Customizing an Open Source ERP System

Dan Björkgren, 40072
Master's Thesis in Computer Science
Supervisors: Annamari Soini, Marina Walden
Faculty of Science and Engineering
Åbo Akademi University
2020-06-17

Abstract:

At a point in time when the use of computers and software within companies and corporations is ubiquitous there is still reason to question their purpose and validity. The mere use of computers is not enough to enable efficient management of our time and work processes. Off-the-shelf software has a tendency to streamline organizations into facsimiles, leaving no room for developing and defining superior business processes. Refining business processes while figuring out what the true value of our endeavors is, be it in the manufacturing of goods or providing a service is, I believe, a path that is remarkably rewarding for both market and staff, as the cost of a commodity may begin to approach its perceived value. The business process becomes a true competitive advantage. This thesis explores the century-old, but often forgotten, models of business process refinement, combining them and the theory behind them with modern era business software into a system that is easily customized to fit any organization. A business process can be characterized as the description of how to complete a certain task. A process has a starting point and an end point with subtasks in between. Depicting the process in the form of a graph is a natural progression from that fact. In this thesis I create a software application for this purpose, storing the graph in a graph database. Since this type of graph has a natural path from starting point to end point it is possible to follow the graph programmatically from start to finish, in other words it is possible to use the graph as a form of source code, given that the correct parameters are available on the graph. Some tasks may involve decisions that are hard to program due to their circumstantial nature. In such cases the process must be halted and a notification be sent to whomever is responsible for allowing the process to continue. As part of the system a workflow engine *is developed that reads the graph and performs tasks as they are encountered. In the simple process given as an example in this thesis, the performed tasks are all functions in an open source ERP software, but the implied possibilities for automation are endless, as long as the task can be performed by a software programmable device. With this system I show that it is possible to automate and orchestrate an off-the-shelf software system in a way that is less limiting with regard to business processes than a standard system, without the enormous effort to create a custom system from scratch.*

Keywords: *Business Process, Enterprise Resource Planning, Workflow Management, Petri Net, Graph Database*

Contents

Abbreviations

API	Application Programming Interface
ASAP	Asyncronous Application Service Protocol (for SOAP)
BOM	Bill of Materials
BPEL	Business Process Execution Language
BPM	Business Process Management
BPML	Business Process Modeling Language
BPMN	Business Process Modeling Notation
BPR	Business Process Re-engineering
BSE	Bovin Spongiform Encefalopati
CORBA	Common Object Request Broker Architecture
CRM	Customer Relations Management
CSS	Cascading Style Sheets
ERP	Enterprise Resource Planning
GDB	Graph Database
GNU GPL	GNU General Public License
HTML	Hypertext Markup Language
IDL	Interface Definition Language
JSON	Javascript Object Notation
MES	Manufacturing Execution System
MRP	Material Resource Planning
MRP II	Manufacturing Resource Planning
OLE	Object Linking and Embedding
ORM	Object Relational Mapping
PCB	Printed Circuit Board
pdf	Portable Document Format
RDBMS	Relational Database Management System
REST	Representational State Transfer
SaaS	Software as a Service
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Name
VAT	Value Added Tax
WfMS	Workflow Management System
Wfd	Workflow Definition
WFSL	Web Services Flow Language
WSDL	Web Service Definition Language
XML	Extensible Markup Language
XPDL	XML Process Definition Language

1 Introduction

This thesis is a continuation of my Bachelor's thesis [41] which outlines my findings on the importance of letting software reflect business processes as opposed to forcing business processes onto a company by the choice of software. The main point of that thesis is that processes that have been worked out through pragmatic continuous improvements are to be regarded as a competitive advantage. Disregarding these processes when introducing new software leads to a nullification of these advantages, equalizing companies to the point where the only differentiator among subcontractors is price. It is not hard to imagine the downward spiral that follows. Business processes are discussed in Section 2.1.

Software is, however, essential in any business adventure at this point, in and of itself it is a competitive advantage over not using software at all. No-one has the appropriate resources available for assembling reports, tracking shipments and orders, let alone tracking manufacturing details or handling traceability, using handwritten notes. The process of manual recording is far too error prone, searching and compiling information far too slow to be realistic with today's workload. Any business that grows beyond one or two employees will have to implement a software system to handle these problems. As a first step, many a small business starts by using spreadsheets. Spreadsheets come with their own set of problems, the findings of van der Aa et al. [42] regarding process information also pertain to business information scattered around the computer network. One answer to this problem may be a well-defined *Workflow Management System*, *WfMS*. With a WfMS it is possible to tie different applications and systems together in a way that resembles the process flow at hand. Each business process is described, usually in the form of a graph, and stored in a format that the WfMS can use. The WfMS is responsible for enacting each application specified in the *process definition* in turn, sending notifications to users when user action is needed. Somehow this concept has never reached any overwhelming popularity. Many systems are said to incorporate workflows and the names of the components of the concept are well known to anyone with a little experience in business processes, but the large benefits and the flexibility of these systems have somehow been lost. Workflow Management is discussed in Section 2.2.

Disregarding the benefits of a true WfMS, at some point the only viable solution is a database or a system of interconnected databases that holds most, if not all, of the information that is collected during the daily operations of

the company, along with accompanying software to record, retrieve, edit, and compile said data. These systems are commonly known as Enterprise Resource Planning systems or ERP systems. Other, more specialized systems like e.g. Manufacturing Execution Systems (MES) may supply and retrieve data to and from the ERP system. As the name implies, ERP systems originate in the enterprise world where thousands of employees have access to the same data set and different departments may be spread over several continents.

The feature set of such a system is hardly the answer for a small entrepreneurial company with less than 20 employees. Over time, as the enterprise market is being saturated, the main manufacturers of these systems are expanding their market into the medium sized and small business segments with less features and, unfortunately, less options of customization. Most feature rich and customizable ERP-systems are also prohibitively expensive, in licensing, hosting, maintenance, and customization. The systems are usually sold preconfigured to conform to "best practice" business processes, leading to the fore-mentioned equalization of similar companies. [41] ERP is the topic of Section 2.3.

Open Source ERP systems are also available. As the source code is freely available, these can theoretically be customized by anyone with the right skill set. Some of these systems emerged in the late 90's with ongoing refinements and expanding feature sets and are actually comparable with the commercial solutions. Others are younger, but still rich enough in features to be viable in small companies. Common to all these systems is, again, the failure to be flexible enough when it comes to business processes. Other negative aspects include that necessary customization like translations and localized legal aspects, e.g. finance reporting style, can become extensive. Additionally, as recently expressed by Jessica Joy Kerr, via a tweet from Avdi Grimm - *"OSS: Your production software depends on a bunch of people's hobbies"* [43].

Due to the relative popularity of ERP systems, and the negative factors listed above and in Table 3, my conclusion is that there is a need for combining these rather disparate views on business processes or, rather, bringing the workflow aspect to the ERP domain in a more concrete manner. I have chosen to do this by amending an open source ERP system with a Workflow Management System and in the process creating a very flexible system that has the features of ERP, including the central database, and the business process adherence of a WfMS, including the graphical tools for defining the processes. Additionally, I will address a few customizations that have been made to the ERP system during my research at Comsel System and a few points of interest arisen from my earlier work at Oy Maxel Ab.

Making a production-ready system capable of handling any workflow is out of scope for this thesis. I will, however, create a rough user interface for creating and editing process graphs, using open source software when possible. I will also create a *workflow engine* that can be utilized to enable the flow of any *case* from start to finish, calling the appropriate ERP API functions in the order needed, automating the flow whenever possible, and sending notifications to the right user whenever needed. The practical work is described starting in Section 5.

The goal of this effort is to evaluate if the flexibility of a workflow-enabled ERP system will affect the overall performance of a business process. In Section 5.2.1 I describe a rudimentary but generally applicable business process encompassing an incoming sales order, checking stock level, deciding on manufacture or purchase if not immediately available for delivery, replenishing the warehouse if needed through to delivery of the items sold. Section 5.2.2 describes a manufacturing workflow triggered by an external application. Two different measurements are then applied comparing the ERP system "out-of-the-box" to the "workflow-enabled" system. Firstly, I am interested in the amount of effort that is needed to enact the example workflow using each variant of the system. I theorize that by measuring the time needed to perform the example workflow from start to finish we should be able to depict the labor involved. Additionally, a comparison of how closely we are able to follow the specified process is of interest. Summarizing the deviation count is a basic measurement, however, some reasoning around the consequences of each deviation is likely useful in this context. Some deviations identified in the manufacturing workflow have also been remedied in the course of this exploration and a discussion is provided as an example of more elaborate tailoring of the chosen ERP system.

2 Enterprise Resource Planning Systems vs. real workflows in manufacturing

2.1 Business Processes

A business process describes the daily work in an organization. It defines what is to be done, the tasks, who should do what, the roles, and how things should be done, the rules, and procedures. Together this specification regulates the business. [14]

The maturity of business processes has been described by McCormack [1] as a linear path from *Ad-hoc*, where nothing is described and you do what must be done, to *Defined*, where the process has been identified and documented, further to *Linked*, where process management is deliberately handled as a strategic instrument, and finally to *Integrated* where process management is integral to the organization and traditional functions (i.e. sales, manufacturing, etc.) are losing their importance.

Over time several ways of categorizing processes have evolved, for example Medina-Mora et al. [15] distinguishes between material, information, and business processes. Material processes are concrete work done in the physical world, e.g. assembling a product. Information processes are automated or partially automated tasks handling information. Business processes describe these processes on a business level, answering the question what is to be done to satisfy a market need or a business contract.

Van der Aalst has another way of describing this, although the categories mainly remain the same. Processes can intuitively be divided as Primary, which are the producing processes, such as manufacturing or service processes, Secondary, or support processes like Human Resources, tooling and marketing, and finally Tertiary, or managerial processes, directing and supervising the other two, such as formulating preconditions and objectives. [2]

Defining the business processes in general will have positive implications on many aspects of the business. Without them, as in *Ad-hoc*, decisions are made on a whim, traceability, as to who has done what, is nearly impossible to achieve, and any form of improvement is out of the question since there is nothing to measure advancements against.

In the nineties, with emerging office automation and client server systems replacing mainframes and terminals, and with the promise of more affordable computerized office systems even for small entrepreneurial businesses, the notion of business process re-engineering (BPR) became a popular buzzword.

BPR was meant to reinvent the way business was done within organizations as the prospect of automating tasks was nearing realization. The idea was to ignore any current processes and reinvent the way to do work while introducing new technology with new possibilities.[16] Most re-engineering projects struggled with resistance to change and ineffective leadership according to Chamberlin. [17] From my own experience I could also add that resistance to change may have different reasons, but in a small business the following cartoon says it all:

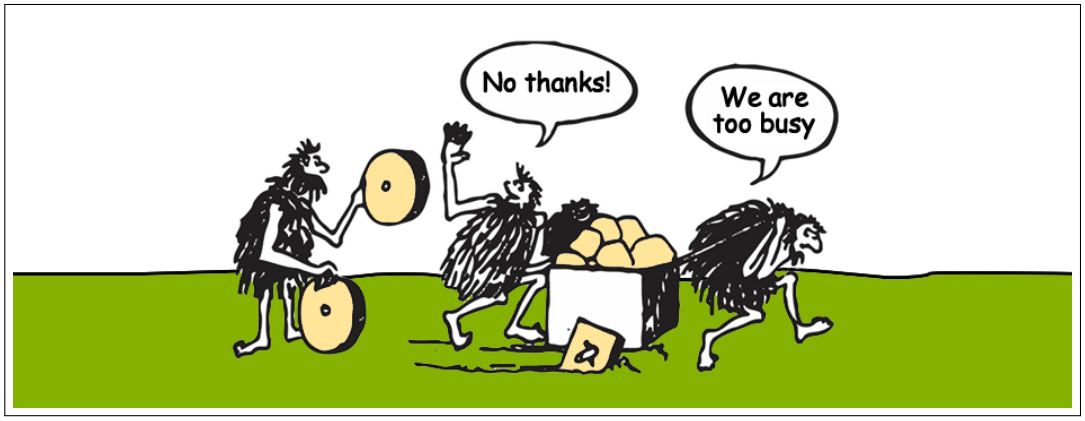

Figure 1: Too busy…[44]

In my point of view the BPR industry may have overemphasized the ”obliteration” part and in the process scared many smaller companies out of any improvements. The changes that Hammer describes in what could be thought of as the BPR manifesto [16] are not that radical. Improving the order handling instead of improving how to clean up the resulting mess seems to me like a natural evolution with the use of computers.

Since this radical BPR was largely a failure [18] a broader and more inclusive approach emerged, e.g. Zairi [19] with a focus on customer satisfaction, reliance on current processes, and the importance of quality systems and documentation, along with continuous improvements. Zairi writes about how to **manage** processes as opposed to re-engineering them and the new movement became known as Business Process Management or *BPM*. BPM is to me an attractive approach since its primary goal according to Hung [20] is ”to improve business processes and so ensure that the critical activities affecting customer satisfaction are executed in the most efficient and effective manner”. Hung further states that ”In order to sustain a competitive advantage and so face the rapidly increasing global competition, companies must continuously implement best practice management principles, strategies and technologies”. This is in line with my earlier findings that business processes are a key factor in reaching competitive advantage.[41]

2.2 Workflow Management

Software systems in use in organizations are often a plethora of applications dedicated to a single or a limited number of activities, even if there is a central database in place, as with ERP systems. To be able to automate the flow of a business process, to create a *workflow*, one would have to describe the business process in a language suitable for a software system to be able to redirect documents and other digital artifacts between these applications in a timely manner. This is called a *process definition*. There will always be some actions that cannot be easily automated, such as the initial input of *case* data, amending *case* data, or fault handling. In many decision situations, only recommendations can be made automatically, while the final decision must be made by a human. This type of human interaction must be dealt with using e.g. notifications to alert the worker that the automated process needs human input. The software used for managing these *workflows* is called a *Workflow Management System*

Workflow Definition

> "The computerized facilitation or automation of a business process, in whole or part." [3]

Workflow Management System Definition

> "A system that completely defines, manages and executes "workflows" through the execution of software whose order of execution is driven by a computer representation of the workflow logic" [3]

Process Definition Definition

> "The computerized representation of a process that includes the manual definition and workflow definition" [3]

Case Definition

> "An instance of a process performed as detailed in a Process Definition. Examples include producing a single item, handling a single insurance claim."

The *Workflow Reference Model* [3] is the standard document developed by the *Workflow Management Coalition*[1] outlining the needed capabilities of

[1] " Founded in 1993, the Workflow Management Coalition (WfMC) is a global organization of adopters, developers, consultants, analysts, as well as university and research groups engaged in workflow and BPM." [45]

a Workflow Management System. The main characteristics are presented in Figure 2.

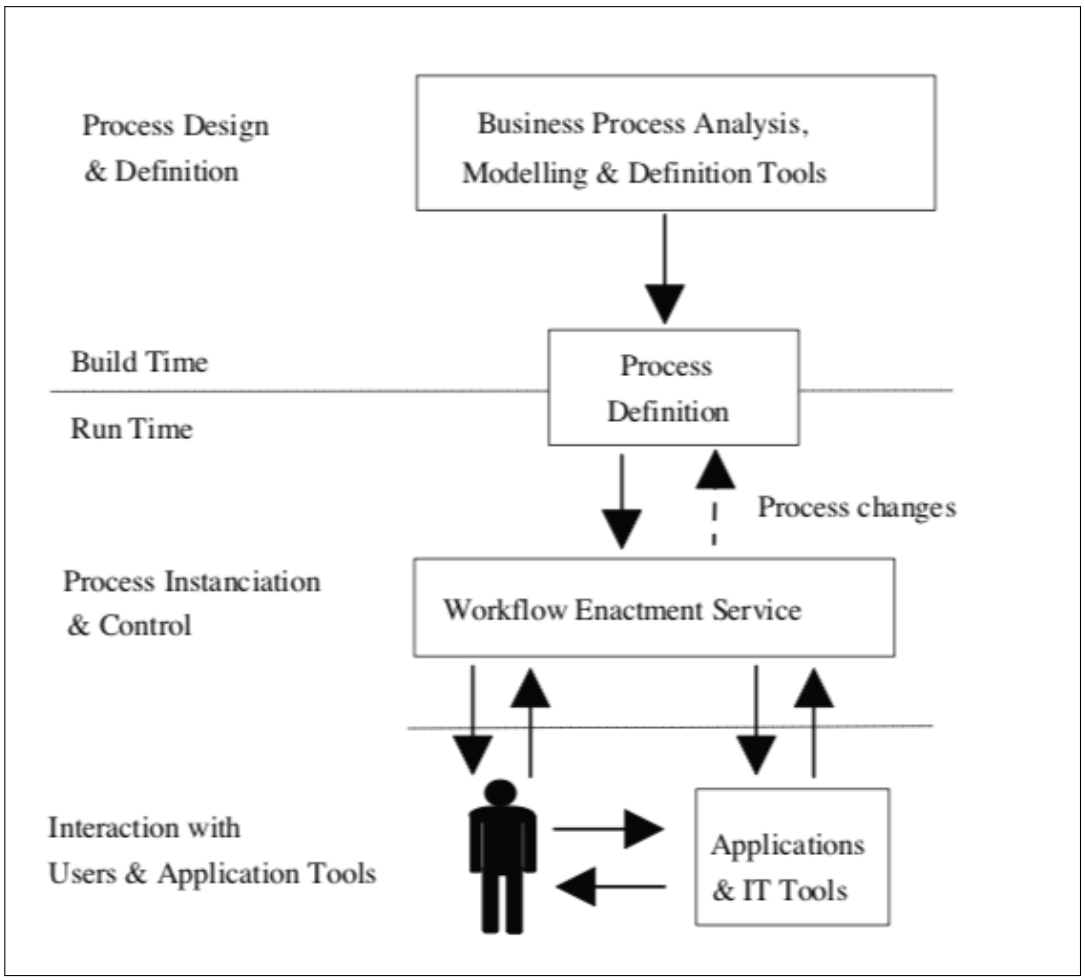

Figure 2: Workflow System Characteristics [3][*sic*]

In short, the workflow of the workflow management system itself includes creating the *process definition* using modeling and definition tools and running a workflow enactment service to manage and apply the workflows to applications and to interact with users.

There are five interfaces defined in or in relation to the Workflow Reference model [4]. We should note that the reference model was developed in the early- to mid-nineties when the World Wide Web was still in its infancy, but distributed computing using *CORBA* messaging was emerging. The fundamental aspects of the *API*s (Application Programming Interfaces) are still relevant, at least as a reference point.

The first interface, which is still relevant, is for exchanging business process definitions. It is defined in WFMC-TC-1025 [46] and has an accompanying XML schema definition XPDL.xsd. [5] The specification can be used for storing business process definitions in an interchangeable format.

The second interface was developed as a way for applications to be able to communicate with varying vendors' workflow engines. This was specified in C as several APIs for handling worklists, activities, and processes. It was later redefined in IDL for use with Corba and OLE.

Interface number three describes the *Invoked Application Functions* which are functions dealing with connecting to external (in respect to the workflow service) applications and invoking activity functions in the external applications. This can be done with or without what is called *Application Agents* (essentially proxy or connector applications). It has later been amalgamated into interface 2.

Interfaces two and three, while they are reasonable and allow for all functionality one would expect from a workflow enactment service, they have apparently remained a theoretical exercise. All workflow systems I have come in contact with define their own APIs.

The fourth interface defines functions for communication between workflow services. This has been revised several times and has also been defined in XML as Wf-XML with implementations over *SOAP*[2]and *ASAP*[3]. As it stands, this interface partly overlaps with interfaces 1 and 2. For example ListDefinitions of interface 4 and WMOpenProcessDefinitionsList of interface 1 in essence do the same thing, i.e. return a list of available process definitions.

The last defined interface is for systems administration purposes. No final implementation of this specification has been made public, however, there is an unnumbered draft of an "Audit Data Specification" available at the WFMC website. [48]

2.2.1 Creating the process definition

The Workflow Reference Model does not give specific advice on what tools to use for this step. Any way of defining a process is acceptable whether textual, graphical, or in a formal language.

Flowcharting, to graphically express the flow of the process, the *Workflow*, was already being done earlier on but became a more explicit part of BPM than it had been in BPR. Different but similar (since most of them were based on Frank and Lilian Gilbreth's work from the 1920s) types of graphs have been

[2]Simple Object Access Protocol is an XML-based transport protocol developed by Microsoft, enabling messaging and remote procedure calls over standard transports like HTTP [21].

[3]Asynchronous Service Access Protocol enables starting, monitoring and controlling web services over the SOAP protocol. It was developed by OASIS (non-profit worldwide consortium promoting open standards "for the global information society") in 2003 but was seemingly abandoned in working draft status in 2004. [47]

proposed as the solution to flowcharting processes. This is true for the *Activity Diagram*, which is part of the UML language, and for the *Business Process Modelling Notation, BPMN*, developed by the Business Process Management Initiative (later merged with the Object Management Group, OMG), endorsed by the Workflow Management Coalition and later standardized in ISO 19510, the latter of which is now the most commonly used diagram type within BPM. [6]

A basic BPMN chart describes a process by means of *Activities* represented as rounded rectangles, *Gateways* (corresponding to decisions in flowcharts) drawn as diamond shapes and *Flows*, directed arrows depicting the order of events. [6]. Finally, we find *Events* represented by circles. In addition, there are variations in border styles and fills with specific meaning and extended symbols that can be inserted into the shapes for more advanced use.

Activities and events are distinguished by time aspect. An event has zero time duration whereas an activity takes time.

