Industrialization of IT: An Information System Architecture for Application System Landscape Providers

Dissertation

zur Erlangung des akademischen Grades

Doktoringenieur (Dr.-Ing.)

angenommen durch die Fakultät für Informatik der
Otto-von-Guericke-Universität Magdeburg

von: **Johannes Hintsch, M. Sc.**
geboren am 24.12.1985 in Ellwangen an der Jagst

Gutachter:
Prof. Dr. Klaus Turowski
Prof. Dr. Gunter Saake
Prof. Dr. Rüdiger Zarnekow

Magdeburg, 2. Juli 2018

Abstract

Information technology (IT) service providers struggle with efficient and integrated production processes when compared to modern manufacturers. Manufacturers produce products build-to-order in a mass-customization approach or engineer-to-order in a highly customized but streamlined production process while using computer-integrated manufacturing approaches. Software's inherent complexity and its heterogeneous implementations make such consistent management of application service production difficult. This thesis examines if the IT service provider type of *application system landscape providers* can implement a production process similarly efficient and integrated as that of manufacturers. The thesis makes the argument that software for operations automation, such as infrastructure as a service and configuration management software, can wrap application software's complexity and its' heterogeneous implementations. Operations automation approaches facilitate an automated, modularized, and standardized build- and -engineer-to-order production process.

This thesis creates its artifact based on an analysis of current technology, a case study of various IT service providers, including *application system landscape providers*, as well as literature. It follows the design science paradigm of information systems research. The main contribution is an *information system architecture for application system landscape providers* (ISAA). The ISAA explores the limits of standardization, automation, and modularization for application system landscape production. A domain model explicates the relationships between relevant entities of application system landscape production on the layers of business, process, integration, software, and infrastructure. Application system landscapes providers can describe the application system landscapes that underlie the application services they provide to customers using three different models: software, infrastructure, and orchestration configuration models. The ISAA's leading application system, the enterprise management system, treats these models as materials, which facilitates integration with secondary activities such as controlling. The thesis proposes a production execution system that orchestrates the production of application services between the ASLP's enterprise management (i.e., an ERP system), IT service management, and IT service production systems (e.g., infrastructure as a service and configuration management systems), similarly to manufacturing execution systems.

The evaluation follows a recognized methodology. It presents a prototypical implementation of the ISAA after a discussion of the research's relevance as well as the ISAA's consistency, applicability, and adaptability. Comprehensively conducted tests with the prototype show the ISAA's feasibility. Expert interviews validate the overall utility of the ISAA. Targeted companies can improve their application service production in terms of quality and efficiency by leveraging this novel automation- and model-based as well as integrated approach.

Abstract in German

IT-Dienstleister haben im Vergleich zu modernen Herstellern physischer Produkte mit der Umsetzung effizienter und integrierter Produktionsprozesse zu kämpfen. Moderne Hersteller produzieren physische Produkte, die im Rahmen eines Mass-Customization-Ansatzes (Build-to-order) oder in einem hochgradig kundenspezifischen, aber dennoch schlanken, Produktionsprozess (Engineer-to-order) unter Verwendung computerintegrierter Fertigungsverfahren produziert werden. Die inhärente Komplexität von Software und ihre heterogenen Implementierungen erschweren ein solch konsistentes Management bei der Produktion von Anwendungsdiensten. Diese Arbeit beschäftigt sich mit Anbietern von Anwendungssystemlandschaften, eines speziellen IT-Dienstleistertyps. Diese Anbieter sollen ihren Produktionsprozess ähnlich effizient und integriert gestalten können wie sie es bei modernen Herstellern von physischen Produkten sind. Die vorliegende Arbeit argumentiert, dass Software für die Betriebsautomatisierung, wie z. B. Infrastructure-as-a-Service- und Konfigurationsmanagementsoftware, die Komplexität der Anwendungssoftware und ihre heterogenen Implementierungen kapseln kann. Ansätze zur Betriebsautomatisierung können einen automatisierten, modularisierten und standardisierten Produktionsprozess (Build- und Engineer-to-order) ermöglichen.

Die Konstruktion des Artefakts der Arbeit basiert auf einer Analyse aktueller Technologien, einer Fallstudie von verschiedenen IT-Dienstleistern, einschließlich Anbietern von Anwendungssystemlandschaften sowie einschlägiger Literatur. Die Arbeit folgt dem Design-Science Forschungsparadigma der Wirtschaftsinformatik. Der Hauptbeitrag ist eine Informationssystemarchitektur für Anbieter von Anwendungssystemlandschaften (ISAA). Die ISAA untersucht die Grenzen von Standardisierung, Automatisierung und Modularisierung für die Produktion von Anwendungssystemlandschaften. Ein Domänenmodell erklärt die Beziehungen zwischen den relevanten Entitäten der Produktion von Anwendungssystemlandschaften auf den Ebenen Geschäft, Prozess, Integration, Software und Infrastruktur. Anbieter von Anwendungssystemlandschaften können die Anwendungssystemlandschaften, die ihren Anwendungsdiensten zugrunde liegen, in drei verschiedenen Modellen beschreiben: in Software-, Infrastruktur- und Orchestrierungskonfigurationsmodellen. Das führende Anwendungssystem der ISAA, das Unternehmensmanagementsystem, behandelt diese Modelle als Materialien, was die Integration mit sekundären Aktivitäten wie z. B. dem Controlling erleichtert. Die vorliegende Arbeit schlägt ein Production-Execution-System vor, das die Produktion von Anwendungsdiensten zwischen den Systemen zum Unternehmensmanagement (im engeren Sinn ein ERP-System), zum IT-Service-Management und zur IT-Service-Produktion (z. B. Infrastruktur als Service- und Konfigurationsmanagementsystem) orchestriert. Diese ähneln Manufacturing-Execution-Systemen, die Hersteller zur Produktionssteuerung physischer Produkte einsetzen.

Die Evaluierung der Arbeit erfolgt nach einer anerkannten Methodik. Nach einer Diskussion über die Relevanz der Forschung und die Konsistenz, Anwendbarkeit und Anpassungsfähigkeit der ISAA wird eine prototypische Implementierung der ISAA vorgestellt. Umfangreich durchgeführte Tests mit dem Prototyp zeigen die Umsetzbarkeit der ISAA. Experteninterviews bestätigen die Nützlichkeit der ISAA. Durch die Nutzung dieses neuartigen automatisierungs- und modellbasierten sowie integrierten Ansatzes können betreffende Unternehmen ihre Produktion von Anwendungsdiensten bezüglich Qualität und Effizienz verbessern.

Contents

List of Figures

List of Tables

List of Abbreviations

AD	Architecture decision
ADE	Architecture description element
API	Application programming interface
ASL	Application system landscape
ASLP	Application system landscape provider
ASP	Application service provider
BOM	Bill of materials
BTO	Build-to-order
CAMS	Culture, automation, measurement, and sharing
CI	Configuration item
CMDB	Configuration management database
CMS	Content management system
COBIT	Control objectives for information and related technology
CPU	Central processing unit
CRM	Customer relationship management
DSL	Domain-specific language
DSR	Design science research
ERP	Enterprise resource planning
ETO	Engineer-to-order
EVAL	Evaluation step
GB	Gigabyte
GBI	Global bike, inc.
HTTP	Hypertext transfer protocol
IaaS	Infrastructure as a service

ID	Identifier
IDE	Integrated development environment
IEC	International electrotechnical commission
IP	Internet protocol
ISAA	Information system architecture for ASLPs
ISO	International organization for standardization
IT	Information technology
ITIL	IT infrastructure library
ITSM	IT service management
ITSP	IT service production
JSON	Javascript object notation
KPI	Key performance indicator
LAMP	Linux OS, Apache HTTP server, MySQL, and PHP
LOC	Lines of code
LTS	Long time support
OAA	Operations automation approach
OS	Operating system
PaaS	Platform as a service
PES	Production execution system
PHP	PHP: Hypertext preprocessor
PPC	Production planning and control
REQ	Requirement
REST	Representational state transfer
RQ	Research question
SaaS	Software as a service
SCM	Supply chain management
SCOR	Supply chain operations reference model

SCSI	Small computer system interface
SE	Software engineering
SLA	Service level agreement
SLES	SUSE linux enterprise server
SOA	Service oriented architecture
SQL	Structured query language
SRE	Site reliability engineering
TB	Terabyte
TOSCA	Topology and orchestration specification for cloud applications
UML	Unified modelling language
URL	Uniform resource locator
VM	Virtual machine
XML	Extensible markup language

1 Introduction

This chapter starts out with motivating the research project of this thesis. Based on the shortly outlined research gap, the research goal, hypotheses, and research questions are established. Furthermore, previous publications of the author are shortly presented as they share content with this thesis. The thesis outline is presented in the last section.

1.1 Motivation

Managing information technology (IT) and manufacturing enterprises differ significantly. Manufacturers can leverage a knowledge base (Hevner et al., 2004) which has had a distinctly longer time to mature. A production theory (Gutenberg, 1983) is part of the manufacturing knowledge base. It developed due to various reasons. One indeed is that the human senses can experience the raw materials and semi-finished goods which are manufactured into a product. The tangible characteristics of a physical entity, such as wood or a rubber tire, are sufficiently comparable based on their common perception (Dargahi and Najarian, 2004). In other words, the population as a whole share a standard concept of physical entities.

A standard concept in IT, describing what the raw materials, the semi-finished goods, or even the products (finished goods) are, is not self-evident. This is because software has no physical form except bit encodings on a carrier medium and the user interface displayed on a screen. Furthermore, creation of pervasive open standards has been aggravated by companies' reluctance to share their proprietary knowledge. One reason is the fear of losing a competitive advantage (Garud and Kumaraswamy, 1993).

IT as an industry has matured nonetheless. Big players in the marketplace have embraced open source and open standards as part of their strategies (Akkermans and Van der Horst, 2002; Fitzgerald, 2006; Andersen-Gott et al., 2012). This embrace has led to platforms which are comparable to standardized semi-finished goods, such as the Java Development Kit (Egyedi, 2001). Further examples include internet communication protocols (Cusumano, 2010) or the trinity of Hypertext Markup Language, Cascading Style Sheets, and JavaScript, nowadays widely used in web application software development (Charland and Leroux, 2011). Standardization is not limited to the technology layer. From a business perspective, companies' IT is increasingly standardized and servitized by transforming it "[...] from silo business functions to cross-functional organization and from product to service orientation" (Conger, 2010). Subsystems of organizations' information systems, socio-technical systems comprising human and machine components (WKWI, 1994), are the subject of such transformations.

Various frameworks and standards discuss the different aspects of servitization of organizations' information systems. Examples include the common practice framework Control Objectives for Information and related Technologies (COBIT) (Lainhart et al., 2012), the service management system standard ISO[1]/IEC[2] 20000 (ISO/IEC, 2011), or the recently published IT4IT reference architecture (The Open Group, 2017). Amongst them, the IT infrastructure library (ITIL) stands out. Some call it a de-facto standard (Wu et al., 2011) and it has significant adoption rates which, however, differ among geographical regions and company types (Marrone et al., 2014; Yazici et al., 2015).

Seeking to increase the efficiency of IT service provisioning further, researchers (Zarnekow et al., 2006; Beimborn et al., 2012; Erbes et al., 2012) and practitioners (Bell and Orzen, 2010; Betz, 2011; Kim et al., 2013) have studied the industrialization of other industries. In this regard, manufacturing has been of particular interest. This adoption research stream has been termed *industrialization of IT* (Hochstein et al., 2007). Four principles commonly describe it: standardization and automation, modularization, continuous improvement, and concentration on core competencies (sourcing) (Hochstein et al., 2007). In spite of these principles having general validity for IT services, further differentiation is required. In particular, IT services may be *equipment based* or *industrial* in contrast to *people based* or *personal* (Thomas, 1978; Becker et al., 2011; Zarnekow, 2007). Although the principles can be applied to personal IT service provisioning, they are particularly relevant for industrial service provisioning (Zarnekow, 2007, p. 6).

In the later stages of industrialization, IT-tool support became a necessity for manufacturers (Klaus et al., 2000). The concentration on core competencies (Prahalad et al., 1990), fulfillment of regulatory requirements (Liang et al., 2004), and the need for efficiency are factors that required increased integration of companies' processes and resources (Grant, 1996). Material requirements planning systems were first used to calculate the required materials for a varying order volume, and later evolved into enterprise resource planning (ERP) systems (Klaus et al., 2000). However, ERP systems are not the only IT tools used by manufacturers. A variety of tools for computer-integrated manufacturing such as computer-aided design and manufacturing execution systems (Scheer, 1997, p. 93; Romero and Vernadat, 2016) are used in today's application system landscapes for product lifecycle management (Frechette, 2011). Harrison and van Hoek (2008, p. 152) reports how physical products may be fully engineered and fabricated after a customer order has been received just-in-time (engineer-to-order), using such application system landscapes. For IT, aiming at higher efficiency, the vision of a comprehensive ERP system to manage IT service providers' resources has repeatedly been expressed (Hofmann, 2009; Lloyd, 2011, p. 145; Richter and Schaaf, 2011; Betz, 2011, p. xxvi; Glohr et al., 2014). Despite this, resource tracking seems more natural for manufacturing than for IT service providers because manufacturing is inherently parts-based.

Some visions for ERP have been published, and usage of ERP systems by IT com-

[1]International Organization for Standardization (ISO)
[2]International Electrotechnical Commission (IEC)

panies has been documented (Botta-Genoulaz and Millet, 2006). Betz (2011) and ITIL (Lloyd, 2011) take a broad perspective when presenting their visions of ERP for IT. Even though they focus on industrial IT service provisioning, the organizations of their readers may not be very similar. For instance, an internal IT service provider might manage desktop workplaces for its branch employees. Its requirements may be very different compared to a company which offers a web-based customer relationship management (CRM) service. On the one hand, the internal service provider might require functionality to instruct, schedule, support, and control field technicians as well as to procure and track hardware. On the other hand, the CRM provider may have invested in subscription-based billing functionality as well as the integration of its CRM and ticket management systems to facilitate a fast and profitability-aware addressment of feature requests (Botta-Genoulaz and Millet, 2006; Hintsch et al., 2015c).

1.2 Hypotheses, research goal, and research questions

Considering extant literature, a standard ERP software for all of IT is still a vision. Therefore, in this thesis, the scope is narrowed to a specific type of IT service provider. It is exemplified that the ERP concept can comprehensively be implemented to achieve utility for IT service providers. A specific type of IT service provider is chosen to accomplish this: companies which offer application services to their customers. They will be referred to as *application system landscape providers* (ASLP). The more general application service providers (ASP) are usually referred to as companies offering customization services for their hosted standard software packages over a wide area network (Susarla et al., 2003).

A dominant product design characterizes mature industries, such as the automobile industry (Kastensson, 2014). Certain standards, or dominant designs, also exist on different layers of the IT stack. For instance, infrastructure as a service (IaaS) offerings are highly standardized. Analysts predict that by 2019 90% of IaaS providers will have been driven out of the market by the two dominating players Amazon and Microsoft (Gartner, 2017c). IaaS can be considered to have a dominant product design. IaaS offerings are often standardized regarding payment models and product features such as the processor architecture (Prodan and Ostermann, 2009). However, such a dominant design on higher layers of the IT stack is not discernible due to complex and heterogeneous technology stacks (Eilam et al., 2006; Färber et al., 2011).

Therefore, focusing on infrastructure provisioning appears to be less relevant as a research topic regarding ERP for IT. As there will increasingly be a smaller number of large providers, such research would only address a small audience. On the other hand, companies that provide application services are likely to grow in numbers as customers' application requirements will remain diverse (Riemer and Ahlemann, 2001; Currie and Seltsikas, 2001; Anderson et al., 2013). Application services can be offered in different ways. On one extreme, it can involve a substantial software development effort. In other cases, typical ASP cases, consulting and customization services may be offered in addition to the managed service offerings. ASLPs focus on the production of application system landscapes. In this sense, they are on the opposite spectrum of companies providing highly customized software to their customers. They provide landscapes of different application systems. Managing the production of heterogeneous application system landscapes is not trivial (Eilam et al., 2006). However, ASLPs are selected because their products are less standardized than utility computing services such as IaaS. They do not focus on very individual customer requirements. Other than traditional ASPs, their focus lies on the design, deployment, and operation of complex application system landscapes (ASL) for their customers.

Two hypotheses are addressed within this thesis.

Hypothesis 1
A dominant parts-based product design can be established for ASLPs. These designs are based on software for different operations automation approaches (OAA) (Wettinger et al., 2016), such as configuration management software (Delaet et al., 2010), container-based virtualization software (Bernstein, 2014b), infrastructure as a service (IaaS) software (Mell and Grance, 2011), and orchestration software (Chang et al., 2006).

OAAs can be used to wrap software's inherent complexity and its heterogeneous implementations (Hintsch et al., 2015a). The second hypothesis addresses the endeavor to adopt the efficiencies that can be observed in modern manufacturing companies.

Hypothesis 2
ASLPs can produce their application services based on models and components, as well as automatically, much like physical products of modern manufactures.

H2 may be constrained. ASLPs that employ OAAs will more easily be able to offer a set of standardized and mass-customizable ASLs of relatively low complexity. Highly customized, complex landscapes that are integrated into the grown, heterogeneous information systems of multi-national corporations may be difficult to fit into a production process that is also used for mass-customized solutions. However, in ASLPs' business, there are no clear-cut borders as the similarity to traditional ASPs exhibits. Hence, both build-to-order (BTO) and engineer-to-order (ETO) scenarios need to be supported (Stevenson et al., 2005; Ahmad et al., 2010). A standardized, mass-customizable (build-to-order), and small ASL might evolve into a highly customized, complex, and large ASL. Similarly, new customers will ask for ASLs. Their specific business requirements can lead to a BTO or ETO scenario. If the customer's requirements can be satisfied in a mass-customized approach, the BTO scenario or otherwise the ETO scenario is applicable.

Research goal
The research goal of this thesis is to increase the efficiency of ASLPs' application service production through standardization, automation, and modularization by creating an information system architecture for ASLPs (ISAA).

The ISAA is used to test and validate the Hypotheses 1 and 2. Furthermore, it can be used by companies to adapt their information system by the provisions of the ISAA. The term production, in this thesis, is understood in a broad sense as a value-creating process. It is not narrowly restricted to mean only the fabrication process in manufacturing. Application systems are application software and data (Stahlknecht and Hasenkamp, 2005, p. 226). Within implementations of the ISAA, they should be based on standard application software to avoid development of new software as much as possible.

The ISAA should be constructed in a way that fits in with practical realities. Therefore, three preliminary research questions need to be answered. IT service providers already have application systems in place to support their IT service production. Consequently, to construct an information system architecture that is supported by an application system landscape the following research question (RQ) needs to be answered.

Research question 1
What do application system landscapes of IT service providers look like today?

The requirements of ASLPs have to be addressed if the ISAA shall be of utility to them. Consequently, the following research question has to be answered.

Research question 2
What are the requirements that application system landscape providers have regarding an information system architecture that supports their application service production process?

Several different approaches for automating operations exist (Wettinger et al., 2016). Therefore, the following research question is addressed.

Research question 3
What operations automation approach is suitable for constructing the information system architecture?

This thesis finalizes the research of a five-year research project, in which several publications were created and released to the community. The next section briefly presents these publications and places them into the content structure of the thesis.

1.3 Publications of the author

The first structured literature review was performed to identify research on ERP for the IT service industry (Hintsch, 2013). A second reviewed formalizations of IT service management frameworks. ITSM frameworks often only exist in prose form with different degrees of structuredness which can hinder their systematic and versatile application (Hintsch and Turowski, 2013). The literature on configuration management software was reviewed in a third paper (Hintsch et al., 2016a). Configuration management software is one technology to approach operations automation. The review results are used in the research background chapter 3.

An interview-based multi case study was performed to address RQ1 and RQ2. It inspected the current state of application system landscapes of IT service providers. The content from two papers that present the study's results, as well as results that were not published before, are used in the thesis (cf. sections 4.2 and B). It is reported on how IT service providers support their value creation processes with ERP systems (Hintsch et al., 2015b) and what systems their application system landscapes are comprised of (Hintsch et al., 2016b). A taxonomy of application systems was developed. The employed case study methodology is discussed in sections 2.1 and 4.1. The case study results are presented in section 4.1. The taxonomy of application systems was refined by Hintsch et al. (2017) and is used in section 5.3.

The ISAA was iteratively constructed. The first publication on the architecture (Hintsch et al., 2015c) presented an initial version of a core model, the domain model. That publication focused on a BTO scenario and the deployment step. The early version of the prototype used in that paper and the architecture itself were gradually extended. The second publication on the architecture presents an extract of the version of the architecture that is also presented in this thesis (Hintsch et al., 2018). It includes the differentiation of ETO and BTO but is extended here by a full discussion of the domain model, the process model, and the application system landscape. Furthermore, the evaluation was extended and is presented in more detail.

1.4 Thesis structure

The thesis is structured as follows. Chapter 2 describes the research design. It is aligned with the design science paradigm of information systems research (Hevner et al., 2004; Peffers et al., 2008). Chapter 3 discusses the research background, including related work. It begins with a detailed overview of all of its' sections, setting the works presented in the research background chapter in context to this thesis. Chapter 4 addresses the three research questions. First, application system landscapes of IT service providers are studied. Secondly, the requirements of the architecture are derived from three cases. And last, an OAA is selected for constructing the ISAA. These preliminary investigations lay the groundwork for the ISAA. The ISAA's architecture description in chapter 5 includes a domain model, a process model for application service production, a model of the application system landscape that supports the information system, and specific models for the central production execution system. Chapter 6 presents the evaluation. This chapter also includes the discussion of the thesis' results, including its limitations. A conclusion is drawn in chapter 7. This chapter includes a summary of the thesis, a discussion of the contribution, and finally, possible directions of future work.

2 Research design

Research on information systems is classified into two paradigms: behavioral and design science. Both paradigms aim at improving the efficiency and effectiveness of organizations' information systems and their usage. Under the behavioral science paradigm, scientists build theories using empirical or qualitative data. Such theories may explain phenomena regarding information systems in organizations, provide insights to organizations' executives, and guide their decision-making process. In contrast, design scientists engineer artifacts to improve the efficiency and effectiveness of organizations' information systems. (Hevner et al., 2004)

This research project employs the design science paradigm. The information system architecture (ISAA) is its artifact. Two scholar groups who discuss design science are frequently[1] cited in the literature: Hevner et al. (2004) and Peffers et al. (2008). Hevner et al. (2004) provide a general framework describing what the essential elements of design science research (DSR) projects are. Researchers are advised in conducting sound DSR as well as in communicating research to their audience. Peffers et al. (2008) propose a process that guides through the necessary activities of DSR. Both articles are used as a basis for this thesis' research design. Figure 2.1 shows the DSR framework by Hevner et al. (2004), instantiated for this research project.

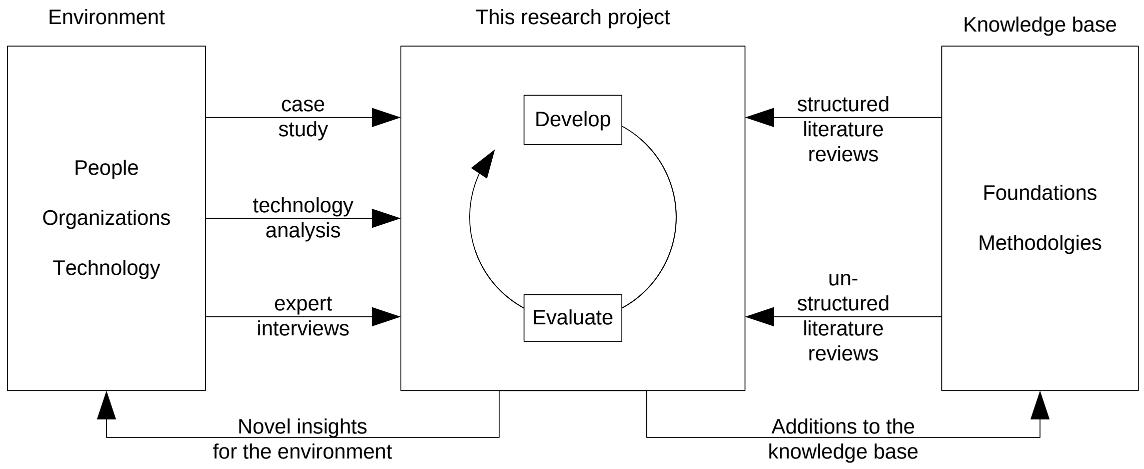


Figure 2.1: Research framework of this research project, based on Hevner et al. (2004).

The project started with a general topic description and then entered a continuous

[1]Hevner et al. (2004) are reported to have 9455 citations (https://goo.gl/cvvmfP) and Peffers et al. (2008) are reported to have 2646 (https://goo.gl/KRT6Y1). Citation numbers were recorded on August 13, 2017.

loop of concept development and evaluation. The loop started with setting the research scope and justifying it. It continued with building the information system architecture and its evaluation. Moreover, finally, the thesis at hand documents the results of the research project. The arrows in Figure 2.1, leading from environment and knowledge base to the central rectangle, represent the input for the research project. The research was conducted to provide novel insights into the environment, particularly to IT service providers. The results are added to the knowledge base in the form of this thesis and preceding publications.

The artifact in this thesis is the information system architecture. ISO/IEC/IEEE (2011, p. 2) define a system architecture to be "[...] fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution." The information system architecture that is proposed needs to be applicable to real-world situations. Therefore, any architecture decision needs to be justified by some rationale. A vital rationale, for example, is that concerns of relevant stakeholders are addressed. Consequently, an architecture decision pertains to one or more concerns of relevant stakeholders. Some decisions, of course, may raise new concerns, and not all decisions can practically be recorded (ISO/IEC/IEEE, 2011, p. 15).

To derive concerns from potential stakeholders for the information system architecture for ASLPs, a case study amongst IT service providers was conducted. The case study's design will be described in more detail in section 2.1.

An analysis of relevant technologies is required because the ISAA makes prescriptions as to how technologies should be employed. Their analysis was performed by studying selected technology products. Also, information on these technologies was derived from literature. Foundations and methodologies (e.g., for case study or structure literature review) were derived from the knowledge base as well. This deduction was made by reviews of the extant literature. The design of the literature review is presented in 2.2.

The additions to the knowledge base and the value for the environment will be discussed in section 7.2. In the next section, the design of the case study, which was conducted in this research project, is presented.

2.1 Case study design

It was necessary to study the environment in which the ISAA is supposed to be implemented to be able to construct it adequately. However, in the studied literature no sufficient account of how IT service providers use integrated application systems could be identified. Therefore, an exploratory case study approach was adopted.

The case study was conducted by following the methodology outlined by Eisenhardt (1989). It started with building a case base, and then cases were sampled to answer specific questions. All data had to be collected, processed, and be pre-analysed. More data analysis and triangulation were performed to answer specific questions (Eisenhardt, 1989).

Table B.1 gives a quantitative overview of the studied cases. Following the European Commission (2005, p. 14), companies of all relevant sizes were selected. Micro companies were not considered as they are not within the scope of the ISAA. Its implementation would likely exceed the capabilities of companies of that size. In addition to IT service providers, ERP software vendors were included in the case base. As their products often include process templates (Klaus et al., 2000), they were assumed to have valuable expertise. Only vendors that specifically addressed the business of IT service providers were included. All companies headquarters were located in Germany.

Case samplings were performed, and results were previously communicated (Hintsch et al., 2015b, 2016b). Hintsch et al. (2015b) inquired about the use of ERP systems by IT service providers. Hintsch et al. (2016b) identified a general structure of the application system landscapes of IT service providers. These results will, first, be discussed in section 4.1. There, rationales are provided for architecture decisions regarding the composition of the ISAA's application system landscape. Furthermore, a case sample is used to derive stakeholder concerns and define architecture requirements in section 4.2. This case sample only includes cases, where application system landscape provisioning is a core part of the business model. Table B.2 gives an overview of all cases, and table B.3 provides a mapping to their use in the specific samplings.

Data was mainly collected through semi-structured expert interviews. This approach was chosen due to the study's exploratory nature. Following the argumentation of Kvale and Flick (2007), this allowed for more flexibility. Furthermore, the course of the interview could be adapted to areas where the interviewee was able to share very relevant knowledge. Interviews were conducted by telephone and by one interviewer. Two interviewers conducted all interviews. First, contact with the interviewees was initiated by sending their companies a half-page description of the research project. Companies were asked to provide experts from the executive level if possible. Their insight into strategic decisions would be of particular value (Kern et al., 2002). However, representatives from middle management or IT architects were also interviewed. As these groups are often responsible for evaluating and selecting IT solutions (Willcocks and Fitzgerald, 1993), their insights would also be of value to the study. Interviews were performed between November 2013 and March 2015.

Voice recording was used for all interviews to be able to transcribe them fully. After transcription, the transcripts were given to the companies to get their consent. In some cases, interviewees asked to remove certain parts from the interview, which was acceptable in all cases as no relevant information was taken out. Data was fully anonymized, but the interviewers were ensured that also the anonymized transcripts would not be fully published. Editing quests mostly pertained to enhanced anonymity requests. In contrast to the expert interview, the interviewees of the case study specifically reported on their companies. That is why the cases are only listed descriptively in the appendix, unlike for the fully available transcripts of the expert interviews. Hintsch et al. (2015b, 2016b) show some of the tables that were generated from the transcripts. The catalog of questions for the interviews is presented in section B.2. The interviews centered mainly around the application system landscapes, the internal and external product representation within the companies' application systems, as well as their production process.

When applicable, questions were asked following up on issues, which arose from previous interviews or within the particular interview. Both, IT service providers and ERP software vendors were asked the same questions. The software companies were asked, first, to provide an account of their internal systems. Also, they were asked about the recommended use of their software product.

The next section reports on the employed methodology to find and review the relevant literature for this thesis.

2.2 Literature review design

Literature reviews are essential to any scientific work as they position the work inside the existing body of knowledge. For information systems research, in particular, the case for a structured approach to conducting literature reviews has been made (Webster and Watson, 2002). Because information systems research is often perceived not to have reached the matureness of natural sciences (Gregor, 2006), Webster and Watson (2002) argue that permanent collection and revaluation of previous work is essential. Structured literature reviews also emphasize the reproducibility principle that scientific work should strive to adhere to (Kitchenham, 2007; Seuring and Müller, 2008).

With the emergence of large literature databases capable of full-text search, electronic search is advocated (Kitchenham, 2007). However, scholars also underline the importance of manual search, for instance by scanning relevant journal's table of contents or performing back- and forward searches (Webster and Watson, 2002; Jørgensen et al., 2005). Substantial criticism on a rising number of poorly conducted literature reviews has been raised (Bandara et al., 2015). Structured literature reviews, if conducted properly, may provide a solid basis for starting a research project or summarizing and evaluating an existing field (Webster and Watson, 2002). However, particularly for students and young researchers, it can be challenging to conduct structured literature reviews. Knowledge about a field may not have reached the maturity of senior researchers. In such cases, the scoping and definition of search parameters can be aggravated (Schryen, 2015). Three structured literature reviews were conducted. However, due to the difficulties discussed, the knowledge base was also searched manually to identify literature that was not found in the tightly scoped literature reviews. Unstructured search also involves the techniques that are advocated by literature review scholars (e.g., back- and forward searches, checking literature of central scholars, or browsing proceedings' and journals' tables of content). The difference mostly is that unstructured search is not documented and not consistently carried out (e.g., not checking all relevant publication outlets with a particular search technique). Figure 2.2 gives an overview of the literature reviews that were conducted during this research project.

The first structured literature review (Hintsch, 2013) searched for research on ERP

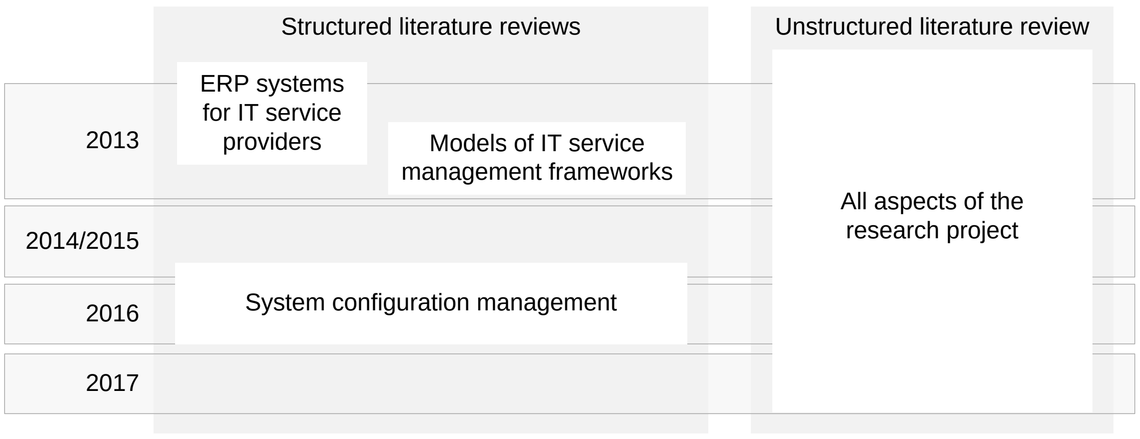

Structured literature reviews
Unstructured literature review
2013
ERP systems for IT service providers
Models of IT service management frameworks
All aspects of the research project
2014/2015
2016
System configuration management
2017

Figure 2.2: Literature reviews conducted in this research project.

systems for IT service provider. Section 3.2.3 will report on some of the findings of the identified literature of this review. The second structured literature review (Hintsch and Turowski, 2013) searched for formalizations of IT service management frameworks. It was conducted because these formalizations could serve as the basis for the data model of the ISAA. The third structured literature review (Hintsch et al., 2016a) identified a substantial amount of publications on and related to configuration management software. Tools for system configuration management were of particular interest in this research project because they were analyzed to construct the ISAA (see section 4.3).

2.3 Summary

As previously stated, Peffers et al. (2008) defined a process for conducting DSR. It consists of the following steps: (1) identify problem & motivate, (2) define objectives of a solution, (3) design & development, (4) demonstration & evaluation, (5) communication. The problem that is addressed by this project was discussed in chapter 1. The research background and research gap are elaborated on in the next section. The objectives of the ISAA are defined in section 4.2 using the results from the case study. There, the practical relevance of the ASLP type is also further established. The ISAA needs to address the concerns of relevant stakeholders, which include the ASLP as an organization, its employees, its suppliers, and its customers. In particular, the ASLP as an organization, employees, and customers are addressed as stakeholders. Their concerns must be analyzed to derive justified architecture decisions. Suppliers are only marginally considered.

The ISAA will be constructed based on the data from case study and literature reviews as well as an analysis of technology. In chapter 6, the ISAA will be evaluated following the framework proposed by Sonnenberg and vom Brocke (2012). The evaluation includes a justification of the research design, the evaluation of the artifact specification, a prototypical implementation and scenario-based testing, and an expert-interview based validation of the fundamental architecture decisions and capabilities of the ISAA. Communication of research is done in form of this thesis.

3 Research background

This chapter is organized into five sections with various subsections to present the different relevant fields of research to the reader. Sections 3.1 - 3.4 present those research fields. Section 3.5 summarizes them and highlights the research gap this thesis addresses.

IT service providers have rarely been the subject of information systems research when compared to other more predominant topics (Hess et al., 2012). For example, no established definition exists what IT services and consequently what IT service providers are (Teubner and Remfert, 2017). This is one reason for defining a specific IT service provider type for which the ISAA is constructed. Therefore, different definitions of IT services and IT service providers are presented in section 3.1.1. IT-related functions within a company may be organized decentralized or centralized, or they may be offered by internal and external service providers (Bergeron, 2002, p. 16). Application service providers, which ASLPs are a specialization of, have received some research attention in particular, in addition to IT outsourcing in general. Success factors of IT outsourcing, including those for ASPs and few other providers, are presented in section 3.1.2. These success factors contribute to the requirements analysis for the ISAA and the answer to RQ2 in section 4.2. In industry, IT outsourcing has been practiced for some time (Lacity et al., 2010). To facilitate the relationships between IT service consumers and providers, for example through standardized IT help desk procedures, various IT service management frameworks have been created by industry consortia and government agencies. These frameworks and their recent developments will be discussed in section 3.1.3. The conceptual origins of the de-facto standard ITIL date back to 1989 (Disterer, 2009). Those origins, centering around the mentioned IT help desk, are still discernible (Marrone et al., 2014). From a research perspective, issues that overlap with ITSM have been studied in the research stream of industrialization of IT. Cloud computing can be seen as a manifestation of some of the principles of IT industrialization (Erbes et al., 2012). Research on the industrialization of IT and cloud computing is presented in section 3.1.4. Cloud computing is relevant to the ISAA both from a technical perspective (e.g., to achieve rapid elasticity) and a business perspective, for example, regarding pricing and service models (Owens, 2010; Iveroth et al., 2013).

A substantial amount of information systems research exists on enterprise systems in general (Schlichter and Kraemmergaard, 2010). Enterprise systems are briefly introduced in section 3.2.1. Factors for the successful adoption of enterprise management systems are relevant to this thesis because the ISAA will be based on such systems. These factors are related to information systems reference models in the sense that they aim at repeatable implementation steps. The factors and reference models are discussed in section 3.2.2. Whereas for other sectors substantial enterprise system-related research exists, few studies focus on how application systems of IT service providers are organized, which is the reason for posing RQ1. The few existing studies are presented in 3.2.3. Observations and prescriptions of the composition of service systems, of which application systems for IT service providers are a subset, have been made on the level of meta-models by constructivist information systems researchers. These meta-models are briefly discussed in section 3.2.4. Also from a constructivist perspective, considerable research has been conducted in the area of applying industrial production methods to IT service production. In section 3.2.5, this transfer-oriented research is presented.

The ISAA should align with current software engineering (SE) process models, and it should be capable of handling application systems of various architectures. Hence, in section 3.3, the current trends in SE process models and system architectures are presented.

Operation of application system landscapes, in this thesis, is based on operations automation approach (cf. Hypothesis 1 and R3). These approaches originate from the need of administering large numbers of computers, for example in grid computing or university labs. Configuration management software is a successor to previously manual administrative work (Talwar et al., 2005; Delaet et al., 2010). This software is discussed in section 3.4.1. At the time when configuration management software first appeared, the variability of computer configurations may have been somewhat limited (Anderson, 1994). However, with the proliferation of IT and IT services, orchestration became a necessity to map various software configurations to infrastructure resources. Orchestration software is discussed in section 3.4.2.

Finally, in section 3.5, the research background is summarized, and the research gap that is addressed by this thesis is highlighted.

3.1 IT service provisioning

Different perspectives exist on what an IT service is. According to Walter et al. (2007), "IT services aim at the effective and efficient satisfaction of the information demand by planning, acquiring and operating IT applications and infrastructure." This definition does not specify whose information demand is supposed to be satisfied. The following section discusses different perspectives that are more elaborate.

3.1.1 IT service definitions and IT service provider types

ITIL defines an IT service to be "[...] provided by an IT service provider. An IT service is made up of a combination of information technology, people, and processes. A customer-facing IT service directly supports the business processes of one or more customers, and its service level targets should be defined in a service level agreement. Other IT services, called supporting services, are not directly used by the business but are required by the service provider to deliver customer-facing services" (Cannon, 2011b, p. 440). Hereby

ITIL expresses what an IT service is composed of, whom it is produced for and by whom, and how its quality goals are agreed upon. The ITIL definition also prescribes a service hierarchization. Both, Walter et al. (2007) and Cannon (2011b) do not make any further ascertainments to the kind of IT service provider or the provided services. Zarnekow (2007, p. 10) and Becker et al. (2011) differentiate IT service providers with industrial and personal service provisioning (cf. section 1.1). Andersen-Gott et al. (2012) also see personal service provisioning as one type of IT service provisioning ("selling IT services such as training, consulting and support").

Teubner and Remfert (2017) do not include personal IT service provisioning in their definition. IT services, according to them, can be characterized by six features: (1) IT services are, first, intangible in the sense that an external factor is required to (2) generate customer value. The external factor can be a customer business process or an information object such as an electronic railway ticket Zarnekow and Brenner (2003). (3) The third feature excludes personal or consulting services. This feature postulates that IT services are provided using an IT-based infrastructure, or as Zarnekow et al. (2006, p. 29) put it, a production infrastructure. A production infrastructure comprises application systems, servers, memory, wide and local area network, as well as workstations (Zarnekow et al., 2006, p. 29). (4) The fourth feature defines services to be provided industrially. Section 3.1.4 further elaborates on the industrialization of IT research stream. (5) Furthermore, services are mass-customized. (6) Lastly, IT services should be provided carefree to the customer. The definition requires that the customer does not come into contact or knows about the technical realization of an IT service (Teubner and Remfert, 2017).

This thesis follows the definition of ITIL. Industrially provisioned IT services are in focus. Application services of ASLPs are equipment-based and thus industrially produced. Personal IT services are included in the ASLP definition when they are provided alongside industrially provisioned application services. Furthermore, application services are often mass-customized, but, they don't always have to be. Custom services can be provided as well. Of course, efficient and effective practices should also be implemented for individualized IT services. Although for some customers it can be beneficial not to be burdened with implementation details, for others such knowledge is necessary. Consider, as an example, services that are reprocessed to higher valued services in a supply chain (Hochstein and Uebernickel, 2006). Not all details need to be disclosed to the customer, but for some customers, it can be necessary to know implementation details, for instance for auditing (Mansfield-Devine, 2014), but also for technical reasons. ASLPs can act as internal or external providers. Internal providers often must support the full range of their organization's business processes. On the other hand, external providers must react to market demand and compete with the offerings in the market (Zarnekow, 2007, p. 31).

The success factors of IT outsourcing are presented next.

3.1.2 Success factors of IT outsourcing

Riemer and Ahlemann (2001); Currie and Seltsikas (2001) study application service providers. ASPs exhibit various differentiating characteristics. For example, ASPs can be specialists or universal providers serving specific or various business functions. Furthermore, Riemer and Ahlemann (2001) differentiate by the degree of standardization, depth of value creation, and consulting effort. According to their classification, individual services do not classify as ASPs. Currie and Seltsikas (2001) define pure-play ASPs which have similar characteristics to those of cloud computing services (Mell and Grance, 2011). For pure-play ASPs, Currie and Seltsikas (2001) name scalability, web-centric application software, and subscription-based pricing models. These criteria are a subset of those defined by Mell and Grance (2011) for cloud computing. Therefore, cloud computing is a progression of concepts rather than a new idea.

So-called *ASP enablers* offer infrastructure and other services in Currie's and Seltsikas' (2001) taxonomy. This layering of value creation has become particularly prevalent with the three cloud computing service models of infrastructure, platform, and software as a service defined by Böhm et al. (2010); Mell and Grance (2011). Very similar to the ASLP business model, Baun et al. (2011) presents landscape as a service providers. Such offerings are "[...] targeted at companies which aim at outsourcing their entire data center including hardware, software, maintenance, and deployment".

Several of the mentioned authors discussed success factors of the ASP business model and IT outsourcing in general (Lacity et al., 2010). Riemer and Ahlemann (2001) list security and privacy as the primary success factor of ASPs. It is followed by a solid backup plan, satisfying individual customer requirements, and enabling customers to switch to a different provider, thus reducing lock-in effects. Furthermore, service quality has to be guaranteed through service level agreements (SLA), precautions against economic fall-out of the provider have to be documented, the provider has to have knowledge of the industry sector (Currie and Seltsikas, 2001), and has to offer full single-source solutions. Client firms outsource to reduce costs, to focus on their core capabilities, access the provider's expertise, achieve business process improvements, or due to technical reasons (Lacity et al., 2010). They often choose not to outsource due to concerns about security and the fear of losing control (Lacity et al., 2010). The studies by Kappelman et al. (2014); Schneider and Sunyaev (2016) indicate that, after a decade, privacy remains an issue for customers, in particular, in cloud computing. The possibility of using private cloud offerings, which better align with company security policies, in addition to public offerings has not entirely convinced all customers (Schneider and Sunyaev, 2016).

Traditional IT outsourcing, in contrast to cloud computing, often exhibits large contracts and custom-tailored IT services. In cloud computing, mass-customized services are often rented by departments rather than through a company-wide structured process (Schneider and Sunyaev, 2016). Looking at the most successful business models in cloud computing, Labes et al. (2017) studied 45 cloud providers. They conclude that specialized cloud providers with customer-oriented industry-sector specific solutions are most successful, which reflects earlier findings (Riemer and Ahlemann, 2001; Currie and Seltsikas, 2001). Providers that enter the market with aggregation services have difficulties competing. Established companies (experienced players) often are solidly competitive. They offer public cloud service with little variance to the mass market and are, also, able to provide good customer support.

Next, IT service management frameworks are discussed.

3.1.3 IT service management frameworks and recent developments

ITIL is recognized as a de-facto standard in the area of ITSM (Wu et al., 2011). ITIL is organized into five publications (Cannon, 2011b, p. X) that are aligned with the ITIL service lifecycle. The lifecycle stages are service strategy, service design, service transition, and service operation. Continual service improvement is applied to the other four stages but is organized in a fifth book (Cannon, 2011b, pp. X and 432–455). Further ITSM frameworks and standards exist. The ISO/IEC 20000 (Disterer, 2009) can be used to certify that organizations have implemented IT service management. ITIL focuses on detailed process descriptions and explains how to implement IT service management (ITSM) . Complementary, the Control Objectives for Information and related Technology (COBIT) (Lainhart et al., 2012) describes key-performance indicators to measure ITSM and other IT-related processes (Cannon, 2011b, p. 395).

While ITIL is in widespread use, it is criticized for its lack of formalization (Hochstein et al., 2005). Consequently, Hintsch and Turowski (2013) reviewed literature that presented formalizations of IT service management. Three works stood out regarding consistency and comprehensiveness. Goeken and Alter (2009) present a COBIT meta-model. It can be used for comparing ITSM frameworks, for example, for completeness. When implementing new processes, the meta-model can be used as guidance to ensure COBIT compliance. In a similar direction, Valiente et al. (2012) develop an ITIL ontology and add semantic constraints. Thereby they can check newly defined process models to be ITIL compliant. Brenner et al. (2009) make a case for the necessity of ITSM application software interoperability. Providers increasingly rely on ITSM software from different vendors. They adapt and extend the Shared Information/Data Model to the domain of IT service management by making it compliant with ITIL and ISO/IEC 20000. Similarly, the IT4IT reference architecture offers an information model to enable inter-tool compatibility (The Open Group, 2017; Andenmatten, 2017).

Investigating how pervasively ITIL is implemented, Marrone et al. (2014) conducted surveys with a total of 623 responses. ITIL's operational level processes, such as incident management, are more likely to be implemented than the tactical/strategic level processes. Earlier versions than the current 2011 edition of ITIL comprised publications on service operation and service delivery, which may explain this tendency. While ITIL originates from a 1989 ITSM publication of the British government the current lifecycle model, including service strategy and continuous improvement, was only introduced in 2007 (Disterer, 2009). Marrone et al. (2014) further explain organizations are more likely to implement operational level processes by citing Lange (2007): “Incident management helps [chief information officers] focus on restoring normal service levels as quickly as possible with minimal disruption to the business. Incident management can also reduce service interruptions in the future, increase the efficiency of in-house IT staff communications and systems in general, and improve user satisfaction”. This observation is also reflected in an industry benchmark that studies the degree to which ITSM software has implemented ITIL processes (Pink Elephant, 2017). While all prior-2007 processes are included, current processes such as Information security management or demand management are not part of the benchmark.

ITIL is criticized by some for its lack of agility, little reflection of current technological developments and accompanied progress in software engineering process models. This can in part be explained its’ origins that date back almost three decades. For instance, Andenmatten (2017) suggests the relinquishment of ITIL’s change advisory boards that expensively had to review individual changes to providers’ IT system landscapes. Automation and virtualization enable software engineers to test their software in production-like environments. No complex quality assurance environments are required (Spinellis, 2012). Automated regression tests can be performed by single developers or in central deployment pipelines (Humble and Farley, 2010, p. 61). These trends are not well reflected in ITIL (Andenmatten, 2017). Continuous delivery and its technical foundations will be elaborated on in sections 3.3 and 3.4.

The increasing convergence of developers and operations is termed DevOps. Developers and operations professionals are required to work closely together to keep up with agility demands of the business (Ebert et al., 2016). Consequently, DevOps is strongly related to agile project management approaches like Scrum (Ebert et al., 2016). While Scrum has some semi-formal guidelines (Dybå and Dingsøyr, 2009), DevOps can be understood as a philosophy of desired organizations’ SE practices such as continuous delivery and supporting technical capabilities (Ebert et al., 2016). DevOps will be discussed in relation to SE process models in section 3.3.1.

Application software, supporting IT service management as defined by ITIL is in widespread use, for example in incident management (Fischer et al., 2012; Marrone et al., 2014; Hintsch et al., 2015b). ITIL states that “[c]urrent tools represent a paradigm shift into the new era of enterprise resource planning (ERP) systems for IT” (Lloyd, 2011, p. 145). As their foundation, ITIL describes databases that are used to manage the data of all ITIL processes. These databases are integrated into a so-called service knowledge management system. Other databases include the configuration management database (CMDB) where information on all configuration items (CI) is stored (Rance, 2011, p. 96). CIs can be software or hardware components, or any other component, that are required to provide a service. In ITIL, no constraint is made concerning the kind of IT service provider for which such systems are suitable. Similarly, Betz (2011, p. 131) states that “[...] a large IT organization needs to have most or all [...]” of the systems he describes. In a multi-case study, Hintsch et al. (2016b) could show that the nature of an IT service provider affects their application system landscape after all. Although different provider types may use similar systems, the systems that are mission-critical differ significantly. Examples of IT service management systems include ServiceNow ITSM or BMC ITSM. In the domain of SAP enterprise management software, the SAP Solution Manager is often used.

IT4IT is designed to be compatible with ITIL. The standard describes the application system components that are required for implementing IT service management functions more specifically than ITIL. DevOps and agility are claimed to be embraced by IT4IT (The Open Group, 2017, sec. 3.3.2)[1]. It is closely related to the provisions made by Betz (2011) who is a key contributor to IT4IT. It is vendor- and process-agnostic (The Open Group, 2017, preface). IT4IT's application system architecture is organized into five levels. Level one to three are part of the standard. Levels four to five are not specified. Software vendors may define how these levels are implemented in their respective products. The standard may achieve interoperability between software products by different vendors as demanded by Brenner et al. (2009). IT4IT's first level provides an overview. It includes four primary activities[2], called value streams: strategy to portfolio, requirement to deploy, request to fulfill, and detect to correct. The application system components are mapped to these value streams, and key data objects are used to represent the information exchange between the components. Level 2 can be understood as a more detailed description of level 1 including relationships between data objects and a more detailed definition of data flow between functional components (The Open Group, 2017, sec. 4.2.3). Level 3 adds scenarios, essential services, and attributes, which key data objects must have at least (The Open Group, 2017, sec. 4.2.6). Scenarios are comparable to groupings of agile user stories (epics). They are used to "explain, enhance, or modify" IT4IT (The Open Group, 2017, sec. 4.2.6). The Open Group (2017, sec. 4.2.6.1) states that the three scenarios "IT Financial Management – IT service cost transparency", "Agile Scenario – DevOps[...]", and "Multi-vendor Incident Management" are currently in development. They have not been published yet. Essential services support the scenarios and can, for example, be implemented as web services to achieve the discussed interoperability between different vendors' software products (The Open Group, 2017, p. 4.2.6). Those services have, however, not been defined yet (The Open Group, 2017, sec. 4.2.6.3). IT4IT does not make any statements as to which providers should use it, but a focus on companies developing their own software under the DevOps philosophy as well as those employing ITIL's incident management is evident.

Gartner (2017b) uses the term IT service support management tools for ITSM software and also points to their origins as IT help desk tools. This origin was also reflected in the interviews from the preliminary case study (see sections 2.1 and B.1) where respondents often put the IT help desk or ticketing software on a level with "our ITSM

[1]The current version of the IT4IT reference architecture is available online at pubs.opengroup.org/it4it/refarch21/, last accessed on January 16, 2018. Consequently, references can only be made to section not page numbers. Sections are abbreviated with *sec.*

[2]IT4IT aligns with the value chain by Porter (1985, p. 37) as was done by Betz (2011, p.45) and Hintsch et al. (2015b).

tool". This statement is not surprising due to the discussed initial focus of ITIL initial on these process areas. Nonetheless, ITSM software products have evolved and now include process support for most of ITIL-defined processes (Pink Elephant, 2017). A trend can be observed that companies like ServiceNow include traditional enterprise management functionality in their ITSM software (Hintsch et al., 2016b). Most of these ITSM software products have integration points with ERP systems. Seamless project and resource management are advertised for SAP solution manager as benefits when integrating with SAP ERP (SAP SE, 2015b). ServiceNow ITSM and BMC ITSM both have interfaces to ERP systems of large vendors, either file or web service based (ServiceNow, Inc., 2015b; BMC, Inc., 2017). Systems like ServiceNow ITSM, a software as a service offering, are trying to become the leading systems in IT service providers' application system landscapes by integrating traditional ERP functionality like human resource or financial management into their portfolio (ServiceNow, Inc., 2015a).

As previously mentioned, ITIL is criticized by some for its rigidness (Andenmatten, 2017). Mansfield-Devine (2014) describes the difficulty in recording all changes made by employees to the IT systems of a company. Comprehensive and consistent change management is advocated by ITIL (Rance, 2011, p. 60). Employees easily divert from regulations to document small changes. To identify the cause in cases of problems, Mansfield-Devine (2014) suggests monitoring all changes made to the application system landscape automatically. Configuration management software, which is discussed in section 3.4, can assist in this attempt to help with audits and problem analysis.

The next section discusses the industrialization of IT and cloud computing.

3.1.4 Industrialization of IT and cloud computing

Within this section, the very limited empirical research on industrialization of IT will be discussed. According to Hochstein et al. (2007), four principles of industrialization should be examined from the perspective of IT service provisioning: standardization and automation, modularization, continuous improvement process, and the concentration on core competencies. In particular, the concentration on core competencies is a principle that is manifested in cloud computing, which is also discussed.

Becker et al. (2011) conducted a study amongst ten IT service providers based in Germany. Cloud computing is seen to have substantial industrialization potential by one of the companies. Regarding product standardization, the majority of companies offers IT services as clearly defined products. ITIL is used for process standardization by the majority of the cases, among other frameworks for process quality and project management. Products are modularized, but in some cases, customer requirements are too individual for mass customization. The service provisioning is reported to be increasingly automated, on average it is at 47 %, but respondents saw room for improvement to automate processes. Regarding capacity management, methods from IT service management are more predominant than from physical goods manufacturing such as production planning and control (PPC) . The companies mainly source hard- and software and only a minority

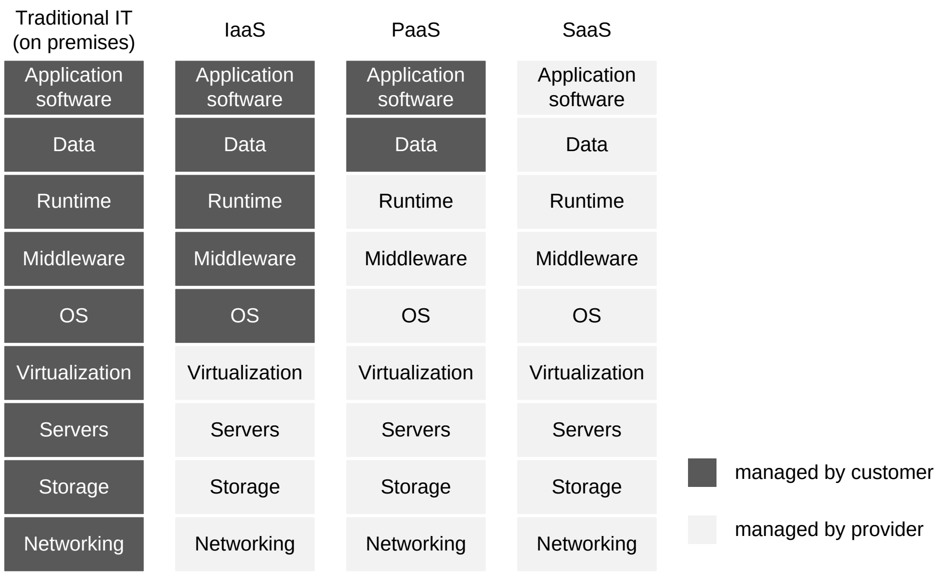


Figure 3.1: IT stack in different service models, based on Stallings (2017).

obtain IT services from other providers. The quality of services is measured by some of the companies who adopt and use standards-based quality management frameworks. Becker et al. (2011) identify the potential for improvements in the areas of sourcing, automation, process standardization, and modularization.

First journals using the term cloud computing appeared in the year 2007 (Yang and Tate, 2012). As stated the concentration on core competencies is particularly observable in cloud computing (Mell and Grance, 2011), but has of course been observable in traditional IT outsourcing for decades (Lacity et al., 2010). The cloud computing intertwines business and technical service concepts. In traditional outsourcing, the provider and consumer usually maintain a close relationship. In cloud computing, relationships can be short-lived and subscription based. Offered services range from public and private infrastructure services (e.g., Microsoft Azure[3]) to human resources services (e.g., Amazon Mechanical Turk[4]). Therefore, the term everything as a service is sometimes used (Baun et al., 2011, p. 17). However, cloud computing is mostly associated with the service models infrastructure (IaaS), platform (PaaS) , and software (SaaS) as a service. Figure 3.1 displays these models in contrast to the traditional on-premise model. The figure shows that a customer can choose to outsource different layers of his IT stack to a cloud service provider.

The technological foundations of the different deployment models will be further described in section 3.4. Significant benefits associated with cloud computing such as elasticity and self-service (Owens, 2010; Hsu et al., 2014) are achieved based on these technological foundations. Reduced costs, another major benefit (Hsu et al., 2014), is also made possible by virtualization and other technological foundations, but also by broader trends such shared services (Bergeron, 2002) and *pay-per-use* (Armbrust et al., 2010; Iveroth et al., 2013). In shared services, customers can profit from reduced costs due to economies of

[3]https://azure.microsoft.com
[4]https://www.mturk.com/

scale on the provider's side. The provider concentrates know-how and equipment to provide services and therefore may be able to offer lower prices at better or equal quality as compared to providing such services in-house.

Pay-per-use pricing models can be attractive in scenarios with varying demand or in situations where full investment decisions are difficult. Consider the example of a provider who wants to try out a new technology that requires more potent hardware. The pay-per-use model enables them to test scenarios without having to invest a sizeable up-front sum. Iveroth et al. (2013) proposes a five-dimensional pricing model that can be applied to products and services. It can also further differentiate cloud computing services. The dimensions are scope (package to attribute), base (cost, competitors' price, and customer value), influence (e.g., price list or negotiation), formula (e.g., fixed price regardless of volume or per unit price), and temporal rights (e.g., subscription or pay-per-use). Particularly relevant for cloud computing are pricing models where temporary access is granted pay-per-use-based (Mell and Grance, 2011).

Hsu et al. (2014) suggests that the adoption of cloud computing is still at its initial stage with 30 % of the studied companies having adopted SaaS, five % PaaS, and 13 % IaaS. Nonetheless, Ng et al. (2017) estimate that in 2017 the revenue of the public cloud computing will have further grown by 18.5 %. Hsu et al. (2014) found that firms with higher IT capability, regarding number of IT employees and IT budget, are more likely to use cloud computing. in contrast to cloud providers' advertisements and popular belief, cloud computing is often not a viable option for small and medium-sized enterprises Hsu et al. (2014). Companies that have a high degree of IT capability and therefore a particular size above that of a small and medium-sized enterprise are more likely to adopt cloud computing.

In the next section, the research background on application systems for enterprises in general and IT service providers, in particular, will be presented.

3.2 Enterprise management systems

Systems "[...] are man-made and may be [comprised of] one or more of the following: hardware, software, data, humans, processes (e.g., processes for providing service to users), procedures (e.g., operator instructions), facilities, materials and naturally occurring entities" (ISO/IEC/IEEE, 2015). Application systems comprise application software and the respective data (Stahlknecht and Hasenkamp, 2005, p. 226). The application software is written for a specific application field such as accounting or human resources. Within this thesis, the term application field will be used synonymously with business function. A business function can be nested (Weske, 2012, p. 75). Consequently, application systems can support larger application fields or just therein contained business functions.

