Lehrstuhl Angewandte Informatik IV
Datenbanken und Informationssysteme
Prof. Dr.-Ing. Stefan Jablonski

Institut für Angewandte Informatik
Fakultät für Mathematik, Physik und Informatik
Universität Bayreuth

Parameterizable Process Views In Imperative Process Models

Myriel Fichtner

April 24, 2019

Universität Bayreuth

Fakultät Mathematik, Physik, Informatik
Institut für Informatik
Lehrstuhl für Angewandte Informatik IV

Master Thesis

Parameterizable Process Views In Imperative Process Models

Myriel Fichtner

1. Reviewer Prof. Dr.-Ing. Stefan Jablonski
Fakultät Mathematik, Physik, Informatik
Universität Bayreuth

2. Reviewer Dr. Lars Ackermann
Fakultät Mathematik, Physik, Informatik
Universität Bayreuth

Supervisors Dr. Lars Ackermann and
Prof. Dr.-Ing. Stefan Jablonski

April 24, 2019

Myriel Fichtner
Parameterizable Process Views
In Imperative Process Models
Master Thesis, April 24, 2019
Reviewers: Prof. Dr.-Ing. Stefan Jablonski and Dr. Lars Ackermann
Supervisors: Dr. Lars Ackermann and Prof. Dr.-Ing. Stefan Jablonski

Universität Bayreuth
Lehrstuhl für Angewandte Informatik IV
Institut für Informatik
Fakultät Mathematik, Physik, Informatik
Universitätsstrasse 30
95447 Bayreuth
Germany

Zusammenfassung

Große Unternehmen müssen sich mit komplexen Geschäftsprozessen auseinandersetzen, die eine Vielzahl von Aufgaben und Teilnehmern beinhalten. Derartige Geschäftsprozesse können unter Verwendung von Modellierungssprachen auf Prozessmodelle abgebildet werden. Dabei werden prozedurale Prozesse mit Hilfe von imperativen Modellierungssprachen wie beispielsweise der *Business Process Model and Notation* (BPMN) visualisiert, die als aktueller Standard im Bereich Geschäftsprozessmodellierung gilt. Um alle relevanten Details eines Prozesses zu modellieren, können Prozessmodelle hunderte von Elementen enthalten. Außerdem werden Prozessmodelle Kunden und Domänenexperten mit wenig Modellierungskenntnissen häufig genau so präsentiert, wie sie vom Prozessdesigner erstellt worden sind. Aus diesen Gründen sind Arbeitsabläufe für Beteiligte nur schwer nachzuvollziehen und wichtige Ausschnitte, die sie betreffen, kaum zu bestimmen. Personalisierte Prozesssichten, in denen Anwender durch Festlegung bestimmter Parameter entscheiden können, welche Teile des Prozessmodells in welcher Form visualisiert werden, adressieren dieses Problem.

Im Rahmen dieser Arbeit werden Ansätze zur Erzeugung verschiedener Sichten auf BPMN-Modelle vorgestellt und implementiert. Dafür werden parametrisierbare Verfahren zur Erzeugung von Sichten auf Prozessmodellen aus vorangegangenen Arbeiten für ihre Anwendung auf BPMN-Modelle angepasst. Mit derartigen Operationen können Prozessinformation individuell reduziert oder aggregiert werden. Diese Techniken werden um Mechanismen erweitert, die Perspektiven auf Prozessmodelle in verschiedenen Repräsentationsformen zulassen. Basierend auf dem Modellierungswerkzeug BPMN.io, werden alle Konzepte implementiert. Die Evaluierung der Implementierung mit künstlich erzeugten und realen Prozessmodellen bestätigt die Korrektheit der Verfahren und ihre flexible Einsetzbarkeit. Die Laufzeittests mittels einer Vielzahl von generierten Prozessmodellen zeigen außerdem, dass das implementierte Aggregationsverfahren auf Prozessmodelle mit über 500 Elementen angewendet werden kann. Erweiterungen des Konzepts und der Implementierung werden in Form von zukünftigen Arbeiten vorgeschlagen.

Abstract

Large companies have to deal with complex business processes involving a multitude of tasks and participants. Such business processes are visualized in process models by using modeling languages. In case of procedural processes, imperative modeling languages are used like for example the *Business Process Model and Notation* (BPMN) which is the recent standard for business process modeling. Process models can contain hundreds of modeling elements to represent all details of large business processes. Furthermore, process models are displayed to customers and domain experts having only limited process modeling knowledge in the same way as modelled by the process designer. Participants can hardly get the workflows of the underlying process or extract the parts that are necessary for them, i.e. they are somehow involved in. For this reason, personalized process views on the process model are needed. This means that users need to be able to decide which parts of the process model are contained in their view and how they are visualized by determining corresponding parameters.

In this work we present an approach to construct different types of process views based on BPMN models. Therefore we adapt parameterizable view operations from related work for their application to BPMN models. Respective operations enable users to reduce or aggregate process information in their desired way. We extend these techniques by view-building mechanisms which enable perspectives on the process model in different representation forms. Furthermore, we provide an implementation based on the modeling tool BPMN.io. The evaluation of our implementation with artificial and real-life models confirms their correct functionality and flexible utilization. The performance tests with a large number of generated process models show that the implemented aggregation procedure can even be used for process models consisting of more than 500 elements. Ideas for further extensions concerning the conceptual approach as well as the implementation are suggested.

Contents

1 Introduction

> *The best ideas emerge when very different perspectives meet.*
>
> — **Frans Johansson**
> Writer, entrepreneur and public speaker

1.1 Motivation

Large companies have to deal with complex workflows to achieve certain goals like developing a product or organizing the internal business structure. To visualize these mainly procedural workflows, the necessary process steps are modeled using an appropriate modeling notation leading to a workflow descriptive imperative process model. There are a lot of modeling languages to represent process models, in particular graphical, textual, abstract or executable languages. The *Business Process Model and Notation* (BPMN) is a standard for business processes and therefore a common used graphical language. BPMN is based on a flowcharting technique and comprises several diagram elements to cover specific modelling issues. In big enterprises the resulting process model gets large and confusing because different user or user groups with a lot of individual process steps have to be involved. Each participant has its own private business process leading to an overall process model that may comprise hundreds of process elements, like activities, gateways and data objects. Mostly not all process elements are relevant for certain stakeholders so they require different perspectives on the processes they are involved in, realized by a customizable visualization and information granularity. For example organizational positions like managers prefer an abstract overview, while technical or executing jobs rely on detailed views [34].
By providing different perspectives on processes and related data the process model gets more comprehensible to the resepective users. The possibility of creating personalized process views enables effective cross-organizational collaborations [17]. This work addresses the construction of process views for process models represented with BPMN. The derivation of process views from a process model demands certain technical conditions as well as requirements with regards to the content. In the next section this problem statement is explicitly given.

1.2 Problem Statement

Process models can be used as visualization of workflows within one enterprise or between several enterprises. If the process model is shared with other companies, there is a trade-off in the level of detail shown in the process view. On the one hand, if the process view reveals too few details of the underlying private business process, then the other partners of the network cannot effectively detect the local state of the business process, which might prevent an effective operation of the network. On the other hand, if the process view reveals all private details, including business secrets, then the provider runs the risk of loosing its competitive edge. Other partners might copy then its way of working, turning from collaborators into competitors. [17]
Not only for this reason, it is necessary that users may customize process views. To get the most meaningful overview of the workflows in which a certain participant is involved in, users should be able to decide themselves which parts of the model should be presented in an abstract or detailed way [6]. In other words, the degree of information loss should be controllable by determining certain parameters.
In order to provide such meaningful and therefore correct process views, the constructed process view has to be consistent with the underlying process model. This condition is satisfied if the orderings of the process model are respected by the view and no additional orderings are introduced in the view [17]. All operations that are available for creating process views have to be consistent. If an operation would lead to a violation of this requirement, it has to be prohibited. That means in addition, that the consistency of all process views and the process model has also to be guaranteed in the other way round. If the model is edited in a process view, all other process views and the underlying process model have to be updated with these changes [26].
To support users in keeping track of the process model and the workflows they are involved in, process views should be flexible and not only one representation form should be possible. In the case of graphical modelling languages like BPMN, process views could be represented as tables or check-lists beside a graphical representation [36]. Further process views should enable control flow and data flow perspectives.
There are only a few approaches that provide adequate techniques for visualizing and abstracting process models and among them less systems that realize these concepts to tackle the presented problem statement. In this thesis we adress techniques for constructing process views in imperative models presented with BPMN. To reach this goal we build up an interactive view modeler based on the established web-based modeling tool BPMN.io [15]. For realization we select appropriate concepts from related work and extend them to satisfy the presented requirements.

1.3 Thesis Structure

The thesis is structured as follows.

Chapter 2

In this chapter we first introduce necessary terms related to our topic. We present background information of BPMN and give an overview of its basic elements. We furthermore discuss the tool support of BPMN and present the functionality of the established web-based modeler BPMN.io. In the second part of this chapter, we present related work concerning process views by classifying them in non-parameterizable and parameterizable approaches. Afterwards we choose established approaches of the latter category and give a detailed and formal explanation of their view-building operations.

Chapter 3

In this chapter we present our conceptual approach. Therefore we define procss models according to BPMN and determine necessary assumptions. We show how we adapt view-building techniques of related work to our definitions and develop further process view mechanisms.

Chapter 4

This chapter presents our implementation which is based on the BPMN.io modeling editor. We show how our concepts are realized and point out relevant implementation details.

Chapter 5

In this chapter we evaluate our view-building mechanisms by using different artificial process models and a real-life model. We furthermore generate a large set of process diagrams and evaluate the performance of the implemented aggregation operations.

Chapter 6

We conclude our work by summarizing our results. We furthermore dicuss the contribution of the work and give suggestions for future work.

2 Preliminaries and State of the Art

In this chapter we introduce BPMN and give an overview of the modeling elements to provide a fundamental understanding of the modeling language. Furthermore, we present the etablished modeling tool BPMN.io by Camunda that supports the creation of process models with BPMN and is used as basis for our implementation presented in Section 4. The second part of this chapter is dedicated to related work concerning process views in imperative models. We present existing methods by classifying them in approaches that construct process views in a more static way and concepts that construct process views in a parameterizable way. Furthermore, we examine two important abstraction mechanisms that focus on the generation of parameterizable process views named aggregation and reduction. We distinguish between different techniques through considering assumptions that are made concering the underlying process model. To be complete, we introduce different definitions that are necessary for this work in the following section.

2.1 Definitions

A **business process** is defined as a set of activites that are performed in coordination in an organizational and technical environment. These activities jointly realize a business goal. Each business process is enacted by a single organization, but it may interact with business processes performed by other organizations [45]. According to [42], there are three types of business processes: management processes, operational processes and supporting processes. Management processes govern the operation of a system, while operational processes constitute the core business of the organization and create the primary value stream. Supporting Processes are processes that support the core processes, for example accounting and technical support.

A **business process model (BPM)** is a visualization of a business process. It consists of a set of activity models and execution constraints between them [45]. Process models are used to reason about processes (redesign) and to make decisions inside processes (planning and control)[2]. A process model can provide a comprehensive understanding of a process and enables the analysization and integration of an enterprise through its business processes [4]. The generation of a BPM is called *process modeling* by using *process modeling languages*. Depending on the underlying business

process, process modeling languages and the resulting BPM can be *imperative* or *declarative* [44]. Imperative process modeling specifies the procedure of how work has to be done and is used in case of routine business processes. These processes are characterized as stable, predictable and determined [39]. In contrast to imperative languages, declarative languages do not specify the procedure a priori and are used in case of flexible business processes [44]. Descriptive buzzwords are dynamic and decision intensive and the requirement for case-specific handling [39].

Process views can be defined as individual views on a certain business process or business process model that hide details of an internal process that are secret to or irrelevant for the customer or stakeholder [17]. Exact meanings of the term process view strongly depend on the use case, the underlying business process and the used modeling language.

2.2 BPMN and tool support

The Business Process Model and Notation (BPMN) is a method of illustrating business processes as process models in a graphical way. The modeling language BPMN was developed by the Object Management Group (OMG) to provide a notation that is readily understandable by all business users, from the business analysts that create the inital drafts of the processes, to the technical developers responsible for implementing the technology that will perform those processes, and finally, to the business people who will manage and monitor those processes [33]. Since being published in its first version 1.0, BPMN is a standard for business process modeling that provides a graphical notation for specifying business processes in a Business Process Diagram (BPD), based on a flowcharting technique tailored for creating graphical models of business process operations [46]. The newest version 2.0 of BPMN was published by OMG in 2011. Beside a lot of other extensions, one goal of this version was to standardize execution semantics for BPMN, which allows tool vendors to implement interoperable execution engines for business processes [1]. Each BPD is made up of a set of graphical elements which are explained in the following sections.

There are a lot of tools that provide a modeling framework to model business processes with BPMN. For example the software *Innovator for Business Analysts* from MID [32], the modeling tool *ARIS Community* by Software AG [3] and the modeling software from Lucidchart [24] which provides a web-based version to model diagrams online. Another established web-based tool with full support of BPMN is *BPMN.io* built by Camunda [15]. Our implementation which is described in Section 4 is built up on the BPMN.io modeling tool. Advantages of this tool that motivate our choice and insights of its technical structure are presented in Section

(a) The three event types: start (left), intermediate (middle) and end (right).

(b) The symbols for exclusive, inclusive, complex and event-based gateways.

(c) The basic task symbol (upper left) and examples of different activity types.

Fig. 2.1: Overview of the symbols according to the three flow object types: event, activity and gateway.

2.2.2. In the following, we give an overview of the basics of BPMN that are needed to understand diagrams modelled with BPMN.

2.2.1 Basics of BPMN

Each BPD is made up of a set of graphical elements which have fixed and explicit shapes. The elements can be classified into the four basic categories flow objects, connecting object, swimlanes and artifacts. The following sections summarize further information and introduce basic elements of each category according to [46]. We want to give a short overview and therefore only present the basic form and often-used types of each element. In general, all elements can be decorated with internal markers to model additional details.

Flow objects

Flow objects refer to the elements, that build a complete process flow, when connected together [41]. They can be separated in the three element types: event, gateway and activity.

Event

Events describe happenings during the course of a business process. They

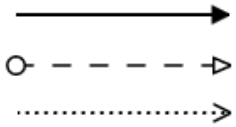


Fig. 2.2: The three types of connecting objects: sequence flow (top), message flow (middle) and association (bottom).

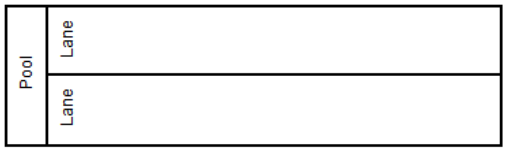


Fig. 2.3: The two swimlane objects: A pool can contain any number of lanes to model further responsibility details.

usually have a cause (trigger) or an impact (result). It is distinguished between start events, intermediate events and end events (cf. Figure 2.1(a)).

Gateway

Gateways control the divergence and convergence of the sequence flow through separating and recombining flows. They determine decision, as well as forking, merging and joining of connections (cf. Figure 2.1(b)).

Activity

An Activity describes an amount of working steps or a single working step within an process. Figure 2.1(c) shows different kinds of activity types.

Connecting objects

Connecting objects connect flow objects in a diagram to create the basic structure of a business process [41]. BPMN 2.0 defines three basic connecting objects which are visualized in Figure 2.2.

Sequence Flow

Sequence flows connect flow objects and are used to determine the order in which activities will be performed in a process.

Message Flow

Message flows represent messages from one process participant to another.

Association

Associations shows relationships between artifacts and flow objects. They are used to show the input and output of activities.

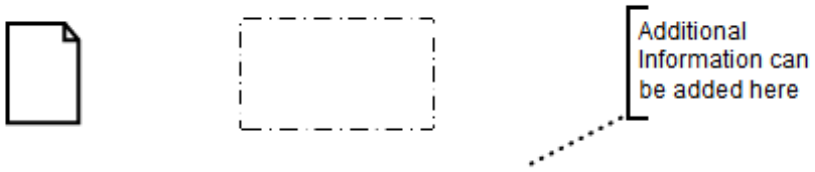


Fig. 2.4: The three artifact objects: data object, group and annotation.

Swimlanes

Swimlanes are used to organize aspects of a process in a BPMN diagram [23]. They visually separate job-sharing and responsibilities of sub-processees within a business process. BPMN supports swimlanes with the two main constructs pool and lane which are depicted in Figure 2.3.

Pool

Pools represent participants in a process. It is used as a graphical container for partitioning a set of activities from other pools. A pool may contain lanes.

Lane

Lanes a sub-partitions within a pool and extend the length of the pool vertically or horizontally. Lanes are used to represent details of responsibilities through categorizing activities.

Artifacts

Artifacts provide a mechanism for adding descriptive information about the process [18]. They are not directly related to the sequence flows or message flows of the process [30]. Any number of artifacts can be added to a diagram, as appropriate for the context of the business processes being modeled. The three artifact types are: data object, group and annotation (cf. Figure 2.4).

Data Object

Data Objects are a mechanism to show how data is required or produced by activities. They are connected to activities through associations.

Group

Groups organize tasks or processes that have significance in the overall process [41]. They can be used for documentation or analysis purposes, but does not affect the sequence flow.

Annotation

Annotations allow the modeler to describe additional flow parts of the model or notation [23].

Besides the presented elements, BPMN defines a strict syntax for the elements and the resulting BPD. In order to construct a correct BPD, these rules have to be respected. All constraints can be found in the international and official standard [33] published by OMG. Furthermore, the standard provides execution semantics which describe the meanings of executing elements in BPMN.

2.2.2 Modeling tool BPMN.io

The developers of BPMN.io provide tooling for viewing and editing BPMN diagrams. It is available both online as a web-based tool and offline as a desktop application. The web-based version supports an usage independent of the installed operating system. In contrast to other modeling tools, their libraries are extensible, embeddable and open source on GitHub. With the available BPMN 2.0 rendering toolkit and web modeler *bpmn-js*, BPMN diagrams can easily be created and modified. It is written in JavaScript and requires no server backend what makes it easy to integrate in any web application. Furthermore, a walkthrough [14] and a lot of examples [12] how to use, modify and extend bpmn-js are available. Beside technical aspects, BPMN.io has a big community that is currently using the toolkit. Consequently the toolkit stays up to date and there are a lot of users and administrators that support others to create own and modern applications. In the next sections we first present the modeling interface of BPMN.io by explaining necessary modelling functions. Second we summarize the structure and functionality of the rendering toolkit bpmn-js.

Modeling interface

The web-based modeling interface of BPMN.io is depicted in Figure 2.5. The interface consists of a canvas or drawing area, a toolbox and several functions to support the modeling creation. Modelling is enabled on the whole surface of the web page so every free space can be used as canvas.
The toolbox consists of 13 elements that can be used to interact with the drawing area. A hand tool is provided to grap and move elements on the canvas symbolized as hand in Figure 2.5. The symbol to the right refers to a lasso tool which select elements within a resizable rectangle. The tools underneath enable the creation and removal of space and the connection of elements in the model. The correct connection type is derived automatically from the object types that will be connected. The other 9 symbols correspond to their explanation in Section 2.2.1 and can be

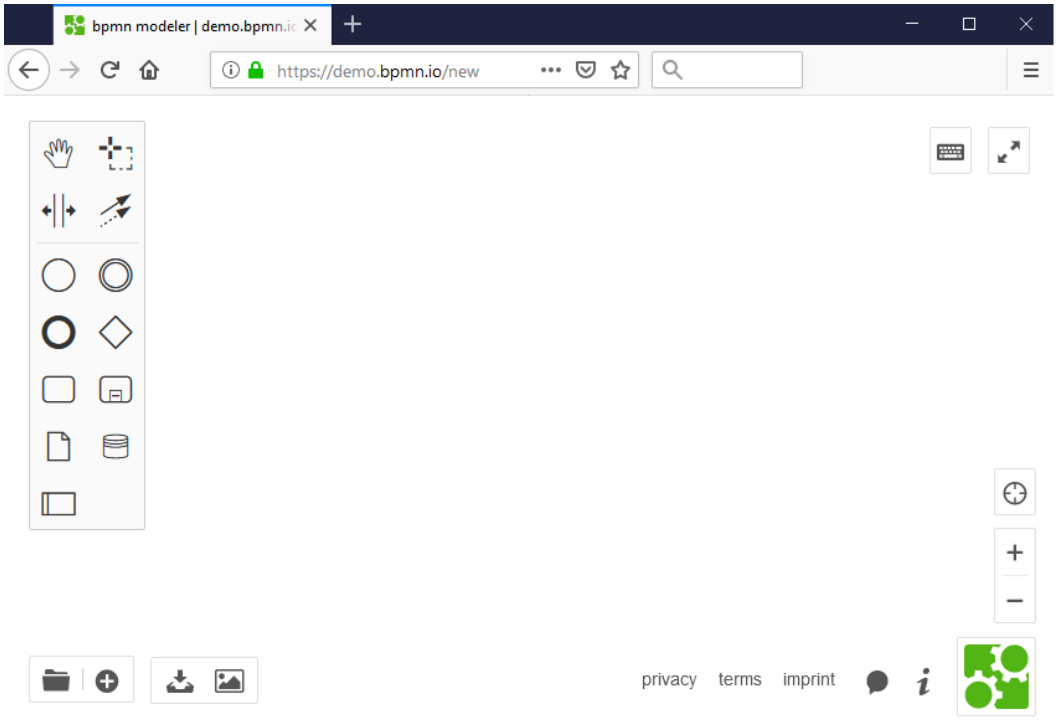


Fig. 2.5: Screenshot of the modeling interface of BPMN.io [9]. The modeling toolbox (left), buttons to save and load the diagram (bottom left) and buttons for making the drawing area more clearly (right).