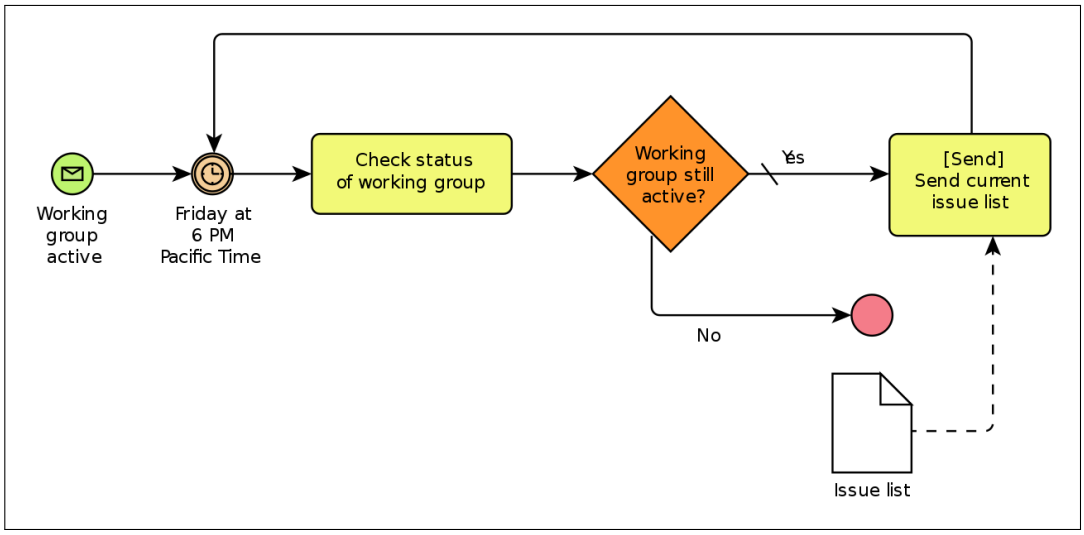

Figure 3: BPMN representation of a process

Unfortunately, BPMN is stateless [49], and lacks in formal semantics [49] which leads to problems verifying and simulating the resulting workflows.

One slightly different approach is the work of W.M.P. van der Aalst which is based on Petri nets [2], [22], [23], [50], [51]. The origin of Petri nets goes back to 1962, proposed by Carl Adam Petri (1926-2010). Petri nets are considered the first formalism to model concurrency. [24] Petri nets consist of two main components, *Places* denoted by circles and *Transitions* denoted by bars in the graphic notation. Formally these components are denoted by p and t respectively. Places and transitions are connected by directed arcs from places to transitions and from transitions to places. This static representa-

tion is amended with dynamic properties showing the execution of the model. Markers (called *tokens* and represented by dots) in the flowchart denote what function is executed at the moment and by moving the markers the net can be simulated. A transition cannot fire unless there is a token available in all preceding places. The transition fires by moving said token to the next place. [25]

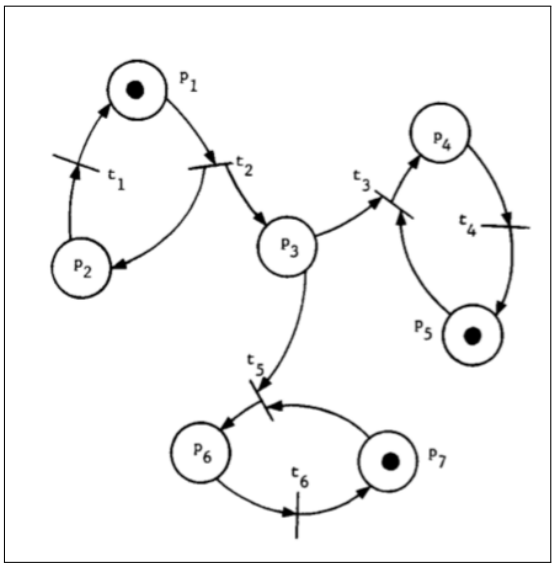

Figure 4: Petri net[25]

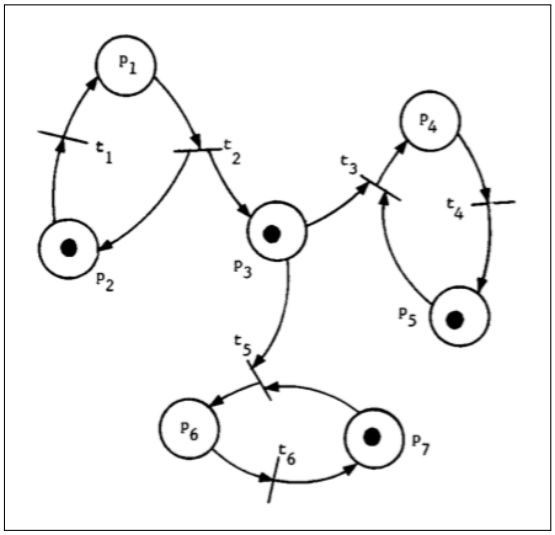

Figure 5: Petri net in the next state, token in p_1 has been moved to p_2 and is duplicated in p_3 [25]

Formally the basic Petri net is defined as a triple (P, T, F). [26] P is a finite set of *places*, T is a finite set of transitions ($P \cap T = \emptyset$), and $F \subseteq (P \times T) \cup (T \times P)$ is a set of *arcs* (flow relation) [26].

It is not possible to express real life workflows with the basic Petri net, however, according to Van Der Aalst, by extending the net with arc weight functions W, (activity) labels A, labeling functions L, reset arc functions R, and an inhibitor arc function H most workflow scenarios can be modelled. Further restricting the net to have a single source place i, a single sink place o, keeping all other nodes on the path from i to o, and prohibiting reset arcs from connecting to o gives us a WorkFlow net. [26] Introducing timing allows us to calculate duration.

Van Der Aalst has done extensive research in the field of workflows, and along with his colleagues he has identified a multitude of *patterns* [27] (cf. Alexander [7], *GoF* [8]) which have been used to identify how different modeling techniques are equipped to handle the modeling of workflows. A few of these patterns are hard to model using Petri nets, a fact that led to the development of YAWL (yet another workflow language). [23]

Following the Workflow Reference Model any process definition should be possible to convert into the XML process definition language defined in the process definition interface [46].

2.2.2 Workflow Enactment Service

The Workflow Enactment Service, as defined in the workflow reference model, is used to interpret the process model definition and direct each *case* through the workflow from start to finish. This includes communicating with applications, to start execution of activities, and communicating with users, essentially thereby controlling the state of each case. The core component in this service is called a *Workflow Engine* and as the reference model is described, several workflow engines can act upon the same workflow instance (case) and may also be distributed on several systems.

2.2.3 Conclusion regarding workflow management systems

A workflow management system could in the best of worlds be used to tie together all the small applications, including spreadsheet files and word processing documents, accounting software and special case applications used in an organization, into a single more or less coherent system to handle automation or semi-automation of a company's business processes. To my knowledge no such system has ever been marketed to small (1-50 employees) businesses and certainly not made popular. Reasons for this anecdotal fact are unknown but could make for an interesting field of study.

Although I have concentrated my efforts regarding workflow systems to the

WfMC Reference Model it is worth noticing that other competing models exist and, in fact, are more generally in use due to the size of the companies that endorse and implement them. For example, SAP, in cooperation with others, developed the BPML language for modeling and executing workflows. BPML, although its notation is XML-based, is based on pi-calculus[4] and thus has the ability to describe processes as first-class entities[5]. [28] OASIS, an organization including among others IBM, Microsoft, and Oracle, has developed the BPEL range of languages, aimed at orchestrating web services over WSDL as workflows. None of these have native graphical representations even though WFSL is graph-based. [52]

2.3 Enterprise Resource Planning Systems

Enterprise Resource Planning (ERP) systems is an umbrella term for software systems aiming to support and control the daily work at a company. They are typically a compilation of different modules handling customer relations, human resources, manufacturing, warehousing, logistics, and so on. All modules talk to a common large database to reduce redundancy and allow for effortless compilation of reports. [29]

These systems have evolved independently from the process oriented approach and their roots can be found in the development of inventory control applications of the early 1960s. [53] *Material Requirements Planning* (*MRP*) introduced the *Bill of Materials* (*BOM*), and time-phased planning to cope with the material flow on the shop floor [9], [53]. *Manufacturing Resource Planning* (*MRP II*), introducing capacity planning and master scheduling [53] evolved from MRP and as more and more areas of the enterprise, like accounting, human resources, and customer relations were added to the central database, ERP was born. Specifically transactional systems consisting of modules with their boundaries (separated by function rather than by organizational division), interfaces between modules (input and output extended by including data sets, such as *working files*, *master files*, *directories of information*, and *tables*), and the *database* as a "common corporate data bank", were defined by S.C. Blumenthal in 1969. [10] As similarities between different industries are obvious, the ERP systems were offered as "off-the-shelf" applications from

[4]Pi-calculus is a model of concurrent computation based on naming. It is a formal way of expressing concurrency using actors (processes, etc.) and interactions between these agents (e.g. messaging).

[5]A first class entity is any data type that can be used as a construct in the language in question e.g. it can be assigned to a variable, passed as a parameter, and so on.

companies such as IBM, SAP, and Oracle to name a few of the most renowned. Workflow logic and functional logic are traditionally embedded in the system itself, i.e. *hardcoded*, but a parameterized approach is being used to enable more flexible systems. *Parameters* are set as preconditions to e.g. user interfaces and control e.g. what fields are present and how the information is routed. These are normally set during the implementation of a system in an organization. Parameters can be of three types according to Soffer et al. [30]; high-level process definitions, the aforementioned preconditions to user-interface sessions, low-level process definitions, choosing algorithms and rules for each action, and value-based parameters that determine how an action is performed. Examples of the third type would include vastly different concepts like VAT rates, planning time spans, or how orders and invoices are numbered. To accommodate for different markets and different fields of business there are usually pre-defined templates for the setting of these parameters. Other means of customization, like programming efforts to change the behavior of the system, are mostly discouraged by vendors who may even refuse future support for packages that have been altered. Instead, implementing companies are encouraged to change their processes to suit the system, alternatively (but, of course, rarely pushed by vendors), to find a more suitable system.

There is logical reasoning behind this unwillingness to alter the software. Many implementations have failed, sometimes leading to the failure of the business itself [31], and the risks involved are numerous. Business Process Re-engineering is cited as an important success factor in ERP implementations [54] and customization is also discouraged broadly due to maintenance risk (updating the ERP package often breaks the customized code). Thus, it is left to the buyer to find the software that fits the company's processes best when choosing an ERP system. Choosing the right system has been found to be an important factor [32], but in reality it is often overlooked and many vendors simply ignore the problem in the sales process. [33] We should note that Blumenthal, as the proposed father of ERP, however, states that *"One should not start out by redesigning the organization merely because that would make its architecture more elegant, or our systems-design job easier."* [10, p. 41]

Open source ERP software projects are also an option, a number of forks of the original Compiere system (Adempiere, Metasfresh), Odoo, and Dolibarr, to name a few. For the practical section of this thesis I have chosen to build on yet another open source system called ERPNext, due to the way customization is supported in this particular project.

2.4 Two paradigms to be merged

The previous sections described two radically different paradigms in business software systems. Workflow Management aims at automating any business process while simultaneously guiding the process to a more mature and "better" state using continuous improvements, removing actions that do not add value, and automating any action ripe for automation. ERP, on the other hand, represents the rigor of "best-practices", practices that may evolve over time with the arrival of modernized versions of the software and force the organization to adapt to these changing practices.

Most papers cited in this thesis are from the '90s and early 2000s. Some are much older. Yet, very little has happened to bring these two paradigms closer together. To me the obvious answer to the problem of fitting software to the business processes would be to combine the two worlds, to create an ERP system that can be customized by drawing process charts graphically.

Some effort has been made, in the form of ERP software vendors acquiring workflow management software companies, e.g. Baan buying COSA and incorporating the workflow solution into the ERP system, or Oracle's Oracle Workflow which is a separate offering in the e-Business Suite range of products. [34] Still, ERP systems are considered mainly parameterized and hardcoded (e.g. [35]).

The possibility to activate ERP systems and their functions from the workflow management system does, of course, remain. This has recently been further simplified with the emergence of web technologies like *REST APIs* which have often been used to bring ERP systems to the modern web platforms.

The term *REST* (*Representational State Transfer*) was first used by Robert Fielding in his PhD dissertation in 2000 [36] and it describes the set of constraints a web-based client-server architecture would impose on the communication between its components. As with all client-server architectures we have the implication that the user interface and storage functionality should be separated. Further, the system should be stateless, in that the server does not keep track of the state of each client, i.e. the *Session state*. This in turn implies that each request to the server must contain all necessary information for the server to be able to handle the request. The architecture also imposes constraints regarding caching, a server can explicitly or implicitly allow or disallow caching, which may reduce repetitive requests. The key to the REST API lies, however, in the *uniform interface* which in short means fetching or manipulating *resources* (any data, including executable code) by means of a *resource identifier* (URN, URL, URI), the resource being transported in a *representation format*

(HTML document, JSON structure, etc.) known to the *user agent* (e.g. internet browser), possibly accompanied by *representation metadata* (data type, time stamp), *resource metadata*, and *control data* (e.g. caching constraints, request parameters). There is also the constraint that the system should be hierarchically layered, in the sense that "each component can not "see" beyond the immediate layer with which they are interacting". [36] As earlier stated, any client application code may be treated as a resource and fetched by the user agent on demand. A *resource* in this sense is no longer a specific file on the storage system as was the case with the static web pages of the early days of the World Wide Web, but can be composed from any number of information sources by the server application. The client, however, receives the data as a single resource.

The result of adhering to these architectural principles is a system that is easily scalable (statelessness, API consistency), easily revised (no components necessarily permanently installed on the client, except for the user agent), and flexible, while unified resource naming provides the key to mapping resources by intent as opposed to physical location, thereby enabling different versions of a resource (HTML, pdf, JSON, XML, etc.) to be requested and delivered, depending on context and capabilities. Replacing server side hardware, operating systems, or server-side application implementation does not require changing the interface, leading to more options in regard to future demands.

Taking advantage of REST APIs is a natural evolution in the development of ERP systems due to increasing diversity of client operating systems, along with the ubiquitous use of internet browsers on all platforms. Internet browsers allow for thin (or even not so thin) clients to run on any platform because of the standardized application runtime environments (Javascript, WebAssembly) in use. This eliminates developing client software for different systems, or forcing customers to use one particular operating system. Furthermore, updates to these client applications can be easily applied simply by changing the resource on the server. This also brings new opportunities to vendors who are now able to sell their systems as *Software as a Service* (SaaS) offerings, bringing ERP "to the cloud". [37]

In Section 5 I set out to implement this kind of merger between Workflows and ERP systems. For the ERP system I have chosen ERPNext. This is not a result of scrutiny on my part, rather, it was suggested by Comsel System that ERPNext would fit their needs better than some alternatives. The customizations done in cooperation with Comsel System are discussed in sections 5.2.2 and 5.6. This system turned out to be a lucky draw for the thesis - it is easily customized and anything and everything can be overridden without

really touching the code of the ERPNext project. For the Workflow Management System, parts of the user interface for creating process definitions will utilize the Open Source library JointJS for creating graphs. Choosing between BPMN and Petri nets is not an easy task, BPMN being a well-known standard in the industry, and Petri nets being formally defined with the possibility of verification and simulation. My decision to use Petri nets is firstly due to the formalism. This will enable verification and simulation to be added at a later stage. BPMN can be converted to Petri nets (e.g. [26], [38], [55]) which indicates that it would be possible to add BPMN editing capabilities later if desired. I have also decided to forego the recommendation in the Workflow Reference Model to use the storage interface defined in the reference model. Instead I will store the models as graphs in a *Graph database*, Neo4J. This is primarily due to my own interest in the graph database technology, but it should also be faster and simpler to interpret the models in the workflow engine. If needed, e.g. to interact with other visualization software, conversion to the WFC storage interface format can be programmed separately. The workflow engine will be programmed from scratch. To keep the workflow engine as generic as possible I will also implement a connector application acting as a proxy between ERPNext and the workflow engine.

2.5 Quality Assurance

One area gaining interest in later years is that of Quality Assurance. Increasingly detailed legal requirements in varying fields, from toys to electronics and beyond, put demands on the safety of goods produced. Ethical and moral demands from consumers are pressuring companies to declare in what way raw materials are sourced. ISO quality management standards (ISO 9000 family) and environmental management standards (ISO 14000 family), and to some extent LEAN manufacturing and other related disciplines put quality assurance in high regard. This is a broad topic but I will briefly touch upon a few issues that we will need to address in this thesis. Although not the focus of the rest of the thesis, the reasoning in this section occasionally refers to problems specifically encountered in the metalworking industry in order to better explain the concepts discussed.

2.5.1 Traceability and Material Identification

Traceability is the procedure of documenting the history of a specific product and its parts. In many industries traceability is mandatory and many times the demands for traceability are regulatory. To meet demands, a manufacturer must have the ability to answer questions pertaining to a specific produced item such as where the raw materials originated from, the quality assurance procedure, and the specific quality assurance measurements obtained. Concerning the raw materials or parts incorporated into a product, it must in some cases be possible to trace them sequentially down to where and how the minerals were sourced.

Material identification is the ability to prove that a specific item has been produced using materials specified by blueprints and order details. For example, if an item is specified to be made of S355J2 round bar steel it should not be made of 34CrNiMo6 or vice versa (which would be a worse mistake). Material identification is a slightly less involved procedure than traceability but is, on the other hand, always mandatory.

2.5.1.1 Reasons to aspire to traceability and identification

Several reasons for, as well as factors contributing to, the need for traceability and identification have been identified [11]. *Age* may have an impact on both the configuration of a product and shelf life. Configuration in this context is a question of what components have been used in building the product, as specifications may have changed over time. Shelf life is intuitively associated with food supplies, but also other products may deteriorate over time and may have to be discarded. Determining the validity of warranty claims will also fall into this category. The *origin* of goods, be it produce or products, may impact the quality of the product. Töyrylä [11] uses the example of BSE, "mad cow decease", which initiated regulations on meat traceability. Similar products from different manufacturers, and even different lots from the same manufacturer, may well differ in quality, which may become obvious over time. Such differences may be used as basis for future purchase decisions when identified. *Destination* may be important in the case of recalls in that some configurations may have been sold on specific markets and the need for global recalls can be reduced. *Customizations* are beneficial to track in order to identify correct spare parts for a specific item. *Errors and variations* may occur in manufacturing or in the supply chain, leading to irregularities in shipments. Finally, *illegal activity* can be traced, examples include supply chain "mishaps", illegal copying of branded goods, and unauthorized distribution.

2.5.1.2 Product recalls

If products need to be recalled, in particular for safety reasons, traceability is invaluable. With traceability it is possible to track the exact products affected. E.g. if a specific heat number has been shown to be defective all products produced from that heat number can be listed by serial number and traced through sales channels to the current owner of the product. Without traceability all products that could possibly have been made from that heat number would have to be recalled and replaced.

In metal work manufacturing, demands for both traceability and identification can be met by considering two things. The heat (or smelt) number must be traced for all raw materials within the factory, along with the corresponding approval and identifying documents, and the quality inspection must produce a document specifying the exact measurements of the item. The quality assurance procedure and what to measure should be specified by the customer.

Naturally it follows that each item produced must be permanently marked with the corresponding heat number and a unique serial number which corresponds to a quality assurance document. This marking can be done with laser imprinting or by stamping. Small parts that are physically impossible to imprint with these numbers are grouped into lots where each lot is correspondingly documented, and the packaging marked.

In the manufacturing of electronic devices, we face a similar issue. Electronic devices are assembled from one or multiple printed circuit boards, each of which is loaded with semiconductors and other components. These components can usually be traced by batch numbers or serial numbers. Would a recall be necessary due to a failing component, items recalled can be limited to those where the faulty component is of a certain batch, as opposed to recalling every item sold.

2.5.2 Implications for manufacture

Raw material such as round bar steel is often stamped or painted with the heat number in a not so permanent way or in the worst scenario there is a separate tag attached to the material in some way, e.g. hanging off some strap used to bunch the bars together. Depending on how the material is stored painted numbers will soon be unreadable due to rust and tags have a tendency to fall off, at the latest when the bunch is opened. There have to be procedures in place to permanently mark the material as soon as possible to eliminate mistakes. Removing a bar from the bunch is the next critical step as the specific bar may or may not be marked and may also be the only bar

that was marked in that bunch. Cutting the bar, whatever the procedure, is the last critical step when each piece must be marked accordingly. Regarding lot production, a lot cannot consist of parts made from more than one heat number if traceability is required, since it would otherwise be impossible to know which part is made from which smelt. (These are exceptional cases, e.g. bolts intended for nuclear plants etc. where traceability is mandatory and definitely specified accordingly in the purchase order. Whether these edge cases are necessary to include in software intended for the average shop floor is arguable.)

Serial or lot numbering procedures are more flexible. Since they are dependent on the quality assurance documents it is not mandatory to mark a piece until the first documented quality assurance measure is performed.

Electronic components in the form of PCBs can, as in the case described in Section 5.6 be marked with a bar code or some other form of machine-readable marking.

2.5.3 Implications for software

Each OrderRow should have aggregated Item or Lot objects that link SerialNumbers or LotNumbers to corresponding HeatNumbers and documentation for both as soon as such documents are produced.

In order to assist operators in marking workpieces there should be a mapping of HeatNumbers to specific Materials so that the operator is less likely to mis-mark a workpiece or, more critically, use the wrong material. This also requires each WorkPiece or Item to be specified as being of a certain Material.

In order to mandate the creation of Identification, Traceability, and Quality Assurance documents it should be possible to include attributes for documentation demands in Workflow Operations.

Traceability and Identification would imply a need for a reasonably flexible structure to facilitate a link between the physical raw material, whatever shape, and the corresponding heat number and documentation. This, in turn, requires the stock-keeping of raw materials to be managed in the application which, in turn, makes it feasible to include functionality for raw material purchase orders and the communication pertaining to this. Purchase orders are usually handled not in a MES but rather an ERP or some similar system. It would be reasonable to create an interface, or rather a plug-in module, to communicate with such software to avoid redundant information processing.

3 ERPNext

This chapter introduces the ERPNext ERP system. It can not be explained easily without a thorough investigation of its underlying (and sometimes intertwined) framework Frappe. After a short introduction, the Frappe framework is described in detail, which will give us an understanding of what parts will need to be customized to achieve our goal of a truly customizable ERP system.

ERPNext is an Enterprise Resource Planning system from the Indian company Frappe Technologies Pvt. Ltd. [56] It is built on their own Frappe framework, which is somewhat similar to Django in usage, but more specialized, and with more features. This is a full-blown ERP system with an abundance of modules. The core modules include *Accounting*, *Stock* (inventory), *CRM*, *Selling*, *Buying*, *Human Resources*, *Projects*, (Customer) *Support*, *Asset*, and *Quality*. Additionally, there are industry-specific modules including *Manufacturing*, *Education*, *Healthcare*, *Agriculture*, *Non-profit*, and *Hospitality*. Finally, there is an integrated web shop and customer portal.

3.1 Frappe framework

The Frappe framework, and thus ERPNext, is built around the concept of document types, or *DocTypes*. Each *DocType* is mapped to a single table in the relational database that is at the core of Frappe. A *DocType* in turn consists of fields, each mapped to its own column in the table. On the other end, each *DocType* corresponds to a single form in the user interface, and each field to a field in this form. A *DocType* may also have one or multiple *Child DocTypes* which represent one-to-many relations.