Traditionally, application software and system software are differentiated (Laudon and Laudon, 2005, p. 203). This distinction is not always expedient though. For example, consider the case of a hypervisor, "[...] computer systems that present a very basic user program interface-one which is so nearly identical to a particular computer machine interface that an operating system intended to support such machines may serve as a hypervisor user program without software modification" (Hendricks and Hartmann, 1979). Hypervisors may be considered as system software, "[which manages] the computer's resources, such as the central processor, communications links, and peripheral devices" (Laudon and Laudon, 2005, p. 203), p.203). However, the hypervisor is also essential to the business function of providing virtualized infrastructure services (see section 3.3.2). System software can refer to software like operating systems or database management systems (Stahlknecht and Hasenkamp, 2005, p. 227). Application systems may in some cases be in such tight dependency with system software, or even hardware that system software and hardware are included in the definition of application systems in the broader sense (Stahlknecht and Hasenkamp, 2005, p. 227). However, within this thesis, this broader definition will not be used. System and application software will be mentioned separately.

The next section will briefly define the established research area of enterprise systems.

3.2.1 Definition of enterprise systems research

Enterprise systems integrate all business functions of an enterprise as well as its resources into a common database to support transactional and analytical purposes (Davenport, 2000, p. 2). Such systems provide managers with the comprehensive view of the company needed for decision-making (Davenport, 2000, p. 2). Gartner (2017a) define these systems to be able to "[...] control all major business processes in real time via a single software architecture on a client/server platform [...]." Most often this idealized definition does not reflect reality. For example, different products by companies such as SAP or Microsoft exist for financial planning, customer relationship management, or business intelligence. Literature also reflects this differentiation (Botta-Genoulaz et al., 2005; Chen and Popovich, 2003; Møller, 2005). Vendors usually sell enterprise software as standard software packages. These packages target an anonymous market and therefore need to be customized for each customer (Klaus et al., 2000). In contrast, custom software is developed for an explicit use case by the IT department of a company or an IT service provider (Rautenstrauch and Schulze, 2003, p. 282). It is a trade-off for a company to choose either standard or custom software. Standard business functions might match the provisions of standard software, while other functions might be critical for competitive advantage and therefore require custom software (Rautenstrauch and Schulze, 2003, p. 282).

Enterprise software is often also referred to as ERP software, although this term is criticized by some (Davenport, 2000, p. 2). The criticism stems from ERP's origin in material requirements planning and that it is usually an umbrella term rather than a specific software (Klaus et al., 2000). Within this thesis, ERP, CRM, or SCM[5] are combined under the term enterprise management. In section 4.1, the application system landscapes of IT service providers are investigated, and enterprise management software is delimited from other application software used by IT service providers.

[5]Supply chain management (SCM)

Defying criticism, the term ERP term is pervasively used to identify a significant research stream within information systems research. Several structured literature reviews have been conducted to summarize the state of the art (Esteves and Pastor, 2001; Botta-Genoulaz et al., 2005; Esteves and Bohórquez, 2007; Schlichter and Kraemmergaard, 2010; Eden et al., 2012). Schlichter and Kraemmergaard (2010) state that the research stream has reached a certain level of maturity with a large body of accumulated academic knowledge in different categories. ERP implementation is the category which is researched most (Eden et al., 2012). In the implementation category, factors of adopting enterprise management systems are most often discussed. Information system reference models have particularly been researched by German DSR-oriented information systems academics (Scheer, 1997; Becker and Schütte, 2004; Fettke and vom Brocke, 2016).

3.2.2 Factors of adopting enterprise management systems and information system reference models

Organizations, according to Gronau (2010, p. 12), implement enterprise management systems due to various reasons. The systems decrease cycle times. Orders, for example, can be processed faster due to a reduced data management effort based on a common database. The financial situation can be advanced with stock control and an automated dunning process. Vendor-provided so-called best practice process templates can improve business processes.

Regarding templates, design science-oriented information systems research has contributed reference models to the knowledge base. These models provide guidelines on how enterprise management systems should be structured. They aim to provide repeatable instructions for deriving company specific models, for example, data models. Scheer (1997) provides a reference model for enterprise management systems in manufacturing, including organizational, process data, and functional views. Each view is detailed into three description levels, ranging from business concept to implementation layer (Scheer, 1997, p. 14). Becker and Schütte (2004) provide such a reference model for commerce. How to explicate the general validity of a reference model is an unsolved research issue (Fettke and vom Brocke, 2016). Although general validity cannot be proven, reference models can be useful and are adapted in practice (Scheer, 1997; Becker and Schütte, 2004; Fettke and vom Brocke, 2016).

The reference model by Scheer (1997, pp. 91–93) is commonly associated with the Y-CIM model. CIM stands for computer-integrated manufacturing. The Y-CIM model describes the relationships between the logistical subsystems (e.g., order management or material management) and the systems supporting product development and the actual fabrication of those products. In product development, parts can be drafted with computer-aided design models including simulating their performance in their designated application. The fabrication of these parts is finally performed by computer numerical control machines (Goettsch and Tosse, 2013). Computer-aided design tools support the modularized construction of more complex parts. In a mature information system that is supported by comprehensive computer-integrated manufacturing technology the following scenario is feasible. Harrison and van Hoek (2008, p. 152): "[...] Formerly, [the ETO] process had taken two weeks, because a design engineer had to develop an entirely new design from scratch based on the customer order. Designs have now been modularised as a result of the new system: that is, a new design is produced from a few hundred standard 'modules' that are held on file. This can be done by a sales engineer in a matter of hours. If a tender is accepted by a customer, it had formerly taken another two weeks to convert the tender information into specifications and drawings for manufacture. Today, it is possible simply to send the accepted tender information to the shop floor, and to use the set of standard engineering information already held on file to act as manufacturing instructions. The following is a list of the main features of the new [just-in-time] system."

Gronau (2010, p. 12) adds the following benefits of enterprise management systems. General productivity improvements may be achieved, for instance, with better asset liability management. Supply chain management features boost the link with suppliers and customers. The systems can also be used as back-end systems that interface with customer-facing web shops. Furthermore, cross-departmental access to data is improved, which helps in planning and controlling. Finally, communication between the company, its customers, as well as suppliers can be improved.

Ngai et al. (2008); Momoh et al. (2010) identified various critical success factors regarding enterprise management system implementation. These factors include the need to overcome the complexity of legacy systems, a working change management program, general good communication within the company, as well as data management to ensure the accuracy of data. Further factors include rigorous testing, for instance of software integration between legacy and ERP systems, and ensuring that the ERP system fits the business practices and processes of the company. Furthermore, excessive customization should be avoided to be able to manage system complexity and not to exceed budgets.The dilemma of internal integration refers to the phenomenon that, globally, integration is a benefit, but that, locally, requirements may not be met as a result of integration. Also, hidden costs (such as those for data conversion, training, or consulting) or limited training can endanger am enterprise management system implementation project. Top management support and training and education, according to Ngai et al. (2008), are the most frequently cited critical success factors in literature.

Enterprise management systems promise various benefits. These critical success factors and challenges show that implementing these systems is a costly and challenging endeavor for organizations. Within the next section, the related work on application system for IT service providers is presented.

3.2.3 Enterprise management systems of IT service providers

To identify literature on enterprise management systems for IT service providers and given the importance of the term ERP for this research field, Hintsch (2013) conducted a literature review titled *ERP for the IT service industry.* This review sought to identify research on ERP systems for IT service providers that considers the available ITSM frameworks as a basis for ERP systems' business process templates. Following the methodology of Seuring and Müller (2008), Hintsch (2013) searched for works in all journals, and conference proceedings ranked A and B by Sprecher der WKWI und GI-FB WI (2008). As described in section 2.2 on literature review design, the review searched for works containing keywords from three categories. IT service providers, as explained in the previous section, are a large group of different companies. Their specific business models are referred to in specific terms. Therefore, the first keyword category contained different terms referring to IT service providers (e.g., infrastructure or platform service provider). In the previous section, the predominance of ITIL as a common practice framework was mentioned. However, various other frameworks have been mentioned in literature (Hintsch, 2013). Therefore, the second keyword category contained the names of all identified ITSM frameworks (Hintsch, 2013). The third category contained different terms referring to enterprise management systems. Only six individual works could be identified. Amongst them, the same research group authored five and only one was an individual research paper.

The research group contributing the five works was part of a competence center at the University of St. Gallen (Ebert, 2009, p. i). Between the years 2007 and 2012 several dissertations and peer-reviewed publications relevant to this thesis were published (Ebert et al., 2007; Ebert, 2009; Vogedes, 2011; Dudek et al., 2012; Pilgram and Vogedes, 2012). They will be discussed in section 3.2.5. Wittgreffe et al. (2006) authored the individual work identified in the review. They report on the architecture of the application system landscape in operation at British Telecommunications. The landscape features a common data model. With its system, British Telecommunications automates activities such as service provisioning, change management, billing, and incident management, although it does not become clear to what extent. To support these different activities, specific platforms exist, many consisting of commercial off-the-shelf software, that are integrated with each other following a service-oriented architecture approach (cf. section 3.3.2). The customers of British Telecommunications require evidence of best practices. Consequently, processes are designed to be compliant with ITIL, particularly regarding service operation. Although a broad range of services is described to be supported by the authors, network, and infrastructure services are in focus. While the architecture is described holistically, discussions concerning the specifics of the data model or implementation are missing.

From a behaviouristic research perspective, Botta-Genoulaz and Millet (2006) investigated the ERP system adoption by service companies. Out of six cases, one software company as well as one telecommunications and internet service company were studied. The former used SAP ERP for material management, sales and distribution (essentially billing), finance and accounting, workflow functionality, and CRM. Also, the company utilized specific software for service delivery and software development. The latter used PeopleSoft for accounting and a separate industry-specific software system for supporting operations. The authors conclude that human resource management is rarely fully integrated, although full integration is one of the main benefits of ERP systems. According to

Botta-Genoulaz and Millet (2006), such seamless integration, as witnessed in ERP systems of production firms, is unintended by the studied companies. However, such integration is required to reach a maturity level of 3 of 5 as defined by Richter and Schaaf (2011). On maturity level 3, "[...] most important processes [need to be] supported by tools and the central tools [need to be] interconnected in a way that they can share data in an efficient way [...]" (the term tool is used synonymously with application system). Level 4 requires comprehensive basic tool-support and that "[...] central processes can work very efficiently because of an optimized tool landscape" (Richter and Schaaf, 2011). The highest level of maturity is reached when "all tool areas are fully implemented, and processes can benefit from optimal tool support [and] continual improvement of the tool landscape is also addressed" (Richter and Schaaf, 2011).

More generally, service system meta-models address the integration of different actors (software, humans, and organizations) in service systems. These are briefly presented in the next section.

3.2.4 Service system meta-models

Übernickel et al. (2006) address the service engineering process specifically and suggest an approach for information systems service engineering, including an IT service meta-model, but no illustration of how instantiations of this model can be used in the respective steps is provided. Böhmann et al. (2003) proposes a concept to build modular services. They emphasize the importance of module boundaries and interfaces. However, their modules are rather abstract like "application support" or "ERP operations and backup". A domain-specific language (DSL), including a meta-model, for IT services is provided by Frank et al. (2009). Its goals include the analysis of information system landscapes, the development of IT management tools for building IT services, as well as the usage of the models to monitor IT services during operation. The DSL is integrated into a framework including other layers of enterprise architecture (Frank, 2014). The DSLs focus on providing graphical modeling to users with different technical knowledge to achieve a better business / IT alignment. As such, the DSL is more intended to document an IT system landscape than to plan or design it.

A very generic meta-model for service systems is provided by Alter (2011). He relates entities that interact in service systems. Such as actors might be customers and providers. Customers can buy products or services that are produced by activities using resources. However, there is no elaboration as to what architecture service systems should have, how products are structured or what resources are used to support the product creating activities. A more specific meta-model that could be used as a basis for a data model for an enterprise management system for IT service providers is presented by Ebert et al. (2007) on the basis of (Übernickel et al., 2006). It includes entities of four categories: resources, process, output, and customer. These entities correspond to those involved in IT service production (Zarnekow, 2007; Teubner and Remfert, 2017). The domain model of the ISAA adopts and adapts the model by Ebert et al. (2007).

The next section describes research of applying industrial methods, mainly from manufacturing, to IT service production.

3.2.5 Industrial methods for IT service production

Applying methods from manufacturing to problems of IT operations was done as early as 1991. Lebrecht (1991) suggested the applicability of PPC systems to the operation of data centers. He demonstrated how scheduling of batch jobs and coordination of printing documents could be performed. The idea of reusing PPC functionality and ERP systems was taken up again by authors like Ebert (2009) and Pinnow (2009). They apply PPC to industrial IT service provisioning. Both of these research works, as well as those of Vogedes (2011) and Dudek et al. (2012), positioned themselves in the IT service production management framework of Zarnekow (2007). In consequence, the works by Ebert (2009), Vogedes (2011), and Dudek et al. (2012) share a common understanding of the structure of IT services and the relationship between IT service provider and consumer.

Zarnekow (2007, p. 109) mentioned ERP system and PPC adoption to IT service provisioning but left a detailed concept to subsequent work. Pinnow (2009) provides such a detailed concept. He describes how to use PPC to use the capacity of virtual machines fully. Technically, IT service production is treated like make-to-order process manufacturing (Pinnow, 2009, p. 36). Current scenarios require rapid scaling to demand (Owens, 2010). This requirement makes capacity planning with infrequent PPC runs a poor choice. Capacity management for virtual machines received considerable research attention outside the research stream of industrialization of IT (Jennings and Stadler, 2014). Wuhib et al. (2013), for instance, demonstrate how to manage capacity of state-of-the-art IaaS systems automatically. Such techniques, closely integrated with the hypervisor, appear to be more practical than the proposed implementation with an ERP system's PPC functionality.

Ebert (2009) and Vogedes (2011) address PPC in industrial IT service provisioning, aligned with the framework by Zarnekow (2007), not only on the infrastructure layer but also the application layer. The authors use a transfer-oriented research design and apply production theory to IT service provisioning (Ebert, 2009, pp. 100–107). Ebert (2009, p. 100) maps the entities of the domain of IT service production to master data of the ERP system, as is done in this thesis (see section 5.1.4). Mainly, two kinds of services are considered (Ebert, 2009; Vogedes, 2011). The first service is a managed desktop service where service technicians physically set-up the computer and react to problems that cannot be solved remotely. The other service is an SAP ERP hosting service. IT services are mapped to materials. Physical and virtual servers are individually modeled as equipment because they are treated without an additional abstraction layer. Furthermore, routings are used to describe the steps in detail that need to be carried out to provide a service. Ebert (2009) uses the ERP's PPC functionality mainly to plan and coordinate the different departments involved in deploying and changing an IT service. Ebert (2009, p. 206) integrates machine capacity planning capability to the concept, for example for physical and virtual servers. From a process perspective, the deployment and operation phases of the service lifecycle are supported.

Dudek et al. (2012) adds variant configuration to the PPC concept of Ebert (2009); Vogedes (2011). However, Vogedes (2011, p. 211) is skeptical about the ability to offer different services. Variability can only be achieved based on alternative IT services in the material master. IT services cannot be modified based on the operations contained in the routings. The routings define how a service can be deployed, which therefore is a constraint. Vogedes (2011, p. 210) describes the integration of manufacturing execution functionality and improvement of automation for the concept as possible paths for future work. Pilgram and Vogedes (2012) summarizes work of the St. Gallen university's competence center and states that IT service production aligned against the framework of Zarnekow (2007) can be supported with ERP systems.

In the next section, current software engineering process models and system architectures will be discussed.

3.3 The development towards DevOps and microservices

General issues that have been discussed in software engineering include managing development costs, time-to-market, dealing with incomplete requirements, software quality, as well as software maintenance (Naur, 1969). These issues remain major challenges for software and systems engineering. In the following section, different process models within software and systems engineering will be discussed leading to the DevOps philosophy. Software engineering process models have interdependencies with system architectures. System architectures that pre-dated the currently en vogue micro-services will be described in section 3.3.2.

3.3.1 Software engineering process models

The process of developing software and the question of how it should be structured has attracted researchers as early as 1956 (Boehm, 1988). Early results of this research field were formalized into the waterfall model that proposed a stepwise approach to the software development process. The steps include the assessment of feasibility of the planned software, creation of software plans and requirements, product design, detailing of that design, the actual coding of the different software units, integration of the units into the complete software, implementation of the software in the production environment, and lastly the operation and maintenance of the software in production. At the end of each step, the work results are validated, verified or tested (Royce, 1970).

Current SE process models still include these steps, but the steps are aligned differently. The waterfall model has been criticized, for example, because each step of the process model is performed concerning the full system. If relevant requirements are missed early on, changes in later phases can be costly (Boehm, 1988). To alleviate such risks Boehm (1988) suggested the spiral model, incorporating the steps of the waterfall model as well as the incremental and evolutionary models. The spiral model is based on risk analysis and prototyping. For instance, each cycle includes objective (re-)determination, risk analysis, development, and planning. Only in later cycles will actual development of the operational prototype begin (Boehm, 1988).

Due to increasing failures of projects that employed plan-driven process models (e.g., waterfall or spiral) agile software process models were introduced in the 1990s (Tarhan and Yilmaz, 2014). On the one hand, plan-driven models emphasize that problems need to be specified entirely and planning to be done rigorously. Furthermore, adherence to pre-defined processes and regular documentation is valued by these models (Tarhan and Yilmaz, 2014). Agile models, on the other hand, favor close cooperation between development teams and customers rather than full upfront requirements analysis, software that works rather than extensive documentation, collaboration with customers over the negotiation of contracts, and rather than planning, responding to changes (Tarhan and Yilmaz, 2014). Based on their qualitative and quantitative study results, Tarhan and Yilmaz (2014) state that agile process models lead to increased productivity, lower defect density, faster defect resolution effort ratio, more effective tests, and more accurate effort prediction capability. Agile process models, among others, include extreme programming or the more general project management method Scrum.

Extreme programming focuses on small releases of software, fast feedback, integration of the customer in the development process as well as continuous integration and testing. Limited documentation and pair programming are also emphasized (Tarhan and Yilmaz, 2014). Continuous integration of new features into the main code branch and continuous delivery of these new features into the production environment depends on tool support (Humble and Farley, 2010). In a continuous deployment pipeline, smaller code units of code are checked into the project's code repository. Next, all available unit tests are run, and the code is checked for possible side effects that were introduced by the newly added code unit. High unit test coverage and comprehensive integration tests are essential to detect errors in new releases early on (Beck, 2002, p. 67).

Continuous delivery has received particular attention since new features, requested by the business, can be integrated into the production system without extensive upfront testing (Feitelson et al., 2013). This rationalization enables a swift reaction to business requirements or changes in the market (time-to-market).

Continuous delivery and its continuous deployment pipeline through which code flows from the version control system to production are also part of the DevOps philosophy (Spinellis, 2012). Thus far, it remains a philosophy rather than an established method Alt et al. (2017). With the business' increasing demand for agility, operations teams are challenged to provide stable production environments that can handle the new features coming from development. The two goals of being able to deploy new features frequently and maintain stable and available systems are in conflict (Alt et al., 2017). Hence, DevOps promotes the close cooperation of developers and system operators (system administrators).

Willis (2010) formulated four basic principles for DevOps that got to be known and recognized (Alt et al., 2017) under the acronym CAMS. It stands for culture, automation, measurement, and sharing. DevOps culture means a shift to cross-department collaboration as well as the stronger integration of the customer into the process (Alt et al., 2017). The stronger relationship with the customer highlights some of DevOps foundations in the agile methodologies (Willis, 2010). All DevOps teams are required to take increased responsibility for the final product and customer satisfaction (Alt et al., 2017). Automation is the second principle, and this principle is a core theme of this thesis. The continuous deployment pipeline, as well as monitoring and orchestration software, are often mentioned as necessary software products in this regard (Willis, 2010; Ebert et al., 2016).

Ebert et al. (2016) align several specific software products to a continuous deployment pipeline that they call "DevOps architecture". It is shown in Figure 3.2. This specific pipeline includes the software Rake for build and code library management. Rake is used by the development team to combine theirs' and others' libraries with the code of the team's current project. How exactly this combination is performed to build executable programs or shippable libraries, is formalized in build instruction files. The Rake build instruction files, together with the project's code libraries, are stored in the version control system (repository). There, the formalized instructions to build the development, testing, and production landscapes are stored as well. The landscape build instruction files can be scripts, language-based, or model-based configuration specifications (Talwar et al., 2005). The continuous integration server, in this case Jenkins, uses software build instruction files, unit, and integration tests, as well as landscape build instruction files to build and test the project's software. The public IaaS service Amazon Web Service[6] and its orchestration service Amazon OpsWorks[7] together with the configuration management software Chef are used to deploy the team's software into production. The software Nagios monitors the cloud-deployed software system in production. Production errors are reported back to the development team that can respond with fixes in the form of new and adapted code. The operation team might also have to adapt the landscape build instruction files (Chef's so-called Cookbooks). The stable operation and maintenance of the whole deployment pipeline lie in the responsibility of the operations team.

Measurement is the third DevOps CAMS principle. It supports continuous improvement. Alt et al. (2017), report that, implementing the DevOps philosophy at T-Systems Multimedia Solutions has lessened the conflict between required agility and speed on one side and required reliability on the other side. Throughput times for new releases were decreased. Decreased throughput times resulted in 75 % lower costs for customers because development and test landscapes had to be in operation in parallel for substantially less time (Alt et al., 2017). Additionally, New Relic, Inc. (2015) name business success, customer experience, application performance, speed and quality as categories for key performance indicator to measure in DevOps.

Sharing is the last CAMS principle. First of all, sharing ideas, methods, tools, and common goals within organizations cannot be taken for granted. Sharing is strongly rela-

[6]https://aws.amazon.com/
[7]https://aws.amazon.com/opsworks/

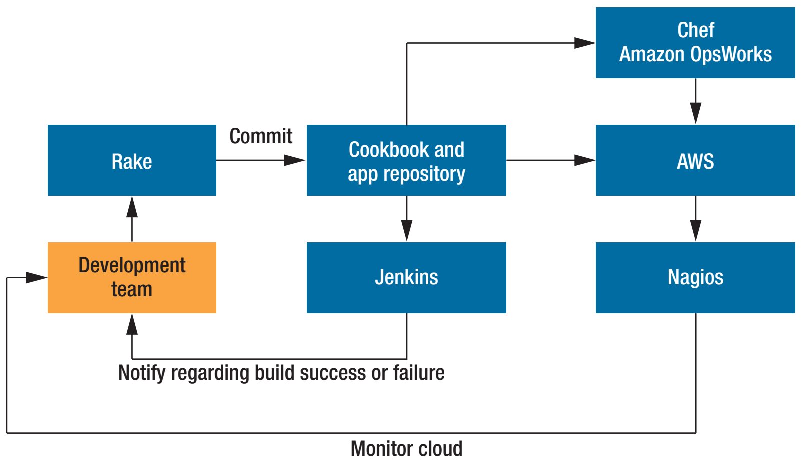


Figure 3.2: Example of a product-specific continuous deployment pipeline for DevOps (Ebert et al., 2016).

ted to the culture of a company (cf. CAMS culture principle). Such an openness, requiring mutual trust, is recommended for DevOps initiatives to be successful (Willis, 2010; Debois, 2011; Alt et al., 2017). Particularly defining and accepting common goals can be a challenging task for development and operation teams to accomplish (Alt et al., 2017). Secondly, sharing can be observed in the use of open source software (Andersen-Gott et al., 2012). It is not far-fetched to say that much of the recent trends including cloud computing and DevOps would not have been possible without open source software. Essential contributions and influences of open source software include easy documentation and testing mechanisms, a modular architecture to support distributed development efforts, an open culture which promotes helping others and finding transparent contribution mechanisms such as facilitated by today's version control systems (e.g., Git in GitLab[8], motivated by GitHub[9]), as well as balancing centralization and decentralization (Gacek and Arief, 2004). Companies may support or provide open source software to sell complementary services within this eco system(Andersen-Gott et al., 2012).

As the DevOps philosophy is rather unspecific, companies must find their interpretation. Site reliability engineering can be understood as a specialization of DevOps. The term was coined by Google where site reliability engineering (SRE) is applied and where it was invented (Beyer et al., 2016). Because Google operates a successful business and internet-based application software on a vast and distributed computing infrastructure, Google's established common practices influence the IT industry (Chen et al., 2009). SRE mainly seeks to address the bespoken goal conflict that developers and operators have (Beyer et al., 2016, p. 4). A site reliability engineer, working in the operations team, will most often have a background in software engineering (Beyer et al., 2016, p. 5). On the one hand, system administrators are often well experienced and capable in the lower levels of the IT stack, mainly when supporting large IT organizations. This experience includes knowledge of the low layers of the network stack or operating system internals (Beyer

[8]https://about.gitlab.com/
[9]https://github.com

et al., 2016, p. 5). On the other hand, software engineers are often more likely and more educated on how to rationalize inefficient processes or on how to automate reoccurring tasks (Beyer et al., 2016, pp. 205-222).

Within SRE, the operations teams will define so-called error budgets. The error budget for an IT service equals one minus the availability target for that service (Beyer et al., 2016, p. 481). If no error occurs, the development team may launch new releases and updates to their software. If an error occurs, releases and updates are frozen until errors are resolved. By default, the budget will be reset monthly. For mature services (with availability targets above 99.99 %) the budget can be bi-monthly. This error budget gives the development team an incentive to work with the operations team to quickly resolve issues and provide software with low bug rates. Also, site reliability engineers are restricted to work on classical operations work (e.g., handling tickets, performing on-call duty, or manual tasks) for a maximum of 50 % of their time. The rest of the time should be spent on ensuring that services run stable and are operable with a high degree of automation (Beyer et al., 2016, p. 6).

Automation is also a key component of current system architectures, such as micro-services. System architectures are discussed in the following.

3.3.2 System architectures

A system architecture states what the system's elements are, how they are arranged and interrelated, what the principles of the system's organization and design are, and how the system shall evolve over its lifecycle (ISO/IEC/IEEE, 2011). This section roughly follows the historical evolution of monolithic software architectures, over component-based software, service-oriented architecture, cloud computing, and finally to microservices. The focus is on software systems. Enterprise architecture is another important field of architecture. It is used to consistently align the business layers (strategy and process) facilitated by an integration layer with the IT layers (software and technology), as discussed by Winter and Fischer (2007). Although enterprise architecture is not discussed in more detail in this section, of course, the architecture of the business has interdependencies with the business as will become evident in the following. The structure of this section aligns with a literature review on microservices by Chlebusch (2016).

The monolithic architecture was the fundamental concept by which early enterprise application software was designed (Klaus et al., 2000). Enterprise application software contains functions for data management, processing, and presentation (3-tier architecture). In monolithic software architectures, these functions are centrally implemented with a high degree of coupling (Fink, 2014). When systems have a certain size they become difficult to manage due to their complexity. Adaptation or replacement of parts of the application system is challenging, and the reuse of these parts is usually not possible (Fink, 2014). Furthermore, an allocation of the application system to more than one computer for concurrent execution is often not possible. Concurrent execution is required for efficiently dealing with increased load. The client-server architecture addresses some of these disadvantages of monolithic architectures (Fink, 2014).

The principle of the client-server architecture is to divide the application system into at least two subsystems (Fettke, 2016). A server subsystem offers its services over defined protocol-based interfaces to a client subsystem. The server may react to client-initiated requests in the order it chooses. Often, the communication between server and client is performed network-based. The client-server architecture addresses some of the issues of monoliths. Functions, such as data management and processing, can now be split into subsystems, which enhances system maintainability (Rautenstrauch and Schulze, 2003, p. 258). Furthermore, load distribution onto more than one computer is facilitated. Application software is composed of multiple programs (Stahlknecht and Hasenkamp, 2005, p. 226). An enterprise application system following a client-server-based 3-tier architecture decomposes its functionality into subsystems. However, also within single programs, decomposition has been continuously researched in software engineering.

Naur (1969) identifies software decomposition as a strategy to achieve better maintainability of software by decomposing it into manageable pieces. Different terms and approaches have been used for decomposable parts and decomposition itself. Naur (1969) refers to decomposable parts as action clusters, whereas McIlroy (1969) calls them software components. Parnas (1972) proposed the modular design by decomposing software design into software units. Finally, Dijkstra proposed dividing software into sequential phases and hierarchical levels (Dijkstra, 1968; Laplante et al., 2008). These principles created the foundations for today's efficient and effective module-based software development. Loosely coupled modules or components, whose elements exhibit a high degree of cohesion (Beck, 2002, p. 107), can be quickly adapted according to requirements without having to implement the functionality.

Business components are specific software components that implement business functions of enterprise application systems (Turowski, 2014). Business components require additional system parts (middleware) to be executed. Such middleware includes work-flow management and communication systems (Turowski, 2014). Workflow management systems support the execution and potential automation of defined business processes (Cardoso et al., 2004). Extracting components from a formerly monolithic architecture into networked subsystems increases communication needs. Systems that handle this communication in business component-based application systems are referred to as object request brokers (Turowski, 2014).

Service-oriented architecture (SOA) is related to component-based architecture. Based on a service registry, consumers can search for and use a service that a provider publishes to the service registry (Gómez, 2012). In contrast to the complex and proprietary enterprise architecture integration platforms centering around the object request brokers, SOA should facilitate higher agility of business processes and improved reusability of software components (Romero and Vernadat, 2016). SOA is often implemented with techniques such as the simple access protocol or enterprise service bus (Siedersleben, 2007). While SOA is usually appreciated as a technical concept, it has organizational implications (Beimborn et al., 2012). If, for example, a company has adopted SOA to design its application system landscape, IT outsourcing is facilitated (Beimborn et al., 2012). Organizing a business architecture in a service-oriented fashion is also a current theme of service science (Alter, 2011).

Fuelled by the increasing demand for computing power, cloud computing became famous in 2006 (Yang and Tate, 2012). "[Cloud computing] promises to provide on-demand computing power with quick implementation, low maintenance, fewer IT staff, and consequently lower cost" (Yang and Tate, 2012). The essential characteristics of services offered via cloud computing are on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service (Mell and Grance, 2011).

Resource pooling is required to achieve lower costs. Consider a company that requires large computational capacities only at particular times in the year. Acquiring the needed computing infrastructure would be costly as compared to procuring those services from an IaaS provider. Resources that are pooled include networks, servers, storage, but also multi-tenant software in PaaS and SaaS offerings (Armbrust et al., 2010).

Virtualization is of central importance to achieve resource pooling (Armbrust et al., 2010). Different forms of virtualization exist. Common to all, an additional abstraction layer is put between the physical computers and the application software. Full virtualization, paravirtualization, processor-supported virtualization, and operating system level virtualization (container-based virtualization) can be differentiated (Hoffmann, 2010). In full virtualization, the virtual machine that is run on a hypervisor exactly resembles real hardware (Hoffmann, 2010). The guest operating system does not know that it is running on virtualized hardware and, consequently, does not explicitly have to be prepared. The hypervisor runs directly on hardware or is running as a program within a deployed operating system. Paravirtualization addresses the performance issues of full virtualization (Hoffmann, 2010). Full virtualization results in a substantial overhead because of complete hardware replication. In paravirtualization, the guest operating system has to be modified. That way specific tasks of the guest can be delegated directly to the hypervisor where they can be executed without suffering from the overhead of virtualization. In processor-supported virtualization, the processor takes over tasks of the hypervisor (Hoffmann, 2010). Compared to paravirtualization the advantage is that the guest operating system does not have to be modified.

In operating system level virtualization or container-based virtualization, containers are not as entirely separated from one another as is the case for virtual machines. Containers share the kernel of the operating system on which they are executed (Hoffmann, 2010). Separation between containers is achieved by using kernel features that carve off a part of the file system (chroot), a contained process space (namespaces), and isolated (cgroups) usage of central processing units (CPU) and memory, as well as isolated disk and network input/output (Pahl, 2015). The software Docker[10], which supports container-based virtualization, has received particular attention recently (Bernstein, 2014b). Based on a

[10]https://www.docker.com/

layered file system, customized containers containing project-specific application software can be built from various container base images that are available in public repositories (Pahl, 2015). Compared to full virtual machines, containers exhibit faster start times and a reduced operational overhead (Soltesz et al., 2007; Mastelic et al., 2016). Their lightweight nature makes them particularly appealing for the cloud computing characteristic of rapid elasticity. New containers can be spawned almost instantly and fine-grained in reaction to quickly changing load scenarios (Bernstein, 2014b; Pahl, 2015).

Automation is key in cloud computing and a requirement for achieving on-demand self-service and rapid elasticity. Broadband internet access is available pervasively in developed countries, with coverage of above 99 % in Europe (European Commission, 2015). Handling the requests of a large number of users requires sophisticated strategies for load balancing at the side of the service provider (Beyer et al., 2016, pp. 223–246). Figure 3.1 shows the different service models of cloud computing (IaaS, PaaS, and SaaS). Self-service can mean that customers use web-frontends of the service, for example in SaaS, but particularly for IaaS and PaaS, the use of application programming interfaces (API) is closely associated with cloud computing (Fehling et al., 2014, p. 4). If an IaaS offering is used to substitute an organization's computing capabilities, following the automation principle, then manually starting and stopping remote virtual machines is infeasible. APIs are used for this purpose.

As cloud computing is considered a hype by some (Leymann et al., 2016), researchers discuss what characterizes software explicitly as cloud-native (Leymann et al., 2016; Kratzke and Quint, 2017). The current architectural style of microservices adheres to the requirements for cloud-nativity (Kratzke and Quint, 2017). Leymann et al. (2016) point out that the core principles of microservice architectures are not new, but were established in SOA. However, microservices add the independent deployability of each service through a deployment pipeline to their architecture concept (Leymann et al., 2016).

Lewis and Fowler (2014) employ nine characteristics to define microservice architectures: (1) componentization via services, (2) organized around business capabilities, (3) products not projects, (4) smart endpoints and dumb pipes, (5) decentralized governance, (6) decentralized data management, (7) infrastructure automation, (8) design for failure, and (9) evolutionary design. Based on Lewis' and Fowler's remarks (2014), they will be described in the following.

Microservices are used to build distributed software. (1) They are small, independent services that act like software components. Such a component is autonomous, replaceable, and extensible. (2) Whereas in monolith-based architecture development teams were often organized around layers such as database and application software, this should not be the case for microservices. There, teams should be assembled around business capabilities. For each business capability, a full stack service is implemented and includes "user-interface, persistent storage, and any external collaborations" (Lewis and Fowler, 2014). This organization achieves a high degree of cohesion within each service. (3) Each team takes responsibility for their services. If a team has built the service, it is also responsible to run it. Usually, a project has a fixed ending date. Then, developers often move on to the next project. The third characteristic shall keep the teams feeling a sense of responsibility for their services, which support the respective business capability. Productization has also been suggested and implemented for IT service provisioning to facilitate more transparent and market-like provider-consumer relationships (Zarnekow and Brenner, 2003; Löffler and Reinshagen, 2014).

Microservices, in contrast to previous architecture styles like component-oriented system architectures, (4) use simple mechanisms for communication. Often, representational state transfer (REST) and JavaScript object notation (JSON) are used for communication (Kratzke and Quint, 2017). In RESTful communication, the state is transferred each time a service is called, which makes it easier to scale out (add nodes) the service (Fehling et al., 2014, pp. 171–174). JSON is a simple markup language without schemas for verification. It can be seen as the smallest common denominator that everybody can agree upon (Koukis et al., 2013).

Monoliths required consistent use of software programming languages and libraries. (5) For microservice architectures, it is recommended that teams select programming languages and tools by the project's specific requirements as well as the teams' preference. (6) Decentralized data management becomes a necessity when each service is a full stack implementation including its own database (cf. second characteristic). Maintaining enterprise-wide data consistency, therefore, is a problem that developers face when developing microservices. However, Lewis and Fowler (2014) state that the resulting drawbacks such as inconsistency and required time for updates are accepted in favor of more business agility. (7) Operations automation approaches are required to operate the heterogeneous software stacks that result from the fifth characteristic. Operations automation approaches (Wettinger et al., 2016) include the different techniques of virtualization and will be further discussed in the following section. Furthermore, rapid service elasticity requires automated procedures to react quickly to changing load situations (Owens, 2010). Also, automated deployment pipelines with automated testing and deployment facilitate the provisioning of new features for a service. (8) Monitoring is as necessary as ever when considering the resulting large infrastructures. At Google, for example, every server program has a built-in simple web server that provides a program's health statistics. Monitoring data can easily be collected from those web servers (Beyer et al., 2016, p. 19). (9) When requirements change, services need to be updated. Updating software while maintaining architectural consistency can be challenging. Here, the previously discussed component-oriented software engineering that modularizes software can help. However, in some cases replacing components rather than refining them is advocated.

This section focused on the architecture of systems and how to engineer them. The next section will shed light on how to automate the operation of the resulting system landscapes.

3.4 Operation of application system landscapes

ITIL defines configuration management as “the process responsible for ensuring that the assets required to deliver services are properly controlled, and that accurate and reliable information about those assets is available when and where it is needed. This information includes details of how the assets have been configured and the relationships between assets.” (Rance, 2011, p. 328). This definition also addresses some of the aspects why configuration management software is used (e.g., reliable information about how an application software is configured). However, ITIL’s common practices for configuration management have a strong focus on managing the configuration items of a service regarding accounting for them in a configuration management database. Configuration management software focuses on how to automatically configure a given number of hosts based on a common configuration specification. To control the sequence of configuration and describe which nodes get what configuration, orchestration software is used. Both types of software are fundamental building blocks of the ISAA. They are discussed in the following two sections.

3.4.1 Configuration management software

Configuration management software has played a crucial role in distributed computing concepts such as grid computing (Fischer et al., 2014). Automation is required in domains other than grid computing as well to cope with the growing numbers of computers. Configuration management software is used to describe and invoke the desired configuration of the software stack of specified computers. Concepts such as modularization and parametrization are used to balance standardization of default configuration specifications against specific configuration necessities of individual computers. In terms of recent trends such as DevOps or microservice architectures, configuration management software is used for automating the configuration in the various environments (development, testing, production) of continuous deployment pipelines (Ebert et al., 2016). This section discusses configuration management software aligned with the results of a structured literature review conducted by Hintsch et al. (2016a).

Various open source and proprietary products for configuration management exist (Delaet et al., 2010). However, software for configuration management, as discussed in this thesis, is not the only means for managing the configuration of software systems. Arcangeli et al. (2015) review work on the automatic deployment of distributed software systems. They address component-oriented software engineering and review technologies such as OSGi that can be used to build software from various interfaced modules (Hall et al., 2011).

Software product lines employ standardization and mass customization (Apel et al., 2013, p. 4) to the efficient production of shippable software programs. The scope of software product lines particularly lies on resource-constrained domains such as embedded computing (Apel et al., 2013, p. 7). Features of a product line, such as transaction management for databases, can be included or excluded as required (Apel et al., 2013, p.

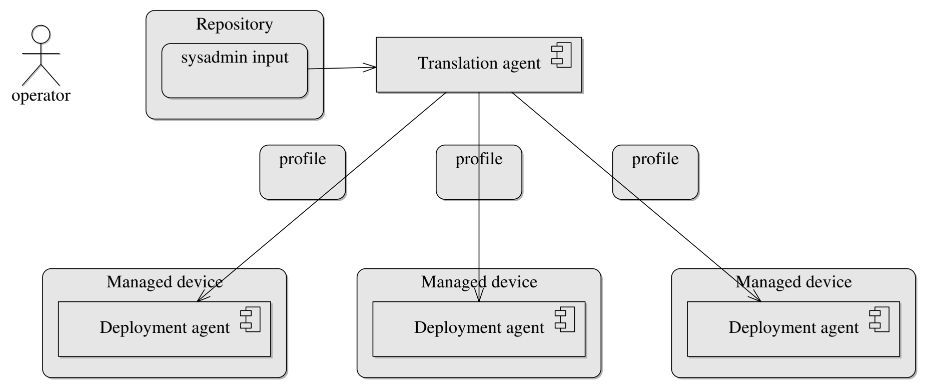


Figure 3.3: Managing the configuration of software on different devices (Delaet et al., 2010).

88). Based on selectable sets of features, a software can be configured, and a shippable software program is generated (Apel et al., 2013, p. 20). Component-oriented software engineering and software product lines both address the development of new software. Configuration management software, in contrast, deploys and configures software on one or more computers and is responsible for the configuration, usually for as long as the computer is running.

Delaet et al. (2010) describe a generic architecture of configuration management software that is depicted in Figure 3.3. The administrator is provided with an interface, which is used to specify the configuration of the managed devices. These specifications are stored in a repository. Device-specific profiles are generated from generic configuration specifications. The deployment agents of the managed devices configure the device as specified.

The first publication on configuration management software appeared in 1994 (Hintsch et al., 2016a). In this paper, Anderson (1994) presented a software to validate and generate configurations for the computers of the computer science department of the University of Edinburgh. Since then, numerous other works have been published on and related to the subject. Hintsch et al. (2016a) provide a categorization scheme for research related to configuration management software. Representative research is discussed below.

Various papers addressed, not surprisingly, cluster and grid computing because large numbers of servers need to be managed. For example, Ballestrero et al. (2014) present an approach where configuration management software is used for simulation experiments at the European Organization for Nuclear Research. In addition, configuration management is an essential area for network management, for example, regarding the remote management of network elements' firmware (Hori et al., 2007). Finally, researchers have presented their work in educational settings, for example, in managing storage systems of libraries (Pop et al., 2014).

Most research interest has been attracted by how the task of configuration is carried out. Here, *three* sub-categories can be differentiated.

The *first* subcategory includes papers that discuss the different techniques of specifying the configuration. Talwar et al. (2005) differentiates script-based, language-based, and model-based techniques. For small node numbers, manual or script-based configuration might be feasible. For complex landscapes, the abstraction mechanisms of language- and model-based configuration management software facilitates management (Talwar et al., 2005). Configuration management languages are often DSLs and sometimes general purpose languages. Delaet et al. (2010) argue that declarative approaches are superior to imperative approaches because they are more stable. Model-based techniques introduce additional abstraction elements as compared to language-based techniques (Talwar et al., 2005).

The *second* subcategory includes works that focus on how configuration specifications can be created. For example, Menzel et al. (2013) present an approach to mine virtual machine image repositories for configuration information. This information is transformed into executable configuration specifications. Herden (2013) proposed model-driven configuration management where configuration specifications are generated from system landscape diagrams. Similarly, the DSL by Frank et al. (2009) and Kaczmarek-Heß and de Kinderen (2017) can be used to model IT system landscapes. However, they make no statements as to how landscapes can be configured based on the models specified in the DSL.

The *third* subcategory describes approaches to high-level configuration specification that are often model-based. For example, Wettinger et al. (2013) describe the Topology and Orchestration Specification for Cloud Applications (TOSCA) and how it can be used with language-based configuration management. TOSCA will further be described in the next section.

Non-functional properties of configuration management software have also attracted various researchers. For example, Vanbrabant et al. (2009) has integrated fine-grained access control into configuration management software (security). Intended or unintended misconfiguration can have disastrous consequences in large sites. The correctness of configuration specifications is addressed by Ruscio and Pelliccione (2014) who proposes to assist administrators by simulating the upgrade of complex systems to predict failures. For administrators without a background in software engineering, configuration specification can be tricky. As an example, Kandogan et al. (2005) proposes a spreadsheet-based framework for system administrators to develop configuration scripts (usability).

Reliability is another non-functional property of configuration management software. Various factors influence the state of a computer's configuration. Programs and humans can change the configuration, and due to software bugs system stability can be impeded. The declarative nature of most current configuration management software addresses this problem. When a certain state is declared the configuration management software inspects the as-is state of the system and converges the configuration towards the to-be state (Nielsen et al., 2011; Hanappi et al., 2016). To ensure that systems are configured as specified, the concept of idempotency is introduced. It states that no matter how often a configuration specification is applied it should always converge to the same state (Hanappi et al., 2016). The fulfillment of this principle is not always trivial, however. Issues may arise from unavailable network connections (Hanappi et al., 2016). Also, implementing idempotent configuration specifications, for example, for the creation of databases can be challenging, but possible (Wettinger et al., 2014). Addressing mostly the network issue, Zhu et al. (2015) suggest to offline preconfigure virtual machine images and only limitedly configure them at runtime. This approach minimizes problems that result from network unavailability in which cases required software packages cannot be downloaded.

An important aspect of operations automation is orchestration, which will be discussed in the following section.

3.4.2 Orchestration software

Orchestration software, as discussed in this thesis, is responsible for instantiating elements such as virtual machines or containers, virtual storage, and network elements. It determines the order of instantiation, configures or invokes configuration of the elements, and can transfer output that results from instantiating and configuring one element as input for instantiating and configuring other elements (OASIS, 2015). Orchestration software can come in the form of standalone orchestration engines as is the case for OpenTOSCA[11]. For the IaaS software OpenStack, which is used for the prototypical implementation of the ISAA, the orchestration engine Heat[12] is integrated as a component into an OpenStack installation. For public IaaS offerings, such as Amazon Web Services, orchestration services are available as well.

Several authors present architectures for automating the provisioning of application services. Kirschnick et al. (2010) use a custom-developed tool landscape to show that automated provisioning of cloud services is possible. They focus only on provisioning. Billing or contractual provisions are not considered. Their approach is centered around service templates. These service templates define how the services are structured using a DSL. End-users of an application service can customize these templates to align the application service with their requirements. As points for future work, Kirschnick et al. (2010) list “self-management capabilities such as fault-tolerant cloud services, service level agreement management, and quality of service assurance”. Glohr et al. (2014) provide a draft of an architecture that offers application service production from sales to operation. For deployment, they use virtualization and a script-based installation of SAP software. Further details are not shared. Glohr et al. (2014) consider “[...] adjusting products, organizations, processes, and ERP systems” as the primary challenge. Mastelic et al. (2016) propose the use of a model-driven approach to deploy application services. Their meta-model consists of three layers. The top layer includes the service’s requirements and uses case definition, the platform-independent definition of the service structure, and the platform dependent specification including the service’s implementation details. Prede-

[11]http://www.opentosca.org
[12]https://docs.openstack.org/heat/

fined templates of are required for model transformations. Their proposed architecture includes components for central management, which includes the orchestration engine, for monitoring, a database, and for managing the services. They present a, mostly custom developed, prototype. Their prototype has interfaces to support Docker containers and full infrastructure virtualization with OpenNebula[13].

Heat (OpenStack Foundation, 2012) has gained some dissemination as the popular OpenStack's default orchestration software. OpenStack is described as a de-facto standard IaaS (Nelson et al., 2014). Heat templates try to be, but are not, fully compatible with the proprietary templates of Amazon's orchestration service. They focus on infrastructure orchestration. All elements of a service that can be managed in OpenStack can be defined in Heat templates and consequently be orchestrated with Heat. These elements include virtual machines, their images, volumes for storage, network elements such as routers or containers. In order to automatically scale application services, policies can be defined. While Heat focuses on the orchestration of infrastructure, it can trigger the configuration of software with configuration management software products.

TOSCA focuses less on infrastructure and more on the application-side of the service (Wettinger et al., 2016). Its service templates include topology templates that define the service landscape. The basic elements for defining landscapes in TOSCA are nodes and relationships. A node type can represent a software product or a virtual machine in a specific configuration. A relationship type could be *hosted on* connecting software and infrastructure. However, relationships between software can also be modeled. In any case, all types have to be separately defined. Management plans in TOSCA are used to describe how and in what order a deployment or change of the landscape should be performed. So-called deployment artifacts are configuration management specifications that perform the actual configuration tasks on each host of the service landscape. TOSCA is not associated with any infrastructure software. Some research-focused implementations (Binz et al., 2014), one proprietary service[14], and prototypical mappings to Heat (OpenStack Foundation, 2015) exist. Lenhard (2016) addresses standards such as TOSCA and constraints their versatility by stating that, even though the standards are comprehensively defined, implementations are often not complete.

While Heat and TOSCA can manage containers, Kubernetes[15] was designed explicitly for container orchestration and has evolved as somewhat of an industry standard. It was created by Google and is supported by other large players in the industry (Bernstein, 2014b). Kubernetes manages Docker containers and helps to assign raw resources such as memory and CPU cores to them. In Docker, inter-container communication is done via network ports, not individual addresses. Consequently, managing communication for a large number of containers is complex. Kubernetes introduces an additional abstract layer: pods. One pod group several containers. Each pod has its own network address, thereby organizing network communication by additional hierarchization (Bernstein, 2014b).

[13]https://opennebula.org/
[14]https://cloudify.co/
[15]https://kubernetes.io/

This was the last section describing the research background for the development of the ISAA. Based on the presented work, the next section will elaborate on the research gap.

3.5 Summary and research gap

The positioning in the literature and research gap will be highlighted in the following manner. First, the presented literature is summarized. Second, the positioning in the literature and research gap is elaborated on. The summary focuses on the general research background. Three sets of directly related work are defined, which are not part of the summary but are summarized and discussed to elaborate on the research gap in section 3.5.2.

3.5.1 Summary

Customers source IT services to reduce costs, concentrate on core capabilities, access the provider's technical excellence or specific domain knowledge. They are concerned about the security and privacy of their data and require service level agreements to guarantee business continuity. Customers require documented mechanisms in case of provider fall out. Agility is a reason for turning to cloud computing, although customers have to accept standardized services. Established companies that offer cloud services with good customer support can be solidly competitive (cf. section 3.1.2). (Riemer and Ahlemann, 2001; Currie and Seltsikas, 2001; Lacity et al., 2010; Kappelman et al., 2014; Schneider and Sunyaev, 2016; Labes et al., 2017)

IT service providers saw sourcing, automation, process standardization, and modularization as areas of improvement in 2011. Cloud computing adoption is increasing, but not every company (less than 50 %) uses cloud computing yet. ITIL-based IT service management is actively practiced in the transition and operation processes. ITIL processes of other stages (strategy, design, and continuous improvement) are following slowly (cf. sections 3.1.3 and 3.1.4). (Becker et al., 2011; Hsu et al., 2014; Marrone et al., 2014; Ng et al., 2017)

Enterprise management systems offer various benefits, but for successful implementation, top management support, training, often substantial outside consulting, and good communication are necessary. Reference models align organizational, process, data, and functional views of enterprise management system architecture. It is an unresolved research problem how to explicate the general validity of a reference model. Reference models are, however, applied with utility in practice. Computer-integrated manufacturing can achieve efficient production of just-in-time and customized products (cf. section 3.2.2). (Ngai et al., 2008; Gronau, 2010; Momoh et al., 2010; Scheer, 1997; Becker and Schütte, 2004; Fettke and vom Brocke, 2016; Harrison and van Hoek, 2008)

Insufficient empirical evidence is available on how application system landscapes of IT service providers are structured or what systems they are composed of. For instance, Botta-Genoulaz and Millet study several service companies, but among the companies related to IT service providers, are only a software company, a bank, and a financial service provider. On a high architectural level, Wittgreffe et al. present the system landscape at British Telecom (cf. section 3.2.3). (Botta-Genoulaz and Millet, 2006; Wittgreffe et al., 2006)

Several meta-models related to service science and IT services in particular have been created. They include a very general but comprehensive service science meta-model, process-meta-models for ITSM, and a more specific IT service production meta-model by Ebert et al. (cf. section 3.2.4). (Ebert et al., 2007; Goeken and Alter, 2009; Brenner et al., 2009; Valiente et al., 2012; Alter, 2011; Hintsch and Turowski, 2013)

To define and delineate the philosophical trend of fostering closer cooperation between developers and administrators (DevOps), the principles culture, automation, measurement, and sharing (CAMS) are postulated. DevOps' advocates not only cross-departmental collaboration between development and operations but also the close integration of the business into the DevOps process as well as product-ownership as a shift in culture. Automation is the second principle. A continuous deployment pipeline, centered around repeatable application system landscapes, facilitated through configuration management software, and automated testing, is central to automation. *If you can't measure it, you can't improve it*: key performance indicators include throughput time but also indicators in the categories of business success, customer experience, application performance, speed, and quality. Sharing, the last principle, is also manifested in open source software. Such licensed software is the basis for a lot of DevOps software. Google has sharpened the DevOps concept according to its necessities. In particular, site reliability engineers are recommended to have educational backgrounds in software engineering as well as sound low-level administrator knowledge. The goal for operations is a very high degree of automation. High automation and fruitful collaboration between developers and administrators are incentivized by simple, and apparently effective, key performance indicators (KPI). System architectures evolved from monolithic architectures to more componentized architectures. Microservices are a recently advocated architecture style that builds upon previous architectures (cf. section 29). (Lewis and Fowler, 2014; Gacek and Arief, 2004; Chen et al., 2009; Willis, 2010; Andersen-Gott et al., 2012; Alt et al., 2017; Ebert et al., 2016; Beyer et al., 2016)

Automation by administrators is not new. With the move away from mainframes to personal computers and large numbers of standardized (commodity) server hardware, the need to consistently manage and automatically configure the software stack on a large number of computers has grown. Domains of application include grid computing, networking management, or education. Research papers started to be published in 1994. However, particularly with trends of DevOps, deployment pipelines and microservice architectures, configuration management software has received wide publicity. Configuration management software takes device-independent configuration specifications from a repository and makes agents configure their specific devices. Research on configuration management software mainly includes identifying ways of how to best specify configuration and on non-functional properties of configuration management (cf. section 3.3.2). (Delaet et al., 2010; Herden, 2013; Wettinger et al., 2013; Hintsch et al., 2016a; Wettinger et al., 2016)

Three sets of related work are directly related to the contribution of the thesis. They were discussed in sections 3.1.3, 3.2.5, and 3.4.2. The research gap, which the ISAA addresses, is defined in the next section against those sets of related work.

3.5.2 Research gap

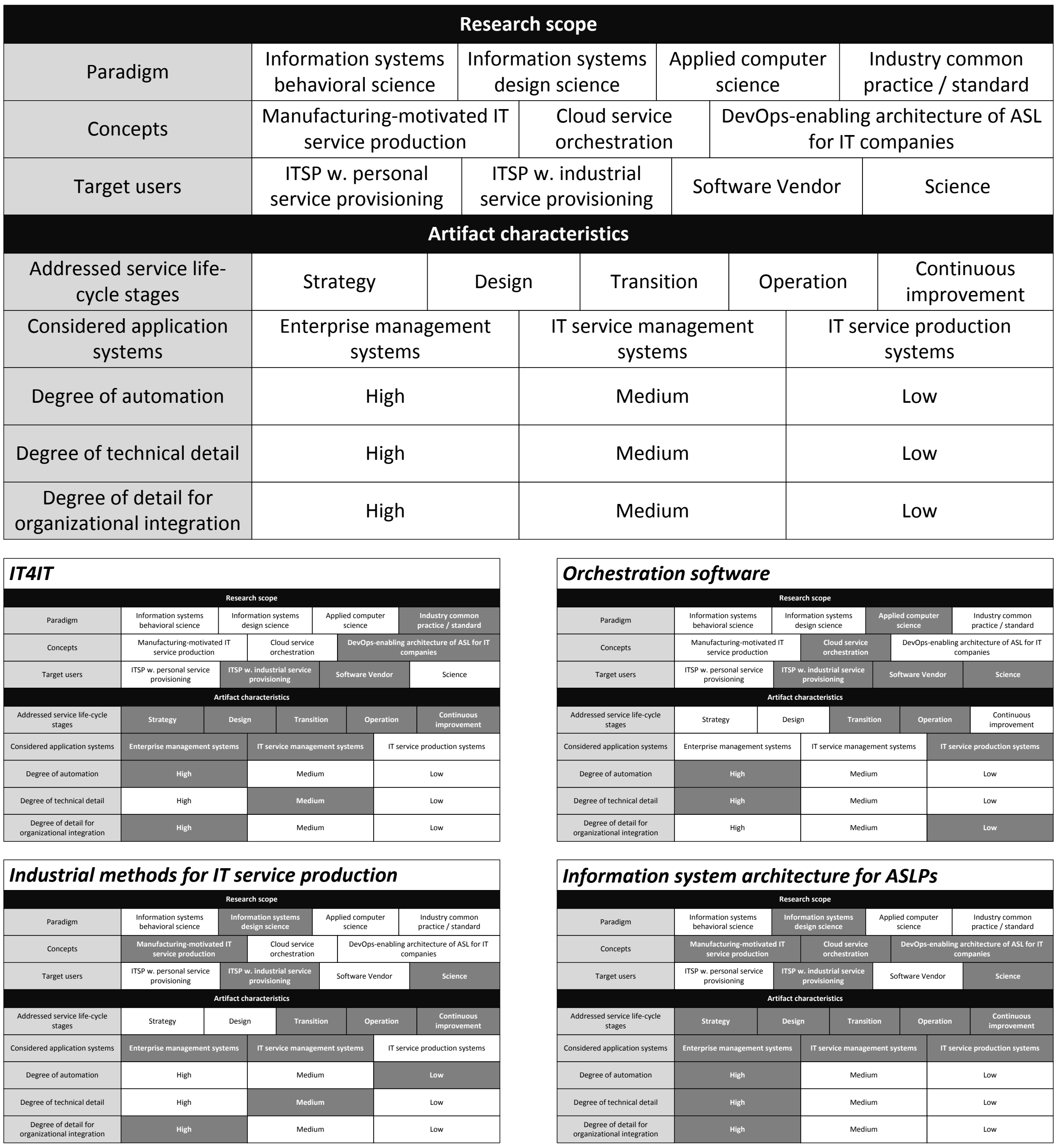
The research goal of this thesis is to increase the efficiency of ASLPs' application service production through standardization, automation, and modularization by creating the ISAA. Two hypotheses address the attempt to adopt practices from manufacturing in regards to these industrialization principles. Computer-integrated manufacturing is one of these practices. The first hypothesis is that dominant parts-based product design can be established for ASLPs. These designs are based on software for different operations automation approaches, such as configuration management software, container-based virtualization software, IaaS software, and orchestration software. Because it is not clear which OAA software is most suitable for the ISAA, RQ3 is formulated: What operations automation approach is suitable for constructing the information system architecture? RQ3 is answered in section 4.3.

The second hypothesis is that ASLPs can produce their application services model and component based. The production process should be automated. RQ1 is posed to understand what systems support the IT service production process: What do application system landscapes of IT service providers look like today? This question has not been sufficiently addressed by previous literature. RQ2 queries for the requirements of the ISAA. Three ASLPs are studied in more detail, and the requirements are derived in section 4.2 to answer RQ2. The literature on success factors of IT outsourcing contributes to these requirements.

Three sets of work are directly related to this thesis (related work). Here, they are called *IT4IT*, *industrial methods for IT service production*, and *orchestration software*. These sets are compared to the thesis in Figure 3.4.

