that is used to connect the message shape with the tasks shape. Similar to the BPMN system choreography diagram, realizability is an important factor. In this case, it means that at least one system of a certain message exchange was involved in the previous message exchange.

Violations:

- Originally, this diagram is not used to model the flow of system interactions. In most cases, both the sender and receiver is a person. However, due to the fact that both the sender and receiver is in fact a role, it can also be (a component of) a system.
- To distinguish this system-oriented BPMN choreography diagram from the process-oriented BPMN choreography diagram that was described in the previous sub paragraph, we have named it BPMN systems choreography diagram.
- Originally, it is not applicable to annotate a service type near the message shape.

GUIDELINES [main stakeholder: Business, IT]

1) As a starting point, consider the BPMN process choreography diagram.
2) Within each message exchange task, change the *sender* to the application component that sends the corresponding message. The same goes for each *receiver*. ***However, if a certain message exchange task does not entail communication between the persons via the BPMS, but only between the BPMS and the persons (and vice versa), either the sender or receiver is the corresponding person instead of a certain system component. For example, when a customer uploads a scan of a hardcopy form to the BPMS, the customer is the sender, whereas the BPMS (component) is the receiver.***
3) In the middle of the *dotted line* that is used to connect each message shape to the corresponding message exchange task, mention the *type of API communication*.
4) Check the following properties:
    a. The modelled message exchange tasks need to adhere to the flows that are allowed between the application components, according to the ArchiMate application cooperation viewpoint.
    b. The *realizability* by ensuring that, for each message exchange tasks, expect the first one within the entire flow, at least one application component was involved in the previous message exchange tasks.

Running example [Appendix D – Figure 9]
Only the tasks that involves an interaction between two systems have been included to the diagram. Most message exchanges occur between two different components of the BPMS. In most cases, during each tasks, only a message is sent by the sender. In case of a request to, for example, a CRM system to exchange the insurance policy, also a respond message is shown.

Traceability: consistency with BPMN process choreography diagram

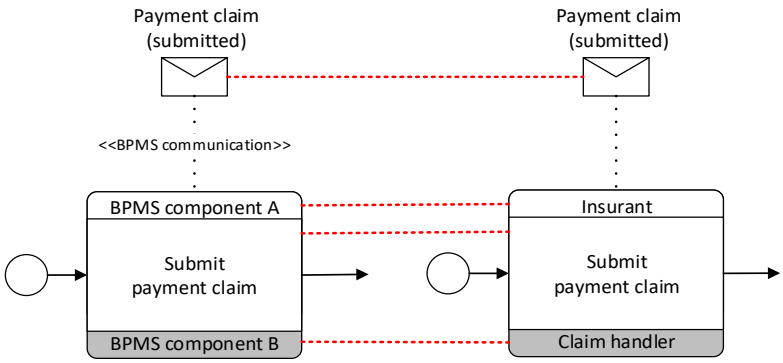


Figure 50: [Mapping] BPMN system choreography diagram <=> BPMN process choreography diagram

The BPMN system choreography diagram is created ***based on*** the BPMN process choreography diagram. Basically, it replaces the names of the persons with the name of the components that involved in the message exchange task. In addition, regarding the communication between the components, the type of communication is added.

Traceability: consistency with BPMN process diagram

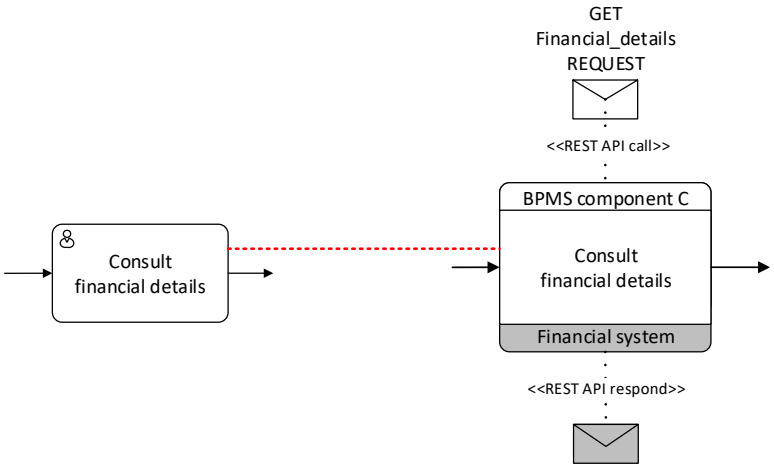


Figure 51: [Mapping] BPMN process diagram <=> BPMN system choreography diagram

The BPMN choreography diagram of the system interactions is a specialization, created ***based on*** the BPMN process diagram. It only visualizes the tasks from the BPMN process diagram that includes interaction between two systems, and in what order this is done.

Traceability: consistency with ArchiMate application cooperation viewpoint

The ArchiMate application cooperation viewpoint determines the possible interactions between the systems. For this, the information/data flows show the interactions. Therefore, the BPMN system choreography diagram ***adheres to*** the ArchiMate application cooperation viewpoint. For example, in Figure 52, the fragment from the ArchiMate application cooperation viewpoint shows a flow called *Financial_details* between component C of the BPMS and the financial system. Thus, this determines that these component can exchange the *Financial_details*. Despite the fact that only one direction arrow

is shown, there is an implication that, in the opposite direction, there is a flow which is a request to the financial system for exchanging the requested data.

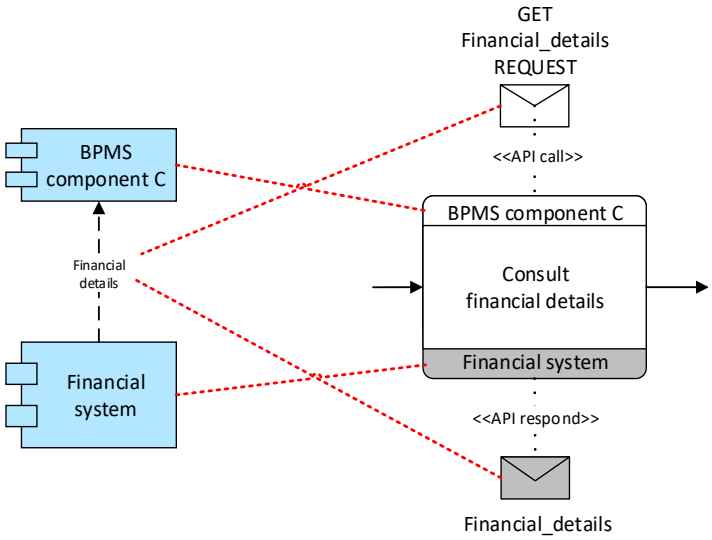


Figure 52: [Mapping] ArchiMate application cooperation viewpoint <=> BPMN system choreography diagram

UML class diagram

Furthermore, to visualize and describe the data model with the data objects and the corresponding structures, the UML class diagram is used. The corresponding shapes are shown in Figure 53. Initially, the UML class diagram is applicable to the *Application components & orchestrations* on the *Process/application decomposition level*. However, due to the fact that data object are in fact also somehow presented at the other architecture levels and corresponding viewpoints of the ADL, in practice, the UML class diagram will be applicable to all architecture levels. Though, the essence is positioned within the *Application components & orchestrations* viewpoint.

A data object is modelled as a *class* which has one or more *attributes*. There are different possible types of relations between the classes, including an association, aggregation, and dependency. The relations have a certain cardinality/multiplicity. For example, zero or more (0..*), which means that a certain class can be associated to none or infinite instances of another class.

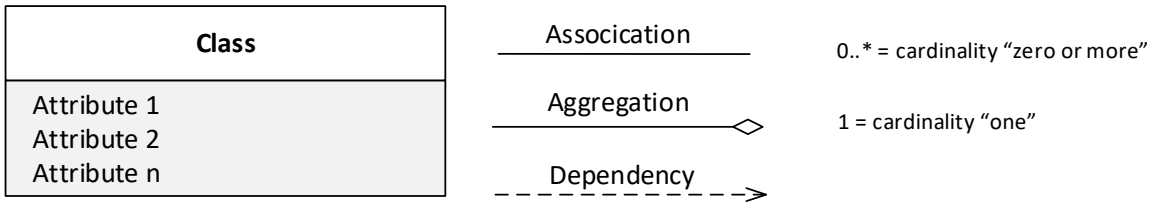


Figure 53: UML class diagram shapes

GUIDELINES [main stakeholder: Business, IT]

1) Indicate all *data objects* that are visualized within the ArchiMate application usage viewpoint. If applicable, list some additional *data objects* that have not been modelled yet.
2) List all message exchanges.
3) For each data object, create a *class* shape, and write down the corresponding *attributes*.
4) Connect the correct *data objects* to each other by means of the right *relations* and corresponding *cardinalities*.

Running example [Appendix D – Figure 10]
Each payment claim is unique. Therefore, a payment claim is submitted by only one Insurant. Thus, there is a one to one cardinality. An insurant has only one set of account details, and can have only one insurance policy. Vice versa, multiple insurants can have the same type of insurance policy. The correctness of a payment claim depends on a set of payment claim requirements a payment claim needs to meet. Furthermore, a payment claim has certain financial details, and the reparation invoice that determines the amount of financial compensation. An important attribute of a payment claim is its status. Within the name of every information/data flow or message flow that represents a payment claims, the status is indicated between brackets. For example, payment claim (submitted).

Traceability: consistency with all other models

The data objects that are specified within the UML class diagram are (indirectly) involved in the other viewpoints. The UML class diagram is used to ***refine the data structure o***f the other viewpoints. In general, the names of classes / data objects are mentioned in most viewpoints. Especially, within both the process diagrams and choreography diagrams which include the information/message flows that are in fact the data objects involved in the payment claim handling process, see Figure 54. Moreover, within the name of each communication flow, the status can be mentioned between brackets, such as the payment claim that has different statuses regarding the car insurance company running example. The relations / structure between the data objects are then specified within the UML class diagram. This structure mainly influences the way of integrating the BPMS with other systems.

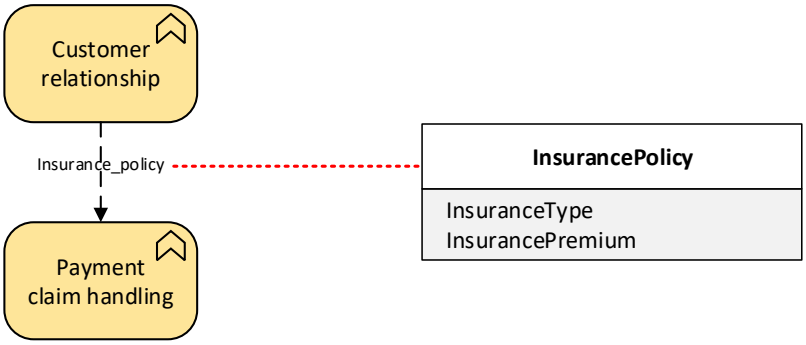


Figure 54: Data objects within other viewpoints

6.8 BPMS implementation level – specifications & guidelines

This architecture level is aimed at the specification of what to build and how to build, tailored to the specific properties of the corresponding BPMS. This then results in platform-specific models which include specific names of components, databases, workflows etc. of the BPMS. Thus, the components, interfaces, integration services etc. of the corresponding BPMS are further elaborated at the lowest architecture level of the ADL.

6.8.1 BPMS design

It depends on the internal structure of the BPMS how this architecture level is elaborated and linked to the business domain level, and process/application decomposition level. Though, regarding the BPMS design, the most important model is the internal view on the components of the BPMS, and the relation between the BPMS as a whole, and the links with the integrated systems. For this, the UML component diagram is used. A selection of the corresponding shapes are shown in Figure 55.

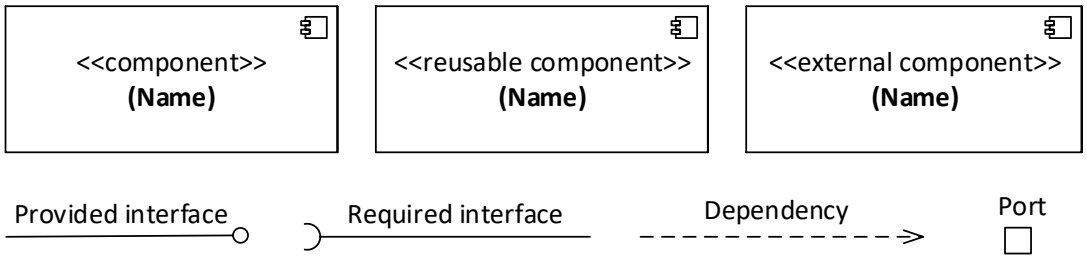


Figure 55: UML component diagram shapes

A *component* is an application/system as a whole, or a certain part of the system, for example, a certain module / coherent piece of functionality. Three different types of component have been defined as a so-called <<stereotype>>. A *component* is any functional part of the BPMS, or an integrated system as a whole. A component of the BPMS might have a *dependency* relation with any *reusable component* of the BPMS, which can be, for example, a certain business rules set that is applicable to multiple cases. The BPMS can also be integrated with systems from external organizations. (Parts of) these systems can be visualized as an *external component*. All types of components communicate with each other by means of interfaces. There are two types of interfaces that can both be further specified, for example, as a SOAP interface. An interface with a closed circle represents a *provided interface* of a certain component. A semi-closed circle represents a *required interface* that makes use of a *provided interface* of an application service that can be used by other systems. For the *required interface*, a certain (user) input is needed. In addition, a *port* can be used to emphasize/expose the interaction point of an interface, and can be, for example, a bi-directional port. Next to the interface shape, its name is shown. Usually, this name represents the data object that is exchanged through the corresponding interface.

GUIDELINES [main stakeholder: IT]

1) Consult the *ArchiMate application cooperation viewpoint*. This viewpoint can partly be extended and/or changed to the UML component diagram that visualizes more details on the internal structure of the BPMS. For example, *reusable components* that are applied, specific types of interfaces, integration services etc.
2) Determine and model the *components* of the BPMS, *reusable components* of the BPMS, *external components* that are used from external organizations, if any.
3) For all components that are part of the BPMS, create a large *component* shape that represents the BPMS. Within this large shape, put the smaller component shapes.
4) Connect the *components* of the BPMS to the applicable *reusable components* by means of the *dependency* relation.
5) Determine which components communicate with each other. Then, indicate which components provide a certain service / functionality, and which components make us of a certain service / functionality. For the former, use a *provided interface* shape. For the latter, use a *required interface* shape. Then, connect these components with each other. For the communication between the BPMS and the integrated systems (which are modelled as components), a port can

be used to emphasize bi-directional communications. In addition, give a name to each interface. Usually, this is the data object that can be exchanged via the interface.

6) If the BPMS does not apply BPMN for modelling and configuring the business process models, translate the BPMN process diagrams to the BPMS's business process notation, and create them within the BPMS.
7) For each *scenario*, apply the corresponding business rules.
8) Determine and build the components of the BPMS that need to be used to realize the specified system interactions from both the *ArchiMate application cooperation viewpoint* and the *BPMN system choreography diagram*.
9) Create the *data structure* based on the *UML class diagram*.
10) Develop / configure the remaining properties of the BPMS.

Running example [Appendix D – Figure 11]
The created UML component diagram is quite similar to the ArchiMate application cooperation viewpoint. The most important difference are the fact that the types of *interfaces* can be specified in more details, and, in general, a more detail view on the internal structure of the BPMS can be created. In this case, two *reusable components* are shown. The other BPMS components depend on them. Furthermore, the bank system has been modelled by means of an *external component* shape. *Ports* have been added to visualize bi-directional communication between the BPMS and the integrated systems via the interfaces.

Traceability: consistency with all other models

Each BPMS has a different internal structure. Though, as described in the theoretical background, a BPMS at least contains (explicit) process definitions / models that are configured for the execution of the business processes. It then depends on the BPMS vendor and the type of BPMS what kind of database, interfaces, standards, reusable components etc. are applicable. This influences the structure and contents of the viewpoints of the ADL. As mentioned before, at least the UML component diagram can be used to create and specify the BPMS implementation level. To make this clear, links can be made between all models, and how they are built / ***refined*** on the corresponding BPMS that is used. For example:

- The translation of the ArchiMate application cooperation viewpoint to the actual implementation of the BPMS including the specific type of standards, interfaces etc. that are provided by the BPMS;
- A certain reusable component of the BPMS which is part of the ArchiMate application cooperation viewpoint;
- A certain component of the BPMS that realizes a certain business process etc.

7. Validation

The previous chapters focused on the design of the intended ADL. In this chapter, we discuss the validation of the practical applicability of the ADL. Eventually, answers have been provided to the last sub research question (SRQ):

- ***SRQ5)*** *Is the designed ADL valid and applicable in practice for the desired purposes?*

7.1 Approach

To validate the practical applicability, we conducted a case study. This entailed that we applied the ADL in practice within a real project. This was done in a structured way, which is described below.

Case study project

After the intended ADL was designed, we selected one of the development projects of BPM Company to serve as our case study project. In consultation with the daily supervisor, a project was selected which lacked of clear, solid architecture documentation, and which was relatively small (but not less complex) in comparison to other projects. In this way, we could better indicate the benefits (effectiveness) of clearer and more solid architecture documentation. In order to be able to create the architecture models, we collected applicable information on the selected project, including contextual information, and application specification(s) documents.

The selected project was focused on optimizing (partly automating) several administrative processes within an organization. The BPMS application has been developed on the Pega Platform in order to replace the functionality of several existing systems. The main objectives were reducing the amount of paper work, and preventing manual user input errors within the existing systems.

Applying the ADL

By means of the collected information, we created the architecture models. For this, we followed the guidelines from the chapter 6. Eventually, the created models were then put and described in a potential ADL document template, see Appendix F. Due to the fact that creating and describing the models was more difficult / time-consuming than expected, only a certain part of the case study project was specified by means of the ADL. Though, we have selected and elaborated this part in such way, that it was suitable and sufficient to validate the essence of using the ADL in practice. Moreover, it was important that the elaborated part could be used to envision the extent to which it is suitable to elaborate the remaining parts of the project in the same way, as well as for other (future) projects.

Semi-structured validation interviews

When the models had been created, we conducted two separated semi-structured validation interviews. One interview was conducted with a business architect. A second interview was conducted with a system architect / developer respectively. In this way, the opinions from both the business perspective and IT perspective were obtained. Both interviewees were involved in the case study project. Hence, they were familiar with the context of the project, and were able to correctly assume what could have been the added value of applying the ADL during the project. The interview questions, the created models and the corresponding guidelines were provided to the interviewees before the interviews were conducted. In this way, more substantiated answers were obtained due to the fact that the interviewees could already judge the models, think of possible improvements, and their own questions in advance.

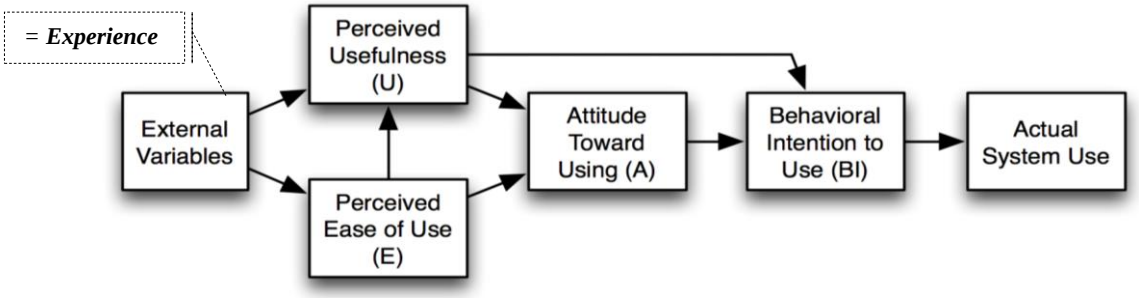


Figure 56: Technology Acceptance model (TAM). Adopted from Davis, Bagozzi, & Warshaw (1989, p. 985)

To create the interview protocol, and create a structured validation approach, we have applied the Technology Acceptance Model (TAM), which is shown in Figure 56. The TAM focuses on several variables that can be accessed to determine/validate the acceptance of new technology (Davis, Bagozzi, & Warshaw, 1989):

- *External Variables*: external factors that (indirectly) influence the other variables;
  - The most important external factor: *Experience;*
- *Perceived Usefulness (U)*: extent to which the ADL enhances the development process;
- *Perceived Ease of Use (E)*: extent to which the ADL can be applied without many effort;
- *Attitude Toward Using (A)*: feelings/expectations of applying the ADL in practice;
- *Behavioral Intention to Use (BI)*: willingness of (partly) applying the ADL in practice;
- *Actual System Use*: actual usage of the ADL in practice.

In Figure 56, the arrows visualize the links between the variables. For example, the arrow from *Perceived Ease of Use* to *Perceived Usefulness* means that the former influences the latter in a certain way. More precisely, the easier it is to use the ADL, the more useful it will be. Despite the fact that the ADL itself is not a technology / software system, the variables were considered as suitable formal indicators for validating the practical applicability of the ADL.

The TAM is partly similar to the Unified Theory of Acceptance and Use of Technology (UTAUT), which targets more/different variables. These additional variables were not relevant for validating the ADL due to the fact that those variables are more technology oriented while, as mentioned before, the ADL itself is strictly not a technology. Therefore, we did not use the UTAUT (Venkatesh, Morris, Davis, & Davis, 2003). The same goes for both version 2 (Venkatesh & Davis, 2000), and version 3 (Venkatesh & Bala, Technology acceptance model 3 and a research agenda on interventions, 2008) that extend the initial TAM in terms of the number of variables and dependencies between them.