Frappe connects to a MariaDB database instance (PostgreSQL connection is in beta), includes an Object Relational Mapping (ORM) framework and provides background job queuing using Python RQ and Redis. The ORM is rather basic and can only handle a single *Document* i.e. an instance of a *DocType* along with its Child DocTypes. Since the system still has to uphold relations between different DocTypes this leads to custom SQL statements wherever such relations must be upheld. This format is a blessing and a curse; for example, there may be many other kinds of relations between document types in an ERP system, and thus between the tables in the database, but these remain the responsibility of the programmer. The database system is not used for much more than raw storage. No foreign keys, nor stored procedures, are

used. All relations are handled by joins in the SQL code embedded in the Python source code. From a data integrity point of view this feels somewhat dangerous, and it also takes its toll on performance since relations are by definition what relational databases are built and optimized for.

Each *DocType* has its designated Python *Module* with application code (both server-side Python code and client-side Javascript) for the *DocType* as well as a JSON structure that defines the fields of the *DocType*. These JSON files are used to initialize the database tables, and to populate user interface forms and lists with the proper fields. Server-side logic for the *DocType* is called a *Controller* in Frappe terminology. A Controller is a normal Python class that must extend the Document class.

The main user interface of Frappe is called *Desk*. Up to and including v. 11 of Frappe the desk resembles the user interface of a smartphone, with numerous clickable icons depicting different functions. Since v. 12 this has been replaced with a hierarchical card layout, rectangular boxes with drop-down menus and/or navigation links. The difference is illustrated in Figures 6 and 7.

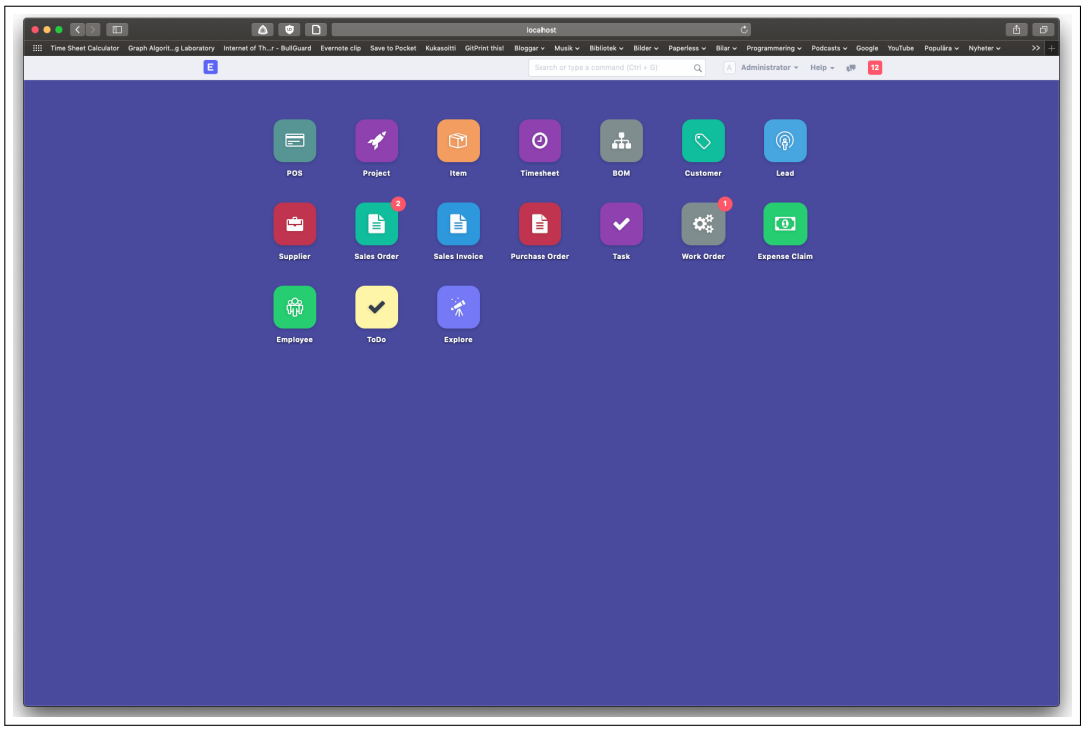

Figure 6: ERPNext v. 11 UI

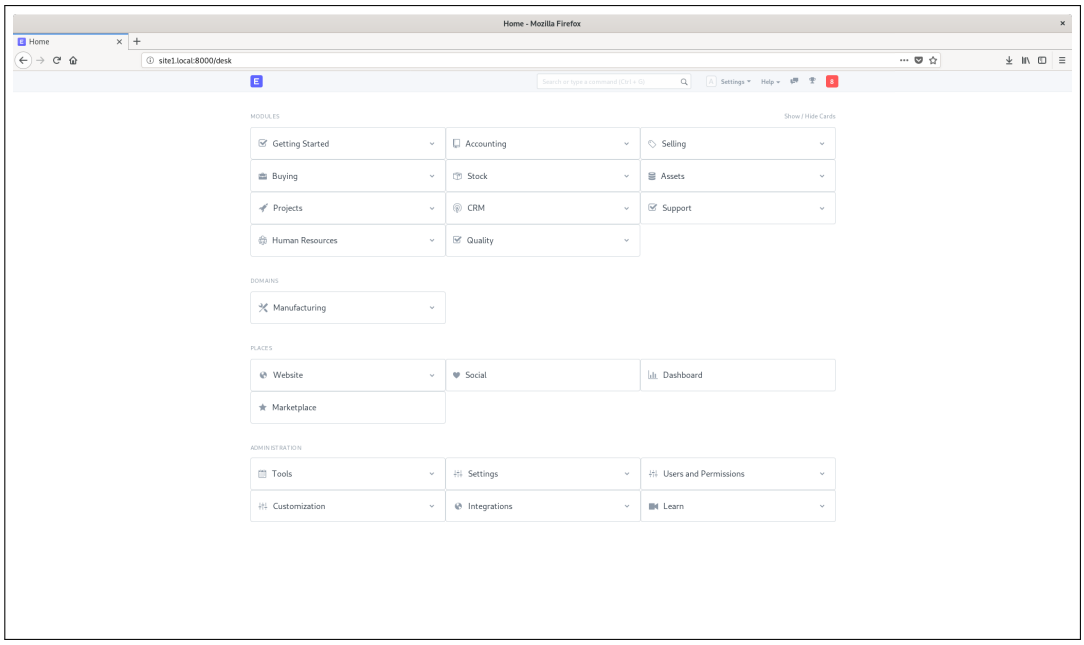

Figure 7: ERPNext v. 12 UI

Security is handled with user accounts and role-based permissions. It is possible to assign multiple roles to a user. Permissions for each role can be set separately on each *DocType* and even for particular fields in a *DocType*. As an administrator can freely define new roles, assign roles to any user, and restrict documents and fields to be edited only by specific roles, this makes for a very flexible, if somewhat cumbersome, system. Icons, or cards in later versions, can be added or removed depending on user or role.

Frappe supports three different ways of creating reports; the *Report Builder*, *Query Reports*, and *Script Reports*. Listed in order of difficulty the Report Builder is a GUI-based interface creating reports for single *DocTypes*. A Query Report is based on an SQL query where special syntax is used for column aliases to configure the layout of the report. Script reports are written in Python and should be developed as part of an application rather than be user definable. This approach allows for very flexible reporting, from user defined one-off lists of inventory to utilizing machine learning as part of assembling reports. Print layouts can be specified with Jinja [57] templates, HTML, CSS, Frappe's own JS Scripting (based on John Resig's JavaScript micro templating [58]), or any combination of these. Based on these templates reports are then converted to PDF format that should be readily printable on any modern system.

Frappe includes workflows as a fundamental concept. It is possible to create rules that in effect override the programmed behavior of the application. One can e.g. modify the handling of a leave application (sick leave, vacation etc.) so

that anyone can create an application, but it would then have to be approved by a manager, or a consecutive row of managers, before it can be submitted.

Unfortunately, each workflow pertains solely to a single document type, and consequently to all documents of that type. This is in stark contrast to the concept of process-based workflows outlined in Section 2.2. Such a workflow would have to span multiple document types, the submission of each perhaps triggering a new document to be created. We would also need to allow for variants of each workflow, depending on different criteria, such as product-specific workflows or even customer-specific workflows. Nothing like that is in any way possible with the Frappe concept of workflow. Whenever I refer to *workflow* in this text, I refer to the broader workflow management, business process kind of workflow, not to the Frappe workflow.

The Customer Portal module already included in Frappe allows for custom static pages to be written in Markdown or HTML, and dynamic content created using Python programming and Jinja templates. Customizing the portal is as easy as arranging the content under the /www folder in any *Frappe App*.

At the core of the Frappe framework is the *Bench* architecture. Figure 8 gives a conceptual overview of the Bench.

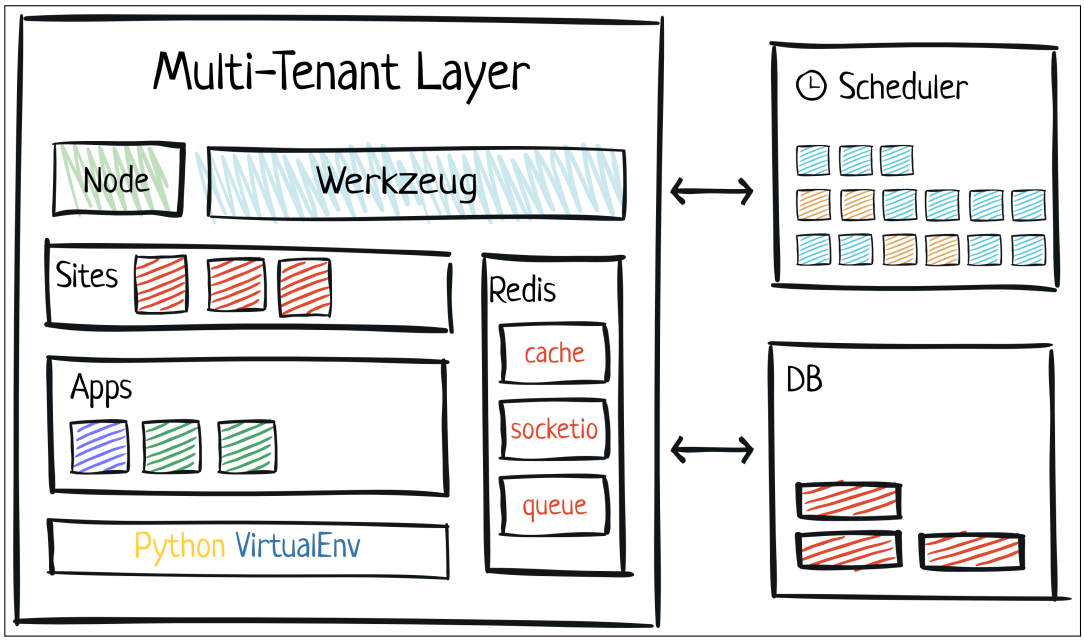

Figure 8: Frappe Bench [59]

The architecture allows for running multiple instances of Frappe applications (sites) on the same physical server hardware, each with its own database schema. Werkzeug refers to the Python web server that is part of Frappe.

Bench also refers to a command line utility that is used for installing, configuring, and maintaining the Frappe environment. Installing Frappe creates the directory structure shown in Figure 9.

Figure 9: Bench Directory Structure [59]

Frappe App is the concept that connects custom built applications to the Frappe framework. An App can define custom DocTypes, code to handle that DocType, and can also be used to override just about any behavior defined in other Apps or Frappe itself. Apps are initially created by running a *Bench*-command that sets up a skeleton app in the form of a local Git [60] repository, in the `/apps` directory inside the Bench directory structure (Figure 9). For more on Git see Section 4.2.

The Frappe app skeleton includes the required subdirectories and a few special files. Frappe itself is an app, and so is ERPNext. When an app has been created, and has been uploaded to a remote Git repository server as its own project, it can be installed locally on any system running Frappe, using a Bench command specifying the remote repository as source repository. Branch or tag can also be specified. Frappe Bench depends on Git for downloading new apps and for keeping the installed apps up to date. It is also possible to configure the system to automatically check if updates are available, and in that case automatically download and install them.

When apps are installed (or created) with the Bench utility a reference is created in the file `apps.txt` in the `sites` directory (see Figure 9). The order

is important as apps can only override the functionality in apps that are referenced previously in the file. The app must also be installed in a *site* (see Figure 8). Overriding functionality is done in several steps. Naturally the new functionality must be programmed. Overrides must also be referenced in the file `hooks.py`. This file has several sections where "hooks", i.e. interception points, can be placed for different purposes. For example, *DocTypes* have events that are triggered in response to user actions, e.g. `on_create`, `on_update`, etc. As an event is triggered, the corresponding Python function is called in response to the event. Using hooks one can make the event mechanism call another function, specified in the hook. Hooks can be defined for all *DocTypes* using the '*' wildcard, or for a specific event targeting a specific DocType. Hooks are not limited to events, however, almost everything in the system can be overridden using hooks. This is, of course, the key to the customizations we will make in Section 5.

Apps may also specify what icons (or cards) are available in Desk.

3.2 ERPNext application

As stated earlier, ERPNext consists of multiple modules. All the modules are always available to the system. When first accessing ERPNext after a new site has been created, the system requires the administrator to specify what kind of business is conducted by the organization. The business domains offered are manufacturing, education, health care, agriculture, nonprofit, hospitality, retail, and service. Multiple domains can be chosen for a single organization. Depending on the chosen domains, the Desk is populated with different cards or clickable icons. These links give access to a wide range of lists and forms for different *DocTypes* deemed relevant to each business domain.

As with any full-blown system, Frappe (and ERPNext) has its own idea of how things should be done. In other words, the workflow is largely pre-defined. In practice these restrictions, if not automated far beyond the imagination of the system's creators, lead to an overwhelming increase in the work needed in any organization.

Assume, for example, that your company produces a product that consists of a number of parts. Assume also that you need to track the origin of these parts using a serial number and a batch number. In this scenario ERPNext demands that these tracked numbers are entered when the shipment is received, i.e. entered into the system. If left out at this point, they can no longer be added at a later date. In practice this is a ludicrous demand for what if

the shipment consists of thousands of items? You would bind up some poor human entering these serial numbers for weeks, introducing as many failure possibilities as the number of items. Meanwhile the serial numbers may not be needed until a product is built out of the parts. This specific example will be addressed in Section 5.

3.3 Customizing ERP Systems

Brehm et al. [61] makes a distinction between nine different types of customization that can be applied to an ERP system, as listed in Table 5. Any combination of these may be used to accomplish the desired functionality. In the same paper they develop a topology of choices for modifying ERP systems, assessing the implications of each modification type on the system. For example, setting parameters is a less involved operation than modifying source code, which in turn implies less risk to the implementation project as a whole. By comparing different combinational options, it would then be possible to assess the overall impact on the system.

Some customization is always needed to implement an ERP system, even if only considering organizational information such as users, roles, and what modules have been implemented. Technically speaking these are configurations, while some vendors choose to view them as customizations.

Whether based on scientific research like Brehm and similar publications, or concrete business experience, ERP vendors are generally reluctant when it comes to customization, other than the setting of parameters. Even this may already impose problems, as a parameter may have an unintended impact on other, otherwise unrelated, modules. Definitions of the following concepts can be found in table 5. Bolt-ons come in different flavors. They can be supplied by the ERP vendor as a way to introduce extra functionality not available in the core package. They may also have been developed by a partner to the ERP vendor utilizing privileged knowledge about the ERP system. These bolt-ons are generally considered safer to implement than a third-party bolt-on from a non-partner. Screen masks and custom reporting tools are generally built in in modern ERP systems. User Exits are predefined places in the ERP code where the code can call an external program if properly configured. This creates the opportunity to include custom code to be executed at these predefined locations. Since the ERP vendor is in control of where in the process this code can be run, it is a generally accepted way of customizing ERP code. Naturally this requires extensive expertise in both business aspects and programming.

More specialized customizations require ERP programming, e.g. development of custom modules. Interface programming is used to communicate with other non-ERP systems, e.g. MES applications, external CRM systems, etc. What is generally considered bad practice and mostly prohibited by vendors, is changing the system source code itself. Normally access to source code is not provided, making this kind of customization impossible.

3.4 Customizing ERPNext

As ERPNext is an open source system, anything and everything can be customized freely. What we need to consider is how to incorporate new features with as little effort as possible. Blindly changing architecture or primary concepts would lead to overly complex changes and most likely a defect system that furthermore would not be able to fully take advantage of future improvements to ERPNext itself. The fact that the Frappe Bench environment is dependent on Git [60] (see Section 4.2) also sets limitations on the way the application can be customized. Changing code directly on a running server breaks automatic updates. As the update pulls the remote Git repository and recognizes local changes, the update often fails with merge conflicts. This effectively closes any possibility of changing code in the existing codebase without having a pull request accepted in the official project, or forking the project, the latter of which would lead to full responsibility for maintaining the application yourself.

Enter Frappe applications [62], Section 3.1. An App in the Frappe framework is a separate Git repository containing any extension or change that you would want to make to the system. A custom application can change the existing system by defining hooks in the file hooks.py. Since v. 11 released in late 2018, just about any function that can be called from the front-end application can be replaced or refined in this manner. Apps can also be used to introduce new document types and to save customizations made through the user interface using *fixtures* (see Section 5).

Much of the system can be customized through the normal user interface. For example, forms, print layouts, workflows (document specific), and even custom fields in a document type can easily be customized to a great extent by a user without any programming knowledge. With the slightest of programming knowledge and a bit of Python (including Jinja Templates), JavaScript, and HTML even more radical customizations can be made directly through the UI. These changes, however, are normally just saved in the database, or saved to

disk in the ERPNext source code directory. This will clutter the codebase of ERPNext itself and may easily lead to problems and merge conflicts during future updates. For a containerized deployment this is definitely the wrong way to go. Whenever the container goes down and is replaced with a fresh one, all changes not stored in the database would be lost. Defining *fixture hooks* (see Section 5) in a custom app mitigates this problem and moves the changes into a separate Git repository.

3.4.1 Process Workflows

My own main focus of this thesis is, as earlier stated, how a company's processes can be supported by an ERP or a similar system. ERPNext is, in more than one way, a good example to make my point. Out of the box it may seem to be very customizable; however, with more experience of the system one realizes that it is very much fixed in the developers' assumptions about how a business works. Document types are small islands with little or no connection to other document types. Where connections exist, they are held together by Python functions within those documents. One example is the connection between *Item* and *Work Order Item* described in Section 5.6.1.

3.4.2 Communication to other critical applications in the organization

Frappe exposes an *Application Programming Interface* (API), or rather two different APIs, to any client that has the ability to connect to the server. The first is a *REST* API (see Section 2.4), which can return raw data for any *DocType*, or resource in the system. The second exposes functions that can return and manipulate any data in the system. The function API is restricted to functions decorated with the Frappe `@whitelisted` *function decorator*[6] in the Python code. There is an abundance of these whitelisted functions and in practice any function can be called by introducing a whitelisted *proxy function* in a custom application.

[6]A *decorator* in Python is a shorthand for extending the functionality of a function. A *decorator function* is a function that takes a pointer to another function as parameter and thus "wraps itself" around the passed function. This construct can be used e.g. to conditionally run code. Extending a function with the functionality of the decorator function is simply a matter of prepending a function with `@decorator_function_name`. In Frappe only `whitelisted`, i.e. decorated with `@whitelisted`, functions can run if called from an external application via the function API.

4 Tools and Technologies used

Current software systems, both cloud based services and on premise systems, often require a large amount of tools and technologies before any, what an old time programmer would refer to as ”real work” i.e. programming, can begin. This endeavor is no exception. This chapter describes the tooling used in the practical section of this thesis.

4.1 Docker

Virtualization, isolating operating system and user space from the actual computer hardware, has been around since the ’60s. [39] Into the early 21st century this was achieved mainly by running complete guest operating systems in virtual machines. Containers, by contrast, are virtualized at the operating system level and the underlying technologies started appearing around 2000. Containers are lightweight in that all guests share the same operating system or run directly on the host operating system. LXC was the first container manager to appear on Linux in 2008. Popularity exploded, however, after Docker, built on top of LXC, launched in 2013. Each container gets its own kernel-level namespace, e.g. user names are separated from the host’s user names. The file system is also separated in a similar manner using a layered filesystem called AuFS. AuFS gives the container its own filesystem but allows, along with other related technologies, administrators to create shared directories when starting a container, in essence a pointer to a directory or file on the host filesystem. This technique allows e.g. databases to retain their data between container startups. Apart from providing a platform to run containers, Docker also includes tools for creating images, the ”blueprint” if you will, for a running container, storing these images in repositories from where they are downloaded upon starting a container that is not already available in the system, and finally tools for managing running containers. There is also a tool called `docker-compose` used to administer the startup and shutdown of any number of related containers. In practice Docker allows for pre-installing applications to be running on any system capable of running Docker containers. Presently this includes any unix-like operating system and even Windows although some limitations in features may exist. It is thus possible to assemble and run a large number of different ”servers” or services on a single system, without contaminating the physical server with all the libraries and services a

certain product may require. Further introducing an orchestrator like Kubernetes into the mix allows for distributing these "servers" over multiple physical servers. Since Kubernetes is only an orchestrating tool to manage containers and in practice only replaces `docker-compose`, I will not discuss it in further detail, however, some references may exist in Section 5.

4.2 Git

Git is a very popular distributed version control system. It was created by Linus Torvalds in 2005 after a fallout between the creator of BitKeeper, the versioning system earlier used by the Linux project, and people in the Linux project over licensing issues. [63] Git is intended for tracking the development history of source code, specifically the Linux project. It is distributed, in the sense that every developer gets his own copy of the complete history. Conceptually, the Git repository can be thought of as a tree structure where *branches* can be created, separating the current file set from the *trunk* until they are again *merged* with another branch, or the trunk. In this manner the codebase may evolve separately on each branch until merged to the trunk. In general, it is customary for each developer to also have his own remote repository, while the master repository for a project is restricted and can only be updated by an administrator. Updates to the master repository are then handled via so called *pull requests* whereby a developer notifies the administrator that proposed changes can be *pulled* from his/her remote, public (i.e. at least the administrator has access), repository. Pull requests can then be reviewed and discussed before being merged with the master repository. *Releases* or *milestones*, points in time on the specific branch, can be tagged using freetext. Using branches and tags it is easy to obtain a snapshot of the project at a specific point in time. The Frappe Bench utility uses Git to handle installations and updates.