IT4IT is primarily aimed at vendors of IT service management software, the DevOps-centric IT4IT provides a comprehensive, but high-level view of an ERP for IT. Primary and secondary business functions of IT companies are differentiated and aligned with a functional component architecture of an envisioned ERP for IT. IT4IT is a standard by the Open Group (cf. section 3.1.3). (The Open Group, 2017)

Industrial methods for IT service production include work on PPC and variant management methods. These have been suggested to address IT service provisioning, but the capabilities of today's software and service offerings for IaaS are not well reflected. Several works align with Zarnekow's framework and share a similar data model for an ERP implementation to support industrialized IT service production. In the ERP data model, hosts are individually represented as equipment and the concept places deployment



Research scope					
Paradigm	Information systems behavioral science	Information systems design science	Applied computer science	Industry common practice / standard	
Concepts	Manufacturing-motivated IT service production	Cloud service orchestration	DevOps-enabling architecture of ASL for IT companies		
Target users	ITSP w. personal service provisioning	ITSP w. industrial service provisioning	Software Vendor	Science	
Artifact characteristics					
Addressed service life-cycle stages	Strategy	Design	Transition	Operation	Continuous improvement
Considered application systems	Enterprise management systems	IT service management systems	IT service production systems		
Degree of automation	High	Medium	Low		
Degree of technical detail	High	Medium	Low		
Degree of detail for organizational integration	High	Medium	Low		

IT4IT

Research scope					
Paradigm	Information systems behavioral science	Information systems design science	Applied computer science	**Industry common practice / standard**	
Concepts	Manufacturing-motivated IT service production	Cloud service orchestration	**DevOps-enabling architecture of ASL for IT companies**		
Target users	ITSP w. personal service provisioning	**ITSP w. industrial service provisioning**	**Software Vendor**	Science	
Artifact characteristics					
Addressed service life-cycle stages	**Strategy**	**Design**	**Transition**	**Operation**	**Continuous improvement**
Considered application systems	**Enterprise management systems**	**IT service management systems**	IT service production systems		
Degree of automation	**High**	Medium	Low		
Degree of technical detail	High	**Medium**	Low		
Degree of detail for organizational integration	**High**	Medium	Low		

Orchestration software

Research scope					
Paradigm	Information systems behavioral science	Information systems design science	**Applied computer science**	Industry common practice / standard	
Concepts	Manufacturing-motivated IT service production	**Cloud service orchestration**	DevOps-enabling architecture of ASL for IT companies		
Target users	ITSP w. personal service provisioning	**ITSP w. industrial service provisioning**	**Software Vendor**	**Science**	
Artifact characteristics					
Addressed service life-cycle stages	Strategy	Design	**Transition**	**Operation**	Continuous improvement
Considered application systems	Enterprise management systems	IT service management systems	**IT service production systems**		
Degree of automation	**High**	Medium	Low		
Degree of technical detail	**High**	Medium	Low		
Degree of detail for organizational integration	High	Medium	**Low**		

Industrial methods for IT service production

Research scope					
Paradigm	Information systems behavioral science	**Information systems design science**	Applied computer science	Industry common practice / standard	
Concepts	**Manufacturing-motivated IT service production**	Cloud service orchestration	DevOps-enabling architecture of ASL for IT companies		
Target users	ITSP w. personal service provisioning	**ITSP w. industrial service provisioning**	Software Vendor	**Science**	
Artifact characteristics					
Addressed service life-cycle stages	Strategy	Design	**Transition**	**Operation**	**Continuous improvement**
Considered application systems	**Enterprise management systems**	**IT service management systems**	IT service production systems		
Degree of automation	High	Medium	**Low**		
Degree of technical detail	High	**Medium**	Low		
Degree of detail for organizational integration	**High**	Medium	Low		

Information system architecture for ASLPs

Research scope					
Paradigm	Information systems behavioral science	**Information systems design science**	Applied computer science	Industry common practice / standard	
Concepts	**Manufacturing-motivated IT service production**	**Cloud service orchestration**	**DevOps-enabling architecture of ASL for IT companies**		
Target users	ITSP w. personal service provisioning	**ITSP w. industrial service provisioning**	Software Vendor	**Science**	
Artifact characteristics					
Addressed service life-cycle stages	**Strategy**	**Design**	**Transition**	**Operation**	**Continuous improvement**
Considered application systems	**Enterprise management systems**	**IT service management systems**	**IT service production systems**		
Degree of automation	**High**	Medium	Low		
Degree of technical detail	**High**	Medium	Low		
Degree of detail for organizational integration	**High**	Medium	Low		

Figure 3.4: Morphological box that highlights the research gap addressed in this thesis.

and operation instructions of services in routings. Manufacturing execution functionality and improvement of automation are suggested for future work (cf. section 3.2.5). (Lebrecht, 1991; Zarnekow, 2007; Pinnow, 2009; Ebert, 2009; Vogedes, 2011; Dudek et al., 2012; Pilgram and Vogedes, 2012; Hintsch, 2013)

Orchestration software triggers the instantiation of elements such as virtual machines or containers, virtual storage, and network. The software determines the order of instantiation and configures or invokes configuration of the elements. It can use the output of instantiating and configuring one element as input for instantiating and configuring other elements. Research prototypes have been documented since 2010, but proprietary and standards-based open source products are available as well. Kubernetes is the de-facto standard for container orchestration and introduces an additional abstraction layer for management (cf. section 3.4.2). (Kirschnick et al., 2010; Bernstein, 2014b; Binz et al., 2014; Glohr et al., 2014; OASIS, 2015; Mastelic et al., 2016; OpenStack Foundation, 2012)

Industrial methods for IT service production and *ISAA* share the same research paradigm. *Orchestration software* is researched primarily in applied computer science, while *IT4IT* is an industry-driven standard. Both, *ISAA* and the *industrial methods for IT service production* share manufacturing-motivated IT service production. However, with the *ISAA*'s focus on computer-integrated service production and consideration of cloud service orchestration approaches it sets a different focus. This different focus has consequences for the data model of the *ISAA*'s architecture as well as for its process architecture.

The *ISAA* can be used to develop software in an agile and DevOps aligned style. Although, the focus lies more on the construction of application system landscapes from existing application software than on the development of new software. Application software development will be considered. However, the focus instead lies on the operations side: how to efficiently create the configuration and orchestration specifications necessary to operate complex landscapes.

Application software can be developed in-house or sourced in the case of the *ISAA*. The *ISAA* and the three sets of related work also slightly differ with respect to their target users. All address IT service providers with industrial service provisioning (internal and external). Specifically, TOSCA and *IT4IT* address software vendors that can use these standards to develop software products. In *IT4IT*, architecture levels four and five are specifically reserved for this purpose. Although *IT4IT* also should be considered by the scientific community, the works from the other two sets and the *ISAA* specifically address a scientific audience.

The characteristics of the sets of related work, compared to the *ISAA* will be described in more detail in the following. The ITIL-defined service lifecycle stages transition and operation are considered by all sets of related work and the *ISAA*, in the sense that deployment and operation of the service are addressed. *IT4IT* is aligned with ITIL and consequently covers the full lifecycle. The *ISAA* includes financial management and the provisions to design new services, which is why the *ISAA* is marked to address these stages as well. All address continuous improvement, except *orchestration software*.

The distinction of an IT service providers' ASL into enterprise management, IT service management, and IT service production systems is a result of the research conducted to answer RQ1 (Hintsch et al., 2016b; section 4.1). While the *orchestration software* addresses IT service production systems, only the *ISAA* integrates these systems with the other two types. Also, *IT4IT* takes a different approach than the *ISAA*. It focuses on abstract functional components, whereas the *ISAA* is based on actual application software products that are used by current companies. *Industrial methods for IT service production* focuses on management methods for supporting IT service production, but does not focus on automation, although automation is seen as an area of improvement in the debate on the industrialization of IT. The degree of technical detail is high for the *ISAA* as compared to *IT4IT* and *industrial methods for IT service production. ISAA* contributes to the idea of using industrial methods for IT service production, but incorporates a higher degree of automation, which changes the overall concept.

Regarding Hypothesis 2, the *ISAA* tests the limits of how ASLPs can produce their application services model and component-based as well as automatically. Therefore, a high level of detail is necessary. *IT4IT* only covers overview levels and has not yet defined more detailed scenarios. Compared to *orchestration software*, the *ISAA* attempts a high degree of organizational integration (e.g., regarding financial management). While in DevOps technical measurements are frequently discussed, business success is advocated by one source (New Relic, Inc., 2015). The *ISAA*, using its integrated approach, attempts to provide the possibility for measuring business success on a fine-grained level that is relevant to operations.

The next section will address the research questions.

4 Preliminary investigations

The preliminary investigations presented here substantiate the construction of the ISAA. In section 4.1, the status-quo of IT service providers' application of system landscapes is described (RQ1). A previously published conference paper (Hintsch et al., 2016b) is the primary basis for the section. Section 4.2 derives requirements that are used for the construction of the ISAA (RQ2) from data of three selected cases. Lastly, in section 4.3, different sets of suitable operations automation technology are compared. Subsequently, one is selected based on its suitability for constructing the ISAA (RQ3).

4.1 Application system landscapes of IT service providers

When constructing an information system architecture that is supported by an application system landscape, it is beneficial to know how ASLs are composed in practice. Constructing the application system landscape close to the prevalent design of practice should facilitate its implementation in companies. Also, the question should be asked if IT service providers, like other companies, already use enterprise management software, in this case, ERP software within their ASLs. ASLPs are often not ASLPs exclusively but have other business parts such as consulting (cf. section 4.2.5). Therefore, studying IT service providers in general, here, is reasonable. In Table B.1 it is stated that the majority of the interviewed companies use ERP systems.

Only one of the interviewed companies, a virtual machine hosting company *C1*, stated that it does not use ERP software. *C1* uses an application system landscape comprised mostly of custom-made software. The ASL is centered around *C1*'s virtualized hosting services. The interviewee of that company stated (translated and copy edited):

> "[...] If you consider the employee size we have set in relation to the number of customers we serve, we, of course, need a good information system. We, of course, use it, every process that we have is automated, everywhere possible. In short, we have no copy and paste. And, we work together with suppliers who provide solvency information, to name an example. This is all fluently integrated, and the software is in-house developed to the largest part. [...]"

C1 has a narrow product portfolio with a large customer base. Most processes, therefore, are automated. Because of high competition in the hosting sector, the company needs to differentiate itself. For example, customers are provided access to the virtual machines' firmware. Compared to competitors, such low-level machine access is unusual.

This product-based differentiation is achieved with in-house developed systems. *C1*'s ASL accommodates the company's specific need for automation and differentiation.

Companies that use standard ERP software use it for material management, sales, finance, and accounting, controlling, as well as for human resources. Some desire additional integrated functionality such as ticket management. Some smaller vendors offer such specialized ERP software. A decisive factor for choosing standard ERP software is their perceived quality in finance and controlling (Gronau, 2010, p. 12).

The interviewee of *C11*, a chief financial officer, was selecting a new standard ERP software for his company at the time of the interview. Amongst the five ERP software products the company had studied two systems stuck out: one with integrated ticketing functionality from a smaller vendor and one without, but coming from a tier-1[1] ERP software vendor. The interviewee of *C11* gave the lack of quality in finance and accounting of the smaller ERP software as the reason for purchasing from the tier-1 vendor (translated and copy edited):

> "[...regarding the tier-1 ERP software,] you don't have to discuss accounting algorithms and obviousnesses, which you sometimes need to discuss with smaller vendors [...]"

Consequently, they planned to integrate the external ticketing software with the newly purchased ERP software. A majority of the studied cases used additional standard IT service management software for ticketing. Other standard software supported business functions of customer relationship management and project management. The studied companies often composed their ASLs in best-of-breed approaches by choosing software from different vendors that were most suitable to support specific business functions (Wittgreffe et al., 2006).

To find out what the other constituents of the application system landscape of IT service providers are, the interview transcripts from the case study (cf. section 2.1 and chapter B) were analyzed. The interviewees reported on business functions that are supported by application systems (cf. section B.2). The business functions and application systems were coded in the interview transcripts. Coded occurrences were then grouped. The coding and grouping led to three major categories of application systems (Hintsch et al., 2016b). These categories are depicted in Figure 4.1.

Application systems of IT service providers include enterprise management, IT service management, and IT service production. Enterprise management software integrates all business functions of an enterprise as well as its resources into a common database (cf. section 3.2.1). IT service management software supports "[t]he implementation and management of quality IT services that meet the needs of the business. IT service management is performed by IT service providers through an appropriate mix of people, processes and information technology" (Cannon, 2011a, p. 16). IT service production (ITSP) systems provide the actual service. ITSP systems include software for operations automation approaches. Clients' application software (e.g., analytics software for a client of an insurance

[1]ERP software vendors are grouped into different tiers based on their size (Simon et al., 2010).

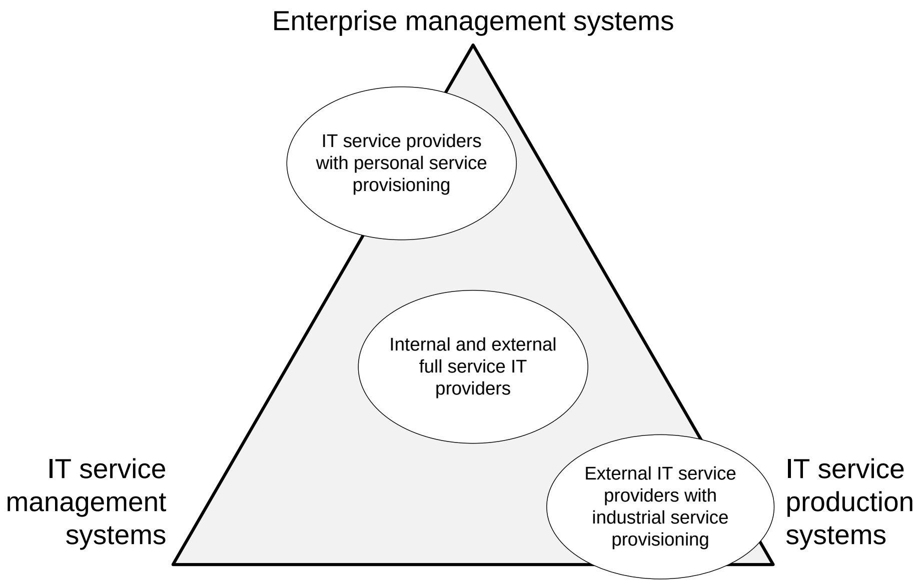

Enterprise management systems
IT service providers with personal service provisioning
Internal and external full service IT providers
External IT service providers with industrial service provisioning
IT service management systems
IT service production systems

Figure 4.1: Comparison of IT service providers' application system landscapes (based on which application systems are mission-critical).

company) can also be understood as IT service production systems. Therefore, the ITSP system can also include enterprise management or IT service management software if a customer requests a specific service.

The application system landscapes of the studied companies contain the three types of applications. ASLPs will need systems to support their enterprise management, provide operational support (e.g., for ticketing), and they will need to produce the IT services equipment-based. Therefore, architecture decision 1 is made:

Architecture decision 1
The ISAA's application system landscape shall comprise three major system categories: enterprise management, IT service management, and IT service production.

The architecture decisions guide the construction of the ISAA (cf. chapter 2). Only central architecture decisions are documented as it is not feasible to document every architecture decision (ISO/IEC/IEEE, 2011, p. 15). They will be further substantiated with requirements that are presented in the next section.

The following study can give some indication to find out which system should be leading in the ASL of the ISAA. Hintsch et al. (2016b) located IT service providers within the triangle. It was studied if the IT service provider type has an influence on the ASL composition (Hintsch et al., 2016b). Studied IT service providers were placed in the triangle based on the application systems within the respective provider's ASL that were most mission-critical. Figure 4.1 depicts three clusters that emerged from this placement.

Providers with external and industrial IT service provisioning are located in the triangle's lower right corner. Their IT service production systems are crucial to their business. Consequently, they are used as a starting point to build the providers' application system landscapes. In the segment of very standardized services, it is hard to follow a cost

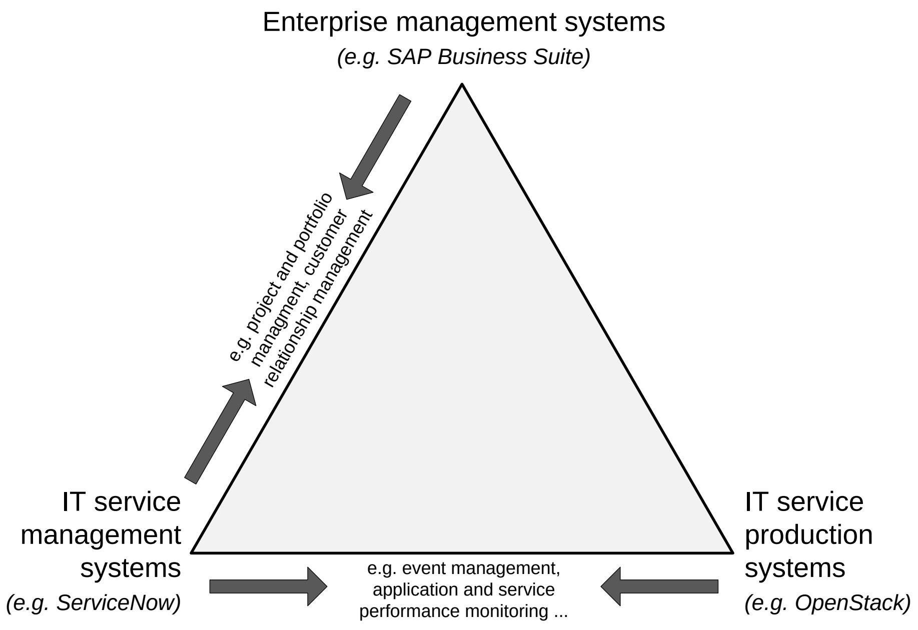

Enterprise management systems
(e.g. SAP Business Suite)
e.g. project and portfolio managment, customer relationship management
IT service management systems
(e.g. ServiceNow)
IT service production systems
(e.g. OpenStack)
e.g. event management, application and service performance monitoring ...

Figure 4.2: Functionality convergence of application software packages.

leadership strategy (Porter, 1980, p. 35). Therefore, these companies try to differentiate themselves by offering competitive functionality. This functionality requires their systems to be highly individualized.

Providers who conduct personal service provisioning are placed between IT service management and enterprise management systems. They are distinctly located closer to enterprise management systems because business functions like project management and human resource management are crucial to their business.

IT service providers who conduct both, internal and external full-service provisioning, are positioned towards the middle of the triangle. Their industrial services contain a high degree of standardization. Therefore, ITSP systems are required to offer these services. Also, IT service management systems are needed to support request and incident management during the service operation phase. Lastly, enterprise management systems are used to financially monitor as well as support the provisioning of additional personal IT services.

ASLPs are primarily industrial service providers. However, as will be shown in the following section, the ISAA has to support different business models containing ASLPs. Often, consulting services are offered in addition to landscape provisioning services. Therefore, ASLPs are not positioned towards the lower right, but rather to the middle of the triangle.

It is necessary to know which should be the leading system in the application system landscape to construct the ISAA. The leading system is the one that triggers actions within other systems and holds the master data (Otto and Schmidt, 2010). One option would be to enrich IT service production systems with enterprise management functionality as is done by cases *C1* and *C24* and make the ITSP the leading system.

In addition to locating IT service providers in the triangle, it was studied how software products of each application system category converge. The convergence was determined based on the business functions that are supported by each application system category. This investigation was based on sets of business functions defined by Lloyd (2011, p. 151), Betz (2011, p. 131), and Schröder and Pilgram (2010). The work by Betz (2011, p. 131) is groundwork to the IT4IT component architecture. Figure 4.2 displays the result. Primarily, a convergence between IT service management and enterprise management software can be observed. This convergence includes, for example, business functions such as project and portfolio management as well as customer relationship management. Such convergence can also be observed between IT service management and IT service production systems, for example, regarding event management as well as application and service performance monitoring.

Existing application system landscape of IT service providers, in most cases, are centered around enterprise and IT service management systems. This is not surprising as these systems, often based on standard software, offer most of the functionality required to manage the IT service provider's business. Therefore, the question has to be answered whether the leading system should be the IT service management or enterprise management system. Considering the business model of the ASLP that includes not only industrial provisioning, but also some limited personal provisioning, the following architecture decision is made.

Architecture decision 2
An enterprise management system shall be the leading application system within the ISAA's application system landscape.

Having analyzed the status quo, contractual, billing as well as service data is already maintained in companies' ERP systems (Hintsch et al., 2015b). Therefore, in the construction of the architecture, it will be investigated how a standard ERP software can support the automated production of ASLs. This support should go beyond aggregating information from other systems. The ERP software should be the primary source of information, also regarding the full structure of the services. Usage of an ERP software, proficient in controlling, should be valuable for adhering to the measurement principle of DevOps (Alt et al., 2017; New Relic, Inc., 2015), in particular, business success.

Adhering to architecture decision 1, the two other application system categories also have to be considered. Therefore, the following architecture decision (AD) is made.

Architecture decision 3
Ticketing and consumption data shall be fed back to the enterprise management system by the IT service management and IT service production systems. Relevant data such as contractual data or structural application service information can be looked up in or integrated from the ERP system.

Ticketing and consumption data originates in the ITSM and ITSP systems. It has to be transferred to ERP system to enable full cost calculation of the services for billing and controlling.

In the next section, requirements of the architecture for making further ADs will be discussed.

4.2 Requirements of the architecture

Three cases from the case study were selected to illustrate the relevance of addressing ASLPs and derive requirements for constructing the architecture. They are presented in Table 4.1. The three cases were selected because they span all relevant company sizes. These are small, medium, and large in regards to their employee size European Commission (2005, p. 14). As discussed in the research design chapter (cf. section 2.1), micro companies were not included. The selected cases are further ASLPs. In addition to the interviews, further data was used to analyze the three companies.

The requirements are grouped into four categories. Relevance (R) is derived from practicability because the ISAA addresses real cases and can achieve utility for these (Hevner et al., 2004). The service needs to be completely (C) described to leverage the integration potential of the ERP system. Requirements specific to the product of ASLPs, application services (S), are formulated. ERP systems are used to manage the processes of a company. Therefore, the production process of ASLPs starting with customer requirements, over service operation and billing to service retirement, is addressed. Consequently, the ISAA has to facilitate production manageability (P).

The four requirement categories and each requirement will be described in sections 4.2.1 - 4.2.4. Section 4.2.5 provides an overview of the requirements.

4.2.1 Architecture relevance

Cases Alpha, Beta, and Gamma illustrate that ASL providing is practiced in different kinds of organizations. They differ in terms of size and business model. If ASLP is only a part of the business model, the architecture must be able to integrate with the other parts. For instance, parameter-based customization, full-fledged software development, and consulting are also practiced by the case companies. The DevOps philosophy has mainly been applied in companies whose primary business is software development (Alt et al., 2017), but resulting microservice architectures can also grow to complex application system landscapes (Lewis and Fowler, 2014). Therefore, the first requirement (REQ) mandates the information system architecture to address different business models containing ASLP (REQ R.1).

Though this thesis focuses on producing ASLs, the ISAA has to facilitate the integration of processes that are out of scope (REQ R.2). For instance, Alpha has fully implemented procurement in their ERP system to be policy compliant. Furthermore, their ERP systems are used for financial planning. In general, some company processes

Business model	Application system landscape
***Alpha** is a large IT service provider.*	
Data comes from an interview, workshops, the company website, and other websites carrying relevant information (e.g., for investors).	
As a global corporation's subsidiary, Alpha's ASLP business segment comprises enterprise application services based on sourced standard software. The services are provided from Alpha's data centers. Customization and consulting services are offered in addition to the application services. Recently, data-center-backed IaaS capabilities were added to provider's portfolio.	For enterprise management, a historically grown application subsystem landscape of four standard ERP systems is used, particularly in financial, quotation, procurement, and order management, as well as human resource management. IT service management and production are based on virtualization and monitoring tools, an ITSM tool suite for ITIL's operation processes, enterprise application-specific management software, as well as in-house developed software automats for operations automation.
***Beta** is a medium enterprise application system landscape provider.*	
Data comes from an interview, the company website, and other websites carrying relevant information (e.g., for investors).	
The company markets in-house developed enterprise application software that customers can operate on-premises. Also, implementation and customization consulting are offered. Increasingly, customers use the managed service offerings under which the customer landscapes are operated in the two data centers of the provider.	The company employs its own ERP software, which includes an industry solution for IT service providers, for its enterprise and partially for its IT service management. While the ERP system is used for financial, quotation, procurement, order and human resource management, it offers market-competitive integration capabilities with the ticket system used by software development and service operations personnel. Operations automation is solved in the form of in-house developed scripts.
***Gamma** is a small enterprise application system landscape provider.*	
Data comes from interviews, workshops, the company website, and other websites carrying relevant information (e.g., for investors).	
The company deploys and operates highly standardized enterprise application system landscapes based on sourced standard application software for its customers' training purposes, while gradually extending its business model to IaaS and other industrial IT service offerings.	Spreadsheet software is used for management of finance, quotations, procurement, and orders. There is an ongoing project to implement an ERP system for replacing spreadsheet-based management. OAA software, as well as system maintenance software specific to the sourced standard application software that is provided to the customers, is used. Incident management is supported with an IT service management tool.

Table 4.1: Overview of illustrative case sample with descriptions about the business model.

do not need to be supported by application systems. For instance, legal activities are not within the scope of Gamma's ERP implementation project. Furthermore, Alpha has not automated its hardware procurement activities by using a just-in-time method, which would promptly compensate low spare capacity, since hardware sourcing requires human bargaining. When requiring process integration, critical success factors of ERP implementation need to be taken into account. In particular, scholars advise using a narrow scope to determine which modules should be implemented without ignoring the vendor-designated inter-module integration points (Momoh et al., 2010). Further, a danger lies in excessive customization of standard software (Momoh et al., 2010). ERP systems will not be the only systems required to implement the architecture. For instance, Alpha practices IT service management aligned with ITIL but with a focus on service operation processes, such as incident or change management. Gamma uses various additional software, and also an IT service management software.

Hypothesis 1 requires companies to implement software for operations automation, such as IaaS software. Thus, the architecture has to integrate with an organization's existing technology stack (REQ R.3). For instance, all case companies operate their own data center. They have to consider the entire IT stack starting from facility management and tasks, such as engineering the data center cabling plan and then actually completing the wiring. Such work needs to be performed by engineers and artisans and cannot be abstracted away without identifying a suitable replacement. Additionally, automation can be viewed skeptically due to work required for formalization (Talwar et al., 2005). Therefore, automation approaches which are evolutionary rather than revolutionary may yield a higher success rate.

The analogy of manufacturing (REQ R.4) adds to the architecture relevance from a scientific standpoint. It contributes to the existing stream of analogy-employing research regarding the industrialization of IT (Zarnekow, 2007; Becker et al., 2011). The ISAA shall leverage this analogy mainly by two means. First, proven application software for manufacturing shall be reused. Second, the reuse of the nomenclature could reduce differences between manufacturing and IT for professionals working in supporting activities such as financial controlling or sales administration. Further, it may bridge the gap between business and IT of the provider. By taking a parts-based view that is consistently managed in the provider's ERP and related systems, the service structure is made transparent to both the sales as well as the engineering staff of the provider.

4.2.2 Complete service description

An essential characteristic of services is that they are offered over a specified period. Service level agreements declare service level targets to be met by the provider during operation of the service (Cannon, 2011b, p. 453). Also, application services can be offered on a subscription or pay-per-use basis (Iveroth et al., 2013). The pricing modalities of the contract between the provider and the consumer need to be reflected in the service description as well. Hence, REQ C.1 is that service level agreements and other relevant contractual specifications need to be considered (Riemer and Ahlemann, 2001).

Resources for computing, networking, and storage as well as software, such as operating systems or databases, have to be orchestrated and configured to provide the required application services. All deployment steps of the customers' ASLs should be automated. Automation should increase deployment speed undoubtedly above that of manual deployment (Talwar et al., 2005). Therefore, the technical service description needs to be sufficient for automated deployment (REQ C.2). At the same time, not all tasks involved in production may be automated. Software engineering is an example. However, also in deployment, a tradeoff can be made between the expenses of formalization required for automation and the money saved by avoiding manual work. In some cases, such as complex or infrequent customizations, manual steps should be addable to the technical service description. Nonetheless, automation should have a high priority (Beyer et al., 2016; Alt et al., 2017).

4.2.3 Application service specificities

For illustrating the merits of operations automation approaches, often illustrative and relatively simple ASLs are selected (Wettinger et al., 2016). However, for the architecture to be relevant (cf. REQ R.3), instantiations of the information system need to support a realistic degree of complexity of the application services' ASLs (REQ S.1). The complexity of the three cases' ASLs that offer their application services, for instance, surpasses the examples of those used by Wettinger et al. (2016). Complexity is higher both in regards to landscape size (e.g., node count) and configuration of each node.

Although standardization helps automation and therefore can reduce costs, customizability is important to fulfill the customer's requirements, too (REQ S.2). On the one hand, an application service may be standardized without variability between customer instances. On the other hand, a service can be engineered and be very customer-specific. Using the mass customization approach (Ahmad et al., 2010; Teubner and Remfert, 2017), some customizability needs to be offered with parameterized standard service offerings as well. The range from standardized to individualized service is also apparent in the cases. While Gamma offers highly standardized services, Alpha offers complex, customized system landscapes to its customers to fulfill most of their requirements due to high market competition.

Another principle of the industrialization of IT is modularization. In order to increase reusability and to achieve customizability, modularization should be used in addition to parameterization (REQ S.3). Modularization is practiced, for instance, by hierarchization of the service catalog. When trying to automatically and semi-automatically assemble application services, modules also have to be composable into a complex ASL. Such a practice would be comparable to computer-integrated manufacturing (cf. REQ R.4 and section 3.2.2).

4.2.4 Production manageability

The production process should exhibit a high degree of automation (REQ P.1) to increase efficiency and quality. For the production process, automation of multiple technical abstraction layers as well as business aspects, such as contractual specifications, has to be integrated. In Alpha's production process, customer and related internal orders are being created based on entries from the service catalog. Such orders lead to instructions issued to the respective departments to set up assets, such as virtual machines or networking equipment. The steps from a customer inquiry to a functioning system are neither supported by a workflow system nor are they automatically performed. One of the most substantial obstacles is seen in organizational integration challenges. Similarly, Beta's hosted service deployment is manually triggered outside their ERP system. Beta's data center administrators use a collection of custom automation scripts to accomplish this task.

When automation would increase efficiency, KPIs should be gathered to measure this variable (REQ P.2). While Alpha and Beta can make KPI-based statements about operational efficiency, Gamma has little capability in this area due to a lower system and process integration.

Beta, but also both other cases, illustrate the necessity to support engineering activities within the production process (REQ P.3). Such engineering activities may take place within internal orders, for instance, when sales or marketing request service creation. For software engineering, traditional and agile methodologies exist (Tarhan and Yilmaz, 2014). However, an increased merging of software and IT system landscape engineering methodologies can be observed, as indicated by the new trend of DevOps (Spinellis, 2012; Beyer et al., 2016). Engineering support methods such as simulations, which are used in manufacturing and assist the early design phase of IT system landscapes (e.g., to optimize availability and costs (Bosse et al., 2016)), are not practiced by the cases. However, engineering tasks that accompany or follow development, such as unit and integration testing are practiced. For instance, Alpha uses automated tests, although not in a continuous delivery style as suggested by Humble and Farley (2010). Alpha's employees continuously have to work with newly introduced application software for the customer (e.g., in-memory databases). Therefore, they cannot rely on pre-existing standardized tests. Tests need to be developed by Alpha for automated testing. All three providers rely on test environments to prepare their services for production. Beta even offers a provider-hosted test environment as a value-added service in its implementation projects before enterprise application systems at customers' sites go live.

It is particularly evident for the global delivery models of Alpha that unambiguous communication is crucial. Modeling tools and their diagrams could facilitate unambiguous communication. Flawless communication is also vital in pure onshore delivery, but modeling tools are not used for system landscape engineering by Alpha. Again, manufacturing with tools such as computer-aided design software acts as a benchmark (Goettsch and Tosse, 2013; Herden, 2013). Also, to modeling tools assisting with unambiguous communication, they could serve as a basis for making system landscapes operational. Mainly in cases were individual ASLs are created based on customer inquiries, price estimates could be compiled from such models in early design phases. Early price estimates can be crucial because costs are a decisive factor in an outsourcing decision (Riemer and Ahlemann, 2001; Currie and Seltsikas, 2001; Lacity et al., 2010; Kappelman et al., 2014; Schneider and Sunyaev, 2016).

The operation phase is crucial in the IT service's lifecycle. Thus, its support is an important requirement (REQ P.4). Particularly noteworthy is the system integration of incident management, as practiced by Alpha and Beta. Such incidents can, for example, result in informing the user about correct usage of the service, in ASL reconfiguration, or in development activities.

Whereas in manufacturing the main billing activities are usually centered around the shipment of the product, in industrial IT service provisioning the act of deployment, as well as the continuous operation of the IT services, is usually invoiced by the provider. Together with financial management, billing is a crucial part of the production process and needs to be supported by the information system architecture (REQ P.5). The cases have different maturity levels for billing as well as financial management. Alpha, for instance, has difficulty in determining its service costs. One of the main reasons of Beta and Gamma for implementing the ERP project is to automate the billing process as well as to make it more consistent.

Next, the overview of the requirements will be provided.

4.2.5 Requirements overview

The requirements that were worked out and presented in this section are used as a rationale to guide the architecture decisions. The architecture decisions are used to construct the ISAA. Table 4.2 provides an overview of the requirements.

4.3 Selection of an operations automation approach

The goal of this section is to select one approach of operations automation. The selected approach will then be used in the next chapter as a basis for constructing the architecture. The decision process is guided by the requirements and by an analysis of different OAAs.

Wettinger et al. (2016) have identified multiple approaches for operations automation that can be performed on different layers of the IT stack. Figure 4.3 shows which approach focuses on which layer. The approaches and their focus on the different layers of the stack will be illustrated in detail in the following.

Infrastructure management allows to create, inspect, modify, and remove infrastructure resources. Such resources can include virtual machines, virtual storage volumes, or network components such as routers, to name some examples. Although infrastructure management, in the context of operations automation, often refers to the management of

ID	Short description
	Architecture relevance
R.1	Address different business models containing ASLP
R.2	Facilitate integration of processes not within the scope of the thesis
R.3	Integrate with organization's existing technology stack
R.4	Leverage potential of manufacturing analogy
	Complete service description
C.1	Consider SLAs and other relevant contractual specifications
C.2	Make technical service description sufficient for automated deployment
	Application service specificities
S.1	Support realistic degree of customer ASL complexity
S.2	Service customizability, as well as standardization, need to be achievable
S.3	Modularization shall be used to increase reusability
	Production manageability
P.1	Achieve a high degree of automation to increase efficiency and quality
P.2	Enable KPI-based management
P.3	Support engineering activities
P.4	Support service operation phase
P.5	Support billing and financial management

Table 4.2: Requirements for designing the ASLP information system architecture.

virtual infrastructure resources, the automated management of bare metal[2] can also be supported (OpenStack Foundation, 2011). Bare metal integration in IaaS systems enables the direct use of physical hardware without the overhead of virtualization. Bare metal integration relies on server functions to cold start the server over a broadcast from the network and to load the operating system from the network (Magherusan-Stanciu et al., 2011).

Automated infrastructure management usually can be performed by using an application programming interface or command-line interface (Bumgardner, 2016, p. 56). Consequently, this approach can be used by administrators manually or script-based to perform specific tasks of managing infrastructure resources. An API facilitates the integration of these functions into custom operations automation software.

In plan-based configuration management, bare metal or virtual machines that have fully loaded their operating system (OS) are configured. The configuration of physical or virtual machines is performed above the OS layer. Although configuration options of the OS can be specified, defining which OS is used is not in the scope of plan-based configuration management (Wettinger et al., 2016). Language and plan-based configuration are used synonymously (Talwar et al., 2005; Wettinger et al., 2016).

On the contrary, in hypervisor-based virtualization, virtual machine images contain the OS and all other layers of platform and software. Plan-based configuration manage-

[2] *Bare metal* refers to executing an operating system directly on the server without a virtualization layer. Direct execution can be beneficial in scenarios where performance is crucial (Bernstein, 2014b).

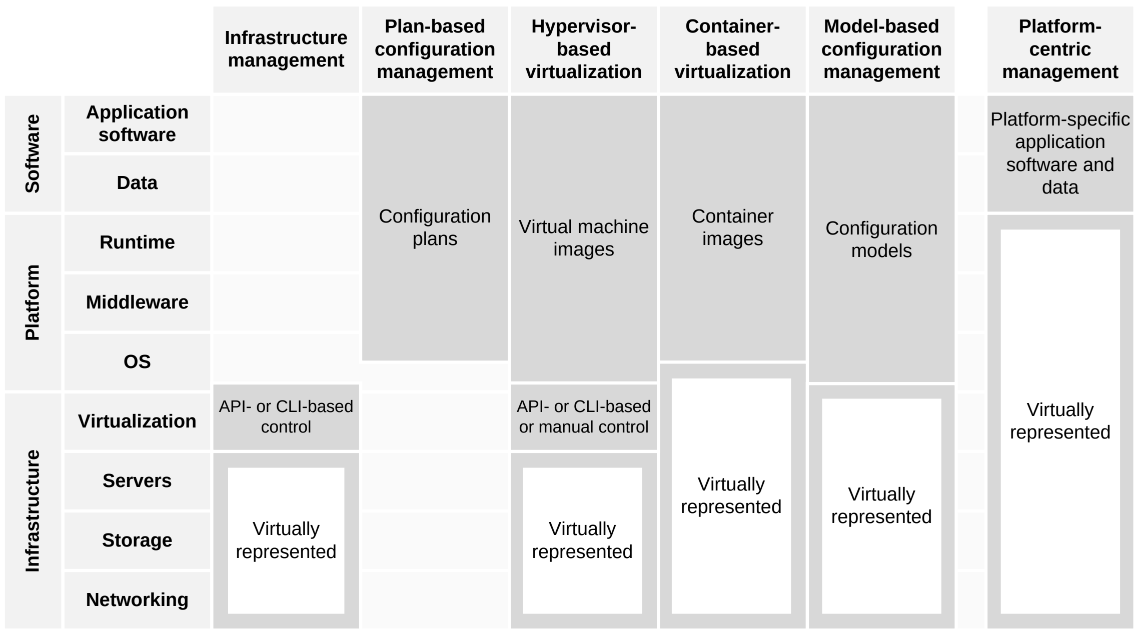

Infrastructure management
Plan-based configuration management
Hypervisor-based virtualization
Container-based virtualization
Model-based configuration management
Platform-centric management
Software
Platform
Infrastructure
Application software
Data
Runtime
Middleware
OS
Virtualization
Servers
Storage
Networking
Configuration plans
Virtual machine images
Container images
Configuration models
Platform-specific application software and data
API- or CLI-based control
API- or CLI-based or manual control
Virtually represented

Figure 4.3: Operations automation approaches focus on different layers of the IT stack.

ment and hypervisor-based virtualization can be combined with infrastructure management, respectively. In this regard, Zhu et al. (2015) compare two approaches of dealing with virtual machine images: heavily-baked images and lightly-baked images. Standard virtual machine images are used across different application scenarios using the lightly-baked approach. So for instance, the same base image is used for deploying a database server host as well as deploying an application server host. After their operating systems have fully loaded, plan-based configuration is used to configure the host as needed. The lightly-baked approach has the advantage of cheap configuration variability. However, it comes at the cost of an increased time required between booting a machine and being able to use the services deployed on it. Also, this approach can have stability issues when, for example, software package repositories need to be contacted to obtain software binaries. When these repositories are not available due to network issues, the deployment can fail (Zhu et al., 2015). Heavily-baked images contain all required configuration. Therefore, they have superior deployment robustness as well as deployment speed. However, when having a large variety of required configurations, large image quantities can become expensive to manage. Even though in the heavily-baked image approach many configuration tasks can be performed before booting a host, some configuration may still be necessary (Zhu et al., 2015). For instance, the management platform on which the heavily baked images are orchestrated may require such configuration. Therefore, also hypervisor-based virtualization will rely on some form of configuration management.

Container-based virtualization is similar to hypervisor-based virtualization in the sense that configuration is done before starting a container. However, a container does not contain its full operating system but shares its host's OS kernel with other containers of that host. Therefore, containers can only be executed on a host with a compatible OS kernel. A prominent representative of container-based virtualization is *Docker*[3]. Companies that want to use Docker have to check if the OS that they use is supported. It supports only very recent versions of Windows. Support for Linux is broader, but for most supported distributions, recent releases or kernel versions are required (Docker, Inc., 2017). Container-based virtualization, in production, does not only rely on containers (cf. section 3.3.2). To manage vast application system landscapes, with elasticity requirements, orchestration frameworks such as *Kubernetes* are used. These manage the scheduled allocation of containers to physical and virtualized resources (Bernstein, 2014b).

A key advantage of the container-based approach is its granularity and the fact that containers deploy faster than full virtual machines (Felter et al., 2014). However, the recent research by Manco et al. (2017) proposes that paravirtualized virtual machines could be modified in a way where they exhibit even faster boot times and higher security compared to container-based virtualization. Comprehensive management system support for such an approach is not available yet though. REQ R.3 requires that the ISAA integrates with an organization's existing technology stack. The versatility of containers may be limited in this regard. For example, companies like Alpha also operate legacy application software for their companies. Moving legacy application software to the cloud stack comes with some challenges (Gholami et al., 2017). Regarding container-based virtualization, a significant challenge is the need for adapting the software to be stateless. Legacy application software designed for on-premises often relies on the state to be stored such as contextual information (Gholami et al., 2017). Following the argumentation of Brandon (2016); The HFT Guy (2016), choosing containers as an underlying approach for operations automation may be problematic because containers are designed to be stateless and, consequently, extensive re-engineering of legacy application software would be required. State can be managed with containers (e.g., filesystem or relational database management systems). However, for fully leveraging containers in microservice and cloud-native architecture styles, where they are often suggested to bring most utility, substantial re-engineering efforts for legacy systems is very likely to be necessary (Lewis and Fowler, 2014; Leymann et al., 2016; Kratzke and Quint, 2017; Gholami et al., 2017).

Model-based configuration management, based on processor-supported virtualization, covers the whole IT stack without abstracting away the infrastructure and OS. Furthermore, operating a diverse OS mix is more feasible on process-supported virtualization than on container-based virtualization. Full and process-supported virtualization support those OS that supports the process architecture of the virtual machine (Hoffmann, 2010).

Wettinger et al. (2016) differentiate infrastructure-centric and application-centric modelling. Infrastructure-centric configuration models, based on defined meta-models, allow to define behavior which can also be implemented with infrastructure management, but in a standard format (OpenStack Foundation, 2012). Also, this approach allows defining which configuration plans or scripts should be executed on which host. Application-centric configuration models include the same functionality. Also, they add means to define relati-

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

onships between software components such as application software and middleware across hosts. These relationships have to be substantiated with implementation artifacts (Binz et al., 2014). Wettinger et al. (2014) state that idempotency and stable, converging configuration is often hard to achieve. However, Hanappi et al. (2016) proposes a method for testing configuration specifications to avoid instability and to achieve idempotency. Idempotency issues apply to both infrastructure- and application-centric styles because both approaches rely on application software configuration to be performed language-based. Scripts could also be used, but Talwar et al. (2005) describes ascending maintainability when using script-, language-, or model-based configuration approaches on large systems. For the scope of this comparison, infrastructure-centric and application-centric configuration management are not differentiated because the layers they address are the same.

Finally, Wettinger et al. (2016) add platform-centric management to their list of operations automation approaches. Platform-centric management is based on the PaaS service model. This model can be appealing when developing new application software because focus can be put only on developing the code. Maintenance and operation of the underlying IT systems is the duty of the PaaS provider. A popular specification is *cloud foundry* that provides a framework for PaaS-based custom application software (Bernstein, 2014a). The *cloud foundry* framework allows for a standardized deployment by the developers. The PaaS provider is responsible for the operation of the developed software. Several PaaS providers have adopted this specification. PaaS has the advantage of abstracting from the need of dealing with infrastructure-related questions. However, this comes at the price of tightly binding one's application software to the architecture of the respective PaaS offering (Wettinger et al., 2016). For companies required to control the whole IT stack, this may not be a suitable choice. This approach is also not selected because essential layers are not represented.

An operations management approach that is selected as the subject of analysis for the ISAA should focus on the whole IT stack and not abstract away specific layers. All approaches, but one, fall short of that requirement. Therefore, the following architecture decision is made:

Architecture decision 4
Operations automation, in the ISAA, shall be based upon model-based configuration management.

Although this approach is chosen, the other approaches have their advantages and could also be relevant for ASLPs. Hypervisor-based virtualization could be combined with model-based configuration management. It could be conceivable to migrate to a heavily-based image approach for frequently requested application services. Also, container-based virtualization could be required by providers that implement the architecture. Consequently, in the evaluation, it should be tested if the ISAA is also compatible with other operations automation approaches. Nonetheless, model-based configuration management has the advantage over virtualization approaches in that automation is explicitly documented. If a virtual machine or container image is not annotated with meta-data or if their creation scripts are not accessible it is not directly discernible what they contain or how they are configured. The configuration models explicitly specify the configuration and how the application system landscape's operation is automated. Therefore, documentation effort is decreased, and possible erroneous communication between operations professionals may be alleviated.

To run application software, some form of automation above the infrastructure layer is necessary. The infrastructure layer should also be automated (cf. REQ P.1). As argued in the introduction, the simple provisioning of infrastructure resources is likely to become a commodity in the coming years. Infrastructure can be sourced from a public provider, or it can be run in a private cloud on-premise. Consequently, the following architecture decision is made.

Architecture decision 5
The ISAA's application system landscape production shall be supported by on-premise or sourced infrastructure as a service.

In the following section, the ISAA will be described.

5 The information system architecture for ASLPs (ISAA)

ASLPs, like most other organizations, interact in value systems with other participants like suppliers and customers. Each organizational participant in the value system has its value chain to produce or consume value, or do both. Each organization's value chain can be decomposed into business functions. Aggregated business functions are decomposed into business functions on a lower abstraction level (Weske, 2012, p. 75). On the lowest level of this decomposition, business functions are called activities. Business processes consist of those activities, and they relate to business functions. Based on Weske (2012, pp. 73–83), this coherence is shown in Figure 5.1. The terminology of Weske (2012) is used when discussing business processes throughout this thesis. The taxonomy depicted in Figure 5.1 also is used to provide context information about the setting in which the ASLP is active (cf. Figure 5.2).

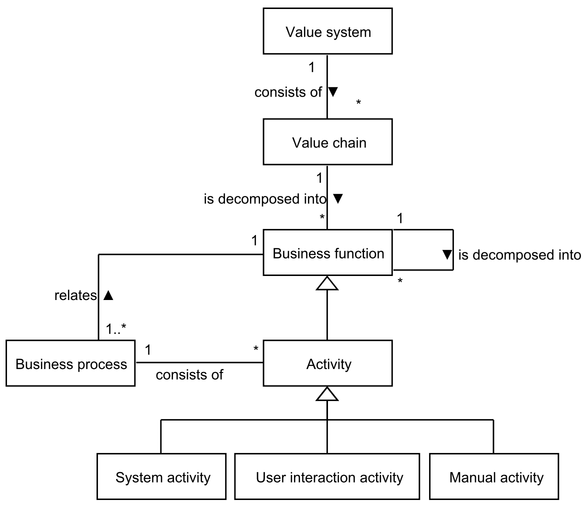


Figure 5.1: Value system and its sub-concepts, based on Weske (2012, pp. 73–83).

Figure 5.2 shows a value system of an ASLP as an example. The ASLP provides a content management system (CMS) as an application service for a customer. The customer uses this application system to support its business process for advertising new products. One business function in this process is the activity *Add product announcement* that directly uses the functionality of the application service. The activity is performed to win new customers and belongs to the business function of marketing and sales.

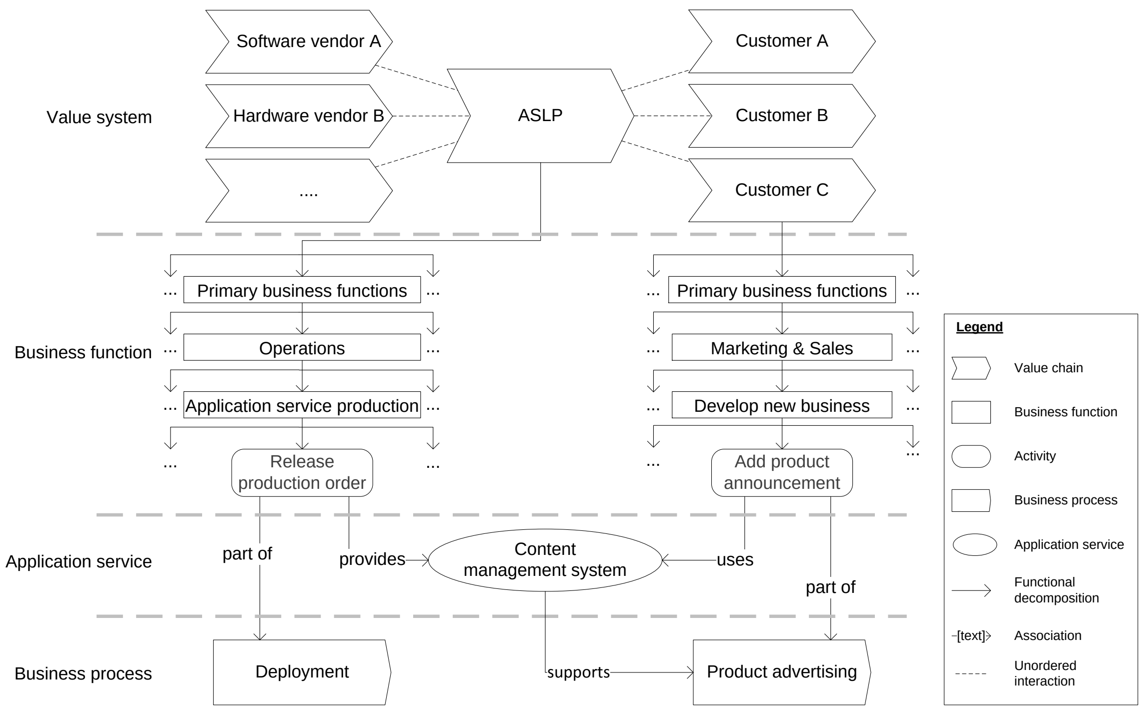


Figure 5.2: Example of an ASLP's value system and subordinate concepts.

For the ASLP, a similar functional decomposition exists: from the most aggregated level of primary business functions (Porter, 1985, p. 37) down to the activity of releasing a production order. This activity provides the application service, which is then used by the customer.

The architecture is constructed from the viewpoint of the ASLP, which is captured in the following architecture decision.

Architecture decision 6
The ISAA shall primarily be designed for application system landscape providers.

Activities can be system activities, user interaction activities, or manual activities (Weske, 2012, p. 74). The system and user interaction activities are executed on an application system. For the ASLP, the application system is one of the three categories: enterprise management, IT service management, or IT service production (cf. architecture decision 1). Standard application software will be the basis for these systems.

Before the business functions and application systems are presented, a domain model will be presented in the following section. "A domain model creates a web of interconnected objects, where each object represents some meaningful individual, whether as large as a corporation or as small as a single line on an order form. [It] mingles data and process, has multivalued attributes and a complex web of associations, and uses inheritance [...]." (Fowler, 2003, p. 116) The domain model serves as a means of explaining the essential entities of the architecture and their relationships between each other. Figure 5.3 shows that the domain model is not associated with a specific layer, but it spans all layers of the architecture. The layers used here correspond to those suggested by Winter and Fischer

(2007) proposed for enterprise architecture, although, here, the term infrastructure is used instead of technology.

The ASLP business model was described in section 4.2 and set into perspective above in this section. After the domain model in the following section, the production process for application services is described in section 5.2. The process includes system and user interaction activities as discussed above. The application system landscape that supports these processes is presented in section 5.3. A new application system, the production execution system, is suggested in section 5.4. It orchestrates the different systems to deploy, change, and terminate the customers' application system landscapes. The domain model will explain how the infrastructure layer is incorporated into the ISAA.

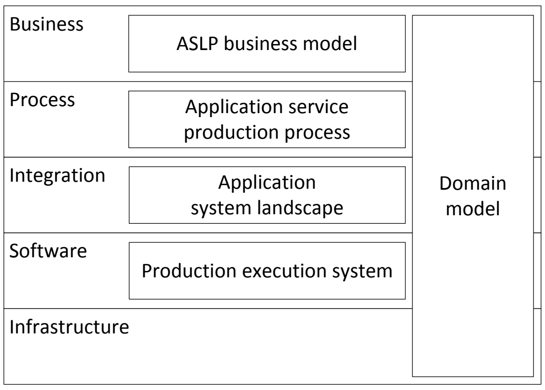

Business
ASLP business model
Process
Application service production process
Integration
Application system landscape
Software
Production execution system
Infrastructure
Domain model

Figure 5.3: Architecture description aligned with standard architecture layers.

To create the architecture description, Unified Modelling Language (UML) diagrams, in accordance with UML 2.5 (Oestereich and Scheithauer, 2012), and the modelling tool Papyrus[1] were used. UML is chosen over more specific tools, such as SysML[2] or Archimate[3], in order to address a broad engineering audience. To model the business processes, process diagrams of the widely used Business Process Model and Notation modeling language, in version 2.0 (Weske, 2012, p. 206), were used. The diagrams were created with the academic version of Signavio Process Manager[4].

5.1 Domain model

The ISAA's domain model cross-cuts the layers. In section 5.1.1, the relevant business concepts, such as application service, resources, and customers are introduced and set in relation to each other. Section 5.1.2 contains a description of how application system landscapes can be deployed with the help of the selected operations automation approach using the introduced resources. In the following section 5.1.3, the involved application systems are first fully introduced. Finally, in section 5.1.4, a mapping of the presented

[1]https://eclipse.org/papyrus/
[2]http://www.omgsysml.org/
[3]http://www.opengroup.org/subjectareas/enterprise/archimate-overview
[4]https://www.signavio.com

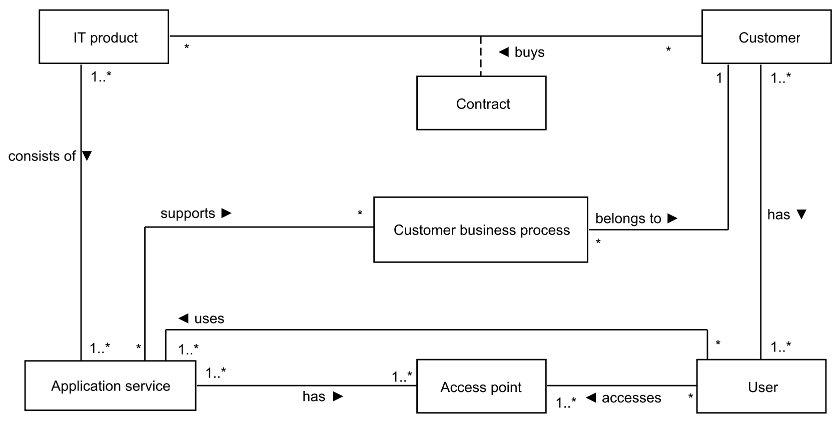


Figure 5.4: ASLP's business relationship with customers.

domain model entities to ERP master data types is presented. Figure A.2 provides a detailed view of the domain model combining all layers.

5.1.1 Business domain

The first part of this layer is depicted in Figure 5.4. It is based on work by Ebert et al. (2007)[5]. One or more application services, which in the case of the ASLP are provided by ASLs, are aggregated to an IT product. Instead of directly selling an application service, an IT product is used as a vehicle to bundle different application services together. The bundling can be useful in order satisfy specific customer demands and declare sales prices that are different from production prices (Zarnekow and Brenner, 2003; Ebert et al., 2007). A customer buys the IT product. It can be an external customer or a different department within the same organization. The application service supports a customer's business process. The customer buys the IT product based on a contract. It contains provisions about the service duration and price of the IT product. The customer has users. This differentiation is important because the user has a different relationship with the ASLP. Users directly employ the application service and require support when problems occur. The provider and the customer are typically responsible for monitoring the quality of the application service (Skene et al., 2010). The perceived quality by the user, however, largely contributes to the image of the provider (Zarnekow, 2007, p. 29). Application services have one or more access points. Such an access point can, for instance, be a standard or a non-standard web interface that is accessed by the users' locally installed client software. It is also possible that all application services are provided via the same access point with

[5]The model of Ebert et al. (2007) was modified in the following way to fit with the ISAA. The IaaS resource subsumes Ebert et al.'s network and hardware entities (cf. architecture decision 5). Their application entity is changed to software, covering a broader spectrum such as the configuration models as well as software in general. The process model is replaced by a more elaborate concept of Weske (2012), including a hierarchy of business function, business process, and activity. The service model can be defined by different compositions of the application service entity, which was renamed from Ebert et al. (2007)'s IT service entity. Their contractor entity is renamed to the word customer, which is more ITIL-conform (Cannon, 2011b, p. 13). Associations were changed by the described modifications, but are not elaborated on here.

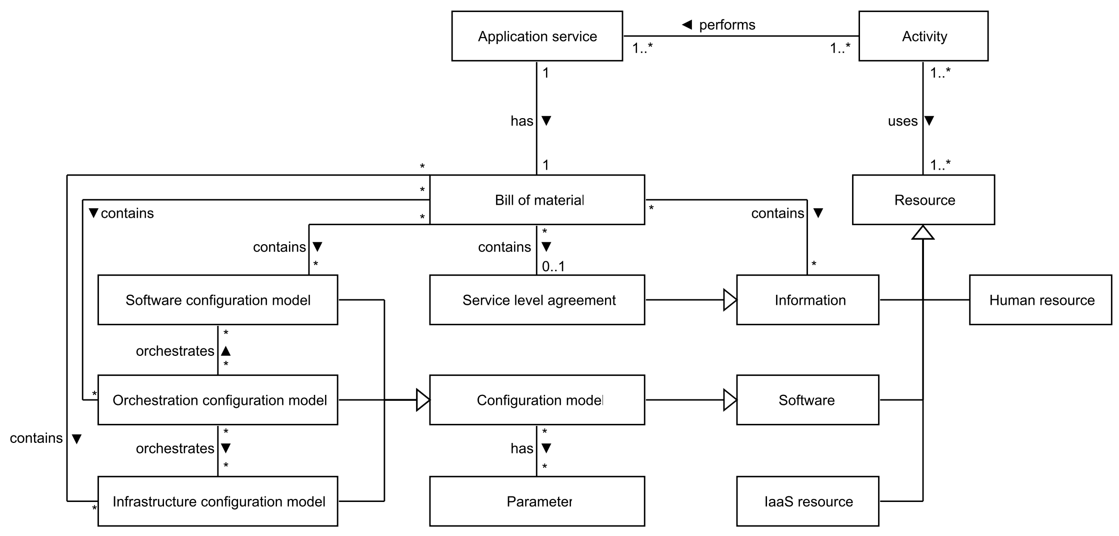


Figure 5.5: Core components of an application service in the ISAA.

different user credentials.

In Figure 5.5 it is shown what resources are used to provide an application service. Several activities are used to perform an application service. These can be human interaction activities or system activities. A system activity of a web-server, for example, is to serve the user a website upon request. As previously shown such activities also exist on higher layers of the stack (cf. Figure 5.1). Activities use four kinds of resources: human resources, software, IaaS resource, and information. Human resources are usually not directly involved in the deployment or operation of application services as this is performed automatically. However, they are required in business functions such as sales or system architecture. Also, if service outages or other incidents occur, they are required. Specific ASLP employee roles will be defined in section 5.2.

Software provides the actual functionality of each application service. This includes programs that directly support an application requirement and that support application software (Stahlknecht and Hasenkamp, 2005, pp. 226–227). However, software, here, may also be a file or a user that was created. IaaS resources form the foundation on which each application system landscape runs. They are further discussed in section 5.1.2. The remaining resource is information.

Information resources may be application licenses required for proprietary software. Furthermore, domain names can be resources which need to be tracked for accurate billing. A particular kind of information is the service level agreement. Different possibilities exist for representing SLAs (Keller and Ludwig, 2003; Berger, 2005; Skene et al., 2010). Looking at the studied cases, company Alpha offers its services with SLAs that are differentiated based on indicators. These indicators are its commitments regarding availability and mean time to restore. The company has three classes (standard to premium) that are sold with its services, regarding both availability and maximum response time. Similarly, Beta offers service levels concerning availability and response times. This simple view of an SLA, corresponding to ITIL's basic SLA requirements (Hunnebeck, 2011, pp. 127–129), is adopted within the scope of this thesis (cf. REQ C.1). As an information resource it may, on the data persistence layer, only contain a constant that states, for example, to which class of response time this service belongs. During service operation, this information is then used, for instance, to respond to incident requests appropriately (cf. REQ P.4). More complex provisions such as customer monitoring and reporting capabilities (Keller and Ludwig, 2003; Skene et al., 2010) are not considered here to ease the comprehensiveness and reduce complexity. The SLA and contract are maintained separately. The contract contains legal provisions as well as the duration and price of the IT product (Cannon, 2011b, p. 432). Legal provisions and price are not within the scope of the SLA resource, which, here, is only used for IT service operation purposes. Treatment of the SLA is summarized in the following architecture decision:

Architecture decision 7
One or no service level agreements shall exist for each application service. In the ISAA, service level agreements only represent the agreed upon service level that pertains to service quality indicators such as availability or maximum response time.

In line with architecture decision 4, configuration models are used as a basis for specifying how application services are made operational. Parameters can be used for mass-customization (BTO scenarios), for instance, to change the secondary storage capacity that is available for a webserver service. Configuration models configure IaaS resources and software. Two configuration model kinds of the ISAA are the software configuration model and the infrastructure configuration model. Whereas the software configuration model configures everything from the running operating system to the application software, the infrastructure configuration model defines the virtualized resources and the base operating system image. The orchestration model coordinates both other models as required (see Figure 5.6).

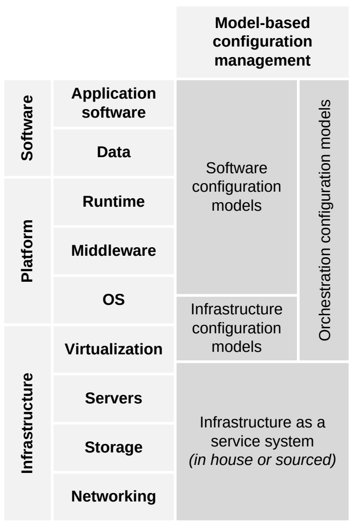

Model-based configuration management
Software
Application software
Data
Platform
Runtime
Middleware
OS
Infrastructure
Virtualization
Servers
Storage
Networking
Software configuration models
Infrastructure configuration models
Orchestration configuration models
Infrastructure as a service system
(in house or sourced)

Figure 5.6: Model-based configuration management on the different layers of the IT stack.

A bill of materials (BOM[6]) is used to list the different resources that make up an application service, leveraging the manufacturing analogy (cf. REQ R.4). It is used as a basis to manage each application service in the enterprise management system and is used as a structure to transport the deployment information into the IT service production system.

The three configuration models are used to fully describe an application service in a way that it can be deployed (cf. REQ C.2). The next section will detail how this configuration is performed.