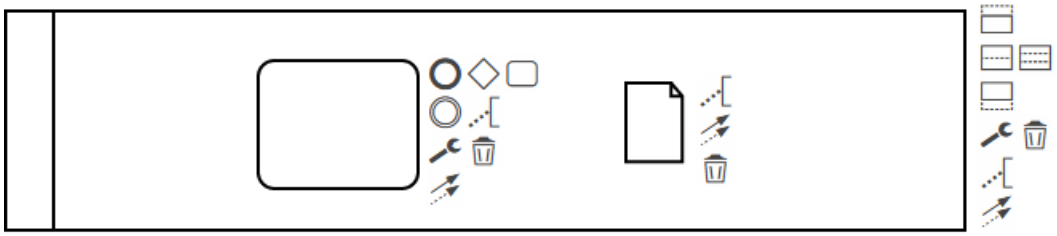


Fig. 2.6: The context menus of different modeling elements in BPMN.io [9].

inserted in the model through first clicking on the object and then clicking on a desired location in the canvas. If the modeling step is invalid in the sense that it is not supported by BPMN because it violates the syntax, an error pops up and the action is suppressed. If the modeling step is correct, a context menu opens which provides additional functionalities to further modulate the element (cf. Figure 2.6). For example the bin removes the element, while the spanner wrench symbolizes the possibility to change the type of the referring object. Different selection possibilites for activties and gateways are given in Figure 2.7. Text annotations can be added through the context menu, while text in terms of inscriptions on elements can easily be added through double click on the selected element.

The buttons on the bottom left of the web page manage the input and output communication. The folder symbol provides the uploading of a BPMN diagram from the local file system. In order to achieve a correct upload, the chosen file referring to

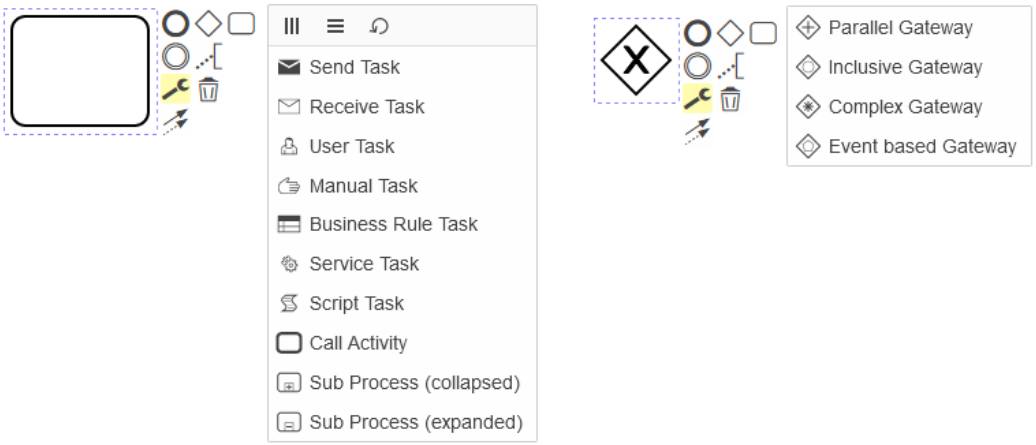


Fig. 2.7: BPMN.io provides different selection possibilities to change the type of elements [9].

a model must contain a proper XML structure like depicted in Figure 2.8. By clicking the button with the plus symbol, the creation of a new BPMN diagram is possible. Therefore the canvas is resetted and all drawn elements are discarded. The other two buttons serve to download the modelled BPMN diagram. The left one saves the diagram as $.bpmn$ file format by transforming it into a proper XML structure. The right one exports the diagram on the cancas as SVG image.
The buttons on the right of the web page provide functions to keep the overview while modeling. By clicking the button with the cross-hair symbol, the zoom is resetted. The two buttons below allow to zoom in and out. The keyboard symbol on the top right toggles an overlay that contains useful shortcuts like $ctrl + Z$ to undo the last change on the model. The other symbol enables modeling in a full screen while the browser menu, url bar and windows taskbar are hidden.

Rendering toolkit bpmn-js

BPMN.io is built up on the rendering toolkit and web-modeler bpmn-js. The library is designed in a way that it can be both, a viewer and web modeler. The viewer is used to embed BPMN 2.0 into an application and to enrich it with own data, while the modeler is used to create BPMN 2.0 diagrams inside an application. To realize these concepts, bpmn-js is built on top of two other important libraries like depicted in Figure 2.9: diagram-js and bpmn-moddle.
The library diagram-js is the renderer and modeler part of bpmn-js. It is built around a number of essential servics that can be understood as functions or instances that may consume other services to do stuff in the context of the diagram. The Core Services are named Canvas, EventBus, ElementFactory, ElementRegistry and GraphicFactory. The developers explain them as follows [14]:

(a) Example of a simple model created with BPMN.io.

```
<?xml version="1.0" encoding="UTF-8"?>
<bpmn:definitions [...]>
  <bpmn:collaboration id="Collaboration_0qnhm44">
    <bpmn:participant id="Participant_0opotxs" name="Participant" processRef="Process_1agvzfp" />
  </bpmn:collaboration>
  <bpmn:process id="Process_1agvzfp" isExecutable="false">
    <bpmn:sequenceFlow id="SequenceFlow_13or9h8" sourceRef="StartEvent_0w79y7a" targetRef="Task_1qxbo2b" />
    <bpmn:sequenceFlow id="SequenceFlow_136alr0" sourceRef="Task_1qxbo2b" targetRef="EndEvent_0sm56pw" />
    <bpmn:startEvent id="StartEvent_0w79y7a">
      <bpmn:outgoing>SequenceFlow_13or9h8</bpmn:outgoing>
    </bpmn:startEvent>
    <bpmn:task id="Task_1qxbo2b" name="Acvtivity 1">
      <bpmn:incoming>SequenceFlow_13or9h8</bpmn:incoming>
      <bpmn:outgoing>SequenceFlow_136alr0</bpmn:outgoing>
    </bpmn:task>
    <bpmn:endEvent id="EndEvent_0sm56pw">
      <bpmn:incoming>SequenceFlow_136alr0</bpmn:incoming>
    </bpmn:endEvent>
  </bpmn:process>
  <bpmndi:BPMNDiagram id="BPMNDiagram_1">
    <bpmndi:BPMNPlane id="BPMNPlane_1" bpmnElement="Collaboration_0qnhm44">
      <bpmndi:BPMNShape id="Participant_0opotxs_di" bpmnElement="Participant_0opotxs" isHorizontal="true">
        <dc:Bounds x="191" y="101" width="459" height="195" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="_BPMNShape_StartEvent_2" bpmnElement="StartEvent_0w79y7a">
        <dc:Bounds x="244" y="176" width="36" height="36" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="Task_1qxbo2b_di" bpmnElement="Task_1qxbo2b">
        <dc:Bounds x="390" y="154" width="100" height="80" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNEdge id="SequenceFlow_13or9h8_di" bpmnElement="SequenceFlow_13or9h8">
        <di:waypoint x="280" y="194" />
        <di:waypoint x="390" y="194" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNShape id="EndEvent_0sm56pw_di" bpmnElement="EndEvent_0sm56pw">
        <dc:Bounds x="589" y="176" width="36" height="36" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNEdge id="SequenceFlow_136alr0_di" bpmnElement="SequenceFlow_136alr0">
        <di:waypoint x="490" y="194" />
        <di:waypoint x="589" y="194" />
      </bpmndi:BPMNEdge>
    </bpmndi:BPMNPlane>
  </bpmndi:BPMNDiagram>
</bpmn:definitions>
```

(b) The XML structure of the model in Figure 2.8(a).

Fig. 2.8: A simple BPMN model created with BPMN.io (top) and its underlying XML structure (bottom).

Canvas

The canvas provides APIs for adding and removing graphical elements. It deals with element life cycle and provides APIs to zoom and scroll.

EventBus

The event bus helps us to decouple concerns and to modularize functionality so that new features can hook up easily with existing behavior.

ElementFactory

The element factory serves to create shapes and connections according to diagram-js' internal data model.

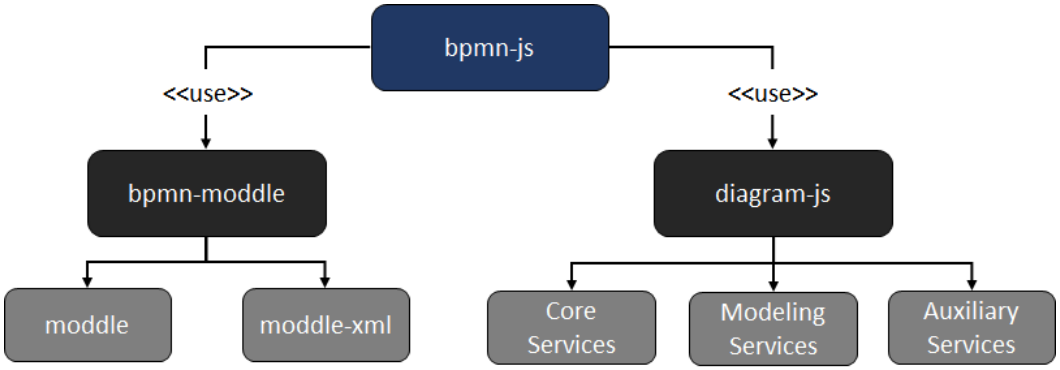


Fig. 2.9: The architecture of bpmn-js according to [14]. It is built upon the two libraries diagram-js and bpmn-moddle which involve additional modules.

ElementRegistry

The element registry knows all elements added to the diagram and provides APIs to retrieve the elements and their graphical representation by an internal id.

GraphicsFactory

The graphics factory is responsible for creating graphical representations of shapes and connections.

Another essential service is the Modeling service. The element registry manages the creation of shapes and connections according to a data model which is implemented by Diagram-js. During modeling, element relationships will be updated corresponding to user operations by the Modeling Service.
The library diagram-js provides additional helpers summarized as Auxiliary Services. A few of them are presented and explained in the following list:

CommandStack

The command stack is responsible for redo and undo during modeling.

ContextPad

The helper context pad provides contextual actions around an element.

Overlays

Overlays provide APIs for attaching additional information to diagram elements.

Modeling

The helper service modeling provides APIs for updating elements on the canvas like moving or deleting.

The library bpmn-moddle reads and writes BPMN 2.0 XML documents and provides the BPMN meta model. On import, it parses the XML document into a JavaScript object tree which can be edited during modeling and then exported back to XML once the user saves the diagram. The module is built on top of the two libraries moddle and moddle-xml. The library moddle offers a concise way to define meta-models in JavaScript, while moddle-xml reads and writes XML documents based on moddle. The BPMN meta-model is essential for bpmn-js, as it allows the validation of BPMN 2.0 documents, provide proper modeling rules and export valid BPMN documents. [14]

2.3 Related Work

In the next sections we present existing approaches concerning the construction of process views of imperative process models. In [25], requirements for process abstractions are analyzed to get user supporting process views. Several case studies have shown that it should be possible to abstract process models by hiding or aggregating information. For this reason we have to distinguish between two types of approaches. Process view constructing approaches that consider abstraction mechanisms like hide and aggregation are summarized as *parameterizable process views*. Inspired by the work of [40], we classify other methods that build process views according to fixed rules and conditions and therefore are often restricted to certain use cases as *non-parameterizable process views*.

2.3.1 Non-parameterizable Process Views

In [16] the concept of views from databases is adapted to workflows which can be understood as automatized business processes. In this context, workflow views should enable cross-organizational workflow interoperability through mechanisms that allow authorized external parties to access only the related and relevant parts of a workflow while maintaining the privacy of other unnecessary or proprietary information. The authors present a meta-model of workflow views (cf. Figure 2.10) and their semantics using a cross-organization workflow example based on a supply-chain e-service. Furthermore, they provide an implementation of the meta-model in XML to enable workflow extensions for cross-organizational interoperability in web services.

The work of [43] is based on the concept of architectural views in the field of software architecture. This can be understood as representation of a system from the perspective of a related set of concerns that has the potential to resolve the complexity challenges in process-driven service-oriented architectures [22]. The presented framework consists of a meta-meta-model, a meta-model and views which

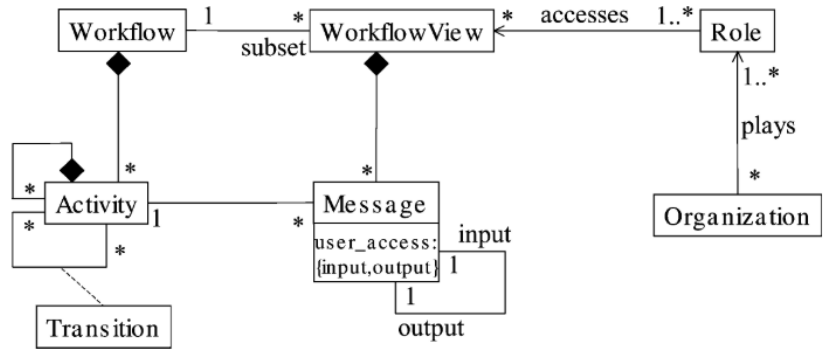


Fig. 2.10: Workflow view meta-model in UML class diagram [16].

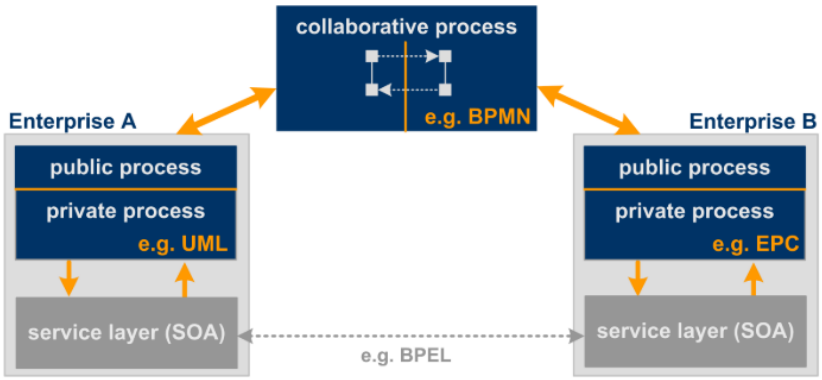


Fig. 2.11: Transformation concept from a private to a public view resulting in a collaborative process [21].

are specified using an adequate framework's meta-model. The work includes an example in the syntax of Business Process Execution Language (BPEL), where different views are presented and executable code is generated.

The authors of [21] present a concept for transforming internal private processes to publicly visible processes in a semi-automatic way through hiding the modelling complexity from the users. Each process presentation (public or private) can be understood as process view. The public view should enable cross-organizational interaction through merging public processes of different enterprises in one collaborative process. Figure 2.11 visualizes this approach. While each enterprise can use its preferred language for modelling internal processes, like the Unified Modelling Language or Event-Driven Process Chains, the collaborative process have to be modelled in a single standard notation like BPMN.

The authors of [27] propose a shared process model that provides different stakeholder views at different abstraction levels. They focus on the requirements and concrete design to synchronize related Business and IT process models, after independent editing. Each view refers to a single model while the shared process model defines correspondences between their model elements (cf. Figure 2.12). The concept is explained using examples modelled in BPMN.

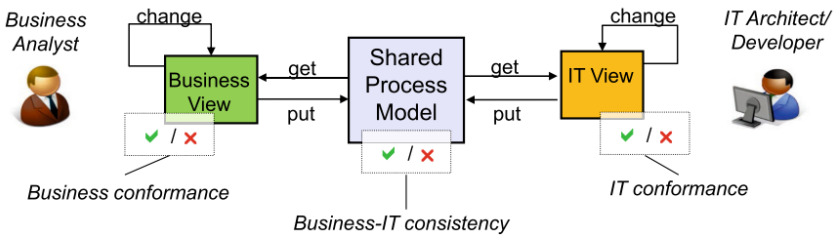


Fig. 2.12: Schematic representation of a process view synchronization via a shared process model [27].

2.3.2 Parameterizable Process Views

The authors of [28] suggest an order-preserving process model abstraction approach making use of reduction rules developed in [38]. They present an algorithm that obeys three principles: activity membership, activity atomicity, and order preservation. The latter principle requires the abstraction to preserve the ordering constraints of the initial model in the abstract model. This issue is discussed based on possible results after using aggregation methods on process models to construct participant oriented process views. Figure 2.13 shows an illustrative example of order preservation in the loop structure. Furthermore, a formal definition of a process model is provided and process execution semantics are specified.

The work of [17] is related to the work of [28] and defines a business process model as structured process model that specifies how a given set of activities is ordered. The used ordering constructs are sequence, choice, parallelism and structured loop. Each activity has a type that refers to one of these ordering constructs. Furthermore, they define a hierarchy relation, while each activity within an ordering construct is a child of the activity with the respective construction type. A process view is created through an acyclic inheritance relation on the activities. In other words, the authors present two concepts to construct a process view from a structured process model: aggregation and customization (cf. Figure 2.14). They define how a given set of nodes from the process model can be aggregated in a correct way into a single

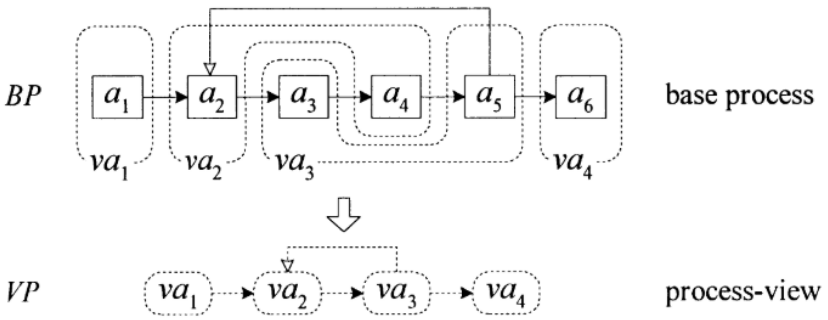


Fig. 2.13: Construction of a process view while respecting the order preservation principle in a loop structure [28].

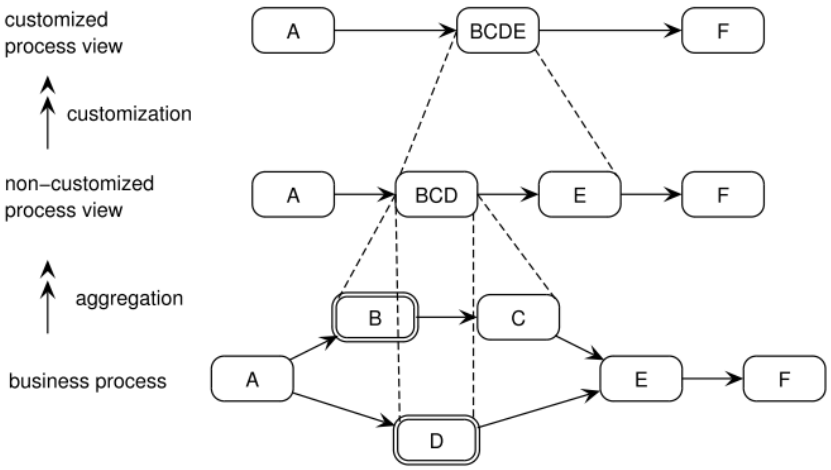


Fig. 2.14: Illustration of approach for generating customized process views [17].

node in the process view by regarding four construction rules. Second, they define how, given a computed aggregate and a structured process model, a customized and structured process view can be derived. These two steps can be repeated arbitrarily often, so a process view can itself be further aggregated into a more abstract process view. The results of this work can be easily adapted to modelling language like BPEL or BPMN.