4.3 Storage

Storing Workflow definitions (Wfd) is an interesting topic that seems to be somewhat overlooked. All papers I have come in contact with during my research for this thesis only mention storage in passing, generally referring to persistent storage.

The *Workflow Reference Model* [3] introduces its XPDL data format, detailed in the XPDL specification, [46] to be stored as an XML structure. XML implies a text file written to disk, or possibly a chunk of XML data in a relational database or key-value store. There are also *native XML databases* available. In my opinion XML is a bad choice for storing Workflow definitions. Admittedly my opinion may partly be a question of bias due to the countless hours I have spent searching XML documents for debugging purposes during my life. XML is **not** *human readable* in practice, despite efforts to express the contrary. [12]. The fundamental problem, however, is that arbitrary graph data is simply a bad fit for XML. XML is conceptually a hierarchical tree structure, with one root node and child nodes. Thus, an XPDL file consists of the *Package* root node, with child nodes *PackageHeader*, *TypeDeclarations*, *Participants*, *Applications*, *DataFields*, and *WorkflowProcesses*. The *WorkflowProcesses* node in turn contains one or multiple WorkflowProcess nodes which in turn have child nodes DataFields, Activities, Transitions, etc. This is conceptually something completely different than having a start node with a transition to an activity. The relations between an Activity node and the transitions that connect it to other Activities are buried deep in the structure of the document, which leads to relatively complex and error prone parsing algorithms when constructing a visual interpretation of the data and vice versa, or when constructing a memory representation suitable for traversing the graph. Even more problematic are the needs of a workflow engine. Simply stated, to move a *case* from one state to another one would only need to know the current state, what transition is to be fired, and the next state for the particular workflow. To accomplish this, we would have to reconstruct the graph in-memory into a more suitable representation, e.g. an adjacency matrix, or list, and search the graph using an appropriate algorithm. To this concern there are no easy solutions, parsing the document in one way or another is necessary to reconstruct the graph once stored. Native XML databases may ease the burden of indexing and searching for workflows and their components but offer no conversion of the data.

Relational databases (RDBMS) have traditionally been used as the standard way of storing business data. Storing graphs in a RDBMS is in my experience somewhat cumbersome. The same disconnection of the different parts of the graph occurs here, as well as in the case of the hierarchical tree structure. Workflows would typically be stored in one table, actions in another, transitions in yet a third table, and so on. Foreign keys would be used to bind them together. Logically and conceptually the work needed to reconstruct the data is equivalent to that of using a hierarchical tree structure, although it

might be easier to retrieve e.g. all transitions to and from a specific activity. It would perchance also be possible to introduce many-to-many relationships using join tables, something inherently impossible in hierarchical tree structures. The problem of easy retrieval and conversion into a graph structure is still as complicated as with the XML document.

A relatively new concept is that of the *Graph Database* (GDB). Appearing first in the early '90s building on the then thirty-year old Network Database model, they almost disappeared towards the end of the decade [40] but have again gained in popularity. The applications are endless, as much of the world we live in can be represented by graphs. From computer networks to social structures to biology, *relations* between *nodes* can easily be detected in real life. The size of these real-world networks can be massive, take as an example the social network of big social media company such as Facebook. Our workflow graphs are orders of magnitude smaller but still lend themselves perfectly to be stored in a graph database. The opportunities for a simple system are to me obvious; by storing the graphs truly as graphs we can offset the work of converting between formats, and instead query the database for relevant parts of a workflow, cf. the previously discussed transition problem. If needed, e.g. to enable third party visual tools, the stored graphs can be converted to XPDL on the fly.

One very popular Graph Database is Neo4J, ranking first at the moment in the DB-Engines ranking (see Table 4). [64] Neo4J Community Edition is an open source project owned and maintained by Neo4J, Inc. Additionally, there is an Enterprise edition with paid licensing and a number of interesting speed, stability, and security features not available in the Community edition. For the initial development research, the Community edition should suffice. Neo4J stores everything as *nodes*, and *relations* between these nodes. Nodes and relations can be classified using *labels*, and hold data as *properties*. A simple example depicting a minimal computer network can be seen in Figure 10.

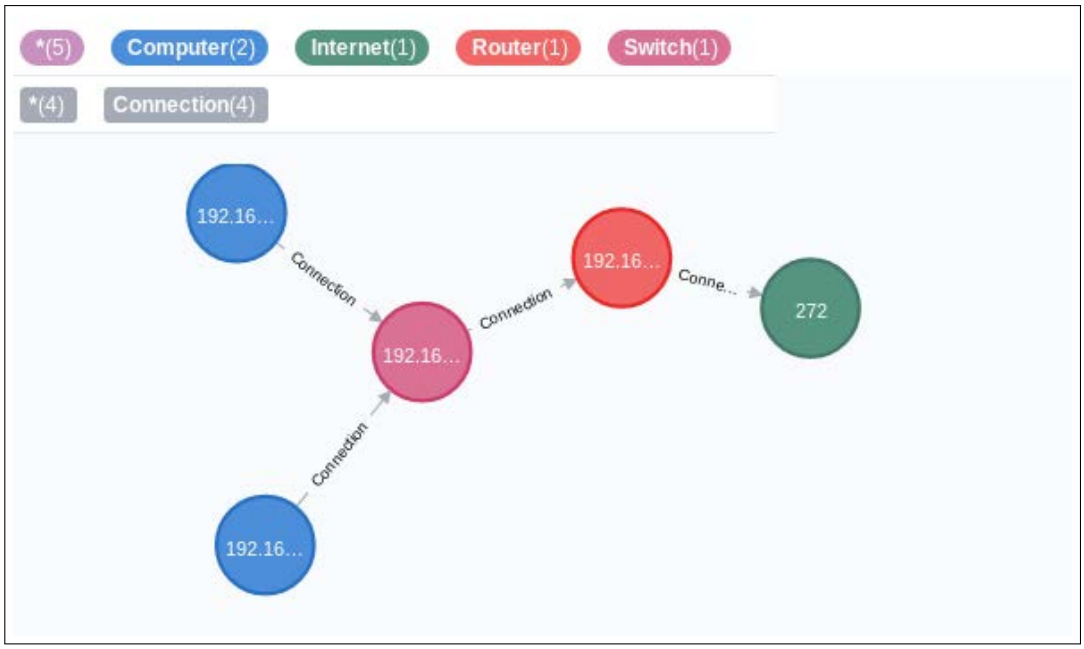

Figure 10: Minimal computer network graph saved in Neo4J

Neo4J has its own query language called Cypher. The language structure is in many respects similar to SQL, but since the underlying concepts are so completely different, in practice, the resemblance is not obvious. Cypher introduces visually descriptive elements that aid in the construction of queries. The syntax to describe a node visualizes the traditional round shape of a node by a parenthesis, and relationships resemble arrows. To create the network depicted in Figure 10 the following Cypher statement could be used:

Listing 1 Example of Cypher statement to create a simple computer network graph

```
CREATE (r:Router {ip:"192.168.1.1"}),
(c1:Computer {ip:"192.168.1.10"}),
(c2:Computer {ip: "192.168.1.11"}),
(s:Switch {ip:"192.168.1.5"}),
(i:Internet),
(r)-[:Connection]->(i),
(s)-[:Connection]->(r),
(c1)-[:Connection]->(s),
(c2)-[:Connection]->(s)
return *
```

In the example I have created the five nodes labeled ”Router”, ”Computer”, ”Switch”, and ”Internet”. Each node has been given a property ”ip”, each with a unique ip-address value. Relations labeled ”Connection” have been created between nodes using the variables denoted to each node in the earlier part of

the statement. Thus, e.g. the variable ”s” holds the node created on line 4, a ”Switch” with ”ip” ”192.168.1.5”. Lastly, the complete graph is returned, e.g. to visualize the graph, as in Figure 10.

4.4 Workflow Editor

Since the user interface of ERPNext is based on web technologies using Javascript, it only makes sense to provide the user with a similar interface for the workflow editor. As stated in Section 2.4 the editor will, initially and for the purpose of this thesis, enable creating Petri nets visually and storing them in the workflow definition database. As not to involve myself too much with the graphical parts of the system, I opted to make use of open source software as the basis for this part. The library of choice is JointJS, which already has building blocks for many of the shapes needed. The application itself is built using the React framework developed by Facebook inc.

4.5 Workflow Engine

The Workflow Engine component will be written in Go. My first motivation is that I want a compiled native language. Primarily this is a question of performance. Every server query that is part of a workflow will engage the services of this component. Therefore, it is of great importance that the services be fast and impose as little overhead as possible. The second factor is the availability of drivers for the Neo4J database. As of this writing there are drivers for Java, Python, Javascript, .NET, and Go. Of these, only .NET (C#) and Go are true compiled languages. Java runs on a virtual machine and Javascript is originally an interpreted language, although virtual machines running Javascript have recently been developed. Since the application will be run in a Linux Docker container (see secs. 4.1 and 5.7) and the Neo4J .NET driver is not listed as compatible with .NET Core (the Open Source version of .NET) and thus is likely to be limited to the Windows environment, Go appears to be the only option. As it happens, Go has excellent built-in features for communicating over HTTP, which will be the primary communications protocol.

4.6 ERPNext Connector for the Workflow Engine

In order to make the workflow engine as generic as possible it will be necessary to implement a connecting element of sorts, to translate between the interface of the workflow engine and the ERP system. This connector, or proxy if you will, is created as a Frappe application, intercepting any calls from a client to ERPNext that are part of a defined workflow. It is my intention to make this as transparent as possible and ignore functions that are *not* part of any workflow, in order to create as little impact on the application's performance as possible. The language of choice is Python, as this is the "native" language of Frappe. This part will be open sourced as it is in direct connection, and to some extent may be considered "linked" with ERPNext and thus bound to the license (GNU GPL) of ERPNext.

4.7 Custom Applications

To enable the workflow described in Section 5.2.1 some custom algorithms have to be implemented. These include logic to handle the removal of items from the "free-to-order" warehouse proposed and add them to the "ordered" warehouse. I have chosen to separate these from the ERPNext connector, as these are specific to the workflows they contribute to. This app is also created as a Frappe custom application.

Additionally, a custom application was made to handle the synchronization between Item and WorkOrderItem described in Section 5.6.1, as well as any custom *fixtures* needed.

5 A Customized ERP System

5.1 Example Organization

Our fictive organization deals in electronics, both as a manufacturer and as a reseller. The core business is designing and manufacturing electronics under the company brand. Manufactured products are assembled from components which are mostly purchased from subcontractors, even if the part itself may be designed in-house. Some components may be readily available standard units from large suppliers. Shelf life is rarely an issue, but traceability is required to avoid mass failure due to components of a certain batch failing over time. In addition to the manufactured products customers may require peripherals and a variety of mounting brackets that may be resold with or without the company's own branding.

The organization has chosen ERPNext as their preferred ERP system. They have found it to be rich in features and flexible enough for their needs. There are, however, a few things they would like to change. Entering received components into the warehouse is cumbersome. In fact, the way the software works does not fit their process at all. Since traceability is an important aspect, components entered should have serial and possibly batch numbers attached. Unfortunately, for this to happen, each serial number must be entered individually, and since a given shipment may consist of thousands of items this is a practically impossible task. ERPNext provides automatic serial numbering, but again these serial numbers are built from patterns. Serial numbers from different suppliers do not adhere to the same pattern for interchangeable products and, once a serial number is entered, it is not possible to change it. Another problem has been identified regarding sales orders. When entering a sales order, there is no easy way to ensure that sold products are available for delivery. There is a small colored dot on the side of the item code on the order form, green if items are available in stock, yellow if available but reserved for other orders, and red if not available in any warehouse. Restocking the warehouse is not the responsibility of the sales personnel but the warehouse manager. Unfortunately, the stock level is not readily visible to the warehouse manager **until wares have been dispatched**, not when an order is entered into the system. This has led to situations where orders have been delayed because replenishing orders were not issued on time.

To remedy these shortcomings a workflow engine based on a number of interesting technologies and theories was implemented and the following chap-

ters will describe it in detail, with emphasis on the latter problem regarding re-stocking to avoid missed delivery dates.

5.2 Custom Use Cases

5.2.1 Sales Order

5.2.1.1 Problem statement

In ERPNext when entering a sales order, one would manually have to see it through to the next step, whatever that is. A possible user scenario would be the following:

1. User enters sales order
2. User checks the available warehouses to see if each item is in stock. Note that the stock level is not readily visible to the user at the time of order entry.
3. If an item is not in stock the user must issue a replenishing order, either a purchase order or, for a manufactured item, a work order. The user must know how to deal with each entry (should a purchase order be issued, or a work order?)
4. Keep track of unfinished orders
5. Keep track of incoming deliveries
6. When stock has been replenished, fulfill the sales order by creating a delivery note.

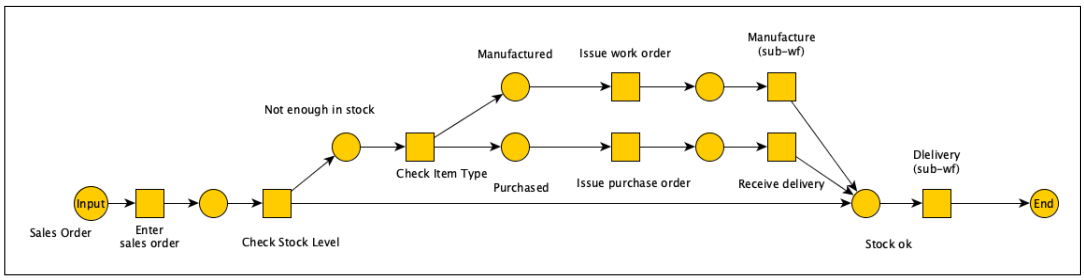

Figure 11: Workflow for the fulfillment of a sales order

Unfortunately, this leaves a lot of opportunities for failed deliveries. There is no reminder to check the stock level, in fact, this task may easily be postponed until the date of delivery comes around, at which point the user will try to create a delivery note which then fails due to lack of items to send. To see what action to take for replenishing, the Item data must be brought on-screen

from the item list, and the view scrolled far down to see if the check box ”is purchase item” is checked. If these actions succeed the list of open orders must constantly be scanned to check if purchase or work orders perchance have been fulfilled so that the sales order can be delivered. Admittedly, this is an attempt at creating the worst imaginable workflow possible for this task. There are simpler ways of achieving the same results in ERPNext. However, ERPNext allows you to perform this task in this complicated manner without questioning.

The workflow promoted in ERPNext documentation [65] is to immediately create a work order for the sales order upon entry. This is again a manual decision to make, and additionally, a decision one must remember to make. This design is based on the assumption that the item in question is *exclusively* made to order, and never kept in stock. We should also take into account the possibility of an item that is kept in stock, but in small quantities that would cover at most a few orders, or that items left over from a previously cancelled order may exist. Both of these are in my experience very common scenarios. Keeping stock value low is one of the key principles of LEAN manufacturing.

ERPNext has an option to automatically issue a re-order on an *Item* [66], however, this only creates a *Material Request* document and notifies the person in charge of purchase orders. He would then have to fulfill the request by creating a *Work Order* or *Purchase Order* as appropriate. Furthermore, the automatic reordering function is based on the *reorder point* philosophy, i.e. a certain stock level is defined for the item, and when this level is reached, a reorder process will be initiated.

To complicate the matter even more, items are *not* moved from the warehouse when a sales order has been placed, rather they are removed on delivery. In the case of raw materials, the stock ledger is updated when issuing a material request as part of a Work Order. This means popular items may be ordered far beyond capacity, as there is no way to know what the *real unsold* stock level is at any point. At the same time, the automatic re-ordering is rendered useless as any stock replenish is likely to occur too late anyway.

The above criticism would be a showstopper, even a deal breaker, in case I would be evaluating this software for corporate use. For this thesis, however, it exemplifies how bad software design can be made reasonably good with a few customizations and workflow management. The reason workflow management is mandatory is the fact that the software *allows* this kind of, I would go so far as to call it, malpractice. We need to stop that and demand the user follows a reasonable workflow.

5.2.1.2 Proposed solution

Using the Workflow editor we should be able to create the following workflow:

1. User enters a sales order.
2. The system automatically checks availability of each item in the sales order and automatically creates *Purchase Orders* or *Work Orders* as appropriate. The user is immediately informed that these new orders are available for confirmation. For any item that can be partly or completely delivered immediately, the system moves the appropriate amount from the *free-to-order* warehouse to an *ordered* warehouse.
3. As such an automatic Work Order is completed, or the corresponding Purchase Order has been entered as received, the user is automatically notified that the Sales Order is ready for delivery.
4. User initiates delivery workflow.

In practice we have the same workflow as in Figure 11 but now we have automated the mind-intensive parts and the user is free to concentrate on work at hand. We eliminate the need for checking the stock level, the need to know what type of document to create in case the available stock is lacking, and finally the need to keep track of pending work orders and deliveries. We have also eliminated the need to create the orders, even though, as stated, they must still be confirmed. Additionally, we have eliminated the possibility of creating work or purchase orders for items already available for delivery. As an extra bonus we have eliminated the ”double spending” possible with the manual workflow.

Some of these functions are readily available in ERPNext and we only need to call the right functions in the right order. Other functions must be custom coded.

A conceptual view of the components needed is shown in Figure 12.

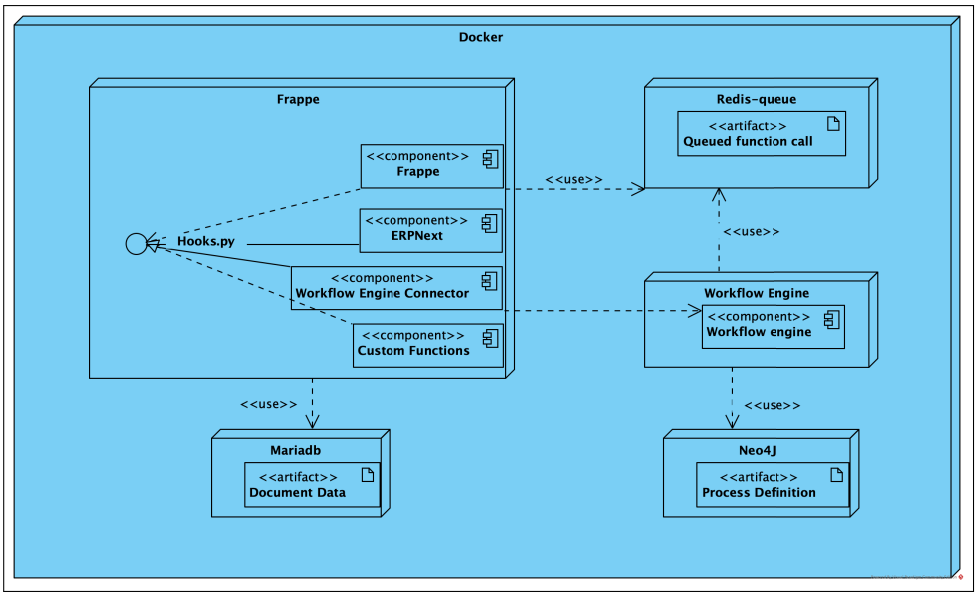

Figure 12: Deployment diagram

The workflow depicted in Figure 11 is rather naïve, e.g. checking the availability must be done for each *item* in the order, and not, as inferred from the diagram, simply for the order as a whole. As a consequence, even for this simple workflow we must resort to high-level Petri nets, introducing sub-nets and semantics for multiple tasks. This construct is shown in Figures 13 and 14. A description of the symbols used can be found in the appendix Figure 30.

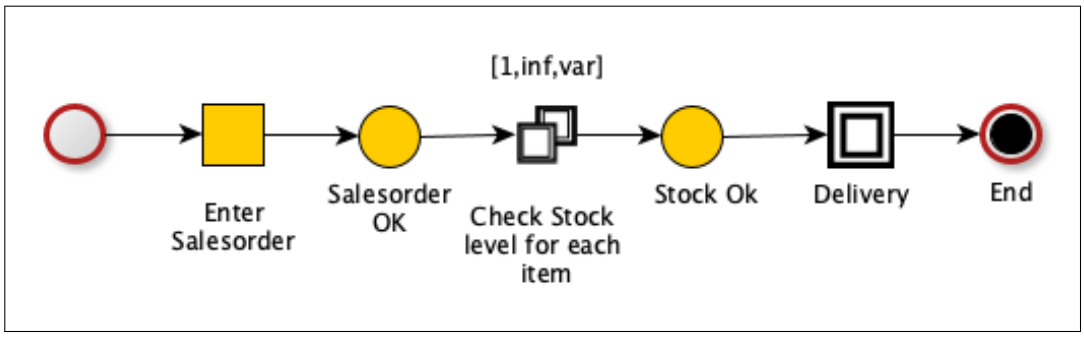

Figure 13: High level Petri net of Sales Order workflow

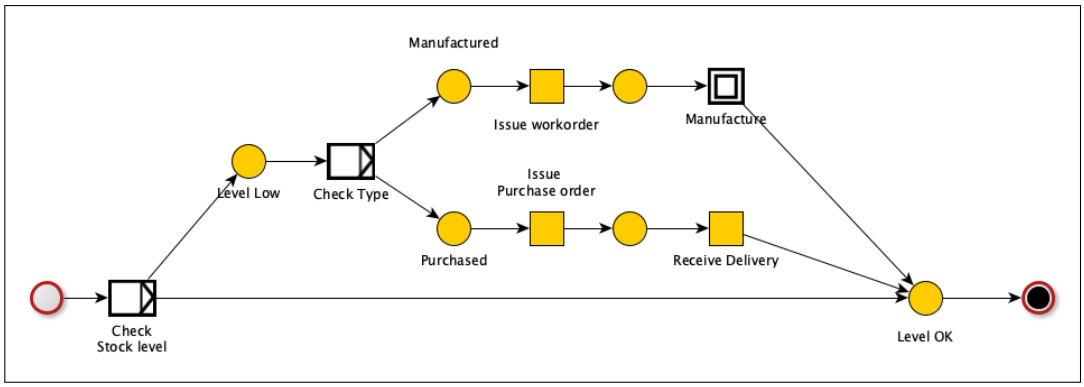

Figure 14: Ensure stock level sub-workflow

5.2.2 Connecting existing platforms to ERPNext

5.2.2.1 Problem statement

An electronic device is assembled from multiple pre-manufactured electronic components in a made-to-order fashion. Additionally, it is to be configured with firmware and tested. Some specifics are determined by operating conditions and customer preference, both firmware version and hardware components to be chosen are determined by these specifics. Each hardware component is marked with serial and batch numbers and is subject to tracking. Already in use is a software system that is used by the worker doing the assembly. This software is used to scan each component and save the specifics of an assembled device, simultaneously giving the device a serial number and attaching the device to a monitoring system.