5.1.2 Operations automation

Figure 5.7 displays how software configuration is conducted on the software layer. Soft-

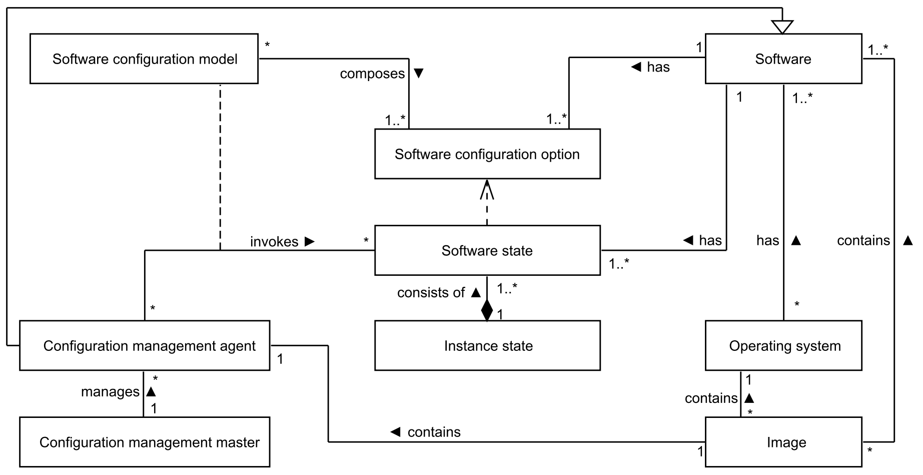


Figure 5.7: Configuration of the software layer.

ware configuration models compose different software configuration options. Configuration models can be modularized as is common in software engineering (cf. section 3.3.1). Configuration options can be as atomic as configuring specific attributes of a file. However, if the complexity of configuration models is abstracted in modules, higher-level configuration options can also be expressed.

Table 5.1 shows examples of different abstraction expressed in the DSLs of three tools for software configuration management. Software configuration on the OS layer can mean changing certain configuration files to configure the OS in the desired way. OS disk mounting configuration is shown in the example where Saltstack[7] is used. The file's system path is defined, its content is read from a defined source, and its access rights are set. On the middleware layer, a Puppet[9] configuration module can be used to set up a new database with a specific name, a user, password, host, and access rights. Moreover and finally, on the layer of application software, three lines of key-value pairs suffice to instruct Ansible[8] to deploy the content management system with a specific administrator username and

[6]cf. Van den Berg (2007, p. 96)
[7]https://saltstack.com/
[8]https://www.ansible.com/

Operating system	Middleware	Application software
`/etc/fstab:` `file.managed:` `- source: salt://` `↪ fstab` `- user: root` `- group: root` `- mode: 644`	`mysql::db { 'wordpress_db':` `user => 'wordpress',` `password => 'pw',` `host => 'localhost',` `grant => ['SELECT',` `'INSERT','UPDATE','DELETE` `↪ '],` `}`	`wp_admin_user:` `↪ admin` `wp_admin_pass: pw` `wp_title: My Title`
Setting mount points with Saltstack[7] (SaltStack, Inc., 2017).	Creating a database with the MySQL configuration module of Puppet (Puppet, Inc., 2017).	Setting up the Wordpress content management system with an Ansible role (Logsdon, 2015).

Table 5.1: Software configuration on different layers using Saltstack, Puppet, and Ansible.

password and a title. Therefore, software configuration options can be atomic as setting file permissions but also perform configuration on the layer of application software.

The software state depends on these options. Examples of components of a software state are a created file, a running or stopped service, or an installed package. The software state makes up one part of the instance state, which will further be described in the following. The software state is invoked by the configuration management agent based on the software configuration model. On every physical or virtual host of an ASL, a configuration management agent is running. A configuration management master manages configuration management agents.

Although operating systems are themselves software, they can contain other software such as system management tools. The operating system is contained in an image that may contain additional pre-installed software.

For the inductive analysis of software configuration management, Puppet was used. Puppet[9] is amongst the frequently named open source tools in this application area (Delaet et al., 2010; Spinellis, 2012; Hintsch et al., 2016a; Wettinger et al., 2016) and is also acknowledged in industry (Fletcher et al., 2016).

Figure 5.8 displays how the configuration model manages the virtualization and some of the OS layer. For the inductive analysis of operations automation on these layers, OpenStack was selected. It is praised by industry analysts (Nelson et al., 2014), has competition-spanning industry support (OpenStack Foundation, 2017), and its adoption rates are increasing (Talligent, 2016).

When using an IaaS system, an image is used to boot an instance. The instance is mostly identical with a virtual machine, but its base images come with integration with the IaaS system. Integration facilitates, for example, authentication and network management (OpenStack Foundation, 2018). Each instance has a lifecycle state of its own, including, for example, such states as started, rebooting, running, or terminated. The instance lifecycle state contributes to the instance state. The instance state may differ from one instant to the next based on the input directed at the instance or the computation currently in

[9]https://puppet.com

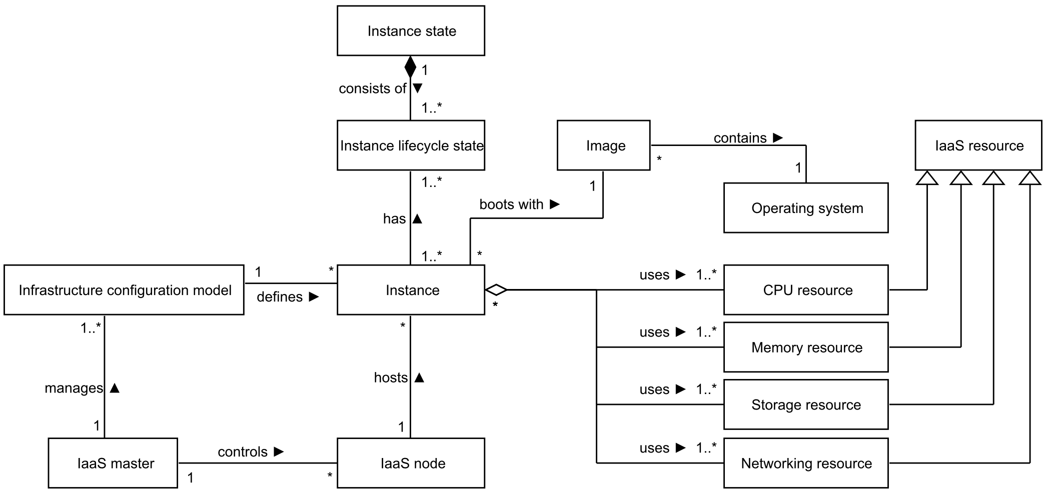


Figure 5.8: Configuration of the infrastructure layer.

progress.

An instance uses different types of IaaS resources. These are CPU, memory, storage, and networking. The available bandwidth quota can differentiate a networking resource. Network resources such as virtual routers or gateways are not depicted in Figure 5.8. Their specifics are not necessary to understanding how a functioning ASL may be set up within the ISAA (Bumgardner, 2016, p. 107).

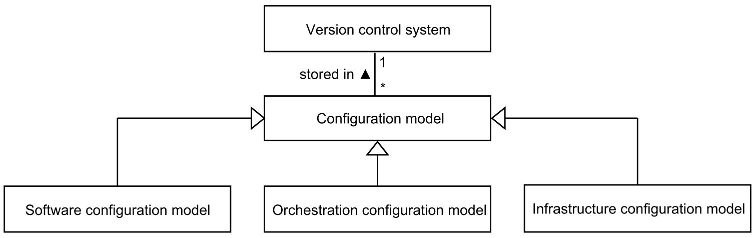

Version control system
stored in ▲
1
*
Configuration model
Software configuration model
Orchestration configuration model
Infrastructure configuration model

Figure 5.9: Version control system.

An infrastructure configuration model defines the instance. Therefore, it defines which IaaS resources the instance comprises and what image it boots from. The infrastructure configuration model is managed by the IaaS master of the IaaS system. The IaaS master also controls the IaaS nodes of the IaaS system that host the instances. The orchestration configuration model orchestrates the other model kinds. In the case of a database server, links the corresponding software configuration model to an infrastructure configuration model, thereby making up the full service. Figure 5.9 shows that all three types of configuration models can be stored using a version control system (Spinellis, 2005). In particular, the software and infrastructure configuration models can be reused between services (cf. REQ S.3).

In the next section, the version control system will be classified with the other central application systems of the ISAA's application system landscape.

5.1.3 Application systems

Several application systems (e.g., IaaS master, version control system, and configuration management master) carrying out the automated ASL production were introduced in the previous parts of the domain model. Figure 5.10 classifies the aforementioned software components by using the application system categories from section 4.1. The Figure also sets the systems in relation to the ERP system, which is leading the production process. The proposed information system architecture narrows the scope to ERP systems for manufacturers (cf. REQ R.4) and the following decision is made:

Architecture decision 8
A standard ERP for manufacturers system shall be used as the enterprise management system.

Other application software such as CRM software may be used in the enterprise management category but is not the primary scope of this architecture. Limited CRM functionality is contained in the ERP system used for analysis.

For the inductive analysis, SAP ERP was selected as the ERP software for manufacturers. It is ranked higher than Oracle Corporation's ERP software by Gartner (Guay et al., 2015) and the two competitors are the only Tier-1 ERP vendors (Simon et al., 2010). Also, SAP ERP's business functionality can be analyzed and extended because all application source code is available in its source when having access to the system (SAP SE, 2016). The assessment of industry accepted analysts addresses REQs R.2 and R.3. Prevalently implemented tools facilitate ERP-based process integration with processes that are not within the scope of the thesis (cf. REQ R.2). Similarly, the integration with organizations' existing technology stacks is facilitated (cf. REQ R.3).

In addition to the three application system categories, a new application system is introduced. It is called the production execution system (PES). The production execution system is similar to manufacturing execution systems that manage the operations on the shop floor of factories producing physical products. Generally, these systems are responsible for the schedule of activities in a factory, management of production activities, document control, data collection and acquisition, labour management (e.g., mapping workers to shifts based on qualifications), quality management, dispatch of production units, maintenance management, product tracking and genealogy, performance analysis, and resource allocation (Saenz de Ugarte et al., 2009). For application service production, the PES acts as an intermediary between the ERP system, the IT service management system, and the IT service production system. The PES orchestrates the IT service production system's sub-systems based on orders in the ERP system (cf. REQ P.1). It initiates billing based on fixed price regardless of volume and per unit price pricing (Iveroth et al., 2013). Furthermore, the PES communicates changes of the application system landscape to the IT service management system. Such changes occur when customer ASLs are added or when changes are applied.

The PES is purposefully designated to be a separate system from the ERP system. As ERP customization and development can be quite costly, this approach should be more cost efficient (Momoh et al., 2010). Data quality is not impeded because the responsibilities of the ERP and production execution system are similar to ERP system's default capabilities of interfacing with a manufacturing execution system to handle production orders (Saenz de Ugarte et al., 2009).

Consequently, the following architecture decision is made:

Architecture decision 9
A production execution system shall act as an intermediary between the three the ERP system, the IT service management system, and the IT service production system. The production execution system shall be implemented separately from the ERP system.

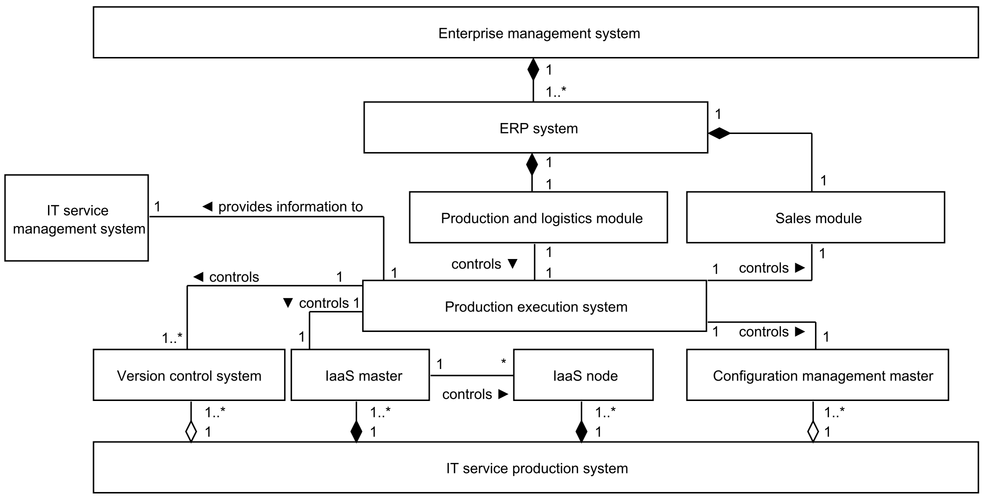
In Figure 5.10, the production execution system and its relationships with other systems are shown.

The ERP system controls the production execution system via its production and logistics module. There is also a reverse connection to the ERP system. The PES controls the sales module, mainly for billing purposes. This will be explained in more detail in section 5.3. The PES also controls the previously introduced software components to configure and deploy the ASLs. Furthermore, relationships between the IT service production system's subsystems exist: the IaaS master manages one or more IaaS nodes. Multiple instances can run on one physical server, or a bare metal instance is used (OpenStack Foundation, 2011). The interfaces, indicated by the control-label associations in the diagram, are usually network based. Of course, both depicted ERP system modules are also interconnected (Gronau, 2010, p. 53), but since these connections are not specific to this architecture, they are not shown in Figure 5.10. In the analyzed ERP software by SAP, interfaces of the standard modules are also not network based and do not require additional attention within the scope of this architecture (Gronau, 2010, p. 41). If a fully service-oriented ERP system or an enterprise management system with several standalone process support systems (e.g., one system for production and one for sales) is used, the interface may be network based. However, for the analyzed ERP software, all modules can be hosted in the software's application server component, which is installed on one instance. The components used by the production execution system as well as the system itself make up the IT service production system.

The primary data entity types that will be used in the architecture are those from the ERP software's master data entity types. Therefore, in the next section, a mapping of entities defined in the domain model to the ERP software's master data entity types will be presented.

5.1.4 Mapping of domain entities to ERP master data

A data model is required to store application services and their components. Following the manufacturing analogy (cf. REQ R.4), this information is stored as master data in



Enterprise management system
ERP system
IT service management system
provides information to
Production and logistics module
Sales module
controls
Production execution system
Version control system
IaaS master
IaaS node
Configuration management master
IT service production system

Figure 5.10: Application systems that are central to ASL production and their typically network-based connections (control associations).

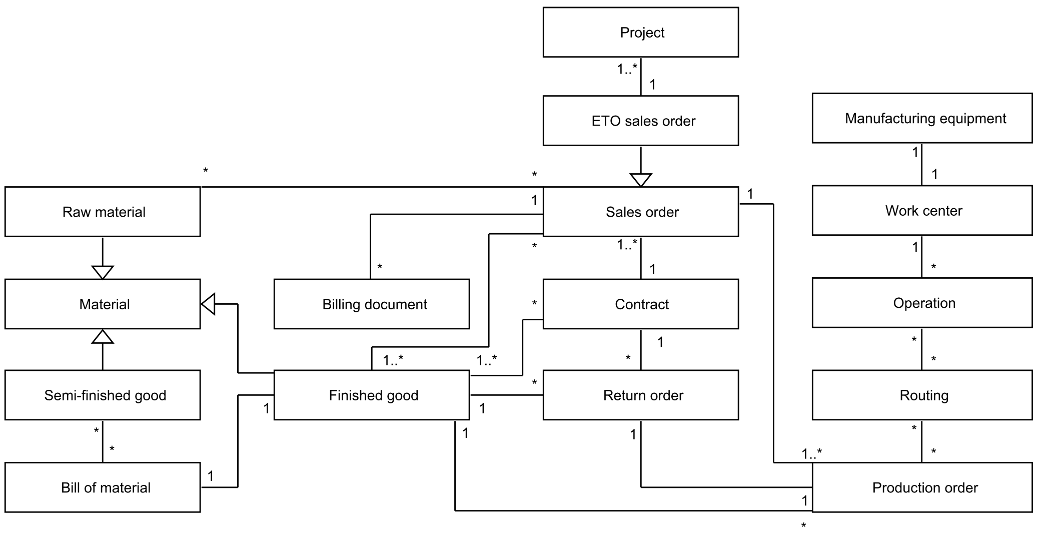


Figure 5.11: ERP master data entity types and their relationships.

the ERP system. Figure 5.11 shows the entity types and the relationships which are used in the ISAA. The mapping of these entity types to entities of the domain model is depicted in Figure 5.12. In addition to the BOM, important master data entity types in computer-integrated manufacturing (Scheer, 1997, p. 93) are the following. A routing defines operations (Sheikh, 2003, p. 446) required to transform raw materials and semi-finished goods into finished goods. Finished goods are sold to customers as products (Van den Berg, 2007, p. 56). To manufacture a good, a production order (Scheer, 1997, p. 235) triggers the start of the manufacturing process on the shop floor of a factory as defined by the routing. The actual transformation from raw material into semi-finished goods is performed with different kinds of manufacturing equipment and the human resources in a work center. A routing may have multiple operations that describe the tasks that are performed in different work centers.

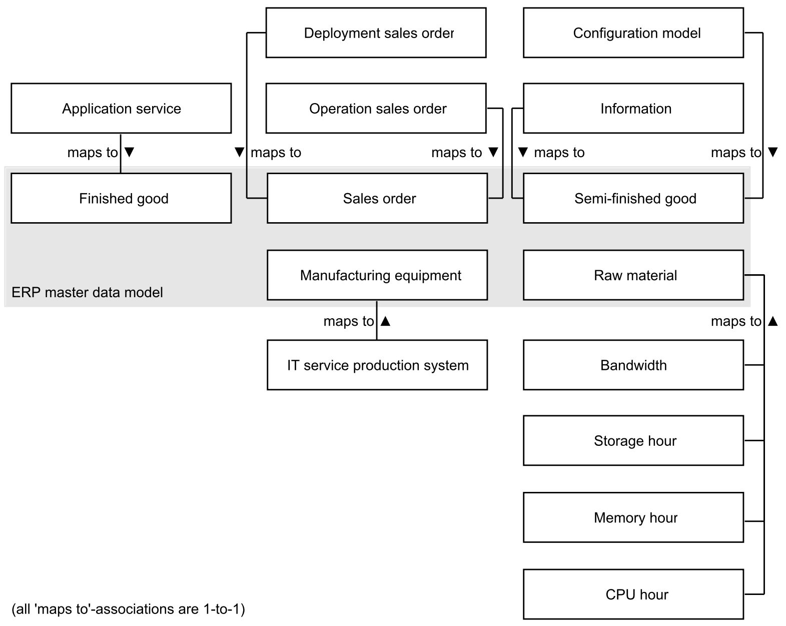


Figure 5.12: Master data mapping (not all associations between standard master data types are shown).

One work center is always related to precisely one piece of manufacturing equipment: the production execution system. While in manufacturing, different equipment is maintained in the system, this is not necessary for the scope of the ISAA, and this is different from previous concepts. Ebert (2009, p. 142) modeled infrastructure resources as equipment to be able to plan capacity. However, when adopting the IaaS system as a base for IaaS resources (cf. architecture decision 5), infrastructure capacity planning can be delegated to an internal or external IaaS provider. For their provisioning of capacity, a management approach such as suggested by Ebert (2009, p. 142) may be selected. When operating the actual IaaS system, the knowledge of actual servers is essential. Within the ISAA, for producing application services, the infrastructure configuration models abstract this detail. If different infrastructure resources are required, these are identified by different infrastructure configuration models. Consider the example shown in Figure 5.13. There, to operate the IT product *fast content management system*, different infrastructure configuration models are available. A bare metal version of a standard SUSE Linux Enterprise Server (SLES) is selected. Therefore, bare metal IaaS resources are billed. For both other infrastructure configuration models, virtual IaaS resources would be billed. Consumption of IaaS resources is only modeled based on the four material types (CPU hour, memory hour, storage hour, and bandwidth hour), which may differ only between different types instances (e.g., hypervisor-based instances or bare-metal instances, or standard instances or high-availability instances).

For each deployment, all components of the production execution system will be employed. The configuration models define the tasks which the components need to perform. For instance, the orchestration configuration model might instruct the IaaS master to spawn an instance based on a given infrastructure configuration model. Therefore, while

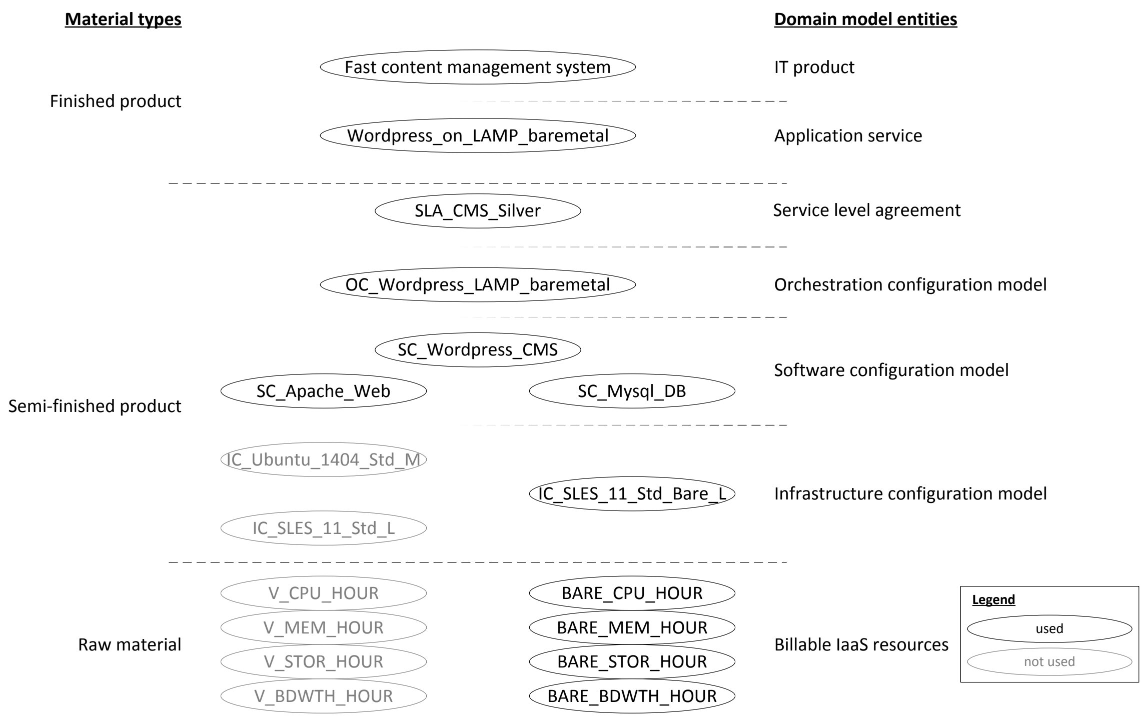


Figure 5.13: Example of different materials making up the IT product *fast content management system.*

configuration models contain the information for application service production, for manufacturing, this information would be contained in the routings. Human resources, such as operators, are usually not involved in the automatic deployment and operation of the application services (cf. REQ P.1). Table 5.2 presents a comparison of how central domain entities are used differently in the domains of ASL and physical goods production.

Within this thesis, a mass customization approach (Ahmad et al., 2010) is followed where standardized products and processes are employed while still offering a degree of variability (cf. REQ S.2). Different methods exist to achieve this. The most individual products will be deployed in engineer-to-order scenarios. For these products, separate corresponding BOMs will be created. Generating BOMs is one way of satisfying different customer demands. However, if the same product or a very similar product is sold, build-to-order scenarios will be favorable because there will be less administrative overhead, for instance, because no BOMs need to be created (Ahmad et al., 2010). To support variation in production, configurable BOMs are used (Scheer, 1997, pp. 120–127). Differentiating ETO and BTO scenarios has consequences for the construction of the application service production process. Hence, this architecture decision is made explicit:

Entities of the domain	**Application system landscape production**	**Physical goods production**
Bill of materials	Contains all configuration models necessary to automatically deploy and operate the application service, as well as SLAs and other information resources	Contains all raw materials and semi-finished goods necessary to manufacture (semi-)finished goods
Routings	Contains operations such as acceptance testing and deployment	Contains all operations necessary to transform a good
Manufacturing equipment	An IT service production system consisting of different components (see Figure 5.10)	Different machines that might be orchestrated by a manufacturing execution system based on routings

Table 5.2: Examples of master data types for application system landscape production in contrast with physical goods production.

Architecture decision 10
Application services shall be provided in two scenarios. First, if customers have particular requirements that cannot be served based on the existing configuration models, the *engineer-to-order* scenario applies. Secondly, in cases where customers can be served from the existing portfolio, the *build-to-order* scenario applies.

Different functional properties of application system landscapes may include user accounts, domain names, or activated application software modules. Non-functional properties may include scaling policies for up and down or for out and in as well as different SLAs. Such variations may be implemented with configurable BOMs (SAP SE, 2013). To deploy an application service, a production order needs to be created which contains corresponding routings. Within the ERP system, the application service is maintained as a finished good. The BOM includes the SLA, information resources, and semi-finished goods, which are the three kinds of configuration models. A manufacturing plant produces physical finished goods that are sold to customers. The term servitization for such companies describes the trend of selling additional services to customers after the main physical product was sold (Zolnowski et al., 2011). ASLPs deploy and then operate their customer services. In this sense, the deployment phase is comparable to the shipment phase, and the operation phase is comparable to the after-sales relationship with the customer in manufacturing. The question arises of how to map these lifecycle phases of application services to ERP systems for manufacturers.

Therefore, when comparing IT services to physically manufactured goods, one fundamental difference is the duration of a service. A physical product is manufactured, sold and shipped to the customer. Apart from maintenance, the manufacturer may not come into contact with the sold items again. In contrast, the IT service is deployed and then operated for a specified or unspecified period by the IT service provider. When using an ERP system for manufacturing this duration has to be represented. Two alternatives exist for modeling this, using the presented master data entity types.

The first alternative is to use a production order to model the duration of the IT service. Routings could define the different lifecycle stages of the IT service. One routing would exist for deployment, one for operation, and one for termination. In case of a three-month contract, the routing for the operation would have a processing time of three months. This would mean that the production order would be open (the period from release to confirmation) for three months. However, this alternative has the disadvantage that the ERP logic of material flows would not be observed. Also, performance indicators such as work in progress would not be usable.

The second alternative is to map the duration of the IT service to a contract. Such contracts have a duration and a stipulated quantity of goods that the customer buys from the provider (SAP SE, 2017c). In this way the production order has a routing for deployment, and after successful deployment, the production order is confirmed successfully, the operation phase starts, and its duration is modeled in the ERP system by the duration in the contract and the corresponding amount of billable IaaS resources. In this scenario, the production order is confirmed when the application service is fully deployed. Therefore, the period in which the production order is open equals the time of deployment and can also be used to monitor deployment times (cf. REQ P.2). This alternative is chosen as stated in the following architecture decision (cf. REQ C.1):

Architecture decision 11
The duration of an application service shall be modeled in a contract.

This decision is made because, from the provider's perspective, the task of making service available to the user is complete when the deployment is completed. Hence, the production order confirmation marks this point. After that, the IT service production system is set up and does not need intervention unless a change becomes necessary.

The contract specifies the business relationship between customer and provider. This contract has one or more deployment and operation sales orders, each having their billing documents (cf. REQ P.5). ETO sales orders are specializations of deployment sales orders that are associated with projects under which engineering activities for individual application services are conducted (cf. REQ P.3). While one deployment sales order is usually created per contract to deploy the application service, multiple operation sales orders are generated during the operation phase of the service (cf. REQ P.4). These operation sales orders bundle together the raw materials, including CPU, memory, and storage hours, as well as the networking bandwidth that was consumed during a billing interval.

If a change becomes necessary, this is handled with a return order (SAP SE, 2017b). The next section will describe the application service production process, including the operation phase, in which changes are performed to the service.

5.2 Application service production

Based on the defined domain model, an overall production process will be defined to manage inquiries, provide services, and bill accounts. It will show, how application services can systematically be produced based on the operations automation approach and configuration models.

Traditional service lifecycle models of IT service management consider the overall application system landscape of an IT service provider (Praeg and Spath, 2008) without making prescriptions concerning how this ASL should be structured. For example, consider an internal IT service provider who operates one ERP system for its manufacturing business. The borders between this provider's IT service production system and the actual service it is providing are less clear-cut than the systems of an ASLP that has implemented this architecture. The IT service production system of the internal IT service provider, which also comprises the ERP system, could offer services to support payroll. Adding new or changing existing ERP services to the service catalog, in this example, means to manipulate the central part of the IT service production system: the ERP system and possibly also the underlying systems.

For an ASLP, service additions or modifications may require a change to the IT service production system, for instance, if a new function in the IaaS system is required. However, the IT service production system should stay constant in its architecture and general configuration apart from things such as adding hardware resources to increase capacity. Customer requirements should be encoded in the configuration models of the application service. Cloud computing services usually strive for multi-tenancy to reduce costs, but this constrains the variability of offered services (Rimal et al., 2011; Mietzner et al., 2011). ASLPs provide application system landscapes, and multi-tenancy is only explicitly required at the infrastructure layer (provisioning of instances). The service portfolio, consequently, is not restrained in terms of variability (cf. REQ R.1). Lifecycle models for cloud computing services, as suggested by Breiter and Behrendt (2009), are scoped to multi-tenant services. Because of the different scope of both general IT service management and cloud computing, the high-level business process for application service production is derived from generic frameworks of manufacturing and IT service management. In the context of this architecture, cloud computing can be seen as an enabling paradigm combining previous research results from service-oriented computing, service science, hardware virtualization, and model-driven software engineering.

For specific services (the ETO scenario), the primary activities of the value chain (Porter, 1985, p. 37) of an ASLP need to be tightly integrated. Compared with physical goods, any IT service has three service characteristics (Praeg and Spath, 2008): services are intangible, the provisioning and consumption of services co-occur (uno-actu principle), and external factors are integrated (e.g., the customer). Therefore, although industrial IT service production is comparable to manufacturing, the sales process is very different because it needs to consider a continuous relationship with the customer during the time of operation. An IT service cannot be stored, marketed, and then sold, as it is possible for physical goods. IT services are consumed while they are produced (uno-actu principle). Activities such as sales and order processing occur before the IT service is realized.

Therefore, the overall production process for application system landscapes of the proposed information system architecture is framed by a textbook order-to-cash process. With this, it is acknowledged that before any IT service provisioning activity occurs, some pre-sales or order processing activity needs to take place. Two process frameworks iteratively enhance the process. One focuses on production in a supply chain, while the other specifically concentrates on IT service management. Figure 5.14 displays this iterative development.

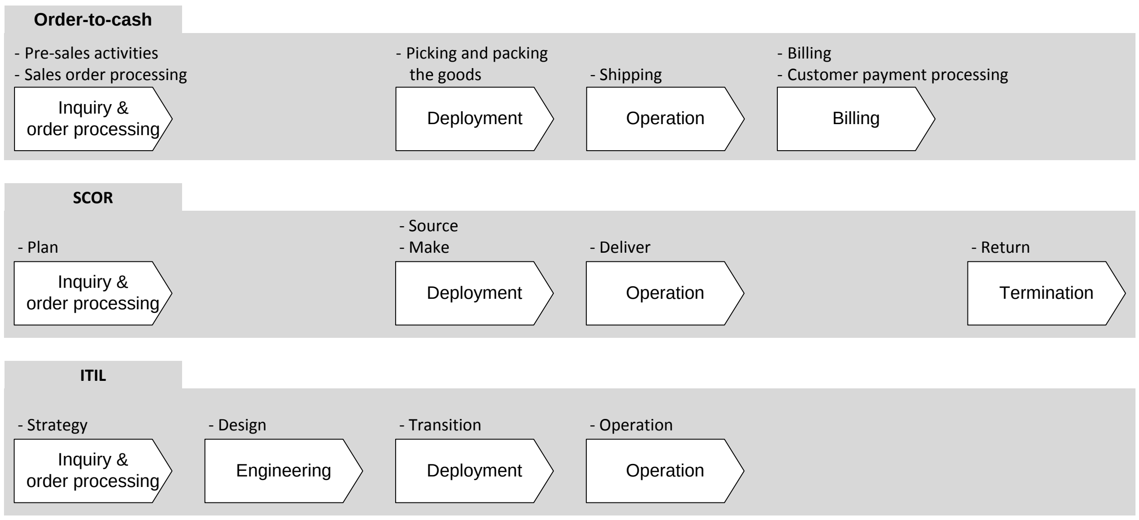


Figure 5.14: Development of the overall production process based on three relevant process frameworks.

The order-to-cash process differentiates pre-sales and sales order processing activities. Pre-sales activities in BTO scenarios may also be conducted by displaying different variations of the application service in a webshop. In such cases, customers may place their orders themselves. In ETO scenarios, providers must repeatedly engage customers to understand their needs and discover how to deliver satisfactory services, which is similar to iterative software engineering process models (Boehm, 1988). Therefore, the term *inquiry* is used to show that the request of a customer needs to be addressed and if the customer makes a purchase decision, the order has to be handled as well. However, the intricacies of order management are only summarized here. Picking and packing the goods is the activity that occurs before shipment. An application service deployment phase is mapped to this. After an order has been picked and packed, it is ready to be shipped. Shipment in the order-to-cash process is similar to the operation phase of IT services. While a product is only shipped once, the IT service is continuously operated. During billing and customer payment processing, this continuity of the IT service needs to be considered.

Previous work suggested to transfer the Supply Chain Operations Reference (SCOR) model to IT service provisioning (Hochstein and Uebernickel, 2006). SCOR adds the concept of a full product lifecycle. Here, the return phase is relevant because customers may

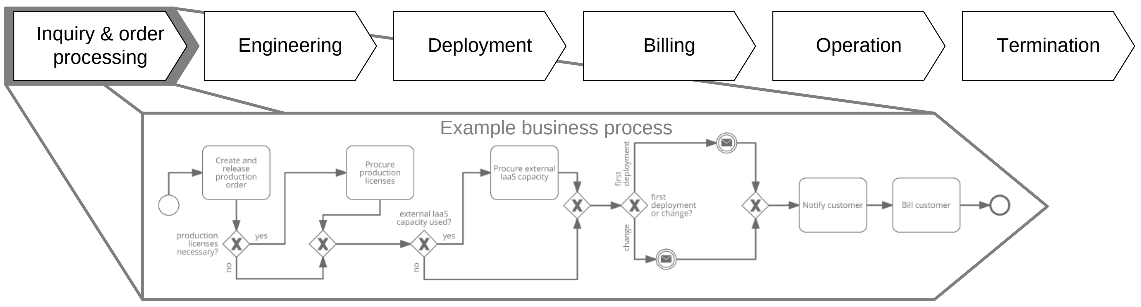

Inquiry & order processing
Engineering
Deployment
Billing
Operation
Termination
Example business process
Create and release production order
production licenses necessary?
yes
no
Procure production licenses
external IaaS capacity used?
yes
no
Procure external IaaS capacity
first deployment
first deployment or change?
change
Notify customer
Bill customer

Figure 5.15: Application service production process.

request that their data is made accessible at the end of a contract to transfer the data and continue using a similar service at another provider. From the provider's perspective, application services have to be terminated in a structured fashion when the customer no longer requires them. This termination has to be reflected in the systems of the provider. Although sourcing (Supply Chain Council, 2010, p. 1.2.1) is not mapped as its phase, external IaaS resources may be sourced, for example in the deployment phase. The method and timing of sourcing depend on the concrete business model of the ASLP and the sourced products and services. For example, the choice between different external IaaS providers in the sourcing strategy should consider different aspects. Despite x86 being the predominant processor architecture the technologies between providers become less open for migration further up the technology stack. For instance, Rackspace uses OpenStack as a basis for its IaaS services while Amazon uses its software. Although the OpenStack open source community strives towards compatibility with Amazon's orchestration format, neither complete nor, more importantly, bidirectional compatibility have been accomplished (OpenStack Foundation, 2012). Therefore, the sourcing strategy, which should not be reconsidered for each customer, is an integral part of the business.

ITIL has five central lifecycle phases: service strategy, service design, service transition, service operation, and continual service improvement (Cannon, 2011b, p. 28). The framework enhances this architecture's overall production process with an engineering phase where the application service is devised according to the customers' requirements. Continuous improvement is not represented in the illustration. ERP systems implicitly address it as one of their benefits is cycle time reduction (Staehr et al., 2012). Zarnekow (Zarnekow, 2007, p. 108) differentiates between IT service providers targeting an anonymous market (BTO case) and those that serve specific customer requirements (ETO case). In this regard, in addition to the customer engagement activities of inquiry and order processing, marketing and market research are essential activities. Their discussion, however, is out of the scope of this thesis. For the BTO case, it is assumed that they occur before the process is executed. For the ETO case, marketing will have been carried out in some form before process execution. However, the customer requirements and the competencies of the provider need to be considered during inquiry, order processing, and engineering. Activities in the engineering phase are carried out in two cases. The first case is when a new service is engineered in an ETO setting for a specific customer, or a new BTO service is engineered for an anonymous market. The second case is when existing services are extended or modified due to feature requests or software faults. In both cases, software engineering activities would be conducted which are based on the configuration models previously described. Figure 5.15 shows the overall process for producing application services based on application system landscapes. It also illustrates that each high-level business process needs to be specified in a more detailed business process. The constructed application service production process is documented in the following architecture decision.

Architecture decision 12
The high-level application service production process shall consist out of the following more fine-grained business processes: Inquiry and order processing, engineering, deployment and billing, operation, and termination.

Each of the high-level business processes will be detailed in business process diagrams in sections 10 - 5.2.5. Four roles are differentiated in these process descriptions, indicated by the activity partitions: administration, architecture, development, and operations. These roles could be further differentiated, but for the sake of simplicity, responsibilities are limited to these four. They are aligned with the responsibilities necessary to perform the activities of each process phase. In BTO scenarios, all activities should be either automated or performed by the administration. In contrast, in ETO scenarios, the other roles will be required as well. The administration role is an aggregate of sales and administrative staff responsible for creating sales orders, procuring, initiating deployment, and billing. The architect designs the application service. The responsibility of this role is to communicate with customers and developers to clarify customer requirements. The architect will work with developers only if new software functionality needs to be implemented. The operations professionals are responsible for supplying the configuration models that are required for deploying and operating the application system landscapes of the customers. Development and operations reflect the current DevOps philosophy (cf. section 3.3.1). The defined roles are reflected in the following architecture decision:

Architecture decision 13
The application service production process shall have the roles of administration, architecture, development, and operations.

Inquiry and order processing will be detailed in the following.

5.2.1 Inquiry and order processing

The business process of inquiry and order processing starts with an incoming customer inquiry. The process is depicted in Figure 5.16. In the first step, the customer requirements are recorded.

In the BTO scenario, the requirements can be recorded by a customer who visits a web portal, views the service catalog, and selects the service. A call center agent could

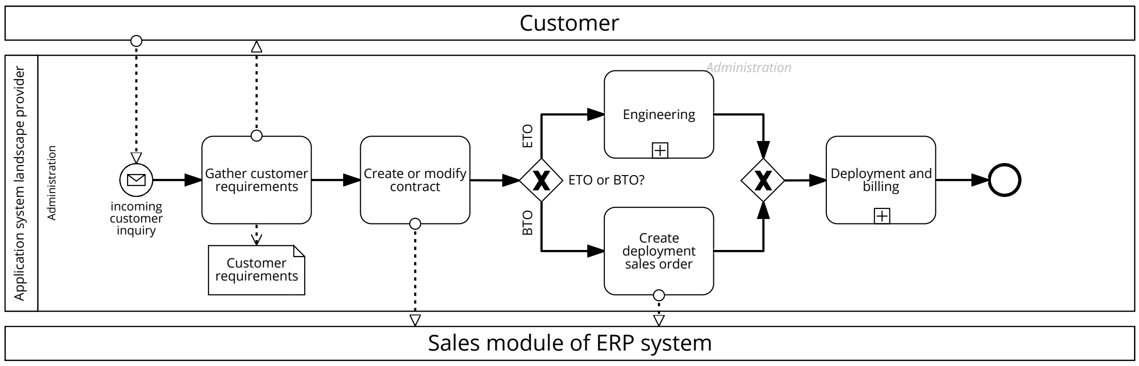

Customer
Application system landscape provider
Administration
incoming customer inquiry
Gather customer requirements
Customer requirements
Create or modify contract
ETO
BTO
ETO or BTO?
Engineering
Create deployment sales order
Deployment and billing
Sales module of ERP system

Figure 5.16: Inquiry and order processing process.

facilitate the customer (Ward et al., 2008). In cases where the service catalog is insufficient, the process continues as ETO. For ETO, gathering the customer requirements may be accompanied by a pre-sales workshop. Depending on whether it is a BTO or ETO scenario, the customer requirements can be coded in a database entry of a webshop system, or it can be a semi-structured document stored along with an inquiry made in the ERP system. If required the customer inquiry could also be answered by a quotation from the provider, which is not modeled, but easily addable Gronau (2010, p. 87).

In the BTO scenario, the customer requirements include infrastructure sizing, duration of the contract and specific services. After the customer has been informed about the application service portfolio in a BTO scenario, the requirements, for example, regarding infrastructure sizing, duration of the contract and specific services are gathered. Based on these requirements, a contract is created. A contract may also be modified if the process is executed as part of the change process that will be discussed in section 5.2.4.

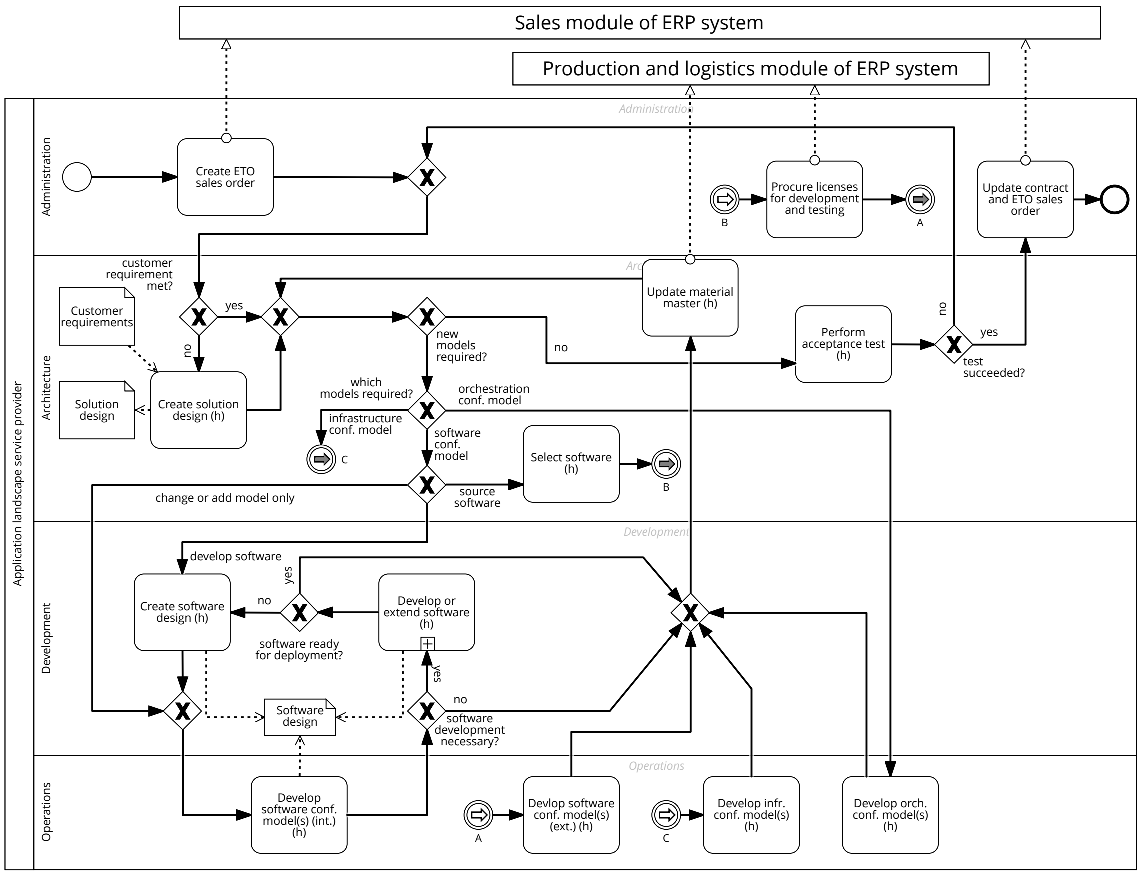
The contract will contain a material that contains application service to be deployed. Furthermore, it will contain materials representing the billable IaaS resources. If a fixed duration is agreed, then the required amount can be added to the contract. For example, for an application service that is running for one month with an instance with one virtual CPU, a total of 720 units of CPU hour material would be added to its contract[10]. In a BTO scenario, the next activity is to create a deployment sales order which contains the application service to be deployed. It is inherited from the contract. The only part of an application system used in this process is the sales module of the ERP system. The provider's capacity is not considered at this stage. As previously discussed, this is a valid assumption considering the cloud computing paradigm with seemingly infinite resources (Armbrust et al., 2010). The production capacity of the provider is constrained only by the available IaaS capacity. The configuration models have infinite stock. A limited stock could exist for licenses, domain names, or public IP[11] addresses in some business cases. Whether or not domain names and public IP addresses are required will also depend on how the provider's access point is designed. All customers could use a central access point. Alternatively, customers or even users could be provided with their customized access

[10]1 CPU * 24 hours * 30 days in a month = 720 CPU hours
[11]Internet protocol (IP)

points. After the deployment sales order has been placed, the deployment and billing process starts, which is depicted in Figure 5.19. In an ETO scenario, the engineering process is performed before deployment and billing. It is described in the next section.

5.2.2 Engineering

The first activity of the engineering process is to create a special kind of sales order. The engineering process is depicted in Figure 5.17. It is unknown at this point which



Sales module of ERP system
Production and logistics module of ERP system
Application landscape service provider
Administration
Architecture
Development
Operations
Create ETO sales order
Procure licenses for development and testing
Update contract and ETO sales order
B
A
customer requirement met?
yes
no
Customer requirements
Solution design
Create solution design (h)
Update material master (h)
new models required?
no
Perform acceptance test (h)
test succeeded?
which models required?
orchestration conf. model
infrastructure conf. model
C
software conf. model
Select software (h)
source software
change or add model only
develop software
Create software design (h)
software ready for deployment?
Develop or extend software (h)
Software design
software development necessary?
Develop software conf. model(s) (int.) (h)
Devlop software conf. model(s) (ext.) (h)
Develop infr. conf. model(s) (h)
Develop orch. conf. model(s) (h)

Figure 5.17: Engineering process.

application service will be sold. Therefore, not all of the required materials may be in the material master. When the ETO sales order is created, a project is created with it. This project is used to financially track the hours spent on developing the required configuration models.

The *h* in parentheses shown in some of the activities in the process diagram abbreviates hours. It indicates that for those activities hours spent working on them by employees are tracked.

After the ETO sales order has been created, a loop starts that tests if all customer requirements have been met. In its first step, a solution design is created. This design can be an architecture description of the service's application system landscape, similar to a computer-aided design model. Possibly, the customer is inquired again to gather more requirements data. Application system landscapes can be drafted based on the models that are available in the ERP system's material master. Also, it can be indicated that existing models need to be modified or new ones created.

An example of a high-level part of architecture description is shown in Figure 5.18. In this example, three software configuration models already exist: one for Apache HTTP[12] Server[13], one for the database server MySQL[14], and one for the application software itself, in this case WordPress[15]. The application server and the database are placed onto two instances that are defined to be bare metal, have a performance class L, and have SLES as an operating system. The main task of this service is to create the orchestration configuration model, but also the database software configuration model needs to be modified.

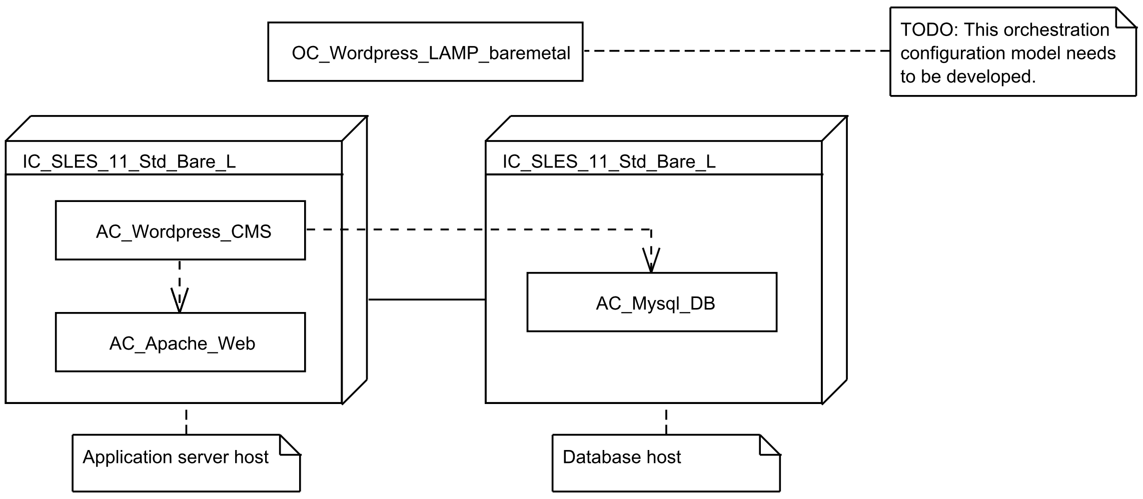


Figure 5.18: An architecture description for the fast content management system as a UML deployment diagram.

When the solution design is created, it needs to be determined whether or not the material master contains all configuration models required to deploy the required application system landscape. Alternatives exist for developing the three kinds of configuration model.

In the *first alternative*, software configuration models are required. Three subsequent alternatives exist for continuing the process.

The *first subsequent alternative* is that either a new software program needs to be developed or an existing one extended. For this, a software design is created. A model-based approach for the software design document would be advisable (Bordeleau, 2014). The models guiding and assisting the software development process (e.g., use case diagrams, class diagrams, or activity diagrams) could be used to inform the creation of the configuration models (software, infrastructure, and orchestration configuration models)

[12]Hypertext Transfer Protocol (HTTP)
[13]https://httpd.apache.org/
[14]MySQL (https://www.mysql.com/) is an open source relational database management system that is named after the Structured Query Language (SQL).
[15]https://wordpress.com/

(Herden, 2013, p. 61). Subsequently, one or more software configuration models are developed. These models are needed for deploying the software programs, but they are also required for testing during the development phase. With these models, the production landscape of the application service can be approximated so that it can even be used for local testing by each developer (Spinellis, 2012). Subsequently, the software is developed. Potentially, standard application software is enhanced, or an application software needs to be developed from scratch. Different software engineering process models can be used for development. The diagram indicates a sub-process. It is not further elaborated on because application software development is not in the thesis' focus. DevOps-aligned models are supported because configuration models to deploy development and test environments for the software are available. Automated unit and integration tests, of course, also have to be developed (Humble and Farley, 2010, p. 135). The developers decide whether or not the software is ready to be deployed (Beyer et al., 2016). The previously developed configuration model is then added to the material master.

The *second subsequent alternative* is that a software configuration model only needs to be added or changed. For instance, a software configuration model for a standard database software might be extended by a configuration option that exposes an existing, but a previously unused feature of the database software. This path omits the steps of software design creation and software development.

The *third subsequent alternative* is when software functionality is sourced. First, software is selected from available vendors. When a vendor has been chosen, licenses are procured to enable development and testing of the software programs. After that, software configuration models are developed which are used to deploy the application software in testing and production environments. This step is necessary as software companies usually do not provide configuration models for their software. Therefore, the operations professionals need to develop appropriate configuration models that make the sourced software automatically deployable in their implementation of the ISAA. Again the new model is added to the material master. The two other configuration model alternatives are infrastructure and orchestration configuration models.

In the *second alternative*, an infrastructure configuration model might need to be developed if a previously unused operating system needs to be deployed or if a new instance performance class is required.

In the *third alternative*, an orchestration configuration model might need to be developed in a case as shown in the example of Figure 5.18. There, software and infrastructure configuration models already existed but needed to be orchestrated for this new service.

An acceptance test is performed when no more new models need to be added. The acceptance test can be performed in multiple stages. For example, first, an approximated environment could be set up, and the final tests are executed in the production environment (Everett and McLeod, 2007, p. 155). The final activity is to update the contract and ETO sales order with the newly created application service. Corresponding billable IaaS

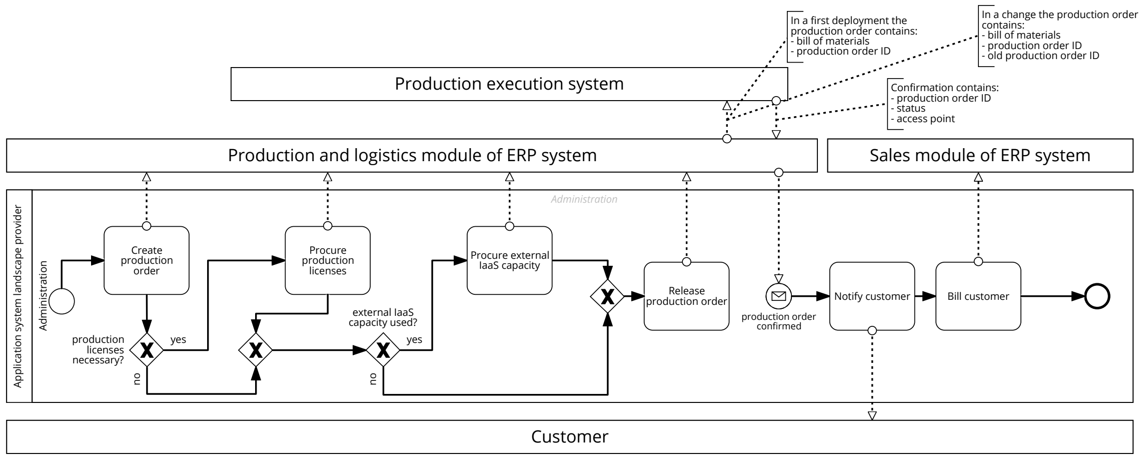

In a first deployment the production order contains:
- bill of materials
- production order ID
In a change the production order contains:
- bill of materials
- production order ID
- old production order ID
Production execution system
Confirmation contains:
- production order ID
- status
- access point
Production and logistics module of ERP system
Sales module of ERP system
Application system landscape provider
Administration
Administration
Create production order
production licenses necessary?
yes
no
Procure production licenses
external IaaS capacity used?
yes
no
Procure external IaaS capacity
Release production order
production order confirmed
Notify customer
Bill customer
Customer

Figure 5.19: Deployment and billing process.

resources are also added.

The next section will outline how deployment and billing are conducted.

5.2.3 Deployment and billing

The deployment and billing process is shown in Figure 5.19. In the first activity, a production order is created based on a deployment sales order. It contains the application service's material which will be used as information for deploying the customer's application system landscape. If the deployment is a result of change the old production order ID is also added to the production order for further processing by the production execution system. In the second step, if production licenses are necessary, they are procured. License procurement may not be necessary, even if proprietary software is used. For instance, the provider could have a stock of licenses that are used for serving different customers over time. In the third step, if external IaaS capacity is required, it is also procured. External IaaS capacity may be required if the provider cannot meet a performance requirement for a geographically dispersed application system landscape in the necessary way (Dikaiakos et al., 2009). Internal orders track internal IaaS capacity consumption. Next, the application system landscape is deployed into production by releasing the production order. The release triggers the production and logistics module of the ERP system to send the production order to the production execution system (the handling of production orders by the PES will be discussed in section 5.4). Next, the customer is notified that the purchased application service may be accessed. The last step is the billing of the customer. This billing (cf. REQ P.5) is performed for the first deployment of the landscape. Set-up costs or engineering costs are only billed once. The production execution system performs continuous billing. The continuous billing by the PES will also be explained in more detail in the next section. Adhering to the provisions made by Iveroth et al. (2013), the following architecture decision is made.

Architecture decision 14
Two styles of application service settlement shall be supported: pay-per-use and fixed price regardless of volume.

Whereas the first alternative is often used in today's cloud computing, the second alternative would be found in more traditional service models. However, also for the second option, infrastructure usage limits can be placed.

In the next section, the operation phase of the application service will be described.

5.2.4 Operation

ITIL defines the following processes for the service operation phase: event management, incident management, problem management, request fulfillment, and access management (Cannon, 2011a, p. 57 - p. 118). Event management is concerned with monitoring the status of IT services and their supporting configuration items. In incident management, incidents such as "[...] unplanned interruptions to an IT service or reduction in the quality of an IT service or a failure of a CI that has not yet impacted an IT service [...]" (Cannon, 2011a, p. 72) are managed throughout their lifecycle. Causes of incidents are problems, which are managed by problem management. In request fulfillment, requests made, usually by users, are tried to be answered in correspondence with the contract and SLA. Requests can range from asking for a password reset to requiring completely or partially new services. Finally, access management is dedicated to "[...] granting authorized users the right to use a service, while preventing access to non-authorized users." (Cannon, 2011a, p. 110). These processes are often seen as the core of the IT service management processes (Marrone et al., 2014). Therefore, they are also part of popular ITSM application software products (Pink Elephant, 2017). As long as an application service is performing as required, no action is necessary. However, incidents and requests can invoke changes or additions to the customer's application system landscape necessary. How such changes or additions are handled in the ISAA is depicted in Figure 5.20.

All actions have to be authorized in the IT service management system. For example, if the action is based on a service request by a user, it is assumed that this request has been authorized in the IT service management system. It does not make sense to shift all activities to the ERP system. The contract's information can be integrated into the IT service management system (e.g., via a role-based read-only web service) as defined in architecture decision 3.

The first step after an incident or service request triggers the process is to decide whether an application service addition or a change is required. If it is an addition, the inquiry and order processing process will be passed through. If the addition is a service that is isolated from an existing one and it is a mass-customizable BTO service it will be treated as a BTO scenario (cf. Figure 5.16). In other cases, it is treated as an ETO scenario. This applies, for example, if a new application service has to be integrated with an existing service. If customer application service integration has not been planned,

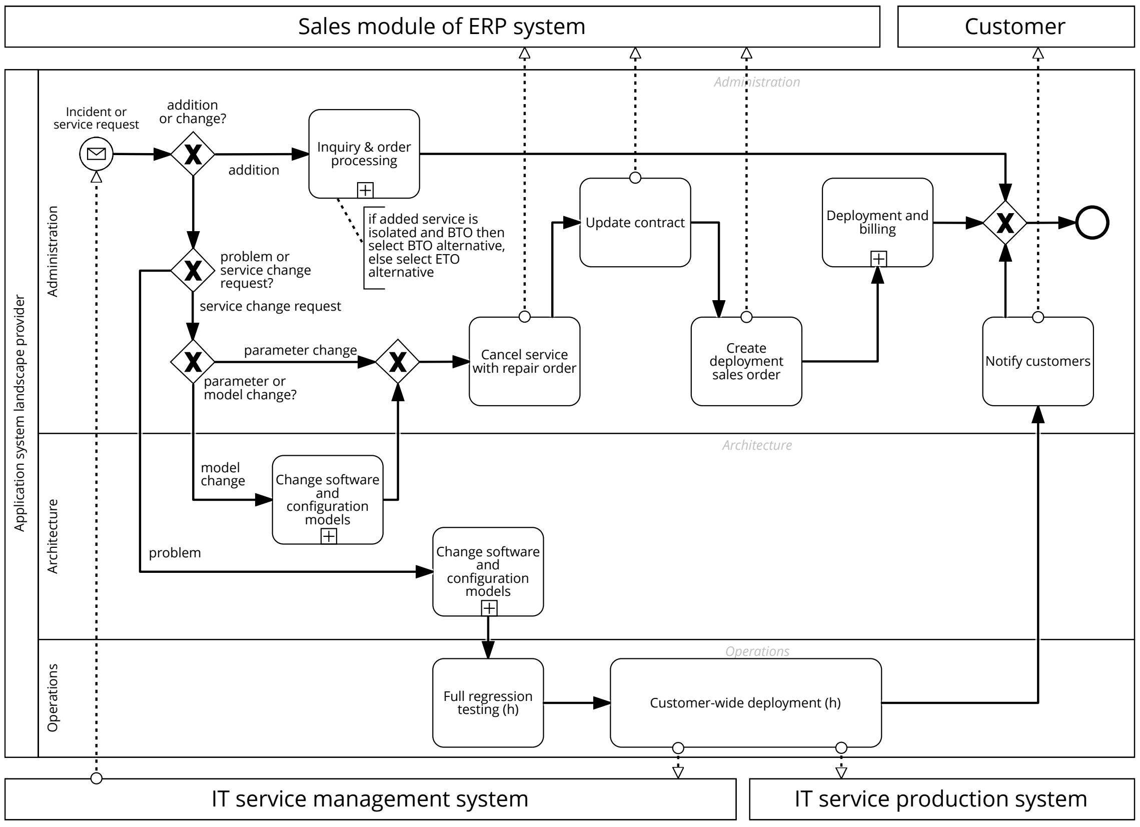

Sales module of ERP system
Customer
Application system landscape provider
Administration
Architecture
Operations
Incident or service request
addition or change?
addition
Inquiry & order processing
if added service is isolated and BTO then select BTO alternative, else select ETO alternative
problem or service change request?
service change request
parameter change
parameter or model change?
model change
problem
Cancel service with repair order
Update contract
Create deployment sales order
Deployment and billing
Notify customers
Change software and configuration models
Change software and configuration models
Full regression testing (h)
Customer-wide deployment (h)
IT service management system
IT service production system

Figure 5.20: Operation phase.

customization and development of the corresponding configuration models and potentially also a modification of software programs will usually be required.