The University of Ulm supported two projects that focus on the research of process views in the context of process models represented with BPMN.

The *Proviado* project [35] that were running since 2005, adresses major issues related to flexible process visualization and monitoring in distributed environments. In 2011, the *ProView* project [37] was newly founded and partially reuses the results from the Proviado project. The research of this project focusses on personalized and updatable process visualizations. As part of these projects, several case studies [7] were conducted to identify three fundamental process visualization dimensions. First, it must be possible to reduce complexity by discarding or aggregating process information not relevant in the given context. Second, the notation and graphical appearance of process elements (e.g., activities, data objects, control connectors) must be customizable. Third, different presentation forms (e.g., process graph, swim lane, calendar, table) should be supported [6]. A list of publications that covers all three dimensions can be found on the projects' web pages [35] and [37]. In both projects, each process corresponds to a process model that can be presented in a graphical notation by so-called *process schemes*. According to [6], a process scheme is a process graph which consists of (atomic) activities and control dependencies between them. For control flow modeling, control gateways (e.g., ANDsplit, XORsplit) and control edges are used. A *process instance* is executed on basis of a particular process model, but comprises additional run-time information to be displayed (e.g., activity states or application data). The application of view-building operations on process schemas or instances results in a process view. They state that process view operations should

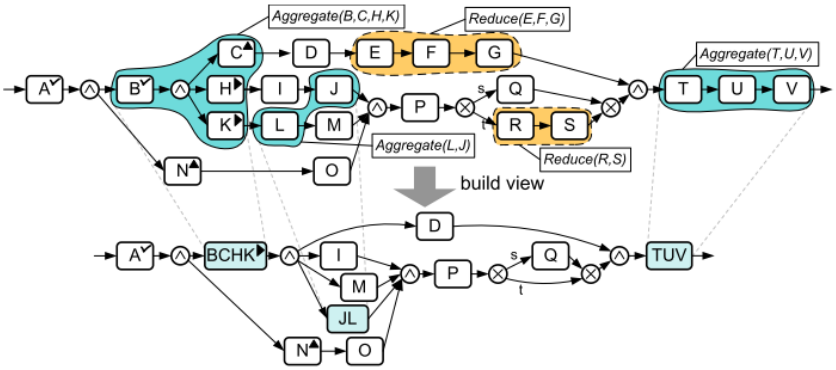


Fig. 2.15: Example of a process instance (top) and an associated process view (bottom) after using the two abstraction concepts aggregation (blue) and reduction (orange) [6].

contain two elementary concepts: It should be possible to remove process nodes (reduction) or to replace them by abstract ones (aggregation). Figure 2.15 shows an example for creating a process view. While the definition of process schemas focuses on the control flow perspective and can be applied to existing activity-orientied modeling languages like BPMN, view operations consider other perspectives as well (e.g, data elements, data flow). Formal definitions and further details can be found in [6] and [36].

The work of [40] resues results of earlier approaches that are presented above. They give a clear differentiation of these techniques and show how they address real world use cases. The authors systematically develop, classify, and consolidate the use cases for business process model abstraction and present a case study to illustrate the value of this technique. They resuse the knowledge of cartographic generalization like presented in [31] that refers to the questions why and when generalizations are needed and how they are reached. They adapt these results to business process modeling and address the last question by providing two abstraction mechanisms: aggregation and elimination.

2.3.3 View-building Operations in Parameterizable Process Views

In this section we present realization concepts for constructing process views. Therefore the two view-building operations aggregation and reduction are explained in detail. Necessary definitions are introduced and different approaches based on assumptions in terms of the process model are outlined.

Assumptions

In previous work process views are constructed based on process models with certain assumptions. The work of [17] and the thesis of [25] that is contributed to the Proview framework are restricted to block-structured process models or structured process models for short. In such models, each block has a unique entry and a unique exit point, and blocks are properly nested. These blocks are also called SESE (Single Entry Single Exit) blocks [19]. Block-structured process models require that splits and subsequent joins have the same type [17]. Publications that are part of the Proviado project lower this restriction. The authors of [36] and [5] assume that branches may be arbitrarily nested, but must be safe (e.g., a branch following a XORsplit must not merge with an ANDjoin). Furthermore, they assume that process models are acyclic and have one start and one end node. Further, it has to be connected; i.e., each activity can be reached from the start node, and from each activity the end node is reachable.
To get an abstract view on a process model, the operations aggregation and customization [17] or alternatively aggregation and reduction (also referred to as elimination or hiding) [37], [35], [40] are used. Aggregation operations enable merging a set of activities into one abstracted node, while reduction operations remove activities in a process [36]. There exist different approaches to realize these operations. In the next sections we summarize some established techniques from related work and organize them as follow: First we present an established technique referring the operation type that is restricted to block-structured process models and afterwards an approach that is applicable for arbitrarily nested, but safe process models.

Aggregation

In order to get a correct process view, it has to be consistent with the underlying process model like already mentioned in section 1.2. To tackle this problem, the authors of [17] specify concistency constraints as construction rules to ensure correct aggregates in block-structured process models. We conclude these rules and necessary definitions:

Rule 1: $X \subseteq agg(X)$

Let X be the set of nodes that have to be aggregated. Denote by $agg(X)$ the set of nodes that the aggregate constructed for X should contain in order to derive a process view consistent with the underlying process model. Naturally, all nodes of X should be in $agg(X)$.

Rule 2: $if\ x, y \in agg(X)\ and\ i \in N\ such\ that\ x < i < y\ then\ i \in agg(X)$

If two nodes x, y are aggregated such x is before y, then every intermediary node i, so $x < i < y$, should be contained in the aggregate as well. Otherwise, if an intermediary node i is not included, the aggregate will not be atomic anymore in the process view.

Rule 3: $if\ x \in agg(X)\ then\ children(x) \subseteq agg(X)$

Denote $children(n) = n' \in N \mid child(n', n)$. While a child: $N \times N$ is a predicate that defines the hierarchy relation with $child(n, n')$ if and only if n is a child (sub) node of n'. Rule 3 states that if a composite node is included in the aggregate, all its children are included as well. This ensures that in the process view aggregates have no children, i.e. no internal details of the aggregate are revealed.

Rule 4: $if\ x \in agg(X)\ and\ parent(x) \in children^{+}(lca(X))\ then\ parent(x) \in agg(X)$

If $c \in children(n)$, node n is parent of c, written $parent(c)$. The variable $children^{+}$ denotes the irreflexive-transitive closure of $children$, respectively. For a set X of nodes, the least common ancestor (lca) of X, denoted $lca(X)$, is the node x such that x is ancestor of each node in X, and every other node y that is ancestor of each node in X, is ancestor of x. Rule 4 describes the condition that if a node is in the aggregate and its parent is a strict descendant of $lca(X)$, so $x \neq lca(X)$, then its parent node has to be aggregated as well.

An efficient algorithm to get the minimal aggregate, i.e. the minimal set satisfying the four construction rules, can be found in the referred work. In order to get a more abstract view, the authors of [17] provide a fifth rule specified as extension.

Rule 5: $if\ children(lca(X)) \subseteq agg(X)\ then\ lca(X) \in agg(X)$

Rule 5 states that if all children of $lca(X)$ are included in the aggregate, node $lca(X)$ itself should be included as well.

Given a set X of nodes to be aggregated, the least common ancestor $lca(X)$ is only aggregated if $lca(X) \in X$. Sometimes, this can lead to process views which are correct, but not intuitive. In the case that a node has a single child, the fifth rule would merge both to a single node to achieve more clarity without too much information loss.

The research group of the Proviado and Proview project presents another approach for constructing aggregates which works for arbitrarily nested, but safe process models. Therefore the authors of [36] and [6] provide different elementary aggregation operations like presented as schema transformations in Figure 2.16. The upper model of each schema transformation corresponds to one use case and highlights the

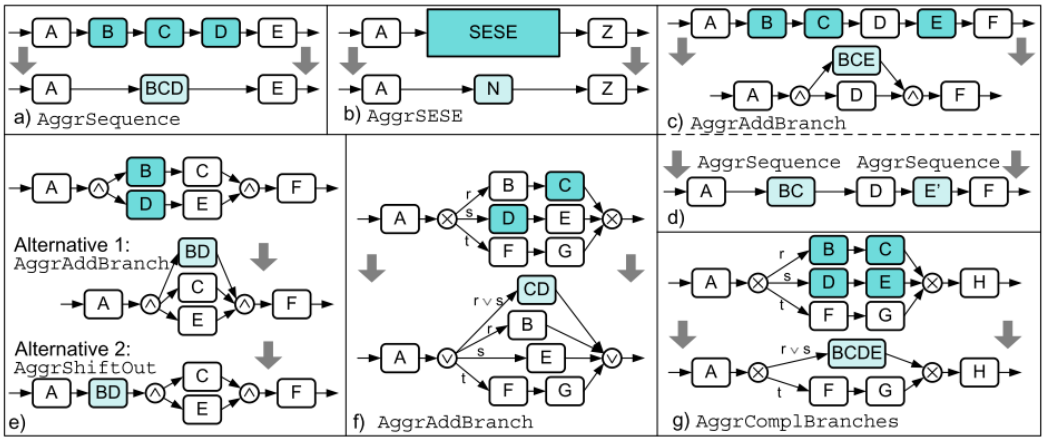


Fig. 2.16: Overview of elementary aggregation operations according to [6]. Each schema transformation shows a process model with selected activities (high saturated blue) and the resulting process model with the aggregation (low saturated blue).

activities users could possibly select for aggregation. The lower model represents the resulting view with the aggregation. In other words, the result of the aggregation depends on the ordering of the selected activities in the process model. Although the transformations preserve the structure of non-affected process regions, the dependencies between the involved activities may be changed. To explain these effect we first have to introduce two definitions provided by the authors.

Definition 1 *Let P be a process model with the set of all activities A and let $V(P)$ be a corresponding view with activity set A'. Then $V(P)$ is strong order-preserving iff $\forall n_1, n_2 \in A$ with $n_1 \neq n_2$ and $n_1 \preceq n_2 : n_1' = VNode(n_1) \wedge n_2' = VNode(n_2) \Leftrightarrow n_1' \preceq n_2'$ while $n_1 \preceq n_2 \Leftrightarrow \exists$ path in P from n_1 to n_2.*

This means in other words, that a view is called strong order-preserving if all paths that exist between activities in the process model, exist in the process view aswell. Aggregation operations that lead to process views that do not satisfy this condition indicate deviations from the underlying process model.

Definition 2 *A dependency set D_P of a process model P with the set of all activities A is denoted as $D_P = \{(n_1, n_2) \in A \times A | n_1 \preceq n_2\}$ and reflects all direct and indirect control flow dependencies between any two activities. Accordantly, $D_{V(P)}$ is the dependency set of a view $V(P)$.*

Using aggregation operations on process models lead to process views that can be assigned to one of the following three classes:

Operation	strong oder preserving	dependency preserving	dependency erasing	dependency generating
AggrSequence	+	+	-	-
AggrSESE	+	+	-	-
AggrComplBranches	+	+	-	-
AggrShiftOut	+	-	-	+
AggrAddBranches	-	-	+	-

Tab. 2.1: Overview of aggregation operation properties according to [6].

- $V(P)$ is denoted as dependency-erasing iff there exist dependency relations in D_P not existing in $D'_{V(P)}$ anymore.
- $V(P)$ is denoted as dependency-generating iff $D'_{V(P)}$ contains dependency relations not contained in D_P.
- $V(P)$ is denoted as dependency-preserving iff it is neither dependency-erasing nor dependency-generating.

The authors apply these definitions to the presented aggregation operations and classify like shown in Table 2.1. For example the operation AggrAddBranch is not dependency-preserving but dependency-erasing because the number of elements in the dependency set decreases. Furthermore, it is not strong order-preserving since the activity C is not reachable from activity B anymore (cf. Figure 2.16c). Although aggregation operations that lead to not strong-preserving process views violate the consistency rules like defined by [17], the results of case studies which are part of the proviado project show, that for the visualization of large processes, minor inconsistencies or information loss will be tolerated if an appropriate visualization can be obtained. The operation AggrSESE like examplarly depicted in Figure 2.16b describes the aggregation of SESE blocks. The Proview project only uses this kind of aggregation operation since it is restricted to block-structured process models [25]. The other presented elementary aggregation operations are not supported.

Reduction

The second elementary operation type to construct abstract process views is the reduction or also referred to as elimination or hiding. While in [17] the second operation technique is called customization and is more like a second phase to construct additional aggregations, other approaches concentrate on providing proper operations to delete or hide activities in the process model leading to an simplified process view. The work of [25] which is part of the Proview Project is restricted

Fig. 2.17: View Creation Operation RedActivity according to [25]: The activity B (blue) and all incoming and outgoing edges are deleted or relinked.

Fig. 2.18: View Creation Operation RedDataElement according to [25]: The data element $d2$ (blue) and all incoming and outgoing edges are deleted.

to block-structured process models and defines functions for deleting activities (RedActivity) and data elements (RedDataElement). For this purpose we define the business process model as central process model (CPM) which contains N activities and D data elements.

RedActivity

The method RedActivity(CPM, n) creates a process view, which corresponds to the CPM except for activity $n \in N$, which is hidden from the user together with its incoming and outgoing control edges (cf. Figure 2.17). Furthermore, the operation reinserts a control edge linking the direct predecessor of n with its direct successor in the resulting process view. All data edges associated with activity n are removed.

RedDataElement

The operation RedDataElement(CPM, d) creates a process view on CPM hiding data element $d \in D$ as well as all data edges associated with d. An example is given in Figure 2.18.

The result of RedActivity could lead to a process model view with a data flow that is not consistent with the data flow of CPM. As opposed to RedActivity, data flow correctness of the process view is preserved while using RedDataElement because all data edges associated with the removed data element are eliminated. In difference to aggregation operations on block-structured process models where whole blocks have to be aggregated to achieve a correct process view, the presented reduction operations may be used within blocks.

The Proviado project which focuses on arbitrarily nested process models provides three elementary reduction operations to achieve simplified views. Based on these

Operation	strong oder preserving	dependency preserving	dependency erasing	dependency generating
RedSequence	+	-	+	-
RedSESE	+	-	+	-
RedComplBranches	+	-	+	-

Tab. 2.2: Overview of reduction operation properties according to [6].

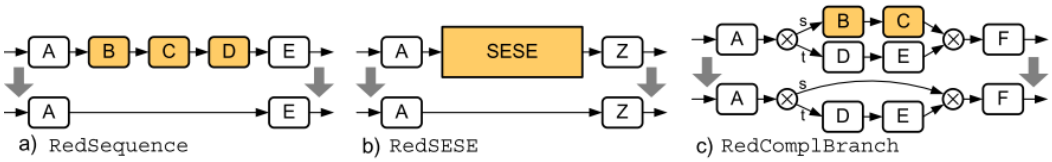


Fig. 2.19: Overview of elementary reduction operations according to [6]. Each schema transformation shows a process model with selected activities (orange) which should be reduced and the resulting process model.

elementary operations, higher-level reduction operations (e.g., for removing an arbitrary set of activities) can be realized. Although the reduction of activities always come along with a loss of information, the presented elementary operations preserve the overall structure of the remaining activities. Similar to the aggregation operarions, they are classified as shown in Table 2.2. The three elementary reduction operations are depicted in Figure 2.19. The reduction of an activity sequence (RedSequence) is realized by removing the respective activities from the process scheme and by adding a new control edge instead (cf. Figure 2.19a). This technique also works within branches of splits. Single or all activities of a branch can be deleted with the operation RedComplBranch like visualized in Figure 2.19c. Their approach also covers block-structured process models by defining the reduction operation RedSESE. This reduction is performed similar to RedSequence. Since there are only two edges connecting the block with its surrounding, the SESE block is completely removed and a new edge between its predecessor and successor is added (cf. Figure 2.19b). In order to reduce not connected activity sets the basic view operation Reduce is provided. In this case, the reduction is performed stepwise. First, the activities to be reduced are divided into subsets, such that each sub-graph induced by a respective subset is connected. Second, to each connected sub-graph, the operation RedSESE is applied, i.e. Reduce is based on the reduction of connected components using the elementary operation RedSESE.

The work of [36] that is part of the Proviado project additionally presents the reduction operation RedActivity that is comparable to the homonymous function from [25]. Moreover the operations RedSequence, RedSESE and RedComplBranch of [6] somehow include the technique of the operation RedActivity.

2.4 Conclusion

In this section we have shown fundamentals of BPMN and the functionality of the modeling tool BPMN.io. We furthermore investigated different approaches to construct process views by classyfing them in techniques that create non-parameterizable and parameterizable process views. Since non-parameterizable process views are too limiting in terms of customization, we focus on parameterizable process views that meet the demands of domain experts. The most promising approaches are [36] and [6] which are part of the Proviado project. Their concept works for non-block structured process models. Block-structured process models does not match with the complexity and workflow of real-life processes. For this reason we choose the view-building operations aggregation and reduction like defined in [36] and [6] and adapt them to BPMN like described in the next chapter. We complement these approaches by enabling further process view mechanisms which include the visualization of processes in different presentation forms. Therefore our concepts tackle two of the three fundamental process visualization dimensions proposed by [6]. In contrast to the work of [36] and [6], our approach is restricted to the modeling phase and does not cover the execution of process models. Since the prototypical implementation of the Proviado/Proview project that realizes the concepts of [36] and [6] is not publicly accessible and base on a seven year old technology, we build up our own application. The implementation of our concepts is based on BPMN.io and can be used as extended web-based modeling tool. As further explained in Section 2.2.2 BPMN.io has a lot of benefits and provides several internal functions that support our concept realization.

3 Concepts

In this chapter we present our process view creation concepts on BPMN models that are following the techniques proposed in [6] and [36] as part of the Proviado project [35]. For this purpose, we define process models according to BPMN in our context as well as assumptions that we make on them. We focus on the construction of parameterizable process views by using the two abstraction mechanisms aggregation and reduction. Furthermore, we present concepts to extract information of the process model and presenting it in matrices in order to get a tabular-like perspective on the process model.

3.1 Definitions and Assumptions

In order to adopt the concepts of [6] and [36] to BPMN, we define a process model as follows. A process model consists of a set of pools (or participants) P, while each pool $p \in P$ has a set of lanes L_p. Furthermore, it consists of a set of start nodes N_S, end nodes N_E, intermediate nodes N_I, activities N_A, gateways N_G, data objects N_{DO} and data stores N_{DS}. The set of all nodes is denoted as $N = N_S \cup N_E \cup N_I \cup N_A \cup N_G \cup N_{DO} \cup N_{DS}$. The elements of N_S, N_E, N_I, N_A and N_G have different types according to the implementation of BPMN.io. These types are modelled through mappings $NT_X : N_X \rightarrow T_X$ with $X = E, S, I, A, G$ where T_X refers to the set of valid types provided by BPMN.io. In addition, we define the mapping $NP : N \setminus N_{DS} \rightarrow P$, assigning each node except data stores to one unique participant. A process model furthermore includes a set of edges $E \subset N \times N$ containing a set of sequence flows $E_S \subset (N_S \cup N_A \cup N_G) \times (N_E \cup N_A \cup N_G)$, while $\forall (a, b) \in E_S : NP(a) = NP(b)$. The set of edges E furthermore consists of a set of message flows $E_M \subset (N_E \cup N_A) \times (N_S \cup N_A)$ with $\forall (a, b) \in E_M : NP(a) \neq NP(b)$ and a set of associations E_A which can be incoming $E_A^I \subset (N_S \cup N_A) \times (N_{DO} \cup N_{DS})$ or outgoing $E_A^O \subset (N_{DO} \cup N_{DS}) \times (N_E \cup N_A)$ resulting in $E_A = E_A^I \cup E_A^O$. Moreover these sets are disjunct $E = E_S \dot{\cup} E_M \dot{\cup} E_A$. Finally we define a process model $PM = (N, E, P, NT_X, NP)$.