Adding ERP to the mix allows the worker to find the correct parts to assemble for a specific sales order while maintaining the stock ledger automatically.

5.2.2.2 Proposed solution

The existing application is modified to be aware of work orders. In essence this includes fetching a list of work orders from the ERP system, letting the user choose a work order, and for each component notify the ERP system that a device has been successfully assembled. For each of these three actions multiple requests have to be made to the ERP application. To accomplish this, with as little impact as possible on the manufacturing application, a separate library was developed. This library is discussed in Section 5.6.

5.3 Workflow Editor

The workflow editor was developed using the React Javascript framework developed by Facebook Inc. Additionally it uses the separate Redux and Thunks frameworks to represent the application's state. The concept of the Redux framework is visualized in Figure 15. When a user interacts with the user interface the changes are calculated using *thunks* (hard to visualize as the term stems from the non-standard past participle of the verb think) and filtered through a *reducer* that interacts with and saves the new state to a central store. This store is by principle immutable, an object in the store should not be changed but replaced whenever the reducer changes the state. The user interface is then re-rendered using the new state from the store. This enables a more rigorous state-handling when there are many components involved in a React app, each component would otherwise need to keep track of its own state, a situation that usually results in nothing working and a debugging nightmare.

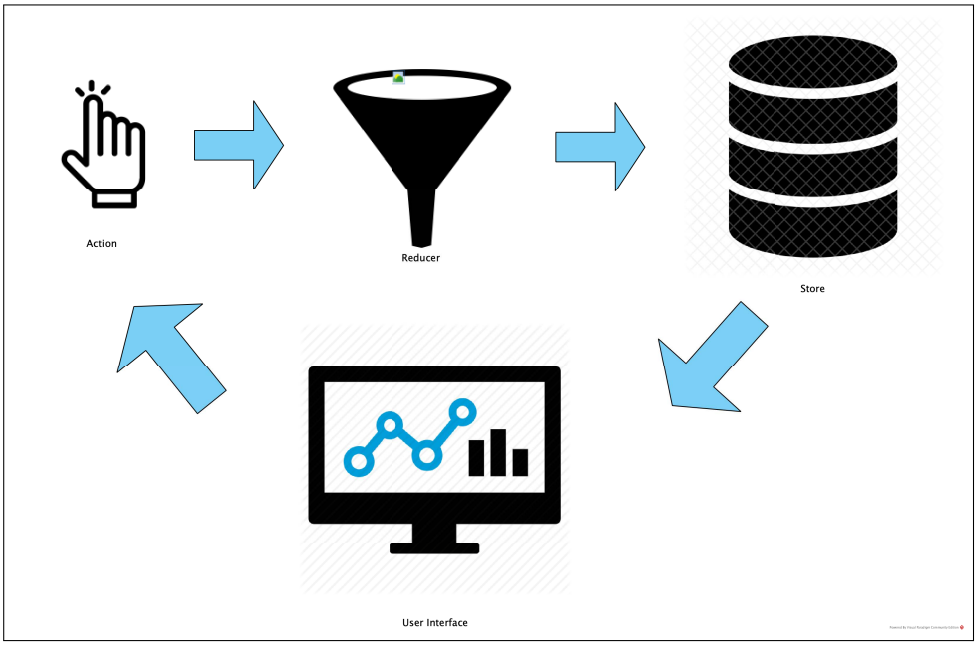

Figure 15: Redux

Due to the design of JointJS, an immutable store for the main UI state in this application, namely the graph itself, is not feasible. The library handles manipulation of the graph within its inner workings and adhering to the immutability principle for the graph would mean rewriting most of the library. Luckily, the immutability principle is not enforced outside of the reducer but is

merely conceptual. This allows us to maintain references to the graph both in the store and in the JointJS framework. The conceptual model of the JointJS framework is depicted in Figure 16.

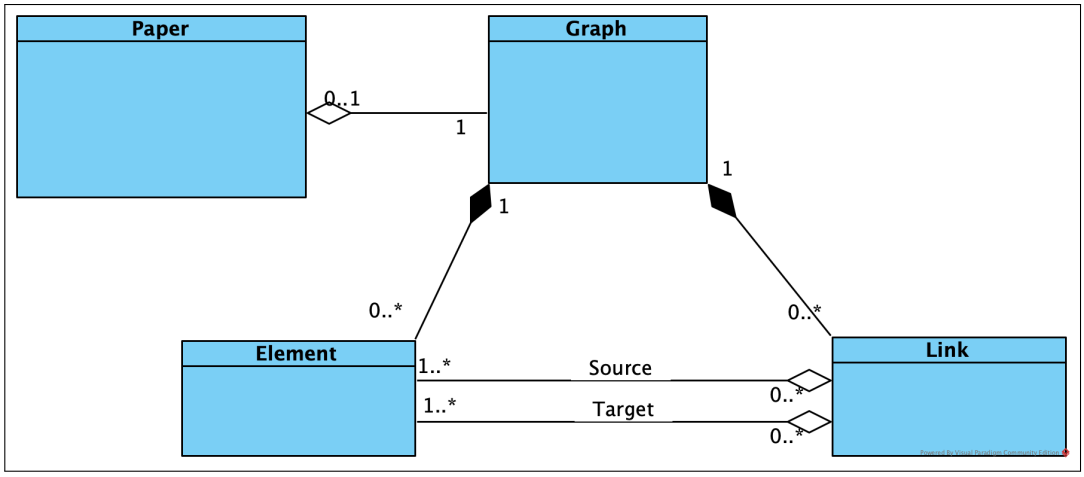

Figure 16: JointJS framework, conceptual class diagram

It consists of the *Paper* representing the drawing area and the *Graph* with its collection of *Elements* and *Links*. An element is an object, or in our case a node in the graph, and links represent the connections between the elements. There are a number of existing element types pre-configured in the framework and these are easily subclassed to extend the functionality. Figure 17 visualizes part of the implementation in this regard. The Element type present in the JointJS library was subclassed with implementation for each of the node types necessary to create process definitions. Figure 17 omits a layer of classes between Element and the classes used for the editor. As JointJS implements a set of classes designed for visualizing Petri nets, including the ability to display tokens, I found it natural to extend on these classes, would implementing simulation ever be of interest.

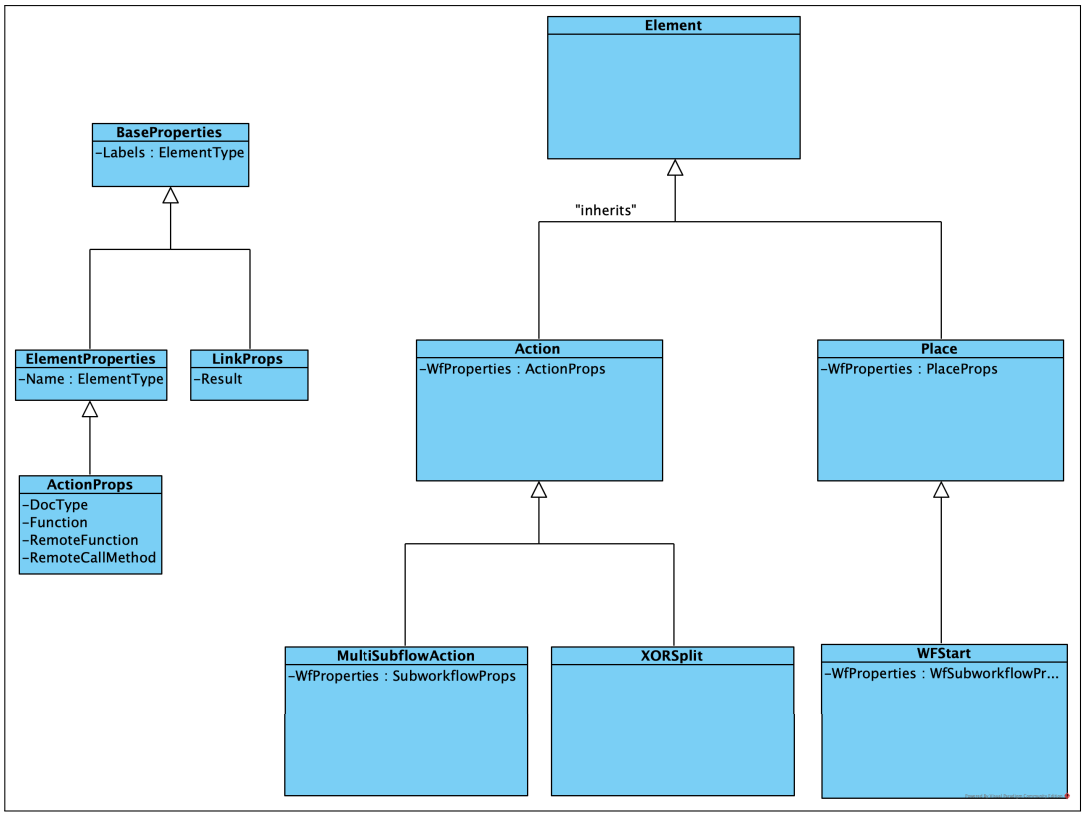

Figure 17: Workflow Editor, partial conceptual class diagram

Each subclass can define its own appearance and other features. The WfProperties property is used as a generic way to store the properties that will eventually be persisted in the graph database.

The main window of the workflow editor consists of a top toolbar with buttons for opening and saving a workflow graph, adding a new graph, and a button for calling up the drawing toolbar (Figure 18). By clicking on one of the tools this becomes the selected tool.

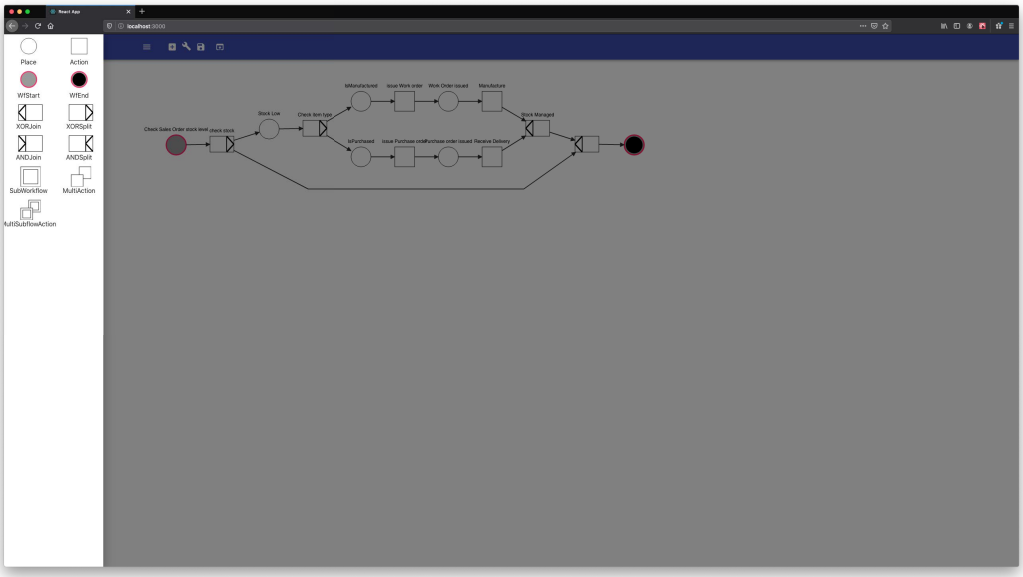

Figure 18: Workflow editor toolbar

Clicking on the paper after selecting a tool renders the corresponding node element on the screen and adds it to the graph. Links, or in our case arcs, are added by dragging from one node to another. Selecting a node brings up a context-aware sidebar that allows the user to fill in properties for the selected node (Figure 19). At the time of capturing this screen the exact properties for each node type were yet to be finally decided on, indicated by the somewhat confusing set of properties in the screenshot below (Figure 19).

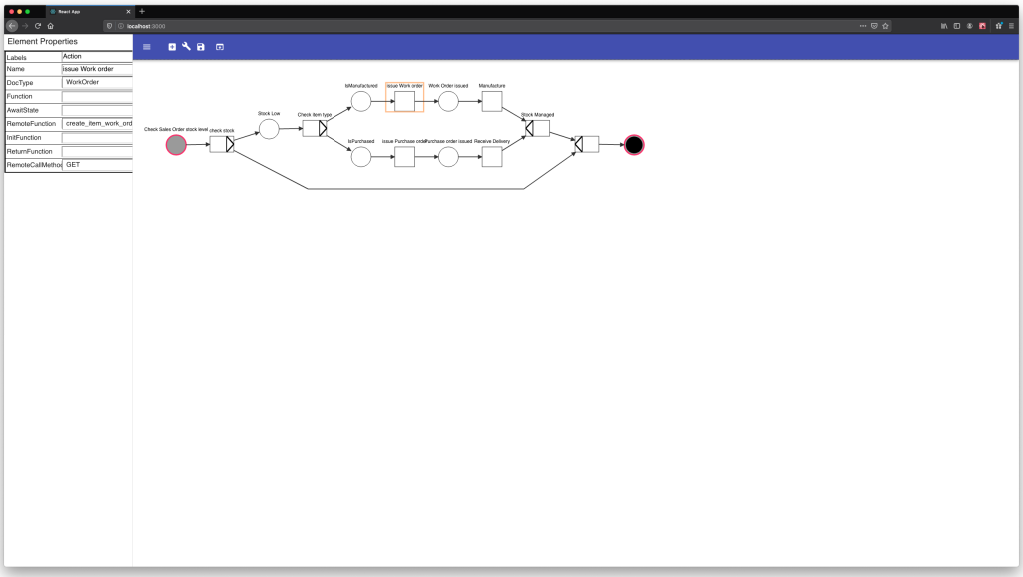

Figure 19: Workflow editor, screenshot

As is, this is a valuable tool for creating the workflow graphs needed for

this project but should not be considered a production-ready application by any standards. Features include creating, editing, loading, and storing process definitions. Storing and loading are handled through REST API calls to the workflow engine, which handles the actual saving and retrieving to and from the Neo4J database respectively. Future improvements would include user experience features, better layout functionality, better error handling, and the ability to define the specific properties for each node type without having to edit the source code.

5.4 Workflow Engine

In this chapter I will discuss the design of the workflow engine and the connector application that ties it to ERPNext. There will also be a brief mention of the custom code needed to implement parts of the automation of our example workflow.

The workflow engine should be a generic component designed to handle workflows. This means that no functionality specific to ERPNext should be placed in the workflow engine. Its sole purpose is to interpret workflows and call the appropriate functions defined in the workflow depending on the state of the document at hand, and the action to take place. This sets the target for our design goals regarding the workflow definitions themselves.

Workflow Definitions will be represented as Petri nets. Petri nets are, as explained more in detail in Section 2.2, graphs consisting of *Places* (P), *Transitions*(T), and the *flow relations* (F) between places and transitions. Representing workflow state can, according to Haesen et al. [67] be accomplished in one of two ways. Either the state is present in the workflow engine, in which case a "working copy", we call this a *process instance*, of the workflow *must* be created for each case going through the system, giving us a *state-aware* workflow engine. Otherwise state information must be present in input data, e.g. on the document instances themselves, making the workflow engine *state-unaware*.

The state-unaware approach requires no process instances. Some changes to the targeted ERP system are likely to be needed as each document must hold the state in a representation the workflow engine can unambiguously interpret. Pure state-unaware engines require more alertness in designing proper algorithms for searching and for managing workflows that include several documents. It is also possible that some state logic must be handled outside of the workflow engine. In other words, this is a more complicated approach involving changes to ERPNext itself.

In this discussion it is wise to take into account how workflows are created and edited. In practice, a workflow definition could change at any point in time as a user is free to bring up the workflow in the workflow editor, make changes, and store the new workflow. The question then becomes whether cases already in progress should adhere to the new workflow, or remain bound to the earlier version. Since workflow editing surely may change vital aspects of preconditions pertaining to workflow routing, i.e. a new requirement may be present for an action to take place, where no such requirement previously existed, I find it imperative that a case that is already initiated would have to obey the rules of the workflow version that were in place when the case was initiated. This implies versioning of the workflows *or* that process instances must exist. Workflow versioning might seem a tempting proposition, however, if the workflow engine is state-unaware, this requires algorithms to choose the right workflow version pertaining to a specific workflow action message received by the workflow engine. In a state aware system this can be avoided by keeping the process instances intact despite changes to the master workflow.

State-aware engines may be *stateful* or *stateless*. A stateful engine keeps track of each process instance and the state is saved in the process instance. A stateless engine derives the state from business data, potentially interacting with the managed system. This would require extensive knowledge of the managed system within the process definitions. As I have chosen to represent the process definition as a Petri net, it is easy to represent state within the definition by simply moving a token within the definition as the case progresses through the process.

Haesen et al. further defines engines as *optional* or *middleman*. For an optional approach, any action can take place with or without interacting with the workflow engine. A middleman approach, on the other hand, requires every action to go through the workflow engine. This implies that a stateful engine *must* be a middleman and an optional engine *must* be stateless.

Our goal is to control the ERP system using process definitions. In this case, a middleman system would be preferred as it would emphasize the ”controlling” part. In the real world, we must unfortunately always deal with exceptions to any rule, including the rules of workflows. Numerous business-related coincidences might require a workflow to be ignored, or parts of the workflow bypassed. This is a dilemma as we have already stated that a state-aware, stateful system would be easiest to implement given our preconditions. Furthermore, there is no way to apply a custom field to all *DocTypes* apart from customizing each *DocType* separately. Common fields are all hardcoded in Python code, which we are not allowed to touch as that would break future

updates. Since there is no way of knowing what *DocTypes* are present on the system (any custom app can add its own *DocTypes*) we can not put a reference to the workflow on the *DocType* itself. We could, however, create our own *DocType*, in the connector application, to effectively create a join table in the relational database as a way to map Documents to Process instances.

As it turns out, in Frappe each DocType implements a *state* property that can be used to enforce state awareness. Unfortunately, the implementation is rather haphazard. Apart from a few DocTypes partially utilizing a common dictionary of states, most states are set within the Python (or in some cases Javascript) code and are impossible to extract in a reliable manner. In some places the function `set_doctype` is called with the new state as parameter while in others a simple `self.state = "NewState"` is used, with or without whitespace between operator and operands. Furthermore, states can be combined, e.g. Purchase Orders may be in the state "To Receive and Bill", which is not listed in the common dictionary despite the fact that Purchase Order is one of the DocTypes in the dictionary. I still regard this state property to be our best option to obtain a state-aware system and suggest this as the solution would state awareness be implemented. The natural solution would be to mine the system for state changing lines of code and build a dictionary of possible states for each DocType. Updates to the system would need careful examination of possible changes but that is still doable. For this thesis implementing full state awareness is not necessary. The state property is, however, used as a trigger for restarting waiting process instances.

The API could reflect the *WAPI* (interfaces 2 and 3 of the Workflow Management Coalition interfaces [13]) as far as possible, as they have been a standard at some point in time. Note, however, that the specification is for the COM/CORBA era of client-server computing and does not match the current standard ways of communication over the internet. For example, the specified protocol seems to assume a stateful server model, incompatible with REST API requests. The Wf-XML 2.0, the "current" implementation of interface 4, is in draft, but seems to have been commonly accepted among vendors of workflow systems. Wf-XML lacks any definition of query parameters, instead any parameters are derived from earlier requests. E.g. requesting a list of the available process definitions should result in a list of URIs being returned by the workflow engine, including any request parameters that are needed for retrieving a specific process definition. In fact, this is a rather uncomplicated approach, as it frees the specification from any details, simultaneously freeing the implementors from restrictions regarding parameters and formats. The specification is, as the name suggests, in XML format over *SOAP*, which

is a rather difficult format to work with. The value of following seemingly abandoned standards to the letter is questionable, the only conceivable reason would be to allow for direct communication with other vendors' workflow engine solutions. This, however, is not a feature I would consider important for this thesis as my main goal is to provide a customizable structure to the ERPNext application. With these reservations I will use the Workflow Management Coalition APIs as a starting point, but I do not intend to follow them to the letter.

In order to keep the workflow engine itself generic, I have devised an interface for communication with the ERP system. It consists of Frappe Hooks as briefly mentioned in Section 3.4 and a restrictive set of REST API calls to collect information from the ERP system. This connector will be discussed in Section 5.4.3.

5.4.1 Implementation of the workflow engine

Implementing the workflow engine is a rather straight-forward task. From the partial class diagram shown in Figure 20 we can see that we need a generic base class representing a node in the graph and that base class needs to implement PassToken and ReceiveToken methods. Since this application is written in Go, which implements inheritance by composition, the actual implementation will differ from the UML but remains conceptually the same. These two methods, along with dynamic properties on the persistent nodes themselves, are all we need to implement a workflow. This simplicity stems from the fact that we use the graph database which directs the flow through the application. We need methods to fetch the previous and next nodes in the sequence, but those do not need to be exposed. We must also handle the special cases of splits, joins, sub-workflows, and end points, each with their own expectations on the number of nodes that precede and follow, and possibly different properties than those of basic action nodes.