Though, both version 2 and 3 of the TAM contain a variable called *Experience*. We have considered this variable as the most important external variable for validating the ADL in terms of the experience with using a BPMS for low-code development, and creating architecture models by means of an ADL. Furthermore, we did not use the Method Evaluation Model (Moody, 2003) for the validation due to its additional variables called *Actual Efficiency*, and *Actual Effectivity*. Namely, in order to assess these variables correctly/effectively, the ADL must already be applied by practitioners.

In Appendix E, the full validation interview protocol can be found. We have used the variables of the TAM to categorize the validation interview questions. Namely, with the exception of *Actual System Use*, we have assessed all variables above through one or more interview questions. This is indicated in the interview protocol. The aforementioned variable was excluded due to the fact that to correctly assess this variable, the ADL must be applied in practice already. This was not the case at the time of conducting this research.

Both validation interviews were recorded and started with a brief personal introduction, and an explanation of the objectives and structure of the interview. Then, we asked the first questions to access the *Experiences* and other *External Variables*. Then, we explained the essence / objective of the ADL. After that, a step-by-step walkthrough was used to discuss the created models. This entailed that, for each model, we explained what guidelines were used to create the model, as well as the meaning and objectives of the model, and the links with other models of the ADL. By means of the step-by-step walkthrough, the *Perceived Usefulness*, and *Perceived Ease of Use* were assessed through several questions. While asking these questions, both interviewees also asked some questions to clarify certain properties of the ADL. Hence, it resulted in small discussions on the created models. After discussing each model, we asked some remaining questions regarding the *Attitude Toward Using*, and *Behavioral Intention to Use*. These remaining questions were aimed at the ADL as a whole, so, not at a particular model.

7.2 Results & discussion

In this paragraph, the results of the validation interviews are elaborated. For each variable, we discuss the answers of both interviewees.

7.2.1 External variables (experiences)

The following questions were asked:

- *How many years of work experience do you have within your current field?*
- *Do you have experiences with applying ADLs? If yes, which one(s)?*
- *What other external factors might influence the perceived usefulness and perceived ease of use of the ADL in practice? How? (Time pressure, ownership issues, experiences etc.)*

There were several external variables that influence both the perceived usefulness and perceived ease of use. At the time of conducting the interviews, both interviewees were working at the senior level. They both had at least four years of experience working with BPMSs / low-code development platforms. Moreover, they both had experience with applying ADLs in practice. Mostly, UML is used to create architecture models.

According to both interviewees, an important external factor that influences the *Perceived Usefulness* and *Perceived Ease of Use* of the ADL is the fact that the customers often force the architects to make use of a certain ADL and/or documentation standard. This implies that it can be difficult to apply another standard (our ADL). This partly deals with another external factor: cultural differences. Though, in most projects, the architects are not required to create and document architecture models. Instead, the system architect voluntarily creates models (as reference work) to clarify certain aspects of the application. Next to a forced standard, another important external factor that was mentioned is time-pressure that goes along with the budget of the customers. This means that a lack of time and money can also prevent the architects to create architecture models. Therefore, there is an implication that the ADL must not be too formal / comprehensive. Furthermore, based on the complexity / size of the project, it needs to be considered what models are worth to be created. Namely, according to the systems architect, for the case study project, several models, including the ArchiMate application usage viewpoint, were quite simple / straightforward.

7.2.2 Perceived Usefulness

In Table 9, a summarized overview of the answers to each question regarding the Perceived Usefulness is given. After this table, we discuss the corresponding results. The models are mentioned by means of abbreviations. For example, the ArchiMate organization structure viewpoint is mentioned as *AOSV*, and the BPMN process diagram is called *BPMNPD*. The same goes for the other models, as well as Table 10.

Table 9: Perceived Usefulness – validation results

Model	Business architect	System architect
How / for what purposes and at what moment exactly would you use this model within a project?		
AOSV	• Deriving business functions, and persons. • Organization vs. users. • Determining the ratios and dependencies between departments (business functions) and the corresponding persons. • Needs to be included to the PSA for determining the project scope. For the case study project, this model could be used to indicate all departments that are involved in the process that is optimized by the BPMS.	• For each model, the target audience needs to be clear. • As a stakeholder matrix. • Determining the points of improvement. Regarding the case study project, by means of this model, it can already by discovered that, for example five different HRM systems are used for HR. • Based on the experiences of the system architect, this model is more IT-oriented. The business (board of directors etc.) will prefer a regular organogram.
ABFV	• Needs to be included to the PSA for determining the project scope. • Visualizing high-level information flows and responsibilities per business function.	• As a communication means to establish a clear interaction with the product owner in terms of deriving high-level properties / requirements of the application. • Needs to be included to the PSA
ABPV	• Creating a high-level view of the business process, outside of the BPMS.	• This model does not add much to the high-level process view in Pega. It is

		preferred to create these process views in Pega immediately.
BPMNPD	• Creating detailed (explicit) business process models for determining certain criteria, including both the input criteria and output criteria of each process task. • Conducting certain analysis.	• A suitable communication means for alignment with the business. • It is preferred to create the BPMNpd first before actually creating the corresponding process flows in Pega. • It is preferred to create process models in conjunction with process owners and users.
BPMNPCD	• Establishing a link between business process and information in a process-oriented way.	• It was hard to say in what situations this model will be relevant. For the case study project, it was less relevant.
BPMNSCD	• Determining the role of each system, and in what order they are used next to the BPMS.	• A high-level, process-oriented view on the system interactions, on top of optional more detailed UML sequence diagrams. • Perhaps, this model already contains too much details to correctly derive the required BPMS components.
AAUV	• Creating a mapping with the business functions in terms of the systems assigned to the relevant business functions.	• Needs to be included to the PSA. • As reference work afterwards. • For the case study project, this model was an overkill due to the fact that only three other systems were integrated with the BPMS. This then did not result in a complex application landscape.
AACV	• In conjunction with the BPMNSCD.	• As a reference work afterwards.
UMLCLASS	• Determining the data structures.	• Needs to be included to the PSA. • Needs to be updated/refined multiple times during the project.
UMLCOM	• Determining the required interfaces, services, and other (reusable components) of the BPMS.	• Needs to be updated/refined multiple times during the project. • Creating a clear view on the reusable components, after the business processes have been modelled.
What benefits / added value do you envision when this model is used in practice?		
AOSV	• Insights into the organization where the BPMS is implemented. • It prevents that wrong assumptions are made regarding the exact part(s) of the organization where the BPMS will be used.	• Possible to create a clear GAP-analysis between the old situation and new situation, and an impact-analysis on the application. • Clear view on the scope of the application for the developers / systems architects. Regarding the case study project, it could give a clear view on the organization parts within the scope of the application.
ABFV	• Discussions on possible links between the applicable business functions can be prevented. If there is a link between two related business functions, it is already clear what the requirements and consequences are for the scope and design of the BPMS application. For the case study project, it was not discovered on time that, in reality, there is an important link between two relevant business functions.	• High-level view on all communication flows. • It can prevent tunnel vision regarding the project scope. Regarding the case study project, the stakeholders were not aware of an important link between two business functions.
ABPV	• The roles of the process participants are visualized explicitly within the process	• This model does not have much added value. The only benefit is the fact that, in contrast to the Pega process flows, this

	models. Pega does not provide this functionality.	model can be used to explicitly visualize the business roles that are involved in the process.
BPMNPD	• The business processes can be modelled in a comprehensive/detailed way.	• In contrast to the implicit BPMN based notation of the Pega process flows, the BPMNPD can be used to create more explicit business process models. • The detailed process view can prevent discrepancy between stakeholder concerns as much as possible. Especially, between process owners and the users.
BPMNPCD	• Very relevant for modelling information intensive business processes in a concise way.	• It was difficult to determine the added value of this model. At least, for the less complex application landscape of the case study project, it is less relevant. Though, it will be more suitable in case many different organizations communicate with each other via a BPMS.
BPMNSCD	• Very relevant for modelling the system interaction within information intensive business processes in a concise way.	• It provides a clear process-oriented overall view on all system interactions that occur for the information exchanges within the system.
AAUV	• A clear mapping can be made with the corresponding business functions of each system next to the BPMS.	• This model is more relevant for complex application landscapes that entails more integrated systems. For the case study project, it was already clear that, next to the BPMS, only three others systems are used within all business processes.
AACV	• Clear insights into the dependencies between the BPMS and the integrated systems.	• It gives a clear view on the communication flows and dependencies between the BPMS and the integrated systems.
UMLCLASS	• Clear, standardized way of visualizing the data structure.	• This is one of the most important models, due to the fact that the data structures determine most properties of how information can be collected and used within the application.
UMLCOM	• Clear view on the components of the BPMS, especially, to visualize the reusable components.	• Clear view on the available reusable components, and what interfaces and services are required for using them.
What properties/aspects do you miss within this model and/or could be adjusted?		
AOSV	• An explicit visualization of the hierarchy such as a real organogram.	• Links between the business actors in order to visualize a certain hierarchy.
ABFV	• Perhaps, multiple abstraction levels could be applied. • Regarding the case study project, an important link between a business function and a business role was not modelled.	• Regarding the case study project, several important links between two business functions were not included.
ABPV	• Regarding the case study project, the names and responsibilities of a few roles were not modelled correctly.	• -
BPMNPD	• In contrast to other projects of BPM Company, the tasks of the business process of the case study project are not assigned to a certain person/roles. Moreover, many tasks are initiated by the BPMS, and a lot of business process are	• In contrast to other projects of BPM Company, the tasks of the business process of the case study project are not assigned to a certain person/roles. Moreover, many tasks are initiated by the BPMS, and a lot of business process are

	carried out in parallel, outside of the BPMS. For this reason, in general, more tasks need to be assigned to the BPMS. This can be done by adding a pool or lane for the tasks of the BPMS. • Adding a status to each data object. For example, a certain request that has been approved → *requested (approved)*.	carried out in parallel, outside of the BPMS. For this reason, in general, more tasks need to be assigned to the BPMS. This can be done by adding a pool or lane for the tasks of the BPMS.
BPMNPCD	• If possible, it somehow needs to be merged with the *BPMNSCD* to correctly visualize the communication flows between systems and roles and vice versa.	• -
BPMNSCD	• If possible, it somehow needs to be merged with the *BPMNPCD* to correctly model the communication flows between systems and roles and vice versa. • The integration push vs. pull could be visualized more explicitly. This means that it needs to be clear at what moments the BPMS asks for information from the process (the persons) and vice versa.	• At an additional lower abstraction level, the UML sequence diagram could be used to elaborate the system interactions in more details.
AAUV	• -	• -
AACV	• -	• -
UMLCLASS	• -	• An additional UML class diagram at the enterprise level which can be plotted on the processes and systems.
UMLCOM	• The correct directions of each interface.	• Regarding the case study project, not all reusable components were correct.

In general, most models of the ADL are quite useful to create and would have an added value as a communication means between different stakeholders when the models would be created both during and at the start of) a project. More precisely, creating the models and discussing them with the right stakeholders would create better scoping (less tunnel vision), more clarity across different architecture viewpoint during a project, and, therefore, less discrepancy. Especially, due to the clear traceability/interrelation links between the models. If the models were created, and, thus be available at the start of the case study project (which was not the fact), a lot of problems/discussions between stakeholders could be prevent afterwards.

Both interviewees agreed on the fact that at least the *ArchiMate business function viewpoint* must be included to a PSA. The reason for this is the fact that this model is quite suitable to determine the communication flows from a high-level point of view. This then can be used to recognize both the responsibilities of and dependencies between the different business functions, and the underlying systems that contain relevant information. Based on this information, the requirements and consequences of the design of the application can be specified. Regarding the case study project, creating the *ArchiMate business function viewpoint* would have prevented the fact that, during a later phase of the project, it was discovered that, apparently, there is an important dependency link between two core business functions. This resulted in project delay due to changes to the scope of the application, and, thus, changes to the requirements for the design of the application. Next to the ArchiMate business function viewpoint, in contrast to the business architect, the system architect also considered the UML class diagram to be important for the PSA. The reason for this is the fact that the data structure and the corresponding data models are one of the core aspects of an application. Furthermore, to save time during the project, it will be useful/effective to elaborate the BPMS design UML component diagram directly after the business processes have been modelled. In this way, reusable components that are applicable to the processes can be indicated as soon as possible in the initial phase of a project. While iteratively developing the application, the UML component diagram is continuously updated.

Regarding the correctness of the created models, which was also considered as an indicator for determining the perceived usefulness, in general, the created models adhere to the most important

requirements/specifications of the case study project's application. However, one mistake was made regarding the business processes and the corresponding choreographies. Apparently, in contrast to other projects of BPM Company, the tasks of the business process of the case study project are not assigned to a certain person/roles. Moreover, there is no communication between the persons via the BPMS application. This communication occurs outside of the application, and is therefore not logged. Thus, there are only communication flows between systems, and between the BPMS application and the persons (and vice versa). Moreover, there is a standard pattern that occurs within the application:

1. *A certain task need to be done, for example, signing and uploading a form by several roles.*
2. *The BPMS application notifies this to all corresponding roles.*
3. *All corresponding roles carry out the task (individually) in parallel.*
4. *Certain checks are performed by one or more other roles. If necessary, go back to step 2.*
5. *The BPMS application checks: tasks done.*

Hence, in short, in most cases, the BPMS application triggers the start of the business processes. This would mean that both the *BPMN process diagram* and the *BPMN process choreography diagram* needed to be adjusted in such way that, in most situations, the BPMS application initiates the communication between the persons among the execution of the business processes. For the *BPMN process diagram*, this would mainly mean that a separated pool (or lane) for the BPMS application is created. This pool then contains all (automated) tasks that are carried out by the application. For the *BPMN process choreography diagram*, this will mainly entail that it needs to be partly merged with the *BPMN system choreography diagram*. More precisely, a choreography diagram needs to be created that visualizes the communication between the BPMS and the persons and vice versa. Both systems and persons are considered as roles within choreographies. Moreover, both the sender and receiver of a message exchange is are roles. Hence, according to the syntax of both diagrams, these adjustment are allowed.

Regarding the aforementioned mistake in terms of the communication (message exchange) between roles and systems of the case study project, the system architect found it difficult to determine the *Perceived Usefulness* of the *BPMN process choreography diagram*. Namely, this diagram would be more suitable for a project that entails more communication between separated organizations, and can get quite complex in terms of the communication flows. Though, the complexity partly depends on the abstraction levels that are specified. Furthermore, in contrast to the business architect, the system architect did clearly saw the added value of the *ArchiMate business process viewpoint*, which is preferred to be created in Pega immediately. In addition to this, though the system architect acknowledged the added value of the *BPMN process diagram* for creating more detailed (explicit) process models.

7.2.3 Perceived Ease of Use

In Table 10, we present a summarized overview of the results regarding the Perceived Usefulness.

Table 10: Perceived Ease of Use – validation results

Model	Business architect	System architect
To what extent do you think this model can be created/used and understood without too much time and effort?		
AOSV	• As an alternative for a regular organogram, the purposes of the model are clear. • It is easy to derive organizational departments, business functions, and responsibilities of process participants. • Within this model, in contrast to a regular organogram, a possible hierarchy is not visualized explicitly.	• Easy to apply based on / next to an existing organogram. • Easy to understand like an organogram. Especially, for the IT stakeholders due to the IT-oriented focus. However, despite the fact that a certain hierarchy has been included, this is not visualized explicitly.
ABFV	• Any person who is familiar with modelling dependencies between business functions, including the interviewed business architects, will have not many effort to create and understand this model.	• Easy to apply and understand by most stakeholders.

	• Though, it was not completely clear when a communication flow is modelled between two business functions.	
ABPV	• The corresponding abstraction level for visualizing the business process is similar to the process flows within Pega. • Initially, it was not completely clear why a business event is modelled outside the process, and why only one business actor has been added. Though, actually, it is not necessary to already assign the business roles certain business actors.	• Easy to apply and understand.
BPMNPD	• This is the standard straightforward way of modelling business processes. It is quite similar to the Pega workflows. • Initially, the purpose and the link between the different abstraction levels was not completely clear. Especially, at what abstraction levels the alternative stages (rejections) are specified.	• Due to the fact that it is similar to the Pega process flows, and for other BPMSs, it is the standard ADL for creating the business process models, it will be easy to apply and understand.
BPMNPCD	• It is clear how the order of the information usage is visualized as a process flow, based on the BPMN process diagram. • If the links between the BPMN process diagrams is made clear, and possible abstraction levels are meaningful, this model will be easy to use.	• Easy to apply and understand all information that is exchanged during business processes in a concise way.
BPMNSCD	• It is clear how the order of the application usage is visualized as a process flow, based on both the BPMNPD and BPMNPCD. • Initially, it was not clear how for each message exchange task the initiated system is modelled.	• Easy to establish and understand the links between the business processes and the corresponding system interactions.
AAUV	• Easy to apply and understand.	• Quite straightforward.
AACV	• Easy to apply and understand.	• Easy to apply and understand.
UMLCLASS	• Easy to apply and understand.	• Every architect must be able to understand a data model that is visualized as an UML class diagram.
UMLCOM	• Initially, it was not clear whether or not / how the modelled interfaces visualize two-way communication flows.	• Easy to apply and understand.

In general, based on the explained guidelines, the models would be easy to understand and adopted/used without many effort. Especially, when the user of the ADL is (partly) familiar with BPMN, ArchiMate, UML, and other related/similar ADLs. By means of both the high-level architecture decomposition model and the high-level ADL model structure, it can be easy understood how the different models are related to each other, as well as the scope/purpose of each model regarding the specification of communication flows within the application landscape of a BPMS. However, to increase the understandability of each model, one must always add a clear description of the model. Moreover, it needs to be clear for what stakeholder(s) / target audience a certain model is created.

There were only some small issues regarding the ease of use (understandability of several models. This was mainly about the purpose and links between different abstraction levels, as well as the meaning of several model shapes. These issues were solved after the guidelines were consulted once again, and when possible adjustments to the guidelines were given.

7.2.4 Attitude Toward Using

The following questions were asked:

- *What are you feelings/expectations when the ADL will be applied in practice?*
- *How could the use of the ADL be stimulated/fostered?*

To foster/stimulate the use of the ADL in practice, awareness of the added value of applying an ADL needs to be created in an effective way. For this, a (lead by example) workshop is a suitable means. When the ADL is applied in practice, it is expected that mainly the project scoping will be improved. Moreover, more attention/awareness of other architecture levels/domains can be obtained. This means that, for example, the business can get a better view on the requirements and consequences for the design of an application in case other business functions need to be involved. Furthermore, as mentioned before, in most situations, the customer organization of a project already determines the ADL and/or documentation standard that needs to be applied. Though, if no particular architecture documentation approach is forced, it is expected that the ADL will serve as a consistent approach for creating architecture descriptions that can reduce the number of discussions (discrepancy) between stakeholders.

7.2.5 Behavioral Intention to Use

The following questions was asked:

- *What reason(s) could you give for (not) using the ADL?*

Overall, both interviewed architects were willing to make use the ADL in practice due to its usefulness and ease of use. By means of the elaborated case study project, it was clear how the ADL will improve the development process in terms of specifying communication flows. Moreover, due to the process-oriented focus in conjunction with the focus on information/data and functionality, the ADL is suitable to specify the most important aspects of a BPMS and the running application in a process-oriented. In addition to this, the clear traceability of the consistency between the different models is considered as a strong property of the ADL. Furthermore, within the scope of the ADL, no other relevant models were missed.

7.3 Summary

Based on the results of the case study validation, we can conclude several points for answering ***SRQ5***:

- *Is the designed ADL valid and applicable in practice for the desired purposes?*

If the designed ADL is perceived as being useful to be applied in practice in an easy way with many effort, it can be concluded that the ADL is applicable in practice for the desired purposes: the specification of the communication flows within the application landscape of a BPMS. The conclusions are formulated by means of briefly evaluating the answers to the questions that have been asked to target the selected variables of the Technology Acceptance Model (TAM), see Figure 56. This also includes an evaluation of how the variables have influenced each other.

So, first, the *External variables*. Due to the fact that both interviewed architects have *Experience* with low-code application development on a BPMS four at least four years, they both were familiar with the process-oriented perspective of a BPMS, and how this provides the low-code development capabilities. Moreover, they both have *Experience* with applying several ADLs, including UML. This has prevented that the *Perceived Usefulness* and *Perceived Ease of Use* could not be assessed correctly in the desired way within the time that was available for the case study validation. Next to *Experience*, other external variables that need to be considers are the complexity / size of the project, time-pressure, and the fact that it is not always possible to apply an own ADL and/or documentation standards

Regardless of several minor differences between the business architect and the system architect due to the different working area, they agreed on the fact that the designed ADL is perceived as a useful means that will result in several benefits when it is applied in practice. In terms of specifying the communication flows within the application landscape of a BPMS, the ADL can be used to create the most applicable models. There were no other models that must be included to the specification of the ADL. However, for several models, especially the BPMN process choreography diagram, it needs to be determined, based on the project size / complexity, to what extent it will be relevant to created them.