Most of the approaches in previous work that are related to the creation of process views are restricted to block-structured process models because abstraction mechanisms are much easier to realize. In practice, these process models do not cover the behavior of reallife processes [5]. For this reason we choose techniques

that are not restricted to block-structured process models. Nevertheless we have to make assumptions on process models to define proper abstraction mechanisms. According to the work of [6] and [36] our implementation is restricted to acyclic process graphs that have one start and one end node for each participant or pool visualized in the process model ($\forall a, b \in N_S : a \neq b \Rightarrow NP(a) \neq NP(b)$). Second, each activity that is referred to a participant can be reached from the respective start node and from each activity the end node is reachable. Although our abstraction operations recognize unconnected nodes, they usually have no benefits with regards to the expressiveness of the model. Third, branchings may be arbitrarily nested but safe. The latter aspect denotes that if paths in branches are merged, they have to be merged with an eligible join node which has the same type as the root node of the branching. According to the BPMN standard, the second gateway is optional. Fourth, only gateways can have multiple incoming or outgoing sequence flows ($\forall n \in (N \setminus N_G)\, \exists_1(a_{in}, b_{in}) \in E_S\, \exists_1(a_{out}, b_{out}) \in E_S : b_{in} = n \wedge a_{out} = n$). This restriction does not apply for associations or message flows.
Gateways can have any amount of incoming and one outgoing sequence flows (opening gateway) or one incoming and any amount of outgoing sequence flows (closing gateway), i. e. $\forall g \in N_G : (\exists_1(a, b) \in E_S : b = g) \vee (\exists_1(a, b) \in E_S : a = g)$.

3.2 Abstraction Mechanisms

In this section we present the functionality of the abstraction mechanisms aggregation and reduction. We follow parts of the approaches of [6] and [36] and explain the conceptual background to realize these techniques on a process model PM as defined above. In contrast to process views that can be understood as different perepectives on a process model (cf. Section 3.3), the application of abstraction operations lead to process views that are created through persistent changes in the process model. This means that the execution of an aggregation or reduction operation has an effect on the number of elements in PM.

3.2.1 Aggregation

Process views that are constructed by using aggregation operations have to be consistent with the underlying process model. In this work we describe consistency as preservation of the reachability between activities. This definition is equivalent to the strong order-preservation criterion in [6] and [36]. For this reason we adopt all of their defined elementary aggregation operations excluding the aggregation AggrAddBranch (cf. Figure 2.16), since this operation is not strong order-preserving (cf. Table 2.1). All aggregation operations are restricted to the aggregation of

activities. We define $N_A^* \subseteq N_A$ as selected set of activities which should be maximally aggregated by the following aggregation operations.

AggrSequence

The operation AggrSequence merges all activities that are connected as sequence into one single node. The challenge to realize this operation is to identify all sequences that occur in N_A^*. In general, a subset $S \subset N_A^*$ is a sequence if its elements are connected by sequence flows, i.e. $\exists(s_1, s_2, ..., s_n)$ with $\{s_1, s_2, ..., s_n\} = S, \forall i \in \{1, ..., n-1\} : (s_i, s_{i+1}) \in E_S$. Since our approach is restricted to acyclic graphs and only gateways may have multiple incoming or outgoing sequence flows, branching is excluded by this definition. We find all sequences in N_A^* by considering the pairwise connectivity of all activities $a, b \in N_A^*, a \neq b$. For this purpose, we check for each pair of activities a, b if the activities are neighbored. This means that there exists a sequence flow between them while a is the source and b is target of this edge or vice versa ($(a, b) \in E_S \vee (b, a) \in E_S$). If this is the case, we create activity c which inherits the connections of these neighbored activities. We call this routine on the set $N_A^* \cup \{c\} \setminus \{a, b\}$ recursively until no further neighbored nodes can be found. Let $N_A^{*'}$ be the set with which the routine was called in the final iteration, then $PM' = ((N \setminus N_A^*) \cup N_A^{*'}, E', P', NT_X', NP')$ with E', P', NT_X' and NP' adjusted according to these changes. In the worst case the algorithm needs $\sum_{i=0}^{\#N_A^*-1}(\#N_A^* - i) \cdot (\#N_A^* - i - 1) = \frac{1}{3}n(n^2-1)$. Through this approach, it is possible to find all blocks of sequences in the selected activity set, although these blocks are not connected with each other. Figure 3.1 illustrates this aspect and shows the result of AggrSequence for a simple process model.

AggrSESE and AggrComplBranches

The aggregation operation AggrSESE serves to aggregate a whole SESE block, while AggrComplBranches merges the branches of a split that opens a block. We define a SESE block as a set of activities enclosed by an opening and a closing gateway. Therefore we first define reachability of node $a \in N$ and node $b \in N$ using edges $E' \subset N \times N$ as $reachable(a, b, E') \Leftrightarrow \exists\{(a_1, b_1), ..., (a_n, b_n)\} \subset E' : a_1 = a \wedge b_n = b \wedge b_i = a_{i+1}$ for $i = 1, ..., n-1$. A subset $B \subset N_A^*$ is a SESE block, if an opening gateway $g_o \in N_G$ with the incoming sequence flow e_o and a closing

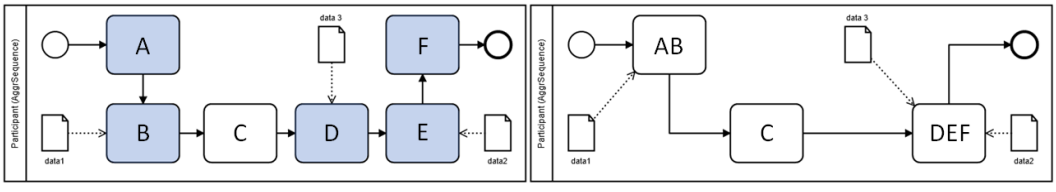


Fig. 3.1: A simple process diagram (top) with selected activity set (blue) and the result after applying AggrSequence (bottom). The selected activities are contained in two separated blocks of sequences resulting in two individual aggregation nodes.

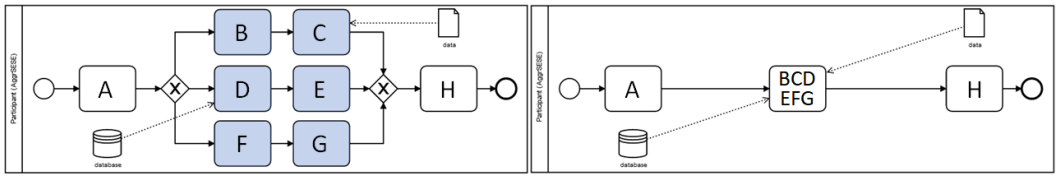


Fig. 3.2: A simple process diagram (top) with selected activity set (blue) and the result after applying AggrSESE (bottom). The selected activities are contained in a SESE block which is replaced by a single activity.

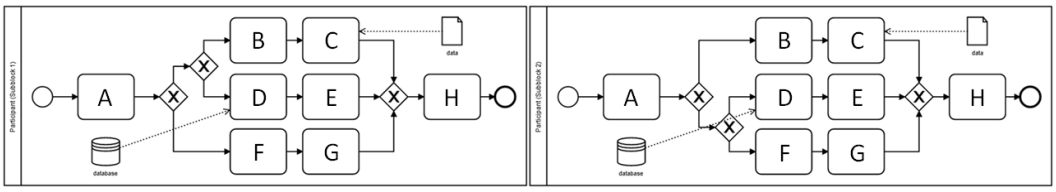


Fig. 3.3: The process diagram in Figure 3.2 can be transformed into three equivalent alternatives through inserting additional gateways. The subblock consists of activities B, C and D, E (left) or D, E and F, G (right) or B, C and F, G.

gateway $g_c \in N_G$ with the outgoing sequence flow e_c exist, so that $B = \{n \in N_A | reachable(g_o, n, E_S \setminus \{e_c\})\} = \{n \in N_A | reachable(g_c, n, E_S^* \setminus \{e_o\})\}$ where $E_S^* = \{(b,a) | (a,b) \in E_S\}$. In this definition we exclude sequences before the opening and after the closing gateway of the SESE block. Without loss of generality, we consider only opening gateways with two outgoing sequence flows. At the end of this section, we will see that all opening gateways with more than two outgoing edges can be transformed into an equivalent structure of opening gateways with two outgoing edges each. Since AggrSESE and AggrComplBranches can only be applied on SESE blocks, we first have to identify the SESE blocks contained in the selected activity set N_A^*. We find all SESE blocks in N_A^* by analyzing all pairs of opening and closing gateways in PM. For each pair (g_o, g_c) with e_o, e_c as defined above, we compute the set $B_o = \{n \in N_A | reachable(g_o, n, E_S \setminus \{e_c\})\}$ and the set $B_c = \{n \in N_A | reachable(g_c, n, E_S^* \setminus \{e_o\})\}$. Then if $B_o = B_c \subset N_A^*$, $B = B_o = B_c$ is a block and can be aggregated to activity c inheriting all incoming and outgoing message flows and associations of the elements of B. The aggregated diagram is then stated as $PM' = (N', E', P', NT_X', NP')$, while N' is N without B and all gateways within the block, including g_o and g_c. The sets E', P', NT_X', NP' are adjusted accordingly. We repeat this routine with the new selected set of elements $(N_A^* \setminus B) \cup \{c\}$ until no further SESE blocks can be found. Figure 3.2 shows the result of AggrSESE on a set of selected activities.

Each SESE block that is opened by a gateway which has three or more outgoing sequence flows can be transformed into an equivalent structure of gateways with exactly two outgoing sequence flows each. The opening gateways of these so-called subblocks inherit the gate type of the opening gate of the original block. This transformation is not unique as explained in Figure 3.3 which shows two alternatives of splitting the SESE block from Figure 3.2 into subblocks. We try all possible

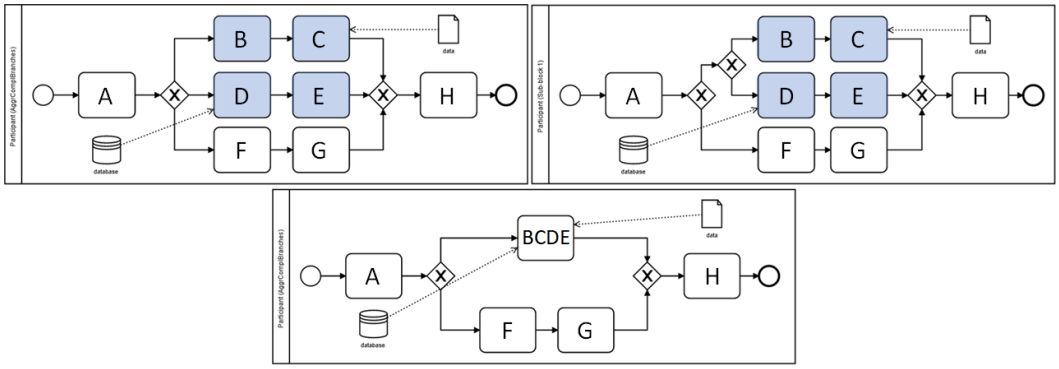


Fig. 3.4: A simple process diagram (top left) with selected activity set (blue). Through identifying the subblock as intermediate step (top right), the application of AggrSESE leads to the same result as applying AggrComplBranches on the origin process diagram (bottom). The branches referring to the selected activities are aggregated into a single branch.

transformations of opening gateways in our algorithm and therefore implicitly apply AggrComplBranches while applying AggrSESE. To be more precise, performing AggrSESE on subblocks lead to the same result as applying AggrComplBranches on the higher-level block which contains the subblocks like shown in Figure 3.4. For this reason, a separate consideration of AggrComplBranches is not necessary.

AggrShiftOut

The operation AggrShiftOut aggregates a selected set of activities following a split into a single node and swaps the positions of the aggregated node with the neighbored gateway. We distinguish between a left shift out (AggrShiftOutLeft) and a right shift out (AggrShiftOutRight). A left shift out aggregates all activities following an opening gateway while a right shift out aggregates all activities preceeding a closing gateway. First we focus on the left shift out. A subset $S \in N_A^*$ can be aggregated by performing a left shift out exactly if there is an opening gateway for which S is the set of its following activities. This means $\exists g_o \in N_G \{b|(g_o, b) \in E_S\} = S$. Then the activities are aggregated by shifting them to the position preceding the split node formalized as follows. The new set of nodes is then $N'_A = (N_A \setminus S) \cup \{c\}$ where c is the aggregation of S. The new sequence flows E'_S are adjusted so that $\forall a \in \{e|(s, e) \in E_S, s \in S\} : (g_o, a) \in E'_S$ and $\{(g_o, b) \in E_S\} \cup E'_S = \emptyset$. Furthermore $(a, g_o) \notin E'_S$ for $(a, g_o) \in E_S$. We add two new sequence flows $(a, c) \in E'_S$ and $(c, g_o) \in E'_S$. In contrast, a subset $S \in N_A^*$ can be aggregated by performing a right shift out exactly if there is an closing gateway for which S is the set of its preceeding activities. This means $\exists g_c \in N_G \{b|(b, g_c) \in E_S\} = S$. Then the activities are aggregated by shifting them to the position following the merge node formalized as follows. The new set of nodes is then $N'_A = (N_A \setminus S) \cup \{c\}$ where c is the aggregation of S. The new sequence flows E'_S are adjusted so that $\forall a \in \{e|(e, s) \in E_S, s \in S\} : (a, g_c) \in E'_S$ and $\{(b, g_c) \in E_S\} \cup E'_S = \emptyset$. Furthermore $(g_c, a) \notin E'_S$ for $(g_c, a) \in E_S$. We add two new sequence flows $(c, a) \in E'_S$ and $(g_c, c) \in E'_S$. The new diagram after performing

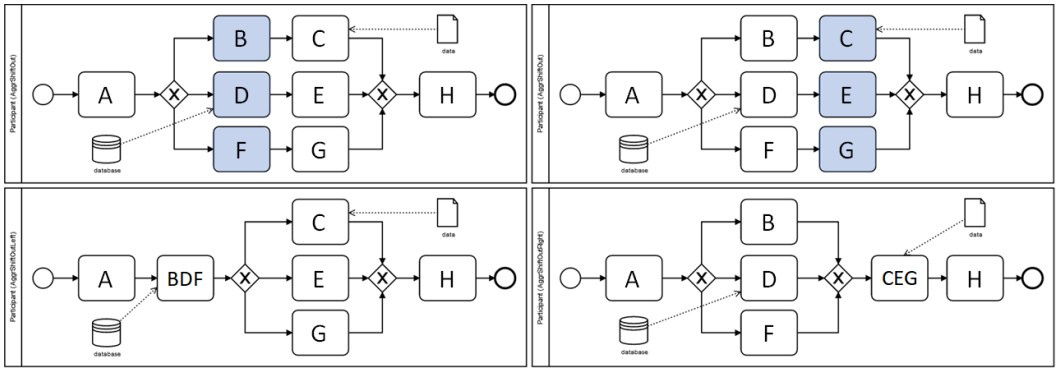


Fig. 3.5: A simple process diagram (top) with different selected activity sets (blue). Depending on the selected activity set, a left shift out (left bottom) or a right shift out (right bottom) is applied.

a shift out is then $PM' = ((N \setminus N_A) \cup N'_A, (E'_S \cup E'_M \cup E'_A), P', NT'_X, NP')$ while E'_M, E'_A, P', NT'_X and NP' are adjusted according to these changes. Figure 3.5 shows the results after applying the two shift out operations. The algorithm checks the condition above for each gateway that is directly preceeding or following an activity of the selected set, where $S \subset N^*_A$ is the set of activities following or preceeding the respective gateway. If the condition is met, S is aggregated as proposed. The authors of [6] furthermore consider the exception, if not all following or preceeding activities of the considered gateway are contained in N^*_A. In this case the authors suggest to insert additional gateways into the model leading to further branching. We exclude this case due to an increasement of the complexity of the model.

Order of aggregation operations

Like already mentioned at the beginning of this section, the operation AggrSequence and AggrSESE/AggrComplBranches are dependency preserving, while AggrShiftOut is dependency generating. This property has to be considered when performing an aggregation by using all three operations. In the following we show, that the execution order of the two dependency preserving operations has no influence on the result while the execution order of all three operations does. If a subset $S \subset N_A$ (in general $S \neq N^*_A$) can be aggregated by AggrSequence, AggrSESE/AggrComplBranches either aggregates S aswell or the subset can still be aggregated after applying AggrSESE/AggrComplBranches. The same is true for interchanging AggrSequence and AggrSESE/AggrComplBranches. We proof this as follows. Since a sequence does not contain any blocks like defined above, AggrSESE/AggrComplBranches can not be applied to it and AggrSequence is the only operation that leads to an aggregation of the selected activity set. In return, a block can contain various sequences. We show that aggregating this sequences via AggrSequence first does not change the result for AggrSESE/AggrComplBranches. Therefore consider the block B with opening gateway g_o and closing gateway g_c and let $S_1, ..., S_n \subset N_A$ be disjunct sequences. Then B' is the same block after applying AggrSequence, i.e. each sequence S_i is

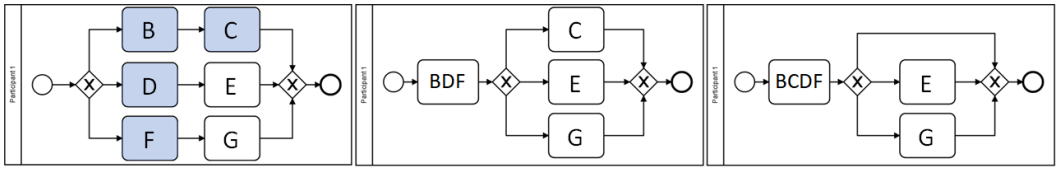


Fig. 3.6: A simple process diagram (left) with a set of selected activities (blue). Performing AggrShiftOut in a first step does not lead to the maximal aggregation and prevents further aggregations (middle), while the execution of AggrSequence in a first step, followed by AggrShiftOut aggregates all selected activities (right).

replaced with an activity s_i for $i = 1, ...n$. Since all activities in the sequence are reachable from its respective start g_o either $S_i \subset B$ or $S_i \cap B = \emptyset$ and therefore $s_i \in B' \Leftrightarrow S_i \subset B$ and B' is still a block. The aggregated activity s_i is removed during AggrSESE of B' exactly if S_i is removed during AggrSESE of B. For this reason the result is the same if AggrSequence and AggrSESE/AggrComplBranches are applied to N_A^* in an arbitrary order until none of them can be applied anymore. In contrast, this result is sensitive to the execution order if we include AggrShiftOut. An example is shown in Figure 3.6. Performing AggrShiftOut in a first step lead to a new selected activity set on which AggrSequence can not be applied anymore. In reverse, the application of AggrSequence in a first step, followed by AggrShiftOut lead to a more aggregated result. For this reason, we decide to perform AggrShiftOut only after aggregating all sequences with AggrSequence to ensure maximal aggregations. We show that our approach does not prevent any AggrShiftOut by previously applying AggrSequence or AggrSESE/AggrComplBranches. If a subset $S \subset N_A^*$ can be aggregated via AggrShiftOut, there must be an appropriate gateway with the precondition from our definition for AggrShiftOut. Since blocks are always enclosed by opening and closing gateways, either all elements of S are contained within a block or no element of S is contained within a block. Therefore AggrSESE/AggrComplBranches either removes S during aggregation or does not have any influence on the elements of S. Also after applying AggrSequence, each element from S either stays the same or is replaced by an aggregated Sequence (an activity) which does not affect the pre-

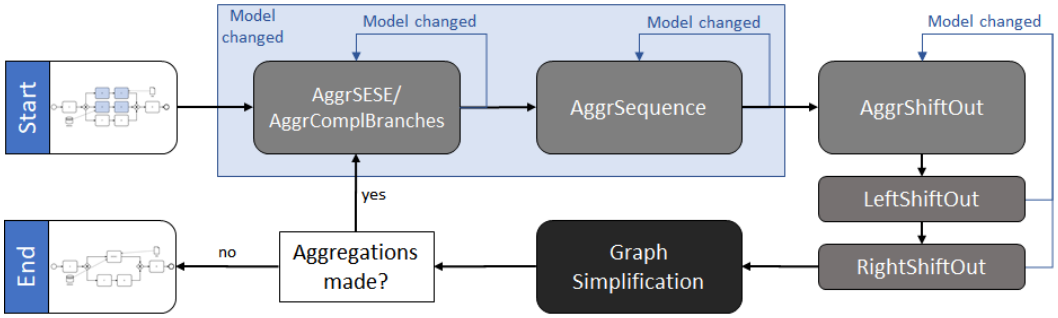


Fig. 3.7: The aggregation procedure: Each aggregation operation is executed in a loop until none of them can be applied on the selected set of activities anymore. AggrShiftOut will be executed only then the repetititon of the other two operations does not lead to further changes in the model.

condition of AggrShiftOut. Figure 3.7 shows our selected execution order. Although AggrSequence and AggrSESE/AggrComplBranches can be interchanged, we will see later in Section 5.3 that the execution order has an impact on the runtime. Since AggrShiftOut can either be a left shift out or a right shift out, first all possible left shift outs are performed followed by right shift outs. After performing all aggregation operations subsequent graph simplifications are required in order to provide the following aggregation routines with well-defined diagrams.