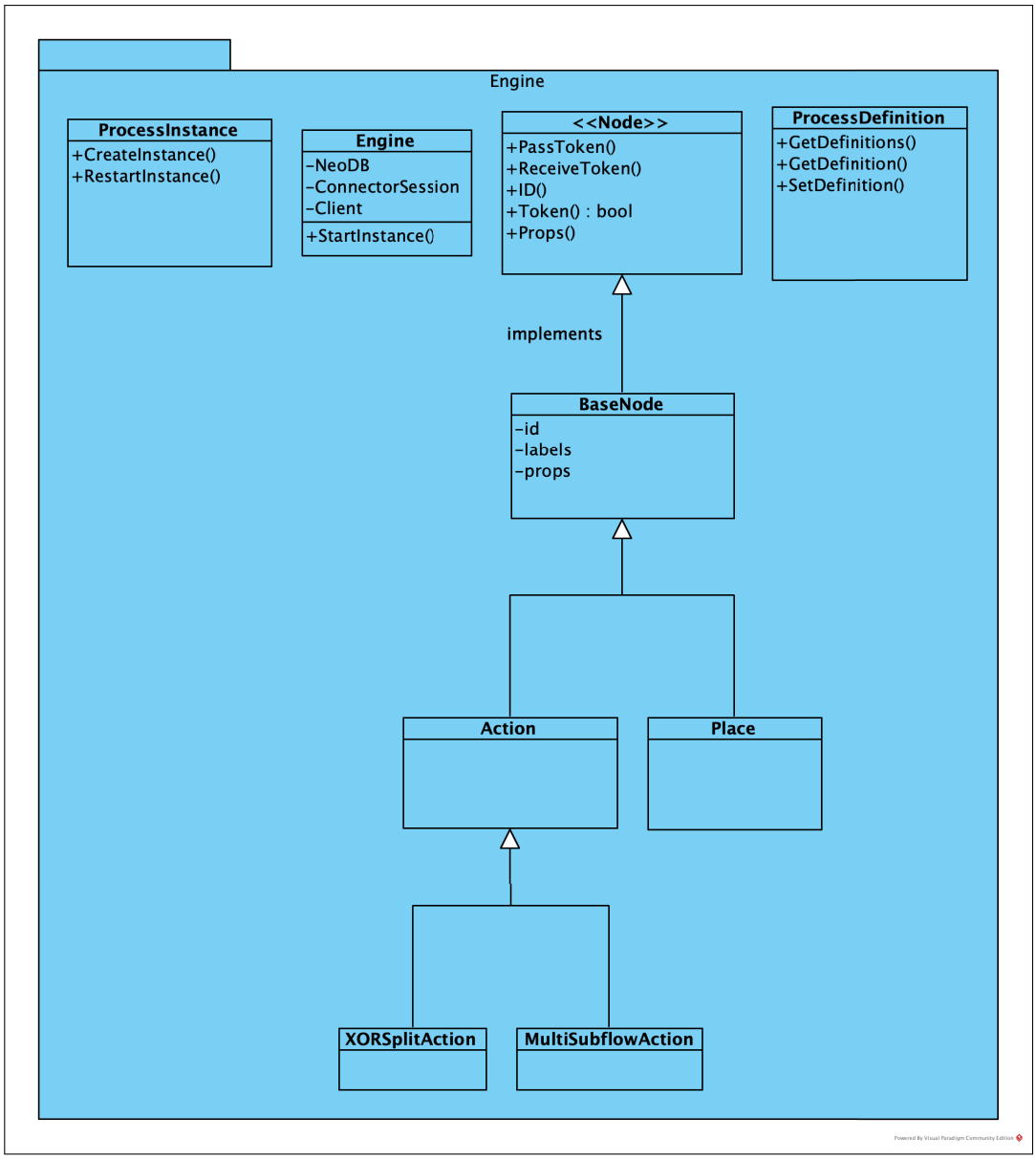

Figure 20: Workflow engine class diagram

In short, a workflow starts by copying a process definition into a process instance. The starting node of the instance passes the token to the node on the other end of its transition arc by calling its ReceiveToken method. The receiving node checks that all preceding nodes have their tokens set and then sets its own token to true and the preceding nodes' tokens to false. It then uses its properties to determine what action to take, if any, performs that action, and passes the token to the next node. The special action "await" will trigger a call to the connector API to wait for an event with the given properties after which the workflow stops. As the awaited event occurs, the connector will restart the workflow which continues in the same manner by passing tokens. The special case of a WfEnd node will trigger a completion of the workflow

and subsequent cleanup. The flow is visualized in Figure 21.

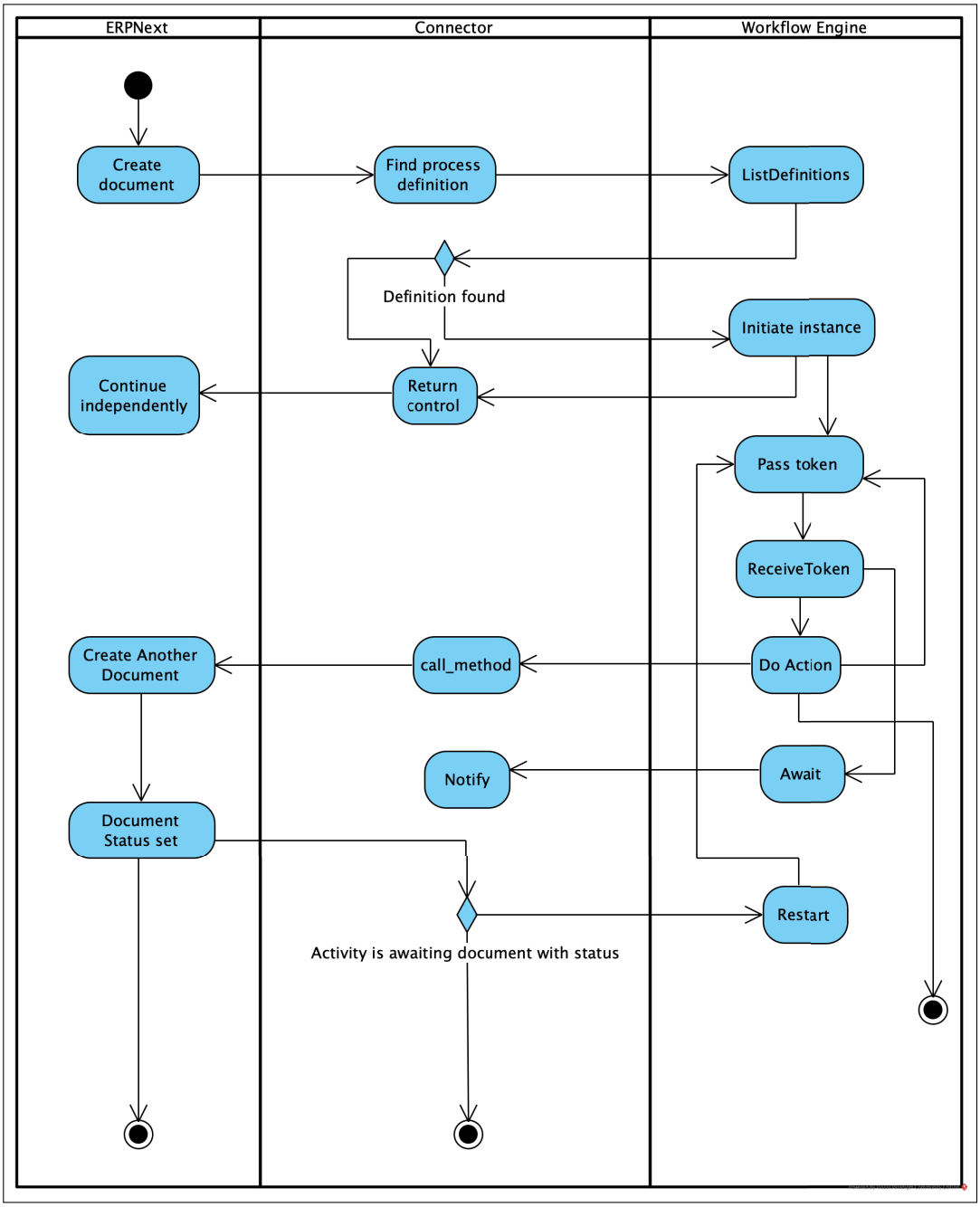

Figure 21: Workflow activity diagram

XORJoin nodes invoke the need for special handling since they by definition have two or more preceding nodes, and thus require only one preceding node to have its token set.

5.4.2 REST API

Figure 22 lists the REST API functions implemented by the workflow engine. The request and response classes that normally would be attached to the

corresponding wide arrows are omitted from this diagram.

5.4.2.1 ListDefinitions

The ListDefinitions API function is used to list the process definitions available. If a JSON body is present in the http request, this will be interpreted as a filter to list only process definitions that meet certain criteria, thus sending the body in Listing 2 will only retrieve process definitions where the DocType property is "Sales Order". Any WfStart node property can serve as a filter and the only limiting factor is the Workflow Editor's capability to create properties for the WfStart node.

Listing 2 Example of JSON message body sent to ListDefinitions

```
{
2     "DocType": "Sales Order"
}
```

5.4.2.2 GetDefinition

GetDefinition retrieves the complete definition specified as a REST parameter using the unique identifier uid assigned to each definition's WfStart node. As an example the api call

`https://wf.example.com/GetDefinition/37252aa1-268c-47f5-a172-dec3b905b5c8`

would return the definition designated by the uid 37252aa1-268c-47f5-a172-dec3b905b5c8. The definition is returned as a JSON object consisting of one list of nodes and one list of arcs. The structure is simple, each node is returned as a JSON object with its properties, and each arc is returned as a JSON object with the unique identities of the nodes defining its endpoints as parameters. This function is primarily used for opening a process definition in the Workflow Editor.

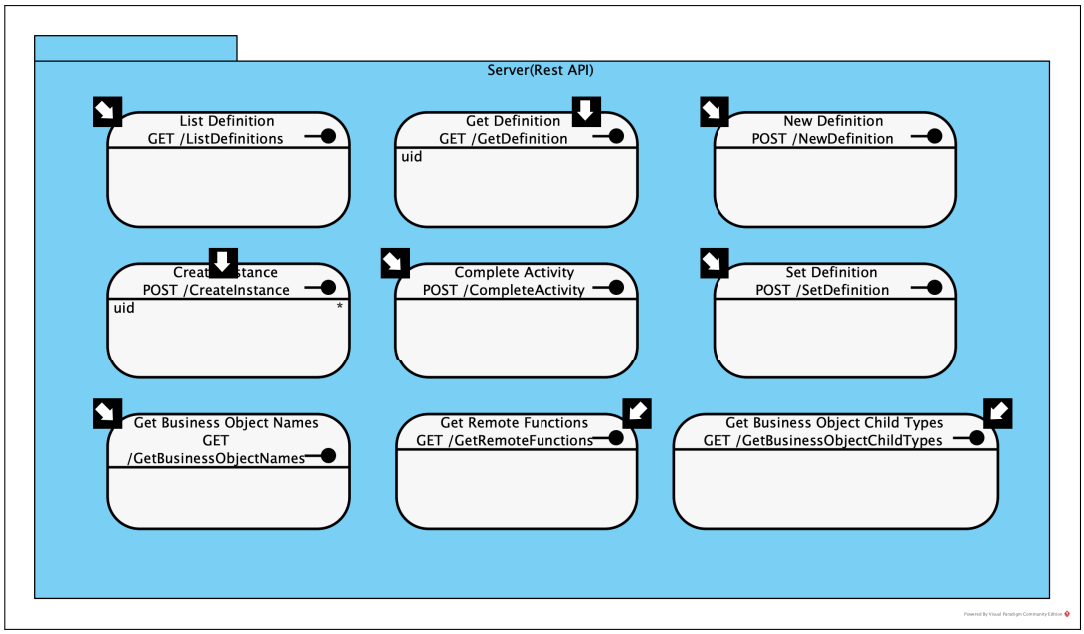

Figure 22: Workflow engine REST API

5.4.2.3 NewDefinition/SetDefinition

These API functions use the same underlying code to update and store the supplied process definition in the graph database. If a process definition having a WfStart node with the uid of the supplied process definition exists, it will be updated, otherwise a new process definition will be stored. Either function can be called for either purpose, despite the naming. By convention it would make no sense to use a function called NewDefinition to store an existing process definition, which is the reason I chose to retain both varieties.

5.4.2.4 CreateInstance

CreateInstance will clone a process definition into an instance, with the option to start it immediately.

5.4.2.5 CompleteActivity

CompleteActivity will search the database for the node with corresponding properties and restart the workflow by having it pass its token.

5.4.2.6 GetBusinessObjectNames

This API function will utilize the connector component to mine the system for available business object names. The function is used by the Workflow Editor to create a convenient drop-down list of available document types.
Its siblings GetBusinessObjectChildTypes, GetBusinessObjectStates, and GetRemoteFunctions serve the same purpose, to obtain system-specific artifact names from the ERP system without having to interact, or have knowledge

of, the ERP system itself. In this manner the Engine serves as a middleman between the Workflow Editor and the custom connector component in order to reduce dependencies.

5.4.3 Implementation of the custom connector

This Frappe custom app keeps the workflow engine generic while providing access to details of the ERPNext system. It serves as a proxy in both directions. Whenever a document is created this app calls the engine to determine if there is a process definition that pertains to this DocType. If so, the app will utilize the engine to instantiate and start a new process instance. Knowing that a document has been created is straightforward using hooks. An example can be found in Listing 3. This construct maps each event (more exist than are present in the example) to a function in the custom application. The star in the example represents all DocTypes, it is also possible to list each DocType separately. The effect is that any time any document is submitted the function `submit` in the wfproxy module will be called, with the document as one parameter. From there we are able to determine its DocType and query the workflow engine for an appropriate workflow definition for that DocType.

Listing 3 Example of the doc_events construct in a Frappe hooks.py file

```
doc_events = {
    "*": {
        "before_insert": "wfproxy.wfproxy.proxy.before_insert",
        "after_insert": "wfproxy.wfproxy.proxy.after_insert",
        "on_update": "wfproxy.wfproxy.proxy.update",
        "on_submit": "wfproxy.wfproxy.proxy.submit",
        "on_cancel": "wfproxy.wfproxy.proxy.cancel",
        "on_change": "wfproxy.wfproxy.proxy.change",
        "on_trash": "wfproxy.wfproxy.proxy.trash",
        "after_delete": "wfproxy.wfproxy.proxy.after_delete"
    }
}
```

For the purpose of this thesis, on_change and on_submit are the most critical and their implementation is similar. As an example, for the submit function Listing 4, a process waiting to be restarted has previously announced so using the notify_event function in the connector app, supplying its identification number, the supposed triggering event (submit, update etc) and the triggering state (e.g. “Completed”). If the expected requirements are met, the instance is restarted by calling the workflow engine REST API. Otherwise the workflow engine is queried for a matching process definition and, if such a process definition exists, starts a new instance.

Listing 4 submit function

```
def submit(doc, method, *args, **kwargs):
    if doc.get("name") in await_dict["submit"]:
        if should_restart(doc,
            **await_dict["submit"][doc.get("name")]):
            restart_process(**await_dict["submit"][doc.get("name")])
    elif doc.doctype not in ignored_docs:
        pd = start_process(doc)
```

The `change` function is similar, but naturally does not start new process instances.

Additionally, the connector app implements functions to obtain a list of callable functions in ERPNext, a list of available DocTypes, a partial list of status designations for some DocTypes, and a proxy function for calling any of the callable functions in ERPNext. A conceptual view of the API, sans the request and response bodies that normally would be attached to the wide arrows, can be viewed in Figure 23.

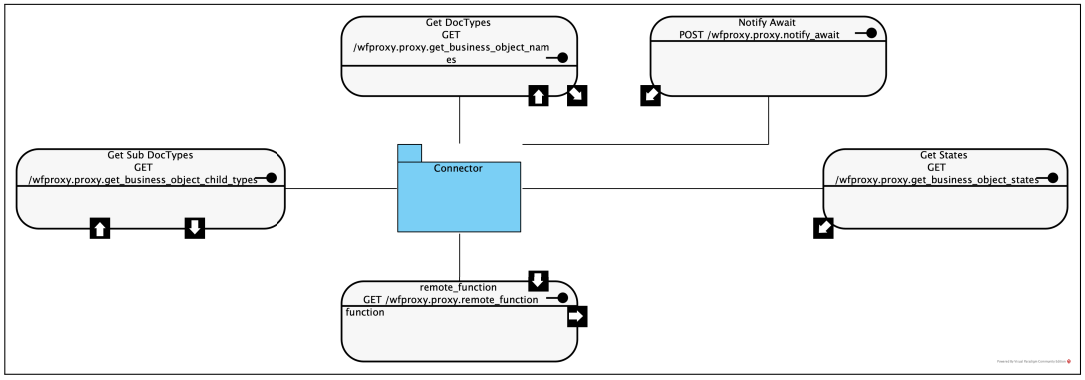

Figure 23: Connector API

5.5 Custom Application

This component ended up being more important than what was initially intended. Since the structure of ERPNext is rather ad-hoc, functions do rarely take arguments in the same order, or take the same type of arguments for that matter. My initial thinking was to supply the right arguments (i.e. argument names) in the Action node along with the remote call function. This would, however, require more than basic knowledge of the inner workings of ERPNext to create functioning workflows for the system. Instead I opted to create smaller routines within this component to deal with the intricacies of ERPNext. Thus, this component includes e.g. a function to create Purchase Orders from

Sales Orders. This will be the function specified in the appropriate Action Node of the workflow, even though there is a `make_purchase_order` function present on the Sales Order document object. These custom functions are rather small in size and simply collect the values that we need to send to the corresponding ERPNext function as parameters. Additionally, this is where I have created functions that are not related to the workflow engine, but rather to the workflows themselves, and which are not already part of the ERPNext system, e.g., a function to check the stock availability of a product. These functions are called through the connector app API using the remote_function API call. This allows for decoupling the workflow system completely from the details of the ERP application. The function name is simply supplied as a node property and passed to the connector app as a parameter. The connector app then calls this function using the Frappe framework. Along with the DocType and the identifier of the document this new document should be based upon, the function name is in most cases enough to create new documents and push the workflow forward.

5.6 Custom Connector for Manufacturing

This component was developed as a solution to the problem described in Section 5.2.2. It was developed in the Go language as a library to be used in other projects. Additionally, the same project can be used as a stand-alone REST API server with endpoints for logging in to the ERPNext system using the OAuth 2.0 protocol, fetching a list of open work orders, and fetching a specific work order by id. OAuth 2.0 is the de facto standard for logging in to an application domain using the credentials of another domain. Thus, it is possible to arrange for logging in to the ERPNext application using the credentials of the proprietary manufacturing system.

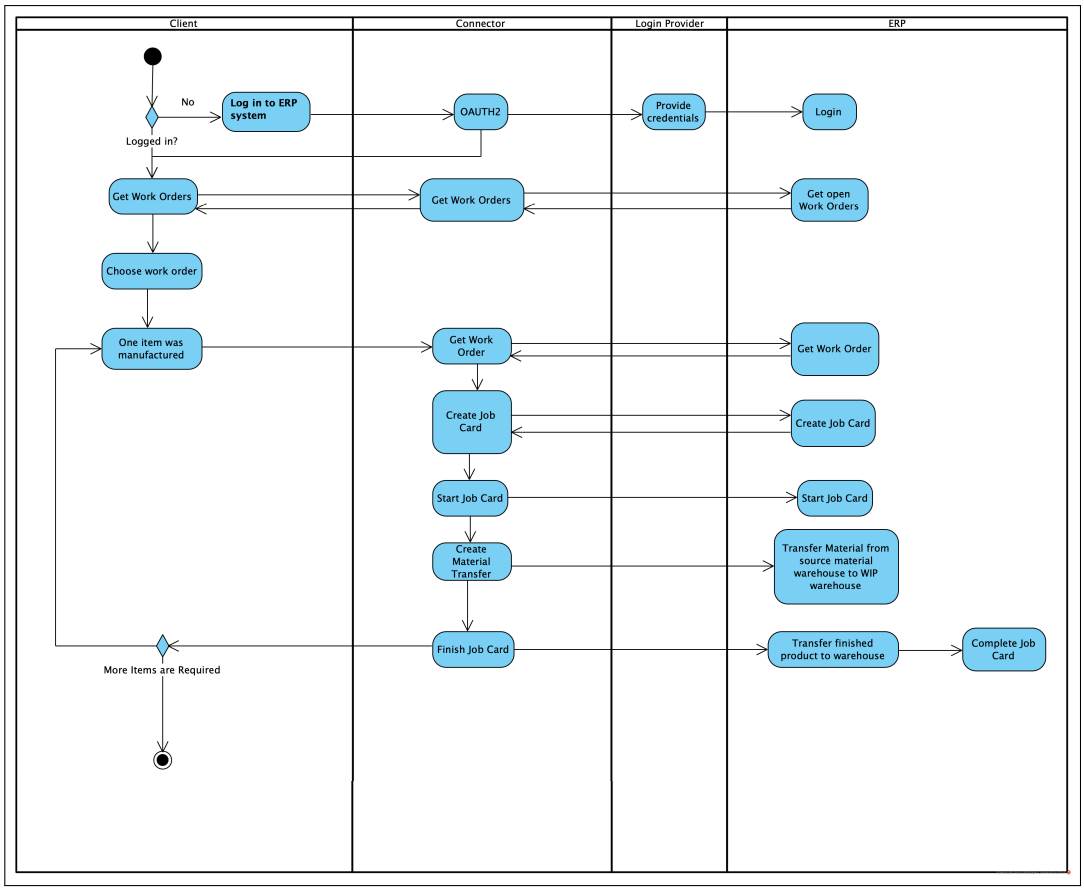

Figure 24: Custom connector usage

The above diagram, Figure 24, describes a possible usage scenario for the component. During manufacturing the appropriate serial numbers for each used hardware component are sent to the custom connector which then invokes material transfers from a source warehouse to the work-in-progress warehouse, finishing off by transferring the finished product to the appropriate warehouse for finished goods. A number of steps needed for this workflow to function are omitted in the diagram, for simplicity. There are, however, no more steps needed between the client and the component which isolates the user from a number of tedious tasks.

5.6.1 Item and Work Order Item

Item is the document type representing any tangible object in the company's possession. This includes both stock and inventory. Relevant to this discussion are items that are used as parts in a manufactured product. Bill of Materials, or BOM is the document describing how to produce, assemble, or modify raw materials and parts into a product or a product bundle. A BOM has a list of parts, each corresponding to an Item. A Work Order is, as the name implies, a document used in manufacturing, describing what should be done

to manufacture a product, and what raw material and parts are to be used to manufacture a product. A Work Order Item is then a record for each part used to create the product. Work Order Items in ERPNext are represented by their own document type and thus their own table in the database. A custom field was required on the Item document type, a separate identifier, Part Number, that represents an item, but is different from the Item Code field that is the default identifier for items. We also needed this identifier to be reflected in the Work Order document type. This proved to be a challenging task, but it gave me a great deal of understanding of the Frappe framework.

As it turns out, creating custom fields on both Work Order Item and Item was not enough. Looking at the code, starting at the function called by the front end when creating a Work Order, I was able to see that the data present in a Work Order Item was in fact fetched from the Bill Of Materials and joined with Item using a hardcoded SELECT clause. Any custom field would be ignored. This is an example of the opinionated way ERPNext is built, the fields in Work Order Item are preconfigured and cannot be easily changed. Changing the code where it would make the most sense is not possible as it will result in merge conflicts at the time of the next update.

The solution was to create a document hook in a custom application, essentially rerouting the original function call (`get_items_and_operations_from_bom`) to a custom function that incorporates the needed changes. Another way to deal with the problem would be refactoring the code to handle custom fields and make a pull request to the project. In this case, however, that option felt a bit overwhelming as it would require new field properties to be created, as one would need a way to specify whether the custom field should be brought over to the Work Order Item or not. Since we are dealing with a JOIN clause we would also need to define if custom fields are to be brought over from Item, BOM Item, or both.