Moreover, both architects think the ADL will be easy to use. There were some small uncertainties regarding the purposes and abstraction levels of different models. But, by means of consulting the applied guidelines in more details, and discussing possible adjustments to the guidelines, these issues were solved. Moreover, most of the potential adjustments to the ADL only required several adjustments to the guidelines in order to make there clearer and more generic. Hence, this had a positive influence on their *Perceived Usefulness*.

Regarding the *Attitude Toward Using*, the positive *Perceived Usefulness* and *Perceived Ease of Use* caused the fact that it is expected that the use of the ADL will mainly result in less discrepancy between stakeholders, due to the traceability of the consistency between the models, and the structured detailed guidelines that can be followed to create and specify the models. Lastly, the positive *Behavioral Intention to Use*, caused by a positive outcome of the other variables, has resulted in the fact that, in general, both architects are willing to try to make use of the ADL in practice to benefit from its added value.

8. Conclusion & Discussion

To conclude the research that is elaborated in this master thesis, in this chapter, we are going to highlight and discuss the main findings. First, we provide the answers to the main research question by means of briefly discussing the answers to each sub research question. Then, the limitations, and validity and reliability threats are discussed which serve as possible research directions for future work. Hence, when we look again at the Information Systems Research Framework from Hevner, March, Park, and Ram (2004), at the one hand, this chapter elaborates on the added value of this master thesis to the practice / environment, and at the other hand, the contributions to the knowledge base / the literature.

8.1 Answers to research questions

In this research, we have investigated BPMSs. Nowadays, most BPMSs are in fact low-code development platforms that can be used for the development of process-driven applications. Namely, business process models are configured on the BPMS for executing business processes. We have focused on the communication flows within the application landscape where a BPMS is implemented. With communication flows we mean information/data flows, message flows regarding the communication and integration between the BPMS and other systems. When the business processes become quite complex, it means that many communication flows can be derived. Especially, due to the fact that, in such situations, usually, a lot of information is collected and used from the integrated systems from multiple business functions. Moreover, it can be difficult to specify the communication flows within the scope of multiple architecture domains, and to ensure the corresponding traceability of the consistency between the different architecture models. To target this problem from the process-oriented perspective of BPMSs, an Architecture Description Language (ADL) can be used. However, it was unclear what the constituents are of a process-oriented ADL that can be used to specify communication flows in BPMS application landscapes. Thus, we have tried to answer the following main research question (MRQ): "*What are the constituents of a process-oriented ADL for specifying communication flows in BPMS application landscapes?"*

Hence, the main objective of the research was to design a process-oriented ADL that supports application development on BPMSs regarding the specification of communication flows within the corresponding application landscape. The exact constituents of the ADL are determined by the answers to the sub research questions. These answers are discussed below.

First, we conducted both a literature review and a desk research in order to provide the answers to the first sub research question (SRQ1): *"What is the role of a BPMS within an application landscape?"* The literature review showed us that a BPMS executes business processes by means of executable business process models. Within an application landscape, a BPMS is the orchestrator of the communication between the BPMS and the integrated systems for the collection and use of the required information from those systems. For this communication, different standards (interfaces, integration services etc.) can be used. Based on these characteristics, a BPMS can be considered as a software-intensive system-of-systems. Guessi et al. (2015) already indicated that there is no ADL available for describing such systems. Furthermore, we looked at the general architecture of a BPMS in order to understand the functionalities of a BPMS. To put this all in a practical context, we have also conducted a desk research on the Pega Platform, which is the BPMS from Pegasystems that is used by the case study organization (BPM Company) for low-code application development. By means of this desk research, we have gathered insights into the practical use of BPMSs. Moreover, we have shown that the architecture of the Pega Platform adheres to the general BPMS architecture (Dumas, La Rosa, Mendling, & Reijers, 2018), as well as the Workflow Reference Model (Hollingsworth, 1995).

The second sub research question (SRQ2) was: "*What needs to be considered when designing a process-oriented ADL for specifying communication flows in BPMS application landscapes?*" By means of answering this question, we wanted to understand the meaning, main building blocks / requirements, and purposes of an ADL. The literature review showed us that, in the past decades, many ADLs have been designed for both general purposes and domain-specific. We tended to fill the gap regarding the lack of research on an ADL that is aimed at describing BPMSs. Especially, in terms of the communication with the integrated systems through APIs and (web) services, as well as inter-

organization communication via a BPMS (Pourmirza et al., 2017). Regarding the process-oriented perspective of the ADL, we have found that it can still be difficult to specify the communication flows within a BPMS application landscape from the viewpoint of multiple architecture domains. Namely, a lot of existing ADLs lacked of a clear view of the traceability of the consistency between architecture models that each aim at a different architecture level of a BPMS. To look at this all from a practical perspective, we have studied the implementation approach of the Pega Platform in order to understand how the intended ADL could be involved in this approach. Eventually, we were able to formulate criteria and corresponding requirements to select and compare existing ADLs that could be used to the construct the intended ADL, the so-called candidate ADLs. This comparison analysis has resulted in three candidate ADLs: *BPMN*, *ArchiMate*, and *UML*.

After the comparison analysis was done, during the design process of the intended ADL, SRQ3 was answered: "*What are the characteristics of the ADL?"* By means of both semi-structured interviews and focus groups, we collected the practitioners' perspectives on the design of the ADL. In this way, we were able to select the relevant models of the candidate ADLs based on the opinions/insights from the practitioners, which were both business architects and system architects. This has resulted in (1) a high-level architecture decomposition model that specifies the scope of the intended ADL in terms of architecture levels and viewpoints, (2) a high-level ADL model structure that specifies the interrelation links between the architecture models that can be created, and (3) the guidelines for practitioners, which answered SRQ4: "*How can the ADL be applied by practitioners?"* To demonstrate this all in a practical context, a fictional running example has been elaborated.

To validate the correctness and practical applicability of the intended ADL, we have performed a case study in order to answer SRQ5: "*Is the designed ADL valid and applicable in practice for the desired purposes?"* The case study validation entailed that we first created the architecture models for a project of BPM Company. Then, the created models were discussed by means of conducting two semi-structured interviews / step-by-step walkthroughs, one with a business architect, and one system architect that were both involved in the case study project. We have used the variables of the Technology Acceptance Model (Davis, Bagozzi, & Warshaw, 1989) to structurally validate the ADL.

Overall, based on the case study validation, we can conclude that the intended ADL is perceived as a valid and useful means regarding the specification of communication flows within BPMS development projects. Especially, to correctly determine the scope of a BPMS application in terms of the supported business processes, required functionalities, and the way of collecting and using information from integrated systems. When the intended ADL was used during the initial phase of the case study project, lots of issues could be prevented afterwards. Mainly, discrepancies between the concerns of different stakeholders. This is mainly due to the traceability of the consistency between the different architecture models that can be created by means of the intended ADL. Namely, this consistency can give more insights into the links between the concerns of different stakeholders, both from the business and IT. This property of our ADL contributes to the lack of ADLs that provide a clear overall view on the interrelation between different architecture domains (Lankhorst, 2017; Vares et al., 2017; Rozanski & Woods, 2012). However, in general, it mainly depends on the project context and project size to what extent each architecture model is relevant to be created. In contrast to business architect, the system architect found it difficult to determine the added value / relevance of several models, including the choreography diagrams. Regarding the ease of use of the intended ADL, most models will be easy to create and understand with not much effort. Only several adjustments to the guidelines of the architecture models were needed in order to solve some small uncertainties regarding the purpose and meaning of several models. Most models are IT-oriented, and will therefore be understandable for most IT stakeholders. Moreover, we can conclude that the ease of use of the intended ADL is partly caused by the fact that the candidate ADLs (BPMN, ArchiMate, UML) are often applied within the industry / originated from the industry. These are not strict formal academic ADLs. This is aligned with one of the main findings of the survey that was run by Malavolta et al. (2013). Namely, their respondents prefer the use of industry wide applied / originated ADLs, due to the fact that, in constrast to the academic ADLs, the practical ADLs seem to fulfill more practitioners' needs. In addition, their survey results already concluded that UML is the most applied ADL in practice.

8.2 Limitations

Our research has several limitations. First of all, the scope of our ADL is limited to BPMSs, and thus solely focuses on the functionalities and capabilities of these systems. More precisely, the ADL is tailored to be used to create and specify architecture models of applications that are developed on a BPMS. Moreover, we have not explicitly looked at the four different types of BPMSs that were shown in Figure 8. Though, in principle, we assume that our ADL can be used to describe communication flows within any type of BPMS due to the common core capabilities. The second limitation is the fact that we have only shown that the ADL is perceived as a promising and useful means that will be easy to use within BPMS projects. This then resulted in a positive behavioral intention to use. These points have been indicated by means of the validation interviews that entailed that we only discussed the possible outcomes of applying the guidelines of the ADL by means of step-by-step walkthroughs. In other words, the ADL was not applied by the two architects that were interviewed. Hence, the actual use of the ADL in practice is not known at this moment. The third limitation deals with the traceability purposes of the intended ADL that are limited to manual textual mappings of model elements across the different architecture levels that can be considered for a BPMS. Thus, it can be quite time consuming to apply the traceability capabilities of the ADL. An advanced tool can support the architectural purposes of the ADL. However, developing such a tool was not part of the scope of our research. Lastly, we have only used qualitative research methods. Though, based on the scope and objectives of the research, we think it is difficult to effectively apply a quantitative research method, such as a survey.

8.3 Validity and reliability threats

Next to the limitations, there are also several threats regarding the validity and reliability of the results of our research. These are described below.

Internal validity

The internal validity deals with the validity of the approach/structure that was followed to conduct this research. This research was conducted within just one organization: BPM Company. More precisely, for the practical perspective, only architects from one organization were involved in the research. Regardless of the fact that these architects were working at different customers of BPM Company, and thus different organizations, they all used the same BPMS: the Pega Platform, which was therefore the only BPMS that has been studied in practice. Moreover, the validation of the ADL was limited to a case study, including two semi-structured interviews. Thus, the case study validation was limited to only one project, and of that project, only two architects have been interviewed.

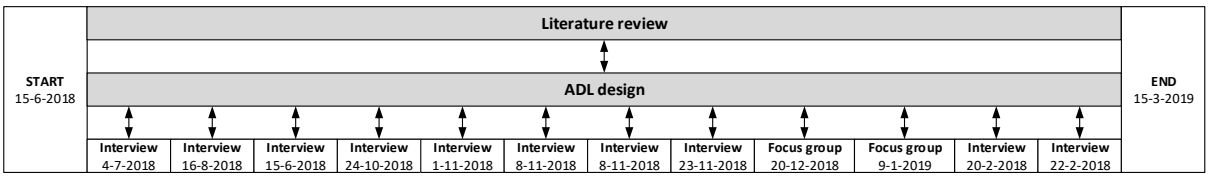


Figure 57: Intended ADL design and validation timeline

Despite these threats, as shown in Figure 57, the ADL was in fact validated multiple times during the design process. Namely, next to the case study validation, during the design process of the ADL, at different moments, we conducted several semi-structured interviews to collect relevant input from practitioners. This input also consisted of possible improvements to the version of the ADL that was designed so far. These potential inputs were then processed before the next interview was conducted in order to get new potential improvements. After these interviews were conducted, we also held two focus groups. These focus groups also resulted in potential adjustments to the ADL. Moreover, a fictional running example was made up to apply the ADL. Furthermore, during the design process of the ADL, we frequently determined the contribution of the ADL to the literature. In this way, the ADL was also validated regarding its scientific value.

External validity

The external validity is about the extent to which it is possible to generalize the results of our research. It is not completely known/clear how the ADL will be used within the context of other BPMS projects where other BPMSs are used. In other words, we can state that it is difficult to generalize the results of

this research. Though, for the semi-structured interviews during the design of the ADL, architects from different projects have been interviewed. During these interviews, it was already somehow validated what kind of models are relevant to be used in different situations. Hence, it this way, we have partly limited the issues regarding the external validity of this research.

Construct validity
As mentioned before, for the case study validation, we created the architecture models by means of applying the guidelines of the ADL. We then used these models to validate the ADL by means of semi-structured interviews that were in fact step-by-step walkthroughs of creating and describing the architecture models. Hence, the practitioners (= architects) did not follow the guidelines by themselves. Instead, to partly reduce this threat, we showed them and discussed with them the possible outcomes of following the guidelines through the created architecture models within the context of the case study project. In this way, at least, we have only checked whether practitioners correctly understood the usefulness and ease of use of our ADL.

Reliability
The reliability aims at the extent to which the same research results will be obtained when future research is applied in the same way. Almost the entire design process of the ADL was a creative process. Therefore, it will be hard to obtain the exact same results when the same research is done again. Though, the design choices and specifications, including the selection criteria and corresponding requirements of the ADL have been strictly/explicitly documented in this thesis.

8.4 Future work
Below, based on the conclusions of the main results of this research, the limitations, and the validity and reliability threats, we propose several research directions for future research.

- *More case studies with various situational factors*. In future research, the validity/correctness and practical applicability of the intended ADL needs to be validated in different organizational / project contexts and project sizes by means of case studies. This then needs to be done in the same way. Thus, by means of semi-structured interviews / step-by-step walkthroughs on the architecture models that have been created in advance. Moreover, it will be interesting to look at how the ADL will be applied when other BPMSs are used. In this way, threats regarding both the external validity and internal validity of this research can be solved.
- *Actual use of the ADL in practice.* In order to get more insights into the practical applicability of the intended ADL, future research needs to focus on practitioners that follow the guidelines by themselves. For example, by conducting a controlled experiment with a group of relevant practitioners that is divided into two groups both working on the same case. One group will then make use of the ADL, while another group is only provided with examples of possible architecture models that can be created when following the guidelines of the ADL. It can then be better determined what the benefits and drawbacks are of the actual application of our ADL.
- *Tool support.* It is required to create an "all-in-one tool" that supports the creation of all architecture models of the ADL. For the traceability purposes, automated mappings and drill-down possibilities, such a tool would foster both the effectiveness and efficiency of applying the ADL in practice. Moreover, automated syntax checks can ensure the syntactical correctness of the created architecture models. Furthermore, this potential tool will foster a correct maintenance and refinement of the architecture models.
- *Other software systems*. Despite the fact that the ADL is aimed at BPMSs, it can be interesting to specify communication flows regarding other types of (process-aware) systems, such as ERP systems. It will then be interesting to determine to what extent other systems can be described in terms of communication flows. Moreover, it can then be clarified to what extent our ADL is indeed limited to BPMSs.
- *Quantitative research*. Future research can try to determine the usability and added value of applying a quantitative research method within the scope of our research. In this way, other results on the practical applicability of our ADL might be obtained.

When this future work is done, our research results will be improved. Moreover, the consequences of the aforementioned limitations, and validity and reliability threats will be minimized.

9. References

Bass, L., Clements, P., & Kazman, R. (2003). *Software architecture in practice.* Boston, United States: Addison-Wesley Professional.

Behjati, R., Yue, T., Nejati, S., Briand, L., & Selic, B. (2011). Extending SysML with AADL concepts for comprehensive system architecture modeling. *European Conference on Modelling Foundations and Applications* (pp. 236-252). Berlin, Heidelberg: Springer.

Brambilla, M., Cabot, J., & Wimmer, M. (2017). Model-driven software engineering in practice. *Synthesis Lectures on Software Engineering*, 3(1), 1-207.

Buckl, S. A., Matthes, F., & Schweda, C. M. (2009). An information model capturing the managed evolution of application landscapes. *Advances in Enterprise Engineering III* (pp. 85-99). Berlin, Heidelberg: Springer.

Butting, A., Haber, A., Hermerschmidt, L., Kautz, O., Rumpe, B., & Wortmann, A. (2017). Systematic Language Extension Mechanisms for the MontiArc Architecture Description Language. *In European Conference on Modelling Foundations and Applications* (pp. 53-70). Springer, Cham.

Chen, R., Liu, Y., Cao, Y., Zhao, J., Yuan, L., & Fan, H. (2018). ArchME: A Systems Modeling Language extension for mechatronic system architecture modeling. *AI EDAM, 32(1),* 75-91.

Cleland-Huang, J., Hanmer, R. S., Supakkul, S., & Mirakhorli, M. (2013). The twin peaks of requirements and architecture. *IEEE Software*, 30(2), 24-29.

Clements, P. C. (1996). A survey of architecture description languages. *In Proceedings of the 8th international workshop on software specification and design* (p. 16). Pittsburgh: IEEE Computer Society.

Davis, F. D., Bagozzi, R. P., & Warshaw, P. R. (1989). User acceptance of computer technology: a comparison of two theoretical models. *Management science*, 35(8), 982-1003.

Deneckère, R., Hug, C., Onderstal, J., & Brinkkemper, S. (2015). Method Association Approach: Situational construction and evaluation of an implementation method for software products. *In Research Challenges in Information Science (RCIS), 2015 IEEE 9th International Conference on* (pp. 274-285). IEEE.

Dumas, M., La Rosa, M., Mendling, J., & Reijers, H. A. (2018). *Fundamentals of business process management (2nd ed.).* Heidelberg: Springer.

Faulkner, S., & Kolp, M. (2003). Towards an Agent Architectural Description Language for Information Systems. *ICEIS (3),* 59-66.

Feiler, P. H., Gluch, D. P., & Hudak, J. J. (2006). *The Architecture Analysis & Design Language (AADL): An Introduction.* Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst.: Pittsburgh.

Geambaşu, C. V. (2012). BPMN vs UML activity diagram for business process modeling. *Accounting and Management Information Systems, 11(4),* 637-651.

Guessi, M., Cavalcante, E., & Oliveira, L. B. (2015). Characterizing architecture description languages for software-intensive systems-of-systems. *n Proceedings of the third international workshop on software engineering for systems-of-systems* (pp. 12-18). n.p.: IEEE Press.

Hevner, A. (2007). A Three Cycle View of Design Science Research. *Scandinavian Journal of Information Systems*, 19(2), 87-92.

Hevner, A., March, S. T., Park, J., & Ram, S. (2004). Design Science in Information Systems Research. *Design Science in IS Research MIS Quarterly*, 28(1), 75-105.

Hollingsworth, D. (1995). *The Workflow Reference Model.* TC00-1003 Issue 1.1. Workflow Management Coalition.

ISO/IEC/IEEE. (2011). *Systems and software engineering – Architecture description, ISO/IEC/IEEE 42010:2011*. Geneva: International Organization for Standardization.

Ko, R. K., Lee, S. S., & Lee, E. W. (2009). Business process management (BPM) standards: a survey. *Business Process Management Journal*, 15(5), 744-791.

Langer, A. M. (2016). *Guide to Software Development.* London: Springer.

Lankhorst, M. (2017). *Enterprise Architecture at Work (4rd ed.).* Enschede: Springer-Verlag.

Leite, J., Oquendo, F., & Batista, T. (2013). SysADL: a SysML profile for software architecture description. *In European Conference on Software Architecture* (pp. 106-113). Heidelberg: Springer.

Levina, O., Holschke, O., & Rake-Revelant, J. (2010). Extracting business logic from business process models. *Information Management and Engineering (ICIME)* (pp. 289-293). n.p.: IEEE.

Luinenburg, L., Jansen, S., Souer, J., Van De Weerd, I., & Brinkkemper, S. (2008). Designing Web Content Management Systems Using the Method Association Approach. *In Proceedings of the 4th International Workshop on Model-Driven Web Engineering (MDWE 2008)*, 106-120.

Malatova, I., Lago, P., Muccini, H., Pelliccione, P., & Tang, A. (2018). *Architectural Languages Today*. Retrieved from Univaq: http://www.di.univaq.it/malavolta/al/

Malavolta, I., Lago, P., Muccini, H., Pelliccione, P., & Tang, A. (2013). What Industry needs from Architectural Languages: A Survey. *IEEE Transactions on Software Engineering, 39(6),* (pp. 869-891).

Medvidovic, N., & Taylor, R. N. (1997). A Framework for Classifying and Comparing Architecture Description Languages. *Jazayeri M., Schauer H. (eds) Software Engineering - ESEC/FSE'97. ESEC 1997, SIGSOFT FSE 1997* (pp. 60-76). Berlin, Heidelberg: Springer.

Mendling, J., & Hafner, M. (2008). From WS-CDL choreography to BPEL process orchestration. *Journal of Enterprise Information Management. 21(5)*, pp. 525-542.

Menge, F. (2007). Enterprise Service Bus. *In Free and open source software conference 2007*, (pp. 1-6 (2)). Sankt Augustin, Germany.

Moody, D. L. (2003). The method evaluation model: a theoretical model for validating information systems design methods. *ECIS 2003 proceedings*, 79.

Muller, J., & Bohm, K. (2011). The architecture of a secure business-process-management system in service-oriented environments. *In Web Services (ECOWS), 2011 Ninth IEEE European Conference on Web Services* (pp. 49-56). Lugano, Switzerland: IEEE.