3.2.2 Reduction

Like presented in Section 2.3.3, the authors of [6] suggest three elementary reduction operations to eliminate sets of activities: RedSequence, RedSESE, RedComplBranches. All of these reduction operations are performed stepwise by repeating the basic reduction operation which eliminates only one single activity. We define a basic reduction as elimination of an activity $n \in N_A$ leading to the reduction of diagram PM to PM': $PM' = (N \setminus \{n\}, E', P', NT'_X, NP')$, where $E' = E \setminus \{(a,b) \in E | a = n \vee b = n\} \cup (n_{in}, n_{out})$ and n_{in} is the source element of the unique incoming sequence flow of n and n_out is the target element of the unique outgoing sequence flow of n. P', NT'_X and NP' are adjusted accordingly. Let $N^*_A \subseteq N_A$ be again the set of selected activities. In contrast to the aggregation operations which are only applicable if $\#(N^*_A) > 1$, the reduction is defined for one single activity. As similar to the aggregation procedure, N^*_A should be maximal reduced. For this reason we allow the elimination of single activities which are not neighbored to other activites in N^*_A in all three reduction operations.

RedSequence

The operation RedSequence removes all sequences $S \in N^*_A$, while single activities $n \in N^*_A$, but $n \notin S$ remain unchanged. Since all activities in N^*_A shall be removed, we perform the basic reduction as described above on each element $n \in N^*_A$ automatically including the reduction of S. This stepwise procedure guarantees a preservation of the process structure and a maximal reduction (cf. Figure 3.8).

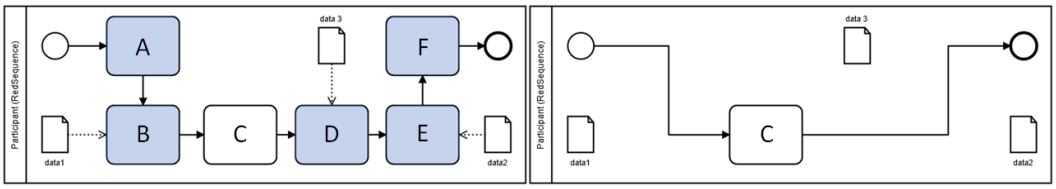


Fig. 3.8: A simple process diagram (top) with selected activity set (blue) and the result after applying RedSequence (bottom). The reduction of each sequence is guaranteed by performing the basic reduction on each activity of the selection set.

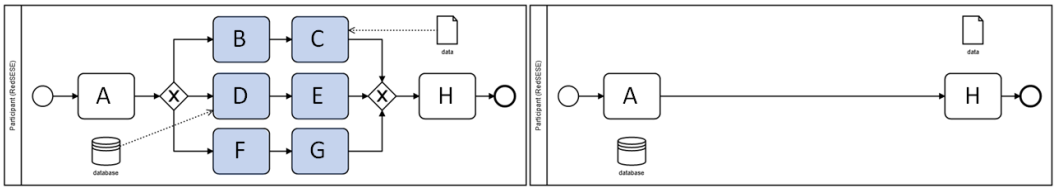


Fig. 3.9: A simple process diagram (top) with selected activity set (blue) and the result after applying RedSESE (bottom). The block-referring gateways as well as associations connected with activities in the block are removed.

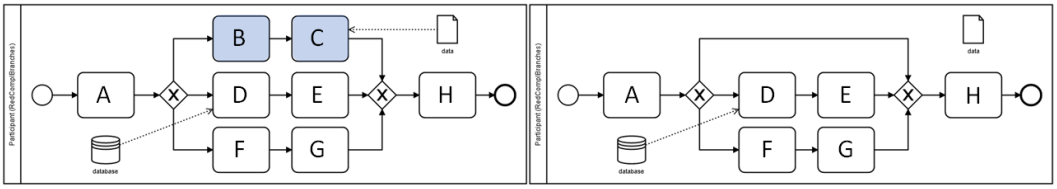


Fig. 3.10: A simple process diagram (top) with selected activity set (blue) and the result after applying RedComplBranches (bottom). Because the block is opened by a XOR gateway, the empty edge is preserved.

RedSESE

The operation RedSESE removes all blocks $B \in N_A^*$. This goal can be achieved by applying RedSequence on N_A^* and performing a postprocessing step. Since the basic reduction operation is not defined for gateways, we have to extend this definition leading to PM'' as follows. $PM'' = (N'', E'', P'', NT_X'', NP'')$ with $N'' = (N \setminus \{n \in B\}) \setminus \{g_o, g_c\}$ and $E'' = (E' \setminus \{(a, g_o), (g_o, g_c), (g_c, b)\}) \cup \{(a, b)\}$. Then P'', NT_X'' and NP'' are adjusted accordingly. The result of RedSESE is visualized in Figure 3.9.

RedComplBranches

The application of RedComplBranches leads to the reduction of branches in a block B. To be more precise, the subset $B' \subset B$ with $B' \in N_A^*$ but $B \notin N_A^*$ is reduced by applying RedSequence. In the case of reducing complete branches of a parallel branching (the opening gateway g_o and the closing gateway g_c are parallel gateways), the resulting sequence flow (g_o, g_c) can be removed as well. As opposed to this, in case of an XOR-or OR-branching, the empty sequence flow must be preserved like shown in Figure 3.10.

3.3 Further view-building concepts

In addition to the abstraction mechanisms inspired by [6] and [36], we develop further concepts to create process views based on BPMN process models. We present another graph-based approach to hide and collapse pools in process models and show how to generate process views visualized as tabular or matrix. Depending on the process model, the matrix view gives better or other insights to the process model

for a certain user than the graphical view or vice versa. We will show examples of resulting process views after applying these approaches in Sections 4.3 and 5.1.

3.3.1 Hide and Collapse

In order to get a better overview of the processes a certain participant is involved in, other participants can be put into the background. Therefore they can be hidden or collapsed. We start with the explanation of hiding a participant. Consider a participant $p \in P$. We remove all elements $n \in N \setminus N_{DS}$ with $NP(n) = p$. We call the new set of nodes N'. Furthermore, we remove all $(a, b) \in E$ with $NP(a) = p \vee NP(b) = p$, leading to a new set of edges E'. Finally we set $P' = P \setminus \{p\}$. NT_X and NP are adjusted according to these changes resulting in a new diagram $PM' = (N', E', P', NT'_X, NP')$. Multiple participant can be hidden by repeating this procedure for each of them.
The other alternative can be achieved by collapsing a pool defined as follows. Consider again a participant $p \in P$. We hide it like described above but add further elements to PM'. We extend the set of nodes N' by an activity n leading to a new set of nodes $N'' = N' \cup \{n\}$. For each message flow $(a, b) \in E_M$ which were removed by the hiding procedure, we add a new message flow (a, n) if $NP(b) = p$ or (n, b) if $NP(a) = p$ resulting in a new set of edges E''_M. Since we defined our edges as sets, E''_M contains only singletons, i.e. no redundant message flows can be added this procedure. The new diagram $PM'' = (N'', (E' \setminus E'_M) \cup E''_M, P', NT''_X, NP'')$ while NT''_X and NP'' are again adjusted. The same way multiple participants can be hidden, it is possible to collape any amount of participants.

3.3.2 Matrix-based Views

We give another presentation of a process model by considering the relation between model elements described as binary mappings $f : X \times Y \to Z$. These mappings can be visualized as matrices where each column refers to an element of X each row refers to an element of Y and each cell z_{xy} holds the result $f(x, y)$. We apply this concept to the following mappings:

Role-Role

This matrix focuses on the mapping $pp : P \times P \to \mathcal{P}(E_M)$ from pairs of participants to the message flows occuring between them. The cells of the matrix then either contain the resulting message flow or the activities it is sourcing from and targeting to.

Role-Data

We distinguish between the relation of participants to data objects and the relation of participants to data stores.

Data Objects

This matrix focuses on the mapping $do : P \times N_{D_O} \rightarrow \mathcal{P}(E_A)$ from a participant and a data object to the associations occuring between them. The cells of the matrix then either contain the the information whether the occuring association is incoming or outgoing or contain the activities it is sourcing from or targeting to.

Data Stores

This matrix focuses on the mapping $ds : P \times N_{D_S} \rightarrow \mathcal{P}(E_A)$ from pairs a participant and a data store to the associations occuring between them. As similar to the data object relation, the cells of the matrix then either contain the the information whether the occuring association is incoming or outgoing or contain the activities it is sourcing from ot targeting to.

Task-Task

This matrix focuses on the mapping $tt : N_A \times N_A \rightarrow \mathcal{P}(E_M)$ from pairs of activities to the message flows occuring between them. Since E_M is a relation $\subset N_A \times N_A$, $tt(a, b)$ has 0 or 1 element. The cells of the matrix either contain the resulting message flow or the data elements which are transferred by it. A data element d is transferred by a respective message flow $(a, b) \in E_M$, if d is connected with activity a by an output association ($\exists (d, a) \in E_A^O$) or d is connected with b by an input association ($\exists (b, d) \in E_A^I$).

4 Implementation of Process Views with BPMN.io

In this chapter we present our implementation based on BPMN.io that realizes our concepts along with platform specific details. We summarize the technical background and present the extension of the user interface of BPMN.io as well as the underlying program architecture and functionality. We focus on important implementation details that show the access and usage of internal functions provided by BPMN.io.

4.1 Technical Background

Our implementation is built up on the web-based BPMN modeling tool BPMN.io [15] like further described in Section 2.2.1. For this purpose, we set up an Angular [20] web application and integrated the rendering tool bpmn-js. Our application uses components which are associated with templates and services. Components, services and routing modules are written in TypeScript (version 3.4.1). Components implement the application data and logic of our concepts through interacting with bpmn-js. The associated templates are written in HTML and determine the represenation of content on the webpage. Services contain data structs that has to be shared across components. The application is developped on a desktop PC running on Windows 10 and the Mozilla Firefox web browser (version 65.0.2).

4.2 Graphical User Interface

In this work we implemented several functions to provide different view-building mechanisms. In order to integrate them in the modeling tool provided by BPMN.io, we built up a graphical user interface based on four components (cf. Figure 4.1). The App Component is the root and contains all other components. It determines the structure of the web page as described by the router. The Full View Component consists of a slide menu and enables the toggling between different process views. It is the parent component and therefore responsible for the navigation between the other two components. The Drawing Area Component includes the toolbar, the modeling toolbox and the canvas. This component creates a new modeler object

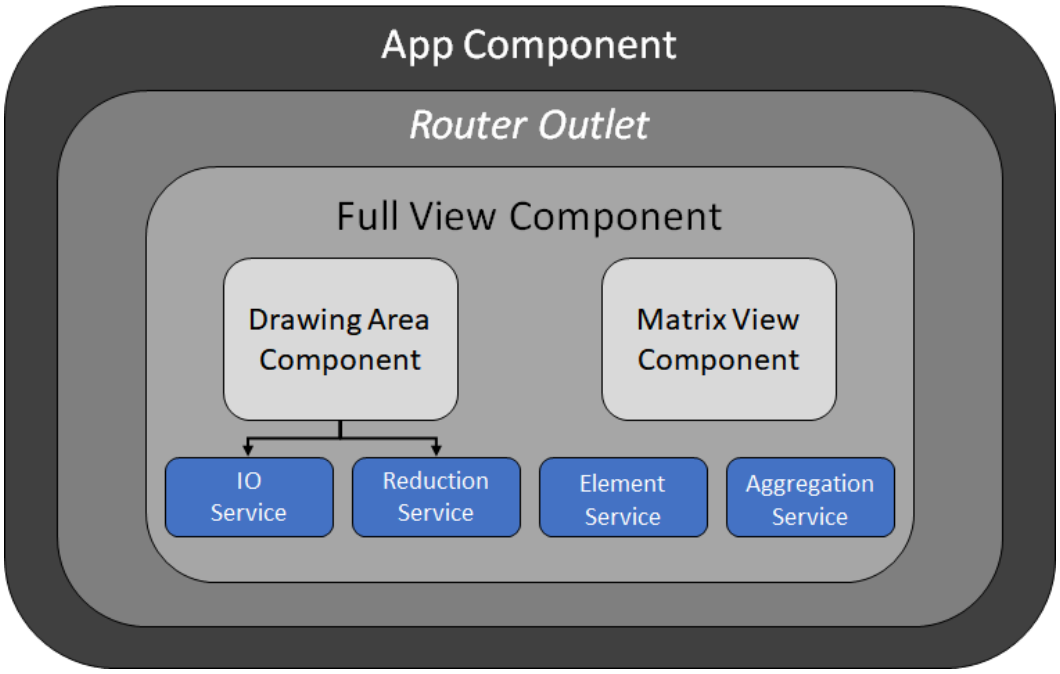


Fig. 4.1: The programming structure of our implementation: The GUI is built up on four components accessing to four services.

and controls important core services like the ElementRegistry. All changes in the model are managed by this component. The third component manages the matrix views, while each matrix view type is assigned to another URL path. We further implemented four services to provide an interface for importing data to and exporting data from the webpage (IO Service), enable graph reductions (Reduction Service) and aggregations (Aggregation Service), and include several auxiliary functions that deal with the elements in the graph, i.e. graph simplification mechanisms (Element Service). All services are accessed by the full view component, while the Drawing Area Component also requires the IO Service and Reduction Service.
The main page of the website is presented in Figure 4.2 and is structured into three parts as follows. While we took over the modeling toolbox and the canvas of BPMN.io, we added a toolbar (I) at the top of the webpage, a slide menu (II) which opens from the left side and is accessible by the menu button, as well as a notification area (III).

Toolbar

The toolbar contains seven buttons which provide different functions. The first two buttons manage the reduction and aggregation of any number of selected elements. They are disabled while no or only one element is selected. As alternative to the reduction button, any amount of elements can be removed by clicking the bin symbol in the element context menu or using the referring keyboard shortcut. Clicking the aggregation button enables the user to choose between an aggregation of the selected elements while preserving dependencies or to allow the generation of dependencies.

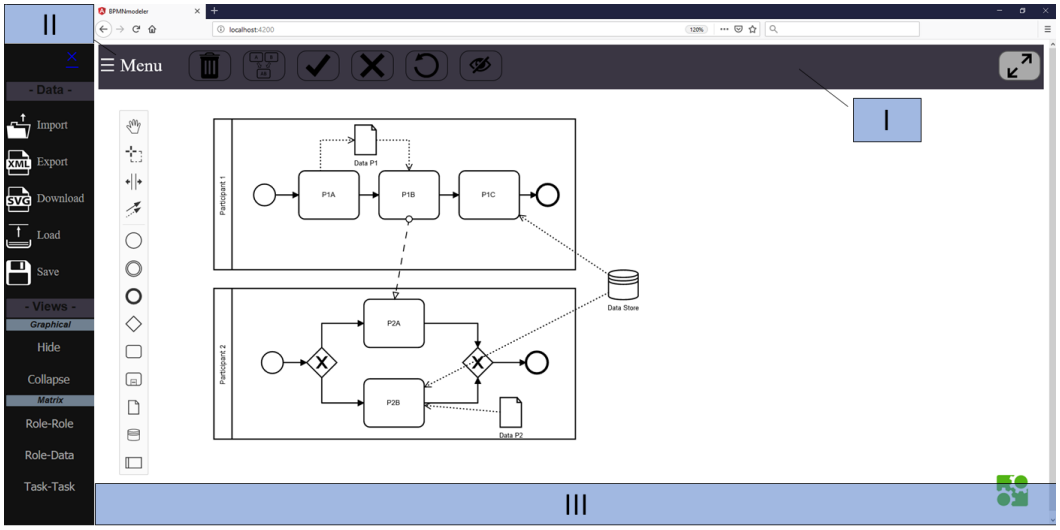


Fig. 4.2: The graphical user interface of the start page with an exemplary model and its three extensions: the toolbar (I), the expanded slide menu (II) and the notification area (III).

The underlying concepts of the reduction and aggregation buttons are described in Section 3, while further implementation details are given in Sections 4.3.1. The next two buttons provide the option to confirm or decline the chosen aggregation function on second thought. The fifth button enables the user to reverse the last reduction or aggregation operation. The sixth button manages the visibility of the text annotations of activities which contains former activity names. The button on the top right of the web page toggles a zoom out function leading to a full view of the diagram.

Slide Menu

The slide menu manages the input and output of data and provides different representation forms of process views. The first two buttons in the data category permit the import and export of process models as XML file from or to the explorer. The third button downloads the diagram as SVG file to the explorer. The other two buttons provide a quick save and quick load of the actual process diagram and serve as temporary storage. The second category enables the user to get two types of process views, which can be understood as different perspectives on the process model: a parameterizable graphical view and a matrix-structured view. Their functionalities and implementations are further described in Section 4.3. In contrast to the two graphical views, the three matrix views are realized on own web pages while the navigation between them is managed by the slide menu. Figure 4.3 shows the webpage of a matrix view for the exemplary model. At the top of the page, the element properties ID, name and type can be switched on and off to determine the level of detail.

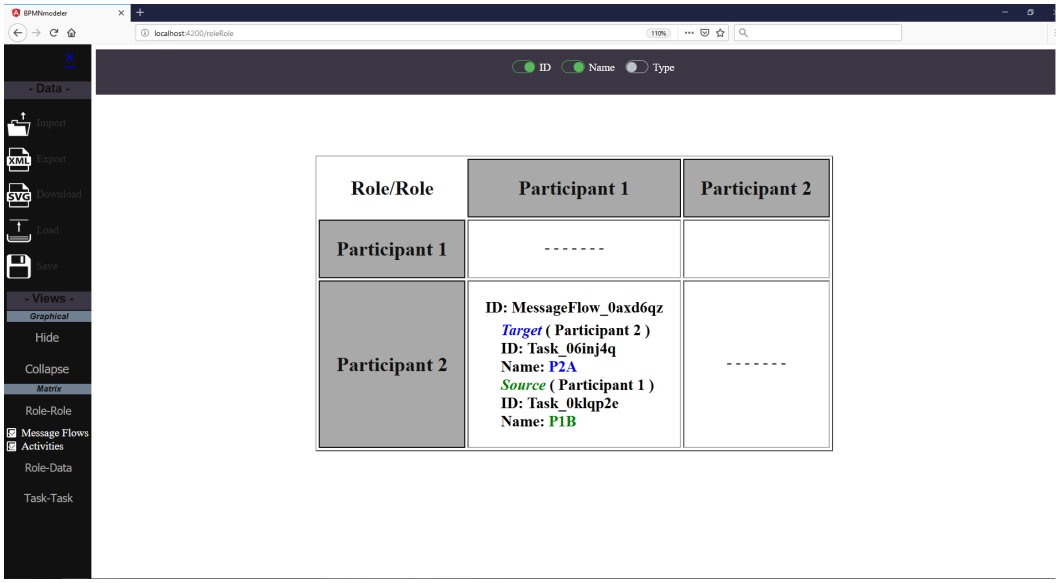


Fig. 4.3: Matrix view referring to the exemplary model in Figure 4.2.

```
modeler;
ngOnInit() {
   this.modeler = new BpmnModeler({
     container: '#js-canvas',
     height: 700,
     keyboard: {
       bindTo: window
     }
   });
}
```

(a) Creation of a new bpmn modeler object.

```
getModeler() {
  this.modeling = this.modeler.get('modeling');
  this.elementRegistry = this.modeler.get('elementRegistry');
  this.graphicsFactory = this.modeler.get('graphicsFactory');
}
```

(b) Access to the core and helper services

Fig. 4.4: Code snippets which describe the initialization procedure and the access to BPMN.io services.

Notification Area

The space at the bottom of the main page is used as notification area to inform the user about actions that will not longer be possible after aggregation on the bottom of the web page. Furthermore, it enables the user to rename the activities that result after the aggregation procedure.