5.7 Dockerizing ERPNext

ERPNext out-of-the-box is meant to be run on a single server with a database instance and three Redis [68] instances running on the same machine. For a small organization this may be a big hurdle as it would require either investing in new server hardware, or bad performance from the application, if deployed to a suboptimal machine. Utilizing the relatively new Docker [69] container technology together with an orchestrator like Kubernetes [70] would give us the possibility to deploy to any number of servers that are not already

fully utilized, or even to a cloud service, completely avoiding the immediate hardware cost. At Comsel System a number of software systems already run on Kubernetes clusters on premise and it was natural that the ERP system should also run on the same platform. There are some efforts made to make ERPNext run in virtualized environments and there is a developer environment available for running on Docker. There have so far been no real efforts to make the system run on Docker or Kubernetes in production. To describe the efforts needed to move ERPNext to a container system we need to look at the installation process. Standard supported installs are done via the Frappe bench command line utility. The utility has numerous options for configuring the system. These include installation of applications (e.g. ERPNext), type of database, migration utilities and so on. All in all, there are 104 standard commands (counted via `bench --help` which lists all the commands) with numerous options on each [71]. Custom commands can also be added simply by calling an add function with a pointer to any Python function as parameter. Unfortunately, this type of installation requires far too much manual interaction to be feasible in a Kubernetes cluster where containers may die and be restarted depending on a number of external factors like resource availability and configuration options. In addition to the bench command ERPNext also comes with an ”easy-install” feature. [72] The easy-install is a system of Ansible [73] Playbooks [74] to determine what system (hardware, operating system etc.) we are installing on. I customized these Playbooks so that they can be used when building a Docker image. Changes include, for example, configuring the system to use databases and Redis instances running in separate containers. Most changes were accomplished by simply adding ”not Docker” as argument to when-clauses in the appropriate playbook.

Listing 5 Example of Ansible playbook with changes

```
# Setup Redis env for RQ
 - name: Setup Redis
   command: bench setup redis
   args:
     creates: "{{ bench_path }}/config/redis_socketio.conf"
     chdir: "{{ bench_path }}"
   when: not docker
```

This work was mostly a question of trial and error, and since building a Docker container with this Ansible-script took more than half an hour on my machine it turned out to be a rather time-consuming effort. Initially each build took even longer due to the vast amount of dependencies required for the application so I ended up splitting the installation into two parts, building

an initial Docker image with preconfigured dependencies that could be downloaded and installed separately, and a second container based on the first that takes on the task of running the easy-install-script. Source for both images can be found in the Code Section of the Appendix.

5.8 Tying it all together

This section contains a summary of how the components in the system work, in a sense the workflow of the final system itself.

5.8.1 Implemented workflow

In order to make a practical working example the workflow depicted in Figure 11 in Section 5.2.1 was first entered into the database using the Workflow Editor described in Section 5.3. Since the workflow handles multiple sales order rows it is in fact necessary to define multiple workflows. The main workflow (Figure 25) will include a sub-workflow as an action, namely the ”Check Sales Order Stock Level” workflow, as depicted in Figure 26.
Each action in the workflows will be defined by the parameters entered in the Element Properties section of the user interface. For example naming the nodes and determining their type is done using this property sheet as seen in Figure 27. More elaborate properties are set depending on the type of node, e.g. in Figure 28 the *Remote function* property is set to `check_stock_availability`. That function is present in the connector app and will be called upon by the workflow engine at the appropriate time, i.e. when that node has received a token from the previous node in the workflow. Depending on the response from that function, either the ”Stock Low” node or the join node preceding the workflow end node will be the next node to receive the token. The respective response, true or false, is entered in the *Result* property of the transitions going out from the split.

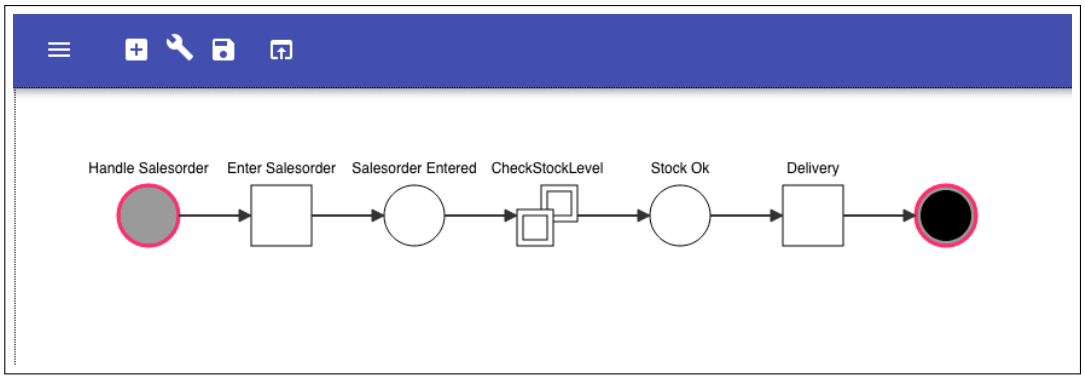

Figure 25: Workflow for the fulfillment of a sales order

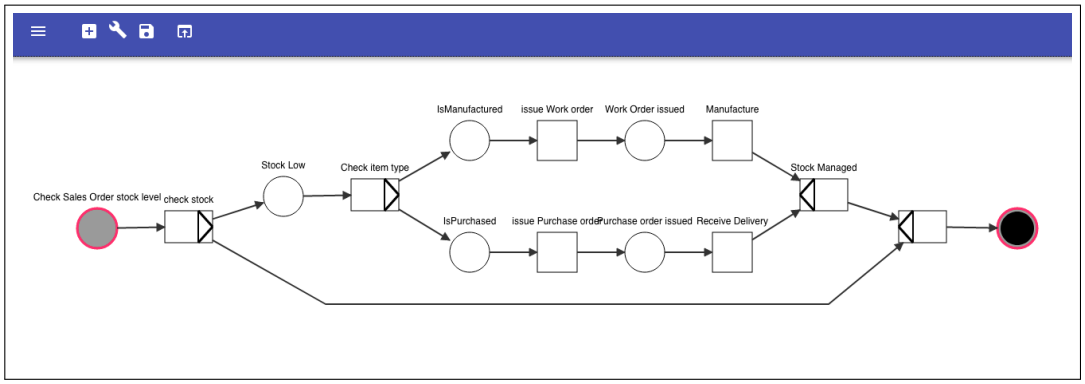

Figure 26: Sub-workflow for creating work or purchase orders if needed

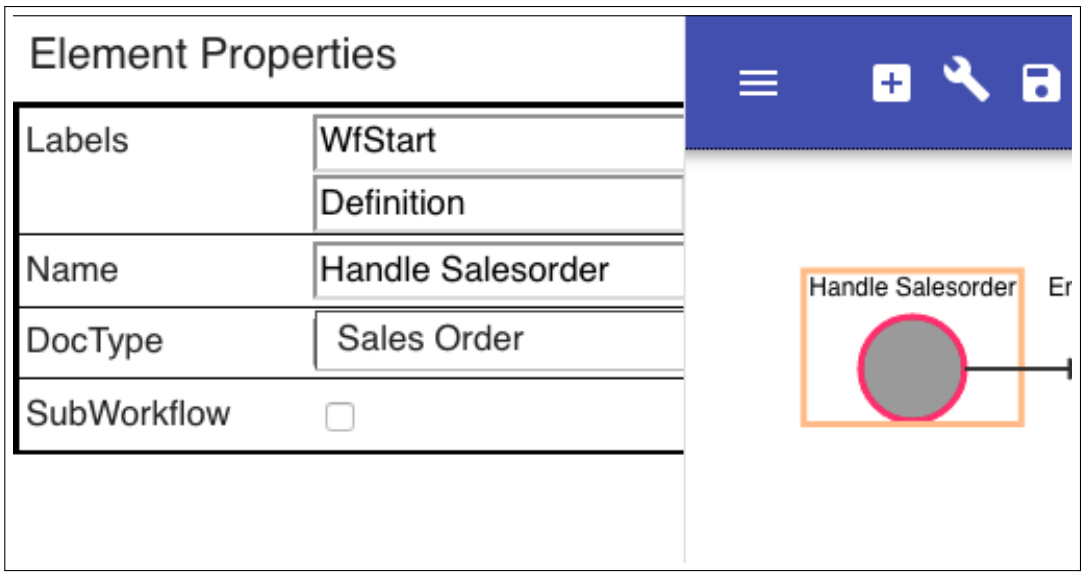

Figure 27: Properties for a start node

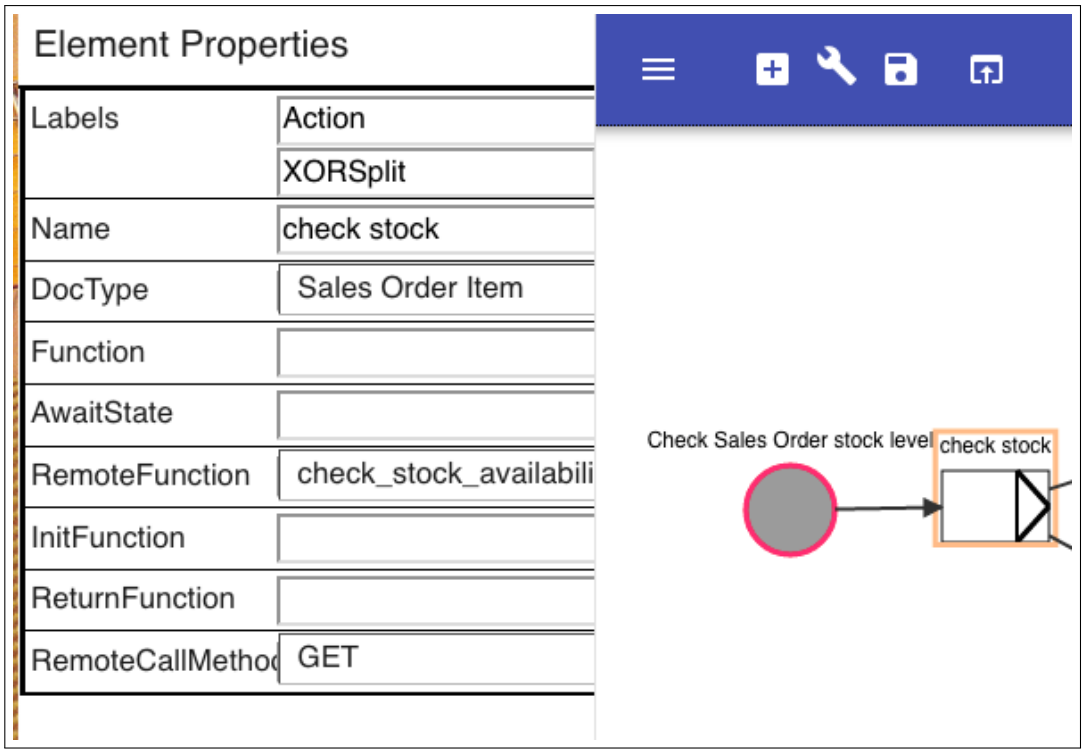

Figure 28: Properties for an XORSplit node

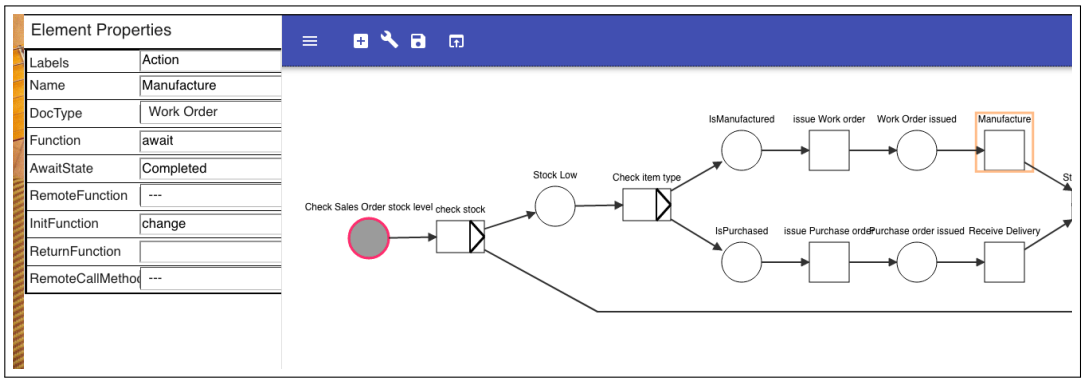

Figure 29: Properties pertaining to an await node

A special type of node is the *Await* node where the workflow stops in order for an external event to take place, as in the case of manufacturing products to fulfill an order (Figure 29). In the above example the property *AwaitState* is used to re-start the workflow when the work order issued in the previous node has reached the state "Completed". Saving the above workflows creates template graphs in the Neo4J database. When the ERPNext user creates a new Sales Order through the ERPNext interface, this action is intercepted by the custom connector application which in turn calls on the workflow engine to start a new workflow. The engine then copies the template workflow and saves the Sales Order's unique identifier as the parameter ResourceID in the start node so as to be able to distinguish different workflow instances and connect

them to the appropriate document in ERPNext. The workflow instance is then started while the ERPNext application, disconnected from this process as it is, proceeds without interference from the workflow engine. Tokens are passed from one node to the next in the workflow, calling any function named by the RemoteFunction property in each node. In this manner, depending on the actual conditions, purchase orders and work orders may be created by the workflow engine in coordination with the custom connector and the ERPNext application.

5.8.2 Problems encountered

Like any software project this one has its share of problems and compromises. Apart from the earlier mentioned conceptual mis-match between the JointJS graphics framework and the Redux framework the following issues are the most prominent.

5.8.2.1 Diverse implementation of functions in ERPNext

Functions in ERPNext, though conceptually similar, may have widely different signatures. This makes it hard for a non-technical user to configure the workflow correctly, were we to include parameter lists, etc. in the workflow definition. To avoid this situation, simpler proxy functions are required for almost any action in a workflow. This partly defies the original purpose of the workflow engine as any new workflow creates the need for programmer expertise. I had hoped for the end user to be able to simply define workflows to control the application but, unfortunately, this is not a likely scenario due to the way ERPNext is implemented. The functions needed are mostly less than ten lines of code in size, and with knowledge of ERPNext or ability to use a debugger, it is an easy task for a programmer to implement them. Nonetheless, I consider this to be an unfortunate turn of events.

5.8.2.2 Document Status and Await

The haphazard construct of status in ERPNext is one of the bigger issues facing someone trying to implement this workflow system in a real-world scenario. There is no simple way to obtain what the range of status codes for a given document could be. Status codes are implemented as text strings in the system and the state may change in a number of ways. No common set of status codes exists, and the way of changing them varies from using functions to assigning new hard-coded values directly. As long as this practice is allowed to continue in the ERPNext community, any workflow where this functionality is needed requires access to the source code and programming experience. As

a way to mitigate these problems, a workaround could be to implement a dictionary of possible values for each document in the custom connector app, but any such effort could be broken by updating to a newer version of the project.

5.8.2.3 Configuration of the ERPNext system.

ERPNext has many features that may not always be obvious. Setting a certain parameter in one document may prevent certain features to be available in other parts of the system. For the implemented workflow to run without errors the Item document must meet certain criteria, for example, a default supplier must be set in order to automatically issue a Purchase Order. For a Work Order to be placed, a BOM for that product must be present. These requirements are hard to mandate in the original system.

6 Conclusion

Creating a workflow engine using a graph database system and connecting it to a randomly selected Open Source ERP system is certainly possible. It is possible to automate parts of the system that otherwise would demand a great deal of manual process routines and rules. Using stateful extended Petri nets keeps the workflow engine algorithms simple, especially when backed by a GDB.

6.1 Measuring results

To evaluate the efficiency of incorporating the workflow described in Section 5.8.1 a series of tests were made, manually entering a sales order with and without the automation in question. In practice the difference can also be calculated by measuring the time it would take to manually create a purchase or work order. One part that is hard to measure is the convenience of not having to remember to check the availability of each item and the consequences of forgetting said task. In practice a sales order with one item can be recorded in about 40 seconds. Looking up stock availability for one item and creating a purchase order takes about three minutes, a task that is fully automated when the workflow is implemented. Extrapolating from the table we can see that a workload of 50 orders each day with five items on each order could save up to 8.25 minutes for each order giving a total of nearly seven hours, or almost one full time worker freed up for other tasks.

Table 2: Sample of time duration needed to create a sales order and ensure all items are available to fulfill the order

#of items	Baseline	with workflow
1	3:18	0:42
3	6:10	0:45
5	9:24	1:10

Parts of the proposed solution were not implemented in practice. Notifications, although quite simple to achieve using Frappes built-in notification system, were omitted for brevity. The same is partly true for moving stock between the *free-to-order* and *ordered* warehouses, however, in this case I found a shortcut as the reorder demand could be calculated from available data.

These are not counted as deviations since the omissions are decision based. The solution fulfills all other requirements from the workflow described in Section 5.2.1.2.

It is my opinion that I have managed to combine a workflow engine with an open source ERP system and shown that this kind of automation is a viable alternative.

6.2 Further research

The software solution created as part of this thesis is far from a finished product. Much work is needed in areas such as error handling and the workflow editor user interface. As such the application is to be considered a rough starting point if e.g. creating a commercially viable product is of interest.

Regrettably there are also performance issues that are visible in all of the ERPNext application as every document creation event polls the workflow engine to see if there is a potential workflow that needs to run. It is not very obvious to people using the application sporadically but definitely noticeable when comparing to an instance of ERPNext running without the connector application. This should be easy enough to remedy for example by storing information about existing workflows in the connector application and thus avoiding the rather costly network traffic when creating documents.

I am not exited about the state handling mechanism in the ERPNext application. I would prefer a more solid solution where states would be more homogeneous across the different document types. Additionally I have doubts about the transfer of control between different document types, or put in another way, the management of depending state over document boundaries. In ERPNext the state of a document may change in response to another document's change of state. As an example, when a delivery is being marked delivered, this also closes the sales order related to that delivery. It is not a given that this event would trigger a state change event for both documents, it is possible that only the delivery document event is ever passed on to the connector. Furthermore this behavior is not consistent throughout the ERPNext application, each case of dependency may be handled differently. There is also the possibility that more than one document may trigger the closing of the sales order - how can we know which documents are being created and how? Should we make the workflows more stringent and in the process deny the use of certain features? More research is needed in regard to these matters.

The current solution creates separate purchase orders for each item in a

sales order. Functionality to automatically consolidate these purchase orders into one for each supplier and possibly even combine purchase orders from multiple sales orders into one, e.g. on a daily basis, would help reduce the workload for the warehouse workers.

In reference to Section 5.8.2.3 it is my belief that mandating certain properties of a document could be accomplished by automation utilizing the same workflow system that has been developed in this thesis. This path has not been explored and could be of interest for further studies.

Appendix

Tables

Table 3: Negative aspects of ERP

Cons	Reason
Cost	ERP systems are prohibitively expensive to a micro- or small business in terms of license fees, implementation, customization, support, and hardware
Lack of flexibility	Vendors push ”best-practice” workflows onto the organization
High risk of failure	Bad fit for the organization

Table 4: Graph Database ranking 8/18/2019 [64]

Aug ’19	Jul ’19	Aug ’18	DBMS	Database Model	Score
1.	1.	1.	Neo4j	Graph	48.39
2.	2.	2.	Microsoft Azure Cosmos DB	Multi-model	29.94
3.	3.	3.	OrientDB	Multi-model	6.28
4.	4.	4.	ArangoDB	Multi-model	5.12
5.	5.	5.	Virtuoso	Multi-model	3.06
6.	6.	10.	JanusGraph	Graph	1.94
7.	7.	7.	Amazon Neptune	Multi-model	1.64
8.	11.	16.	Dgraph	Graph	1.31
9.	9.	6.	Giraph	Graph	1.26
10.	8.	9.	GraphDB	Multi-model	1.14

Table 5: Types of ERP customization [61]

Tailoring Type	Description		Examples
Configuration (customization, in SAP parlance)	Setting of parameters (or tables), in order to choose between different executions of processes and functions in the software package	Define organizational units; create standard reports; formulate available-to-promise logic; use of a standard interface to an archive system	All layers
Bolt-ons	Implementation of third-party package designed to work with ERP system and provide industry-specific functionality	Provide ability to track inventory by product dimensions (e.g., 2 500 m. lengths of cable do not equal 1 1000 m. length)	All layers
Screen masks	Creating of new screen masks for input and output (soft copy) of data	Integrate three screens into one	Communication layer
Extended reporting	Programming of extended data output and reporting options		Design new report with sales revenues for specific criteria
Workflow programming	Creating of non-standard workflows	Set up automated engineering change order approval process	Application layer and/ or database layer

Tailoring Type	Description		Examples
User exits	Programming of additional software code in an open interfaceDevelop a statistical function for calculating particular metrics	Application layer and/ or database layer	
ERP Programming	Programming of additional applications, without changing the source code (using the computer language of the vendor)	Create a program that calculates the phases of the moon for use in production scheduling	All layers
Interface development	Programming of interfaces to legacy systems or 3rd party products	Interface with custom-build shop-floor-system or with a CRM package	Application layer and/ or database layer
Package code modification	Changing the source-codes ranging from small change to change whole modules	Change error message in warning; modify production planning	Can involve all layers

Petri net diagram symbols

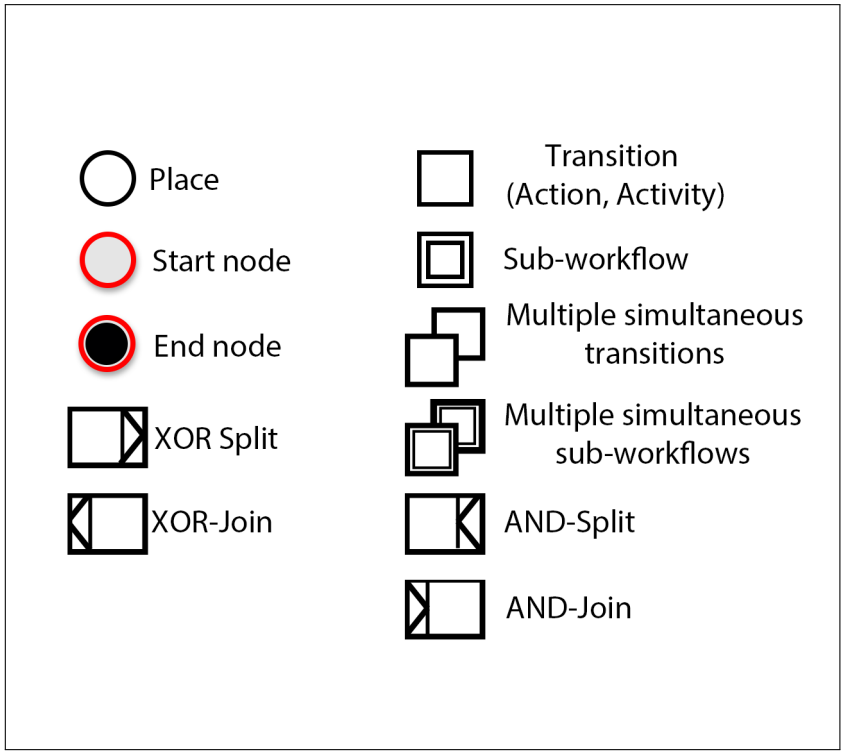

Figure 30: Petri net diagram symbols. These symbols are partly borrowed from YAWL [23]

Sammandrag

Denna avhandling är en uppföljning till min kandidatavhandling *IT-system för små företag* [41]. Den beskriver mina reflektioner om ett IT-systems roll i ett litet företag och framför teorin att de dagliga processerna i ett litet industriföretag borde styra systemets funktion och inte tvärtom. Ett företags processer är en del av företagets identitet och ska uppfattas som en konkurrensfördel. Om standardlösningar tillåts styra företagets processer blir slutresultatet en likriktning av konkurrerande företag inom samma bransch och det enda återstående konkurrensmedlet blir produktens pris. Detta leder till att prispressen ökar vilket medför att företaget måste fokusera på kostnadsinbesparingar i stället för att i första hand utveckla områden som kvalitet och leveranssäkerhet. Min pro gradu-avhandling är en fortsättning på samma tema och behandlar hur man praktiskt kunde gå tillväga för att utveckla system som är tillräckligt flexibla för att undvika problemen med likriktning.