If it is no addition, a change has to be made to an existing application service. Such a change can be necessary because the user made an authorized service change request or because a problem was identified that requires a fix. If a service change request is made two further alternatives exist. The first alternative is that a parameter in a BTO scenario is changed. In this case, the existing service is canceled with a return order. The contract is then updated with the same application service, but newly parametrized. Then a deployment sales order is created with the old production order ID which will be forwarded to the new production order for deployment and billing in the next business process.

If no parameter is changed, an actual model change is required, and the business process *change software and configuration models* is run through (Figure 5.21). The next steps are the same as those of the parameter change alternative. Managing the frequency of such requests is essential because this may lead to a scattered service catalog since such changed services have to be kept separate from their originals as those may still be in operation elsewhere.

If a problem was identified in one of the configuration models, in software programs or a combination of them, again, that software program and corresponding software models need to be changed. As this may not be related to only one application service in production, a full regression testing across all deployed and affected application services and their parametrizations is necessary. After that, the changed software and configuration models are deployed customer-wide in the IT service production system. Customer-wide deployment is not done via production orders as this is not related to individual contracts. Changes to the landscapes also need to be recorded in the ITSM system's CMDB. After the deployment, customers have to be notified.

Figure 5.21 shows the subprocess in which software and configuration models are changed. Again, for those activities time spent is recorded. The time is recorded in a project for each material which is managed in the project management module of the ERP system. The subprocess of changing software is not further discussed but can be managed according to one of the software engineering process models that were discussed in section 3.3.1.

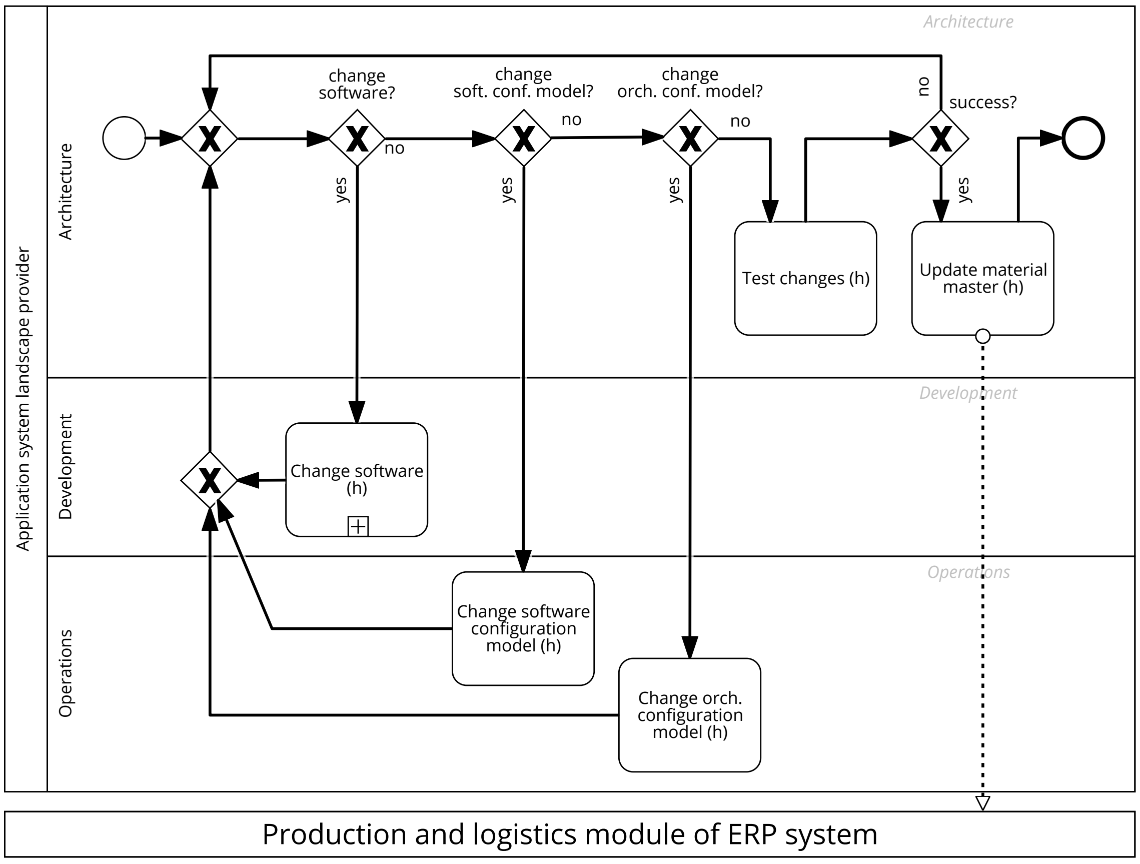

Application system landscape provider
Architecture
Development
Operations
change software?
change soft. conf. model?
change orch. conf. model?
success?
yes
no
Test changes (h)
Update material master (h)
Change software (h)
Change software configuration model (h)
Change orch. configuration model (h)
Production and logistics module of ERP system

Figure 5.21: Change of software and configuration models.

In the subprocess *change software and configuration models*, software is changed if required. Subsequently, software and orchestration configuration models are also changed if required. Testing is performed as the last step and not individually after each change action because changes usually rely on each other. The material master is modified according to the change if the test was successful. Changing the configuration of the infrastructure configuration model in a running service is not possible because this would mean that the affected instances would need to be exchanged. If such a change would be necessary a new application service with the required infrastructure configuration models could be developed in an ETO scenario. Then a data migration could be performed between both landscapes. A data migration, however, would be a manual service. It is also possible to design services in such a way, using, for instance, rolling upgrades (Zhu et al., 2015). This would be the scope of each application system landscape but is not by default in scope in the ISAA.

The next section will discuss how the termination of application services is handled in the ISAA.

5.2.5 Termination

"The company exists to make money." – Goldratt (2004, p. 46)

Just like any company, the goal of an ASLP should usually be to prolong a lucrative contract with a customer. Therefore, when a contract has a termination date, the customer should be contacted by the provider regarding an extension of the contract. If the customer agrees to an extension, the contract can be extended. If no extension is agreed upon the contract will end as originally mandated.

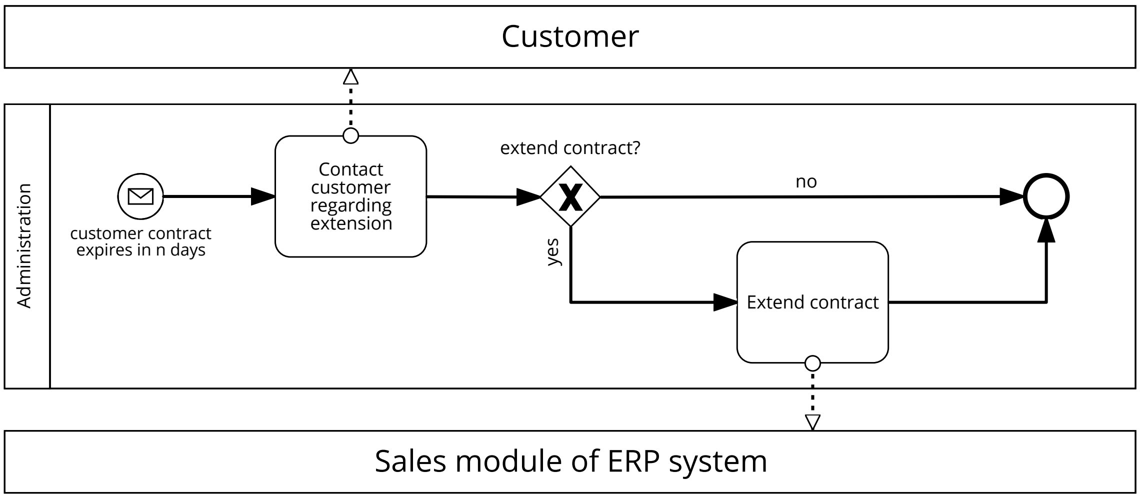

Customer
Administration
customer contract expires in n days
Contact customer regarding extension
extend contract?
no
yes
Extend contract
Sales module of ERP system

Figure 5.22: Contract handling before the termination date.

The technical termination of the application service and depletion of all related resources is performed automatically by the production execution system. This will be outlined in section 5.4.4.

In the next section, the application landscape system of the ISAA will be described in detail. The description builds upon the presentation of the domain model in section 5.1.3.

5.3 Application system landscape

As discussed in the preliminary investigations, the primary application systems required for producing application service are enterprise management, IT service management, and

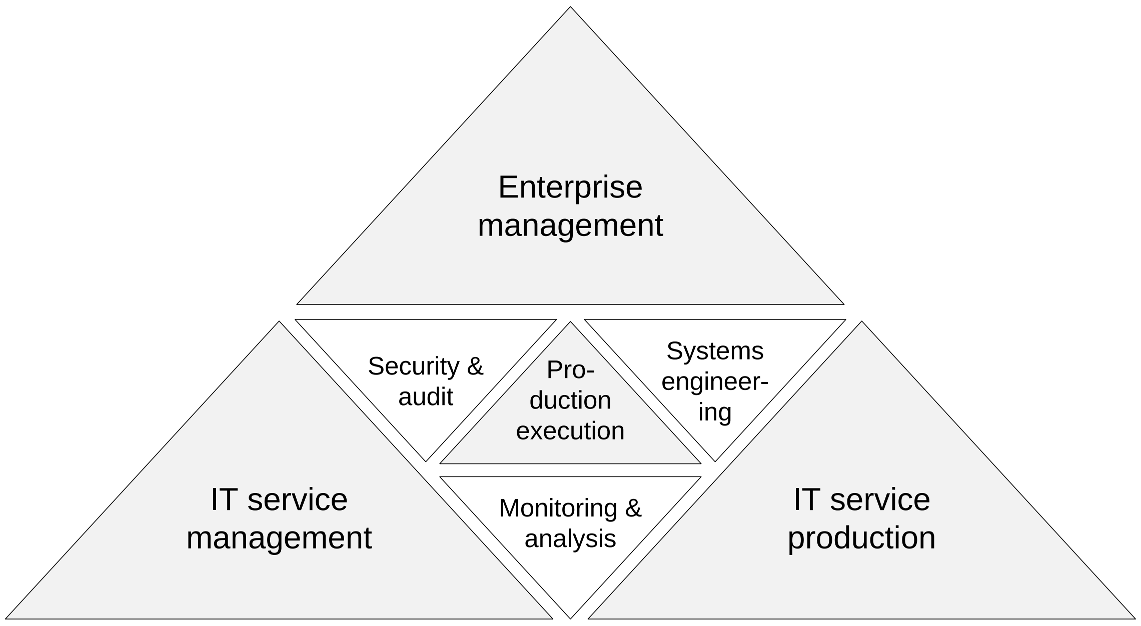


Figure 5.23: Application system categories in ASLP's landscapes.

IT service production systems. According to Hintsch et al. (2017) three further application system categories exist. These may support the production indirectly. The three categories originate from the application fields of security and compliance, systems engineering, and of monitoring and analysis. Security and audit software "[keeps] the impact and occurrence of information security incidents within the [acceptable] levels" (Lainhart et al., 2012, p. 113). They also facilitate internal and external security auditing. Systems engineering software supports activities ranging from requirements engineering, modeling and design, to construction and testing (Mobus and Kalton, 2015). Finally, monitoring and analysis software records log and performance data about infrastructure, platforms, as well as applications. Furthermore, means for analyzing the data to evaluate functional as well as non-functional properties of the monitored IT systems are provided (Hintsch et al., 2017). Figure 5.23 shows all system categories in the application system landscape of an ASLP.

Security and monitoring measures, implemented in application systems, are essential to avoid unauthorized or hazardous activities and to react to outages quickly or to investigate systems. Depending on the nature of the business, different emphasis may be put on security and monitoring.

An important step towards useful monitoring can already be made by collecting log files centrally. Then, alerts are triggered if error messages of remote hosts start to occur. Countermeasures can subsequently be taken. Central collection of log files is an area to which configuration management software often is applied[16]. Implementations of the ISAA could add configuration models for monitoring to the service BOM by default. These monitoring models would command each customer instance to periodically report its health status and that of its software stack to a central server. Various strategies can

[16]Syslog is the central logging system in the Linux OS. On the Puppet configuration repository 180 modules were found dealing with Syslog on 25.02.2018 (https://forge.puppet.com/modules?q=syslog).

be followed (Hernantes et al., 2015; Spring, 2011a,b).

Security and analysis tools may have no permanent integration with the rest of the application systems. An example is a role-based access rights analysis. Here, low-level access rights scattered across various systems can be extracted into file dumps and analyzed in specific software. Permanent integration could be advantageous for some customers but is not required in all circumstances (Fuchs and Pernul, 2013). Security and monitoring systems are included, but will not further be discussed.

Figure 5.24 shows the systems of the application system landscape in a UML component diagram. Offering and usage of interfaces are depicted. Almost all components are executed on centrally managed hosts. One component, the integrated development environment (IDE), is executed on the computers of the engineers who are responsible for creating and maintaining the configuration models. An IDE belongs to the systems engineering category. The IDE illustrates how new configuration models may be loaded into the version control system via its write interface. From the version control system, the configuration models are served to the other systems.

In addition to the centrally managed hosts that form the core of the provider's ASL and the IDE, three further components are included in the diagram. They represent the critical components in a customer instance to deploy and change the customers' application services. The configuration management agent uses the operating system's API to configure the software on the instance. The package management system is used to gather packages and install them if new software needs to be installed. All instances are handled by an IaaS node controller of the physical host on which they are executed.

Software packages are loaded from a definitive media library (Rance, 2011, p. 99). The definitive media library is a repository that stores different versions of software programs (e.g., beta and stable). Software marked as stable by developers via flags in the version control system may be sent to the definitive media library by the continuous integration system if all automated tests have been successfully passed. The continuous integration system also interfaces with the configuration management and IaaS system to deploy landscapes into test environments.

The ERP system contains modules for project management, financing and accounting, production and logistics, sales, and human resources, as well for customization following Gronau's description (Gronau, 2010, p. 9). A project management module extends Gronau's description. The ERP system is the leading system to support the production process. It communicates with the production execution system. This is necessary for deploying the application services according to customer orders and also for initiating requested changes. The PES and the ERP system communicate via their APIs.

The IT service management system interfaces with the ERP system. It may report ticketing data back to the ERP system for billing and cost accounting purposes (cf. REQ P.5). For example, user help desk activities, as supported by the ITSM system, may be billed to the customer separately or be considered for internal cost accounting. Also, an integration or synchronization of data on customers, the service catalog (material master),

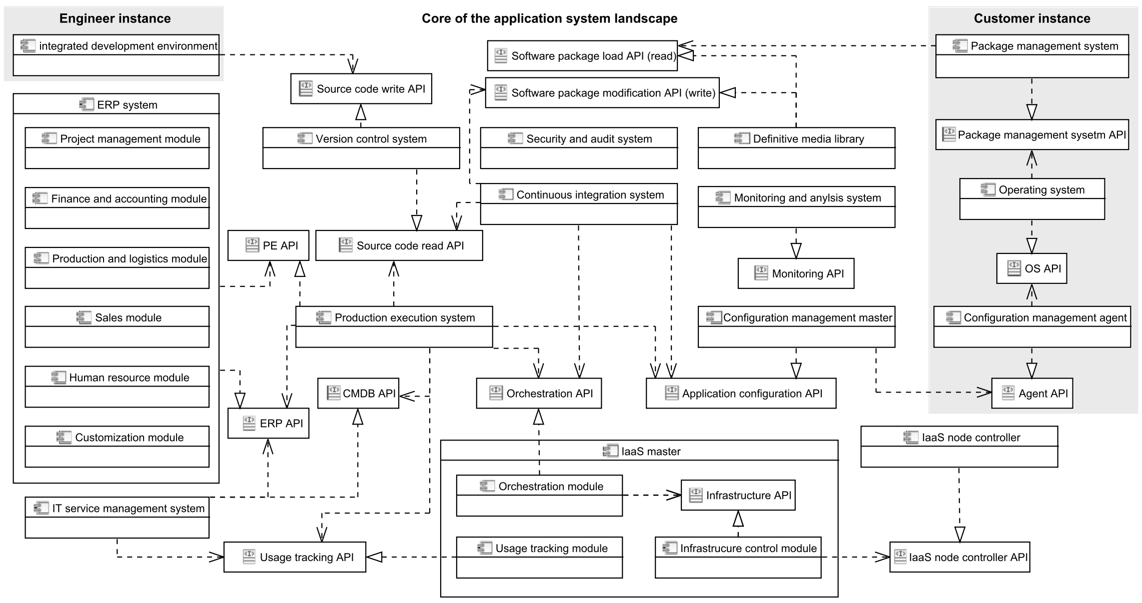

Engineer instance
integrated development environment
Core of the application system landscape
Customer instance
Package management system
Software package load API (read)
Source code write API
Software package modification API (write)
ERP system
Project management module
Version control systems
Security and audit system
Definitive media library
Package management sysetm API
Finance and accounting module
Continuous integration system
Monitoring and anylsis system
Operating system
Production and logistics module
PE API
Source code read API
Monitoring API
OS API
Sales module
Production execution system
Configuration management master
Configuration management agent
Human resource module
CMDB API
Orchestration API
Application configuration API
Agent API
ERP API
Customization module
IaaS master
IaaS node controller
Orchestration module
Infrastructure API
IT service management system
Usage tracking API
Usage tracking module
Infrastrucure control module
IaaS node controller API

Figure 5.24: Application system landscape depicted in a UML component diagram.

and current contracts is advisable to facilitate efficient processes and consistent data (cf. architecture decision 3 and section 3.2.2).

The IT service management system communicates directly with the IaaS master host to retrieve information about the current IaaS node resource utilization for capacity management. Capacity planning is performed outside the ERP system because the IaaS master should make rapid, automatic adjustments for fluctuating demand (Mell and Grance, 2011). For on-premise hosting, spare capacities need to be maintained. Capacity planning can be based on sales data from the ERP. Planning runs, however, are not performed every time an application service is deployed. The costs for IaaS capacity can be distributed across the customer base since IaaS capacity is highly standardized. Figure 5.12 shows the four types of IaaS resources. IaaS capacity planning can be decoupled from the individual sales orders and production orders. The provider may have teams responsible for infrastructure and applications, respectively. Therefore, as stated earlier, the focus of the ISAA is on the application software on the software layer which is reasonable because the infrastructure is increasingly becoming a commodity (Owens, 2010).

Reacting to changes in demand depends upon a company's capability of monitoring service usage, scaling the application system landscape effectively, and commanding sufficient IaaS capacity. Therefore, the IaaS components of the IT service production system can be maintained as an internal supplier with an infinite material stock. This internal supplier is then only billed after the resources have been used. Subsequently, the infrastructure department or an external provider can plan capacity. Historical monitoring data may serve as the basis for planning (Gmach et al., 2007). The following architecture decision reflects the treatment of infrastructure resources.

Architecture decision 15
Infrastructure resources (CPU, memory, storage, and bandwidth hours) shall be treated as infinite. IaaS services are either operated in-house or sourced.

The IT service management system also uses the usage tracking API of the IaaS master to retrieve utilization data. This data is integrated into the capacity management process of the ITSM system. When new capacity is required in the form of new hardware in an on-premise hosting scenario, this new hardware would be manually ordered from a vendor. Hardware is ordered in bulk to be able to negotiate discounts. The ERP system's procurement functionality would be used for this purchase. This is the way Alpha manages capacity. The monitoring and analysis system provides a monitoring API. This API is used by all systems in the landscape to provide monitoring information centrally. As discussed above, each customer's instance can be induced by default configuration models to send its logging information. Similarly, the input data for security and audit data is not explicitly shown. This may come from the CMDB (Desai et al., 2006) or from directory information systems that manage access authorization (Fuchs et al., 2011; Fuchs and Pernul, 2013). : The production execution system interfaces with most other systems. It will be discussed in the next section.

5.4 Production execution system

The production execution system mainly acts between the ERP system and the other systems to orchestrate service production. Its system activity *service deployment* will be described section 5.4.1, which is next. When a change to a landscape becomes necessary, the PES directs this *service change* as presented in section 5.4.2. The system activity *service billing* is described in section 5.4.3. If a service's contract expires and is not extended as discussed in section 5.2.5, the service will be retired, and the resources dismantled. The system activity *service termination* is described in section 5.4.4. The production execution system needs to maintain data particularly for billing and for changing a service. The corresponding *data model* is discussed in section 5.4.5.

5.4.1 System activity: Service deployment

Figure 5.25 illustrates the role of the PES when automatically deploying a new service. In the first step, the PES receives the bill of materials and the production order identifier from the ERP system's production and logistics module. Subsequently, a unique identifier for this service is generated. This is necessary to be able to associate different production orders with the service in case it is changed in the course of its lifecycle. In the third step, the version control system is queried for the configuration models contained in the BOM, and they are retrieved. Then, in the fourth step, the PES posts the service's unique identifier, the retrieved orchestration configuration model, and potentially several infrastructure configuration models to the IaaS master. The unique service identifier is also used by the IaaS master to identify the service throughout its lifecycle. The IaaS master then spawns the instances as specified by the infrastructure configuration models and sets up the network as defined in the orchestration configuration model. The PES gets back a mapping in which different roles such as *webserver* or *databaseserver* are mapped to the actual instance addresses that were provided by the IaaS system. This information is required for configuring the software of the application service. The mapping between roles and software configuration models is maintained in the orchestration configuration models.

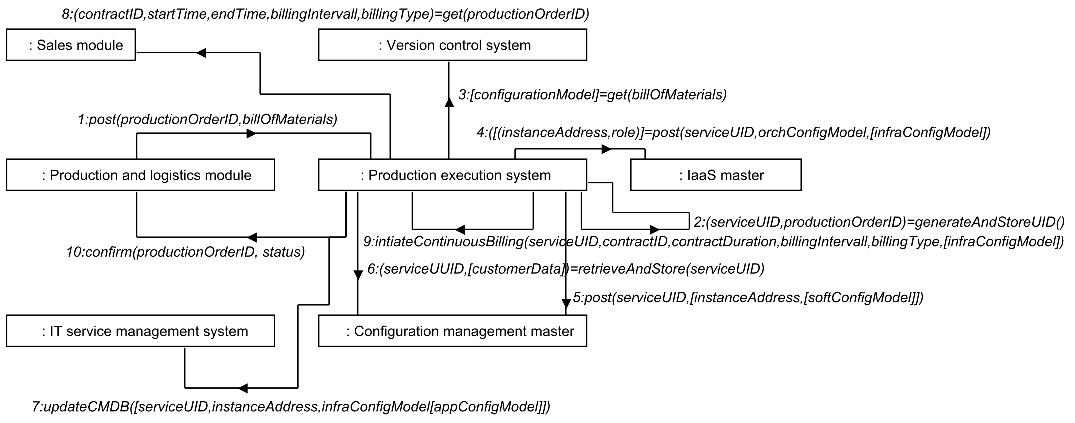

8:(contractID,startTime,endTime,billingIntervall,billingType)=get(productionOrderID)
: Sales module
: Version control system
3:[configurationModel]=get(billOfMaterials)
1:post(productionOrderID,billOfMaterials)
4:([(instanceAddress,role)]=post(serviceUID,orchConfigModel,[infraConfigModel])
: Production and logistics module
: Production execution system
: IaaS master
2:(serviceUID,productionOrderID)=generateAndStoreUID()
9:intiateContinuousBilling(serviceUID,contractID,contractDuration,billingIntervall,billingType,[infraConfigModel])
10:confirm(productionOrderID, status)
6:(serviceUUID,[customerData])=retrieveAndStore(serviceUID)
5:post(serviceUID,[instanceAddress,[softConfigModel]])
: IT service management system
: Configuration management master
7:updateCMDB([serviceUID,instanceAddress,infraConfigModel[appConfigModel]])

Figure 5.25: PES's system activity *service deployment*.

In the fifth step, the configuration management master receives the unique service identifier, the instances' addresses, and the software configuration models that are used to configure the application service's software on each instance. Figure 5.26 describes in more detail the steps that are taken by the configuration master and PES to configure each instance.

In the sixth step, the user data will be retrieved. This data is stored in the database of the PES where it can be accessed when the user is notified of the ensued service provisioning. This data can, for instance, be shared via e-mail or a separate authorized web service.

The seventh step is to update the CMDB of the IT service management system about the newly deployed application service. This includes information about the instances on the infrastructure layer, for example, size and addresses, as well as the information on the software layer, for example, what applications are running on the instances.

In the eighth step, the contract, the contract identifier, as well as start and end time, the interval length in which billing shall be conducted (e.g., monthly or yearly), and the billing type (fixed billing or usage-based billing) are retrieved from the ERP system. This data is then used to initiate the continuous billing for the application service. During the continuous operation of the application service infrastructure costs are incurred. Therefore, the infrastructure configuration models are required to determine, for fixed billing, how many infrastructure hours need to be billed in a given billing interval.

The final step of the deployment process is the confirmation of the production order that the PES sends to the ERP system.

The configuration master authenticates each instance based on the address received from the PES. Then the software configuration is posted to the configuration management agent. The mapping of the instance and software configuration is performed based on the role definition in the orchestration configuration model. Figure 5.26 displays components that exist in each customer's instance. In the third step, the configuration management agent will use the operating system's API to prompt the package management system to install the software packages of the application service's application software as defined by the software configuration models.

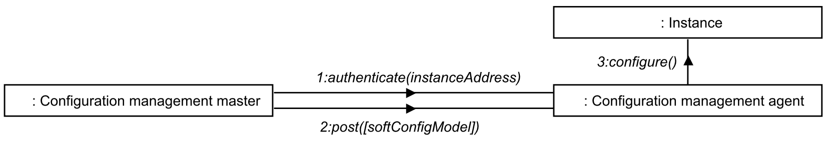

: Instance
3:configure()
1:authenticate(instanceAddress)
: Configuration management master
: Configuration management agent
2:post([softConfigModel])

Figure 5.26: Configuration of software on each instance.

In the next section, the change of a service will be described.

5.4.2 System activity: Service change

The system activity *service change* is depicted in Figure 5.27.

The ERP system starts the activity by posting the production order identifier, the bill of materials, and the old production order identifier to the PES. The next step is to update the connection between the service's unique identifier and old production order identifier with the new production order identifier.

In the third step, the new configuration models are retrieved from the version control system. The fourth step sends the configuration models to the IaaS master in the same way as it was done in the deployment step. Here, the service is updated as defined by the orchestration model. For instance, the network structure might be changed, new instances started, or new policies are configured such as those for scaling.

In the fifth step, the configuration master is updated with the new configuration information. This includes potentially new instances that were added and needed to be configured. However, also instances that were already deployed in the old production order might be updated in their software configuration.

In the sixth step, the CMDB is updated with an entry for each instance including again the service's unique identifier, instance address, infrastructure configuration model, and software configuration models. In the seventh step, the change is confirmed.

The next section will describe the system activity called service billing.

5.4.3 System activity: Service billing

Figure 5.28 illustrates how the PES initiates billing based on a fixed amount of IaaS resources. As described in section 5.2.1, for example, an instance with one CPU would

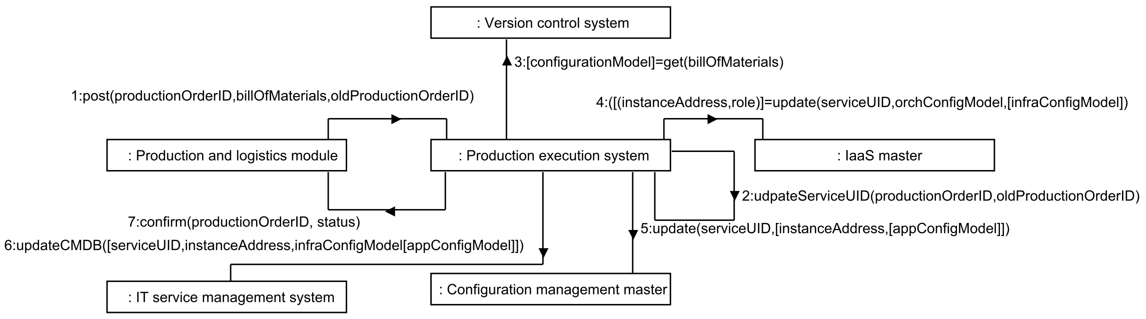

: Version control system
3:[configurationModel]=get(billOfMaterials)
1:post(productionOrderID,billOfMaterials,oldProductionOrderID)
4:([(instanceAddress,role)]=update(serviceUID,orchConfigModel,[infraConfigModel])
: Production and logistics module
: Production execution system
: IaaS master
2:udpateServiceUID(productionOrderID,oldProductionOrderID)
7:confirm(productionOrderID, status)
5:update(serviceUID,[instanceAddress,[appConfigModel]])
6:updateCMDB([serviceUID,instanceAddress,infraConfigModel[appConfigModel]])
: IT service management system
: Configuration management master

Figure 5.27: PES's system activity *service change.*

consume a fixed amount of 720 CPU hours per month[10]. In the first step, the consumption is calculated using the service's unique identifier. Consumption can be represented as a quadruple for all four billable IaaS resources. For the calculation, the billing interval and the instances' infrastructure configuration models are used. The consumption is then used to create a sales order on the respective service's contract. A sales order identifier is received back from the sales module of the ERP system. In the third step, the sales module is used again, to create and send out an invoice based on the acquired sales order identifier.

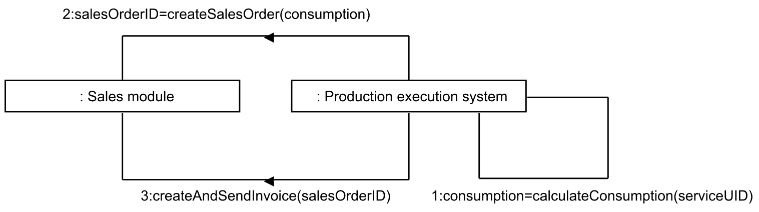

2:salesOrderID=createSalesOrder(consumption)
: Sales module
: Production execution system
3:createAndSendInvoice(salesOrderID)
1:consumption=calculateConsumption(serviceUID)

Figure 5.28: PES's system activity *service billing (fixed).*

Figure 5.29 illustrates how the PES initiates billing based on variable IaaS resource consumption. In the first step, the consumption is gathered from the IaaS master. For this, the start and end time for the necessary consumption calculation as well as the service's unique identifier are sent to the IaaS master. The second and third equal those of the fixed billing system activity.

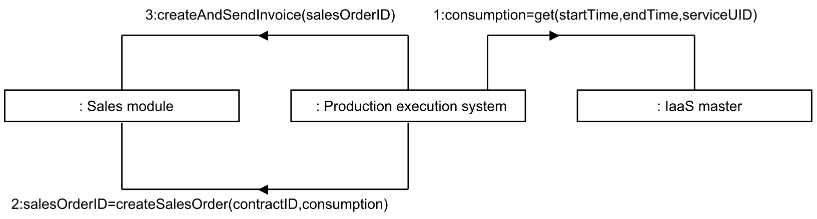

3:createAndSendInvoice(salesOrderID)
1:consumption=get(startTime,endTime,serviceUID)
: Sales module
: Production execution system
: IaaS master
2:salesOrderID=createSalesOrder(contractID,consumption)

Figure 5.29: PES's system activity *service billing (based on usage).*

The next section will describe how a service is terminated.

5.4.4 System activity: Service termination

Figure 5.30 shows the system activity to terminate a service and the dismantling of its resources. In the first step, the instances are backed up. These can be used to restore the service. They can also be used to provide the customer with the data that was produced over the run-time of the service. The second step is to terminate the instances of that service using its unique identifier. The third step is to delete the service on the configuration management master. In the fourth step, the service is deleted from the PES, and in the fifth step, the service is marked as retired in the CMDB. Finally, the service's contract is reported to be terminated.

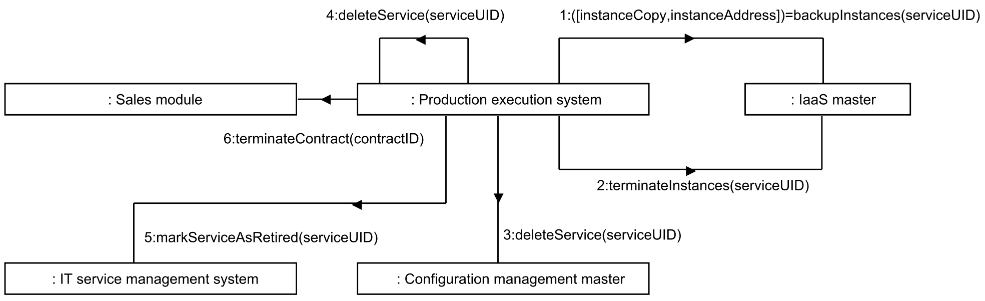

4:deleteService(serviceUID)
1:([instanceCopy,instanceAddress])=backupInstances(serviceUID)
: Sales module
: Production execution system
: IaaS master
6:terminateContract(contractID)
2:terminateInstances(serviceUID)
5:markServiceAsRetired(serviceUID)
3:deleteService(serviceUID)
: IT service management system
: Configuration management master

Figure 5.30: Service termination.

The next section will describe the data model of the PES.

5.4.5 Data model

Figure 5.31 shows UML class diagram. It represents the minimal set of information that the PES needs to handle. It contains four classes: *ApplicationService*, *Instance*, *CustomerData*, and *Contract*.

For the users of the customer to be able to access the application services, they, for example, need initial credentials. Also, specific addresses of the access point may be required. To offer a wide variety of information to be stored, the *CustomerData* class has the fields *key* and *value*. Depending on the service specificities, different keys (e.g., password, username, or URL[17]) and corresponding generated values instantiate these fields in the database. Access to this data may be given to the customer when the first invoice is created and sent out (cf. section 5.2.1).

For fixed billing, the *Instance* class has the fields *instanceAddress* and *infraConfigModel*. Based on this data, the consumption for fixed billing can be calculated based on the *infraConfigModel* that is managed by the IaaS master.

The class *ApplicationService* itself contains, amongst the links to the other classes, the fields *serviceUID* and *productionOrderID* that are used for managing the application service across different production orders that may occur when the service is changed. The contract class includes the fields *contractID*, *startTime*, *endTime*, *billingInterval*, and *billingType*. All of this information are required for sales order and invoice creation.

[17]Uniform resource locator (URL)

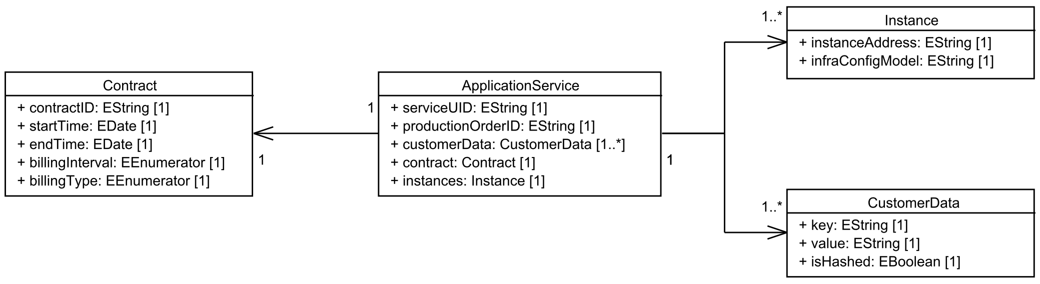

Contract
+ contractID: EString [1]
+ startTime: EDate [1]
+ endTime: EDate [1]
+ billingInterval: EEnumerator [1]
+ billingType: EEnumerator [1]
1
1
ApplicationService
+ serviceUID: EString [1]
+ productionOrderID: EString [1]
+ customerData: CustomerData [1..*]
+ contract: Contract [1]
+ instances: Instance [1]
1
1..*
Instance
+ instanceAddress: EString [1]
+ infraConfigModel: EString [1]
1..*
CustomerData
+ key: EString [1]
+ value: EString [1]
+ isHashed: EBoolean [1]

Figure 5.31: Model of the data handled by the PES.

The evaluation of this thesis will be presented in the next chapter.

6 Evaluation

Final validity that design science artifacts achieve utility for humans or organizations can only be achieved in naturalistic settings (Sonnenberg and vom Brocke, 2012; Hevner et al., 2004). DSR produces prescriptive knowledge, and until this knowledge is applied in practice, it is often assumed to have no truth-like value (Sonnenberg and vom Brocke, 2012). Artificial environments are used to evaluate this knowledge in its design stage. It is difficult to craft artificial environments to resemble the real world exactly. Analytical, experimental, testing, and descriptive evaluation methods (Hevner et al., 2004) all are subject to this difficulty. Only observational evaluation methods including case and field studies directly address the goal of final validity. Such studies must be carefully designed, may be expensive, and their cases must be carefully sampled (Eisenhardt, 1989). Therefore, proving that a DSR artifact achieves utility (validation) can be challenging.

Consider the example of enterprise application systems that automatically adapt to the stress levels of users: Adam et al. (2017) propose an approach of implementing such systems to overcome a range of challenges ranging from technical feasibility to ethical acceptability. They base their design on theoretical foundations and utilize results of interviews and focus group discussions. However, they are not able to finally validate their research. Therefore, they propose to use laboratory studies first and then later validate their results with field studies. Because such studies are expensive, they are not performed from the beginning. Their article communicates unproven prescriptive knowledge.

Several works from the three sets of related work also lack proven validity (cf. section 3.5.2). Ebert (2009) uses a prototypical implementation and scenarios to verify his concept of transferring production planning and control to IT service providers. This approach is also chosen by other authors of important related work (Wettinger et al., 2013; Kirschnick et al., 2010; Mastelic et al., 2016).

For the artifact at hand, the ISAA, a case or field study evaluation would require implementing the whole ISAA or parts of it with an ASLP. This would require a company to be willing of implementing such a novel concept. Large and complex projects (Michel, 1998; Finney and Corbett, 2007) are required to implement ERP systems in companies. Such a project would be beyond the scope of a thesis. There is no instance known to the author where such an evaluation method was utilized to prove validity, considering relevant related work (Ebert, 2009; Wettinger et al., 2013; Kirschnick et al., 2010; Mastelic et al., 2016).

Nonetheless, the bespoken authors have identified research gaps and have built their artifacts methodologically sound. Sonnenberg and vom Brocke (2012) argue that artifacts can have truth-like value in earlier design stages even if their validity is not finally proven, for instance, if preliminary knowledge is derived from appropriate and validated theories. To assist researchers to communicate their research in early design stages successfully, Sonnenberg and vom Brocke (2012) have suggested a step-wise approach to evaluate DSR artifacts. Table 6.1 displays these four evaluation steps with their input and output.

Step	Input	Output
EVAL 1	Problem statement and/or observation of a problem, research need, design objectives, design theory, existing solution to a practical problem	Justified problem statement, justified research gap, justified design objectives
EVAL 2	Design specification, design objectives, stakeholders of the design specification, design tool and/or design methodology	Validated design specification, justified design tool and/or methodology
EVAL 3	Instance of an artifact (prototype)	Validated artifact instance in an artificial setting (proof of applicability)
EVAL 4	Instance of an artifact	Validated artifact instance a naturalistic setting (proof of usefulness)

Table 6.1: Four steps of DSR evaluation, adapted from Sonnenberg and vom Brocke (2012).

In the first step, the relevance (Hevner et al., 2004) of the research project are evaluated. Research problems are worked on either because previous literature did not address them at all or because it did not address them sufficiently. Insufficient solutions can take the form of existing DSR artifacts, or entirely new problems come from practice. Thus, the research gap needs to be identified and justified. To design a solution to a problem, objectives need to be defined that can be used to decide whether or not the solution sufficiently solves a problem. Therefore, justified design objectives are the third output of the first step.

Before the second evaluation step, a design specification has been created. Such a specification has stakeholders, and a methodology or tool or a compilation of the two has been devised to create the specification. The goal of this step is to validate the design specification and justify the selection of design tool and methodology. Different criteria such as simplicity, clarity, completeness, or level of detail are used in this evaluation step. Methodologies for evaluation include logical reasoning, simulation, or benchmarking.

An instance of the artifact, a so-called prototype, exists when the third step of DSR evaluation begins. The goal is to validate the artifact instance in an artificial setting. Here, the applicability of the artifact to solve a specific problem is proven. Suitable evaluation criteria in this step are, for example, feasibility, efficiency, or effectiveness. Evaluation methodologies in this step are often centered around the prototype, including a demonstration or experiment with the prototype, but also other approaches such as focus group discussions may be applicable.

In the fourth and final evaluation step, an instantiation of the artifact is validated in a naturalistic setting. Here, the usefulness is proven. Different criteria such as fidelity with real-world phenomena or impact on the environment and users of the artifact as well as others can be used for evaluation. Methods in this step include, as discussed before, case studies or surveys.

Table 6.2 lists which methods and criteria were used for evaluation of the ISAA.

Step	Evaluation methods	Evaluation criteria
EVAL 1	Study of the knowledge base and environment	Relevance
EVAL 2	Logical reasoning, demonstration	Consistency, applicability, and adaptability
EVAL 3	Scenario-based descriptive evaluation, functional (black box) testing, expert evaluation	Fidelity with real world phenomena, feasibility, utility

Table 6.2: Steps for evaluating the ISAA.

To conduct the first evaluation step, primarily, the knowledge base and environment were studied (Hevner et al., 2004). Section 6.1 will outline that the research addressed a relevant research problem.

In the second step, the architecture description of the ISAA is evaluated. It is preliminarily shown that the artifact has the capability of reaching the research goal and was soundly constructed. For this, logical reasoning and demonstration were employed as methods. The architecture description was evaluated based on the criteria of consistency, applicability and adaptability (Prat et al., 2015). The architecture consistency is elaborated on based on the stringent derivation of architecture decisions from relevant rationale. Applicability is demonstrated by instantiating the architecture description. Adaptability is shown by demonstrating that the artifact can be used in scenarios that were not in the central research scope. This is important to ensure broad applicability.

For the third step, the ISAA, based on the architecture description, was instantiated into a prototype. The prototype verifies that the ISAA can technically be implemented. As evaluation methods, scenario-based descriptive evaluation, functional (black box) testing, and expert interviews were used. Employed criteria are the architecture's fidelity to real-world phenomena, its feasibility of instantiating, and its utility. Mathematical proofing is not conducted. Formalizations required for proofs would be infeasible (Hanappi et al., 2016) because the ISAA consists of various components. Even segments of the architecture would constitute a large number of variables. Therefore, such formal evaluation methods are not implemented in favor of descriptive and testing methods. It is primarily shown that the ISAA can be instantiated. Because so many components on different layers are involved in the ISAA, the assembly of the prototype was not trivial. The architecture's fidelity to real-world phenomena is shown by testing it in selected real-world-ajar scenarios. Furthermore, feasibility tests regarding automation were conducted. Lastly, the ISAA was evaluated based on expert interviews. As previously argued, an observational evaluation method such as a case or field study (Hevner et al., 2004) would be challenging to implement. This would require implementation in a real setting. Implementation of the ISAA in a real organization would exceed the scope of the thesis. Therefore, the expert interview methodology is selected to validate the ISAA and results from the previous evaluation steps. Experts were selected with a broad educational background to validate the ISAA regarding its overall feasibility and utility.

6.1 Evaluation of relevance (EVAL 1)

Hevner et al. (2004) provides guidelines for conducting DSR: (1) Design as an artifact, (2) problem relevance, (3) design evaluation, (4) research contributions, (5) research rigor, (6) design as a search process, and (7) communication of research. Guideline 2 and 5, addressing relevance and rigor, have been the object of some debate (Turner et al., 1991). Particularly behavioral information systems research, where quantitative methods (e.g., statistics) and qualitative methods (e.g., single-case study) are used (Eisenhardt and Graebner, 2007) and sometimes are seen competitively, have discussed the balance between rigor and problem relevance. Advocates of quantitative research methods (positivists) may discard results of qualitative research methods (non-positivists) as unsound. Similarly, as Sonnenberg and vom Brocke (2012) state, DSR often has a problem to state its truth-like value. A balance between rigor and relevance is important. If rigor prevails, design scientists are in danger of losing their raison d'être because artifacts for problems without practical relevance are designed. If rigor is discarded, a researcher's contribution may be limited and unsound. Although DSR is described as a paradigm, it can also be seen as a research process itself (Peffers et al., 2008). Adhering to the process by Peffers et al. (2008) (identify problem & motivate, define objectives of a solution, design & development, demonstration & evaluation, and communication) exhibits some rigor. It aligns the guidelines differently. DSR as a research process is a meta-level above actual research methods such as prototyping or case study (Wilde and Hess, 2007). Such concrete information systems research methods should also be applied so that a DSR project exhibits rigor.

That efficient IT service provisioning is a relevant problem was addressed in chapter 1 (Guideline 2). The research gap was highlighted and outlined in chapter 3 (Guideline 4). In section 4.2, objectives were derived from the environment and the knowledge base. Literature reviews (Hintsch, 2013; Hintsch and Turowski, 2013; Hintsch et al., 2016a) and case studies (Hintsch et al., 2015b, 2016b) were conducted in order to produce data and insights to develop these objectives (Guideline 5). The objectives guided the construction of the artifact. Consequently, it is concluded that the problem statement, the research gap, and the objectives are justified. They were derived methodologically and with reason.

In the next section, the consistency of the architecture construction will be discussed based on the architecture decisions and their rationale.

6.2 Validation of consistent architecture decisions (EVAL 2)

Consistency is important to avoid contradictions (Prat et al., 2015). If architecture descriptions are inconsistent and contradictory, choosing implementation alternatives may be ambiguous, and the architecture's instantiation will likely be ineffective or inefficient. Methodologically, the architecture construction is aligned with standard 42000 by ISO/IEC/IEEE (2011). "Architecture rationale records explanation, justification or reasoning about architecture decisions that have been made" (ISO/IEC/IEEE, 2011, p. 7). The architecture decisions were developed throughout chapters 4 and 5, the preliminary investigation and the presentation of the ISAA. The architecture descriptions are put together in Table 6.4. Their rationales are summarized in Table 6.5.

The architecture decisions made, have affected elements of the architecture description such as models on all layers of the architecture. Therefore, Table 6.3 provides an overview of the affected parts. The mapping of decisions to parts is made on the level of architecture layers. The ASLP business model (1), application service production process (2), application system landscape (3), and production execution system (4) are aligned with the following layers respectively (cf. Figure 5.3): business (1), process (2), integration (3), software (4), and infrastructure (5). Virtualization (5) is not a contribution to the architecture, but its layer (infrastructure) certainly is affected by the architecture decisions.

ID	Domain model	ASLP business model	Application service production process	Application system landscape	Production execution system	Virtualization
1	x		x	x		
2	x		x	x		
3	x			x		
4	x		x	x	x	x
5	x			x		x
6	x	x	x			
7	x					
8	x		x	x		
9	x			x	x	
10		x	x			
11	x		x			
12			x			
13			x			
14			x		x	
15			x			

Table 6.3: Parts of the ISAA that are affected by the architecture decisions.

Various architecture decisions affect the domain model as it cross-cuts all layers. Only those entities shown in the models in this thesis are considered here.

ADs 1, 2, and 8 affect the production process because they introduce different systems that are used in the process. These ADs affect the ASL regarding what systems are in the landscape. The selection of a standard ERP for manufacturers (AD 8) affects those two architecture parts because, for example, a production order is handled in the production process. The ERP system's module selection was also affected by AD 8.

ADs 3 and 5 affect the application system landscape. Although the ITSM and ITSP systems are used in the production process, these decisions do not affect the process. AD 5 is the basis for using virtualization within the ISAA. Therefore, virtualization has to be taken into account here.

AD 4 has consequences for all parts except the business model. It is the basis for application service description and assembly in the ISAA. Thereby it affects the application service production process and the parts that carry out deployment and operation (ASL, PES, and virtualization) of application services.

The focus on ASLPs (cf. AD 6) affects, naturally, the business model and the application service production process. For instance, the focus is put on the provisioning and construction of ASLs rather than on software development.

AD 7 regarding the SLA only affects the domain model. There, the mapping to the master data of the ERP system is discussed. This AD is then manifested in the prototype for evaluation (cf. section 6.5.1).

The decision to introduce the PES (AD 9) affects the application system landscape and the PES itself. The application service production process is not affected because, from the process-perspective, the ERP system is handling all production relevant data. The PES (e.g., depicted in Figure 5.19) is only exchanging data with the ERP system but is not directly involved in the application service production process (cf. section 5.2), for instance through a user interaction activity.

AD 11 is reflected in the domain model and the production process. The contract is set into relation with other entities in the domain model and worked with in the production process.

Finally, ADs 10, 12, 13, 14, and 15 affect the production process. AD 14 also affects the PES because the different billing modalities need to be reflected by the PES. These ADs are not reflected in the domain model as they affect the process directly.

This section has shown how the architecture decisions have affected the different layers of the architecture. Every architecture decision has a rationale. The rationales are displayed in Table 6.5. They transparently state, based on the requirements of section 4.2 and the presented literature, why the architecture decisions were made. The architecture decisions were the primary guidelines for the construction of the ISAA. The described and documented procedure is inline with the provisions made by the ISO/IEC/IEEE (2011). Hence, here it is argued that the architecture decisions were consistently created based on a state-of-the-art approach to then guide the construction of the ISAA.

In the next section, the exemplary application services that are used throughout the rest of the evaluation are presented.

ID	Architecture decision
1	The ISAA's application system landscape shall comprise three major system categories: enterprise management, IT service management, and IT service production.
2	An enterprise management system shall be the leading application system within the ISAA's application system landscape.
3	Ticketing and consumption data shall be fed back to the enterprise management system by the IT service management and IT service production systems. Relevant data such as contractual data or structural application service information can be looked up in or integrated from the ERP system.
4	Operations automation, in the ISAA, shall be based upon model-based configuration management.
5	The ISAA's application system landscape production shall be supported by on-premise or sourced infrastructure as a service.
6	The ISAA shall primarily be designed for application system landscape providers.
7	One or no service level agreements shall exist for each application service. In the ISAA, service level agreements only represent the agreed upon service level that pertains to service quality indicators such as availability or maximum response time.
8	A standard ERP for manufacturers system shall be used as the enterprise management system.
9	A production execution system shall act as an intermediary between the three the ERP system, the IT service management system, and the IT service production system. The production execution system shall be implemented separately from the ERP system.
10	Application services shall be provided in two scenarios. First, if customers have particular requirements that cannot be served based on the existing configuration models, the *engineer-to-order* scenario applies. Secondly, in cases where customers can be served from the existing portfolio, the *build-to-order* scenario applies.
11	The duration of an application service shall be modeled in a contract.
12	The high-level application service production process shall consist out of the following more fine-grained business processes: Inquiry and order processing, engineering, deployment and billing, operation, and termination.
13	The application service production process shall have the roles of administration, architecture, development, and operations.
14	Two styles of application service settlement shall be supported: pay-per-use and fixed price regardless of volume.
15	Infrastructure resources (CPU, memory, storage, and bandwidth hours) shall be treated as infinite. IaaS services are either operated in-house or sourced.

Table 6.4: Architecture decisions.

ID	Rationale
1	ASL should reflect practical realities (REQ R.3). ASLP business model requires three system categories.
2	Maintain full structure and production data of the application services in the system for billing and financial accounting (REQ P.5) as well as KPI-based management (REQ R.2).
3	Ticketing and consumption data originates in ITSM and ITSP systems. Data has to be sent to ERP system for full-service cost calculation (REQ P.5).
4	Model-based configuration management covers relevant layers of IT stack. In contrast to container-based management, it closer adheres to REQ R.3 (integration with existing technology stack).
5	Little differentiation may be achieved with infrastructure management. High degree of automation is necessary (REQ P.1).
6	Narrowing the scope to test Hypotheses 1 and 2.
7	SLA needs to be considered (REQ C.1). Proposed solution reflects indicators used by cases Alpha and Beta.
8	Architecture decision adheres to manufacturing analogy (REQ R.4).
9	Production in ASL needs to be orchestrated and automated (cf. REQ P.1). External component (from ERP system) should be more cost efficient as existing interfaces for manufacturing execution can be used. PES development and ERP system customization can be independent.
10	Depending on specific business model, ETO scenarios might occur (REQ R.1 and S.1). Companies may economically be compelled to serve specific customer requests.
11	System needs to be aware of service duration (REQ C.1). The production order is not used map duration to support standard KPI-based management functionality of ERP system (REQ P.2).
12	Process is derived from standard order-to-cash process (REQ P.4), SCOR reference model (Hochstein and Uebernickel, 2006), and ITIL (de-facto ITSM standard). Core production steps are considered (REQ P.3, P.4, and P.5). Pre-sales activities such as marketing are not included.
13	Reduced set of basic roles and aligned with DevOps (cf. section 3.3.1).
14	Extremes of pricing formula after Iveroth et al. (2013) are supported (REQ P.5).
15	Separation of ASL provisioning from infrastructure (AD 5). Application services just consume infrastructure resources. Separation of concerns of software and infrastructure provisioning.

Table 6.5: Rationales for architecture decisions (ADs are represented by ID).

6.3 Exemplary application services for the evaluation (scenarios)

The ISAA supports the production or application services in two main scenarios: build-to-order and engineer-to-order. In BTO scenarios, on the one hand, customers are served standard application system landscapes that vary only in their parametrization. Application services provided in ETO scenarios, on the other hand, are characterized by a high degree of individuality. For instance, new configuration models that previously had not existed in the provider's service catalog may have to be created. Table 6.6 gives an overview of the three exemplary services.

Service	**Type**	**Description**
Remote desktop	BTO	This service provides the customer's users with the means to access dedicated desktops via an access point that also acts as the web-based client.
SAP ERP	BTO	This service provides the customer with access to a full installation of a pre-configured ERP system from the software vendor SAP.
CMS	ETO	Web applications may have different architectural requirements. With this service, the provider can engineer web applications serving various needs.

Table 6.6: The three application services used throughout the evaluation.

In a remote desktop service offering, the provider operates instances that contain customer-required application software. Users can access the instances with low-cost hardware (e.g., thin-clients) instead of requiring fully-equipped workstations. Access is available from any location with sufficient network bandwidth (Deboosere et al., 2012; Song et al., 2017).

Typically, IT departments or internal IT service providers are responsible for managing the desktop computers of their companies' employees (Dernbecher et al., 2013). Remote desktop services have been utilized long before cloud computing became popular because of centralized management's advantages. For instance, Citrix-based terminal services have been used by companies since the 1990s (Wikipedia, 2003). With a focus on resource sharing in cloud computing, operating remote desktop services for customers that are dispersed geographically can also be appealing. In such a case, work-loads differ based on the working-hours of customers in different geographical locations. Consequently, work-loads can be managed and overall energy consumption and maintenance effort be reduced. Amazon has launched its remote desktop service offering in 2014 (Perez, 2014). Similarly, Gamma plans to introduce such a service in addition to its primary enterprise application services. The selling proposition is to ease the effort customers have in preparing their computers for the training programmes offered by Gamma. Consequently, the remote desktop service may be offered in different configurations regarding its vertical and horizontal scale as well as its application software configuration. A remote desktop service is based on an application system landscape although it is not comparable with enterprise application software systems such as ERP system concerning complexity.

The second BTO service is an ERP hosting service. Larger organizations typically use such application software, but also small and medium-sized ones (Klaus et al., 2000). As it is the backbone of many companies' operations, the ERP system's operation can be demanding concerning maintaining acceptable performance and availability rates. Many internal IT service providers have teams that specialize in the company's ERP system. In the case of SAP, large firms often have established so-called SAP competence centers where ERP knowledge is concentrated among a small group of people (Kræmmergaard and Rose, 2002). Concerning outsourcing, several ASLPs provide ERP-featuring application system landscapes to their customers. Among them are companies like ITtelligence, Accenture, or IBM, but also the three company cases Alpha, Beta, and Gamma of this thesis. In the evaluation, the ERP hosting service is selected to investigate if a more complex application software can be handled within the architecture.

The third service used for evaluation is a content management system, which was used in the examples throughout the previous chapter. Content management systems often are the basis for corporate websites (Grossniklaus and Norrie, 2002). In addition to the content presentation, they feature shopping modules and are often also integrated with other application systems such as web shops or back-office databases (Grossniklaus and Norrie, 2002). It is used to demonstrate how the architecture can support ETO scenarios.

Throughout the next sections, these services will be used as examples. The next section will validate the applicability and adaptability of the architecture description.

6.4 Validation of applicability and adaptability of architecture (EVAL 2)

The ISAA was described in chapter 5. The main description components are the domain model class diagrams, the production process diagrams, as well as the component, communication, and class diagrams describing the application system landscape and production execution system.

All but the domain model class diagrams describe the processes supported by the application systems and the application system landscape of the ISAA. Therefore, the evaluation of the process as well as the application system landscape will be performed based on the prototype. Consequently, this section will be restricted to validate the domain model regarding applicability and robustness. As the domain model is the basis for the whole architecture, its evaluation regarding applicability and robustness should also have validity for the architecture as a whole.

Based on architecture decision 4 model-based configuration management was selected as a basis for construction of the ISAA. This decision makes it questionable if the ISAA could also be used when approaches such as container-based virtualization are used. Container-based virtualization combined with orchestration frameworks is appealing as it promises easy portability and scalability of application services (Pahl, 2015). Additionally, the ASLP business model, as defined in this thesis, is focused on providing full application system landscapes for each customer. Multi-tenancy exists on the infrastructure layer as multiple customers or tenants share the same physical resources such as networking equipment or storage abstracted through virtualization. However, as software as a service is a prominent service model (Anderson et al., 2013), multi-tenancy on the application layer should also be supported by the architecture.

The domain model has similarities to a meta-model for domain-specific languages. Evans and Kent (1999) suggest devising "[...] object diagrams, snapshots, that the model must/must-not accept" to test meta-models. This is done to demonstrate how application services can be described within the architecture's scope.

Based on object diagrams, an application service will be described in section 6.4.1. It shows the applicability of the architecture to a specific service on all relevant layers. Sections 6.4.1 and 6.4.2 will evaluate the capability of the ISAA to process container-based and multi-tenant application services. These sections validate that the ISAA can be adapted to different problems than those originally anticipated.

6.4.1 Model-based application services

The domain model describes application service production on all layers of the information system. A possible state of the business layer is depicted by the object diagram in Figure 6.1.

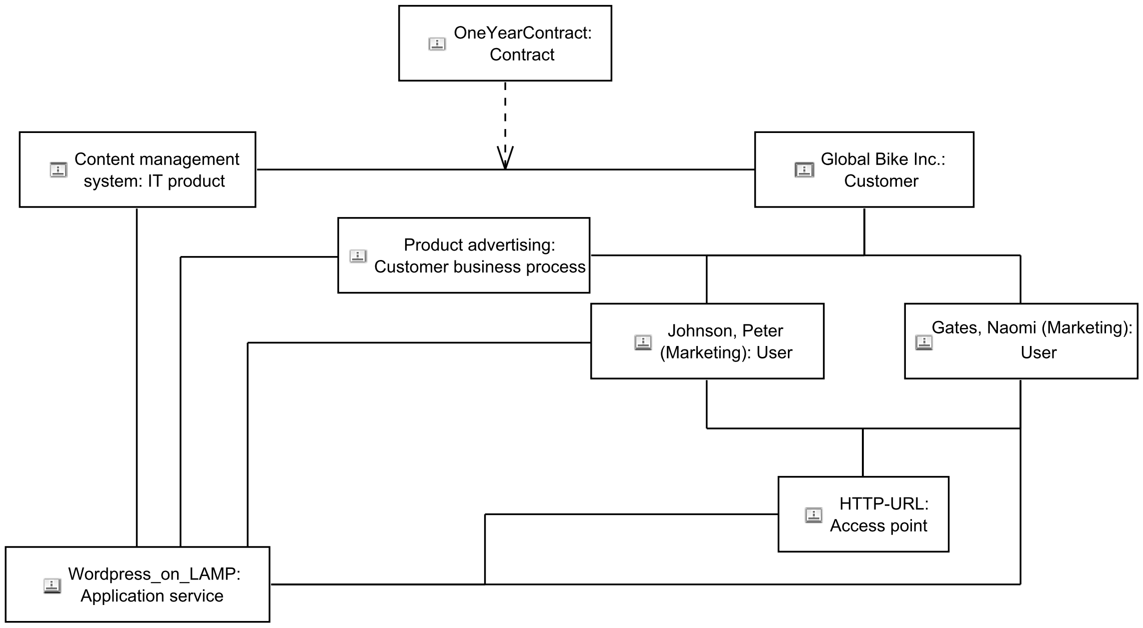

OneYearContract: Contract
Content management system: IT product
Global Bike Inc.: Customer
Product advertising: Customer business process
Johnson, Peter (Marketing): User
Gates, Naomi (Marketing): User
HTTP-URL: Access point
Wordpress_on_LAMP: Application service

Figure 6.1: Business entities of an exemplary CMS service.