4.3 Realization of Concepts

To realize the concepts presented in Section 3, we reutilize internal functions and modules of BPMN.io. In an initial step, a new bpmn modeler object has to be created like visualized in Figure 4.4(a). This object is the root of the modeling tool and manages the model on the canvas as well as all interactions with it. Like already described in Section 2.2.2, BPMN.io provides access to the model elements

by the ElementRegistry, to their graphical representation by the GraphicsFactory and enables actions like adding and deleting elements with the helper service Modeling. Since we need all of this services in our implementation, we request them from the modeler object like depicted in Figure 4.4(b). We use the functions provided by the respective services as listed:

elementRegistry.filter(condition): returns an array of the elements that fullfill the condition.

elementRegistry.get(id): returns the element corresponding to a given id leading to the possibility to request attributes like incoming or outgoing edges, the name or the parent element.

elementRegistry.getAll(): returns a list of all model elements

elementRegistry.getGraphics(element, boolean): returns the graphical SVG representation of a given element.

graphicsFactory.update(type, element, gfx): updates the visualization of an element according to the given gfx.

modeling.updateProperties(element, property: value): updates a property of an element with a given value.

modeling.appendShape(source, type, position, target): adds a new element to the model according to the parameters.

modeling.createConnection(source, target, type, parent): adds a new connection to the model, connecting source and target.

modeling.connect(element1, element2, attrs, hints): connects element1 with element2.

modeling.reconnectStart(connection, element, position): sets element as source of a connection at a given position.

modeling.reconnectEnd(connection, element, position): sets element as target of a connection at a given position.

modeling.removeConnection(connection): deletes a connection from the model.

modeling.removeShape(element): deletes an element from the model.

Since all services are connected to each other by the modeler, they are automatically kept up to date if changes occur in the model. In general, the presented functions according to the ElementRegistry can be understood as requests to get insights to an element, while the functions referring to the Modeling service perform persistent changes on the model. The latter ones have further to be distinguished between functions that effect the number of elements in the model (i.e. appendShape or removeShape) and those that change attributes of elements but preserve the number of elements (i.e. updateProperties or reconnectStart). According to this we differentiate between view-building mechanisms that lead to persistent changes in the model and views that can be understood as different perspectives on the model without effecting its underlying structure. Furthermore, we distinguish between the presentation form of process views in diagram-based and matrix-based. These four dimensions lead to a classification of our implementation like shown in Tabular

		Complexity Reduction	
		non-persistent	**persistent**
Presentation Form	**Diagram based**	Hide Collapse	Aggregation Reduction
	Matrix based	Role-Role Role-Data Task-Task	

Tab. 4.1: Classification of the implemented view-building concepts depending on the presentation form and the type of complexity reduction.

4.1. In terms the user changes the process model by adding, removing or changing model elements or by executing persistent process view mechanisms, all services are automatically updated through the bpmn modeler object they are connected with. This effect transfers to the provided process views which access all model elements by the ElementRegistry. This furthermore ensures that all process views stay up-to-date and no inconsistency occurs between the views.
In the following sections, we show how we implemented the different views by utilizing the provided functions of BPMN.io by differentiating between diagram-based views and matrix-based views. Like further explained in Section 3, we focus on processes which involve multiple participants modelled as pools.

4.3.1 Diagram-based Views

Hide and Collapse

The view-building functions hide and collapse enable the user to temporarily reduce the complexity of a process model. For this purpose, we retrieve a list of all modelled participants by the ElementRegistry and provide check boxes for each entry. All check boxes are marked as selected by default, showing the diagram with all its details on the canvas. The user can hide or collapse any amount of participants by deselecting the referring check boxes under the tag *hide* or *collapse*. Since both approaches are graphical representations, modeling is allowed while one of the views is active.

Hide
If a user deselects a check box under this tag, the pool that refers to the deselected participant and all its elements disappear. For this purpose we implemented the method setVisibilityOfElements to manipulate the visibility of a given element by using the update function of the GraphicsFactory like shown in Figure 4.5(a). For all model elements within a pool, the parent value of each element refers to the same participant. Due to this, we can easily identify all modeling elements of a pool by using the ElementRegistry and manipulate their visibility like presented in

```
setVisibilityOfElement(graphicsFactory: any, elementRegistry: any, element: any, checked: boolean)
 if (checked === false) {
   element.hidden = true;
 } else {
   element.hidden = false;
 }
  const type = element.waypoints ? 'connection' : 'shape';
  const gfx = elementRegistry.getGraphics(element, false);
  graphicsFactory.update(type, element, gfx);
}
```

(a) Elements are made invisible by setting their hidden attribute to true and updating the GraphicsFactory.

```
setVisibilityOfRole(graphicsFactory: any, elementRegistry: any, participantID: string, checked: boolean) {

  this.setVisibilityOfElement(graphicsFactory, elementRegistry, elementRegistry.get(participantID), checked);
  for (let c = 0; c < elementRegistry.get(participantID).children.length; c++) {
    this.setVisibilityOfElement(graphicsFactory, elementRegistry, elementRegistry.get(
      elementRegistry.get(participantID).children[c].id), checked);
      .
      .
      .
```

(b) A participant is hidden by hiding the element that refers to the pool and all of its child elements.

Fig. 4.5: Implementation of the hiding procedure of a pool.

Figure 4.5(b). Outgoing or incoming message flows that originate from or target activities of the hidden pool are made invisible in a postprocessing step. This also applies for data which is modelled wihin another pool but have an associations to an activiy of the considered pool. Since these connection types can refer to more than one participant, their parent attributes are related to the collaboration which is the container of all pools.

Collapse

The deselection of a check box under this tag leads to a full aggregation of a pool into one activity. In fact, there occurs no real aggregation in terms by using one of the aggregation operations decribed in the following section. Similar to the hiding procedure, the pool that refers to the deselected participant and all its elements are made invisible by calling setVisibilityOfRole for the respective participant (cf. code snippet 4.5(b)). In difference to it, we further add an temporarily existing activity to the model which represents the collapsed pool. Each message flow and association that interacts with the pool like described above, an additional, also temporarily existing, connection with the proper type is added to the model and the original connections are made invisible. In case of incoming edges, the new connection targets the newly created activity and inherits the source of the original connection. In case of outgoing edges, the target of the new connection is determined by the original connection and the new acticity is set as source. If multiple connections would be created, which have the same source and target, we avoid this redundancy by adding only one representative connection. We realized this procedure by using the functions appendShape and createConnection provided by the Modeling service. The temporarily existing elements are deleted from the model, if the collapsed view is reversed or another view is selected. Figure 4.6 shows the result of this procedure.

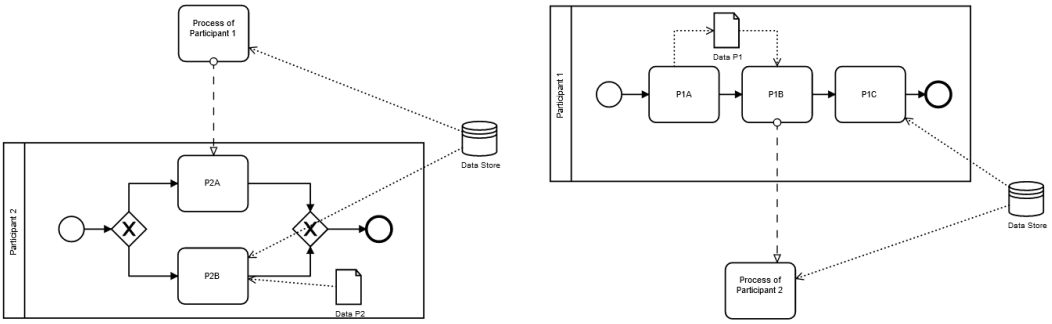


Fig. 4.6: Resulting process view after collapsing participant 1 (left) and participant 2 (right) of the model visualized in Figure 4.2.

```
createAggregation(modeling: any, elementRegistry: any, elements: Array<any>, reconnections: Array<ConnectionStruct>): any {
  const elementIDs = new Array<String>();
  for (let e = 0; e < elements.length; e++) {
    elementIDs.push(elements[e].id);
  }

  const newTask = modeling.appendShape(elementRegistry.get(elements[0].parent.id), {type: 'bpmn:Task'},
    {x: elements[0].x + 100, y: elements[0].y + 100}, elementRegistry.get(elements[0].parent.id));
  [...] // save names of the selected elements for the new text annotation
  this.eleS.reconnectEdges(modeling, elementRegistry, newTask.id, reconnections);

  for (let e = 0; e < elementIDs.length; e++) {
  [...] // remove text annotation of selected elements manually
    modeling.removeShape(elementRegistry.get(elementIDs[e]));
  }
  [...] // add text annotation to the new task and determine new content
  modeling.updateProperties(newTask, {name: 'aggregatedTask'});
  return newTask;
}
```

Fig. 4.7: The implementation of the aggregation procedure. Secondary code is omitted but commented.

To keep consistency, changes in the process model should only be made between and within fully expanded and not collapsed pools.

Aggregation and Reduction

Aggregation

Each aggregation operation presented in Section 3.2.1 aggregates a number of selected activities into one single activity. This can be realized through the following three steps. First, a new activity has to be created which represents the aggregation. Second, all necessary reconnections with the new task have to be performed. The reconnection of sequence flows depends on the aggregation operation, while associations and message flows that are connected with activities that will be aggregated have to be reconnected with the new task. In a last step, all activities that take part in the aggregation as well as remaining unnecessary edges have to be removed. We implemented these steps in the method createAggreation by using functions of the Modeler Service like shown in Figure 4.7. The method is used in the implementation of all aggregation operations AggrSequence, AggrSESE/AggrComplBranches and AggrShiftOut. The edges that have to be reconnected are passed as parameter and are calculated in a preprocessing step which may differ depending on the aggregation

```
aggregateSelectedElements(selectedElementIDs: Array<string>, option: string): boolean {
 [...]
 while (true) {
   while (aggrSequence || aggrBlocks) {
     aggrBlocks = this.aggS.trySubBlockAggregation(this.modeling, this.elementRegistry, filteredSelectedElements);
     this.eleS.graphSimplificatonAggregation(this.modeling, this.elementRegistry);
     aggrSequence = this.aggS.aggregatePairwise(this.modeling, this.elementRegistry, filteredSelectedElements);
     if (aggrSequence || aggrBlocks) {
       changesDone = true;
     }
   }
   if (option === 'DepPreserving') {
     break;
   }

   aggrShift = this.aggS.aggregateShiftOut(this.modeling, this.elementRegistry, filteredSelectedElements);
   this.eleS.graphSimplificatonAggregation(this.modeling, this.elementRegistry);
   if (!aggrShift) {
     break;
   } else {
     aggrSequence = true;
     aggrBlocks  = true;
     changesDone = true;
   }
 }
 [...]
```

Fig. 4.8: Code snippet of the while-loop in the method aggregateSelectedElements. The variable filteredSelectedElements contains only the activities of the selected set of elements.

operation. The names of the activities that will be replaced through the aggregation are stored in a text annotation associated with the new activity to keep the overview of model changes. The method createAggregation can be reused without adaption in the implementation of further aggregation concepts.

To guarantee the maximal possible aggregation of a selected set of activities, we recall the aggregation operations in a while-loop which is breaked if none of the operations can be applied anymore, or in other words, if no further changes occur in the model (cf. Figure 4.8). In the graphical user interface, the users are able to choose between a dependency preserving aggregation (AggrSequence, AggrSESE/AggrComplBranches or to allow dependency generating operations (AggrShiftOut) in addition. To realize these options, the dependency preserving aggregations are wrapped in a inner while-loop, because they are called in both cases. The outer while-loop is breaked immediately after the inner loop is breaked, if the first option is selected. In case further aggregation operations are implemented, they can be easily integrated in this procedure. If the new operation is dependency preserving or dependency generating, its function call can be added to the block containing function calls with the same dependency type. If the operation refers to a new dependency type, i.e. dependency erasing, a new case distinction has to be added similiarly to the dependency generating case.

Reduction

We implemented the three reduction operations to eliminate sequences(RedSequence), blocks (RedSESE) and branches of blocks (RedComplBranches) in one single method which covers all cases. Like described in Section 3.2.2, the reduction of elements have to comply certain rules. For example the incoming and outgoing sequence flows of the removed activity have to be reconnected or to be replaced, since the

process flow would be interrupted and other activities would lose their connection to start or end nodes. The removing function removeShape provided by the Modeler service meet these requirements in case of sequence flows becaue they automatically preserve the process structure and consistency of the model by default. For each activity in the selected activity set, we first remove all incoming and outgoing message flows and associations of it by using the function removeConnection. Subsequent we remove the respective activity from the model with the function removeShape. Then the toolkit automatically performs the necessary reconnections of the sequence flows reffering to the deleted activity. Furthermore, the function keeps all empty branchings possibly resulting through this procedure independent of the branching type. In the case of blocks caused by parallel branchings, this fact could lead to any number of redundant empty branches preventing the elimination of the corresponding gateways. For this reason we implement two graph simplification mechanisms. First we eliminate all redundant sequence flows by using the function removeConnection resulting in unique empty control edges between opening and closing gateways of any type. Second we remove all gateway from the model by using removeShape which have only one incoming and one outgoing sequence flow exlucing XOR/OR gateways. Therefore we guarantee a correct execution of RedSESE and RedComplBranches leading to well-defined process diagrams. Remark that our approach called by the button at the top of the web page is restricted to activities and only associated gateways or connections are deleted automatically. Executing the procedure on a selection that does not contain any activities would lead to no changes in the model. However, connections and other shapes, like gateways, text annotations or start and end nodes can be directly removed from the model through the respective context menu or toggling the delete key. The elimination of shapes by using one of this options could lead to interruptions in the control flow and should be used wisely. In contrast to our reduction approach, these built-in functions directly call removeShape or removeConnection. The application of removeShape on activities having incoming or outgoing message flows or associations deletes the activity and all of its referring connections from the model resulting in two not connected diagrams. The same applies for gateways having multiple outgoing or incoming sequence flows.

4.3.2 Matrix-based Views

In order to realize the matrix-based views we compute the view-related elements via the ElementRegistry and visualize them as table defined in the HTML template referring to the Matrix View Component. The structure of an examplary table is shown in Code Snippet 4.9. We implemented all three views like described in Section 3.3.2 according to this example. As you can take from the figure, the table is built up line by line while a row is introduced by the <tr> environment and the content of a

```
<table [...]>
  <thead>
  <tr>
    <th [...]>Role/Role</th>
    <th *ngFor = "let role of fullView.roles" [...]>{{role.name}}
    [...]
    </th>
  </tr>
  <tr *ngFor = "let role of fullView.roles; let outerIndex = index">
    <th [...]>{{role.name}}
   [...]
    </th>
    <th *ngFor = "let mf of role.messageFlows; let innerIndex = index">
      <div *ngIf="outerIndex == innerIndex">- - - - - - -</div>
      <div *ngFor = "let m of mf;let lastIndex = last">
       [...]
          <div *ngIf = "IDSelected" [...]> ID: {{m.id}}</div>
          <div *ngIf = "nameSelected && m.name != undefined" [...]> Name: {{m.name}}</div>
          <div *ngIf = "typeSelected" [...]> Type: {{m.type}}</div>
       [...]
      </div>
    </th>
  </tr>
  </thead>
</table>
```

Fig. 4.9: The creation of the Role-Role Matrix containing information about message flows between participants in the cells.

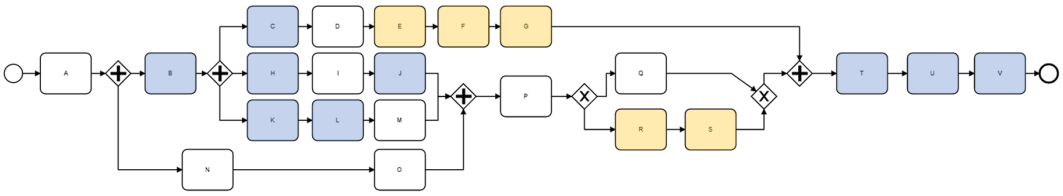
cell is contained in a <th> environment. Depending on which view is selected the first row of the table referring to the first <tr> environment contains the names of participants in the model in the case of Role-Role, the names of data elements in the case of Role-Data and the names of activities in the case Task-Task. The second <tr> environment is nested in a for-loop and therefore creates multiple rows. The first cell of each of this rows contains the name of a participant in the case of Role-Role and Role-Data or the name of an activity in the case Task-Task. These cells and the first row of the table build the matrix structure while the content of each other cell is related to exactly one entry in the first row and one entry in the first column. The other cells are filled with ids, names or type descriptions of model elements depending on which check box is selected in the slide menu. This can either be message flows or activities in the case of Role-Role, the information of input/output data or referring activities in the case of Role-Data or message flows and associated data elements in the case of Task-Task. Since we access the content which should be visualized in this cells by using a for-loop, the respective information has to be pushed in an array consistent to the order of the entries in the first row of the table. We compute this order for each matrix view in a preprocessing step.

5 Evaluation

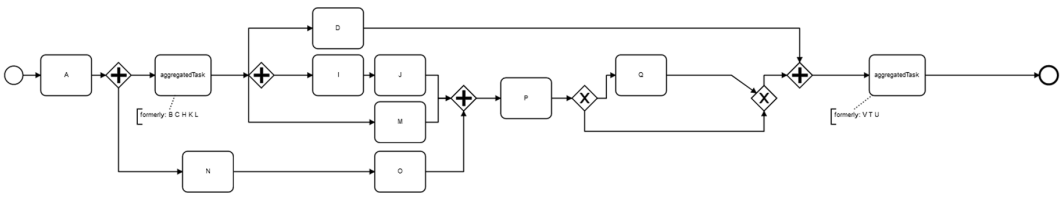
In this section we evaluate our implementation by presenting results of different experiments. For this purpose we applied our concepts to manually created process models and real-life process models. Furthermore, we present results of runtime measurements evaluated by different parameters.

5.1 Artificial Models

Since we implemented the concepts of [6] and [36], it is reasonable to test the abstraction mechanisms on the non-block-structured process model which is used as example in their work over and over again (cf. Figure 2.15). We rebuilt the model on the webpage with the appropriate BPMN elements provided by BPMN.io and added a start and end event like depicted in Figure 5.1(a). According to the example, we reduce the activities E, F, G and R, S in a first step. Afterwards the activities B, C, H, K, as well as L, J and T, U, V are selected for aggregation. Figure 5.1(b) shows the result after applying the aggregation option which allows dependency generating aggregations. The reduction operation on the selected activities lead to the same result as in the work of [6]. The empty branch is preserved in the XOR-branching.



(a) The exemplary process model according to [6] and [36].



(b) The resulting process model after reducing the activities E, F, G, R, S and aggregating the activities B, C, H, K, L, J, T, U, V by allowing dependency generating aggregations.

Fig. 5.1: A process model (top) and the result (bottom) after applying the abstraction mechanisms on a selected set of activities. The set of activities highlighted in blue are aggregated, while the activities highlighted in yellow are reduced.

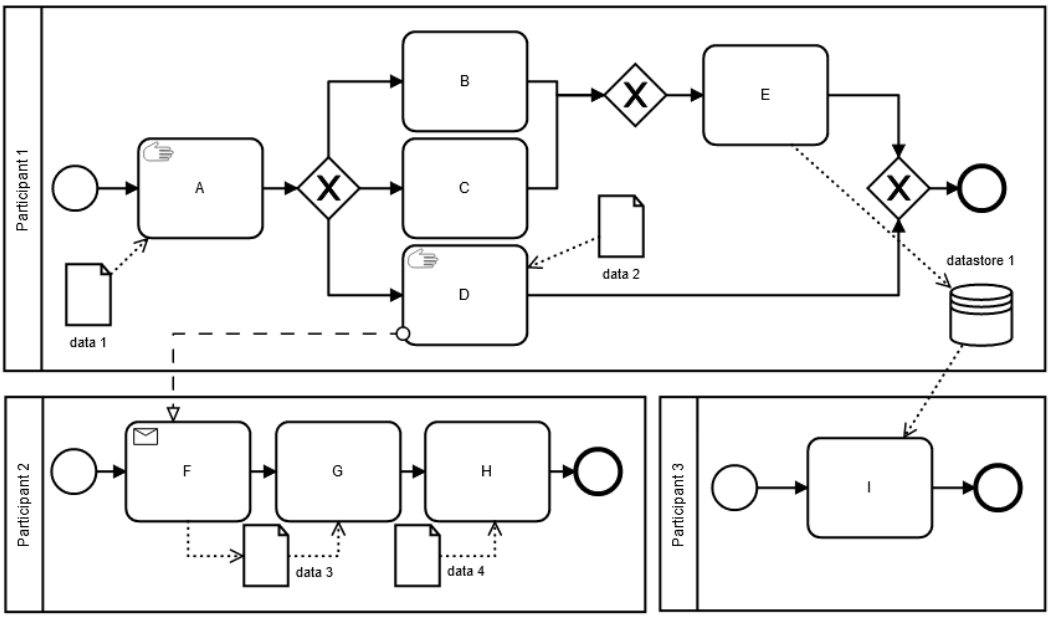


Fig. 5.2: A simple process diagram with multiple participants and a message flow between them.