När ett företag växer ökar behovet av datalagring. Uppgifter om tillverkning, kundåtaganden lagernivåer och liknande, som i ett enmansföretag kan skötas med papper och penna, måste nu vara åtkomligt för flera anställda i realtid. Statistiska beräkningar behövs som grund för beslutsfattande. Ett första steg kan vara användningen av kalkyleringsprogram men användningen av dessa blir ofta komplicerad till exempel när kalkylark kopieras och olika uppgifter introduceras i olika kopior eller när mängden olika kalkylark blir för stor. En bättre lösning kan då hittas i affärssystemet.

Affärssystem (*ERP, "Enterprise Resource Planning"*) är stora och dyra system uppbyggda av moduler. Kännetecknande för ett affärssystem är att alla dessa moduler använder en gemensam databas för att lagra uppgifter om verksamheten. De kräver också en stor anpassningsinsats av företaget, antingen måste företaget anpassas till affärssystemet eller tvärtom. Problematiskt i sammanhanget är att affärssystemens leverantörer ofta är av den åsikten att företaget bör förändras för att passa systemet. Många systemleverantörer har utvecklat branschvisa standardlösningar där leverantören valt ut de moduler och de regler för verksamheten som anses vara bästa praxis. Att komma med egna anpassningskrav är i den situationen både dyrt och riskabelt. Som det engelska namnet antyder är systemen ursprungligen utvecklade för stora multinationella bolag. På senare tid har den marknaden delvis mättats varvid leverantörerna svarat med att anpassa systemen för allt mindre företag. Arvet efter storbolagens processer är ofta till nackdel för små företag och leder till stela system som ändrar arbetssättet i ett mindre företag på ett negativt sätt.

Det finns också affärssystem med öppen källkod på marknaden. Fördelen med öppen källkod är att oberoende utvecklare kan anlitas för att anpassa systemet. Öppen källkod har sina egna begränsningar främst när det gäller ansvaret för den förändrade koden, frågan är om övriga delar av systemet kan hållas uppdaterade över tid utan att förändringarna orsakar oförutsedda fel. Också ansvaret för det ursprungliga projektet måste beaktas, man måste ta ställning till om man kan lita på att projektet fortgår och hålls uppdaterat vartefter myndigheternas krav ändras, i synnerhet då många sådana projekt är helt beroende av frivilligarbete, d.v.s. är någon annans hobby.

Avhandlingen utgår från teoretiska diskussioner om IT-systemens motsvarighet till affärsvärldens processer, nämligen *arbetsflöden* (*workflows*). Dessa arbetsflöden var ursprungligen avsedda att sammanfoga helt åtskilda system, exempelvis bokföringsprogram och system för lagerhållning. Tanken var att dessa *system för hantering av arbetsflöden* (*WfMS, workflow management systems*) skulle förmedla meddelanden mellan olika program för att styra i vilken ordning uppgifter kunde utföras och för att automatisera processer genom att starta upp de program som behövdes för en specifik uppgift. Denna ursprungsidé står i stark kontrast mot affärssystemens enhetliga uppbyggnad. Vartefter affärssystemen har vunnit marknadsandelar har tanken på arbetsflöden tonats ner eftersom arbetsflödena redan sköts enligt vad systemleverantörerna anser vara bästa praxis.

Parallellt med att bästa praxis blivit allt mer dominerande i IT-system avsedda för allt mindre företag har ändå tanken på arbetsflöden förts vidare inom akademiska kretsar. *Business Process Re-engineering, BPR*, vars förespråkare hävdade att företagets processer måste skrotas och göras om med de nya förutsättningar som datoriseringen ger, och *Business Process Management, BPM*, som i mildare form omarbetar processerna över tid är två historiskt sett viktiga åsiktsriktningar. Båda beskriver hur företagsprocesserna kan förbättras med hjälp av IT och båda har någon form av diagram för att beskriva processer. W. M. P. van der Aalst har under många år arbetat med processhantering och förespråkar en variant av Petrinät som modell för processer och arbetsflöden.

Att förena dessa två vitt skilda tankemönster kunde vara ett sätt att prioritera företagets processer och samtidigt utnyttja de fördelar ett affärssystem ger, utan att behöva utveckla ett skräddarsytt affärssystem från grunden. För att undersöka denna idé, skapar jag i denna avhandling ett system för hantering av arbetsflöden och kombinerar det med ett affärssystem i syfte att automatisera och styra affärssystemet enligt mer eller mindre godtyckliga regler. Slutresultatet är inte ett komplett system utan ska ses som en validering av konceptet.

Min *"Workflow engine"* (ung. arbetsflödesmotor) är uppbyggd kring processmodeller i form av grafer med noder och bågar. Processmodellerna bygger på van der Aalsts modifierade Petrinät där noderna representerar aktiviteter och tillstånd medan bågarna representerar en förändring av tillstånd. Dessa modeller lagras i en databas som fungerar enligt grafkonceptet (eng. Graph Database) varvid man undviker den i sammanhanget vanliga konverteringen mellan grafiska nät och hierarkiska strukturer. Detta möjliggör att fler varianter av processer kan beskrivas. Processgrafen har alltid en startnod, eller *källa* och en slutnod, *utlopp*. Ett externt program kan starta en process genom att anropa arbetsflödesmotorn som då kopierar den valda processen till ett *ärende* (eng. *process instance*) och låter en *pant* (eng. *token*) passera genom grafen. När en nod tar emot panten utförs den procedur som är lagrad som ett attribut på den noden varefter panten skickas vidare till nästa nod. Den procedur som ska utföras är vanligen en funktion i ett externt program som tillhör, eller kommunicerar med, affärssystemet som ska styras. I de flesta processer kan man också behöva invänta manuella åtgärder varvid noden anropar en väntefunktion och stoppar flödet. Flödet startas igen när de villkor som specificerats i noden uppfylls.

ERPNext är ett affärssystem utvecklat av det indiska företaget Frappé, licensierat som öppen källkod. Systemet är uppbyggt kring det centrala begreppet dokumenttyp ("*DocType*"), motsvarande verkliga dokument i en verksamhet. Sådana dokument kan till exempel vara inköpsorder, försäljningsorder och ordermottagning men även entiteter som kunder, lagervaror och inventarier beskrivs i form av dokument.

När ett dokument skapas eller modifieras i användargränssnittet är det möjligt att styra om anropet och så att säga genskjuta den befintliga algoritmen. Detta gör det möjligt att anropa arbetsflödesmotorn för att skapa ärenden eller starta väntande ärenden. Tack vare systemets uppbyggnad är det enkelt att utveckla små rutiner som arbetsflödesmotorn kan anropa och vi kan därför både styra och automatisera affärssystemet.

Som exempel på hur denna lösning kan vara till nytta i användningen av ett affärssystem skapar jag en processmodell för kundorderhantering. I standardutförande är denna del av processen onödigt arbetsdryg i ERPNext, eftersom systemet inte tillräckligt tydligt klargör för operatören huruvida det finns tillräckligt med varor i lager för att leverera beställningen inom utsatt tid. I praktiken innebär detta att den som skapar ordern måste öppna lagersaldot för varje beställd produkt och dessutom kontrollera att de befintliga varorna inte reserverats för tidigare inkomna beställningar. Med arbetsflödesmotorn kan systemet automatiskt både kontrollera lagersaldot och vid behov skapa

inköpsbeställningar, alternativt tillverkningsorder för produkter som tillverkas på plats.

I denna avhandling visar jag att det är möjligt att kombinera principerna för arbetsflöden med standardprogramvara i form av affärssystem och att man delvis kan automatisera de sistnämnda för att underlätta den dagliga verksamheten i ett litet företag.

Bibliography

[1] K. McCormack and W. C. Johnson, *Business process orientation: Gaining the e-business competitive advantage*. Boca Raton: St. Lucie Press, 2001.

[2] W. van der Aalst and K. M. van Hee, *Workflow management: Models, methods and systems*, 1. MIT Press paperback ed. Cambridge, Mass.: MIT Press, 2004.

[3] W. M. Coalition, *Workflow Reference Model*, WFMC-TC-1003rd ed. 1995.

[4] L. Fischer and W. M. Coalition, Eds., *Workflow handbook 2004: Published in association with the Workflow Management Coalition WfMC*. Lighthouse Point, Fla: Future Strategies Inc, 2004.

[5] Robert M Shapiro, Mike Marin, Tim Stephenson, and Wojciech Zurek, *Xpdl.Xsd*. http://www.wfmc.org/docs/TC-1025_bpmnxpdl_24.xsd.

[6] M. Dumas, *Fundamentals of business process management*, 1st ed. New York: Springer, 2013.

[7] Alexander and C. Alexander, *A Pattern Language Towns, Buildings, Construction*. Cary: Oxford University Press USA - OSO, 1977.

[8] E. Gamma, Ed., *Design patterns: Elements of reusable object-oriented software*. Reading, Mass: Addison-Wesley, 1995.

[9] M. Lewis and N. Slack, Eds., *Operations management: Critical perspectives on business and management*. London ; New York: Routledge, 2003.

[10] Blumenthal, S.C., *Management Information Systems: A framework for planning and development*. NJ: Prentice Hall, 1969.

[11] I. Töyrylä, *Realising the potential of traceability: A case study research on usage and impacts of product traceability*. Espoo: Finnish Acad. of Technology, 1999.

[12] M. Birbeck, Ed., *Professional XML*, 2nd ed. Birmingham, UK: Wrox Press, 2001.

[13] W. M. Coalition, *Workflow Management Application Programming Interface (Interface 2&3) Specification*, WFMC-TC-1009th ed. 1998.

[14] D. Georgakopoulos, M. Hornick, and A. Sheth, "An overview of workflow management: From process modeling to workflow automation infrastructure," *Distributed and Parallel Databases*, vol. 3, no. 2, pp. 119–153, Apr. 1995.

[15] R. Medina-Mora, T. Winograd, R. Flores, and F. Flores, "The action workflow approach to workflow management technology," *The Information Society*, vol. 9, no. 4, pp. 391–404, Oct. 1993.

[16] M. Hammer, "Reengineering Work: Don't Automate, Obliterate." *Harvard Business Review*, vol. 68, no. 4, pp. 104–112, 1990.

[17] J. Chamberlin, "Business Process Reengineering: A retrospective look. Part two," *Management Services*, vol. 54, no. Spring 2010, pp. 13–20.

[18] R. Valentine and D. Knights, "TQM and BPR - can you spot the difference?" *Personnel Review*, vol. 27, no. 1, pp. 78–85, Feb. 1998.

[19] M. Zairi, "Business process management: A boundaryless approach to modern competitiveness," *Business Process Management Journal*, vol. 3, no. 1, pp. 64–80, 1997.

[20] R. Y.-Y. Hung, "Business process management as competitive advantage: A review and empirical study," *Total Quality Management & Business Excellence*, vol. 17, no. 1, pp. 21–40, Jan. 2006.

[21] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana, "Unraveling the Web services web: An introduction to SOAP, WSDL, and UDDI," *IEEE Internet Computing*, vol. 6, no. 2, pp. 86–93, Mar. 2002.

[22] W. M. P. van der Aalst and T. Basten, "Inheritance of workflows: An approach to tackling problems related to change," *Theoretical Computer Science*, vol. 270, nos. 1-2, pp. 125–203, Jan. 2002.

[23] W. M. P. van der Aalst and A. H. M. ter Hofstede, "YAWL: Yet another workflow language," *Information Systems*, vol. 30, no. 4, pp. 245–275, Jun. 2005.

[24] W. M. P. van der Aalst, "Business process management as the 'Killer App' for Petri nets," *Software & Systems Modeling*, vol. 14, no. 2, pp. 685–691, May 2015.

[25] J. L. Peterson, "Petri Nets," *ACM Computing Surveys*, vol. 9, no. 3, pp. 223–252, Sep. 1977.

[26] W. M. P. van der Aalst *et al.*, "Soundness of workflow nets: Classification, decidability, and analysis," *Formal Aspects of Computing*, vol. 23, no. 3, pp. 333–363, May 2011.

[27] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros, "Workflow Patterns," *Distributed and Parallel Databases*, vol. 14, no. 1, pp. 5–51, Jul. 2003.

[28] H. Smith, "Business process management—the third wave: Business process modelling language (bpml) and its pi-calculus foundations," *Information and Software Technology*, vol. 45, no. 15, pp. 1065–1069, Dec. 2003.

[29] T. H. Davenport, "Putting the Enterprise into the Enterprise System," *Harvard Business Review*, vol. 76, no. 1998/07/Jul/Aug1998, pp. 121–131.

[30] P. Soffer, B. Golany, and D. Dori, "ERP modeling: A comprehensive approach," *Information Systems*, vol. 28, no. 6, pp. 673–690, Sep. 2003.

[31] Scott, Judy, "The FoxMeyer Drugs' Bankruptcy: Was it a Failure of ERP?" *AMCIS 1999 Proceedings. 80.*, 1999.

[32] Y. van Everdingen, J. van Hillegersberg, and E. Waarts, "Enterprise resource planning: ERP adoption by European midsize companies," *Communications of the ACM*, vol. 43, no. 4, pp. 27–31, Apr. 2000.

[33] K.-K. Hong and Y.-G. Kim, "The critical success factors for ERP implementation: An organizational fit perspective," *Information & Management*, vol. 40, no. 1, pp. 25–40, Oct. 2002.

[34] J. Cardoso, R. P. Bostrom, and A. Sheth, "Workflow Management Systems and ERP Systems: Differences, Commonalities, and Applications," *Information Technology and Management*, vol. 5, no. 3/4, pp. 319–338, Jul. 2004.

[35] W. M. P. van der Aalst, M. La Rosa, and F. M. Santoro, "Business Process Management," *Business & Information Systems Engineering*, vol. 58, no. 1, pp. 1–6, Feb. 2016.

[36] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," 2000.

[37] B. Johansson and P. Ruivo, "Exploring Factors for Adopting ERP as SaaS," *Procedia Technology*, vol. 9, pp. 94–99, 2013.

[38] A. Kheldoun, K. Barkaoui, and M. Ioualalen, "Formal verification of complex business processes based on high-level Petri nets," *Information Sciences*, vols. 385-386, pp. 39–54, Apr. 2017.

[39] J. E. Smith and Ravi Nair, "The architecture of virtual machines," *Computer*, vol. 38, no. 5, pp. 32–38, May 2005.

[40] R. Angles and C. Gutierrez, "Survey of graph database models," *ACM Computing Surveys*, vol. 40, no. 1, pp. 1–39, Feb. 2008.

[41] D. Björkgren, "IT-system för små industriföretag," Bachelor's thesis, Unpublished, Åbo Akademi, 2017.

[42] H. van der Aa, H. Leopold, F. Mannhardt, and H. A. Reijers, "On the Fragmentation of Process Information: Challenges, Solutions, and Outlook," in *Enterprise, Business-Process and Information Systems Modeling*, vol. 214, K. Gaaloul, R. Schmidt, S. Nurcan, S. Guerreiro, and Q. Ma, Eds. Cham: Springer International Publishing, 2015, pp. 3–18.

[43] Avdi Grimm, "OSS: Your production software depends on a bunch of people's hobbies - @jessitron," *Twitter*, 21-Aug-2019. [Online]. Available at: https://twitter.com/avdi/status/1163993453467963392?s=21. [Accessed: 23-Aug-2019].

[44] Unknown, *Cavemen too busy.*.

[45] "Workflow Management Coalition." [Online]. Available at: https://www.wfmc.org. [Accessed: 27-Sep-2019].

[46] W. M. Coalition, "Process Interface – XML Process Definition Language." 10-Mar-2005.

[47] Jeffrey Ricker, Mayilraj Krishnan, and Keith Swenson, "Asynchronous Service Access Protocol (ASAP) Version 1.0." OASIS.

[48] "Audit Data Specification." Workflow Management Coalition, 15-Jul-1998.

[49] P. Wohed, W. M. P. van der Aalst, M. Dumas, A. H. M. ter Hofstede, and N. Russell, "On the Suitability of BPMN for Business Process Modelling," in *Business Process Management*, 2006, pp. 161–176.

[50] W. M. P. Aalst, "Three Good Reasons for Using a Petri-Net-Based Workflow Management System," in *Information and Process Integration in Enterprises*, T. Wakayama, S. Kannapan, C. M. Khoong, S. Navathe, and J. Yates, Eds. Boston, MA: Springer US, 1998, pp. 161–182.

[51] W. M. P. van der Aalst and K. M. van Hee, "Framework for business process redesign," in *Proceedings 4th IEEE Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE '95)*, 1995, pp. 36–45.

[52] van der Aalst, W.M.P., Dumas, M., ter Hofstede, A.H.M., and Wohed, P., "Pattern-Based Analysis of BPML (and WSCI)." Faculty of IT, Queensland University of Technology, Brisbane, Australia, 2002.

[53] J. C. Wortmann, "Evolution of ERP Systems," in *Strategic Management of the Manufacturing Value Chain: Proceedings of the International Conference of the Manufacturing Value-Chain August '98, Troon, Scotland, UK*, U. S. Bititci and A. S. Carrie, Eds. Boston, MA: Springer US, 1998, pp. 11–23.

[54] B. Wong and D. Tein, "Critical Success Factors for ERP Projects," 2003.

[55] R. N. Gottumukkala and D. T. Sun, "Modeling and Assessment of Production Printing Workflows Using Petri Nets," in *Business Process Management*, 2005, pp. 319–333.

[56] "About Frappe." [Online]. Available at: https://frappe.io/about. [Accessed: 19-Mar-2019].

[57] The Pallets Project, "Jinja Template Engine," *Jinja*. [Online]. Available at: https://palletsprojects.com/p/jinja/. [Accessed: 14-Aug-2019].

[58] John Resig, "JavaScript Micro-Templating." [Online]. Available at: https://johnresig.com/blog/javascript-micro-templating/. [Accessed: 14-Aug-2019].

[59] Frappe Technologies, “Bench.” [Online]. Available at: https://frappe.io/docs/user/en/architecture. [Accessed: 14-Aug-2019].

[60] “Git.” [Online]. Available at: https://git-scm.com/. [Accessed: 19-Mar-2019].

[61] L. Brehm, A. Heinzl, and M. L. Markus, “Tailoring ERP systems: A spectrum of choices and their implications,” in *Proceedings of the 34th Annual Hawaii International Conference on System Sciences*, 2001, p. 9.

[62] “New App.” [Online]. Available at: https://frappe.io/docs/user/en/tutorial/new-app. [Accessed: 19-Mar-2019].

[63] Joe Barr, “BitKeeper and Linux: The end of the road?” [Online]. Available at: https://www.linux.com/news/bitkeeper-and-linux-end-road. [Accessed: 14-Aug-2019].

[64] DB-Engines, “DB-Engines Ranking of Graph DBMS.” [Online]. Available at: https://db-engines.com/en/ranking/graph+dbms. [Accessed: 18-Aug-2019].

[65] “ERPNext for manufacturers (Make-to-Order.” [Online]. Available at: https://erpnext.com/docs/user/videos/learn/manufacturing-make-to-order. [Accessed: 20-Aug-2019].

[66] “Auto Creation of Material Request.” [Online]. Available at: https://erpnext.com/docs/user/manual/en/stock/articles/auto-creation-of-material-request. [Accessed: 20-Aug-2019].

[67] R. Haesen, S. Goedertier, K. Van de Cappelle, W. Lemahieu, M. Snoeck, and S. Poelmans, “A Phased Deployment of a Workflow Infrastructure in the Enterprise Architecture,” in *Business Process Management Workshops*, vol. 4928, A. ter Hofstede, B. Benatallah, and H.-Y. Paik, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 270–280.

[68] “Redis.” [Online]. Available at: https://redis.io/. [Accessed: 18-Apr-2019].

[69] “Docker: Lightweight Linux Containers for Consistent Development and Deployment | Linux Journal.” [Online]. Available at: https://www.linuxjournal.com/content/docker-lightweight-linux-containers-consistent-development-and-deployment. [Accessed: 18-Apr-2019].

[70] “Production-Grade Container Orchestration.” [Online]. Available at: https://kubernetes.io/. [Accessed: 18-Apr-2019].

[71] “Bench Command cheat-sheet.” [Online]. Available at: https://frappe.io/docs/user/en/bench/resources/bench-commands-cheatsheet.

[72] “Frappe Bench.” [Online]. Available at: https://github.com/frappe/bench#easy-install.

[73] A. Hat Red, “Ansible is Simple IT Automation.” [Online]. Available at: https://www.ansible.com. [Accessed: 18-Apr-2019].

[74] “Working With Playbooks — Ansible Documentation.” [Online]. Available at: https://docs.ansible.com/ansible/latest/user_guide/playbooks.html. [Accessed: 18-Apr-2019].