Object Management Group. (2012). *Service oriented architecture Modeling Language (SoaML) Specification version 1.0.1.* Retrieved from Object Management Group: http://www.omg.org/spec/SoaML/1.0.1

Object Management Group. (2013). *Business Process Model and Notation (BPMN) 2.0.2 Specification.* Retrieved from Object Management Group: http://www.omg.org/spec/BPMN

Object Management Group. (2016a). *Decision Model and Notation (DMN) V1.1.* Retrieved from Object Management Group: https://www.omg.org/spec/DMN/1.1

Object Management Group. (2016b). *Case Management Model and Notation (CMMN) version 1.1.* Retrieved from Object Management Group: http://www.omg.org/spec/CMMN/1.1

Object Management Group. (2017a). *UML 2.5.1 Specification.* Retrieved from Object Management Group: https://www.omg.org/spec/UML/2.5.1

Object Management Group. (2017b). *Systems Modeling Language Version 1.5.* Retrieved from Object Management Group: http://www.omg.org/spec/SysML/1.5/

Oquendo, F., Leite, J., & Batista, T. (2016). Software Architecture in Action: Designing and Executing Architectural Models with SysADL Grounded on the OMG SysML Standard. *Undergraduating Topics in Computer Science.* Springer.

Pega Academy. (2018). *The Role of the Pega BA.* Retrieved from Pega Academy: http://pegasystems2.http.internapcdn.net/pegasystems2/student_guides/Role_of_the_Pega_BA_Student_Guide.pdf

Pegasystems. (2018a). *About Pegasystems*. Retrieved from Pegasystems: https://www.pega.com/about

Pegasystems. (2018b). *Build for Change: Situational Layer Cake.* Retrieved from Pegasystems: https://www1.pega.com/insights/resources/build-change-situational-layer-cake

Penicina, L. (2013). Linking BPMN, ArchiMate, and BWW: Perfect match for complete and lawful business process models? *In PoEM (Short Papers)*, 156-165.

Pourmirza, S., Peters, S., Dijkman, R., & Grefen, P. (2017). A systematic literature review on the architecture of business process management systems. *Elsevier*, 66, 43-58.

Ravesteyn, P., & Versendaal, J. (2007). Success Factors of Business Process Management Systems Implementation. *Proceedings of the 18th Australasian Conference on Information Systems (ACIS 2007).* Toowoomba.

Ravesteyn, P., & Versendaal, J. (2009). Constructing a situation sensitive methodology for business process management systems implementation. *PACIS 2009 Proceedings.* Paper 70.

Recker, J., & Mendling, J. (2015). The State of the Art of Business Process Management Research as Published in the BPM Conference. *Business & Information Systems Engineering, 58(1)*, 55-72.

Rozanski, N., & Woods, E. (2012). *Software Systems Architecture - Working With Stakeholders, Using Viewpoints and Perspectives (2nd ed.).* Courier: Addison-Wesley.

Sheng, Q. Z., Qiao, X., Vasilakos, A. V., Szabo, C., Bourne, S., & Xu, X. (2014). Web services composition: A decade's overview. *Information Sciences, 280*, 218-238.

Smirnov, S., Reijers, H. A., Weske, M., & Nugteren, T. (2012). Business process model abstraction: a definition, catalog, and survey. *Distributed and Parallel Databases, 30(1)*, 63-99.

The Open Group. (2017). *ArchiMate 3.0.1 Specification, an Open Group Standard* . Retrieved from The Open Group: http://pubs.opengroup.org/architecture/archimate3-doc/

van der Aalst, W. (2013). Business Process Management: A Comprehensive Survey. *ISRN Software Engineering*, 1-37.

Vares, F., Amiri, M. J., & Parsa, S. (2017). Towards a model-driven development of enterprise systems. *Computer Science and Software Engineering Conference (CSSE)* (pp. 42-48). Shiraz, Iran: IEEE.

Venkatesh, V., & Bala, H. (2008). Technology acceptance model 3 and a research agenda on interventions. *Decision sciences*, 39(2), 273-315.

Venkatesh, V., & Davis, F. D. (2000). A theoretical extension of the technology acceptance model: Four longitudinal field studies. *Management science*, 46(2), 186-204.

Venkatesh, V., Morris, M. G., Davis, G. B., & Davis, F. D. (2003). User acceptance of information technology: Toward a unified view. *MIS quarterly*, 425-478.

Visual Paradigm. (2018a). *What is Case Management Model and Notation (CMMN).* Retrieved from Visual Paradigm: https://www.visual-paradigm.com/guide/cmmn/what-is-cmmn/

Visual Paradigm. (2018b). *Full ArchiMate Viewpoints Guide.* Retrieved from Visual Paradigm: https://www.visual-paradigm.com/guide/archimate/full-archimate-viewpoints-guide/

Weske, M. (2012). *Business Process Management.* Heidelberg: Springer.

Wohlin, C. (2014). Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering. *18th International Conference On Evaluation And Assessment In Software Engineering* (p. 38). London: ACM.

Zúñiga-Prieto, M., Insfran, E., & Abrahão, S. (2016). Architecture description language for incremental integration of cloud services architectures. *2016 IEEE 10th International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Environments* (pp. 16-23). n.p.: IEEE.

Appendix A – Pega Platform architecture

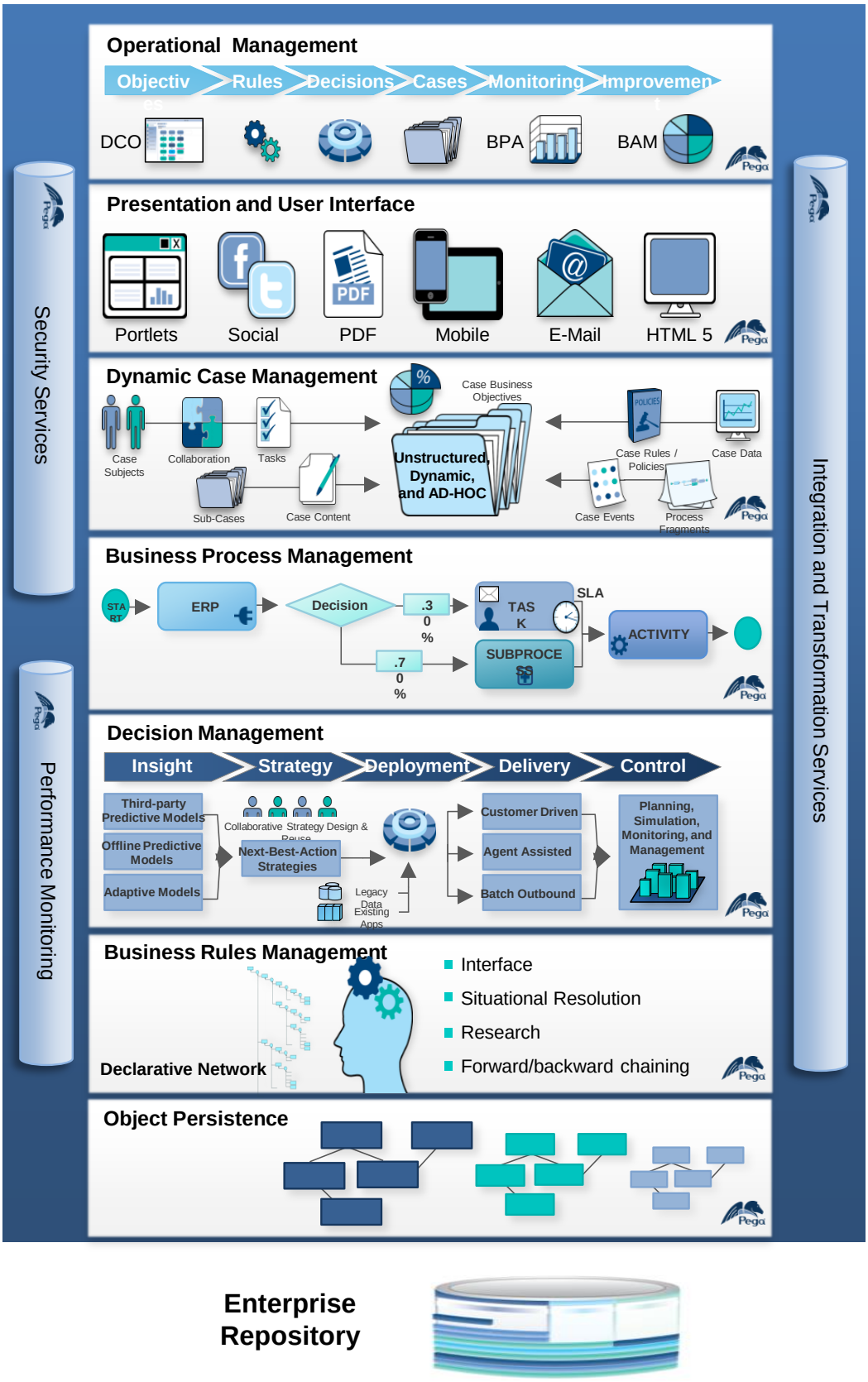


Figure 58: Pega Platform - functional architecture

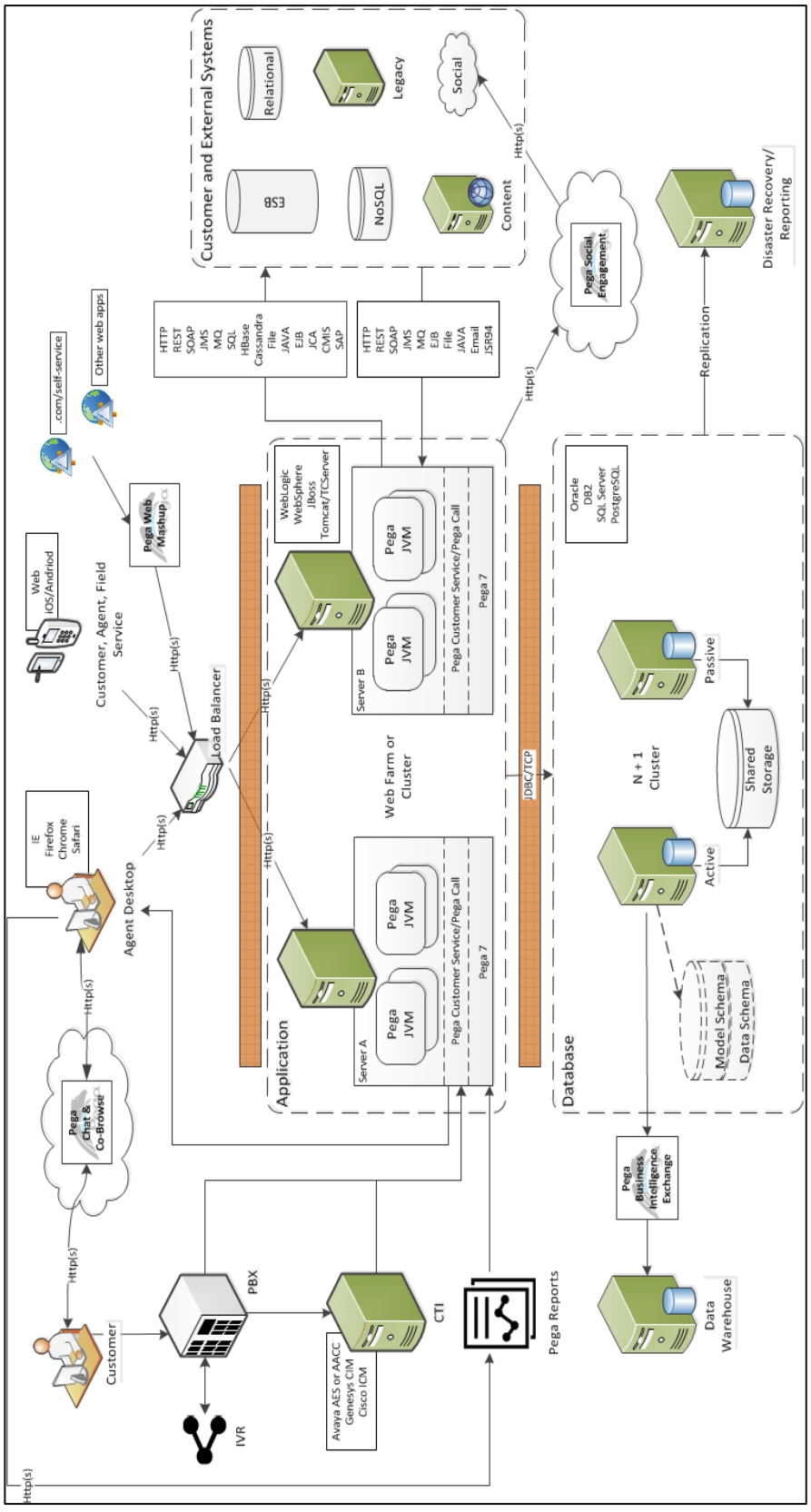


Figure 59: Pega Platform - technical architecture

Appendix B – Interview protocol ADL requirements & practice

Introduction

- Personal introduction of student and interviewee
- Explanation of the objective and structure/duration of the interview
  - ADL: *"any form of expression for use in architecture descriptions"*
  - Intended ADL: communication flows in BPMS application landscapes (EA vs. SA)
  - [Show MDA and SOA models]
- [Ask for permission to record the interview / start taking notes]

Context

1) In short, what is the objective and progress of your current running project?
2) What is a typical working day / what are your main tasks and responsibilities?
3) How and at what moments do you use architecture descriptions during the project?
4) What language(s) do you use for describing an architecture? Can you give an (fictional) example of how you create an architecture model/description?
5) What challenges do you experience when determining and specifying the architecture and related aspects of a Pega application, if any?

Requirements

6) To what extent does an ADL need to have a degree of "freedom" when creating and specifying architecture models? Do you want to be constrained by strict guidelines that you must follow? Why (not)?
7) What architectural purposes, e.g. analysis, validation, refinement, do you (want to be able to) perform with an architecture description?
   - *Analysis: consistency, deadlocks, requirements, change impact, cost/value etc.*
   - *Validation: compliancy intended - implemented architecture, traceability etc.*
   - *Refinement: incremental/iterative application development and description*
   - *Design decisions capturing*
8) Which aspects (viewpoints and static/dynamic structure) of a Pega application are most relevant for you, and how would/do you model and specify these aspects? [Show example viewpoint models]
   - *Business process related / software architecture related*
   - *Communication flows (information/data flows, message flows)*
   - *Web services and interfaces/APIs*
   - *Choreographies and orchestrations*
9) When you create an architecture model/description, what details (abstraction/granularity) do you include, and what details do you exclude/ignore? Why?
10) What is not being modelled and described now, but needs to be done/fostered in the future?
11) When you create an architecture model/description, do you use a certain architecture style, e.g. component-based, and layered style? Why (not)?
12) If an ADL can be used within an advanced tool, what functionalities does such a tool need to support?
13) Do you have any other requirements?

Intended ADL

- Explanation of comparison analysis table → [show example ADL models]
- Intended ADL → [show current version]
- Short elaboration of the fictional example case by means of the intended ADL

Practice

14) What are your expectations (trade-offs) when the intended ADL is applied in practice?
15) How would you use the intended ADL in practice?

Wrap-up

- Summary of most important given answers

16) Do you have any further questions and/or remarks?
17) Can I mention your name in my thesis report, or do you want to stay anonymous?
18) Can I contact you for further questions / validating the current version of the ADL?

- Further contact and remaining steps of this research

**

Appendix C – Running example: car insurance company case

Introduction

Consider a fictional car insurance company that covers car damages. This includes small car damages, like dents, that do not request immediate action because the car can still be used safely. To request a desired compensation to repair car damage, an insurant creates a new payment claim. For this, the car insurance company has an application that runs on a BPM platform, in short, a BPMS, that is integrated with several other systems within the company. All systems run on an application server, which is connected to a database server. Notice that all mentioned activities are carried out within the BPMS.

1) Submit

To create and submit a new payment claim, an insurant needs to fill out a digital damage form in order to provide the required information on the car damage. At least the insurance information of the insurant, and a photo of the car damage + description of the cause of the car damage need to be collected and included. In addition, it might be necessary to add the insurance information of the insurant that was involved in the cause of the car damage. After the payment claim has been created completely, it is submitted to the car insurance company.

2) Register

When a new payment claim has been received, a claim handler (= an employee) receives a notification that the payment claim needs to be fully registered. For this, an integrated CRM system is assessed to get all personal/situational information on the corresponding insurant. Eventually, a payment claim specification has been created. A confirmation of a fully registered payment claim is automatically sent to the insurant.

3) Check

After the full registration, the payment claim is checked. This entails that it is determined whether or not the payment claim adheres to certain predefined requirements for being a legit payment claim. These requirements are collected from a separated DMS. In this way, false/incorrect payment claims can be indicated. The check at least automatically indicates if the damage form has been filled out correctly/completely. In addition, the insurance policy of the insurant is reviewed. This is done in order to determine what part of the costs for repairing the car damage can be compensated/covered. Another part of checking the payment claim is the estimation of the expected financial compensation. This is done by a financial controller. For this, financial details are collected from a financial system. These details are added to the payment claim specification.

4) Approval

In most cases, a checked payment claims results in an approved payment claim, which is communicated to the insurant. The claim handler then adds the contact information on a suitable garage that can repair the car damage. Then, the approved payment claim specification is sent to the insurant.

A. Reject

If a payment claim is rejected, then an email is sent to the corresponding insurant with the reason why the claim has been rejected. A possible reason can be the fact that the caused car damage is not covered within the corresponding insurance policy, or that certain information was missed out. Eventually, the insurant needs to submit a new payment claim from the beginning. The rejection is registered in the CRM system.

5) Calculate

After the car damage has been repaired at a garage, which takes X days, the insurant receives an invoice from the garage. This invoice is added to the payment claim specification by the insurant. Then, a financial controller calculates the actual compensation by means of the financial details from the financial system. It might be the fact that the invoice is not completely compensated, for example, because the actual costs are higher than the expected costs. The calculated compensation is communicated to the claim handler.

6) Arrange

When the claim handler has received the actual calculated compensation, the claim handler arranges the desired for the insurant in order to complete the arrangement of the payment claim. The arranged payment claim specification that is sent to the insurant is registered in the separated financial system. The financial system is integrated with an external banking system of the customer in order to automatically send the money to the account number of the insurant via his/her bank. The financial system also exchanges the details on the arranged payment claim to the CRM system. After all this, the claim handling procedure has been completed.

A. Reject

It is also possible that the costs are not compensated after all. For example, in case the costs are lower than expected, and thus need to be paid completely by the insurant, or when another type of reparation has been done which is not covered by the corresponding policy of the insurant. Hence, despite the fact that, initially, the payment claim was approved, it eventually is completely rejected after all. An email of this is sent to the insurant. The rejection is also registered in the CRM system.

**

Appendix D – ADL document template: running example models

This appendix contains a template that can be used to create a document to specify the ADL architecture models. Chapter 6 refers to this appendix for the architecture models that have been created for a fictional running example. Hence, in this appendix, it is presented how an architecture document created by means of the ADL might look like.

ARCHITECTURE DECOMPOSITION SPECIFICATION DOCUMENT

A specification of communication flows

Document title:	Architecture decomposition specification
Project name:	Car insurance company (fictional running example)
Author(s):	Jeremy Loppies
Date:	15-3-2019
Version:	0.1

1. Introduction

This section describes the context and purpose of this document, as well as a glossary of the terminology that is used.

1.1 Purpose of this document

This document elaborates on the architecture of the BPMS application that was developed. Different architecture layers and the corresponding viewpoints are decomposed, specified, and interrelated with each other. For this, architecture models are shown and specified that have been created by means of a new process-oriented architecture description language (ADL). This ADL has been designed to fit the specification of the purposes of BPMSs, especially the applications that are developed on them.

The scope of the ADL is the specification of communication flows (information/data flows, and message flows) within the application landscape in which the BPMS is implemented. More precisely, the ADL can be used to specify how information/data is collected and used by the BPMS when executing the business processes. This is done by means of different viewpoints, including the order in which the BPMS and the integrated system(s) are used to correctly collect and use the information/data through APIs and (web) services for the execution of the business processes.

1.2 Project context

The BPMS application development project was conducted for a fictional car insurance company that covers car damages. This includes small car damages, like dents, that do not request immediate action because the car can still be used safely. The core business of this company are the payment claim that need to be processed. To request a desired compensation to repair car damage, an insurant creates a new payment claim. Within the old situation, multiple systems were used for this. In the new situation, a BPMS has been implemented to optimize the business processes, and orchestrate the functionalities of the separated systems.

1.3 Glossary

Table 1: Glossary

Term / abbreviation	**Definition**
ADL	Architecture Description Language
BPMS	Business Process Management System
Term x	Definition x

2. Business domain level

This section elaborates on the business domain in order to clarify the context in which the project was situated.

2.1 Business functions

This paragraph describes the business functions that are related to the project context.