The model-based application service presented in this section is the content management service already discussed in the previous sections. It is based on the frequently used software stack comprising the *L*inux operating system, *A*pache HTTP Server, *M*ySQL database management system, and *P*HP[1] programming language. This stack is known by the acronym LAMP. Based on the LAMP stack the web blogging software Wordpress is configured, which is frequently not only used for blogging, but for CMS. The combination of infrastructure, software, and orchestration configuration models that define this service are depicted in Figure 6.2.

The application service Wordpress on LAMP uses five software configuration models. These include two libraries that provide supporting configuration mechanisms to the software configuration models for the Apache HTTP Server and the MySQL database management system. The vendor of Puppet supplies all four of those models. For Wordpress itself, a popular community configuration model is selected (*hunner-wordpress*). All modules are available on the repository *PuppetForge*[2]. The orchestration configuration model specifies which software configuration models are used to deploy the respective software on the database instance and the web-server instance. The same class is instantiated twice because the objects serve different roles. However, in cases of horizontally scaling, one object could be used while adding the scaling amount to the bill of materials. This is displayed in Figure 6.11. In addition to the configuration models, a standard service level SLA is contained for this service.

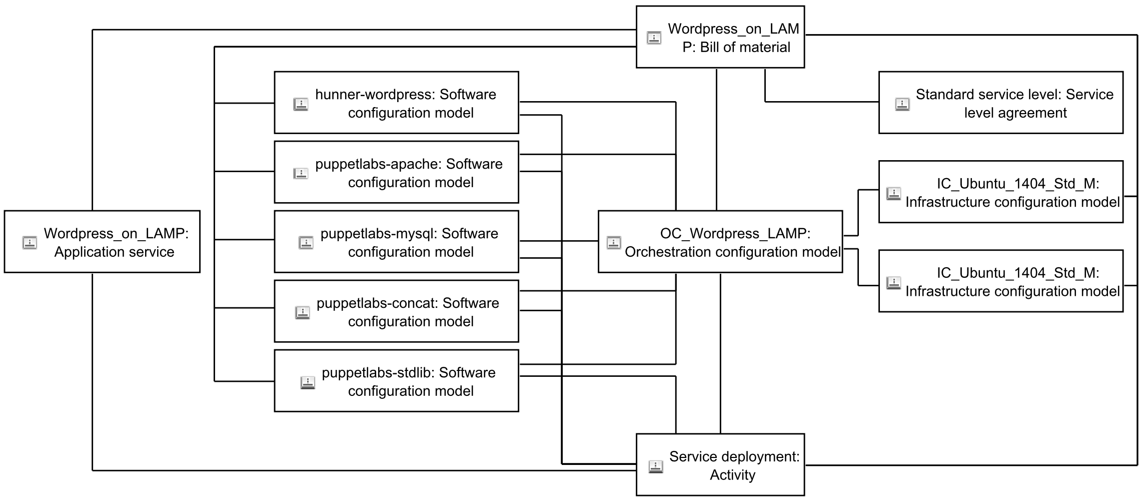

Wordpress_on_LAMP: Bill of material
hunner-wordpress: Software configuration model
Standard service level: Service level agreement
puppetlabs-apache: Software configuration model
IC_Ubuntu_1404_Std_M: Infrastructure configuration model
Wordpress_on_LAMP: Application service
puppetlabs-mysql: Software configuration model
OC_Wordpress_LAMP: Orchestration configuration model
IC_Ubuntu_1404_Std_M: Infrastructure configuration model
puppetlabs-concat: Software configuration model
puppetlabs-stdlib: Software configuration model
Service deployment: Activity

Figure 6.2: Defining models of the content management service.

Figure 6.3 displays a state in which the software configuration model *puppetlabs-apache* has been used to configure the software Apache HTTP Server. It only shows an extract. Specifically, it shows that the model provides two configuration options. The first is used to install the corresponding operating system package through the package handling utility *APT*. The second option is used to set up a virtual host on the standard HTTP port 80 for serving incoming requests. The configuration management agent induces the two produced software configuration models. They contribute to the instance's state of being

[1]PHP: Hypertext Preprocessor (PHP)
[2]https://forge.puppet.com/

deployed, with the Apache HTTP Server running, and listening for incoming requests.

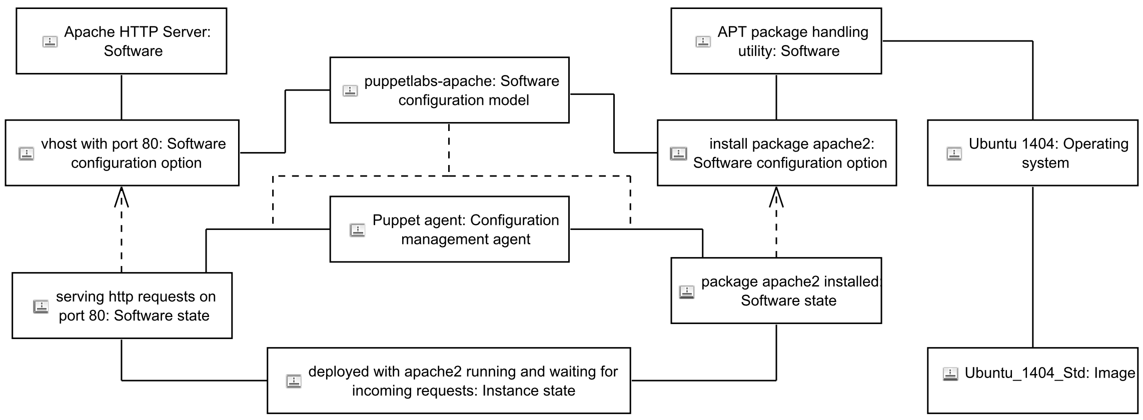

Apache HTTP Server: Software
puppetlabs-apache: Software configuration model
APT package handling utility: Software
vhost with port 80: Software configuration option
install package apache2: Software configuration option
Ubuntu 1404: Operating system
Puppet agent: Configuration management agent
package apache2 installed Software state
serving http requests on port 80: Software state
deployed with apache2 running and waiting for incoming requests: Instance state
Ubuntu_1404_Std: Image

Figure 6.3: State induced with software configuration models (extract).

Figure 6.4 displays an extract of the objects when then the service is being deployed on the infrastructure layer. Two instances are required: one for the database and one for the web server. In the shown extract, the web server has not been fully built yet, as its instance lifecycle state is *BUILDING*. In contrast, the database instance has already been build, and its lifecycle state is *ACTIVE*. The object diagram furthermore shows the virtual hardware configuration of both instances as defined by their infrastructure configuration models. The instances are hosted on the IaaS node for which the hardware configuration is also specified.

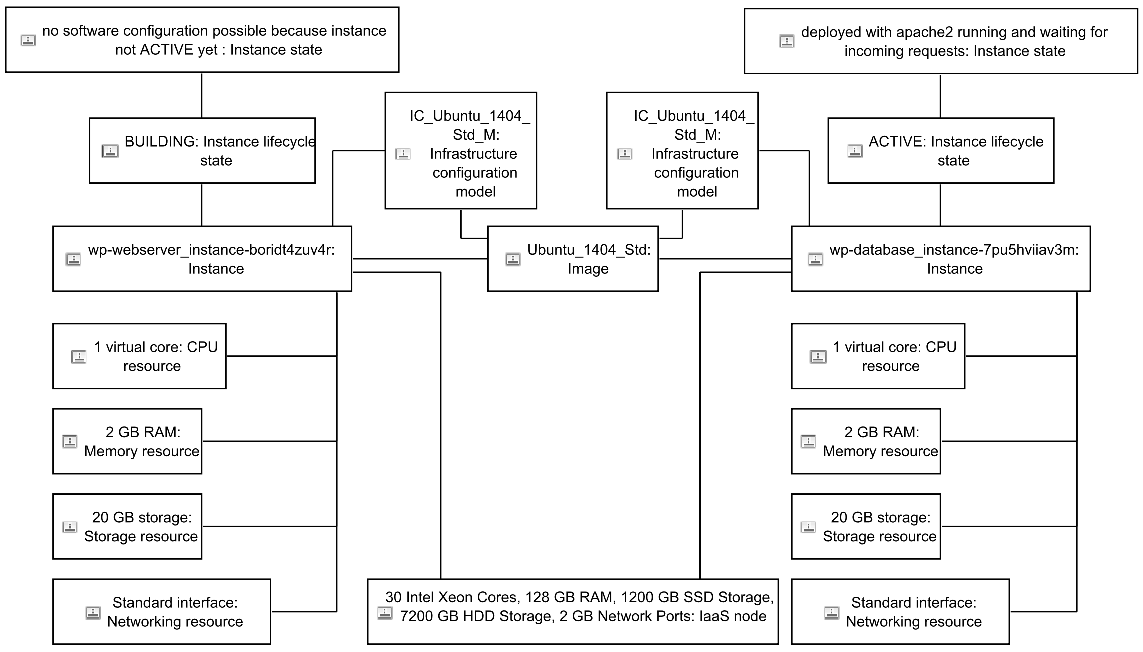

no software configuration possible because instance not ACTIVE yet : Instance state
deployed with apache2 running and waiting for incoming requests: Instance state
BUILDING: Instance lifecycle state
IC_Ubuntu_1404_Std_M: Infrastructure configuration model
IC_Ubuntu_1404_Std_M: Infrastructure configuration model
ACTIVE: Instance lifecycle state
wp-webserver_instance-boridt4zuv4r: Instance
Ubuntu_1404_Std: Image
wp-database_instance-7pu5hviiav3m: Instance
1 virtual core: CPU resource
2 GB RAM: Memory resource
20 GB storage: Storage resource
Standard interface: Networking resource
30 Intel Xeon Cores, 128 GB RAM, 1200 GB SSD Storage, 7200 GB HDD Storage, 2 GB Network Ports: IaaS node
1 virtual core: CPU resource
2 GB RAM: Memory resource
20 GB storage: Storage resource
Standard interface: Networking resource

Figure 6.4: State induced with infrastructure configuration models (extract).

Within this section, it was shown that the architecture's domain model could be used to describe application services on all layers. In the next two sections, two modifications to the made architecture decisions will be made and tested how this affects the architecture's ability to support respective landscapes.

6.4.2 Container-based virtualization

As stated, container-based virtualization has several advantages. However, architecture decision 4 mandated the use of model-based configuration management as the basis for the construction of the ISAA. Therefore, it should be tested if the operations automation approach of container-based virtualization is compatible with the architecture.

Figure 6.5 again displays a sample state of an application service that was deployed.

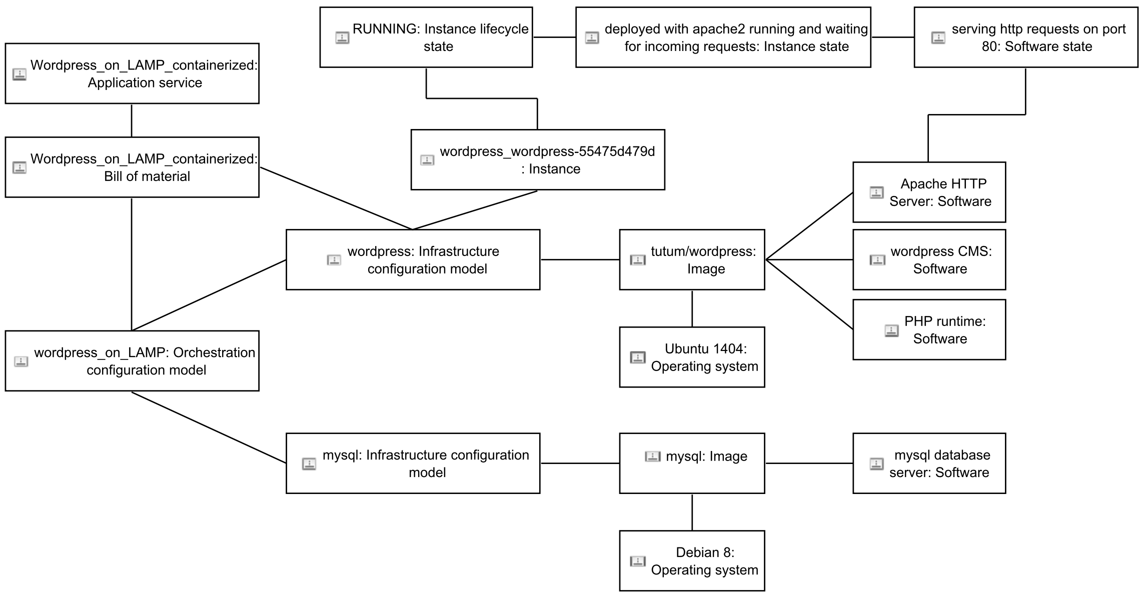

RUNNING: Instance lifecycle state
deployed with apache2 running and waiting for incoming requests: Instance state
serving http requests on port 80: Software state
Wordpress_on_LAMP_containerized: Application service
Wordpress_on_LAMP_containerized: Bill of material
wordpress_wordpress-55475d479d : Instance
Apache HTTP Server: Software
wordpress: Infrastructure configuration model
tutum/wordpress: Image
wordpress CMS: Software
PHP runtime: Software
Ubuntu 1404: Operating system
wordpress_on_LAMP: Orchestration configuration model
mysql: Infrastructure configuration model
mysql: Image
mysql database server: Software
Debian 8: Operating system

Figure 6.5: Container-based application service.

For the container-based operations automation, the content management service can be defined with only three configuration models: an orchestration and two infrastructure configuration models. Containers do not include a full operating system. In contrast to lightly baked virtual machine images, they contain the completely preconfigured stack of application software. Therefore, no software configuration models are necessary in this case. Nonetheless, the application service can be fully described with this approach. For instance, the image that includes the content management system contains preconfigured versions of the Apache HTTP Server, Wordpress, and the PHP runtime. The configuration model, which in this case only defines the image, describes the instance. When spawned, the instance offers the same functionality as described in Figures 6.4 and 6.3. The equivalent to the configuration management agent, for container-based virtualization, is not part of the domain model, but could easily be added.

In the next section, the supportability of multi-tenant application services is tested.

6.4.3 Multi-tenant application services

Figure 6.6 illustrates the way the architecture can be used to provide multi-tenant application services. The figure has four quadrants based on two dimensions. Horizontally, application system landscape production, as mainly discussed in the ISAA, and application service production with multi-tenancy above the infrastructure layer are discerned. Vertically, domain model instances and ERP master data instances are differentiated. This mapping is presented to illustrate the built-in realization of multi-tenancy of application software like ERP software can be modeled in the ISAA. The ERP system's master data is crucial here.

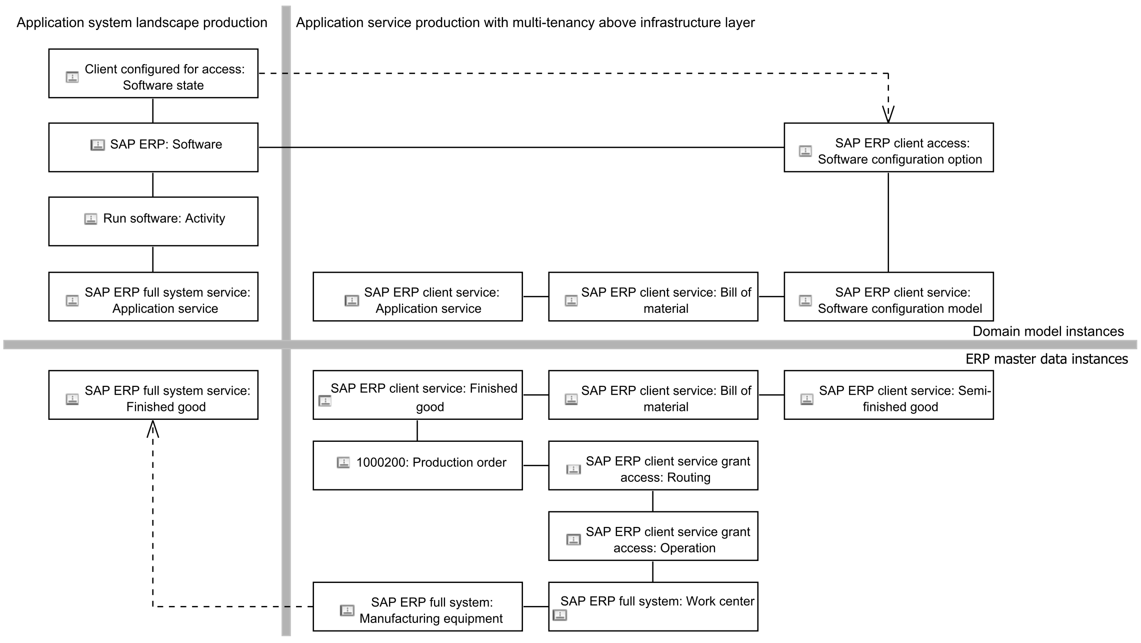

Application system landscape production
Application service production with multi-tenancy above infrastructure layer
Client configured for access: Software state
SAP ERP: Software
Run software: Activity
SAP ERP full system service: Application service
SAP ERP client access: Software configuration option
SAP ERP client service: Application service
SAP ERP client service: Bill of material
SAP ERP client service: Software configuration model
Domain model instances
ERP master data instances
SAP ERP full system service: Finished good
SAP ERP client service: Finished good
SAP ERP client service: Bill of material
SAP ERP client service: Semi-finished good
1000200: Production order
SAP ERP client service grant access: Routing
SAP ERP client service grant access: Operation
SAP ERP full system: Manufacturing equipment
SAP ERP full system: Work center

Figure 6.6: Multi-tenant application service.

The multi-tenant service is based on a *SAP ERP full system service* (lower left quadrant). The service, as depicted, is in the state of *client configured for access* (upper left quadrant). As built-in multi-tenancy, clients are used in ERP systems to logically separate organizational entities such as company branches or different companies in one ERP system (Gronau, 2010, p. 389). The state, in this case, depends on the software configuration option *SAP ERP client access* (upper right quadrant). It belongs to the multi-tenant application service *SAP ERP client service*. Within the ASLP's ERP system this is mapped to the corresponding master data instances. However, the production order does not relate to the usual manufacturing equipment (ASLP's core IT service production systems), but to *SAP ERP full system*.

This mapping could be used to operate multi-tenant services for several customers, but based on one ERP system.

6.4.4 Summary

This section has demonstrated that the domain model can describe different kinds of application services. This includes application services for which the ISAA is designed: those supported by model-based configuration management exhibiting multi-tenancy only on the infrastructure layer. However, also container-based application system landscapes can be defined. Specifically, orchestration and infrastructure configuration models are required for container-based application system landscapes. Small changes to the domain model would have to be made to reflect container management systems. Also, multi-tenant application services can be modeled if the application software supports multi-tenancy.

Customers could require matching consultancy service, for example, to customize their ERP system. Such services could be represented as finished products in the ERP system for instance as *consultant hours*. This practice is followed by personal IT service providers, whose representatives were interviewed as part of the case study described in section 2.1.

The next section will describe the prototype-based evaluation.

6.5 Evaluation based on prototype (EVAL 3)

A prototype was developed to verify the information system architecture. It is used to verify that the proposed application system landscape is implementable and that the processes can be executed as described. Furthermore, the prototype is used to investigate if the proposed automation is feasible with respect to cost and time savings. Also, using it with three different application services, the ISAA's fidelity with real-world phenomena is tested.

6.5.1 Prototype

The development of the prototype started in June 2014. This first version was used in the evaluation of the presented first version of the architecture (Hintsch et al., 2015c). Development of the second version started in 2016. While the first version of the prototype focused on the deployment step and used a custom XML[3]-based mechanism to transport data from the ERP system to the ITSP system, the second prototype supports the full process and uses almost exclusively standard SAP ERP customization templates.

6.5.1.1 IT system landscape

Figure 6.7 depicts its IT system landscape in an UML deployment diagram. SAP ERP was used as an ERP system for manufacturing. The used ERP client was pre-customized with an educational company dataset for manufacturing[4]. The client is running in an ERP system instance on a virtualized host. The system contains the depicted business modules, analog to Gronau's Gronau (2010, pp. 9–11) generic ERP architecture.

[3]eXtensible Markup Language (XML)
[4]Global Bike, Inc. (GBI) is a fictional company used for ERP system-based education (Weidner, 2012).

The ERP client's customization was extended. First, the IT products, application services, and associated master data were added. In order to be able to maintain versioning data for the configuration models in the material master, the material master was extended. Remote function call routines were developed in order to be able to extract the data from the ERP system.

SAP provides several customization templates to its customers in order adapt the system. Various industry-specific and cross-industry customization scenarios are provided. For the ISAA prototype, three primary customization scenarios were realized: To be able to implement the ETO scenarios of the ISAA, an ETO configuration template (SAP SE, 2014b) and several base templates were implemented. Similarly, for the BTO scenario, a corresponding template was implemented (SAP SE, 2015a). The interface between the SAP ERP system and the PES is based on a manufacturing execution integration scenario (SAP SE, 2014a).

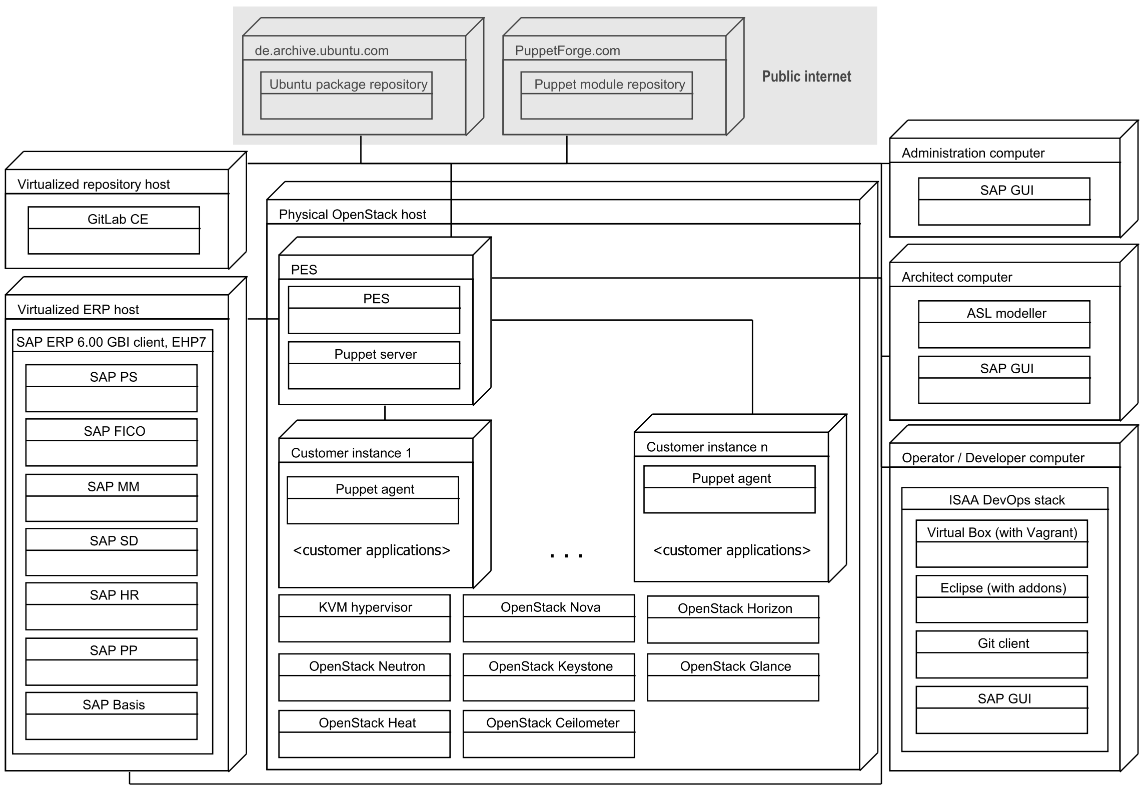


Figure 6.7: Application system landscape of the prototype in a UML deployment diagram.

OpenStack, version 12, was used as the IaaS software. OpenStack was upgraded once during the prototype development in 2015 due to stability issues. Version 12 was the then current version. The key components necessary to implement the architecture were present in version 12. Therefore, no further upgrade was performed. Contrary to production setups, a single physical host was used to deploy OpenStack for the prototype. Therefore, the IaaS master and IaaS node shared the same host. The physical host was a server blade with four Intel Xeon CPUs, 128 GB[5] of main memory, 7.2 TB[6] of serial attached SCSI[7] storage, and 1.6 TB of solid-state disk storage. OpenStack consists of core modules and optional modules. The following core modules were used for the prototype. OpenStack Nova was used for managing the hypervisor that runs the virtual instances, which in this case was the Kernel-based Virtual Machine Hypervisor. Nova uses a virtualization library called Libvirt to communicate with this hypervisor. Neutron manages the virtual networks. Keystone is OpenStack's identity service, and Glance is primarily responsible for storing the instances' images. Furthermore, the prototype installation contained three optional modules: Heat, for orchestration, Ceilometer, for monitoring, and Horizon, for providing a web interface for user interaction. In addition to managing and hosting the customer instances that form the infrastructure basis for the customers' application services, OpenStack instances are also used to host the PES, which will be discussed in section 6.5.1.2. Also, a virtualized repository host was included that contains the version control system GitLab community edition.

Two components from the public internet were part of the landscape. One was the software package repository of Ubuntu. Ubuntu was used as one important operating system throughout the prototype. In production set-ups, a local package repository may be used to avoid network problems and to control which package versions are in the repository. Configuration models were loaded from puppetforge.com and a local version control system.

Three computer configurations are suggested to support the different roles (administration, architecture, development, and operation) in the prototype. To conduct the tasks of the administration, the SAP *G*raphical *U*ser *I*nterface suffices. It allows to connect to the SAP ERP system and perform all necessary steps to complete the designated tasks. The architect computer has the ASL modeler for architecting application system landscapes. This software will be described in more detail in section 6.5.1.4, in addition to being able to connect to the SAP ERP system. Lastly, the operator and developer use computers with the so-called ISAA DevOps stack. It contains a desktop hypervisor and management software for running virtualized approximations of production application system landscapes for testing and development. In addition, the IDE Eclipse is included with add-ons, such as the ASL modeler. A client of the version control system Git, as well as the SAP *G*raphical *U*ser *I*nterface, were also included.

It was renounced to include a continuous integration system into the prototype. Its feasibility is prominently shown (Humble and Farley, 2010; Ebert et al., 2016). Also, as not in primary focus, a monitoring and a security system were not included (cf. section 5.3).

[5]Gigabyte (GB)
[6]Terabyte (TB)
[7]Small Computer System Interface (SCSI)

6.5.1.2 Production execution system

The PES was developed in Java. To provide the PE API to the ERP system, the SAP Java IDoc Class Library 3.0 was utilized. Production orders are sent in the IDoc format to the PES. For the production execution system to use the ERP API, the SAP Java Connector 3.0 was implemented. It allows to execute remote function calls on the SAP ERP system.

Shell scripts were used to communicate with the Puppet master, to control Heat, the orchestration module of OpenStack, to interact with Ceilometer, the metering module of OpenStack, and to load configuration models from the version control system. An interface was programmed to integrate them into the otherwise Java-programmed PES.

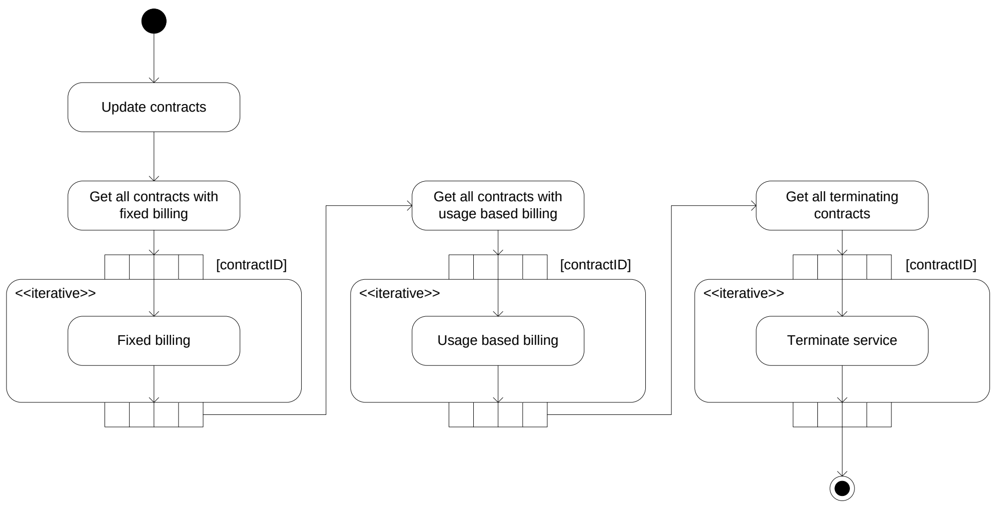


Figure 6.8: Thread starting billing activities.

The PES will react to incoming production orders. However, it also will periodically run the thread displayed in Figure 6.8 for conducting billing activities. The first step is to iterate through all contracts that are marked as fixed billing. A custom text field of the contract master data is used to differentiate fixed billing from usage-based billing contracts. When all contracts in the particular interval have been billed as required, the thread continues. All usage-based contracts are iterated, and the instances' usage is retrieved via the interface from Ceilometer. In the last step, all terminating contracts are retrieved. Next, they are iterated, and each one is terminated.

6.5.1.3 Exemplary application services

Table 6.7 shows which software was used to implement each of the application services.

For the remote desktop service, the software Guacamole was used. Guacamole[8] offers a client based on Hypertext Markup Language 5 to access desktop sessions remotely via different protocols such as *Remote Desktop Protocol* or *Virtual Network Computing*. Guacamole has dependencies to the MySQL database management system and the Apache

[8]http://guacamole.incubator.apache.org/

Tomcat application server, also for proxying connections over port 80, Apache HTTP Server was used. For the prototype, the Remote Desktop Protocol was selected, as the client machines ran with Windows Server 2012. Furthermore, some other minor dependent software packages were used. An Ubuntu Server 14.04 LTS[9] image was utilized to run the server instance that acts as the service's access point. On all of the Windows clients, the office suite LibreOffice is installed. Table C.12 shows the structure of configuration models of this service. Two configuration models are used for the infrastructure. For the database, application and HTTP servers, software configuration models from puppetforge.com were used. One orchestration configuration model orchestrates the whole service. For the central server, one software configuration model was developed as well as for the Windows clients. The following listing shows a part of the orchestration configuration model. The parameters *CLIENTCOUNT* and *CLIENTSIZE* are used to define how many clients and of which size are deployed for a service instance respectively.

```
parameters:
  CLIENTCOUNT:
    type: number
    label: Client count
    description: Number of clients for the remote desktop
    default: 2

  CLIENTSIZE:
    type: string
    label: Client size
    description: Size of the clients
    default: m1.small
```

Application service	**Main software**	**Images**
Remote desktop	Guacamole, MySQL, Apache HTTP Server, Apache Tomcat, LibreOffice, and other dependencies	Ubuntu Server 14.04 LTS (Trusty Tahr), Windows Server 2012
SAP ERP	SAP ERP 6.04 SP 13 NetWeaver 7.01 SP 13 with GBI 2.20r001	SUSE Linux Enterprise 11 SP3 with SAP dependencies
CMS	Apache HTTP Server, MySQL, Wordpress, and other dependencies	Ubuntu Server 14.04 LTS (Trusty Tahr)

Table 6.7: Technical implementation of the application services.

[9]Long time support (LTS)

SAP ERP is restricted to specific operating systems. Therefore, a preconfigured image of SUSE Linux Enterprise 11 SP3 was created using SuseStudio[10]. The software configuration model installs SAP ERP using the system copy mode and unattended version of the SAP installation software (SAP SE, 2017a). The following listing shows an excerpt from the software configuration model. It is written in Puppet's domain-specific language. It shows the directory initialization that is necessary to commence with the installation of the ERP software.

```
$sap_dirs = ['/sapinst/sapdb', '/sapinst/sapmnt', '/sapinst/usrsap', ]
    file { $sap_dirs :
        ensure => directory,
        owner => 'root',
        group => 'root',
        mode => '755',
        require => Exec['setup_vdb']
    }
```

The start configuration of the content management service that is engineered to order in the following section has repeatedly been used as an example throughout the thesis. All software configuration models are loaded from puppetforge.com. The orchestration configuration model combines them to deploy the service. In the following listing, the configuration of the service's access point is defined. The triple bracket enclosed word is used to insert parameters that are in this case retrieved from the IaaS master upon spawning the instances.

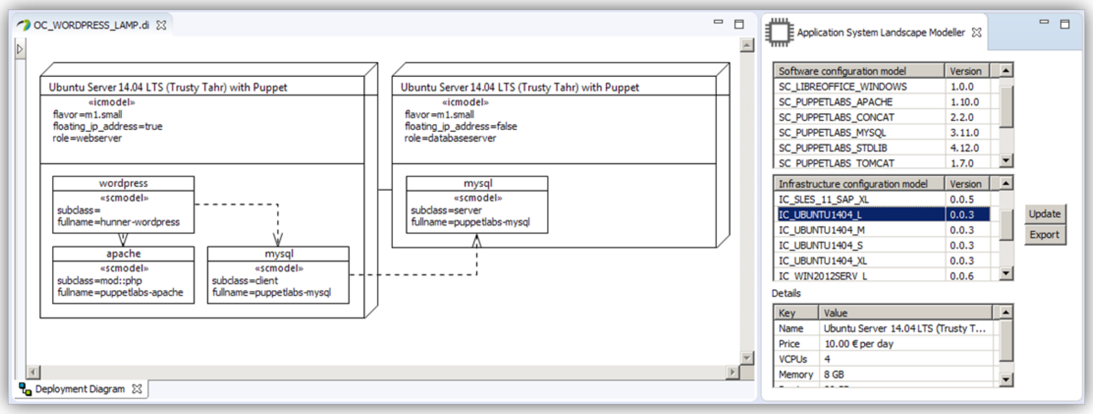
```
    apache::vhost { '<<<access_point>>>':
        port    => '80',
        servername    => '<<<access_point>>>',
        docroot => '/var/www/wordpress',
    }
```

6.5.1.4 Computer aided configuration model generation

A tool was created to facilitate the creation of a landscape solution design at the beginning of the engineering phase. It is based on the integrated development environment Eclipse and the modeling Plugin Papyrus. Figure 6.9 shows an example in which the content management service is modeled.

Architects can use the application system landscape modeler (ASL modeler) to create a deployment diagram of the service's landscape. The ASL modeler interfaces with the SAP ERP system to display the available software and infrastructure configuration models. Architects can view details for each model such as pricing information. The deployment diagrams are extended by UML stereotypes (Oestereich and Scheithauer, 2012, p. 292). This enables the definition of arbitrary attributes within the model.

[10]https://susestudio.com/



Software configuration model	Version
SC_LIBREOFFICE_WINDOWS	1.0.0
SC_PUPPETLABS_APACHE	1.10.0
SC_PUPPETLABS_CONCAT	2.2.0
SC_PUPPETLABS_MYSQL	3.11.0
SC_PUPPETLABS_STDLIB	4.12.0
SC_PUPPETLABS_TOMCAT	1.7.0

Infrastructure configuration model	Version
IC_SLES_11_SAP_XL	0.0.5
IC_UBUNTU1404_L	0.0.3
IC_UBUNTU1404_M	0.0.3
IC_UBUNTU1404_S	0.0.3
IC_UBUNTU1404_XL	0.0.3
IC_WIN2012SERV_L	0.0.6

Key	Value
Name	Ubuntu Server 14.04 LTS (Trusty T...
Price	10.00 € per day
VCPUs	4
Memory	8 GB

Figure 6.9: Application system landscape modeller.

After the solution design has been drafted, the model can be used for different purposes. The first purpose is to communicate with the customer if the suggested solution adheres to his requirements. The second purpose is to derive initial price estimates. Because all data is retrieved from the ERP system, pricing data can also be derived. Consequently, price estimates can be made based on the available configuration models. Finally, the deployment diagrams can be used as a starting point for the orchestration configuration models. Because the deployment diagrams are encoded as standard XML, they can be manipulated easily. For the ASL modeler, language transformations[11] from the XML-representation of the UML deployment diagram into the Heat and Puppet orchestration configuration models were used that are required for orchestration in the prototype.

In the following, process run-throughs are described, which were performed with the prototype.

6.5.2 The production process in different variations and phases

For the production process run-throughs, each phase is included, and the ETO and BTO scenarios are differentiated. Tables are used to list the process steps. The first process description includes a table within this section. The other tables can be found in the appendix. Here, those run-throughs are described only in the text as the tables are somewhat repetitive.

6.5.2.1 Full build-to-order

The build-to-order process starts with gathering the customer requirements. For the prototype, the customer requirements were implicitly gathered and directly recorded in the ERP system. Table 6.8 displays the phases and activities that are performed to produce the service. The table also shows the application system context and the nature of the activity: namely one with human interaction or system activity.

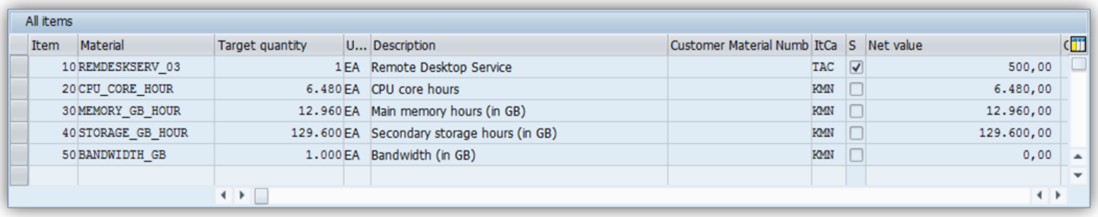
[11]The transformation was based on the eXtensible Stylesheet Language.

Business process	**Activity**	**System context**	**Activity type**
Inquiry and order processing	*[Incoming customer inquiry]*		
	Gather customer requirements	SAP-VA11	HI
	Create contract, with configuration for five small clients of remote desktop service	SAP-VA41	HI
	Create deployment sales order	SAP-VA01	HI
Deployment	Create production order	SAP-CO01	HI
	Release production order	SAP-CO02	HI
	Service deployment	PES	SYS
	[Production order confirmation]		
Operation	{*Service operating*}	ITSP systems	SYS
Deployment	Notify customer	SAP-VL01N	HI
Billing	Bill customer	SAP-VF01	HI
	{*Service billing*}	PES	SYS
Termination	Service termination	PES	SYS

	Legend
Activity	regular system or user interaction activity, [event], continuous system activity
Application system context	SAP- precedes SAP ERP transactions: VA1[1/2] - Inquiry management (create/change), VA4[1/2] - Contract management (create/change), VA0[1/2] - Sales order management (create/change), VL0[1/2]N - Outbound delivery management (create/change), VF0[1/2] - Billing document management (create/change), CAT2 - Maintaining timesheet data, CO0[1/2] - Production order management (create/change), MM0[1/2] - Material management (create/change), CS0[1/2] - Bill of materials management (create/change), IW5[1/2] - service notification management (create/change)
Activity type	HI - user interaction activity, SYS - system activity

Table 6.8: Process steps for deploying a BTO service.

After the inquiry, a contract is created to which the details entered in the inquiry are copied. The contract contains the application service's IT product representation (a finished good). A remote desktop service was used. When the remote desktop service is entered into the contract, the system recognizes that it is a configurable material and asks for the configuration parameters. The contract also contains the necessary amount of resources to run the service for the agreed-upon period. Figure 6.10 shows an extract from the contract creation transaction (VA41). In this case, the service has a duration of 30 days. A small configuration for the five clients was agreed upon. Consequently, the following calculation leads to the displayed quantities. The remote desktop server (access point) has four virtual CPU cores, eight GB of main memory, and 80 GB of secondary storage. For each small client one CPU core, two GB of main memory and 20 GB of secondary storage are necessary. This, for example, results in a total of 6.480 CPU hours for 9 CPU cores $(4 + 5 * 1)$ that are active 24 hours a day for 30 days. Also, 1000 GB of bandwidth is agreed upon.


All items

Item	Material	Target quantity	U...	Description	Customer Material Numb	ItCa	S	Net value
10	REMDESKSERV_03	1	EA	Remote Desktop Service		TAC	✓	500,00
20	CPU_CORE_HOUR	6.480	EA	CPU core hours		KMN	☐	6.480,00
30	MEMORY_GB_HOUR	12.960	EA	Main memory hours (in GB)		KMN	☐	12.960,00
40	STORAGE_GB_HOUR	129.600	EA	Secondary storage hours (in GB)		KMN	☐	129.600,00
50	BANDWIDTH_GB	1.000	EA	Bandwidth (in GB)		KMN	☐	0,00

Figure 6.10: Materials in the contract for remote desktop service.

Once the contract has been saved, a deployment order is generated. Figure 6.11 shows, as a UML object diagram, the remote desktop service and how on the different abstraction levels of the ISAA the production order is represented. Once the production has been created and released, it is sent to the PES. There, the system activity *deployment* commences.

The PES confirms the deployment of the service and reports the access points' address back to the ERP system. This information is used in the notification of the customer. The customer is then billed with an initial deployment invoice. During the operation of the service, the PES will generate sale orders and invoices. At the end of the contract, if it is not prolonged, the service is terminated. In the prototype, the full instances are stored to disk before being terminated. These disk images are then made available to the customer via a web download.

In Table C.1 the change of a service parameter during the operation phase is shown. This process starts with a service request made by a user. It is entered into the ERP system in the form of a customer notification with transaction IW51. The deployed remote desktop service is canceled with a repair order taking the customer notification as a reference. The contract is then updated with the new number of clients. When the production order has been sent to the PES, this system updates the information associated with the unique identifier of this service. The IaaS master then spawns a new instance, and the configuration master configures it as requested. The process is completed with notifying and billing the customer.

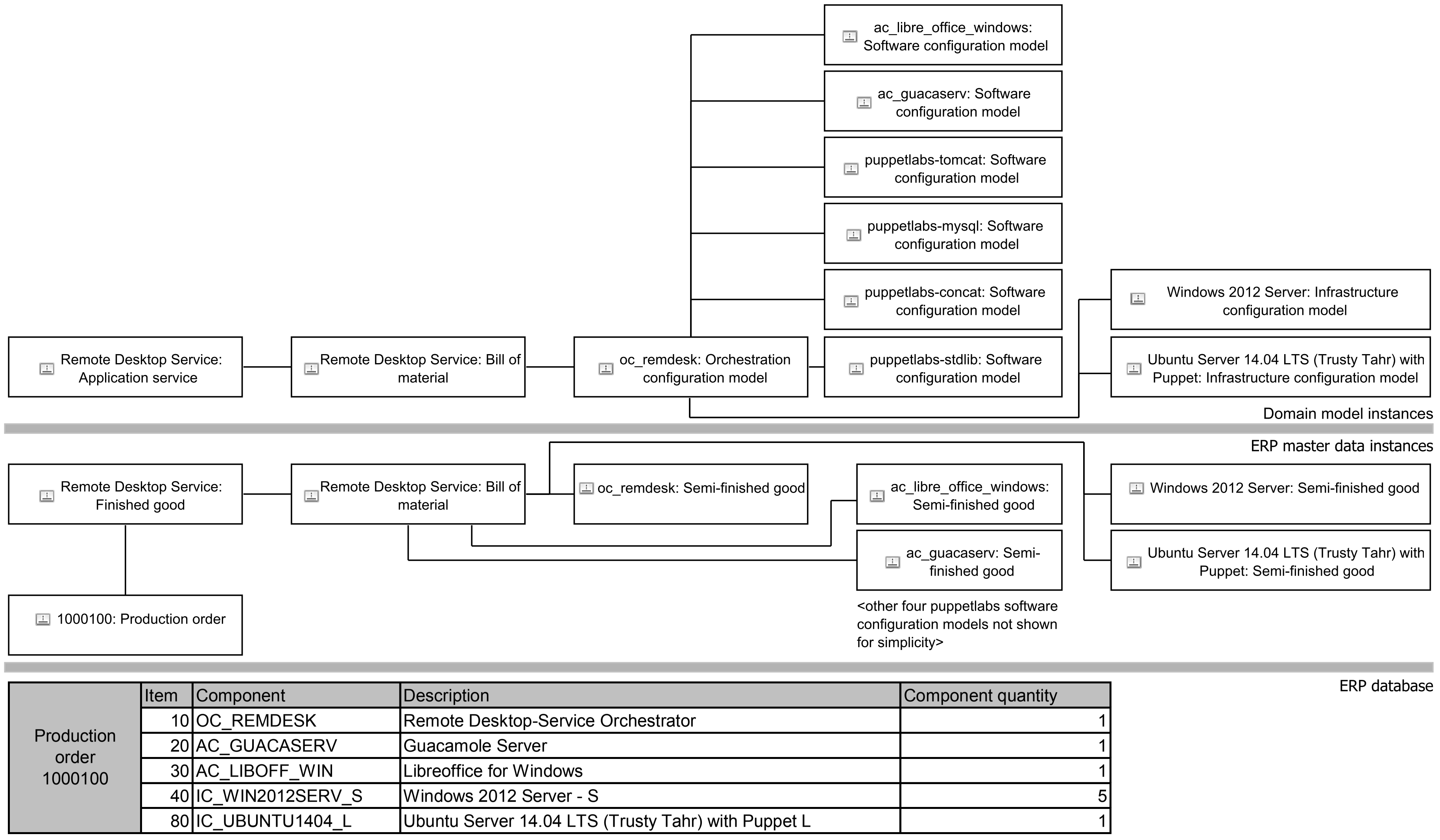
In the next section, the engineer-to-order process is discussed.

6.5.2.2 Full engineer-to-order

Table C.2 shows how engineer-to-order production is performed. In this case, the customer does not request a content management service, but a ticketing system.

When the ETO sales order is placed, a project is created to which all efforts for engineering the new service are posted. The software Redmine[12] is selected during the *Create solution design* activity. As a software configuration model is already available

[12]www.redmine.org

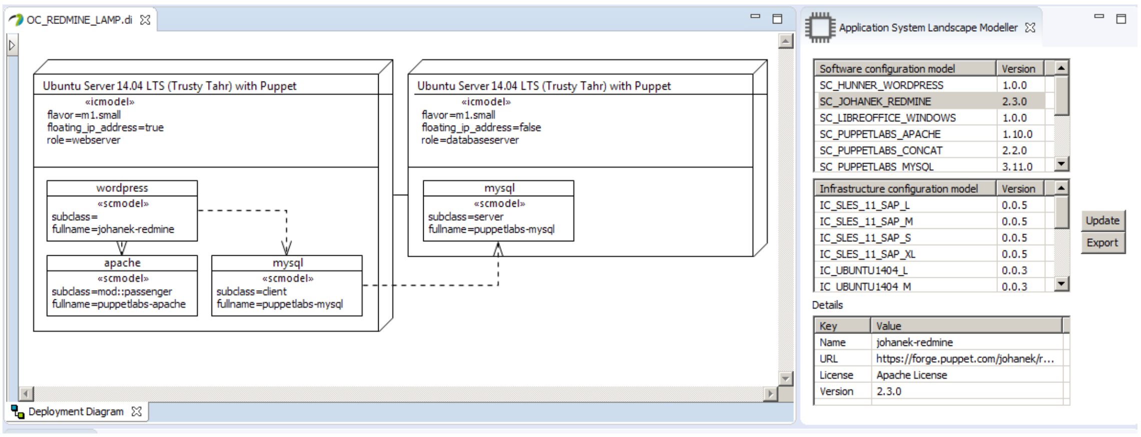

ac_libre_office_windows: Software configuration model
ac_guacaserv: Software configuration model
puppetlabs-tomcat: Software configuration model
puppetlabs-mysql: Software configuration model
puppetlabs-concat: Software configuration model
puppetlabs-stdlib: Software configuration model
Remote Desktop Service: Application service
Remote Desktop Service: Bill of material
oc_remdesk: Orchestration configuration model
Windows 2012 Server: Infrastructure configuration model
Ubuntu Server 14.04 LTS (Trusty Tahr) with Puppet: Infrastructure configuration model
Domain model instances
ERP master data instances
Remote Desktop Service: Finished good
Remote Desktop Service: Bill of material
oc_remdesk: Semi-finished good
ac_libre_office_windows: Semi-finished good
ac_guacaserv: Semi-finished good
Windows 2012 Server: Semi-finished good
Ubuntu Server 14.04 LTS (Trusty Tahr) with Puppet: Semi-finished good
<other four puppetlabs software configuration models not shown for simplicity>
1000100: Production order
ERP database

	Item	Component	Description	Component quantity
Production order 1000100	10	OC_REMDESK	Remote Desktop-Service Orchestrator	1
	20	AC_GUACASERV	Guacamole Server	1
	30	AC_LIBOFF_WIN	Libreoffice for Windows	1
	40	IC_WIN2012SERV_S	Windows 2012 Server - S	5
	80	IC_UBUNTU1404_L	Ubuntu Server 14.04 LTS (Trusty Tahr) with Puppet L	1

Figure 6.11: Application service showing entities on different abstraction levels.

for this software on puppetforge.com, only the orchestration configuration model needs to be developed. Figure 6.12 shows how the configuration model is drafted in the ASL modeller during the activity *create solution design*. Subsequently, it is developed using the ISAA DevOps stack. After the development is completed, the material master is updated. Then, an acceptance test is performed. For the acceptance test, the service is deployed in a testing environment in the ITSP system. Finally, the contract and ETO sales order are updated with the newly created service.

After that, the process continues the same way that it has in the BTO scenario.



Ubuntu Server 14.04 LTS (Trusty Tahr) with Puppet
«icmodel»
flavor=m1.small
floating_ip_address=true
role=webserver

wordpress
«scmodel»
subclass=
fullname=johanek-redmine

apache
«scmodel»
subclass=mod::passenger
fullname=puppetlabs-apache

mysql
«scmodel»
subclass=client
fullname=puppetlabs-mysql

Ubuntu Server 14.04 LTS (Trusty Tahr) with Puppet
«icmodel»
flavor=m1.small
floating_ip_address=false
role=databaseserver

mysql
«scmodel»
subclass=server
fullname=puppetlabs-mysql

Software configuration model	Version
SC_HUNNER_WORDPRESS	1.0.0
SC_JOHANEK_REDMINE	2.3.0
SC_LIBREOFFICE_WINDOWS	1.0.0
SC_PUPPETLABS_APACHE	1.10.0
SC_PUPPETLABS_CONCAT	2.2.0
SC_PUPPETLABS_MYSQL	3.11.0

Infrastructure configuration model	Version
IC_SLES_11_SAP_L	0.0.5
IC_SLES_11_SAP_M	0.0.5
IC_SLES_11_SAP_S	0.0.5
IC_SLES_11_SAP_XL	0.0.5
IC_UBUNTU1404_L	0.0.3
IC_UBUNTU1404_M	0.0.3

Details

Key	Value
Name	johanek-redmine
URL	https://forge.puppet.com/johanek/r...
License	Apache License
Version	2.3.0

Figure 6.12: Modelling the new orchestration configuration model.

In the next section, the operation phase is discussed in two variations.

6.5.2.3 Different changes in operation

Table C.3 shows the activities to be performed when a service request requires altering a model. In this instance, it is the creation of another database in the ticketing service of the customer. The software configuration model is changed. Development times are posted to the project of the respective material. Then the material master is updated, and the service canceled with a repair order. The contract is updated, and a new deployment sales order is created. Subsequently, the changed service is released into production, and the usual steps for billing and notification follow.

Table C.4 shows the activities for fixing a bug in a customer's landscape. In this case, the software configuration model is changed. The changes are tested, and the material master is updated. Again, all efforts are posted to the project of the respective material. Projects are generated when the respective materials are first created. Material creation occurs in customer-initiated ETO scenarios or when the provider decides to develop new service components. Because the bug can affect multiple services, a full regression test of the services that the respective material is used it has to be performed. Once the regression test has been passed, the customer-wide deployment commences. After this, the customers are informed. Depending on the agreements with the customer or company policy, a customer notification can also be sent out before deployment.

The customer-wide deployment is not initiated from within the SAP ERP system. This would mean that repair and production orders would have to be created for every affected service instance. Therefore, these activities are performed, and only the efforts are posted to the respective material's project.

6.5.2.4 Extending contract before termination

When the end of a contract approaches, the customer should be asked to extend the contract. If the customer agrees to extend the contract, this needs to be reflected in the ERP system. Therefore, the administration employee would modify the duration of the contract by prolonging the dates. Also, in the case of fixed price regardless of volume, the amount of consumable IaaS resources would need to be increased. Increasing resource would not be necessary for pay-per-use billing. The steps are shown Table C.5.

In the next section, the feasibility of automation is discussed.

6.5.3 Feasibility of automation

In this section, the feasibility of automating the deployment of application system landscapes is evaluated. The development of configuration models is not for free (Talwar et al., 2005). For developing configuration models and maintaining them, costs are incurred just like for any other software. These include initial planning and development costs as well as reoccurring development costs (Zarnekow et al., 2004).

Employing configuration models has a variety of benefits as discussed throughout the thesis. Amongst them, automating deployment and operation is a crucial advantage. Careless mistakes occurring in manual deployment can be avoided (Talwar et al., 2005). Another benefit may be the increase in speed gained when employing automated deployment.

The following model compares automated and manual deployment and modification in terms of the time incurred. The deployment or modification of an application system landscape includes different steps, such as the creation of a file. All steps have a total execution time. The time in which the system is in the state of execution is $E = \sum_{i=1}^{n} e_i$ where e_i is the time of an execution step and n the total number of steps. In particular, in the manual case, each execution step is preceded by preparation time. This includes the operator entering a command or filling out a form in a user interface. The sum of preparation time is $P = \sum_{i=1}^{n} p_i$ where p_i is the preparation time for each step. Wait time can occur between steps. The operator might go for a coffee when a job was started. His absence may surpass the time in which the system was in an executing state. Wait time could also occur in an automation setting where a resource is busy, and therefore a process has to wait for a certain step. The sum of wait time occurred in the deployment of one service is $W = \sum_{i=1}^{n} w_i$. Consequently, the total time for a deployment can be described by the following equation:

$$D = \sum_{i=1}^{n} e_i + p_i + w_i$$

In Table 6.9 an example of automated and manual deployment times is sketched. It shows for $n = 3$ the manual deployment and the automated deployment for $n = 2$. Different numbers of steps can occur because configuration models can abstract multiple manual steps. Consider, for example, the creation of a file. Manually creating a file can include four steps. The file is created, access rights and the file's owner are changed. Then, the file's content is edited. This is shown in the following snippet of shell commands.

```
$ touch /etc/ssh/sshd_config
$ chmod 600 /etc/ssh/sshd_config
$ chown root:root /etc/ssh/sshd_config
$ vi /etc/ssh/sshd_config
```

On the other hand, a declaration in Puppet DSL, as shown below, would only require being executed once. It could also be executed repeatedly and would always, predictably, assure the same file state (idempotency).

```
file { '/etc/ssh/sshd_config':
  source  => 'puppet:///modules/sshd/sshd_config',
  owner   => 'root',
  group   => 'root',
  mode    => '0640'
}
```

Table 6.9 shows that only p_1 is larger than zero. All other steps are not preceded by a preparation step larger than zero. Preparation step p_1 is equal to the time the configuration management master takes to translate or compile the configuration model to platform- and node-specific configuration instructions. These instructions are sent to the agent to perform the configuration (Delaet et al., 2010). No wait time is assumed to occur after the automated steps.

In particular, if few service instances are deployed the development effort of automation can be questionable. The development time required for a particular service can be defined as *devtime*.

$$devtime = \frac{loc}{pr}$$

Lines of code are defined as*loc* and *pr* as development productivity for developing the configuration models. The question that is addressed in this section is at how many deployments m the following inequation becomes true for three services of the prototype.

$$m * (\sum_{i=1}^{n} e_i + p_i + w_i) < dev + m * (\sum_{i=1}^{o} e_i + p_i + w_i),$$

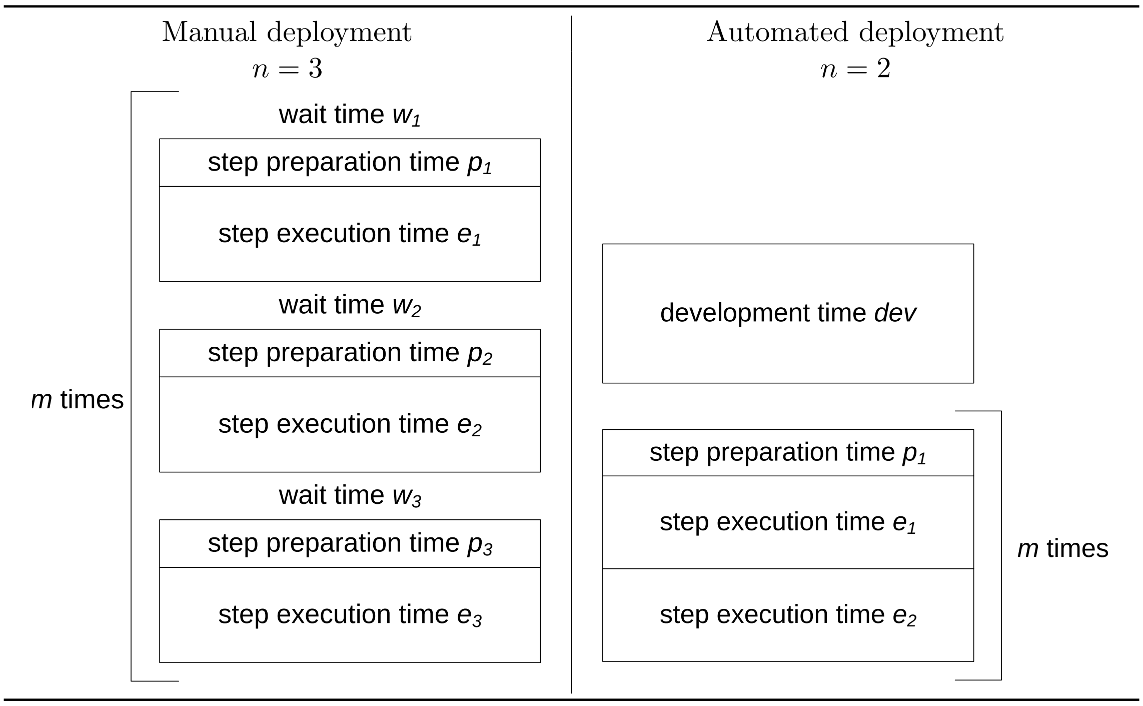

Manual deployment
$n = 3$
wait time w_1
step preparation time p_1
step execution time e_1
wait time w_2
step preparation time p_2
step execution time e_2
wait time w_3
step preparation time p_3
step execution time e_3
m times
Automated deployment
$n = 2$
development time dev
step preparation time p_1
step execution time e_1
step execution time e_2
m times

Table 6.9: Multiple deployments of a service in a manual and automated fashion.

where n is the number of manual and o the number of automated steps. In short, this can be defined as

$$m * D_{man} < dev + m * D_{auto},$$

where $D_m an$ is the total manual deployment time and $D_a uto$ the total automated deployment time.

6.5.3.1 Deployment times

For all tests, the initial version of the application service of the ETO scenario was used. The full list of steps necessary for manually deploying each service is shown in the appendix in Tables C.6, C.7, C.8, C.9, C.11, and C.10.

The author tested the manual deployment; no further individuals were asked to perform the deployments. These times are highly dependent on the individual who performs the work. They depend on such factors as typing speed and familiarity with the process. These manual deployment times only have illustrative character. Table 6.10 displays the measured times for deployment.

Table 6.10 shows how long it took to deploy each service. The deployments were performed with the instructions displayed on the second screen of the author's computer. For the remote desktop as well as for the content management service the accumulated preparation time exceeds the actual execution time of the deployment. This is not true for the SAP ERP service where more than seven hours were required to execute the installation step. This long execution time can be explained by the fact that the ERP software was provisioned fully customized with data for training purposes.

Service	Number of steps	D_{man}	$\sum_{i=0}^{n} p_i$	$\sum_{i=0}^{n} e_i$	$\sum_{i=0}^{n} w_i$
Remote desktop	79	0:28:04	0:14:42	0:10:20	0:03:02
SAP ERP	36	7:40:20	0:12:52	7:25:17	0:02:08
Content management	35	0:07:34	0:05:05	0:01:42	0:00:44

Times written format *hh:mm:ss*.