In contrast to the work of [6], the aggregation differs in two points. First of all, the shift out that is applied on the activities C, H, K also affects the activity L. Second, the activity J stays unchanged in the whole procedure. The first result is due to the order of function calls in our implementation. As explained in Section 4.3.1, first all sequences are aggregated by executing AggrSequence recursively. This results in the aggregation of the activities K and L in a first step leading to a left shift out that consideres the activities C, H and KL. In a final step the activities B and $CHKL$ are aggregated. The result in [6] can be reached by performing a step-wise aggregation. Aggregating the activities C, H, K by applying a shift out through selecting the dependency generating option and aggregating the activities B, CHK afterwards through selecting any option, would lead to the same result. In difference, the aggregation of activity J in the selected activity set is not possible by using one of the implemented aggregation operations. The authors of [6] aggregate the activities J and L by applying the function AggrAddBranch which is not implemented in our work. Due to this, our result presented in Figure 5.1(b) is correct as it performs the maximal possible aggregation of the selected activity set.

In the next example we consider process models with multiple participants. The process diagram in Figure 5.2 consists of three participants with different workflows. To further evaluate the aggregation operation, we select all process model elements. Figure 5.3 shows the process model after applying the aggregation concepts of our implementation. Since all pools, data elements and events are still visualized as before, the result demonstrates that only activities are considered in the aggregation process. Like expected, the result further shows that the aggregation of activities is only executed within pools. Sequence flows are manipulated, while message

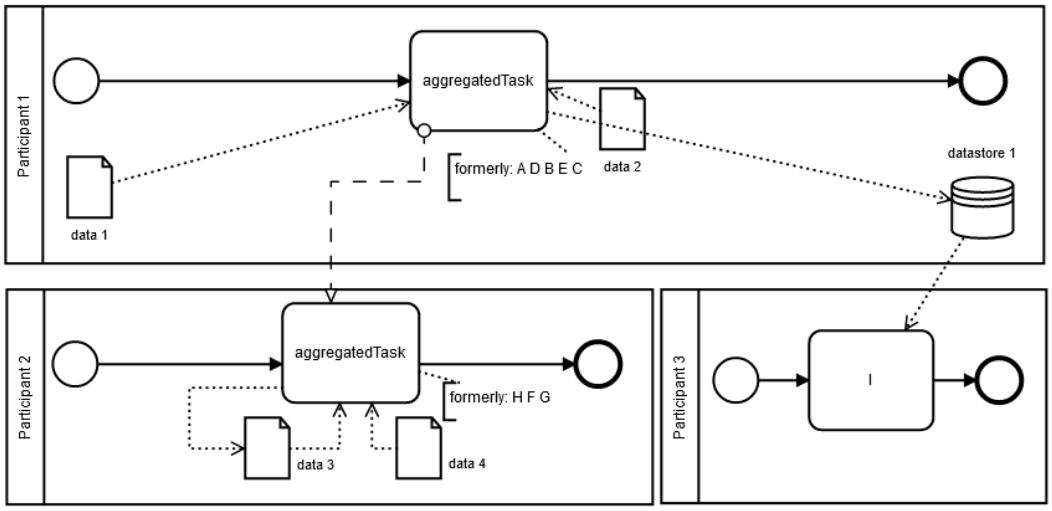


Fig. 5.3: The result after aggregating all activities of the process model in Figure 5.2.

flows and associations are preserved but reconnected with aggregated activities in a correct way. In some cases, self-loops may occur through aggregating activities. Since we assumed that process models have to be acyclic in this work, all derived views resulting after using abstraction mechanisms have to be acyclic aswell. Due to this assumption, we remove self-loops, i.e. sequence flows that have the same activity as target and source, that possibly occur after aggregation. This restriction does not apply for message flows and associations. For this reason the data object with the label data 3 is correctly associated as input and output of the aggregated activity HFG of Participant 2. Since the operation aggrShiftOut can not be applied to the selected activity set, both options of aggregation procedures lead to the same result.

This example further shows that the application of an aggregation operation leads to different types of information loss. In the case of aggregating a sequence of activities, the ordering of the referring activities gets lost. The aggregation of blocks leads to a loss of the gateway types which describe whether the branches in a block are alternatives or parallel processes. The information of the orderings of the activities which are included in a sequence can be easily kept by adapting the text annotation box which is created in our implementation if an aggregation is performed. Similarly, the gateway types of an aggregated block can be stored. In order to preserve the clarity of a created process views, we restrict to text annotations which keep the names of the activities that are eliminated by performing an aggregation procedure.

The scenario presented in Figure 5.4 is a process diagram which contains of four participants and a lot of message flows and associations. Although that model is not complex concerning its control flow, it is hard to get an overview of the interactions and dependencies between the participants and data elements. Especially the workflow of Participant 2 surrounded by the other participants is hard to understand. The moving of the pool in the viewer improves the result only marginally. In such

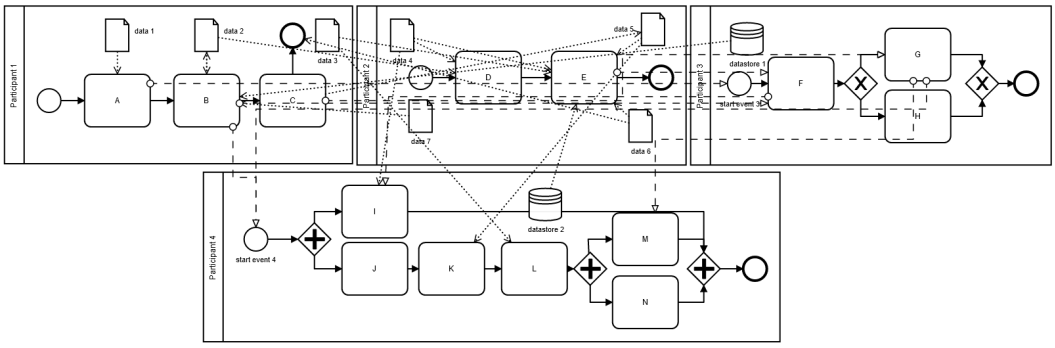


Fig. 5.4: A process model with four participants and nine data elements leading to confusing and crossing message flows and associations.

Role/Role	Participant 1	Participant 2	Participant 3
Participant 1	- - - - - - -		
Participant 2		- - - - - - -	
Participant 3	*Target* (Participant 3) Name: F *Source* (Participant 1) Name: B	*Target* (Participant 3) Name: F *Source* (Participant 2) Name: E	- - - - - - -
Participant 4	*Target* (Participant 4) Name: start event 4 *Source* (Participant 1) Name: B		*Target* (Participant 4) Name: I *Source* (Participant 3) Name: F

Fig. 5.5: The role-role matrix view of the process diagram in Figure 5.4 with focus on the activities that are involved by different participants. Due to lack of space we ommit the last column which is left empty referring to Participant 4 as source. The grey lines stand for further entries.

cases, the strengths of the matrix view are revealed. In contrast to the graphical views, the matrix view provides a structured overview of the interactions between the participants and the used data. As shown by Figure 5.5 Participant 2 has only one outgoing sequence flow that targets activity F of Participant 3. This information is hardly to get in the graphical view without using abstraction mechanism which lead to persistent changes in the process model. Another aspect concerns the data objects. The high number of associations that result from a high number of data elements are crossing the pools. Like already mentioned, their depositioning in the viewer to get better insights on which participant uses which data object is possible but time-consuming and the improvement is limited. The matrix view enables a compact representation of this information like shown in Figure 5.6.

Role/Data	data 2	data 1	data 3	data 7	data 4	data 6	data 5
Participant 1	Input/Output	Input		Input		Input	Output
Participant 2	Input		Input		Input	Input	Input
Participant 3							
Participant 4			Input		Input		Input

Fig. 5.6: The role-data matrix view of the process diagram in Figure 5.4 with focus on the data objects that can be the input or output of activities of participants.

5.2 Real-life Models

A real-life process model is given in Figure 5.7. It describes the interaction between the participants customer and company and visualizes the process steps that are necessary to accomplish a customer's request. The process model contains three different subprocesses which are highlighted in different colors. The subprocess *Execute Order* is highlighted in two colors, since it contains another subprocess. An aggregation of the subprocesses would be possible by applying AggrSequence but would not be meaningful with regards to the evaluation of our implementation. For this reason we expand all subprocesses and color them according to their affiliation in the collapsed visualization as shown in Figure 5.8. We slightly modified the process model, i.e. added start events, so it meets our requirements. The light-blue highlighted activities correspond to a subprocess in the process *Execute Order* whose activities are colored dark-blue. The expanded business process model contains of three participants and the pool referring to the Company is divided into five lanes. The diagram shows how that a lot of participants are involved in a process

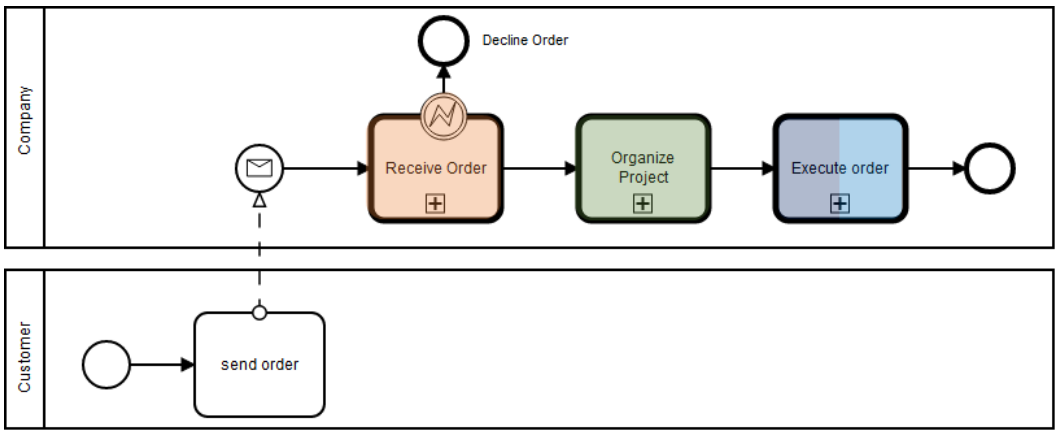


Fig. 5.7: Real-life process model describing a business process involving two participants.

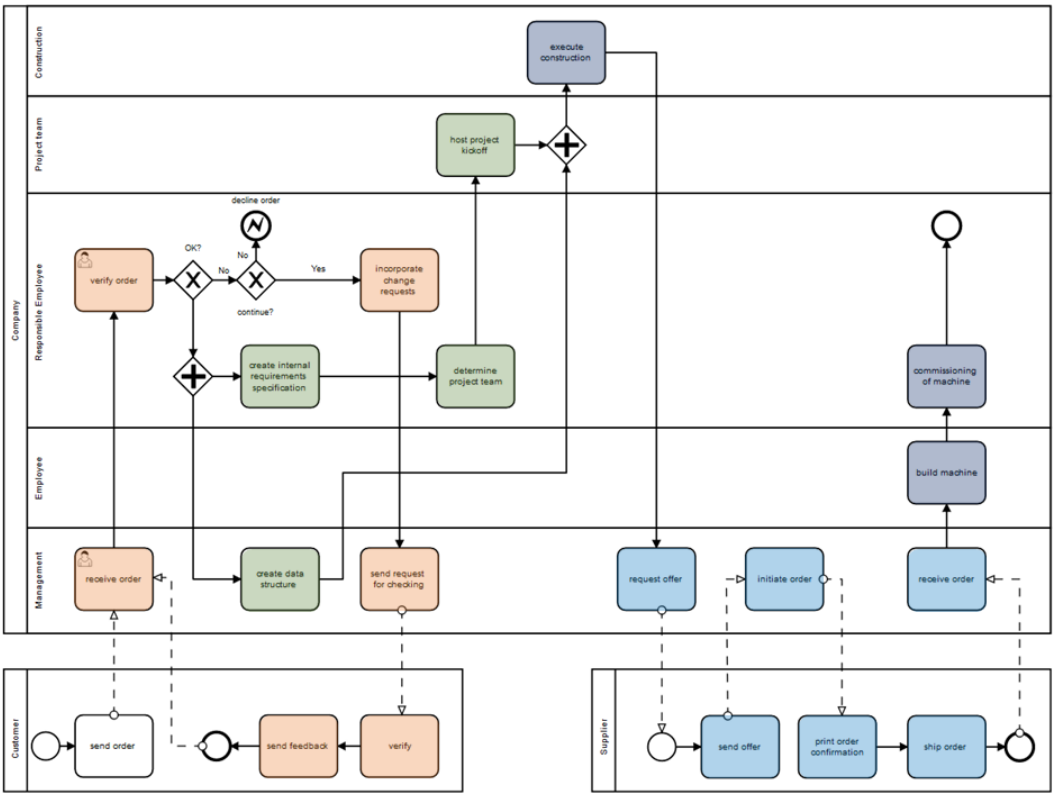


Fig. 5.8: Expanded version of the real-life process model shown in Figure 5.7. The activities can be assigned to the subprocesses in the collaped model by their color.

and that many steps are necessary between them to achieve a certain goal. From this perspective, the collapsed process model can be interpreted as process view using subprocesses to hide internal process details of the company and to make the process model shareable with other companies. Since our aggregation mechanisms can also be used to hide business secrets, we aggregate all elements referring to a subprocess. For this we perform a step-wise aggregation, by first selecting all activties of the subprocess *Receive Order* and aggregating them with the option dependency preserving. Both options would lead to the same result, since the aggregation operation shift out could not be applied. Then we repeat this procedure for the subprocesses *Organize Project* and *Execute Order*. In the latter case, we select all of the blue highlighted activities because they also represented by one subprocess in the collapsed process model. Figure 5.9 shows the resulting process model. While the activities of the subprocess *Organize Project* could be aggregated into one single activity, the activities of the other subprocesses are aggregated into three activities. Two activities of the subprocess *Execute Order* remain even unchanged in the process model, since the aggregation between activity that refer to different participants is not supported. In contrast, the aggregation between lanes of the same pool is possible, what leads to a loss of information. Consider for example the activity *Execute Order-1* which results from the aggregation of the activities *execute construction* and *request offer*. The latter was modelled within

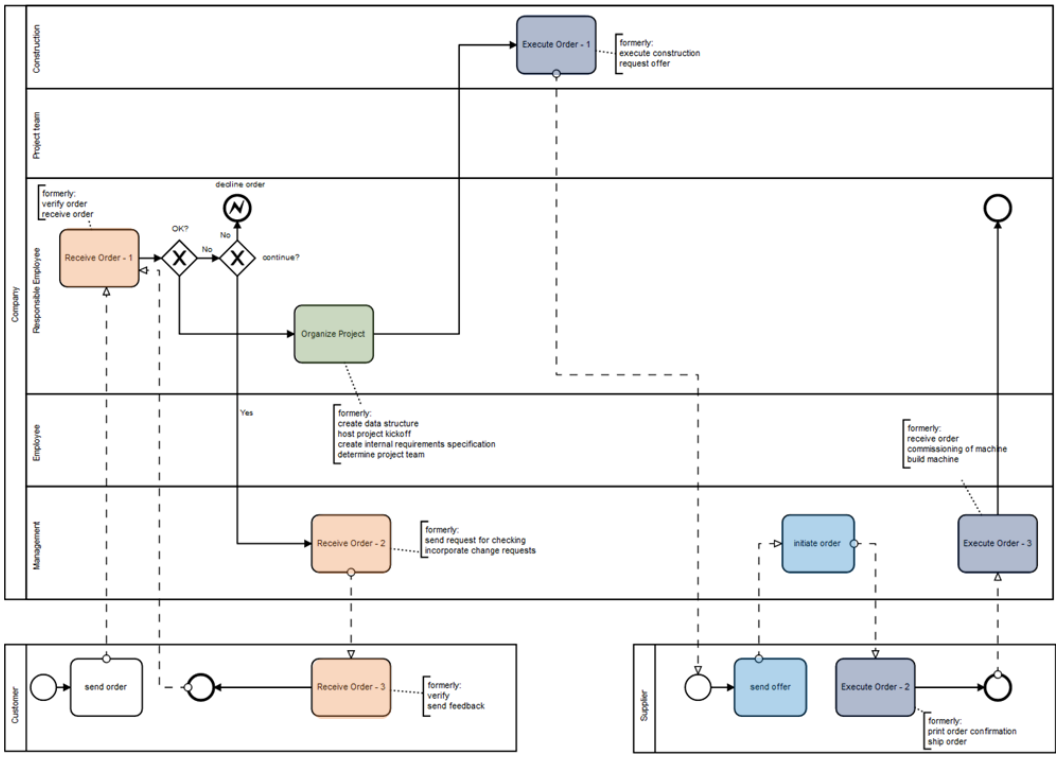


Fig. 5.9: The resulting process model after performing a step-wise aggregation of the activity set referring to the same subprocess in Figure 5.8.

the lane *Management*, while the aggregated activity is modelled within the lane *Construction*. After aggregation it is not possible to reconstruct which lanes were initially involved. This is explained by the internal structure of a process diagram in BPMN.io. The parent attribute of each activity refers to the corresponding pool instead of the corresponding lane. Due to this, lanes have no child elements and can only be used as graphical element to model more detailed diagrams. Associations between lanes and activities could only be reconstructed by the positioning of the elements in the diagram. For this reason we eliminate all lanes, leading to a more compact process model like shown in Figure 5.10. Compared to the collaped process diagram in Figure 5.7, the model consists of one additional paticipant, as well as six additional activities. In case of the activities of subprocess *Organize Project*, the aggregation leads to the same result as modeling them as collapsed subprocess. Allowing aggregations between different participants could lead to a result more comparable to the collapsed process diagram, but would also lead to the same information loss as discussed for the aggregation between lanes. In conclusion, our implemented aggregation operations can not be used instead of subprocesses but are a great support in terms of creating models which reveal any amount of detail.

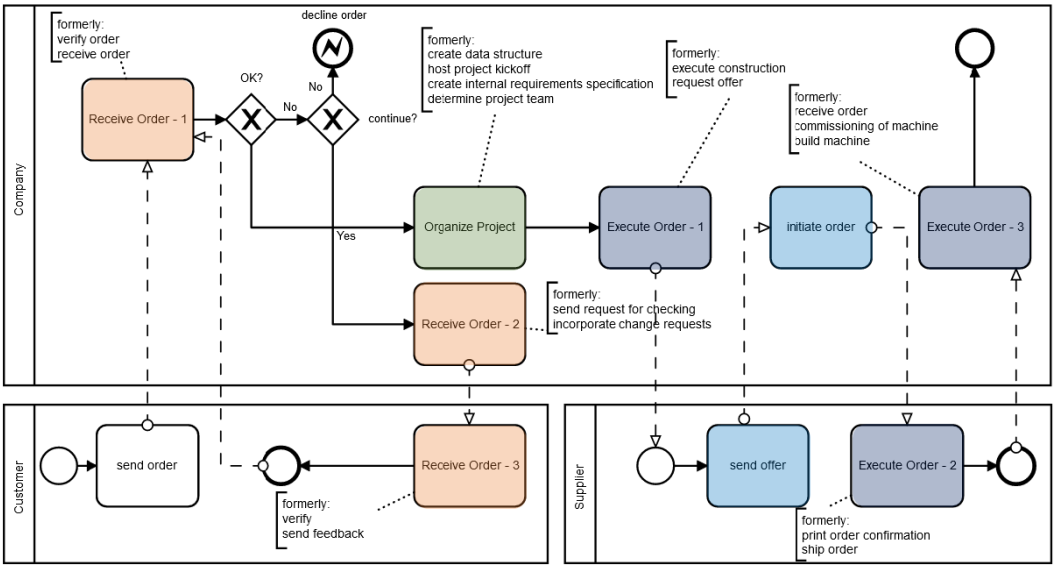


Fig. 5.10: The result after eliminating all lanes of the pool *Company* in the process model shown in Figure 5.9.