2.1.1 ArchiMate organization structure viewpoint

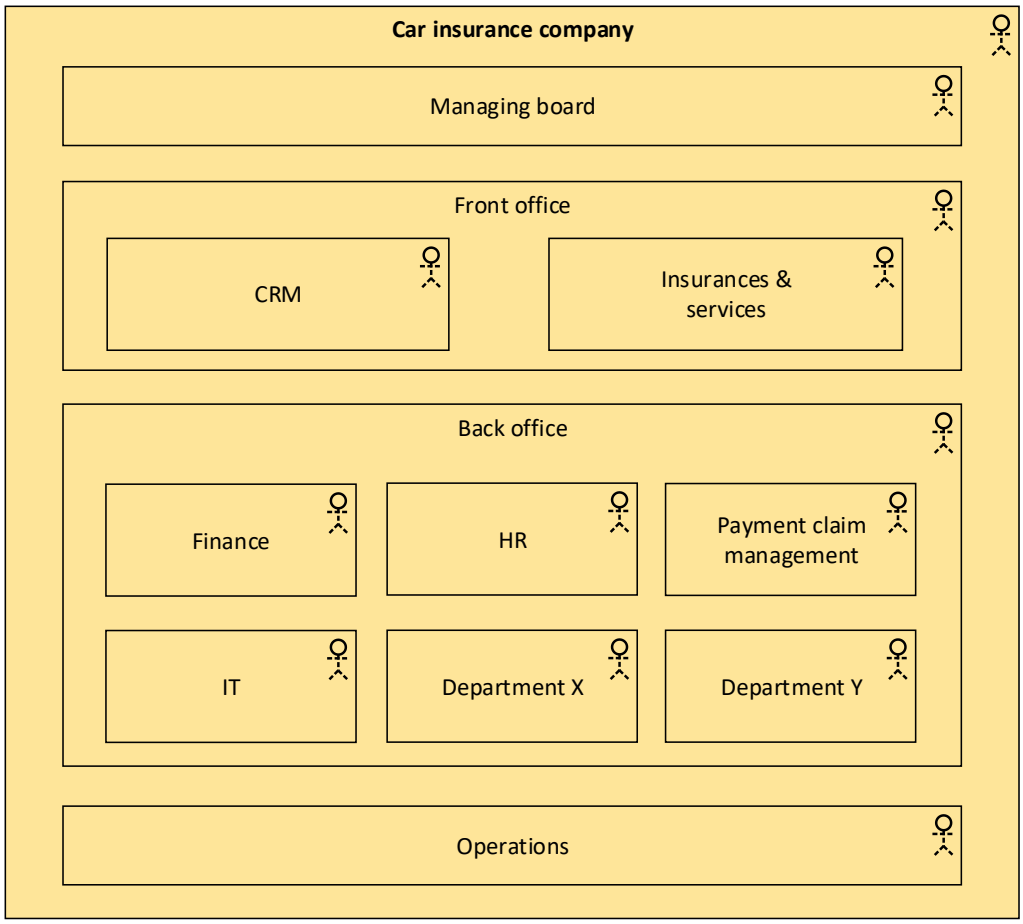


Figure 1: ArchiMate organization structure viewpoint

This viewpoint visualizes the different parts (departments) of the car insurance company as *business actors*. The car insurance company has been modelled as a large business actor. It is divided into the front office, back office, the managing board, and operations. Both the front office and back office consists of multiple departments / business units, including Finance, and HR. The ArchiMate organization structure viewpoint does not depict a certain hierarchy regarding the organization structure. Though, a certain organization hierarchy has been considered while creating the model. Therefore, the managing board has been positioned at the top. Then, below of the managing board, the remaining departments are shown.

2.1.2 ArchiMate business function viewpoint

Based on the ArchiMate organization structure viewpoint, the ArchiMate business function viewpoint has been created.

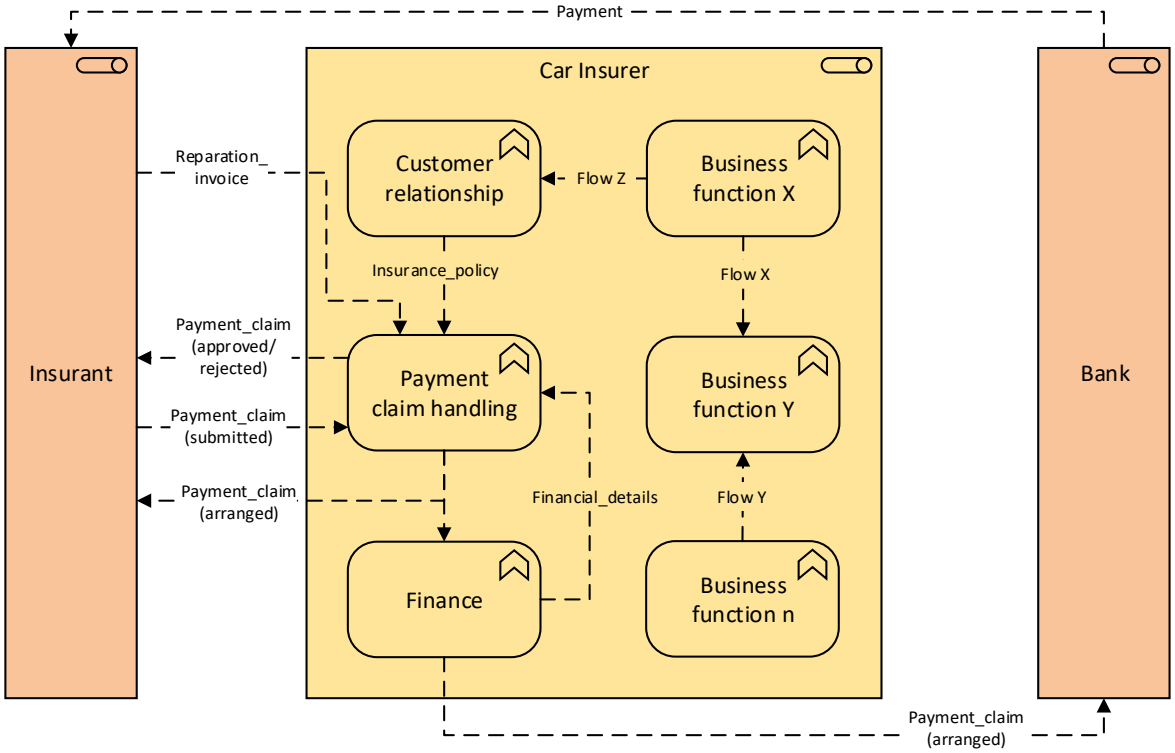


Figure 2: ArchiMate business function viewpoint

This viewpoint is a high-level view of the communication flows (information/data/message exchanges) between the business functions of the car insurance company. The viewpoint solely visualizes the business functions that are applicable to the project.

The car insurance company has been modelled as a business role. Several business functions are shown, including Payment claim handling, and Finance, that are involved in the process of handling a payment claim. Namely, the Finance business function is involved due to the fact that it provides the required financial details for the payment purposes. Both the insurant and the bank have been modelled as an external business role. There are several information flows from an external business role to the company. For example, an insurance payment claim that is exchanged from an Insurant to the Payment Claim Handling business function of the company. Due to the scope of the project, the *Payment claim handling* business function has the most ingoing and outgoing communication flows.

3. Process/application decomposition level

This section focuses on the decomposition of the business processes and the underlying application components that are used. This is done by means of different viewpoints.

3.1 Business processes

In this paragraph, the business processes that are optimized by the BPMS are decomposed and specified.

3.1.1 ArchiMate business process viewpoint

To give a high-level view of the business processes, the ArchiMate business process viewpoint is used.

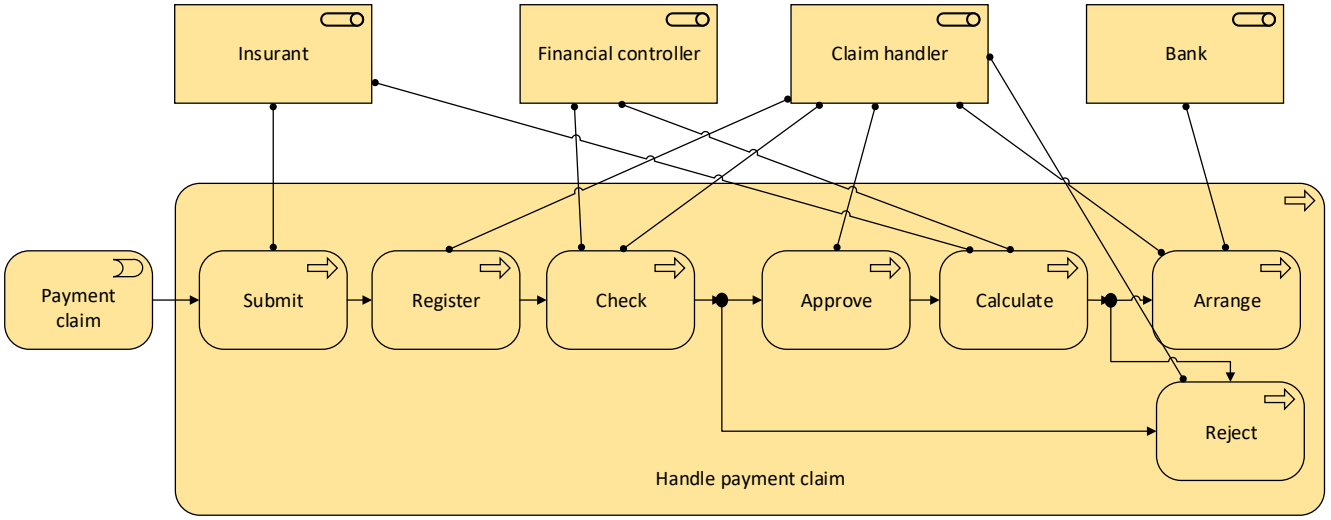


Figure 3: ArchiMate business process viewpoint

The overall business process is called *Handle payment claim.* The business event is a new payment claim that needs to be handled. The business process is divided into multiple smaller sub business processes that are carried out in a certain order. These processes are in fact different stages / phases. There are two decision points that have been modelled by means of a junction. By means of the assignment relation, each business role is assigned to one or more business processes.

3.1.2 BPMN process diagram [high-level overview]

The ArchiMate business process viewpoint is elaborated in more details by means of the BPMN process diagram.

First, below, the overall business process *Handle payment claim* is shown from a high-level point of view, by means of visualizing the stages as sub processes.

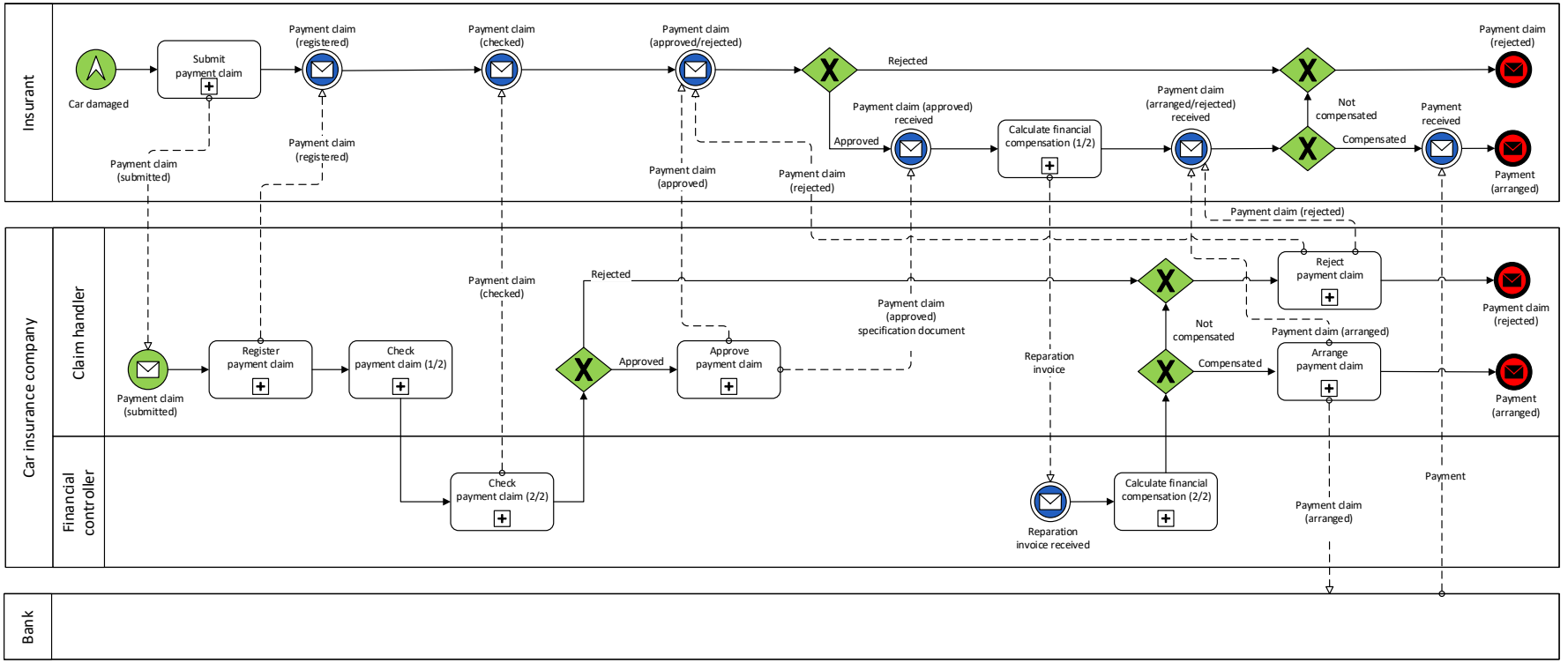


Figure 4: BPMN process diagram [high-level overview]

Three separated pools have been created: the car insurance company, insurant, and bank. The pool of the bank is considered as a black box. The car insurance company has two lanes that represent the process participants/actors within the company: claim handler, and financial controller. The lanes visualize the executor of each task. Two abstraction levels has been created. One large diagram visualizes multiple sub processes that are carried out in a certain order. Each sub process is further elaborated within a separated sub process diagram. This then contains the actual task are executed. Most tasks are carried out by means of a user action within the application. Several tasks are carried out automatically. Within the process, there are four exclusive gateways / decision points regarding the approval or rejection of a new payment claim.

Furthermore, there are several intermediate message events that triggered by a message flow. For example, the reparation invoice that in sent by the insurant to the claim handler. For two sub processes, multiple persons are involved, including *Check payment claim* which is started by the Claim handler (1/2), and then it is carried out by the Financial controller (2/2). Between the sub processes, many message flows are shown. These message flows entail certain external information/data exchanges between the Insurant and a person from the car insurance company. In addition, the sequence flows between different lanes within the car insurance company pool, for example, the arrow from *Check payment claim* (1/2) to *Check payment claim* (2/2) are so-called internal message exchange that can be elaborated in more details at a lower abstraction level.

3.1.3 BPMN process diagram [sub processes]

Each sub process can be elaborated in a separated diagram. These diagrams are shown below.

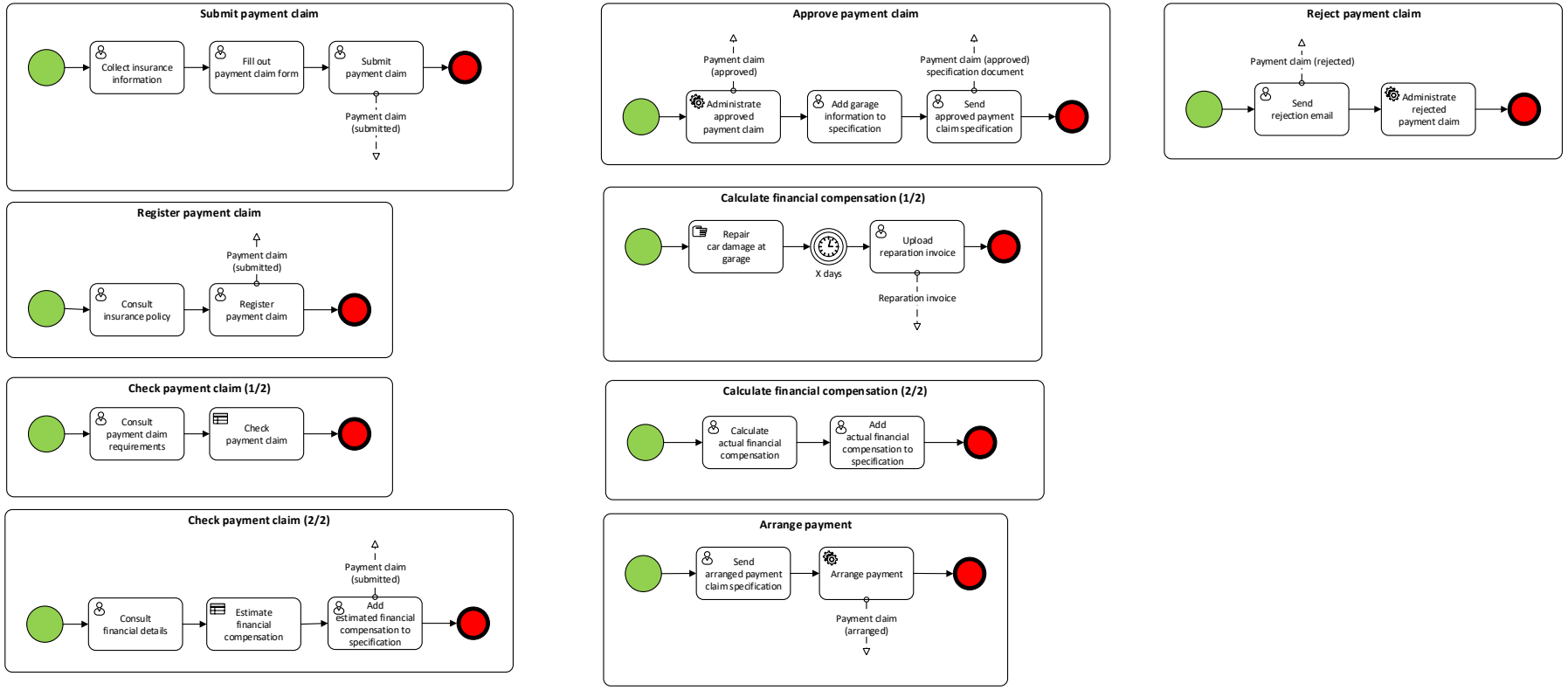


Figure 5: BPMN process diagram [sub processes]

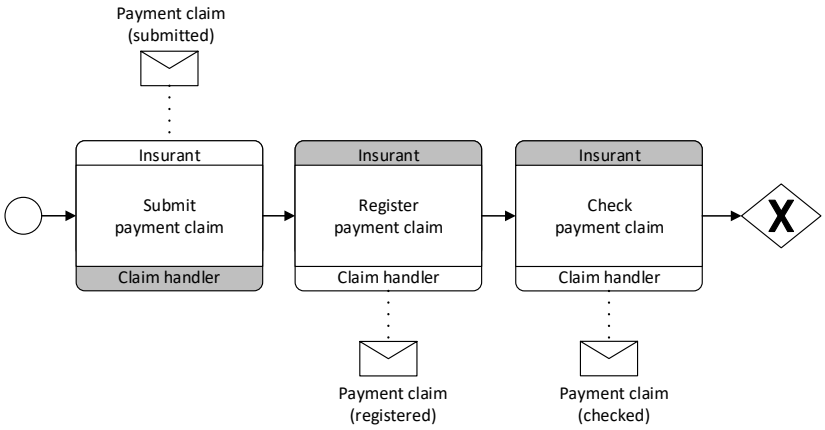
3.2 Choreographies & scenarios

In this paragraphs the choreographies and scenarios are specified. The choreographies focus on the order in which messages are exchanged within the business processes. The scenarios are then all possible paths that are determined by the decisions points based on certain business rules within the business processes.

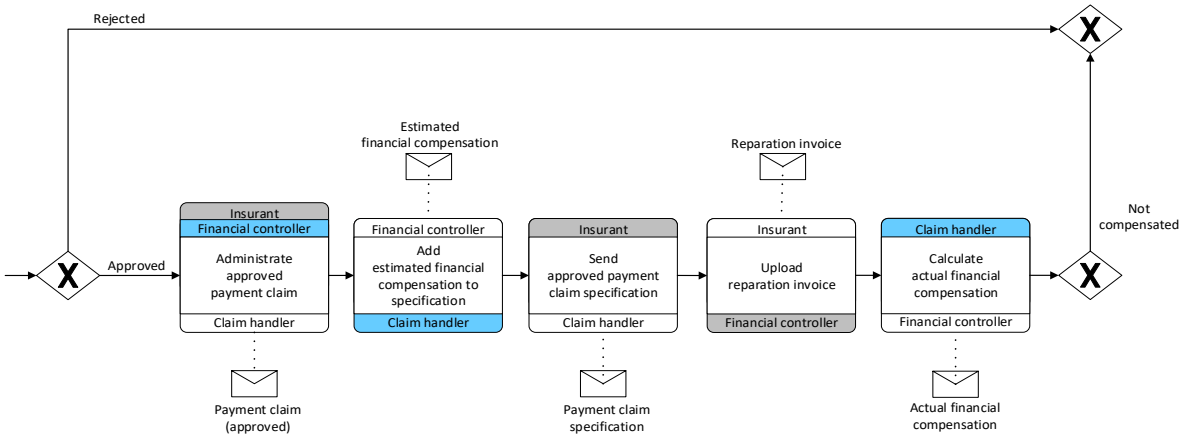
3.2.1 BPMN process choreography diagram

Basically, the BPMN process choreography diagram is a specialization of the message flows within the BPMN process diagram. Therefore, a BPMN process choreography diagram only contains the tasks that are involved in a certain message exchange, both internal and external.