Table 6.10: Manual deployment times for the three services.

The automated deployment, as expected, took less time compared to manual deployment. Table 6.11 shows the deployment of the three services in different configurations. Each service in a particular configuration was deployed five times to get an average time. During that time no additional load was placed onto the physical OpenStack host.

The remote desktop service was run in six different configurations. Small, medium and large clients and different numbers of clients were used. Different sizes have a smaller effect than the effect additional clients have on the overall deployment time. This is obvious as an additional client requires an additional configuration run on that instance of the configuration agent.

Service, deployment configuration	*avg*	*min*	*max*	*sd*
Remote desktop, 2 small clients	0:16:08	0:15:17	0:16:33	0:00:27
Remote desktop, 2 medium clients	0:17:12	0:17:01	0:17:25	0:00:08
Remote desktop, 2 large clients	0:17:32	0:17:24	0:17:41	0:00:06
Remote desktop, 3 small clients	0:18:37	0:18:04	0:19:02	0:00:25
Remote desktop, 3 medium clients	0:19:59	0:19:42	0:20:12	0:00:10
Remote desktop, 4 small clients	0:20:50	0:19:30	0:21:41	0:00:49
SAP ERP, standard	7:40:45	7:27:12	7:54:18	0:13:33
Content management, standard	0:03:55	0:03:49	0:04:05	0:00:06
Content management, parallel deployment	0:04:57	0:04:34	0:05:08	0:00:12

Times written format *hh:mm:ss* , *avg* - average, *min* - shortest time, *max* - longest time, *sd* - standard deviation

Table 6.11: Automated deployment times for different configurations.

The content management service was deployed in a standard and parallel fashion. The parallel deployment started all five service deployments concurrently. As expected, this resulted in a more extended deployment time per service instance. The SAP ERP service exhibits a comparatively high variance of its deployment time as indicated by a standard deviation of 13 minutes and 33 seconds. Relative to its overall deployment time, this variation is similar to that, for example, of the different remote desktop deployment configurations.

Figures 6.13 and 6.14 show the memory and CPU utilization of the PES' instance. For $t = 1, ..., 8$ the deployment of the remote desktop and the content management service

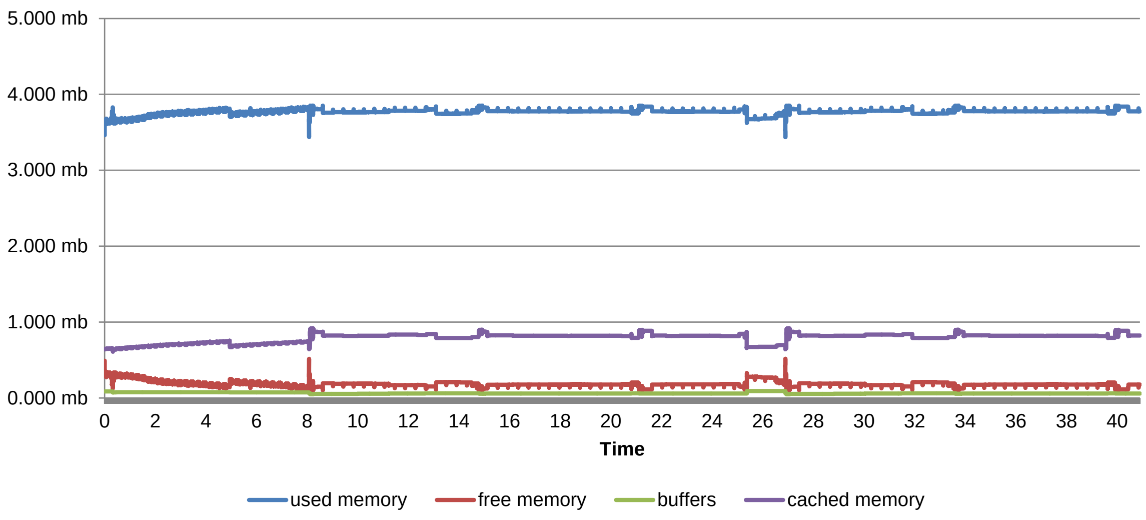


Figure 6.13: Memory utilization of the PES during deployment *(for $t = 1, ..., 40$)*.

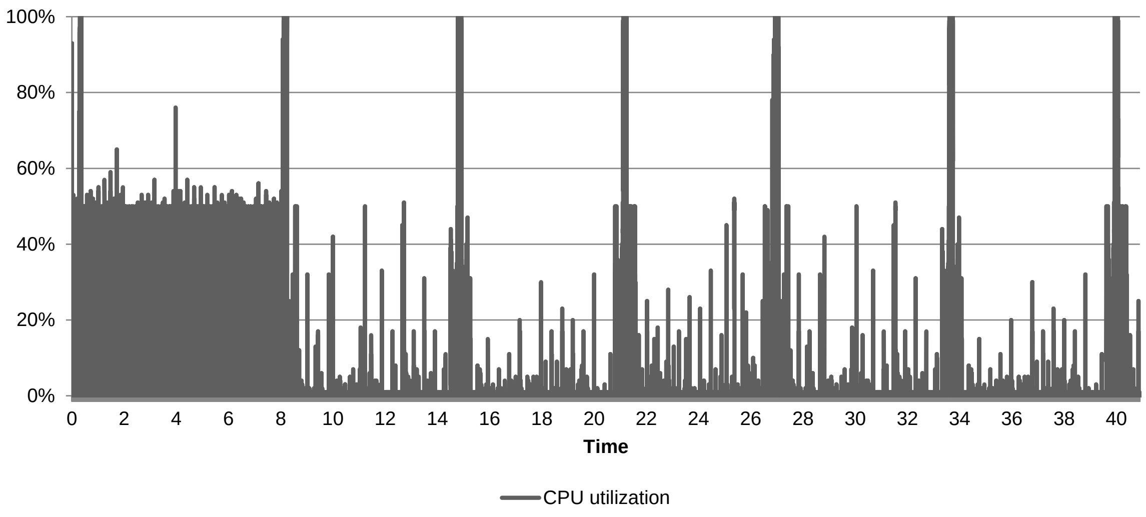


Figure 6.14: CPU utilization of the PES during deployment (for $t = 1, ..., 40$).

was performed. In $t = 8$, the deployment of SAP ERP started. Each interval between the CPU utilization peaks from $t = 8, ..., 40$ represents the time in which an SAP ERP service instance is deployed. There is a last peak at $t = 40$. When a deployment was completed, log data was copied from the instance to a central location on the PES. The usage of the memory reflects the memory handling of the Java-based Puppet master. An increase very close to $t = 0$ can be spotted. Here, the content management service was deployed in parallel. After that, drops occur twice after the deployment of an SAP ERP service has started.

The consumption statistics indicate that the PES has to be sized according to the expected demand. It could also be possible to apply a load distribution strategy in which deployments of the services are spread out evenly across the available time.

The next section presents a comparison between the manual and automated deployment.

6.5.3.2 Comparison

A software development productivity indicator is required to compare the manual deployment with the automated deployment. Because the automation of service deployment or modification is preceded by the development of the corresponding models the time required for development has to be added in the case of automated deployment (see the previous section).

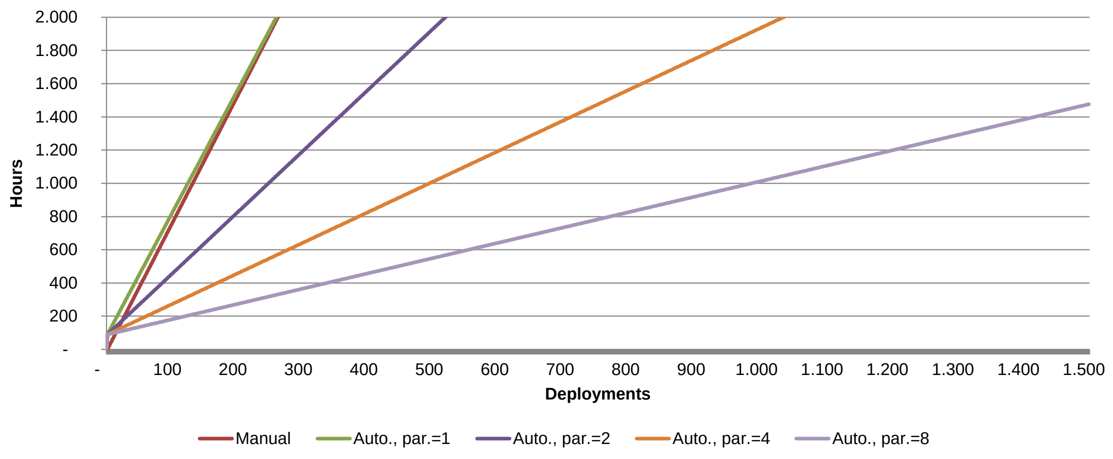


Figure 6.15: Manual deployment of SAP ERP compared to automated deployment.

Different statements are made about developer productivity. Brooks (1995, p. 237) indicates that productivity largely depends on the complexity of the development project. He reports between two and 28 lines of code (LOC) per day depending on project complexity. The assessment originated in the domain of operating system and compiler development. For developing the configuration models in the DSL, developer productivity of 20 LOC per day will be assumed in the following examples. This number is closer to the optimistic 28 because it is assumed that the DSL assists the developer in developing the models (Delaet et al., 2010).

Figure 6.15 shows a comparison of the manual deployment of SAP ERP compared with the automated deployment. A total of 225 LOC were necessary to formulate the configuration models for SAP ERP. If deployments are automated, without parallelization, manual and automated deployment break even at 324 deployments. With parallelization, the break-even point occurs before a hundred deployments in all calculated parallelization cases.

Figure 6.16 shows a comparison for the remote desktop service. Here, the LOC counted in all utilized models was 14.472. This includes the configuration models obtained from puppetforge.com. If all of these models would have to be developed from scratch the break-even point would be at 13.529 deployments with eight deployments being performed in parallel. The large majority of LOC comes from the externally obtained configuration models. These models do not focus on one function of the software they configure but support all functions. Also, they support various operating systems. When counting only the LOC that had to be added (264), the break-even point is reached quicker. Figure 6.17

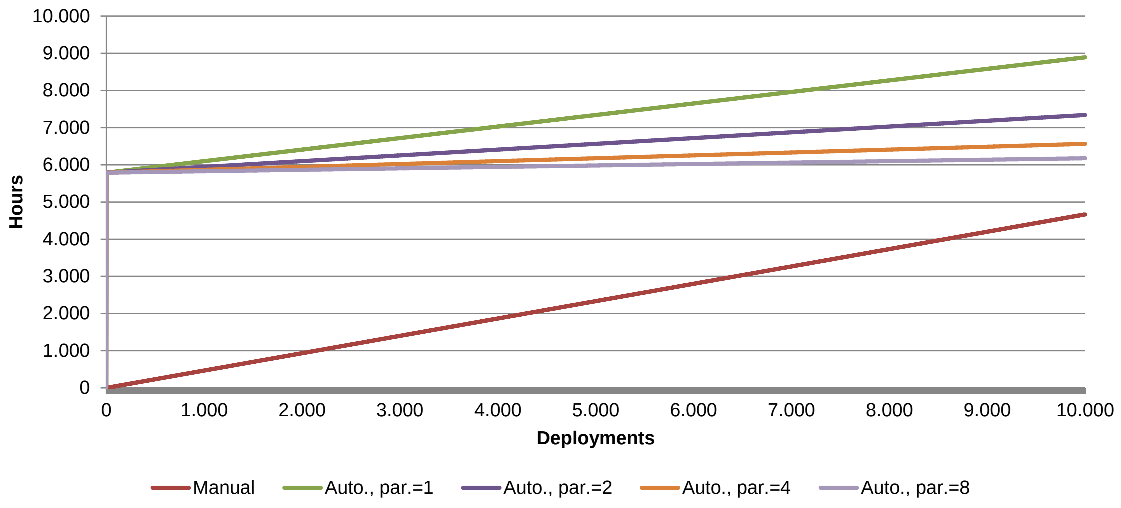


Figure 6.16: Manual deployment of the remote desktop compared to automated deployment (external development time added).

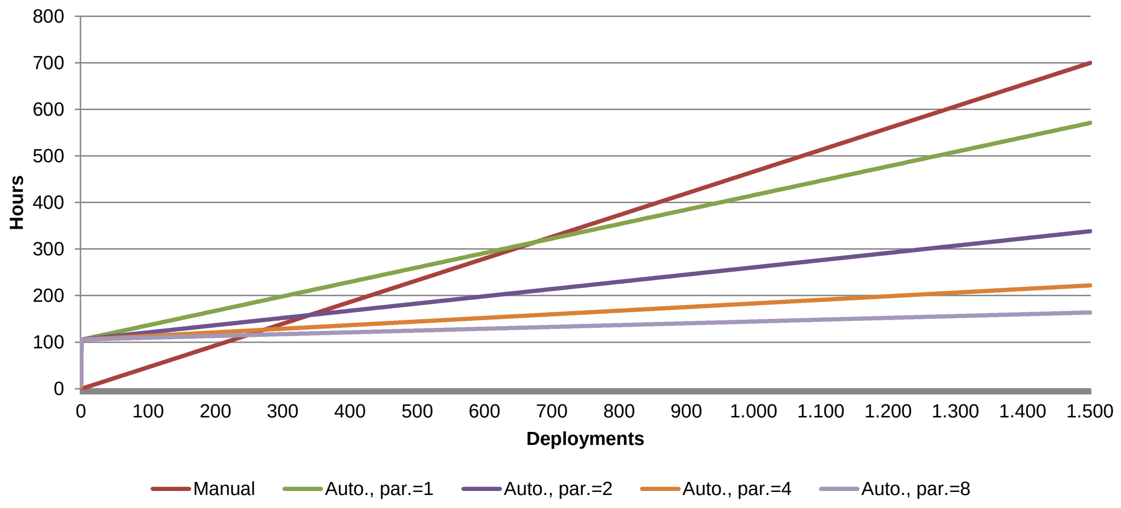


Figure 6.17: Manual deployment of the remote desktop compared to automated deployment.

shows this. Here a break-even point is reached at 670 deployments without parallelization. When using parallelization, this point is reached before 400 deployments for all modes of parallelization.

Automating deployments is not for free. If only small quantities of service instances are deployed over a service's lifetime, the automation might not be feasible. However, deployments may not only occur in the production environments. If each developer requires an approximation of the production landscape in order to implement new features or test more productively (Spinellis, 2012), the required quantity of deployments can quickly grow. Also, a service may not be deployed only once. New versions may repeatedly be deployed. In such cases, these models may also be used frequently, and the automation effort would be well invested.

Figure 6.18 shows the correlation between costs per deployment and number of de-

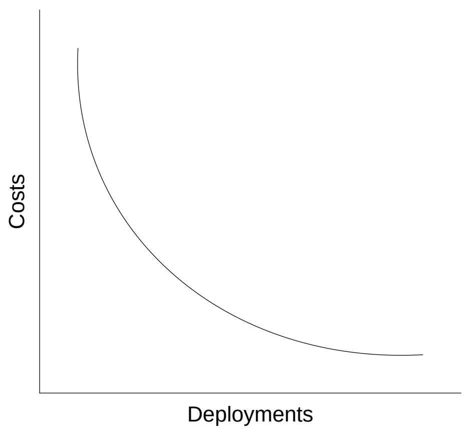


Figure 6.18: Costs per deployment decrease when the number of deployments increases.

ployments. It summarizes the results of this section's investigations. As the number of deployments increases, the costs per deployment decrease.

Whether or not automation is feasible depends on how often the business function needs to be performed. However, also greater reliability and inherent documentation are benefits of automating operations with model-based configuration management.

This section (section 6.5) presented a prototype used for evaluation. The next section maps the prototype's entities to those of the ISAA.

6.5.4 Mapping of prototype's entities to ISAA's entities

Tables 6.12 and 6.13 present a mapping between elements of the prototype and the ISAA. Prototype elements are, for example, a concrete application service such as *SAP ERP, standard*. The ISAA's equivalent would be the general *Application service* entity. On the one hand, *SAP ERP, standard* is part of Table 6.11 where the deployment times of different configurations of the three sample application services were shown. On the other hand, the *Application service* is a central concept and entity in the architecture description of the ISAA. Instead of the word entity, the AD element is used in alignment with ISO/IEC/IEEE (2011, p. 7). "An [architecture description] element [(ADE)] is any construct in an architecture description. AD elements are the most primitive constructs discussed in this International Standard [...]", (ISO/IEC/IEEE, 2011, p. 7). The occurrence columns show where the prototype element or the ADE were displayed.

The ADEs are displayed in various Figures and Tables and may be entities of the domain model diagrams, but are also full diagrams such as that for inquiry and order processing (cf. Figure 5.16). Whereas the *SAP ERP, standard* has a direct match with an ADE (*Application service*), the prototype element *Remote desktop, 2 small clients* has none. It is a parameterized BTO service. Although the parameterization is part of the models (cf. Figure 5.5), no specific ADE exists in the displayed diagrams for a parameterizable *Application service*, it is only described in the text. This is marked by an *x* in the column *S.*. Some limit had to be made as to which ADEs to display in diagrams and which not. Those that are not displayed could be easily added.

Prototype element	Occurrence	ADE	S.	Occurrence
ASL modeller	F. 6.7	Integrated development environment	x	F. 5.24
Computer (e.g., Operator / Developer computer)	F. 6.7	Instance		Fs. 5.8, 5.24, and A.2
Customer instance	F. 6.7	Instance		Fs. 5.8, 5.24, and A.2
Eclipse (with addons)	F. 6.7	Integrated development environment	x	F. 5.24
External repositories (e.g., Ubuntu package repository)	F. 6.7	Definitive media library		F. 5.24
Git client	F. 6.7	Integrated development environment	x	F. 5.24
GitLab CE	F. 6.7	Version control system		Fs. 5.10, 5.24, 5.25, and A.1
Hosts (e.g., virtualized repository host, virtualized ERP host)	F. 6.7	Instance		Fs. 5.8, 5.24, and A.2
KVM hypervisor	F. 6.7	IaaS master	x	Fs. 5.8, 5.10, 5.25, and A.2
OpenStack Ceilometer	F. 6.7	Usage tracking module		F. 5.24
OpenStack Glance	F. 6.7	IaaS master	x	Fs. 5.8, 5.10, 5.25, and A.2
OpenStack Heat	F. 6.7	Orchestration module		F. 5.24
OpenStack Horizon	F. 6.7	IaaS master	x	Fs. 5.8, 5.10, 5.25, and A.2
OpenStack Keystone	F. 6.7	IaaS master	x	Fs. 5.8, 5.10, 5.25, and A.2
OpenStack Neutron	F. 6.7	IaaS master	x	Fs. 5.8, 5.10, 5.25, and A.2
OpenStack Nova	F. 6.7	Infrastructure control module		F. 5.24
Physical OpenStack host	F. 6.7	Manufacturing equipment		Fs. 5.11 and 5.12 as well as T. 5.2
Production execution system (PES)	F. 6.7	Production execution system		Fs. 5.10, 5.24, and 5.25
Puppet agent	F. 6.7	Configuration management agent		Fs. 5.24, 5.25, and A.2
Puppet server	F. 6.7	Configuration management master		Fs. 5.10, 5.24, 5.25, and A.2
SAP Basis	F. 6.7	Customization module		F. 5.24
SAP ERP 6.00 GBI client, EHP7	F. 6.7	ERP system		Fs. 5.10, 5.24, and 5.25
SAP GUI	F. 6.7	Integrated development environment	x	F. 5.24
SAP MM	F. 6.7	Production and logistics module		Fs. 5.10, 5.24, and 5.25
SAP PP	F. 6.7	Production and logistics module		Fs. 5.10, 5.24, and 5.25
SAP SD	F. 6.7	Sales module		Fs. 5.10, 5.24, and 5.25
Virtual Box (with Vagrant)	F. 6.7	Integrated development environment	x	F. 5.24

Legend: S. - The prototype element is a subcomponent of this ADE and only described in text, F. - Figure, Fs. - Figures., and T. - Table.

Table 6.12: Mapping of elements of the application system landscape of the prototype to the elements of the ISAA (ADE).

This mapping explicates, representatively for all displayed prototype elements, to which ADEs they belong. In EVAL 2 (cf. section 6.2) such a mapping was shown in the object diagrams that are typed to the domain model entities. This mapping shows the applicability of the ISAA for the prototype and its' applicability for the sample services of EVAL 2.

In the next section, the ISAA is evaluated based on expert interviews.

Prototype element	Occurrence	ADE	S.	Occurrence
OC_REDMINE_LAMP	F. 6.9	Orchestration configuration model		Fs. 5.5, 5.6, 5.8, 5.25, A.1, and A.2
Ubuntu Server 14.04 LTS (Trusty Tahr) with Puppet	F. 6.9	Infrastructure configuration model		Fs. 5.5, 5.6, 5.8, 5.25, A.1, and A.2
wordpress	F. 6.9	Software configuration model		Fs. 5.5, 5.6, 5.7, 5.25, A.1, and A.2
Gather customer requirements	T. 6.8	User interaction activity		Fs. 5.1 and A.2
Inquiry and order processing	T. 6.8	Business process		Fs. 5.1, 5.16, and A.2
SAP-VA11	T. 6.8	Sales module	x	Fs. 5.10, 5.24, and 5.25
CPU_CORE_HOUR	F. 6.10	CPU hour		F. 5.12
REMDESKSERV_03	F. 6.10	Application service		Fs. 5.4, 5.5, 5.12, 5.31, and A.2
Thread starting billing activity	F. 6.8	System activity		F. 5.1
Remote desktop, 2 small clients	T. 6.11	Application service	x	Fs. 5.4, 5.5, 5.12, 5.31, and A.2
Remote desktop, 4 small clients	T. 6.11	Application service	x	Fs. 5.4, 5.5, 5.12, 5.31, and A.2
SAP ERP, standard	T. 6.11	Application service		Fs. 5.4, 5.5, 5.12, 5.31, and A.2

Legend: S. - The prototype element is a subcomponent of this ADE and only described in text, F. - Figure, Fs. - Figures., and T. - Table.

Table 6.13: Mapping of prototype elements to the elements of the ISAA (ADE).

6.6 Evaluation based on expert interviews (EVAL 3)

Three experts were interviewed to evaluate the ISAA with respect to its feasibility and utility (Hevner et al., 2004; Sonnenberg and vom Brocke, 2012). These experts were selected based on their expertise in different areas relevant to the ISAA. Their expertise spans the spectrum of educational backgrounds and occupation from management to information systems to computer science. Two of the respondents have 18 years and one 35 years of working experience. Table 6.14 provides an overview of the experts.

ID	Exp.	Education	Position	Occupation
1	18 y.	Master in Information systems	Department lead	Oversees development of the order-to-cash process of an internet service provider.
2	35 y.	Master in Management	Department lead	Oversees the consulting activities in statistics, finance and business intelligence of an industrial IT service provisioning.
3	18 y.	Master in Computer science	Department lead	Oversees the infrastructure management activities of an ASLP.

Exp.: Working experience in years

Table 6.14: Overview of experts, sorted by their interview ID.

In the following section, the interview design, including the posed questions, is presented.

6.6.1 Interview design

A presentation preceded all interviews via Skype. The presentations lasted about 45 minutes and presented the central concepts of the ISAA. The slides are provided in section C.4 in the appendix. After the presentations, the respondents were asked if any questions had arisen during the interview to clarify them. The interviewees were asked to not answer specifically for their current company, but for ASLPs instead. Subsequently, the interviews commenced. All interviews and presentations were conducted in German. The interviews were voice recorded, transcribed, and translated afterward. The respondents were provided with the interview transcripts for their consent for publication.

The posed questions are presented in section C.5. Questions are aligned with the architecture decisions presented throughout the chapters 4 and 5. The question catalog consists of three main blocks. An interview started with an opening question to introduce the respondent into the conversation. It was posed in a way to start the opening of the interview on a positive note (Kromrey and Strübing, 2009, p. 358). The first question block was concerned with the application services. Here, questions were posed with respect to the feasibility of the configuration models and the production process. The second block was concerned with the application system landscape of the ISAA. Here, it was inquired as to whether or not the design of the landscape was feasible. In the third question block, the interviewees were asked if they would implement the ISAA if they had the resources. Finally, they were asked if, in their view, the ISAA offers utility.

6.6.2 Results

When asked how the respondents assess the increasing standardization and automation of IT they all acknowledged its necessity. They emphasized that it is a means of keeping costs under control with increasingly scarcely available human resources. Respondent 3 said that it is a means of managing the increasing expectations and responsibilities that IT is confronted with.

Next, it was inquired if the reuse of configuration models between different application services would increase the overall efficiency. The respondents affirmed this. Again, respondent 3 added that the configuration models would need to be flexible enough in order to meet different scenarios. All operational tasks can be recorded in configuration models. Expenses incurred for development and maintenance can be directly allocated to these models. All respondents affirmed that this is helpful for determining the profitability of services.

The pre-engineering price assessment that is enabled by the application system landscape modeler (section 6.5.1.4) was considered helpful by the respondents. Respondent 3 added that it would be helpful particularly in scenarios where a high variation exists among application services. In his experience, making early price assessments is often tricky.

The respondents narrow the supportability of application services to those that can previously be modeled. Respondent 2 points to services that require individual customization. Such services could be handled in the ETO process case. However, respondent 3 adds that the more specialized the services become, the higher the required investment will be. This does not only apply to the services themselves, but also to the application system landscape to support the ISAA. This is also evident from the investigation of the feasibility of the automation in section 6.5.3. The more often a model is reused, the more profitable it will become. This reusability may diminish the more specific its intended use becomes. Models have to be deployed a specific number of times in paid services to become profitable. However, technically no concerns as to the implementability of different application services were raised.

With respect to the ISAA-supported service levels, the respondents agree that help desk response times and availability are the two most important indicators. However, they would add further indicators. Respondent 1 indicated that compliance and customer satisfaction would be necessary measurement criteria. However, their implementation would be out of scope for this thesis. Nonetheless, compliance assessments could be significantly eased by the available information in the configuration models (Talwar et al., 2005; Delaet et al., 2010). Respondent 3 adds that response time behavior and recovery times would be necessary indicators. Furthermore, he sees time to solution as helpful, but only very few providers are capable of providing a service level based on this measure. Agreed upon recovery times could just be added as constants to the existing materials in the ISAA. Help desk employees could sort incidents based on their urgency. Response time behavior depends on the technical performance of the services. Such an indicator could be included in a way that would deploy application system landscapes on hosts with different performance characteristics, for instance, the available network bandwidth per second.

Regarding the different payment methods, the respondents agreed that the two designated ones (fixed billing and pay-per-use) are the basic methods that need to be supported. Respondent 1 added that instant billing is a popular method. This could be realized with a different billing interval, for example, daily.

The respondents were then asked whether or not the two modes of service production, BTO, and ETO, were sufficient. They agreed that these are the two commonly practiced modes of production. Respondent 1 added that the ETO mode could be enhanced by integrating software as a service offerings from different providers, for example, engines for tax calculations. Such offerings could be integrated based on software configuration models that would wrap the offerings' APIs. Thus, they could make these services usable in ISAA-provisioned application system landscapes.

Finally, in the first question block, interviewees were asked if they deem the automated provision and operation of application services as envisaged in the ISAA to be practical and useful. The respondents affirmed this. Respondent 3 added that a high degree of standardization amongst the provisioned application services would be necessary. Furthermore, he sees the danger that particular technical requirements could surpass the capabilities of the production execution system. If such application services are requested, this could hinder profitability. Overall, however, an agreement was voiced.

The second question block inquired about the application system landscape that supports the ISAA. First, respondents were asked if the ERP system should be the leading system of the landscape. Also, it was asked whether or not storing the complete application service structure in the ERP system made sense. Here, respondents 2 and 3 agreed. Respondent 1 raised several concerns, but also some agreement. Referring to the proprietary ERP software, he said that the ASLP would make himself very much dependent on one team managing the ERP system. Also, the license model of the ERP software could negatively impact its profitable use. Although he values a single point of truth, in particular for controlling purposes, he suggests that such a concentration would hinder flexible adaptation to customer demands. He alternatively suggests to only aggregate information in the ERP system. The information would not have to be created there, but could instead be aggregated in the ERP system. He agreed with the necessity to store the structural service information in the ERP system in its entirety, also specifically, for controlling purposes. When performing mergers and acquisitions, an approach with high integration could be challenging to maintain. When starting a company, the proposed approach should be followed, he added. Respondent 3 also agreed that the approach would support controlling.

At the end of a contract, the service is terminated. The ISAA allows providing the customers with copies of the IaaS instances. The respondents were asked if this is sufficient. Respondent 1 referred to a practice going even further. His company required partners to provide them with the full source code and the configuration models of services they had outsourced to them. Respondent 2 said that, although not common practice, in his opinion the proposed practice was not sufficient. If he were a customer, he would require providers to hand out human-readable exports at the end of a contract. Such an end-of-contract procedure would go beyond the capabilities of standardization. Here, additional individual consulting services could be offered. Respondent 3 assessed the proposed provisioning of IaaS instance copies to be going beyond what is currently offered by providers. In his opinion, providers currently only provide the possibility, if at all, to download data via provided APIs.

The respondents supported the requirement that the infrastructure layer of the IT stack needs to be provided by IaaS software. The requirement is seen as satisfiable. Furthermore, the considered application software was assessed to be comprehensive.

In the last part of the interview, two final questions were asked. The first question was that if the respondents were executives of ASLPs and had sufficient means if they would implement the ISAA. Respondent 1 only partially agreed. He said that two schools of thought exist. One school would follow the direction suggested by the ISAA. The other, which he would prefer, would not have the ERP system as the leading system, but rather as an aggregating system. He added that the culture and organization would have to grow alongside the technological progress. Respondent 2 said that for standardized service he would implement the architecture. He said that its accounting capabilities would be beneficial. Respondent 3 affirmed that the enterprise management, the IT service management, and the IT service production system are the crucial systems for ASLPs. However, he said that it was a philosophical question which one would be the leading system.

When asked if the ISAA would increase the quality and efficiency of application system landscape creation, all respondents answered positively. Respondent 2 limited it to standardized services, as did respondent 3. Respondent 3 added that having the ERP system as the leading system would help with audits, documentation, and release processes. It would increase traceability. Also, he added, the architecture would be particularity valuable for standardized services.

In the next section, the results of the evaluation are discussed.

6.7 Discussion

Whereas for manufacturing standardized parts exist, those standardized building blocks for IT are not evident. In software engineering, libraries or conventional protocols can be reused. Nevertheless, a myriad of different software stacks exists on which application software is based. The first hypothesis proposes how this gap can be addressed for ASLPs.

Hypothesis 1
A dominant parts-based product design can be established for ASLPs. These designs are based on software for different operations automation approaches, such as configuration management software, container-based virtualization software, infrastructure as a service software, and orchestration software.

The presented configuration models introduce a wrapper. They encapsulate heterogeneous software stacks and expose defined interfaces. This enables the production execution system to deploy application services with various software stacks. In this regard, these configuration models are the equivalent of materials in manufacturing.

The configuration models formalize configuration information for the deployment and operation of the services (cf. REQ C.2). The materials in the material master correspond to those models. All employees can see an application service's components with access and authorization for the ERP system. An architect or salesperson can investigate the service's structure by actual production-relevant information. This increase in transparency could lead to a similar situation as in manufacturing where the ERP system represents a full and accurate view of the products of a company.

The proposed software stack does not impose new architectural paradigms. It can integrate with existing set-ups. Companies have to use infrastructure as a service software or procure respective resources if they are to adopt the architecture. This can be a problem for companies with a very heterogeneous or specialized set of hardware (cf. REQ R.3). However, the majority of applications should be covered by this approach.

Service levels are maintained as materials in the bill of materials that represent the composition of a particular application service. Contractual specifications such as duration and cost are maintained in contracts and sales orders (cf. REQ C.1).

Customizability, as well as standardization of the application service's construction, need to be achievable (REQ S.2). Modularization should be used (REQ S.3). Using modularization helps to achieve customizability based on a set of standard modules. The benefits of modularization become evident when the content management service only needs modification in the orchestration configuration model. Furthermore, customizability is achieved through the configurable BOMs. This way, for example, instance numbers or service level can be adapted to customer requirements.

In addition to mass-customization (build-to-order), engineer-to-order production is also supported in the ISAA (cf. REQ P.3). Motivated by the efficiencies of modern manufacturers that can build products from part libraries in computer-aided design software, the second hypothesis is formulated.

Hypothesis 2
ASLPs can produce their application services based on models and components, as well as automatically, much like physical products of modern manufactures.

Few components are required to set up a service (an instance, an image, and the configuration models). This makes development in production-like environments very easy. Employees responsible for development and operations can approximate application system landscapes in a production-like manner using desktop virtualization locally. These locally available application system landscapes can increase engineering productivity and quality because the engineers can have unrestricted access to production-like ASLs. They can rapidly and automatically deploy ASLs based on the configuration models on their personal computers and experiment with the ASLs without effects elsewhere.

The engineering activities are managed in the ERP system's project management system. The engineers can track their time with the ERP system's time tracking functionality. For the initial drafting of the application system landscape, they can be supported by an integrated modeling environment that is integrated with the ERP system. For such ETO projects, based on the models, early price estimates can be obtained quickly and accurately. Alpha, for instance, has the problem that their architects are not able to make price estimates for ASLs that are inquired for by their customers. Such estimates could be easily achieved by using the configuration models. The different configuration models are the elements that are aligned in solution designs (e.g., UML deployment diagrams) when designing customer ASLs. Prices can be attached to these configuration models in the ERP system. Price estimates can be made accurately. The interviewed experts welcomed such functionality. They were doubtful that highly individualized application services might be produced with profit in the suggested architecture. The architecture is designed for reuse of the configuration models. If the configuration models are too specific according to customer requirements, then this reuse may not be possible, the experts argued. In such cases, the deployment and configuration approach of the ISAA could be mixed with manual deployment instructions that are encoded in routings as suggested by Ebert (2009).

It was demonstrated that instantiations of the ISAA can support ASLs of different complexity and application software of different kinds (cf. REQ S.1). A desktop service, as well as a proprietary ERP service, could be deployed and operated. However, respondent 3 pointed out that deployment is only one step in the lifecycle of an ERP system. Nonetheless, everything that can be configured by using an API can be configured automatically. Any software that runs on the operating systems that are supported by the employed IT service production systems can be deployed. It is just a question of how complex the configuration models become. The boundaries between software development for operations automation and other business functions blur in this regard. Consider, for example, the business process customization within an SAP ERP system. As long as customization can be triggered through APIs that the configuration models can access, the system may be customized with the models. Respondent 3 added that flexible (e.g., parametrizable) configuration models would increase their versatility.

Furthermore, to configure complex application software more effectively and with less effort, specific configuration management software could be used. It would be possible to have an ITSP system with more than one configuration management master and different kinds of software configuration models. For instance, a specialized configuration master (e.g., SAP Solution Manager) could facilitate the configuration of a proprietary ERP system (e.g., SAP ERP) with less effort than a general purpose configuration management software. The effort of creating software configuration models for the specialized configuration management system could be lower due to pre-defined templates or specific DSLs provided by the software. Still, general purpose software could be used for the other application services. Such a mix of configuration management software would be possible, but the number of different configuration management software products should be kept low not to increase complexity to an unmanageable extent.

Profitability considerations should guide what to automate and what not. Companies must answer the question of how much effort they want to invest in creating the configuration models. Formalizing for automation is expensive as was discussed in section 6.5.3. Unprofitable models, however, should be readily identifiable with the presented architecture using analytical accounting and profitability analysis available in standard ERP systems. Furthermore, full automation and formalization reduce the chances of errors and the dependence on experts. In this regard, providing ASLP personnel with effective tool-sets is crucial to facilitate efficient creation of configuration models. Coping with the scarcity of human resources was a problem highlighted by all three interviews. Therefore, companies, even in specific areas with low repetition, should investigate the advantages of automation. A case like Gamma would not be as reliant on the expertise of few employees, for instance in cases of sickness, because their operation knowledge would be formalized in the configuration models. Also, in settings where changes are frequent, the proposed models may often be employed.

One cornerstone of process automation (cf. REQ P.1) in the ISAA are the configuration models that make application service descriptions operationalizable. The other cornerstone is the application system landscape. Respondents indicated that maintaining such a landscape in practice is expensive. The set-up of the prototype and SAP ERP system, mainly regarding its customization, was very time-consuming. SAP ERP system customization, in accordance with SAP's configuration templates (cf. section 6.5.1.1), was mainly carried out by a student assistant for one year and five months with a forty hour per month employment contract. The decoupling of the PES from the ERP system (cf. AD 9) proofed sensible in this regard. Its functionality could be implemented separately and with fewer integration-constraints that needed to be considered. The automation, in the prototype, was limited to the time onwards from which the production order is released to commence deployment or realize customer ASL changes (cf. REQ.4). ERP-based process automation was not realized as this is not new and could be implemented (Cardoso et al., 2004).

Existing functional integration of the ERP system's modules (cf. REQ R.2) can be utilized. For instance, human resource management is integrated with project management. This integration is necessary, for example, to be able to staff development projects according to the availability of the developers. ERP functionality is also available in areas such as procurement or for liquidity management procedures such as dunning. Requirement REQ R.2 is met if standard software is employed which provides the required functionality. Using the ERP system as a central point for initiating all service creation is seen critically by the experts. This could slow down the ASLP's reaction speed to new market demands. However, the resulting single point of truth characteristic is seen positively, in particular for controlling purposes. When implementing the ISAA, a balance between integration effort and agility has to be found. Success factors for enterprise systems adoption (cf. section 3.2.2) need to be considered when implementing the ISAA. Top management support needs to be considered when implementing and customizing the proposed application systems. Also, the enforcement of using a limited set of operations automation approaches likely will require a substantial amount of training as well as the mentioned top management support.

The ISAA is particularly relevant for companies that have a broad portfolio of diverse services built on a set of standard modules (cf. REQ R.1). It requires investment to create or source the necessary configuration models as well as the needed application system landscape. If a service is only sold a few times, this investment may not pay off. However, companies that must set up a multitude of application system landscapes for their customers can profit from the information system architecture. For instance, it can decrease the time-to-market for landscapes based on new application software. Also, DevOps-employing companies or those that deploy microservices may benefit from the provisions of the ISAA. In this regard, it was shown that containers could be handled by the architecture as well as multi-tenant application services.

Relevant related work focuses on only on different operations automation approaches (cf. sections 3.4 and 4.3) or proposes only a high-level architecture (cf. section 3.1.3). In contrast, the ISAA exemplifies comprehensively on all relevant layers how an organizationally aligned operations automation approach can be implemented. The measurement principle (cf. CAMS principles for DevOps), in particular regarding business success, can be followed by ISAA-implementing organizations (cf. REQ P.2). The configuration models provide a consistent way to describe the provisions necessary for operating complex application system landscapes. Implementing the architecture is a question of capability both in maintaining the infrastructure and being able to develop services based on the three kinds of models fully.

The manufacturing analogy has been employed throughout the thesis (cf. REQ R.4). The analogy notably guided the domain model mapping to the ERP master data. Here, the prototype could be implemented without non-standard customization required. To implement the continuous nature of a service, custom fields in the contract had to be used to differentiate between different billing types. Also, the PES was necessary in order to bill regularly (cf. REQ P.5).

Based on this discussion, it is concluded that Hypotheses 1 and 2 are valid. The research goal of the thesis was to increase the efficiency of the ASLP's application service production.

Research goal
The research goal of this thesis is to increase the efficiency of ASLPs' application service production through standardization, automation, and modularization by creating the ISAA.

That goal was reached. The efficiency of application production is increased compared to manual deployments or deployments that are based on heterogeneous automation techniques due to the proposed standardization, automation, and modularization. This could be shown in EVAL 3, where automated and manual deployments were compared. The utility of the ISAA, in particular regarding efficiency and quality, was confirmed by the respondents of the expert interviews.

The next chapter will conclude this thesis.

7 Conclusion

This thesis will be concluded in three parts. First, a summary is provided. Secondly, the research contributions are highlighted. Moreover and finally, an outlook on future work is provided.

7.1 Summary

The research stream of industrialization of information technology (IT) studies how industrialization principles, which have been successfully applied in traditional industries, can achieve utility in IT service provisioning. In contrast to IT services, the composition of physical products is more straight-forward. This thesis has made the argument that a standard concept in IT, describing what the raw materials, the semi-finished goods, or even the products (finished goods) are, is not self-evident. Identifying such a concept would help to manage IT service provisioning similarly to how it is done in manufacturing. Aiming at higher efficiency, comprehensive software support for the IT service production process is proposed by this thesis. As in manufacturing, the application system landscape could be led by an enterprise resource planning (ERP) system.

Whereas previous literature has addressed the industrialized IT service providers as a whole, this thesis embarked upon a different path.

The scope was narrowed to a specific type of IT service provider. It was exemplified that the ERP concept can comprehensively be implemented to achieve utility for IT service providers. Companies which offer application services to their customers are the chosen IT service provider type. This thesis referred to these companies as application system landscape providers (ASLP). ASLPs focus on the production of application system landscapes. They provide landscapes of different application systems to their customers. Unlike traditional application service providers, their focus lies on the design, deployment, and operation of complex application system landscapes for their customers. This focus allowed a concentration on the specific necessities of ASLPs. Large infrastructure as a service providers (e.g., telecommunication companies) could use the architecture to forward-integrate application service production. Particularly, small and medium-sized businesses that have difficulty in operating their growing IT landscapes in the cloud (Hsu et al., 2014) might profit from such offerings. For quickly testing complex landscapes and assessing time-to-market the ISAA could be of value. Also, large internal IT service providers that develop their companies' application services in a microservice approach might profit from the ISAA.

To be able to treat application system landscapes that underlie application services in a parts-based manner, they have to be described in a specific form. Service design was addressed by Hypothesis 1. It stated that dominant parts-based product design could be established for ASLPs. Such designs are based on software for different approaches of automating application service operations. Hypothesis 2 assumed that ASLPs could automatically produce their application services based on models and components, much like physical products of modern manufactures.

Two scenarios were differentiated in this regard. Build-to-order (BTO) scenarios apply to situations where application services constitute low variability. Following a mass customization approach, application services in BTO scenarios can be parametrized to fit with customer requirements. The thesis proposed an engineer-to-order (ETO) approach where customer requirements are transformed into application system landscapes to satisfy more specific customer requests.

The overall goal of the thesis was to increase the efficiency of ASLPs' application service production through standardization, automation, and modularization. To achieve this research goal and to test Hypothesis 1 and 2, an information system architecture for ASLPs (ISAA) was proposed. Before constructing the ISAA in a design science aligned manner, three research questions had to be answered:

(1) *How do application system landscapes of IT service providers look like today?* Based on case study results, application system landscapes of IT service providers are composed of three application systems grouped in three main categories: enterprise management, IT service management, and IT service production systems. Application systems from these categories form the basis of the ISAA's application system landscape.

(2) *What are the requirements that application system landscape providers have regarding an information system architecture that supports their application service production process?* 14 requirements were derived from the analysis of three ASLP cases as well as literature. These requirements pertained to the overall relevance of the ISAA, complete service description, specificities of application services, and production manageability.

(3) *What operations automation approach is suitable for constructing the information system architecture?* Different operations automation approaches were analyzed to construct the ISAA. Model-based configuration management was selected because it represents all layers of the IT stack and imposes low requirements for architectural changes of the application software that it is managed with.

The thesis proposes that an application service is composed of three configuration models (cf. section 5.1.2). These are software configuration, infrastructure configuration, and orchestration configuration models. Software configuration models specify what software packages are installed and how software is configured on a specific instance. Infrastructure configuration models define instances. Orchestration configuration models define which software configuration is applied to which instances. The relationship between the configuration models and other entities of the ISAA was described in a domain model that cross-cuts the different architecture layers. In addition to the business model, a production process was presented describing how application services are produced in the ISAA: from inquiry and order processing over deployment and billing to termination. This process is supported by an application system landscape that features a production execution system. The production execution system orchestrates the different systems. This orchestration causes application services that are sent with production orders from the ERP system to be deployed on machines of an infrastructure as a service system.

The evaluation was conducted in a systematic stepwise approach (Sonnenberg and vom Brocke, 2012). Three steps were conducted. First, the relevance of the research was justified by highlighting the research gap. In particular, three sets of related work are relevant for this thesis (cf. section 3.5.2). Orchestration software, industrial methods for IT service production, and the recently published IT4IT reference architecture. IT4IT and the industrial methods for IT service production have a similar approach and facilitate organizationally integrated IT service production. IT4IT is more aligned with the Information Technology Infrastructure Library's IT service management. The authors of the industrial methods for IT service production related work set suggested adopting ERP systems for manufacturers (cf. Figure 4.2; Hintsch et al., 2016b). However, both only consider limited technical detail. The ISAA considers IT service production systems in the organizational context including sufficient technical detail to deploy and operate application services. The selected model-based configuration management approach provides enough versatility not to limit application software to a certain technology stack. IT4IT, as a recent international standard, shows the relevance of the ISAA's addressed topic. In the future work section, a possible alignment between the ISAA and IT4IT will be outlined. From the perspective of orchestration software, the ISAA provides an organizational framework in which development and operation of application software can take place.

Secondly, the artifact description, in this case, the architecture description of the ISAA, was validated. It was validated by using the provisions suggested (ISO/IEC/IEEE, 2011). It transparently detailed how the architecture decisions were made by illustrating the rationale behind each decision with case study-derived requirements. Finally, based on a prototype the feasibility of the architecture was shown. Expert interviews were used to validate that the architecture can be of utility to ASLPs. In the evaluation, it was concluded that Hypotheses 1 and 2 are valid. The three configuration models can be used to compose application services like parts in physical manufacturing (Hypothesis 1). Application system landscape production, following the ISAA, can resemble the production of modern just-in-time and ETO manufacturers. Customers can place their orders, sales personnel, possibly assisted by architects, make first application system landscape drafts. Development may be necessary for individual requests, but mostly available modules can be used, similar to part libraries in computer-aided manufacturing (Harrison and van Hoek, 2008). The application service is then deployed, and its operation starts. Continuous billing commences, all with a high degree of automation.

The research goal was reached by showing that the ISAA can increase the efficiency of application products. It uses a standardized way of describing application services, automated processes, and a modularized product portfolio, which is mirrored technically by the configuration models.

The next section discusses the contribution of this thesis.

7.2 Contribution

Figure 7.1 provides an overview of the artifact of this thesis: the ISAA. The figure's left side shows a production-based view of the ISAA. Various input factors are transformed into output factors, such as customer requirements and software. Output factors for ASLPs are application services. In contrast to manufacturers, for IT service providers no factory exists as a production factor.

Their information system can be considered to include the ASLP's factory. The information system and its architecture that includes the workforce, processes, and the application system landscape are the primary focus of this thesis.

The right side of Figure 7.1 features a T-view of the ISAA. On the upper side, it shows the spectrum from strategic over tactical to operational considerations. The ISAA is located closer to operational than to strategic considerations. Operational considerations include how to deploy application services manually. Strategic considerations include what types of services can be deployed with the ISAA. Most considerations pertain to tactical considerations. The focus of the ISAA lies on the domain model, the production execution system and the application service configuration.

The thesis makes the following contributions. They correspond to the focus displayed in Figure 7.1.

1. An information system architecture that explores the limits of standardization, automation, and modularization for application system landscape production.

2. A domain model that explicates the relationships between relevant entities of application system landscape production on the layers of business, process, integration, software, and infrastructure.

3. A prescription of how to compose application system landscapes based on three kinds of configuration models that can be used analogously to materials in physical product fabrication: software, infrastructure, and orchestration configuration models.

4. A production execution system that orchestrates the production of application services between the enterprise management, IT service management, and IT service production systems.

This research has implications for practitioners and researchers. By conducting a comparison, practitioners may use the architecture to evaluate their information system architecture while seeking to increase efficiency as well as service quality. For researchers, the architecture contributes to the existing stream of analogy-employing research, in particular within the area on the industrialization of IT research. This research contributes

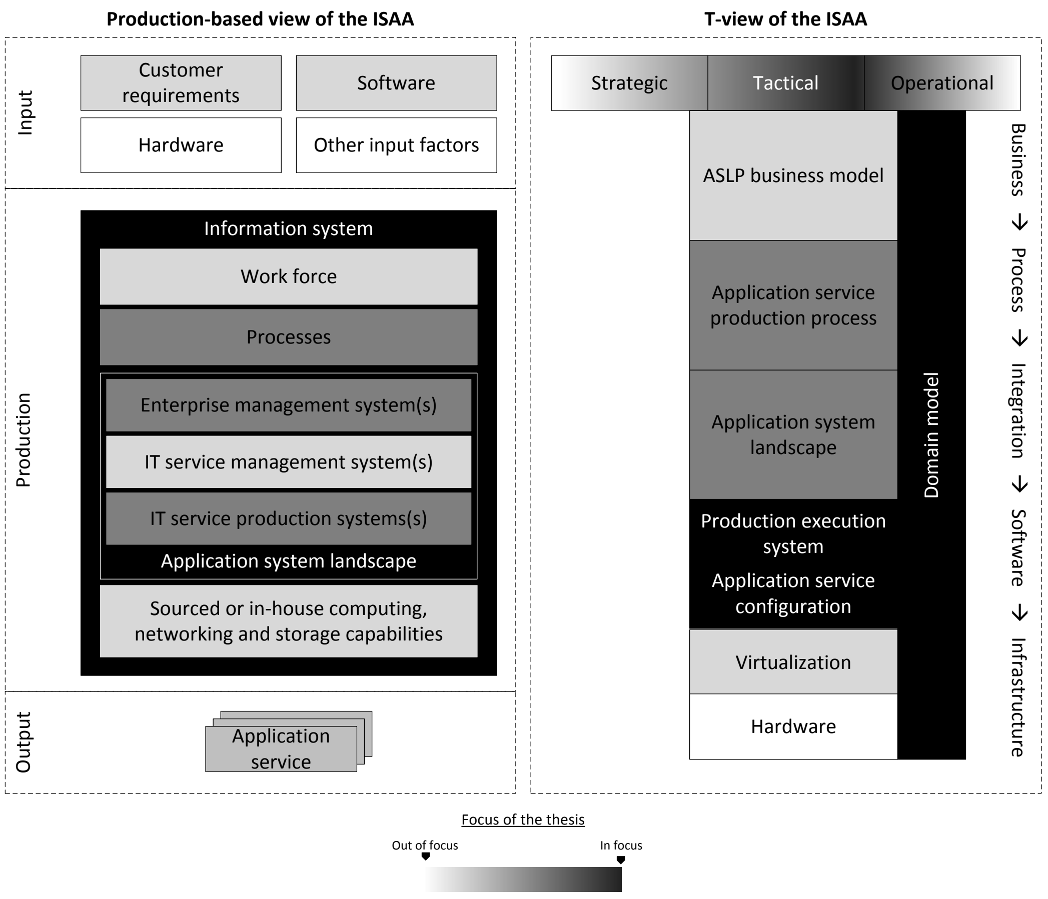

Production-based view of the ISAA
Input
Customer requirements
Software
Hardware
Other input factors
Production
Information system
Work force
Processes
Enterprise management system(s)
IT service management system(s)
IT service production systems(s)
Application system landscape
Sourced or in-house computing, networking and storage capabilities
Output
Application service
T-view of the ISAA
Strategic
Tactical
Operational
ASLP business model
Application service production process
Application system landscape
Production execution system
Application service configuration
Virtualization
Hardware
Domain model
Business → Process → Integration → Software → Infrastructure
Focus of the thesis
Out of focus
In focus

Figure 7.1: Parts of the information system architecture that are in this thesis' focus.

by concretely and comprehensively testing the limits of transferring automation and standardization concepts, in particular, ERP and computer-integrated manufacturing, from fully automated, mass-customized physical goods production to IT service production by employing the particular case of ASLPs. Furthermore, this research adds theoretical understanding to the relatively seldom investigated IT service production domain (Hess et al., 2012).

7.3 Future work

The costs for automation, based on configuration models, were discussed in the evaluation. However, costs would also be incurred when first implementing the ISAA. For instance, costs for licenses, customization and development, and training of staff would likely be incurred. A preliminary analysis of the total cost of ownership for the ISAA would be valuable. Such an analysis could help companies that would be interested in adopting the ISAA in their assessment. A total cost of ownership analysis could help them conduct a cost-benefit analysis. An analysis of the total cost of ownership could contrast different types of ERP software (e.g., proprietary against open source) or IaaS set-ups (e.g., sourced or in-house operated).

The evaluation included steps EVAL 1 - EVAL 3 of the evaluation approach for design science research by Sonnenberg and vom Brocke (2012). EVAL 4, the validation of an instance of the ISAA in a naturalistic setting (Sonnenberg and vom Brocke, 2012), could be another path of future work. If developing the ISAA further and attempting to transfer it to a reference model such validation would be necessary. Considerations for such an implementation would include thoughts on whether the architecture should be implemented in a big-bang or incremental approach (Davenport, 2000, p. 173-181). The ISAA could support the incremental approach with recommendations as to which layers, components, and prescriptions of the ISAA could be implemented together as well as proposing an order.

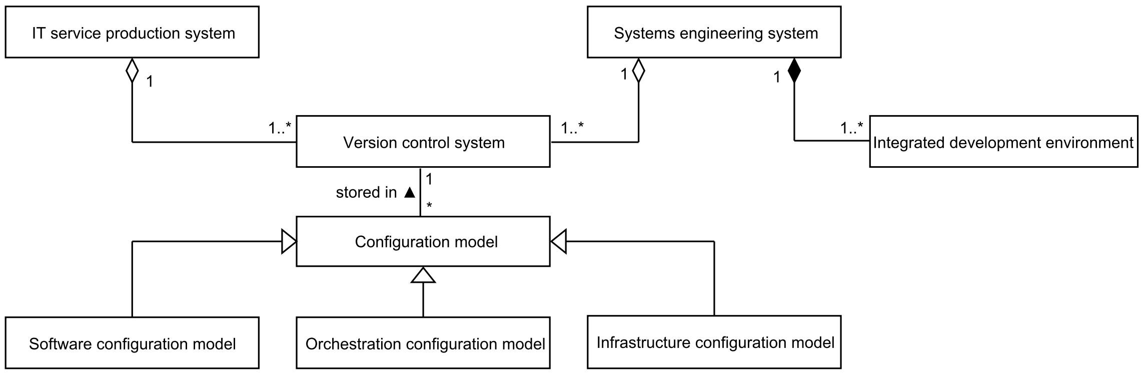
Companies might further be motivated to consider the ISAA if it would align with a standard such as IT4IT. Similarities to IT4IT have already been shown (Hintsch et al., 2016b) based on IT4IT's antecedents (Betz, 2011). For instance, the IT service production system that is not part of the IT4IT reference architecture could be integrated. Also, the presented application service production process could be defined as an IT4IT scenario (cf. section sec:rb-itsm-frameworks). Whereas the IT4IT is vendor agnostic and only defines functional components, the ISAA categorizes the application system landscape in software packages that are available on the market. An alignment seems promising.

Within the ISAA, application services that are based on existing software may be provisioned, but new application software development is also addressed. In software engineering, model-driven development has been used for more than a decade. Deriving or integrating models for operation from or with models for development could yield efficiency and effectiveness improvements. An approach that seems natural is to use deployment or component diagrams of the unified modelling language (UML) for such a purpose, but other modeling languages might also be viable options. Future work could investigate if combining or integrating development with operations models is feasible.

Finally, the ISAA advocates the use of ERP software that was initially developed for manufactures. Hybrid value creation has been practiced for some time (Velamuri et al., 2011). Physical products are sold with value-adding services. With the proliferation of the Internet of Things or Industry 4.0, these types of offerings will likely gain market share. The ISAA could be used to support traditional physical goods production, on the one hand, and application service provisioning, on the other hand. If for instance, cars are ordered in a mass customization approach, application services that are operated on servers controlled by the manufacturer could be deployed as well. This hybrid use of an ERP system would avoid system proliferation and integrate value creation in a single ERP system capable of handling both the production of physical products and application services.

The ISAA promotes standardization, automation, and modularization. Following these paths of future work, quality and efficiency of IT service production could be increased within the domain of ASLPs and beyond.

A Domain model of the ISAA


IT service production system
Systems engineering system
Version control system
Integrated development environment
Configuration model
Software configuration model
Orchestration configuration model
Infrastructure configuration model
stored in ▲
1
1..*
1
1..*
1
1..*
1
*
Figure A.1: The version control system in its relationship with the systems for ITSP and systems engineering.

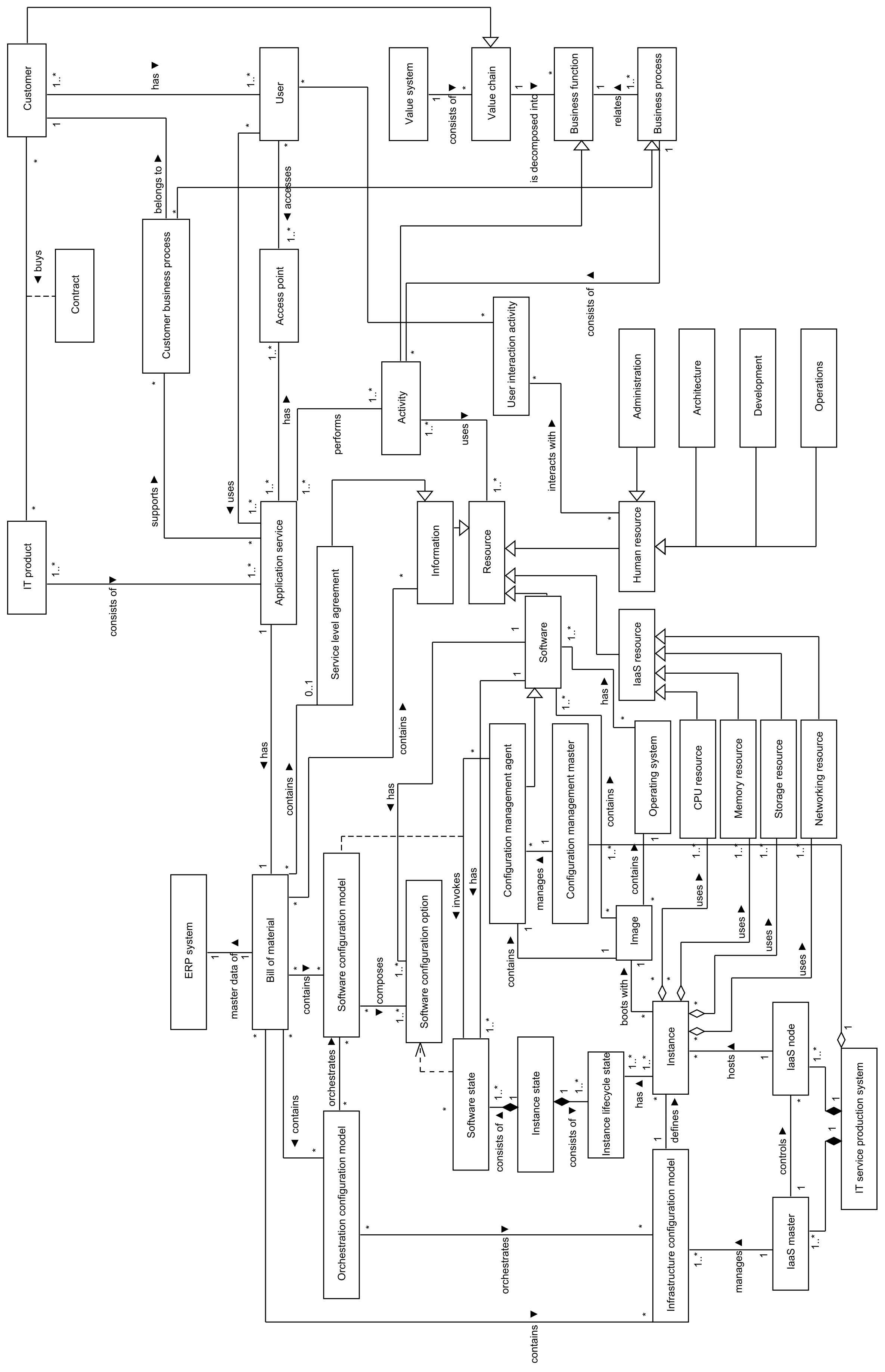


Figure A.2: View of domain model that shows entities of all layers as well as some of their relationships (in order to retain readability, not all entities or relationships could be included).

B Supplementary material for case study

B.1 Overview of cases

Case statistics				**Interview length**		
Size	*Cases*		*Cases*		*Time^{min}*	*Transcripts^{wds}*
Small	9	ERP	24	Min	25	3,380
Medium	4	No ERP	1	Mdn	37	5,649
Large	12			Avg	45	5,667
		IT service provider	17	Max	82	12,232
Total	**25**	Vendor	8	**Total**	**1,121**	**159,429**
						min: in minutes, wds: in words

Table B.1: Quantitative overview of the cases and collected data.