5.3 Runtime Measurements

The evaluation of the runtime of our aggregation procedure requires a large amount of (real-life) process models. Despite the increasing availability of such data, they do not fit our assumptions, e.g. acyclic process diagrams. Therefore we use the Processes and Logs Generator (PLG) provided by [8]. This software is designed to help researchers in the construction of a large set of processes and corresponding execution logs. The following parameters can be defined while generating a process diagram.

- ANDBranches: the maximum number of AND branches (must be > 1), default value: 5
- XORBranches: the maximum number of XOR branches (must be > 1), default value: 5
- loopWeight: the loop weight (must be in $[0, 1]$), default value: 0.1
- singleActivityWeight: the weight of single activity (must be in $[0, 1]$), default value: 0.2
- skipWeight: the weight of a skip (must be in $[0, 1]$), default value: 0.1
- sequenceWeight: the weight of sequence activity (must be in $[0, 1]$), default value: 0.7
- ANDWeight: the weight of AND split-join (must be in $[0, 1]$), default value 0.3
- XORWeight: the weight of XOR split-join (must be in $[0, 1]$), default 0.3
- maxDepth: the maximum network deep, default value 3
- dataObjectProbability: probability to generate data objects associated to sequences and events, default value 0.1

Detailed explanations of the parameters are given in the referred work. For our evaluation, we focus on the parameters ANDBranches, XORBranches, loopWeight, ANDWeight and XORWeight. The other parameters will keep their default value. Furthermore, we set the loopWeight to 0.0 for each generated graph, since our approach is limited to acyclic process diagrams. We generate three test sets with different parameters while each set consists of 1000 generated process diagrams. The first test set is created with the parameters $(5, 5, 0.0, 0.2, 0.1, 0.7, 0.3, 0.3, 3, 0.1)$ leading to diagrams containing 1 to 60 activities and 0 to 24 gateways. The second test set increases the number of branches by keeping the number of gateways and is based on the parameters $(15, 15, 0.0, 0.2, 0.1, 0.7, 0.3, 0.3, 3, 0.1)$ leading to diagrams containing 1 to 476 activities and 0 to 96 gateways. The increase of the maximum number of branches also leads to an increase of the number of gateways and activities. This follows from the fact, that each branch targets an activity and therefore additional activities have to be generated. Furthermore, a higher number of activities induces a higher probability for generating gateways. The third test set increases the number of gateways by keeping the number of branches of the first test set $(5, 5, 0.0, 0.2, 0.1, 0.7, 1.0, 1.0, 3, 0.1)$ leading to diagrams containing 1 to 80 activities and 0 to 36 gateways. We evaluate the runtime of the implemented aggregation procedure by performing a full aggregation on each diagram, i.e. selecting all diagram elements. Since the PLG generates only block-structured process diagrams, we focus on our dependency preserving approach. By restricting to the operations AggrSequence and AggrSESE, the result will always be a process diagram consisting of an start event which is connected to the aggregated task which is in turn connected to the end event. Like shown in 3.2.1, the execution order of AggrSequence and AggrSESE is irrelevant with regards to the resulting process diagram. Nevertheless the execution order can have an influence on the performance. For this reason we run all tests for both cases. We use a desktop PC running on Windows 10 with an Intel(R) Core(TM) i5-2500 CPU @ 3.3 GHz, 8.0 GB of RAM and the Mozilla Firefox web browser (version 65.0.2). The results of our runtime measurements are shown in Tabular 5.1.

For all test sets the order of first performing AggrSESE and then AggrSequence lead to better results. The runtime of this order is about one-third faster for the first and the third test set than interchanging the two operations. For the second test set this order is about three times faster. This result can be explained on the basis of the number of gateways. Since our implemented algorithm referring to AggrSESE consideres pairs of opening and closing gateways, its runtime depends on the number of gateways in the process model. The second test set consists of five times more activities than gateways. For this reason, AggrSESE has to consider significant less pairs than AggrSequence. Furthermore, the aggregation of blocks leads to a decrease of the number of activities in the model. AggrSESE is able to aggregate blocks containing nested blocks resulting in a strong reduction of the number of modeling elements. This means in turn, that AggrSequence need to consider less combinations

Runtime	Test set 1 #Activities: 1-60 #Gateways: 0-24	Test set 2 #Activities: 1-476 #Gateways: 0-96	Test set 3 #Activities: 1-80 #Gateways: 0-36
AggrSequence AggrSESE	497.03 s SQ: 58.5 % SE: 41.5 %	15426.41 s SQ: 65.2 % SE: 34,8 %	958.75 s SQ: 44.8 % SE: 55.2 %
AggrSESE AggrSequence	366.77 s SQ: 36.4 % SE: 63.6 %	4954.72 s SQ: 8.6 % SE: 91.4 %	613.86 s SQ: 23.0 % SE: 77.0 %

Tab. 5.1: The computation times of the aggregation mechanisms executed on the three test sets. Besides the full runtime, we present the percentages of AggrSequence (SQ) and AggrSESE (SE).

of activities. The percentages show, that AggrSESE computes the main part of the aggregation (91.4%) resulting in an elimination of a lot of activities so that AggrSequence only constitutes a small part of the runtime (8.6%). For the first and third test set, executing AggrSESE first in the procedure similiarly leads to a shorter computation time but in contrast having a higher pecentage for AggrSequence. This is due to the fact that the relation between the number of activities and the number of gateways is less than in the second test set.

Concluding we can state that the implemented execution order leads to correct and runtime efficient results in case of block-structured process models. Our results show that the performance mostly depends on the number of elements in such process models. The computation time of the aggregation of all activities in block-structured process models consisting of more than 500 elements needs on average less than 5 seconds. Although our measurements base on generated process models, our results also apply for real-life process models. This is due to the fact that the generated models as well as real-life process models always consist of a higher number of activities than number of blocks. A block enclosed by 2 gateways will always contain at least 2 activities, otherwise the split would not be necessary. An exception is made for blocks initiated by XOR-or OR-branchings. In this case one empty control flow edge may exist and therefore only 1 activity would be contained in the minimal case, leading to an equal number of blocks and activities. Since AggrSESE is dependency preserving, such blocks would not be aggregated by this operation. This could lead to an increase of the runtime of AggrSequence and would need further evaluation.

6 Conclusion

We summarize and discuss the results of this work and give suggestions for future work.

6.1 Summary

In this work we show how different and consistent process views of an underlying business process model can be created. For this we first give an overview of previous work which present mechanisms to create process views based on process models. Therefore we classified concepts of related work in parameterizable and non-parameterizable approaches (cf. Section 2.3). Since the latter ones does not meet the requirements formulated in Section 1.2, we focus on parameterizable process views. From these, we select view-building mechanisms that are not restricted to block-structured process models and adapt them to BPMN by preserving the syntax and semantic of this modeling language further presented in Section 2.2.1. For this purpose we define a process model in our context and determine necessary assumptions (cf. Section 3). We complement the abstraction mechanisms reduction and aggregation from related work with own approaches which enable hiding and collapsing pools in a BPMN model and provide a matrix-based view on the process. In Section 4 we describe the implementation of our concepts which is built on the web-based modeling tool BPMN.io presented in Section 2.2.2. We give insights into the graphical user interface and present implementation details which explain how the internal functions of BPMN.io are used and how they support our concepts. We show the functionality of our implementation regarding different case examples of artifical process models as well as a real-life process model (cf. Section 5). Furthermore, we evaluate the performance of our aggregation algorithm with a large number of generated process models.

6.2 Contribution

Since the Proviado [35] and Proview [37] projects are not supported anymore, there exists no working implementation of the abstraction mechanisms aggregation and reduction presented in [6] and [36]. As to the contribution of this work, we

provide an implementation of these concepts as well as the realization of further view-building techniques. In contrast to non-parameterizable approaches, the implemented abstraction mechanisms enable the creation of parameterizable process views, i.e. the user determines the set of elements which should be modified as well as the algorithm which should be used for the aggregation. The degree of abstraction can be adjusted according to the use case. Furthermore all of our concepts work for non-block-structured process models as well as for block-structured process models. Therefore we meet an important requirement of real-life BPMN models.
Since the implementation is built on BPMN.io, it can be easily extended with further functions regarding multiple dimension. Besides, BPMN.io is a modern and constantly updated tool ensuring our framework a long-term usability. The growing community continually develops new extensions leading to a wide scope for improvements and future work. The results of our evaluation show the correct functionality of our implementation regarding aggregation and confirm the determined execution order of the respective algorithms for block-structured process models.

6.3 Future Work

Future work should concentrate on extending the conceptual approaches as well as increasing the functionality of our implementation. The abstraction mechanisms in our implementation are restricted to the aggregation or reduction of activities. In contrast, the work of [25] also covers the aggregation of data objects. This approach should be implemented to extend our pool of functions of diagram-based and persistent process views. With regards to our aggregation operations, the authors of [6] and [36] provide further concepts which were excluded in our work in order to avoid inconsistent process views. The authors state that users accept minor inconsistencies in return of a higher model abstraction proved by several case studies. Inspired by this aspect, our implementation could be extended with these operations following our parameterizable approach.
Another aspect is the space of representation forms. In our work we cover diagram-based and matrix-based perspectives. Our current implementation enables three matrix views revealing interactions between a few model elements. However there are more aspects in a diagram that could be considered, e.g. the decisiveness or the depth of a pool until a certain activity is reached.
As illustrated in Table 4.1, this work enables persistent and non-persistent graph-based process views. In contrast, matrix-based views are restricted to non-persistent approaches. Since the matrix-based views can reveal details of the model that are hard to figure out in the graph like shown in Section 5.1, it could be helpful to also use this view for modeling to add for example further pools or to use persistent abstraction mechanisms. We suggest this point as future work in order to complete

our approaches.
In order to realize our algorithms, we made certain assumptions on process models like described in Section 3.1. These assumptions restrict the application of our implementation in case of real-life process models. For this reason, future work should concentrate on lowering them. For example the restriction to acyclic process models can be neglected by using graph transformations like suggested by [29] to transform cyclic to acyclic graphs or by extending our algorithms with appropriate stop criterions.
Overall we can state, that this work tackles two of the three fundamental process visualization dimensions proposed by [6]. Our approach enables the reduction of complexity by using the abstraction mechanisms aggregation and reduction or hide and collapse. Furthermore, we provide different presentation forms of a process by supporting matrix-based and graph-based views. Although our work does not cover the third dimension referring to the graphical appearance of process elements, this aspect can be easily realized in our implementation. BPMN.io supports different styles of process models as well as the creation of custom elements [11]. It further enables users to create custom overlays [13] or styling elements with colors [10] leading to an uncomplicated realization of the third dimension.
In general, future work should support users in constructing individual and compact process views which contain all needed information spread in the underlying business process model.

Bibliography

[1] Gustav Aagesen and John Krogstie. *Handbook on Business Process Management 2*. 2014, pp. 219–250. arXiv: arXiv:1011.1669v3 (cit. on p. 5).

[2] W.M.P. van der Aalst. *ProM: Process Mining*. 2016 (cit. on p. 4).

[3] Software AG. *ARIS Community*. https://www.ariscommunity.com/aris-express/bpmn-2-free-process-modeling-tool (cit. on p. 5).

[4] Ruth Sara Aguilar-Savén. „Business process modelling: Review and framework". In: *International Journal of Production Economics* 90.2 (2004), pp. 129–149. arXiv: arXiv:1504.02218v1 (cit. on p. 4).

[5] Ralph Bobrik. *Konfigurierbare Visualisierung komplexer Prozessmodelle*. 2008, p. 272 (cit. on pp. 19, 26).

[6] Ralph Bobrik, Manfred Reichert, and Thomas Bauer. „Parameterizable views for process visualization". In: *Information Systems* (2007), pp. 1–17 (cit. on pp. 2, 17, 18, 20–22, 24–27, 31, 33, 34, 48, 49, 58–60).

[7] Ralph Bobrik, Manfred Reichert, and Thomas Bauer. „Requirements for the visualization of system-spanning business processes". In: *Proceedings - International Workshop on Database and Expert Systems Applications, DEXA* 2006.August (2005), pp. 948–954 (cit. on p. 17).

[8] Andrea Burattin. „PLG2: Multiperspective Process Randomization with Online and Offline Simulations". In: *Online Proceedings of the BPM Demo Track 2016* (2016) (cit. on p. 55).

[9] Camunda and contributors. *BPMN modeler*. https://demo.bpmn.io/new. 2019 (cit. on pp. 10, 11).

[10] Camunda and contributors. *Color Elements in bpmn-js*. https://github.com/bpmn-io/bpmn-js-examples/tree/master/colors (cit. on p. 60).

[11] Camunda and contributors. *Custom Elements in bpmn-js*. https://github.com/bpmn-io/bpmn-js-examples/tree/master/custom-elements (cit. on p. 60).

[12] Camunda and contributors. *Examples of bmpn-js*. https://bpmn.io/toolkit/bpmn-js/examples/ (cit. on p. 9).

[13] Camunda and contributors. *Overlays in bpmn-js*. https://github.com/bpmn-io/bpmn-js-examples/tree/master/overlays (cit. on p. 60).

[14]Camunda and contributors. *Walkthrough of bpmn-js*. https://bpmn.io/toolkit/bpmn-js/walkthrough/ (cit. on pp. 9, 11, 13, 14).

[15]Camunda and contributors. *Web-based tooling for BPMN, DMN and CMMN*. https://bpmn.io/ (cit. on pp. 2, 5, 37).

[16]Dickson K.W. et al. Chiu. „Workflow view driven cross-organizational interoperability in a web-service environment". In: *Lecture Notes in Computer Science* 2512 (2002), pp. 41–56 (cit. on pp. 14, 15).

[17]Rik Eshuis and Paul Grefen. „Constructing customized process views". In: *Data and Knowledge Engineering* 64.2 (2008), pp. 419–438 (cit. on pp. 1, 2, 5, 16, 17, 19, 20, 22).

[18]Valeria Evgeneva. *Elma Blog: Lesson 6: Using Artifacts and Data Objects in BPMN*. http://www.elma-bpm.com/2016/06/27/lesson-6-using-artifacts-and-data-objects-in-bpmn/. 2016 (cit. on p. 8).

[19]Khaled Gaaloul, Rainer Schmidt, Selmin Nurcan, Sérgio Guerreiro, and Qin Ma. *Enterprise, Business-Process and Information Systems Modeling*. Stockholm: Springer, 2015 (cit. on p. 19).

[20]Google. *Angular*. https://angular.io/ (cit. on p. 37).

[21]Volker Hoyer, Eva Bucherer, and Florian Schnabel. „Collaborative e-business process modelling: Transforming private EPC to public BPMN business process models". In: *Lecture Notes in Computer Science* 4928 LNCS (2008), pp. 185–196 (cit. on p. 15).

[22]IEEE. „Recommended Practice for Architectural Description of Software Intensive Systems". In: *Technical Report IEEE-std-1471-2000* (2000) (cit. on p. 14).

[23]Lucid Software Inc. *BPMN Diagram Symbols & Notation*. https://www.lucidchart.com/pages/bpmn-symbols-explained (cit. on pp. 8, 9).

[24]Lucid Software Inc. *Lucidchart*. https://www.lucidchart.com/pages/de (cit. on p. 5).

[25]Jens Kolb. „Abstraction, Visualization, and Evolution of Process Models". PhD thesis. University of Ulm, 2015, p. 288 (cit. on pp. 14, 19, 22–24, 59).

[26]Jens Kolb and Manfred Reichert. „A flexible approach for abstracting and personalizing large business process models". In: *ACM SIGAPP Applied Computing Review* 13.1 (2013), pp. 6–18 (cit. on p. 2).

[27]Jochen Küster, Hagen Völzer, Cédric Favre, Moisés Castelo Branco, and Krzysztof Czarnecki. „Supporting different process views through a Shared Process Model". In: *Software and Systems Modeling* 15.4 (2016), pp. 1207–1233 (cit. on pp. 15, 16).

[28]Duen Ren Liu and Minxin Shen. „Workflow modeling for virtual processes: An order-preserving process-view approach". In: *Information Systems* 28.6 (2003), pp. 505–532 (cit. on p. 16).

[29]David E. Martin and David E. Martin. „Models of Computational Systems—Cyclic to Acyclic Graph Transformations". In: *IEEE Transactions on Electronic Computers* EC-16.1 (1967), pp. 70–79 (cit. on p. 60).

[30]NobleProg Training Materials. *BPMN 2.0 Artifacts*. https://training-course-material.com/training/BPMN_2.0_Artifacts. 2014 (cit. on p. 8).

[31]Robert B. McMaster and K. Stuart Shea. „Generalization in Digital Cartography“. In: *Resource Publication of the Association of American Geographers* (1992), p. 67 (cit. on p. 18).

[32]MID. *Innovator for Business Analysts*. https://www.mid.de/leistungen/tools/innovator/business-analysts?gclid=EAIaIQobChMI1ova1dfq4AIVD-J3Ch2ilw5XEAAYASAAEgLB0_D_BwE (cit. on p. 5).

[33]OMG. „Business Process Model and Notation 2.0 (BPMN)“. In: January (2011), p. 538. arXiv: `arXiv:1011.1669v3` (cit. on pp. 5, 9).

[34]Manfred Reichert. „Visualizing large business process models: Challenges, techniques, applications“. In: *Lecture Notes in Business Information Processing* 132 LNBIP (2012), pp. 725–736 (cit. on p. 1).

[35]Manfred Reichert and Ralph Bobrik. *Proviado - Visualizing Large Business Processes*. https://www.uni-ulm.de/in/iui-dbis/forschung/abgeschlossene-projekte/proviado/ (cit. on pp. 17, 19, 26, 58).

[36]Manfred Reichert, Jens Kolb, Ralph Bobrik, and Thomas Bauer. „Enabling personalized visualization of large business processes through parameterizable views“. In: *Proceedings of the 27th Annual ACM Symposium on Applied Computing - SAC '12* (2012), p. 1653 (cit. on pp. 2, 18–20, 24–27, 34, 48, 58, 59).

[37]Manfred Reichert, Jens Kolb, and Klaus Kammerer. *ProView - Personalized and Updatable Process Visualizations*. https://www.uni-ulm.de/in/iui-dbis/forschung/laufende-projekte/proview-project/ (cit. on pp. 17, 19, 58).

[38]Wasim Sadiq and Maria E. Orlowska. „Analyzing process models using graph reduction techniques“. In: *Information Systems* 25.2 (2000), pp. 117–134 (cit. on p. 16).

[39]Stefan Schönig. *Process Aware Information Systems: Process Mining [Slides for the Lecture]*. Bayreuth, 2018 (cit. on p. 5).

[40]S. et al. Smirnov. „Business process model abstraction: A definition, catalog, and survey“. In: *Distributed and Parallel Databases* 30.1 (2012), pp. 63–99 (cit. on pp. 14, 18, 19).

[41]Zyxware Technologies. *BPMN Flow Objects*. https://www.zyxware.com/categories/bpmn (cit. on pp. 6–8).

[42]Techopedia. *Business Process*. https://www.techopedia.com/definition/1168/business-process (cit. on p. 4).

[43]H Tran, U Zdun, and S Dustdar. „View-based and Model-driven Approach for Reducing the Development Complexity“. In: *in Process-Driven SOA. In: Intl. Working Conf. ...* (2007) (cit. on p. 14).

[44]Barbara Weber, Stefan Zugal, Jakob Pinggera, and Jan Mendling. „Imperative versus Declarative Process Modeling Languages: An Empirical Investigation – Experimental Material“. In: (2011) (cit. on p. 5).

[45]Mathias Weske. *Chapter 1: Introduction in Business Process Management: Concepts, Languages, Architectures*. Springer Science & Business Media, 2012, pp. 1–24 (cit. on p. 4).

[46]Stephen White. „Introduction to BPMN“. In: *BPTrends* (2004) (cit. on pp. 5, 6).

List of Figures

List of Tables

Declaration

I herewith declare that I have composed the present thesis myself and without use of any other than the cited sources. Sentences or parts of sentences quoted literally are marked as such; other references with regard to the statement and scope are indicated by full details of the publications concerned. The thesis in the same or similar form has not been submitted to any examination body and has not been published. This thesis was not yet, even in part, used in another examination or as a course performance. Furthermore I declare that the submitted written (bound) copies of the present thesis and the version submitted on the data carrier are consistent with each other in contents.

Bayreuth, April 24, 2019

Myriel Fichtner