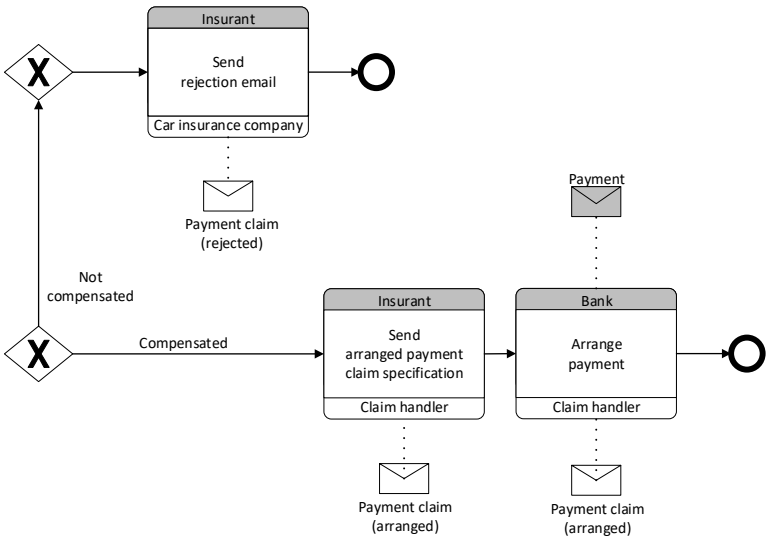
Figure 4 results in large, complex BPMN process choreography diagram. Therefore, below, the resulting diagram has been divided into three separated parts that each need to be read from left to right.



[Part 1/3]



[Part 2/3]



[Part 3/3]

Figure 6: BPMN process choreography diagram

During the entire process of handling the payment claim, 11 messages are exchanged in an asynchronous manner. Most messages are exchanges between the claim handler or financial controller, and the insurant. Some messages are sent between the claim handler and financial controller. Thus, these are internal message exchanges. There is also one message that is sent to the bank, which also sends a reply message back to the claim handler about the processed payment. In total, there are three possible scenarios: 1) the payment claim is completely approved, 2) the payment claim is rejected after the first check, and 3) the payment claim is rejected after the calculation of the actual financial compensation.

3.3 Application components & orchestrations

This paragraphs describes the use of the application components that are orchestrated in a certain way for executing the business processes on the BPMS.

3.3.1 ArchiMate application usage viewpoint

First, the ArchiMate business process viewpoint is refined below within the ArchiMate application usage viewpoint.

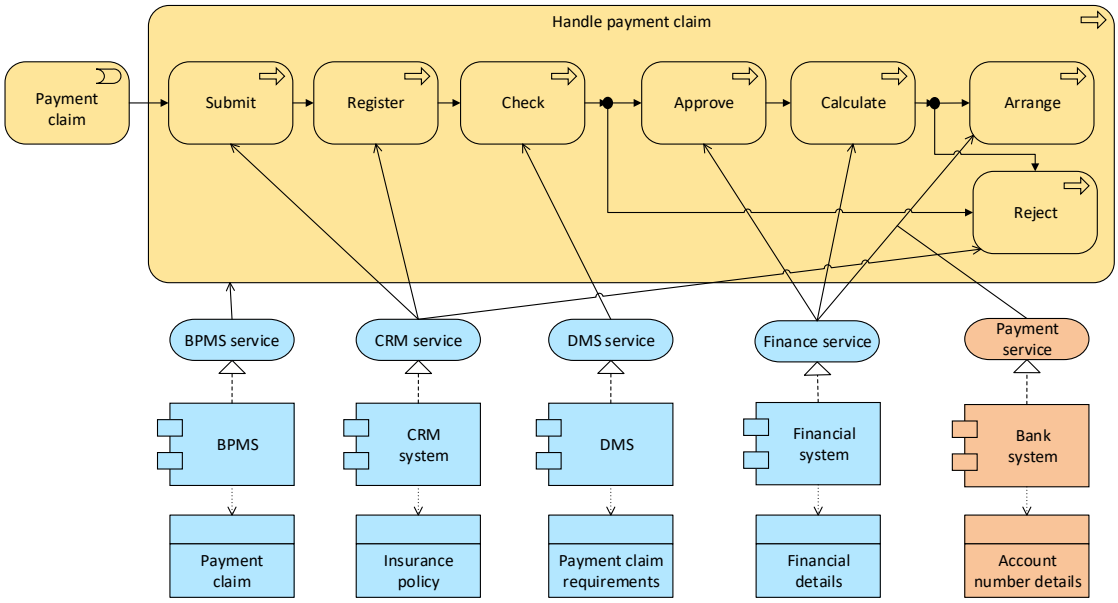


Figure 7: ArchiMate application usage viewpoint.

The BPMS is used for all business processes. Next to this, a CRM system, a financial system, and a DMS are used, as well as an external bank system. These systems have access to the required data objects, and provides certain services to the business processes.

3.3.2 ArchiMate application cooperation viewpoint

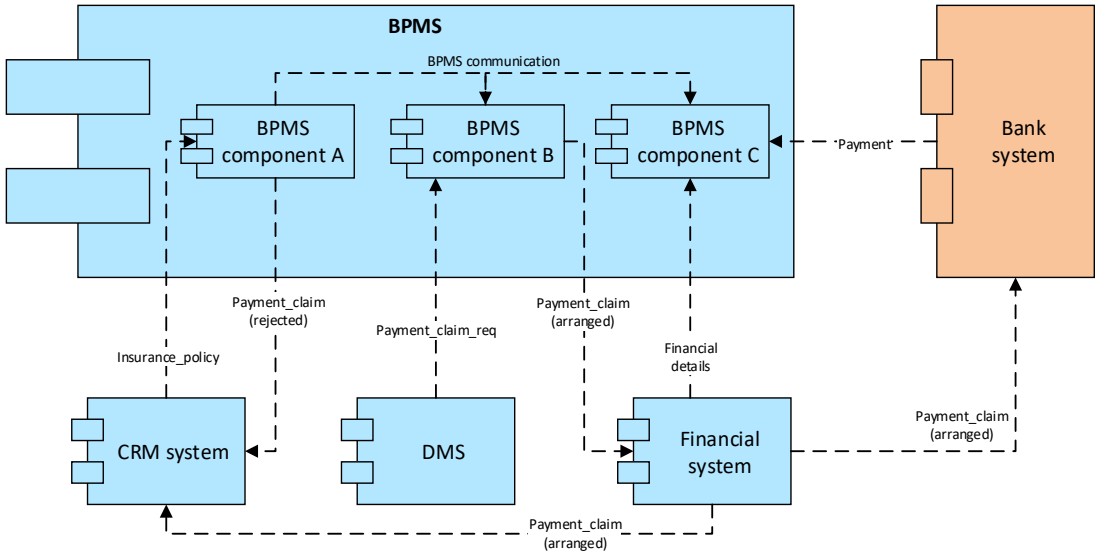


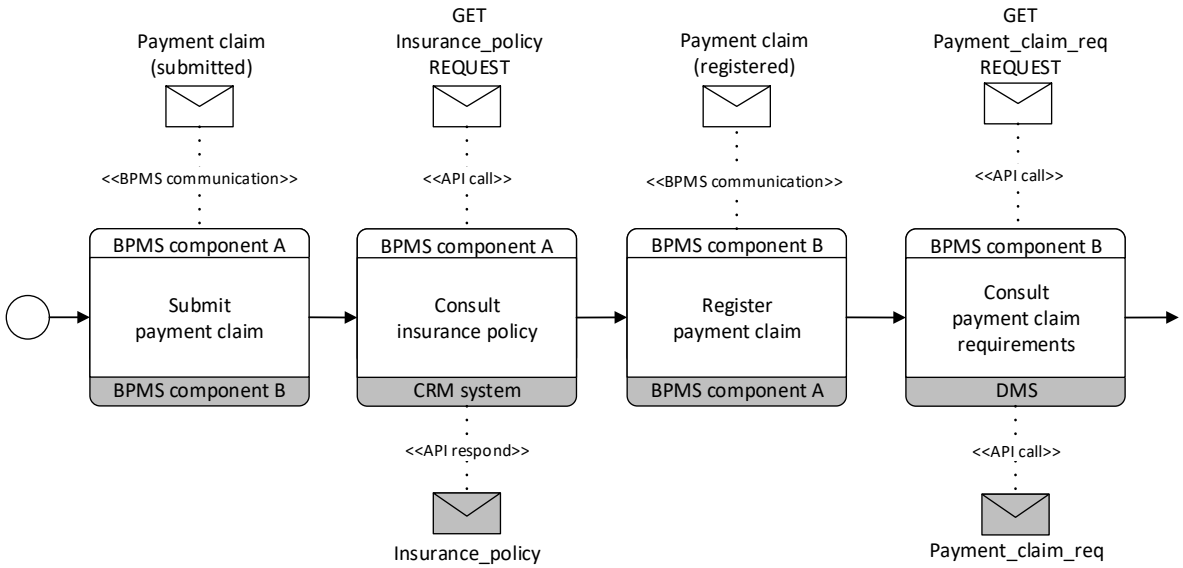
Figure 8: ArchiMate application cooperation viewpoint.

The BPMS can be decomposed into multiple components. The four aforementioned systems are integrated with the BPMS. In the ArchiMate application cooperation viewpoint below, the communication flows between the BPMS and the other systems are shown.

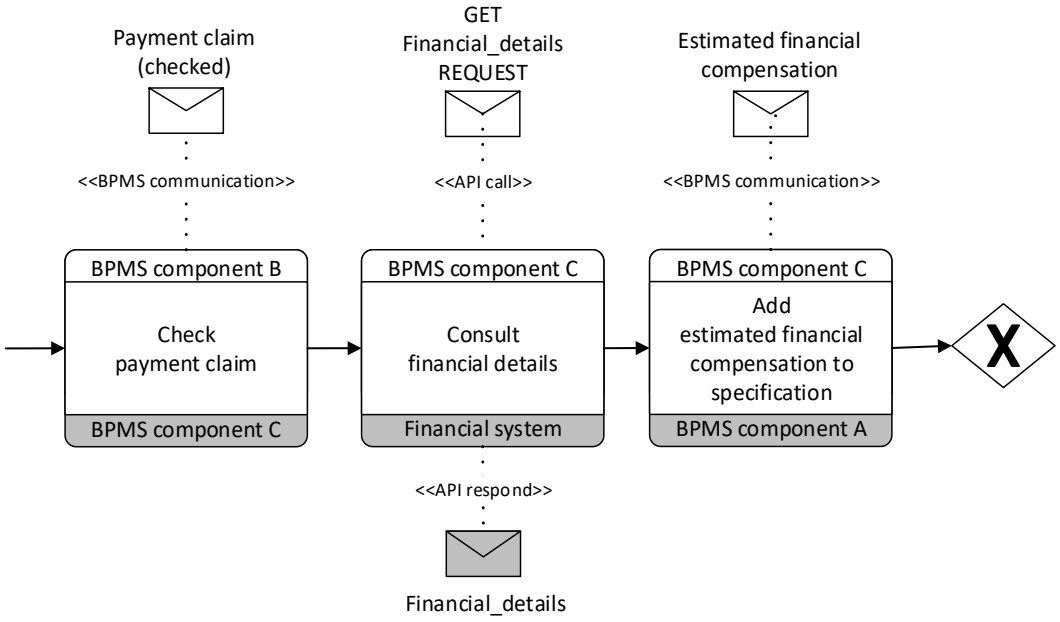
3.3.3 BPMN system choreography diagram

Next to a BPMN process choreography diagram, also a BPMN system choreography diagram is used to visualize a system-oriented choreography. Thus, the in order in which each system is used. This order needs to be aligned with the flows within the ArchiMate application cooperation viewpoint.

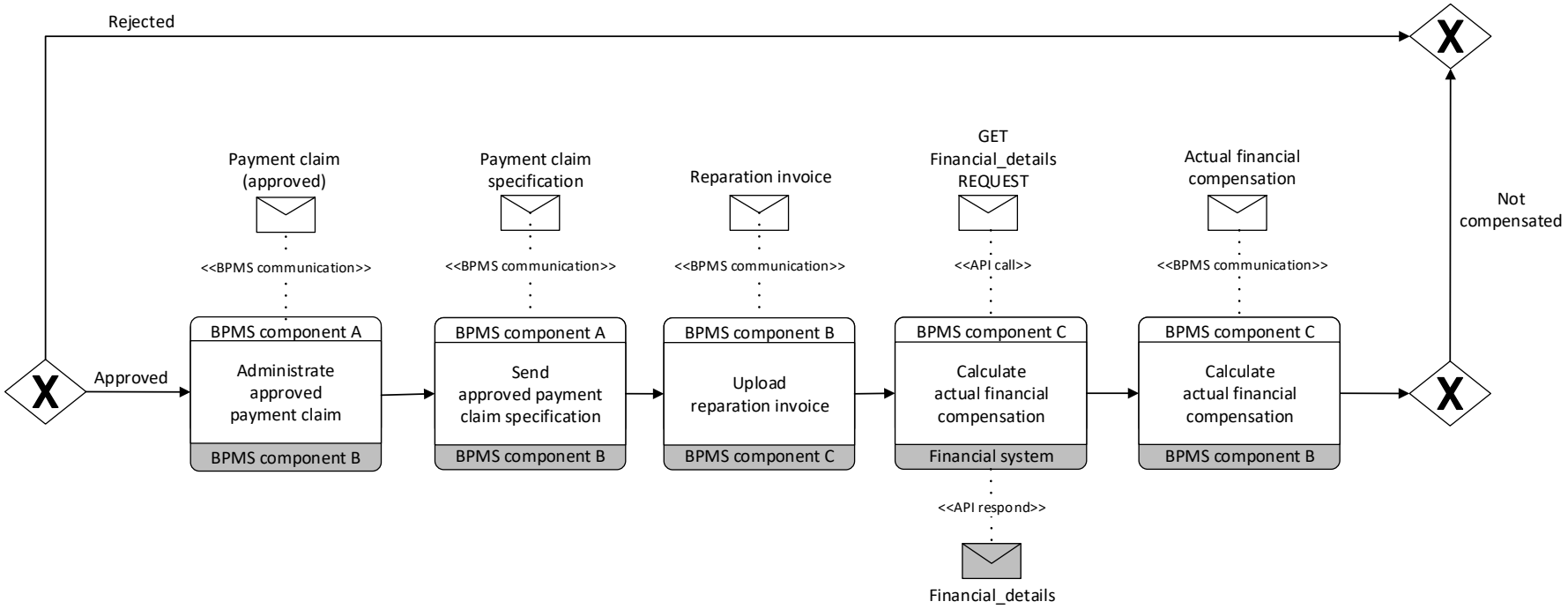
Below, the BPMN system choreography diagram is shown in four separated parts. Only the tasks that involves an interaction between two systems have been included to the diagram. Most message exchanges occur between two different components of the BPMS. In most cases, during each tasks, only a message is sent by the sender. In case of a request to, for example, a CRM system to exchange the insurance policy, also a respond message is shown.



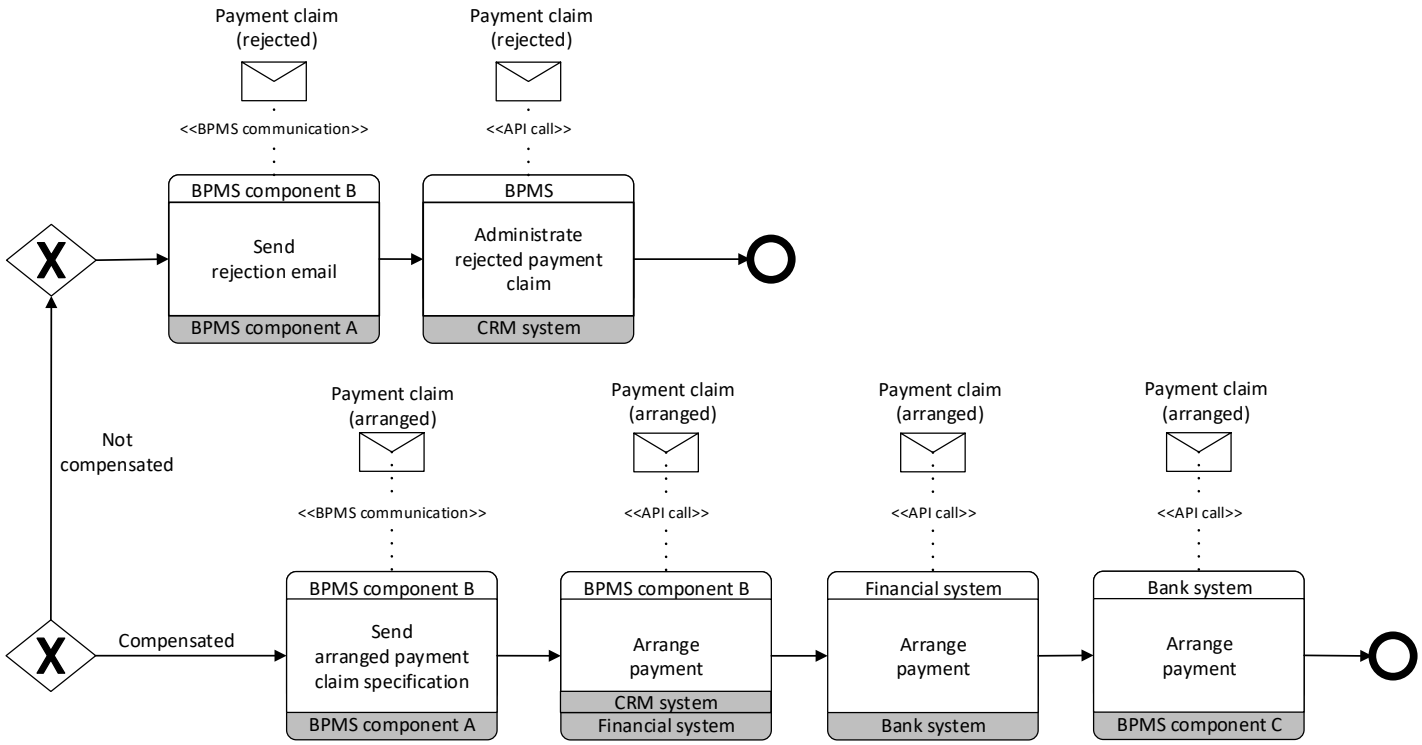
[Part 1/4]



[Part 2/4


Rejected
Payment claim
(approved)
<<BPMS communication>>
BPMS component A
Administrate
approved
payment claim
BPMS component B
Approved
Payment claim
specification
<<BPMS communication>>
BPMS component A
Send
approved payment
claim specification
BPMS component B
Reparation invoice
<<BPMS communication>>
BPMS component B
Upload
reparation invoice
BPMS component C
GET
Financial_details
REQUEST
<<API call>>
BPMS component C
Calculate
actual financial
compensation
Financial system
<<API respond>>
Financial_details
Actual financial
compensation
<<BPMS communication>>
BPMS component C
Calculate
actual financial
compensation
BPMS component B
Not
compensated

[Part 3/4]



[Part 4/4]

Figure 9: BPMN system choreography diagram

3.3.4 UML class diagram

Within the other architecture models, many data objects are mentioned. The core of the underlying data structure is visualized below by means of a UML class diagram. Thus, this diagram does not contain all data classes.

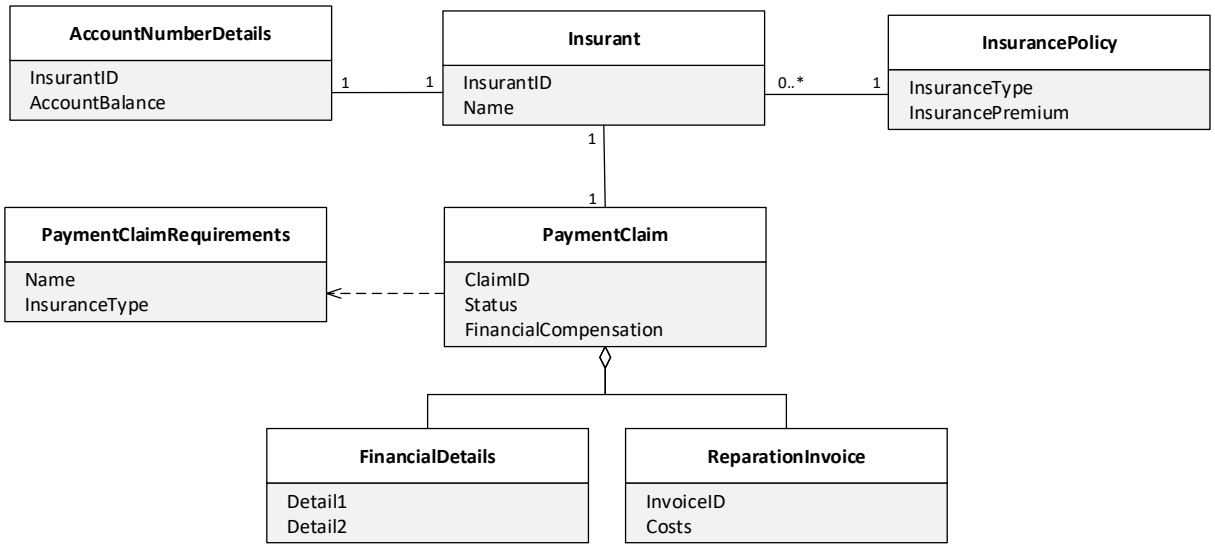


Figure 10: UML class diagram

Seven classes are shown. The most important one is called *PaymentClaim*. Each payment claim is unique. Therefore, a payment claim is submitted by only one Insurant. Thus, there is a one to one cardinality. An insurant has only one set of account details, and can have only one insurance policy. Vice versa, multiple insurants can have the same type of insurance policy. The correctness of a payment claim depends on a set of payment claim requirements a payment claim needs to meet. Furthermore, a payment claim has certain financial details, and the reparation invoice that determines the amount of financial compensation. An important attribute of a payment claim is its status. Within the name of every information/data flow or message flow that represents a payment claims, the status is indicated between brackets. For example, payment claim (submitted).

4. BPM implementation level

This section aims at the internal design of the BPMS that has been used. In this case, it is the Pega Platform from Pegasystems.

4.1 BPMS design

In this paragraph, the internal design of the Pega Platform is decomposed. For this, the most applicable diagram is the UML component diagram.

4.1.1 UML component diagram

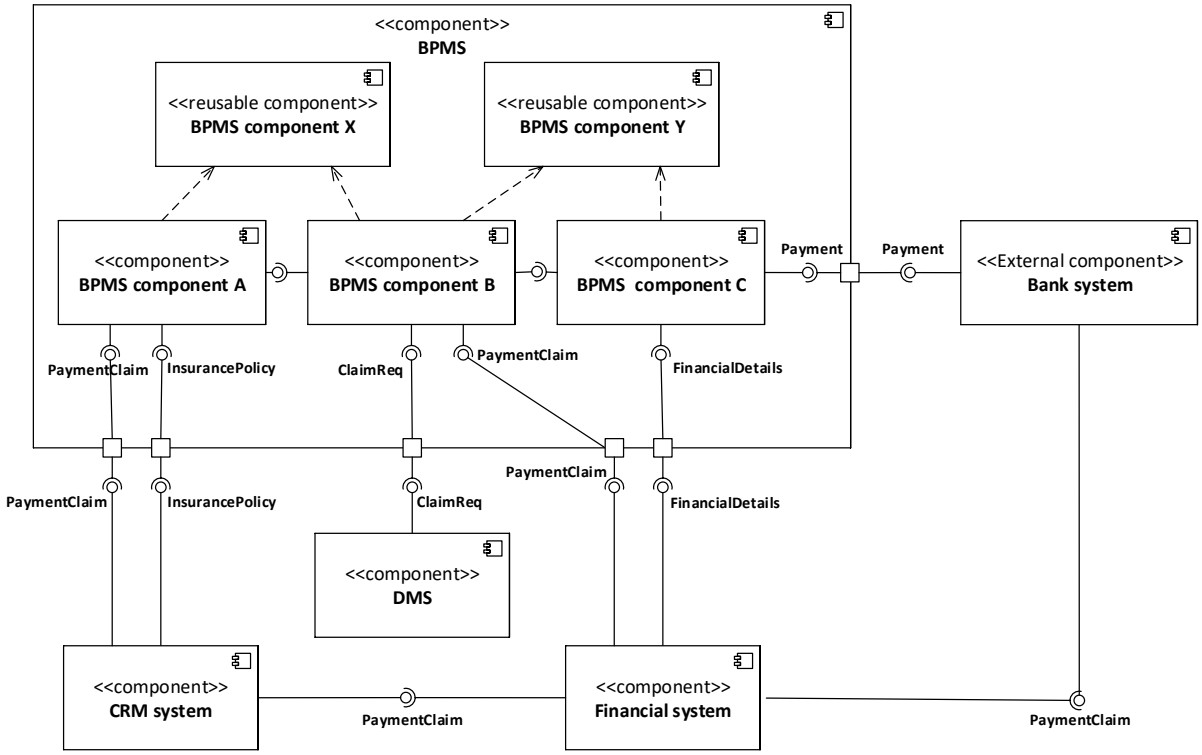


Figure 11: UML component diagram diagram

The created UML component diagram is quite similar to the ArchiMate application cooperation viewpoint. The most important difference are the fact that the types of interfaces can be specified in more details, and, in general, a more detail view on the internal structure of the BPMS can be created. In this case, two reusable components are shown. The other BPMS components depend on them. Furthermore, the bank system has been modelled by means of an external component shape. Ports have been added to visualize bi-directional communication between the BPMS and the integrated systems via the interfaces.

Eventually, based on this UML component diagram, it can be mapped what business processes and the corresponding choreographies are realized by each Pega component. For example, it can be specified that the Register payment claim is supported / realized by BPMS component A. At the business domain level, this aforementioned business process then depends on the Payment claim handling business function.

5. Alignment between requirements and architecture design

In the table below, the alignment/consistency between the elements across the different architecture models has been specified for one example. In this way, eventually, a traceability of the communication flows across the models can be created. For this, also the corresponding requirements / user stories are indicated.

Table 2: Alignment between requirements / user stories and architecture design

USid	Business actor	Business function	Business process	Application component	Data object	BPMS component
US001	Payment claim management	Payment claim handling	Register payment claim	CRM system	PaymentClaim	BPMS component A

Appendix E – Interview protocol ADL validation

Introduction

- *Personal introduction*
- *The objectives and structure of the interview*
- *Ask for permission to record the interview / start making notes*

External variables → Experiences

External factors that (indirectly) influence the other variables.

1. How many years of work experience do you have within your current field?
2. Do you have experiences with applying ADLs? If yes, which one(s)?
3. What other external factors might influence the perceived usefulness and perceived ease of use of the ADL in practice? How? (Time pressure, ownership issues, experiences etc.)

- *Short explanation of the essence of the ADL by means of the high-level architecture decomposition model, the high-level ADL diagram structure, and the Twin Peaks model.*
- *For each question on the Perceived Usefulness and Perceived Ease of Use:*
  - *Explain the meaning and objectives of the model*
  - *Explain the guidelines that have been followed to create the model*
  - *Explain the interrelation link with the other models within the scope of the ADL*
- *For all questions: why?*

Perceived usefulness (U)

Extent to which the ADL enhances/improves the development process.

4. How / for what purposes and at what moment exactly would you use this model within a project?
5. What benefits / added value do you envision when this model is used in practice?
6. What properties/aspects do you miss within this model and/or could be adjusted?

Perceived ease of use (E)

Extent to which the ADL can be applied without many effort.

7. To what extent do you think this model can be created/used and understood without too much time and effort?

Behavioral intention to use (BI)

Willingness of (partly) applying the ADL in practice.

8. What reason(s) could you give for (not) using the ADL?

Attitude toward using (A)

Feelings/expectations of applying the ADL in practice.

9. What are you feelings/expectations when the ADL will be applied in practice?
10. How could the use of the ADL be stimulated/fostered?

**

Appendix F – ADL document template: case study models

“Due to confidential information, this appendix has been left blank in this public version of the master thesis. This removed appendix can be found in the confidential version.”

Appendix G – Scientific paper (draft)

Towards a process-oriented ADL for specifying communication flows in BPMS application landscapes

Jeremy Loppies, Jan Martijn E.M. van der Werf and Marcela Ruiz

Department of Information and Computing Sciences
Utrecht University
Princetonplein 5, 3584 CC Utrecht, The Netherlands
jeremy_loppies@hotmail.com, {j.m.e.m.vanderwerf, m.ruiz}@uu.nl

Abstract. Nowadays, a BPMS system (BPMS) is often a low-code development platform that can be used to develop process-driven applications. This can result in a complex BPMS application landscape with a lot of communication flows (data/information flows, message flows) through APIs and (web) services. To specify these communication flows in a structured way, we design a process-oriented Architecture Description Language (ADL). Previous related research does not particularly focus on this domain. To design the intended ADL, relevant literature is combined with the perspectives of relevant practitioners that we gathered through both semi-structured interviews and focus groups. The design process has resulted in a process-oriented ADL that is a coherent set of several models of BPMN, Architecture, and UML. The results of a case study validation, including semi-structured interviews with practitioners, show that our ADL is perceived as a useful and valuable means that will be easy to apply and understand within BPMS development projects.

Keywords: Architecture Description Language (ADL), Communication Flows, Business Process Management System (BPMS), Application Landscape, Traceability

1 Introduction

In the past decades, Business Process Management (BPM) has become a mature discipline that is widely applied within organizations. Both practitioners and scientific researchers recognize the importance and relevance of BPM in the industry (van der Aalst, 2013). BPM can be defined as a way to map, construct and optimize business processes in a structured manner. In this way, the organizational objectives can be obtained in a better way (Weske, 2012). Emerging technologies such as Business Process Management Systems (BPMSs) have fostered the automation of end-to-end business processes. A BPMS can be defined as a software intensive system that supports the execution and monitoring of business processes by means of (partly) automating activities (Dumas, La Rosa, Mendling, & Reijers, 2018). Moreover, nowadays, a BPMS is often a low-code development platform that can be used to devel-

oped process-driven applications. The foundation of such applications are executable business process models that are both configured and executed on the BPMS.

A BPMS that is used within an organization belongs to the application landscape. Within this running environment, a BPMS communicates with other existing systems through interfaces in order to collect and use the information that is required for the execution of the business processes. For example, customer information might be gathered from an integrated CRM system. Eventually, this can results in cross-organizational (or cross-functional) business processes that are executed and orchestrated by a BPMS (Rozanski & Woods, 2012). In this paper, we look at the so-called communication flows within a BPMS application landscape. With communication flows, we mean data/information flows and message flows (choreographies) that can be specified at different architecture levels. Thus, when a lot of business processes are executed by a BPMS, many communication flows of the collection and use of information can be derived within the application landscape. Hence, from a process-oriented point of view, it can be difficult to visualize and describe the integration between the business processes, BPMS functionality, and information/data in an easy and unambiguous way. Especially, specifying in what way a BPMS relates to and communicates (transferring information) with other relevant systems from different business functions across the application landscape.

Architecture descriptions support the communication/reasoning on the architecture of a certain system. For this research, we assume that a so-called Architecture Description Language (ADL) would be a suitable solution for specifying communication flows in BPMS application landscapes in a coherent and structured way. Previous researchers barely focused on the design of ADL for our desired purposes (Clements, 1996; Malavolta, Lago, Muccini, Pelliccione, & Tang, 2013; Guessi, Cavalcante, & Oliveira, 2015). Moreover, the properties of our intended ADL are still unclear. Therefore, we tend to answer the following research question (RQ): *"What are the constituents of a process-oriented ADL for specifying communication flows in BPMS application landscapes?"* Hence, we want to design a process-oriented ADL that supports application development on a BPMS in terms of specifying communication flows within the corresponding application landscape. In short, this means that we first need to understand the common software architecture behind a BPMS, and how a BPMS is implemented and used within an organization's application landscape before actually designing our intended ADL.

The remaining sections are structured as follows. First, in section 2, the theoretical background is elaborated. Then, section 3 focuses on the research methods and the data that is used. After that, in section 4, the results of the design process of the intended ADL are discussed. Next, section 5 aims at the case study validation. Eventually, in section 6, we conclude this paper. The references can be found in the last section.

2 Background

Business Process Management (BPM) is *"a body of methods, techniques and tools to discover, analyze, redesign, execute and monitor business processes in order to optimize their performance" (*Dumas, La Rosa, Mendling & Reijers, 2018, p. 6). This continuous optimization of business processes is done in a structured way by following the steps of the BPM life cycle (Dumas et al., (2018). Basically, this life cycle entails that, for each business process, a process model is created and specified in order to analyze the possible points of optimization for a redesigned process. Monitoring the redesigned business process results in insights into its performance, which might be a trigger to revise the business process again in the same way.

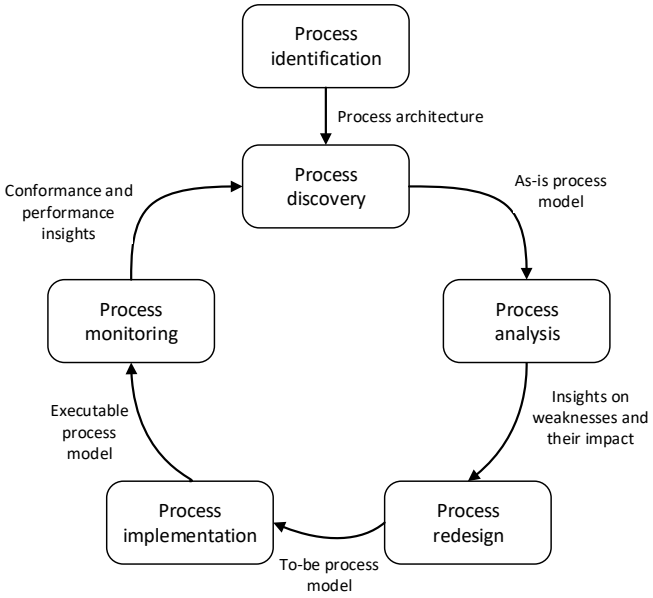


Fig. 1. BPM life cycle. Adopted from Dumas et al. (2018)

A BPMS is *"a generic software system that is driven by explicit process representations to coordinate the enactment of business processes"* (Weske, 2012, p. 5). Basically, a BPMS is a software intensive system / process-aware system that provides the functionalities/modules for partly automating the steps of the BPM life cycle. Besides process automation for workload reduction, a BPMS also provide insight into the performance (efficiency) of the business processes, and simplifies the evolution of business processes within the BPM lifecycle. A BPMS ensures that activities/events of the business processes are carried out at the right time and at the right place. Therefore, explicit executable (BPMN) process models need to be loaded into the BPMS.

The today's general architecture of a BPMS is quite similar to the Workflow Reference Model (Hollingsworth, 1995). A BPMS consists of several tools/modules and repositories (the software components) and corresponding communication flows (in-

formation exchange) through interfaces. These are shown in Figure 2. Basically, a BPMS can be seen as a system that is a coherent set of several tools (modules), repositories, and interfaces between them. Nowadays, most interfaces are configured in conjunction with / as web services in order to be able to access components of the BPMS via the internet. Usually, a BPMS runs within a service-oriented architecture (SOA), which is a widely applied architecture style (Dumas, La Rosa, Mendling, & Reijers, 2018). Ko, Lee and Lee (2009) already indicated the raising importance of SOA for BPM within the industry. Basically, by means of SOA, application components provide their business functionalities as (web) services to other applications. These services can then be invoked through interfaces. SOA makes it easy to add, remove, and reuse application components. Moreover, a BPMS can provide capabilities that simplify Enterprise Application Integration (EAI). Thus, basically, a BPMS makes use of small pieces of functionalities of the integrated systems in a certain order for the execution of the business processes.

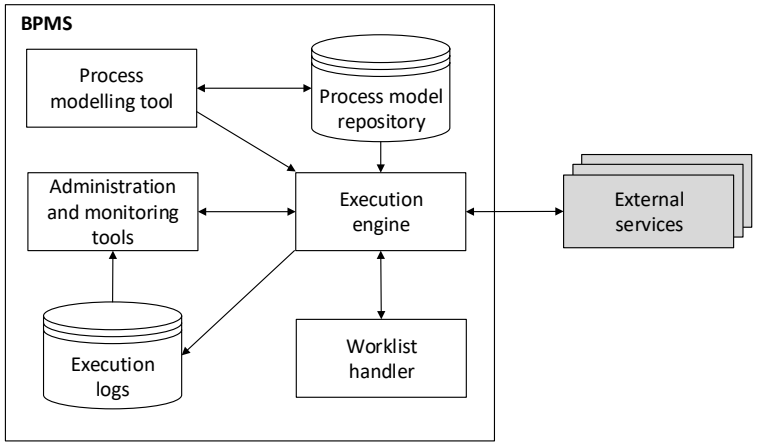


Fig. 2. General architecture of a BPMS. Adopted from Dumas et al. (2018)

This research focuses on the communication flows within a BPMS application landscape as part of the software architecture of a BPMS: "*the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both*" (Bass, Clements, and Kazman, 2003, p. 45). This architecture can be specified by means of an Architecture Description Language (ADL). We define an ADL as *any type of graphical / modelling language that can be used to visualize and specify the architecture of a system*. This definition is aligned with the definition of an ADL, according to ISO/IEC/IEEE (2011, p. 10): *"any form of expression for use in architecture descriptions"*. Next to strictly called ADLs, we also consider UML-based languages and (general) modelling languages as ADLs during this research. General/informal box-and-line languages are out of scope in order to avoid ambiguity regarding the model shapes. An ADL is meant for either a general or particular / domain-specific purpose in the field of software systems, According to Medvidovic & Taylor (1997), the main building blocks of an ADL are: *components*,

connectors, *configurations*, and *tool support*. Some existing ADLs only have a textual notation. An ADL provides different architectural purpose (e.g., analysis and validation of architecture documentation), and usually supports multiple viewpoints and abstraction levels.

3 Research method

To conduct this research in a structured way, we applied the Information Systems Research Framework (Hevner, March, Park, & Ram, 2014) as our main research method. In short, applying this method entailed that, based on the business needs / expertise from the environment (the practitioners), applicable knowledge from the knowledge base (literature, methodologies etc.) was gathered and used in order to develop and evaluate/validate a certain artefact. In this case, a process-oriented ADL that is tailored to the application landscape of a BPMS. Building and validating the ADL iteratively based on this framework ensures that it sufficiently contributes new scientific knowledge to the knowledge base, and that it is applicable in practice for achieving the desired objectives. Hevner et al. (2004) define two paradigms regarding Information Systems Research. At the one hand, Behavioral Science is about the development and justification of knowledge for predicting and/or describing relevant phenomena within the context of the business need(s). On the other hand, Design Science focuses on the creation and evaluation of artifacts that have been designed to tackle a particular business need. Due to the fact that our objective is to design and evaluate an artifact (the ADL), Design Science is the most suitable paradigm.

In addition to this, we utilized the Method Association Approach (Luinenburg, Jansen, Souer, Van De Weerd, & Brinkkemper, 2008) in order to structurally design and validate our intended ADL based on suitable existing ADLs.

4 Intended ADL design process

By means of a literature review, desk research, and several explorative semi-structured interviews with relevant practitioners, we determined the criteria and corresponding requirements that were used to select and analyze/compare suitable existing ADLs, the so-called candidate ADLs. There were several reasons why we excluded many existing ADLs from the comparison analysis, including their provided purposes, scope, and the fact that several ADLs are outdated and/or not applied in practice anymore. Eventually, we have selected and compared the following candidate ADLs:

- *Business Process Model and Notation (BPMN);*
- *ArchiMate;*
- *Unified Modelling Language (UML).*

Based on the comparison analysis, we determined to what extent each candidate ADL could support the creation of the intended ADL. In other words, the selection of the models of the candidate ADL that are relevant for our intended ADL.

Criteria	Requirement	BPMN	ArchiMate	UML	Interviews (n=5)	Focus groups (n=2)
Syntax and semantics	Graphical	✓	✓	✓	[5/5]	[2/2]
	Textual	✓	✓	✓	[5/5]	[2/2]
	Formality	Semi-formal	Semi-formal	Semi-formal	Semi-formal	Semi-formal
Viewpoints	Context	X	✓	✓	[4/5]	[2/2]
	Functional	X	✓	✓	[5/5]	[2/2]
	Information	✓	✓	✓	[5/5]	[2/2]
	Concurrency	O	X	✓	-	-
	Development	X	X	✓	-	-
Abstraction levels	System-Aggregation	X	2	4	Level 3	Level 3
	Data-Aggregation	1	1	4	Level 3	Level 3
Architecture styles	Component-based	X	✓	✓	[3/5]	[2/2]
	Layered style	X	✓	✓	[3/5]	[1/2]
	Client/server (n-tier)	X	✓	✓	[4/5]	[2/2]
	Service-oriented architecture	X	✓	✓	[5/5]	[2/2]
Architectural purposes	Creation	✓	✓	✓	[5/5]	[2/2]
	Analysis	O	O	✓	(2/5)	-
	Refinement	O	✓	✓	[5/5]	[2/2]
	Validation	✓	✓	✓	[5/5]	[2/2]
	Distributed systems	O	O	✓	[5/5]	[2/2]
	Common architecture links	O	O	O	[5/5]	[2/2]
	Design decisions capturing	Annotations	Annotations	Annotations	-	-
	Tool support	✓	✓	✓	[5/5]	[2/2]
Components	Business process related	Inter/intra	Intra	Inter/intra	Inter/intra	Inter/intra
	Software architecture related	Static	Static/dynamic	Static/dynamic	Static	Static
Connectors	Information/data flows	✓	✓	✓	[5/5]	[2/2]
	Message flows	✓	X	O	[5/5]	[2/2]
	Interfaces (APIs)	O	O	✓	[5/5]	[2/2]
Configuration	Category	Implicit	Implicit	Implicit	Implicit	Implicit

✓: provided, O: partly provided, X: not provided, -: mentioned as not relevant, [1/5] & [1/2]: mentioned as relevant 1x etc.

Table 1. ADL comparison analysis results

In Table 1, a summarized overview of results of the comparison/analysis of the most suitable existing ADLs is shown. ✓ indicates a requirement that is fully provided in a process-oriented way by the corresponding candidate ADL. Requirements marked with O are partly provided in a process-oriented way. X indicates a requirement that is not provided at all. Next to these literature review results, in the right two columns of the table, the results of both the semi-structured interviews and the focus groups have been included.

To acquire the practitioners' perspective on the requirements of the intended ADL, we conducted five semi-structured interviews with both Pega business architects and Pega system architects that each were involved in three different BPMS projects. All interviews were recorded. In addition, notes of the answers of the interviewees were made. An important part of the interview protocol were the examples of architecture models of the candidate ADLs that were shown and discussed in order to determine the relevant parts of each candidate ADL, the preferred formality, level of details / abstraction levels etc. Moreover, in this way, the interviewees could explicitly determine, for example, what kind of model they prefer for creating a business process model: the BPMN process diagram, ArchiMate business process viewpoint, and/or the UML activity diagram. In Table 1, for each requirements, it is indicated how many interviewees (five in total) mentioned that it is relevant for the intended ADL. Thus, [1/5] means that a certain requirement was mentioned as relevant for the ADL by only one interviewee, [2/5] by two interviewees and so on up to [5/5]. In addition, [-] means that a certain requirement is not relevant for the ADL. Below, we discuss the results per criteria.

Next to the semi-structured interviews, we also conducted two focus groups in order to gather the opinions from multiple practitioners at once on the design of the intended ADL. One focus group was held with 12 Pega business architects, and another focus group was conducted with 10 Pega system architects. Both focus groups were not recorded. Instead, only notes of the most important answers and/or points have been made during the discussions. These were then interpreted to target the selection criteria in Table 1. First, a general explanation of ADLs, and a specific explanation of the perceived properties and purposes of the intended ADL was given. After that, small discussions were conducted by means of example models of the candidate ADLs. The main objective was to confirm the results of the semi-structured interviews. In Table 1, similar to the semi-structured interviews, for most requirements, the results of the focus groups are shown in a certain way. Due to the fact that two focus groups were held, [1/2] means a certain requirement is relevant for the intended ADL according to only one focus group, whereas [2/2] means that both focus groups think it is a relevant requirement.

For most criteria and corresponding requirements of the ADL, the results of the semi-structured interviews, and focus groups are aligned with each other. Regardless of minor differences, the individual results between the business architects and system architects were quite similar to each other regarding both the interviews and focus groups.

The high-level architecture decomposition model in Figure 3 visualizes that our intended ADL focuses on three architecture levels regarding the development of an application that runs on a BPM platform / BPMS: *business domain level*, *process/application decomposition level*, and *BPMS implementation level*. These levels

aim at both business and IT related viewpoints, and the interrelations (translation) between them. Each level contains one or more viewpoints. For example, the *Business process* on the *Process/application decomposition level*. For each viewpoint, one or more suitable models of the candidate ADLs (ArchiMate, BPMN, UML) have been selected. Hence, basically, the ADL is in fact a coherent set of existing types of diagrams that are interrelated and complementary to each other in certain ways.

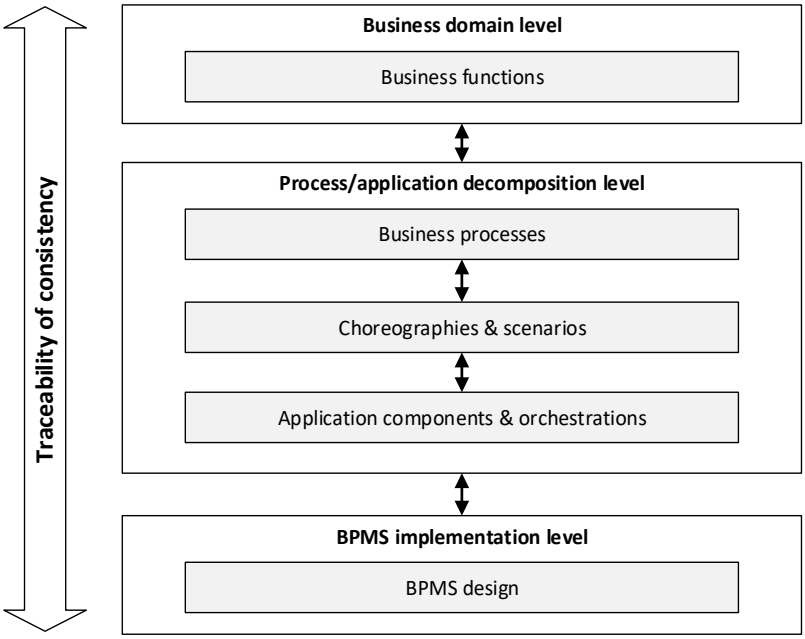


Fig. 3. High-level architecture decomposition model of the intended ADL

In Figure 4, the high-level ADL model structure is shown. This model is a high-level visualization of the syntax of the ADL, and shows the interrelations between the selected architecture models of the candidate ADLs. Therefore, it can be considered as the meta-model of the intended ADL. For each architecture level, and the corresponding viewpoints, this model shows what models from the candidate ADLs have been adopted, based on the results of the literature review, semi-structured interviews, and focus groups on the design of the ADL. The arrows between each model indicate the interrelations / mappings between them. For example, the relation "*refines*" from the BPMN business process diagram [high-level overview] to the ArchiMate business process viewpoint means that the BPMN business process diagram [high-level overview] is a detailed/extended version of the ArchiMate business process viewpoint. The BPMN business process diagram [high-level overview] itself is then used to derive the contents of the BPMN process choreography diagram. The same goes for the other relations that are visualized. Due to the fact that the BPMN system choreography diagram is applicable to both the *choreographies & scenarios*, and *application components & orchestrations*, we have partly placed it on both the aforementioned viewpoints.

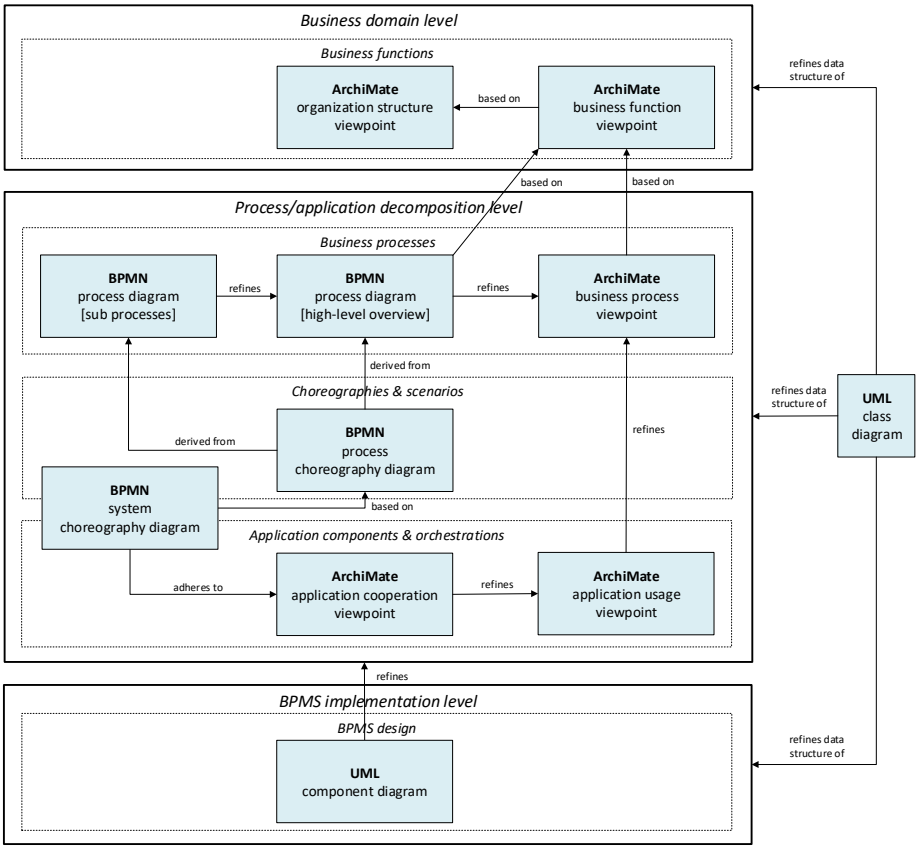


Fig. 4. High-level ADL model structure

Practitioners should apply the ADL in a certain way. Basically, the syntax of the ADL already specifies what is allowed and what is not allowed to be done in what way. Though, in general, there is a certain approach for applying the ADL in practice. By means of the Twin Peak model (Cleland-Huang, Hanmer, Supakkul, & Mirakhorli, 2013), we visualize the essence of applying the ADL in practice. This is shown in Figure 5. The essence of the Twin Peaks model is the fact that both the requirements and the architecture design need to be aligned/consistent with each other. This means that they are both specified in correspondence with each other in an iterative way. We have extended the original Twin Peaks model with the *high-level architecture decomposition model* of the intended ADL that we have placed in the middle. We have turned this high-level model to the side in order to position it in the desired way between the two peaks. Namely, we have colored the requirements peak yellow, which represents the business. On the right side, we have colored the architecture peak, which represents the IT. Thus, initially, the requirements are defined by the business, and are then iteratively specified in collaboration with the IT. As shown in Figure 5, it is indicated what viewpoints are assigned to the business stakeholders and

what viewpoints are meant for the IT stakeholders. We have done this by assigning the corresponding color of the peak to the applicable viewpoint of the ADL. In this way, a high-level division of tasks has been created.

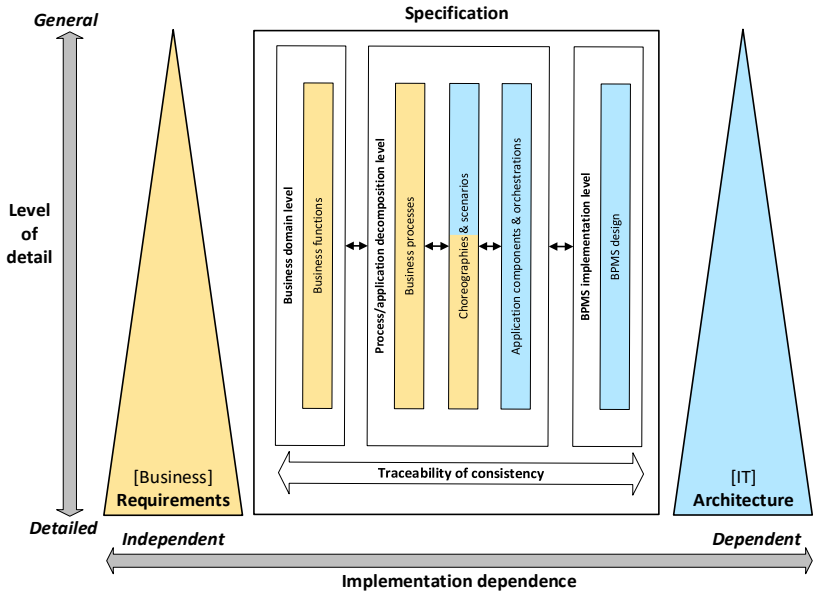


Fig. 5. The ADL within the Twin Peaks Model

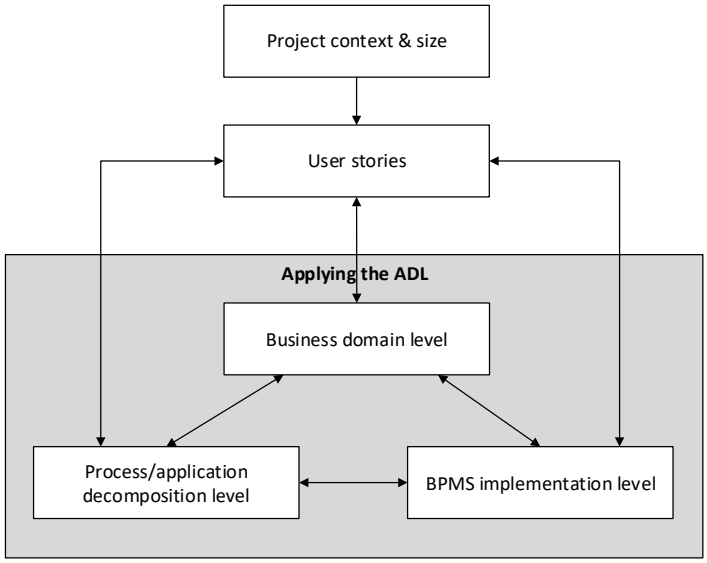


Fig. 6. General process of applying the ADL

Figure 6 visualizes the general process of applying the ADL. First, the project context needs to be examined in order to determine properties such as the organization structure, the complexity of the application landscape, and the number of actors / process participants that are involved in the corresponding business processes. Next to the project context, the most important properties of projects is the project size / duration. The project size / duration might influence the extent to which it is necessary and/or possible to correctly create and maintain create architecture models.

To put the Twin Peaks model in context, based on the project context & size, and the user stories, the first iteration of creating the models figuratively occurs at the top of both peaks at a high abstraction level. Here, general descriptions of the requirements in terms of business functions, business processes, and partly the choreographies and scenarios for designing the architecture are specified. More precisely, after high-level requirements / user stories have been formulated, the architecture models of the ADL are created. For the creation of each model, we have separately elaborated a detailed guideline, including a specification of syntactical violations (if any), for the practitioners that is demonstrated by means of a fictional running example.

After the first iteration, it depends on certain situational factors what the next steps will be. Usually, this is caused by new and/or changing user stories (requirements). Thus, after several iterations have been done, a different order of creating/maintaining the models might be more applicable, and/or a certain model might not be refined anymore. For example, if it is not relevant anymore to further refine the requirements in terms of business functions at the business domain level due to the fact that the specifications become more concrete. Moreover, eventually, it may also be relevant to start a new iteration at the BPMS design level. For example, to determine the specification of the other viewpoints based on the available (reusable) components within the BPMS design. Hence, during the next iterations, in general, the architecture specification becomes more specific when reaching the bottom of the peaks. Moreover, during each iteration, the requirements also become closer to / more dependent on the actual implementation of the architecture design of the application (Cleland-Huang, Hanmer, Supakkul, & Mirakhorli, 2013). Therefore, we prefer to use the term *Project Architecture (PA)* instead of *Project Start Architecture (PSA)* to refer to the created architecture models, because they are refined/updated multiple time during the entire project.

5 Validation: case study and semi-structured interviews

To validate the practical applicability, we conducted a case study. This entailed that we applied the ADL in practice within a real project of a BPMS software consultancy company. The selected project was focused on optimizing (partly automating) several administrative processes within an organization. The BPMS application was developed on the Pega Platform of Pegasystems, and needed to replace the functionality of several existing systems. The main objectives were reducing the amount of paper work, and preventing manual user input errors within the existing systems. In order to be able to create the architecture models by means of following the guidelines of our ADL, we collected applicable information on the selected project, including contextu-

al information, and application specification(s) documents. Eventually, the created models were then put and described in a potential ADL document template.

When the models had been created, we conducted two separated semi-structured validation interviews. One interview was conducted with a business architect. A second interview was conducted with a system architect / developer respectively. In this way, the opinions from two the business perspective and IT perspective were obtained. Both interviewees were involved in the case study project. Hence, they were familiar with the context of the project, and were able to correctly assume what will be the added value of applying the ADL during the project. The interview questions, the created models and the corresponding guidelines were provided to the interviewees before the interviews were conducted. In this way, more substantiated answers were obtained due to the fact that the interviewees could already judge the models, think of possible improvements, and their own questions in advance.

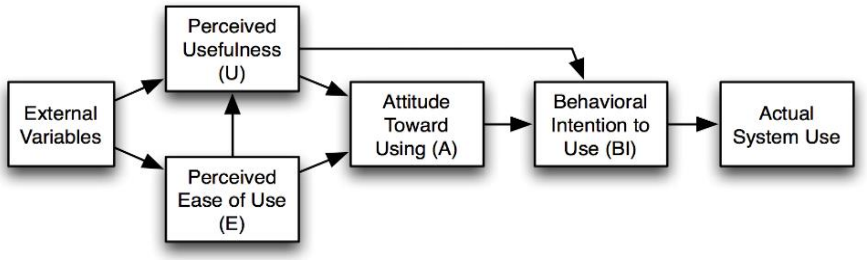


Fig. 7. Technology Acceptance model (TAM. Adopted from Davis, Bagozzi, & Warshaw (1989, p. 985)

To create the interview protocol, and create a structured validation approach, we have applied the Technology Acceptance Model (TAM), which is shown in Figure 7. We asked questions to target each variable of the TAM. The arrows visualize the links between the variables. For example, the arrow from *Perceived Ease of Use* to *Perceived Usefulness* means that the former influences the latter in a certain way. More precisely, the easier it is to use the ADL, the more useful it will be.

Based on the results of the case study validation, we can conclude that, regardless of several minor differences between the business architect and the system architect due to the different working area, the designed ADL is perceived as a useful means that will result in several benefits when it is applied in practice. In terms of specifying the communication flows within the application landscape of a BPMS, the ADL can be used to create the most applicable models. There were no other models that must be included to the specification of the ADL. Moreover, they both think the ADL will be easy to use. There were some small uncertainties regarding the purposes and abstraction levels of different models. But, by means of consulting the applied guidelines in more details, and discussing possible adjustments to the guidelines, these issues were solved. Moreover, most of the potential adjustments to the ADL only require several adjustments to the guidelines in order to make there clearer and more generic. Hence, this had a positive influence on their *Perceived Usefulness*. Eventually, the positive Behavioral Intention to Use, caused by a positive outcome of the other

variables, has resulted in the fact that, in general, both architects are willing to try to make use of the ADL in practice to benefit from its added value.

6 Conclusion and future work

We have designed a process-oriented ADL that can be used to specify communication flows within the application landscape of a BPMS. It has resulted in a coherent set of several models of BPMN, ArchiMate and UML. We have specified the constituents and the essence of the ADL by means of a high-level architecture decomposition model, a high-level ADL model structure, and both a general guideline and detailed guidelines for creating and linking the individual architecture models. One of the main constituents of our ADL is the traceability of the consistency between different architecture models that each target the communication flows from a different architecture viewpoint. The case study validation has shown that the ADL is perceived as a useful and valuable means for its desired purposes that will be easy to apply and understand within BPMS development projects.

In future research, more case studies need to be conducted in different organizational contexts in order to determine the practical applicability of our ADL regarding multiple different situational factors. Moreover, it will be interesting to look at how the ADL will be applied when other BPMSs are used. Next to this, future research needs to focus on practitioners that follow the guidelines by themselves, for example, by conducting a controlled experiment. Namely, our practitioners did not apply the ADL by themselves. Furthermore, it is required to create an "all-in-one tool" that supports the creation of all architecture models of our ADL. Especially, regarding automated syntax checks, traceability purposes / automated mappings, and drill-down possibilities.

7 References

Bass, L., Clements, P., & Kazman, R. (2003). *Software architecture in practice.* Boston, United States: Addison-Wesley Professional.

Cleland-Huang, J., Hanmer, R. S., Supakkul, S., & Mirakhorli, M. (2013). The twin peaks of requirements and architecture. *IEEE Software*, 30(2), 24-29.

Clements, P. C. (1996). A survey of architecture description languages. *In Proceedings of the 8th international workshop on software specification and design* (p. 16). Pittsburgh: IEEE Computer Society.

Davis, F. D., Bagozzi, R. P., & Warshaw, P. R. (1989). User acceptance of computer technology: a comparison of two theoretical models. *Management science*, 35(8), 982-1003.

Dumas, M., La Rosa, M., Mendling, J., & Reijers, H. A. (2018). *Fundamentals of business process management (2nd ed.).* Heidelberg: Springer.

Guessi, M., Cavalcante, E., & Oliveira, L. B. (2015). Characterizing architecture description languages for software-intensive systems-of-systems. *n Proceedings of the third international workshop on software engineering for systems-of-systems* (pp. 12-18). n.p.: IEEE Press.

Hevner, A., March, S. T., Park, J., & Ram, S. (2014). Design Science in Information Systems Research. *Design Science in IS Research MIS Quarterly*, 28(1), 75-105.

Hollingsworth, D. (1995). *The Workflow Reference Model.* TC00-1003 Issue 1.1. Workflow Management Coalition.

ISO/IEC/IEEE. (2011). *Systems and software engineering – Architecture description, ISO/IEC/IEEE 42010:2011*. Geneva: International Organization for Standardization.

Ko, R. K., Lee, S. S., & Lee, E. W. (2009). Business process management (BPM) standards: a survey. *Business Process Management Journal*, 15(5), 744-791.

Luinenburg, L., Jansen, S., Souer, J., Van De Weerd, I., & Brinkkemper, S. (2008). Designing Web Content Management Systems Using the Method Association Approach. *In Proceedings of the 4th International Workshop on Model-Driven Web Engineering (MDWE 2008)*, 106-120.

Malavolta, I., Lago, P., Muccini, H., Pelliccione, P., & Tang, A. (2013). What Industry needs from Architectural Languages: A Survey. *IEEE Transactions on Software Engineering, 39(6)*, (pp. 869-891).

Medvidovic, N., & Taylor, R. N. (1997). A Framework for Classifying and Comparing Architecture Description Languages. *Jazayeri M., Schauer H. (eds) Software Engineering - ESEC/FSE'97. ESEC 1997, SIGSOFT FSE 1997* (pp. 60-76). Berlin, Heidelberg: Springer.

Rozanski, N., & Woods, E. (2012). *Software Systems Architecture - Working With Stakeholders, Using Viewpoints and Perspectives (2nd ed.).* Courier: Addison-Wesley.

van der Aalst, W. (2013). Business Process Management: A Comprehensive Survey. *ISRN Software Engineering*, 1-37.

Weske, M. (2012). *Business Process Management.* Heidelberg: Springer.