ID	Size[1]	Description	Data sources[2]
C1	small	IT service provider	I,O
C2	small	Software company	I,O
C3	large	ERP vendor	I,O
C4	small	ERP vendor	I,O
C5	small	ERP vendor	I,O
C6	medium	ERP vendor	I,O
C7	large	ERP vendor	I,O
C8	large	ERP vendor	I,O
C9	small	ERP vendor	I,O
C10	small	SaaS ERP provider	I,O
C11	medium	IT service provider	I,O
C12	small	IT service provider	I,O,W
C13	large	IT service provider (internal & external)	I,O
C14	large	IT service provider (internal & external)	I,O,W
C15	medium	IT service provider	I,O
C16	large	IT service provider (internal)	I,O
C17	medium	IT service provider for precision farming	I,O
C18	large	IT service provider (internal & external)	I,O
C19	small	IT service provider	I,O
C20	small	IT service provider	I,O
C21	large	ERP vendor	I,O
C22	large	IT company (hardware, software, services)	I,O,W
C23	large	IT service provider	I,O
C24	large	Telecommunications and Internet company	I,O
C25	large	Knowledge intensive company - internal IT service provider	I,O

1: based on employee size European Commission (2005, p. 14)

2: I (interview transcript), O (information available online), W (workshop documentation)

Table B.2: Full case overview.

	Identifiers in different publications of Hintsch			
ID	**2015b**	**2016b**	**2018**	**Thesis alias**
C1	Case A	Alpha		
C2	*unidentified*[1]			
C3	*unidentified*[1]			
C4	*unidentified*[1]			
C5	*unidentified*[1]			
C6	Case C		Beta	Beta
C7	*unidentified*[1]			
C8	*unidentified*[1]			
C9	*unidentified*[1]			
C10	*unidentified*[1]			
C11	*unidentified*[1]	Gamma		
C12			Gamma	Gamma
C13	*unidentified*[1]			
C14	*unidentified*[1]	Iota	Alpha	Alpha
C15	*unidentified*[1]			
C16	*unidentified*[1]	Zeta		
C17	*unidentified*[1]			
C18	Case F			
C19	*unidentified*[1]			
C20	Case B	Beta		
C21	Case G			
C22	*unidentified*[1]			
C23	Case E	Epsilon		
C24	Case D	Delta		
C25		Theta		

1: data from this case, although unidentified, was used to compile a list of which business functions were supported by application systems

Table B.3: Mapping of cases with previous publications.

B.2 Question catalogue

1. General company background, including key figures and business models
2. Type and architecture of the employed application systems
    a) Use of application systems?
        i. Integrated?
        ii. Which manufacturer? Or is it custom software?
    b) Which departments involved? Which ERP modules are used?
    c) Total number of users?
    d) Architecture?
3. Internal and external product and service portfolio representation
    a) What is the structure of the respective sales product?
    b) How are the products represented in the ERP system?
    c) Are there any basic products / modules that are used multiple times?
    d) Are service level agreements maintained? If so, how are you represented?
    e) Do you produce the product entirely by yourself?
    f) How are services and products billed?
4. Application system support of core production processes
    a) Which core processes do you cover with the ERP system?
        i. Planning
        ii. Procurement
        iii. Manufacture
            A. Do you use a different type of IT support in the production process?
            B. Can you visualize dependencies between the specification and the production of services (e. g. hard disk capacity / CPU capacity for new products considering the services currently in production)?
            C. Can you simulate production alternatives in the specification beforehand?
            D. Project planning of development projects?
        iv. Delivery / Performance
            A. Provision and billing orchestrated by workflow control?
            B. Is it possible to initiate deployment from the ERP system?
        v. Return
    b) Key figures / metrics

      i. Can you make statements about the profitability of individual products?
      ii. Do you have direct indicators for the service phases?
   c) Do you use a process framework for the design of service processes (ITIL, SCOR,...)?
      i. If no, why not?
      ii. If yes, which parts of it are implemented?

5. Representation of the organization in the application systems
   a) Representation of the organization in the general information system?
   b) Are there different roles or competence profiles that can be assigned to specific activities?

C Supplementary material for EVAL3

C.1 Different variations and phases of the production process

Business process	**Activity**	**System Context**	**Activity type**
Operation	[Service request]		
	Cancel service with repair order	SAP-IW51, SAP-VA01	UIA
	Update contract, create a new configuration of the service with six clients	SAP-VA42	UIA
	Create deployment sales order	SAP-VA01	UIA
Deployment	Create production order	SAP-CO01	UIA
	Release production order	SAP-CO02	UIA
	Service deployment	PES	SYS
	[Production order confirmation]	SAP ERP	UIA
Operation	{*Service operating*}	ITSP system	SYS
	Notify customer	SAP-VL01N	UIA
Billing	Bill customer	SAP-VF01	UIA

	Legend
Activity	regular system or user interaction activity, [event], continuous system activity
Application system context	SAP- precedes SAP ERP transactions: VA1[1/2] - Inquiry management (create/change), VA4[1/2] - Contract management (create/change), VA0[1/2] - Sales order management (create/change), VL0[1/2]N - Outbound delivery management (create/change), VF0[1/2] - Billing document management (create/change), CAT2 - Maintaining timesheet data, CO0[1/2] - Production order management (create/change), MM0[1/2] - Material management (create/change), CS0[1/2] - Bill of materials management (create/change), IW5[1/2] - service notification management (create/change)
Activity type	UIA - user interaction activity, SYS - system activity

Table C.1: Changing a service parameter in the operation phase.

Business process	**Activity**	**System Context**	**Activity type**
Inquiry and order processing	[Incoming customer inquiry]		
	Gather customer requirements	SAP-VA11	UIA
	Create contract	SAP-VA41	UIA
Engineering	Create ETO sales order	SAP-VA01	UIA
	Create solution design	SAP-CAT2, ASL modeller	UIA
	Develop orchestration configuration model	SAP-CAT2, ISAA DevOps stack	UIA
	Update material master	SAP-CAT2, MM01, CS01	UIA
	Perform acceptance test	SAP-CAT2, ISAA DevOps stack, ITSP system	UIA, SYS
	Update contract and ETO sales order	SAP-VA42, SAP-VA02	UIA
Deployment	Create production order	SAP-CO01	UIA
	Release production order	SAP-CO02	UIA
	Service deployment	PES	SYS
	[Production order confirmation]	SAP ERP	
Operation	{*Service operating*}	ITSP system	SYS
Deployment	Notify customer	SAP-VL01N	UIA
Billing	Bill customer	SAP-VF01	UIA
	{*Service billing*}	PES	SYS
Termination	Service termination	PES	SYS

	Legend
Activity	regular system or user interaction activity, [event], continuous system activity
Application system context	SAP- precedes SAP ERP transactions: VA1[1/2] - Inquiry management (create/change), VA4[1/2] - Contract management (create/change), VA0[1/2] - Sales order management (create/change), VL0[1/2]N - Outbound delivery management (create/change), VF0[1/2] - Billing document management (create/change), CAT2 - Maintaining timesheet data, CO0[1/2] - Production order management (create/change), MM0[1/2] - Material management (create/change), CS0[1/2] - Bill of materials management (create/change), IW5[1/2] - service notification management (create/change)
Activity type	UIA - user interaction activity, SYS - system activity

Table C.2: Process steps for deploying an ETO service with a new software configuration model.

Business process	**Activity**	**System Context**	**Activity type**
Operation	[Service request]		
	Change software configuration model	SAP-CAT2, ISAA DevOps stack	UIA
	Test changes	SAP-CAT2, ISAA DevOps stack, ITSP system	UIA, SYS
	Update material master	SAP-CAT2, MM02	
	Cancel service with repair order	SAP-IW51, SAP-VA01	UIA
	Update contract	SAP-VA42	UIA
	Create deployment sales order	SAP-VA01	UIA
Deployment	Create production order	SAP-CO01	UIA
	...		

	Legend
Activity	regular system or user interaction activity, [event], continuous system activity
Application system context	SAP- precedes SAP ERP transactions: VA1[1/2] - Inquiry management (create/change), VA4[1/2] - Contract management (create/change), VA0[1/2] - Sales order management (create/change), VL0[1/2]N - Outbound delivery management (create/change), VF0[1/2] - Billing document management (create/change), CAT2 - Maintaining timesheet data, CO0[1/2] - Production order management (create/change), MM0[1/2] - Material management (create/change), CS0[1/2] - Bill of materials management (create/change), IW5[1/2] - service notification management (create/change)
Activity type	UIA - user interaction activity, SYS - system activity

Table C.3: Changing a software configuration model for a customer in the operation phase.

Business process	Activity	System Context	Activity type
Operation	[Incident]		
	Change software configuration model	SAP-CAT2, ISAA DevOps stack	UIA
	Test changes	SAP-CAT2, ISAA DevOps stack, ITSP system	UIA, SYS
	Update material master	SAP-CAT2, MM02	UIA
	Full regression testing	SAP-CAT2, ISAA DevOps stack, ITSP system	UIA, SYS
	Customer-wide deployment	SAP-CAT2, ITSP system	UIA, SYS
	Notify customers	E-Mail	UIA

Legend

Activity	regular system or user interaction activity, [event], continuous system activity
Application system context	SAP- precedes SAP ERP transactions: VA1[1/2] - Inquiry management (create/change), VA4[1/2] - Contract management (create/change), VA0[1/2] - Sales order management (create/change), VL0[1/2]N - Outbound delivery management (create/change), VF0[1/2] - Billing document management (create/change), CAT2 - Maintaining timesheet data, CO0[1/2] - Production order management (create/change), MM0[1/2] - Material management (create/change), CS0[1/2] - Bill of materials management (create/change), IW5[1/2] - service notification management (create/change)
Activity type	UIA - user interaction activity, SYS - system activity

Table C.4: Fixing a bug.

Business process	Activity	System Context	Activity type
Termination	[Customer contract expires in n days]		
	Contact customer regarding extension	E-Mail	UIA
	Extend contract	SAP-VA42	UIA

Legend

Activity	regular system or user interaction activity, [event], continuous system activity
Application system context	SAP- precedes SAP ERP transactions: VA1[1/2] - Inquiry management (create/change), VA4[1/2] - Contract management (create/change), VA0[1/2] - Sales order management (create/change), VL0[1/2]N - Outbound delivery management (create/change), VF0[1/2] - Billing document management (create/change), CAT2 - Maintaining timesheet data, CO0[1/2] - Production order management (create/change), MM0[1/2] - Material management (create/change), CS0[1/2] - Bill of materials management (create/change), IW5[1/2] - service notification management (create/change)
Activity type	UIA - user interaction activity, SYS - system activity

Table C.5: Extending the contract of a service.

C.2 Manual deployments

i	Context	Command	p_i	e_i	w_i
1	Horizon	[spawn one m1.large instance named 'guaca-server' of 'Ubuntu Server 14.04 LTS (Trusty Tahr)']	0:47	0:11	0:02
2	Horizon	[associate floating ips for server and one client]	0:06	0:06	0:01
3	Horizon	[spawn three m1.small instances named 'guaca-client' of 'Windows 2012 Server Evaluation - IMAGE v6']	0:32	0:12	0:01
4	Horizon	[associate floating ip one client]	0:07	0:04	0:01
5	Horizon	[associate floating ip one client]	0:07	0:04	0:01
6	Horizon	[associate floating ip one client]	0:07	0:04	0:01
7	WSsh@oc	[ssh to server (rs) guaca-server]	0:06	0:15	0:02
8	Bash@rs	sudo apt-get update	0:07	0:10	0:01
9	Bash@rs	sudo apt-get install –yes libcairo2-dev libjpeg62-dev libpng12-dev libossp-uuid-dev libfreerdp-dev libpango1.0-dev libssh2-1-dev libwebp-dev libssl-dev build-essential maven openjdk-7-jdk nfs-common tomcat7 mysql-server-5.5 mysql-client-5.5 apache2 libapache2-mod-proxy-html	1:51	0:14	0:01
10	MLIn@ds	[set mysql root password to 'root']	0:04	0:00	0:00
11	Apt-get@ds	[installation continues]	0:00	2:28	0:00
12	Bash@rs	sudo mount 10.0.20.22:/network_files /mnt/	0:08	0:00	0:02
13	Bash@rs	sudo tar -xzf /mnt/guac/guacamole-client-0.9.13-incubating.tar.gz -C /opt/	0:13	0:01	0:01
14	Bash@rs	sudo tar -xzf /mnt/guac/guacamole-server-0.9.13-incubating.tar.gz -C /opt/	0:05	0:01	0:00
15	Bash@rs	sudo chown -R root:root /opt	0:05	0:00	0:00
16	Bash@rs	cd /opt/guacamole-server-0.9.13-incubating/	0:04	0:00	0:02

Legend: Horizon - OpenStack Horizon web interface, WSsh - Putty on Windows, oc - operator computer, rs - remote desktop server, MLIn - MySQL installer, MLSh - MySQL shell, wc - windows client, Cmd - Windows shell, {*value that varies between deployments*}, *[complexer gui command or entered described as text], times shown as* mm:ss

Table C.6: Manual deployment of the remote desktop service (1/4).

i	Context	Command	p_i	e_i	w_i
17	Bash@rs	sudo /opt/guacamole-server-0.9.13-incubating/configure –with-init-dir=/etc/init.d	0:13	0:20	0:01
18	Bash@rs	sudo make -C /opt/guacamole-server-0.9.13-incubating	0:06	1:03	0:02
19	Bash@rs	sudo make install -C /opt/guacamole-server-0.9.13-incubating	0:06	0:06	0:02
20	Bash@rs	sudo ldconfig	0:03	0:00	0:01
21	Bash@rs	sudo echo GUACAMOLE_HOME=/opt/guacamole-server-0.9.13-incubating — sudo tee -a /etc/environment	0:42	0:00	0:06
22	Bash@rs	sudo echo GUACAMOLE_HOME=/opt/guacamole-server-0.9.13-incubating — sudo tee -a /etc/default/tomcat7	0:30	0:00	0:00
23	Bash@rs	cd /opt/guacamole-client-0.9.13-incubating	0:04	0:00	0:12
24	Bash@rs	sudo mvn package	0:03	1:48	0:02
25	Bash@rs	sudo cp guacamole/target/guacamole-0.9.13-incubating.war /var/lib/tomcat7/webapps	0:06	0:00	0:09
26	Bash@rs	sudo cp /mnt/guac/guacamole.properties /opt/guacamole-server-0.9.13-incubating/	0:17	0:00	0:00
27	Bash@rs	sudo chmod 644 /opt/guacamole-server-0.9.13-incubating/guacamole.properties	0:12	0:00	0:01
28	Bash@rs	sudo mkdir /opt/guacamole-server-0.9.13-incubating/lib	0:08	0:00	0:01
29	Bash@rs	sudo mkdir /opt/guacamole-server-0.9.13-incubating/extensions	0:08	0:00	0:00
30	Bash@rs	sudo cp /mnt/guac/guacamole-auth-jdbc-mysql-0.9.13-incubating.jar /opt/guacamole-server-0.9.13-incubating/extensions	0:12	0:00	0:04
31	Bash@rs	sudo chmod 655 /opt/guacamole-server-0.9.13-incubating/extensions/guacamole-auth-jdbc-mysql-0.9.13-incubating.jar	0:15	0:00	0:00
32	Bash@rs	sudo cp /mnt/guac/mysql-connector-java-5.1.44-bin.jar /opt/guacamole-server-0.9.13-incubating/lib	0:08	0:00	0:03
33	Bash@rs	sudo chmod 655 /opt/guacamole-server-0.9.13-incubating/lib/mysql-connector-java-5.1.44-bin.jar	0:11	0:00	0:00
34	Bash@rs	mysql -u root -proot	0:06	0:00	0:03
35	MLSh@rs	CREATE DATABASE guacamole_db CHARACTER SET utf8;	0:12	0:00	0:01
36	MLSh@rs	GRANT ALL PRIVILEGES ON guacamole_db.* TO 'guacamole_user'@'localhost' IDENTIFIED BY 'root';	0:24	0:00	0:02
37	MLSh@rs	FLUSH PRIVILEGES;	0:05	0:00	0:01
38	MLSh@rs	quit	0:01	0:00	0:00

Legend: Horizon - OpenStack Horizon web interface, WSsh - Putty on Windows, oc - operator computer, rs - remote desktop server, MLIn - MySQL installer, MLSh - MySQL shell, wc - windows client, Cmd - Windows shell, {*value that varies between deployments*}, *[complexer gui command or entered described as text], times shown as* mm:ss

Table C.7: Manual deployment of the remote desktop service (2/4).

i	Context	Command	p_i	e_i	w_i
39	Bash@rs	mysql -u root -proot guacamole_db < /mnt/guac/001-create-schema.sql	0:13	0:00	0:02
40	Bash@rs	mysql -u root -proot guacamole_db < /mnt/guac/002-create-admin-user.sql	0:03	0:00	0:01
41	Bash@rs	cd	0:01	0:00	0:02
42	Bash@rs	vi /mnt/guac/connections.sql.sample	0:06	0:00	0:00
43	Vi@rs	:%s/ip_address/[client1-ip]/g	0:16	0:02	0:01
44	Vi@rs	:w connections1.sql	0:06	0:00	0:01
45	Vi@rs	u	0:00	0:00	0:00
46	Vi@rs	:%s/ip_address/[client2-ip]/g	0:18	0:00	0:01
47	Vi@rs	:w connections2.sql	0:04	0:00	0:01
48	Vi@rs	u	0:00	0:00	0:00
49	Vi@rs	:%s/ip_address/[client3-ip]/g	0:06	0:00	0:03
50	Vi@rs	:wq connections3.sql	0:20	0:00	0:01
51	Bash@rs	mysql -u root -proot guacamole_db < connections1.sql	0:08	0:00	0:06
52	Bash@rs	mysql -u root -proot guacamole_db < connections2.sql	0:01	0:00	0:01
53	Bash@rs	mysql -u root -proot guacamole_db < connections3.sql	0:01	0:00	0:01
54	Bash@rs	sudo a2enmod proxy	0:05	0:00	0:01
55	Bash@rs	sudo a2enmod proxy_wstunnel	0:03	0:00	0:01
56	Bash@rs	sudo a2enmod proxy_http	0:02	0:00	0:00
57	Bash@rs	nslookup [rs-ip]	0:07	0:00	0:04
58	Bash@rs	sudo vi /etc/apache2/sites-enabled/000-default.conf	0:07	0:00	0:06
59	Vi@rs	[erase and add: <VirtualHost *:80> ServerName {rs-dns-name} <Location /> Order allow,deny Allow from all ProxyPass http://localhost:8080/guacamole-0.9.13-incubating/ flushpackets=on ProxyPassReverse http://localhost:8080/guacamole-0.9.13-incubating/ ProxyPassReverseCookiePath /guacamole-0.9.13-incubating/ / </Location> <Location /websocket-tunnel> Order allow,deny Allow from all ProxyPass ws://localhost:8080/guacamole-0.9.13-incubating/websocket-tunnel ProxyPassReverse ws://localhost:8080/guacamole-0.9.13-incubating/websocket-tunnel </Location> </VirtualHost>]	0:44	0:00	0:00
60	Vi@rs	:wq	0:01	0:00	0:00
61	Bash@rs	sudo service guacd restart	0:16	0:01	0:05

Legend: Horizon - OpenStack Horizon web interface, WSsh - Putty on Windows, oc - operator computer, rs - remote desktop server, MLIn - MySQL installer, MLSh - MySQL shell, wc - windows client, Cmd - Windows shell, {*value that varies between deployments*}, *[complexer gui command or entered described as text], times shown as* mm:ss

Table C.8: Manual deployment of the remote desktop service (3/4).

i	Context	Command	p_i	e_i	w_i
62	Bash@rs	sudo service tomcat7 restart	0:05	0:06	0:01
63	Bash@rs	sudo service apache2 restart	0:02	0:03	0:01
64	Bash@rs	exit	0:01	0:00	0:00
65	WSsh@oc	[ssh to server (wc) windows client 1]	0:12	0:03	0:22
66	Cmd@wc1	net use N:\\host-10-0-20-22.openstacklocal\network_files /user:ubuntu f834jSX	0:26	0:01	0:00
67	Cmd@wc1	cmd /C ”start /wait msiexec /qn /i N:\\LibreOffice_5.4.3_Win_x64.msi /log C:\\libreOffice_install.log INSTALLLOCATION=”C:\\Program Files\LibreOffice” ISCHECKFORPRODUCTUPDATES=0 REGISTER_ALL_MSO_TYPES=1 RebootYesNo=No”	0:07	0:57	0:00
68	Cmd@wc1	net user Administrator RemDesk-001	0:09	0:00	0:04
69	Cmd@wc1	exit	0:01	0:00	0:00
70	WSsh@oc	[ssh to server (wc) windows client 2]	0:12	0:03	0:22
71	Cmd@wc2	net use N: \\host-10-0-20-22.openstacklocal\network_files /user:ubuntu f834jSX	0:26	0:01	0:00
72	Cmd@wc2	cmd /C ”start /wait msiexec /qn /i N:\\LibreOffice_5.4.3_Win_x64.msi /log C:\\libreOffice_install.log INSTALLLOCATION=”C:\\Program Files\LibreOffice” ISCHECKFORPRODUCTUPDATES=0 REGISTER_ALL_MSO_TYPES=1 RebootYesNo=No”	0:07	0:53	0:00
73	Cmd@wc2	net user Administrator RemDesk-002	0:10	0:00	0:03
74	Cmd@wc2	exit	0:01	0:00	0:00
75	WSsh@oc	[ssh to server (wc) windows client 3]	0:12	0:03	0:20
76	Cmd@wc3	net use N: \\host-10-0-20-22.openstacklocal\network_files /user:ubuntu f834jSX	0:26	0:01	0:00
77	Cmd@wc3	cmd /C ”start /wait msiexec /qn /i N:\\LibreOffice_5.4.3_Win_x64.msi /log C:\\libreOffice_install.log INSTALLLOCATION=”C:\\Program Files\LibreOffice” ISCHECKFORPRODUCTUPDATES=0 REGISTER_ALL_MSO_TYPES=1 RebootYesNo=No”	0:06	0:59	0:00
78	Cmd@wc3	net user Administrator RemDesk-003	0:09	0:00	0:04
79	Cmd@wc3	exit	0:01	0:00	0:00
		Total	14:42	10:20	3:02

Legend: Horizon - OpenStack Horizon web interface, WSsh - Putty on Windows, oc - operator computer, rs - remote desktop server, MLIn - MySQL installer, MLSh - MySQL shell, wc - windows client, Cmd - Windows shell, {*value that varies between deployments*}, *[complexer gui command or entered described as text], times shown as* mm:ss

Table C.9: Manual deployment of the remote desktop service (4/4).

i	Context	Command	p_i	e_i	w_i
1	Horizon	[spawn one m1.xlarge instance named 'sap-server' of 'SAP_SLES_11_SP3.x86_64-0.0.7']	0:23	0:10	0:02
2	Horizon	[associate floating ip]	0:05	0:04	0:04
3	Horizon	[create volume with 200 GB]	0:12	0:06	0:02
4	Horizon	[attach volume as /dev/vdb with ss]	0:05	0:04	0:08
5	WSsh@oc	[ssh to server (ss) sap-server, with x-forwarding]	0:36	0:04	0:04
6	Bash@ss	groupadd -g 1000 sapinst	0:04	0:02	0:05
7	Bash@ss	usermod -G sapinst root	0:06	0:00	0:01
8	Bash@ss	mkdir /sapinst	0:03	0:00	0:01
9	Bash@ss	mount 10.0.20.22:/network_files /mnt/	0:09	0:00	0:01
10	Bash@ss	echo 1 > /sys/bus/pci/rescan	0:12	0:00	0:01
11	Bash@ss	sfdisk –force /dev/vdb < /mnt/sap-host-setup/vdb.layout	0:13	0:02	0:01
12	Bash@ss	ls /dev/vdb1	0:03	0:00	0:02
13	Bash@ss	mkswap /dev/vdb1	0:04	0:04	0:02
14	Bash@ss	swapon /dev/vdb1	0:09	0:00	0:00
15	Bash@ss	mkfs.ext3 /dev/vdb2	0:07	0:52	0:01
16	Bash@ss	mount /dev/vdb2 /sapinst	0:10	0:01	0:01
17	Bash@ss	chown root:sapinst /sapinst	0:05	0:00	0:01
18	Bash@ss	chmod 775 /sapinst	0:03	0:00	0:02
19	Bash@ss	mkdir /sapinst/sapdb	0:04	0:00	0:02
20	Bash@ss	mkdir /sapinst/sapmnt	0:06	0:00	0:01
21	Bash@ss	mkdir /sapinst/usrsap	0:03	0:00	0:00
22	Bash@ss	chmod -R 755 /sapinst/*	0:07	0:00	0:01
23	Bash@ss	ln -s /sapinst/sapdb /sapdb	0:07	0:00	0:01
24	Bash@ss	ln -s /sapinst/sapmnt /sapmnt	0:07	0:00	0:01
25	Bash@ss	ln -s /sapinst/usrsap /usr/sap	0:09	0:00	0:01
26	Bash@ss	mkdir /sapinst/installfiles	0:05	0:00	0:00
27	Bash@ss	bash /mnt/jdk-6u45-linux-x64-rpm.bin	0:03	0:21	0:03
28	Bash@ss	cp -r /mnt/sap_erp_files/* /sapinst/installfiles	0:11	12:04	0:02
29	Bash@ss	cd /sapinst/installfiles/SPM/SPM_extracted	0:15	0:00	0:22
30	Bash@ss	echo host{*last-ip-digit-group*} > /etc/hostname	0:27	0:00	0:49
31	Bash@ss	hostname -F /etc/hostname	0:08	0:00	0:00
32	Bash@ss	vi /etc/hosts	0:04	0:00	0:05
33	Vi@ss	[add hostnames 10.0.20.{last-ip-digit-group} host{last-ip-digit-group}.openstacklocal host{last-ip-digit-group} host-10-0-20-{last-ip-digit-group}.openstacklocal]	0:36	0:00	0:00
34	Vi@ss	:wq	0:01	0:00	0:00
35	Bash@ss	./sapinst	0:10	0:31	0:01
36	Sins@ss	[the following installation option is used SAP NetWeaver 7.0 including Enhancement Package 2 > Software life cycle Options > System Copy > MaxDB > Target System Installation > Central System > Based on AS ABAP > Central System several further parameters need to be entered]	7:20	430:52	0:00
		Total	12:52	445:17	2:08

Legend: Horizon - OpenStack Horizon web interface, WSsh - Putty on Windows, oc - operator computer, ss - SAP ERP server, {*value that varies between deployments*}, *[complexer gui command or entered described as text], times shown as* mm:ss

Table C.10: Manual deployment of the SAP ERP service.

i	Context	Command	p_i	e_i	w_i
1	Horizon	[spawn two m1.small instances named 'wordpress-mysql' of 'Ubuntu Server 14.04 LTS (Trusty Tahr)']	0:28	0:11	0:02
2	Horizon	[associate floating ip with webserver instance (wordpress-mysql-1)]	0:05	0:04	0:02
3	Horizon	[associate floating ip with database instance (wordpress-mysql-2)]	0:03	0:04	0:03
4	WSsh@oc	[ssh to database server (ds) wordpress-mysql-2]	0:06	0:04	0:09
5	Bash@ds	sudo apt-get update	0:01	0:09	0:01
6	Bash@ds	sudo apt-get install –yes mysql-server-5.5 mysql-client-5.5	0:18	0:03	0:00
7	MLIn@ds	[set mysql root password to 'wordpress']	0:04	0:00	0:00
8	Apt-get@ds	[installation continues]	0:00	0:21	0:00
9	Bash@ds	mysql -u root -p	0:07	0:00	0:02
10	MLSh@ds	CREATE DATABASE wordpress CHARACTER SET utf8;	0:12	0:00	0:02
11	MLSh@ds	GRANT ALL PRIVILEGES ON wordpress.* TO 'wordpress'@'{*ws-ip*}' IDENTIFIED BY 'wordpress';	0:32	0:00	0:03
12	MLSh@ds	FLUSH PRIVILEGES;	0:07	0:00	0:01
13	MLSh@ds	quit	0:01	0:00	0:01
14	Bash@ds	sudo vi /etc/mysql/my.cnf	0:04	0:01	0:00
15	Vi@ds	[set bind-address to 0.0.0.0 -> bind-address = 0.0.0.0]	0:12	0:00	0:00
16	Vi@ds	:wq	0:05	0:03	0:01
17	Vi@ds	sudo service mysql restart	0:01	0:00	0:01
18	Bash@ds	exit	0:01	0:00	0:00
19	WSsh@oc	[ssh to webserver server (ds) wordpress-mysql-1]	0:09	0:04	0:01
20	Bash@ws	wget http://wordpress.org/wordpress-3.8.tar.gz	0:12	0:04	0:00
21	Bash@ws	tar xvfz wordpress-3.8.tar.gz	0:06	0:01	0:00
22	Bash@ws	sudo chown -R www-data:www-data wordpress	0:07	0:00	0:02
23	Bash@ws	sudo chmod -R 755 wordpress	0:07	0:00	0:01
24	Bash@ds	sudo apt-get update	0:03	0:09	0:01
25	Bash@ws	sudo apt-get install –yes apache2 php5-mysql libapache2-mod-php5	0:18	0:19	0:01
26	Bash@ws	sudo rm -rf /var/www/html/index.html	0:06	0:00	0:01
27	Bash@ws	sudo mv wordpress/* /var/www/html	0:09	0:00	0:02
28	Bash@ws	cd /var/www/html	0:10	0:00	0:01
29	Bash@ws	sudo cp wp-config-sample.php wp-config.php	0:11	0:00	0:00
30	Bash@ws	sudo vi wp-config.php	0:06	0:01	0:03
31	Vi@ws	[create database connection, write: define('DB_NAME', 'wordpress'); define('DB_USER', 'wordpress'); define('DB_PASSWORD', 'wordpress'); define('DB_HOST', '{*ds-ip*}');]	0:36	0:00	0:00
32	Vi@ws	:wq	0:02	0:00	0:00
33	Bash@ws	sudo a2enmod mpm_prefork	0:05	0:01	0:00
34	Bash@ws	sudo service apache2 restart	0:06	0:02	0:01
35	Bash@ws	nslookup {*floating-ip*}	0:05	0:01	0:02
		Total	5:05	1:42	0:44

Legend: Horizon - OpenStack Horizon web interface, WSsh - Putty on Windows, oc - operator computer, ds - database server, ws - web server, MLIn - MySQL installer, MLSh - MySQL shell, {*value that varies between deployments*}, *[complexer gui command or entered described as text], times shown as* mm:ss

Table C.11: Manual deployment of the CMS (with MySQL).

C.3 Automated deployments

The LOC measurements of Tables C.12, C.13, and C.14 were performed with the tool github.com/AlDanial/cloc in version 1.74. The following files and directories were ignored (Puppet, Inc., 2018)

- files with extension .md (markdown files are used for documentation)
- files with extension .json (json files contain metadata and checksums)
- files with extension .po (po files contain localizations)
- directory files (contains configuration files)
- directory templates (contains template files that become configuration files)
- directory tests (contains tests for the module
- directory spec (contains tests for plugins)

Code in the following domain-specific and general purpose programming languages was counted (Puppet, Inc., 2018):

- Ruby
- Puppet
- Yaml

Configuration model type	Creator	Name	Child	Version	LOC
Orchestration	jhintsch	oc_wordpress		0.0.1	63
Software	jhintsch	ac_liboff_win		0.0.1	11
Software	jhintsch	ac_remdesk		0.0.1	190
Software	puppetlabs	apache		1.10.0	4.765
Software	puppetlabs	mysql		3.11.0	2.310
Software	puppetlabs	stdlib	x	4.21.0	4.908
Software	puppetlabs	concat	x	2.2.1	454
Software	puppetlabs	tomcat	x	1.7.0	1.771
				Total	14.472

Table C.12: Structure of the remote desktop service and LOC for each model.

Configuration model type	Creator	Name	Child	Version	LOC
Orchestration	jhintsch	oc_saperp		0.0.1	42
Software	jhintsch	sc_saperp		0.0.1	183
				Total	225

Table C.13: Structure of the SAP ERP service and LOC for each model.

Configuration model type	Creator	Name	Child	Version	LOC
Orchestration	jhintsch	oc_wordpress		0.0.1	89
Software	hunner	wordpress		0.0.1	306
Software	puppetlabs	apache		1.10.0	4.765
Software	puppetlabs	mysql		3.9.0	2.160
Software	puppetlabs	stdlib	x	4.12.0	3.952
Software	puppetlabs	concat	x	2.2.20	394
				Total	11.666

Table C.14: Structure of the CMS (with MySQL) and LOC for each model.

System context	Step	*avg*
PE system	Get models	0:00:23
PE system	Post models	0:00:00
IaaS system	Setup of instances	0:00:30
PE system	Connection configuration	0:08:10
Configuration agent	Install package dependencies	0:02:22
Configuration agent	Store and extract files of guacamole	0:00:01
Configuration agent	Setup environment variables	0:00:00
Configuration agent	Install guacamole server	0:01:31
Configuration agent	Install guacamole web client	0:01:44
Configuration agent	Setup Apache Tomcat	0:00:18
Configuration agent	Setup Apache HTTP Server	0:00:12
Configuration agent	Install MySQL and setup database	0:00:28
Configuration agent	Load remote desktop connections into database	0:00:00
Configuration agent	Mount network dir	0:00:03
Configuration agent	Install LibreOffice	0:02:54

Times written format *hh:mm:ss* , *avg* - average

Table C.15: Automated deployment of the remote desktop service with three small clients.

System context	Step	*avg*
PES	Get models	0:00:02
PES	Post models	0:00:00
IaaS	Instance setup	0:00:38
PES	Connection configuration	0:01:27
Configuration agent	Configure user	0:00:00
Configuration agent	Configure hostnames	0:00:00
Configuration agent	Mount and partition volume	0:00:36
Configuration agent	Mount network storage	0:00:00
Configuration agent	Create directory structure	0:00:03
Configuration agent	Install Java	0:00:16
Configuration agent	Copy installation files	0:09:45
Configuration agent	Execute SAPinst	7:27:58

Times written format *hh:mm:ss* , *avg* - average

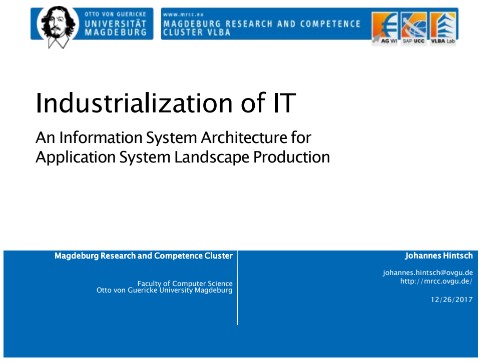
Table C.16: Automated deployment of SAP ERP with three small clients.

System context	Step	*avg*
PES	Get models	0:00:18
PES	Post models	0:00:00
IaaS	Instance setup	0:00:25
PES	Connection configuration	0:02:21
Configuration agent	Setup MySQL	0:00:24
Configuration agent	Setup Wordpress	0:00:04
Configuration agent	Setup Apache and Vhost	0:00:23

Times written format *hh:mm:ss* , *avg* - average

Table C.17: Automated deployment of CMS (with MySQL).

C.4 Presentation for expert interviews



Industrialization of IT

An Information System Architecture for Application System Landscape Production

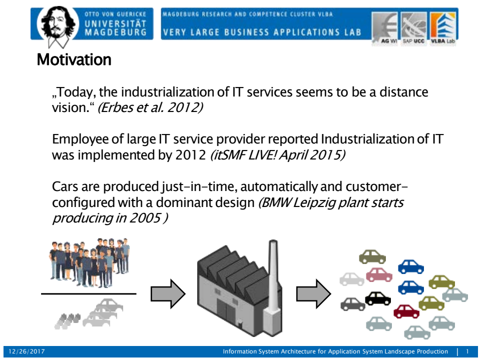
Magdeburg Research and Competence Cluster

Faculty of Computer Science
Otto von Guericke University Magdeburg

Johannes Hintsch

johannes.hintsch@ovgu.de
http://mrcc.ovgu.de/

12/26/2017



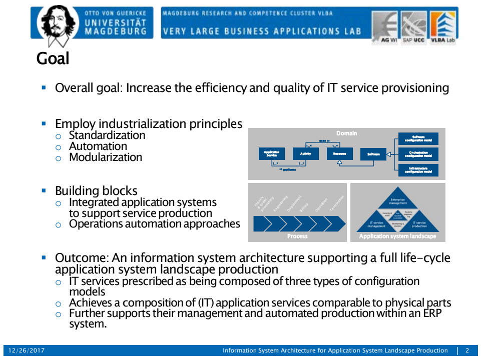
Motivation

„Today, the industrialization of IT services seems to be a distance vision.“ *(Erbes et al. 2012)*

Employee of large IT service provider reported Industrialization of IT was implemented by 2012 *(itSMF LIVE! April 2015)*

Cars are produced just-in-time, automatically and customer-configured with a dominant design *(BMW Leipzig plant starts producing in 2005)*

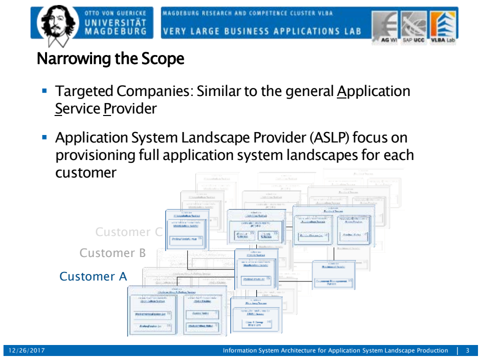
12/26/2017 Information System Architecture for Application System Landscape Production | 1



Goal

- Overall goal: Increase the efficiency and quality of IT service provisioning
- Employ industrialization principles
  - Standardization
  - Automation
  - Modularization
- Building blocks
  - Integrated application systems to support service production
  - Operations automation approaches
- Outcome: An information system architecture supporting a full life-cycle application system landscape production
  - IT services prescribed as being composed of three types of configuration models
  - Achieves a composition of (IT) application services comparable to physical parts
  - Further supports their management and automated production within an ERP system.

12/26/2017 Information System Architecture for Application System Landscape Production | 2

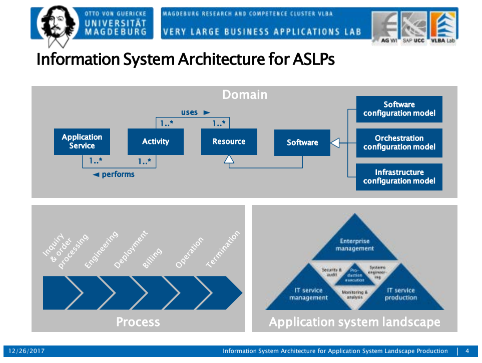


Narrowing the Scope

- Targeted Companies: Similar to the general Application Service Provider
- Application System Landscape Provider (ASLP) focus on provisioning full application system landscapes for each customer

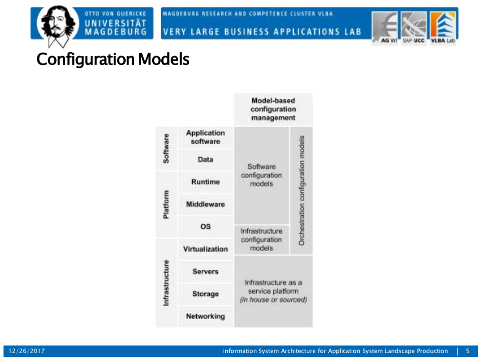
Customer C
Customer B
Customer A

12/26/2017 Information System Architecture for Application System Landscape Production | 3



Information System Architecture for ASLPs

12/26/2017 Information System Architecture for Application System Landscape Production | 4

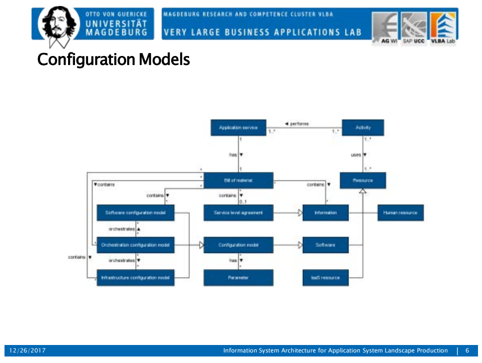


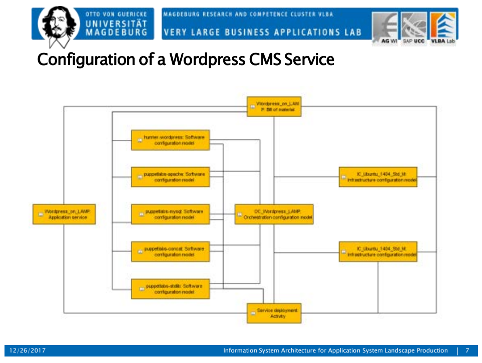
Configuration Models

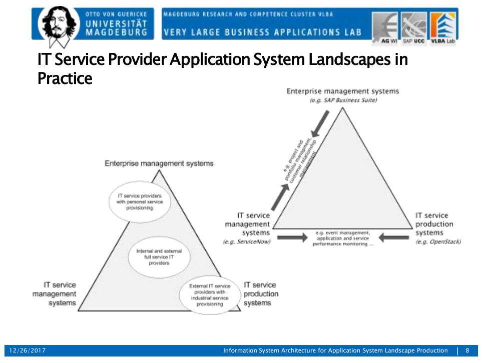
Model-based configuration management
Software
Application software
Data
Platform
Runtime
Middleware
OS
Infrastructure
Virtualization
Servers
Storage
Networking
Software configuration models
Infrastructure configuration models
Orchestration configuration models
Infrastructure as a service platform
(in house or sourced)

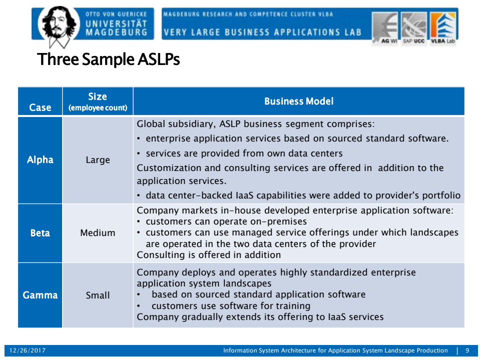
12/26/2017 Information System Architecture for Application System Landscape Production | 5

Figure C.1: Slide deck of presentation to inform experts (Slides 0 - 5).




OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG
MAGDEBURG RESEARCH AND COMPETENCE CLUSTER VLBA
VERY LARGE BUSINESS APPLICATIONS LAB
AG WI
SAP UCC
VLBA Lab
Configuration of a Wordpress CMS Service
Wordpress_on_LAMP: Bill of material
hunner-wordpress: Software configuration model
puppetlabs-apache: Software configuration model
puppetlabs-mysql: Software configuration model
puppetlabs-concat: Software configuration model
puppetlabs-stdlib: Software configuration model
Wordpress_on_LAMP: Application service
OC_Wordpress_LAMP: Orchestration configuration model
IC_Ubuntu_1404_Std_M: Infrastructure configuration model
IC_Ubuntu_1404_Std_M: Infrastructure configuration model
Service deployment: Activity
12/26/2017
Information System Architecture for Application System Landscape Production | 7

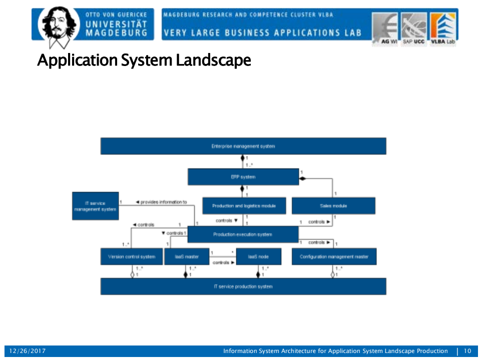

OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG
MAGDEBURG RESEARCH AND COMPETENCE CLUSTER VLBA
VERY LARGE BUSINESS APPLICATIONS LAB
AG WI
SAP UCC
VLBA Lab
IT Service Provider Application System Landscapes in Practice
Enterprise management systems
IT service providers with personal service provisioning
Internal and external full service IT providers
External IT service providers with industrial service provisioning
IT service management systems
IT service production systems
Enterprise management systems
(e.g. SAP Business Suite)
e.g. project and portfolio management, customer relationship management
IT service management systems
(e.g. ServiceNow)
e.g. event management, application and service performance monitoring ...
IT service production systems
(e.g. OpenStack)
12/26/2017
Information System Architecture for Application System Landscape Production | 8


OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG
MAGDEBURG RESEARCH AND COMPETENCE CLUSTER VLBA
VERY LARGE BUSINESS APPLICATIONS LAB
AG WI
SAP UCC
VLBA Lab

Three Sample ASLPs

Case	Size (employee count)	Business Model
Alpha	Large	Global subsidiary, ASLP business segment comprises: • enterprise application services based on sourced standard software. • services are provided from own data centers Customization and consulting services are offered in addition to the application services. • data center-backed IaaS capabilities were added to provider's portfolio
Beta	Medium	Company markets in-house developed enterprise application software: • customers can operate on-premises • customers can use managed service offerings under which landscapes are operated in the two data centers of the provider Consulting is offered in addition
Gamma	Small	Company deploys and operates highly standardized enterprise application system landscapes • based on sourced standard application software • customers use software for training Company gradually extends its offering to IaaS services

12/26/2017 Information System Architecture for Application System Landscape Production | 9


OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG
MAGDEBURG RESEARCH AND COMPETENCE CLUSTER VLBA
VERY LARGE BUSINESS APPLICATIONS LAB
AG WI
SAP UCC
VLBA Lab
Application System Landscape
Enterprise management system
1
1..*
ERP system
1
1
1
IT service management system
1
◀ provides information to
Production and logistics module
Sales module
1
controls ▼
1
1
1
controls ▶
◀ controls
1
▼ controls 1
Production execution system
1
controls ▶
1
1..*
1
Version control system
IaaS master
1
*
IaaS node
controls ▶
Configuration management master
1..*
1
1..*
1
1..*
1
1..*
1
IT service production system
12/26/2017
Information System Architecture for Application System Landscape Production | 10

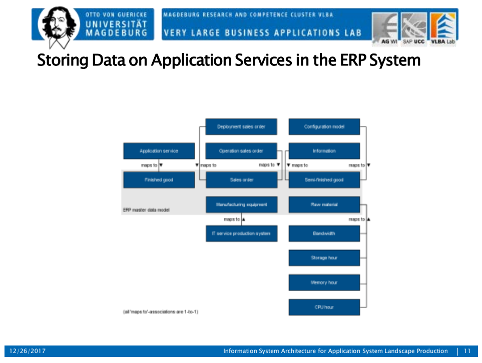
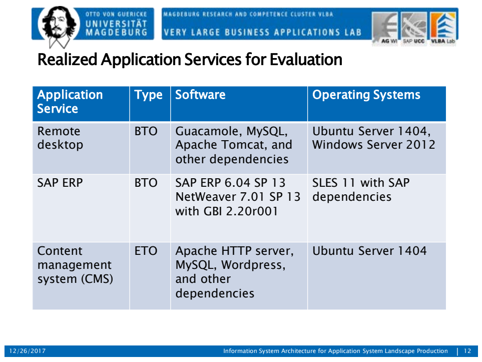


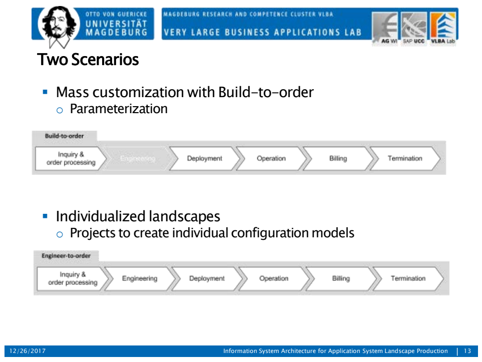
Figure C.2: Slide deck of presentation to inform experts (Slides 6 - 11).



Realized Application Services for Evaluation

Application Service	Type	Software	Operating Systems
Remote desktop	BTO	Guacamole, MySQL, Apache Tomcat, and other dependencies	Ubuntu Server 1404, Windows Server 2012
SAP ERP	BTO	SAP ERP 6.04 SP 13 NetWeaver 7.01 SP 13 with GBI 2.20r001	SLES 11 with SAP dependencies
Content management system (CMS)	ETO	Apache HTTP server, MySQL, Wordpress, and other dependencies	Ubuntu Server 1404

12/26/2017 Information System Architecture for Application System Landscape Production | 12



Two Scenarios

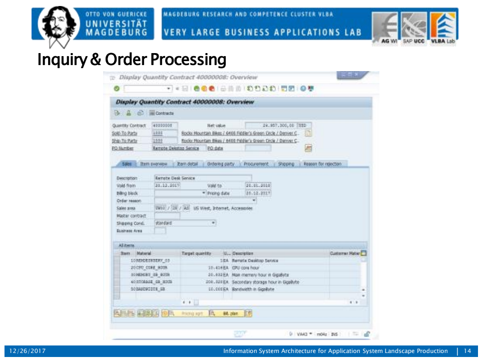
- Mass customization with Build-to-order
  - Parameterization

Build-to-order
Inquiry & order processing
Engineering
Deployment
Operation
Billing
Termination

- Individualized landscapes
  - Projects to create individual configuration models

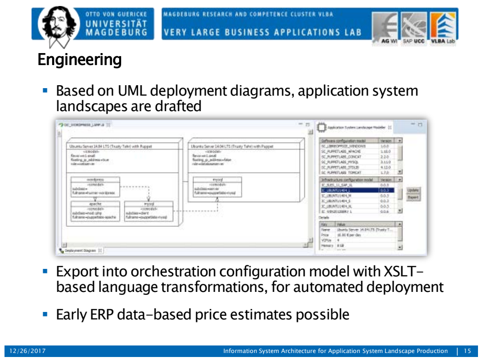
Engineer-to-order
Inquiry & order processing
Engineering
Deployment
Operation
Billing
Termination

12/26/2017 Information System Architecture for Application System Landscape Production | 13



Inquiry & Order Processing

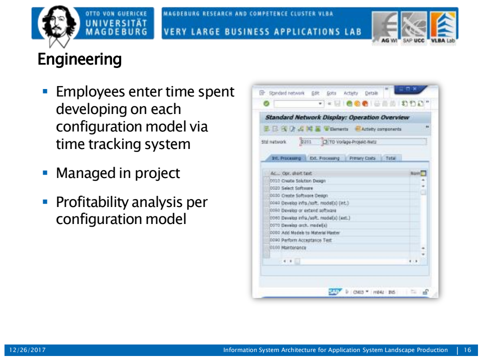
12/26/2017 Information System Architecture for Application System Landscape Production | 14



Engineering

- Based on UML deployment diagrams, application system landscapes are drafted
- Export into orchestration configuration model with XSLT-based language transformations, for automated deployment
- Early ERP data-based price estimates possible

12/26/2017 Information System Architecture for Application System Landscape Production | 15

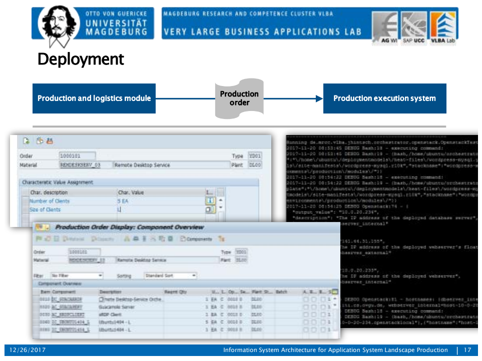


Engineering

- Employees enter time spent developing on each configuration model via time tracking system
- Managed in project
- Profitability analysis per configuration model

Standard Network Display: Operation Overview

12/26/2017 Information System Architecture for Application System Landscape Production | 16

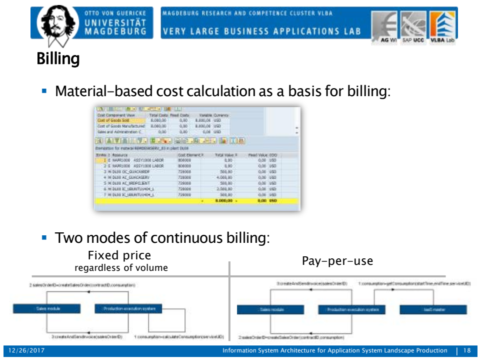


Deployment

Production and logistics module
Production order
Production execution system
Production Order Display: Component Overview

12/26/2017 Information System Architecture for Application System Landscape Production | 17

Figure C.3: Slide deck of presentation to inform experts (Slides 12 - 17).

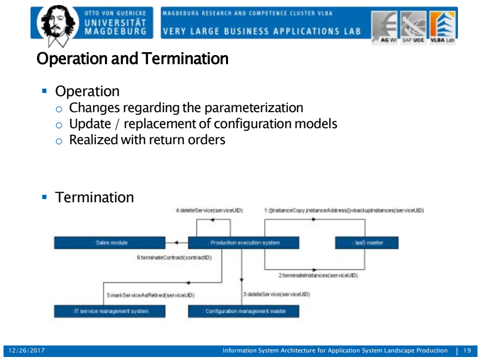


Billing

- Material-based cost calculation as a basis for billing:
- Two modes of continuous billing:

Fixed price
regardless of volume
Pay-per-use
2:salesOrderID=createSalesOrder(contractID,consumption)
: Sales module
: Production execution system
3:createAndSendInvoice(salesOrderID)
1:consumption=calculateConsumption(serviceUID)
3:createAndSendInvoice(salesOrderID)
1:consumption=getConsumption(startTime,endTime,serviceUID)
: Sales module
: Production execution system
: IaaS master
2:salesOrderID=createSalesOrder(contractID,consumption)

12/26/2017 Information System Architecture for Application System Landscape Production | 18

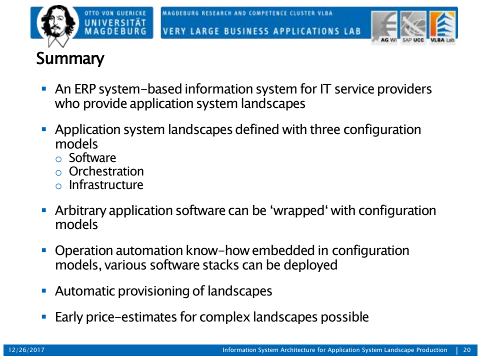


Operation and Termination

- Operation
  - Changes regarding the parameterization
  - Update / replacement of configuration models
  - Realized with return orders
- Termination

4:deleteService(serviceUID)
1:([]instanceCopy,instanceAddress[])=backupInstances(serviceUID)
: Sales module
: Production execution system
: IaaS master
6:terminateContract(contractID)
2:terminateInstances(serviceUID)
5:markServiceAsRetired(serviceUID)
3:deleteService(serviceUID)
: IT service management system
: Configuration management master

12/26/2017 Information System Architecture for Application System Landscape Production | 19



Summary

- An ERP system-based information system for IT service providers who provide application system landscapes
- Application system landscapes defined with three configuration models
  - Software
  - Orchestration
  - Infrastructure
- Arbitrary application software can be 'wrapped' with configuration models
- Operation automation know-how embedded in configuration models, various software stacks can be deployed
- Automatic provisioning of landscapes
- Early price-estimates for complex landscapes possible

12/26/2017 Information System Architecture for Application System Landscape Production | 20

Questions?

Magdeburg Research and Competence Cluster
Faculty of Computer Science
Otto von Guericke University Magdeburg

Johannes Hintsch
johannes.hintsch@ovgu.de
http://mrcc.ovgu.de/
12/26/2017

Figure C.4: Slide deck of presentation to inform experts (Slides 18 - 21).

C.5 Questions for expert interviews

1. Opening question

   a) How do you assess the increasing standardization and automation in IT?

2. Application services

   a) The architecture proposes the reuse of configuration models between different application services provided to customers. Does this reuse increase overall efficiency? If not, what is problematic?

   b) All operational tasks are recorded in configuration models. Expenses incurred can be directly allocated to these models. Is this helpful in determining the profitability of products? If no, why not?

   c) Application services are compiled from configuration models. The development of these models entails costs that can be attributed to the models. This makes it easy to determine the costs for an application system landscape before engineering. Is this determinability helpful? If no, why not?

   d) Can you imagine application services that cannot be implemented with the architecture? If so, what are they?

   e) Service levels are used in the architecture to signal to the helpdesk which response times have been agreed upon or to quantify the agreed service availability. Is it sufficient looking at the current standard practice? If not, how should the agreements be extended?

   f) Two types of service settlement are addressed: pay-per-use and fixed price regardless of volume. Are these types of settlement sufficient? If not, which settlement types still need to be included?

   g) In principle, there are two types of services: Engineer-to-order and build-to-order. Are these two types sufficient? If not, what other type should be added?

   h) Is the automated provision and operation of application services as described above practicable and useful? If no, why not?

3. Application system landscape

   a) The application system landscape of the information system architecture has an ERP system as the central leading system. Here, the application services are not only stored as sales products but are represented in their complete structure. Does storing this complete storage make sense? If no, why not?

   b) Copies of the IaaS instances can be made available to the customer when terminating application services. Is this service sufficient upon termination of the contract or would it have to be extended?

c) The concept requires the entire infrastructure layer of the IT stack to be provided by IaaS software. Can this requirement be met? If not, how should the architecture be adapted?

d) The application systems of the information architecture essentially comprise three types of application systems: Enterprise Management, IT Service Management, and IT Service Production Systems. Security & audit, systems engineering, and monitoring & analysis systems are other systems that are not in focus. Does this selection correspond to the most critical systems of ASLP? If not, how should the selection be adjusted?

4. General

   a) If you were an ASLP and had sufficient means to implement the information system architecture, would you implement it? If no, what would have to be adjusted? What would be the criteria for introducing the architecture?

   b) Does the proposed information system architecture increase the quality and efficiency of the creation of application system landscapes?

C.6 Transcripts of expert interviews

The full German transcripts were translated with a free online translation service (available at https://www.deepl.com/translator) and edited for correctness manually afterwards.

C.6.1 Interview 1

Presentation is started and then is paused at slide 3.

RES. A question, Johannes, you are very much interested in processes, systems, do you also consider the agile environment? If you use waterfall, you use Scrum, use Kanban?

INT. I'm not covering it in detail, but I would say that the architecture is basically compatible with different software engineering process models. Of course, this is something else when such a landscape is going live and then used compared to normal software development. There, you can continuously add features. But, the fact that a service can change is of course considered.

RES. You must be familiar with Docker. You can run applications where your main infrastructure is different from the application in the Docker container. So you also have mix models, so that you really define Infrastructure as a Service, with, ok this is my platform. Nowadays, people say AWS[1] or Microsoft is the non-plus ultra in the areas, at the push of a button, and then they roll it out. And the application itself then runs in a Docker container and then you are completely independent again what you take for an operating system. That means you can also run matrix operating systems without disturbing each other anyway.

[1]Amazon web services (AWS)

INT. Exactly, the applications can be very strongly encapsulated. That is one of the core themes of the architecture.

RES. Yes, the infrastructure and platform level, you can almost say that they merge. If you're using something like Spring Boot, you don't have to worry about whether your operating team will come along at all. Because then you define in principle the complete complexity only about the software layer and the guys in the company do nothing more than put up boxes. So even my company has three different strategies because we were separated. And that is very different. The company daughter where I am now, we're going in the direction of Kubernetes. So basically learn from the big ones. Google, Netflix and Co and what a management model they have, they are very strong. And you really notice how this old viewpoint, ERP and SAP really rubs off on the new viewpoint: speed. And, we are right in the middle of it. That's why I can tell you so much about what different problem areas exist there. I always say it's a triad: you have the technology, the culture and the organization that you have to take with you. The technology is here. All you have to do is see if you can turn them around and make sure your organization grows with you. And culture must be created. I don't know how you deal with these topics afterward, that you have to create the environment. We notice more and more that this classical environment, in which many of us have learned. We are just realizing that we have to put in an extreme change management to keep up with this.

Presentation is continued and finished.

INT. Now we are done with the presentation. Do you have any questions yet to clarify?

RES. You have put a lot of effort into it, one can tell. You have also combined the classic SAP world with the new Operation Toolings. Well, that's got hands and feet. I can give you some insights from my company. There are basically two currents. One comes from the classic SAP corner. Of course, we have an SAP Competence Center which is responsible for SAP support. They would basically see your proposal as the right one and apply it. And then there is the group of non-SAP operators, who see exactly the opposite approach as the right one. This means that SAP has a major disadvantage. There are several disadvantages with such a large ERP system. It is very difficult to find employees, which can definitely be an exclusion criterion to adopt such a strategy. When you try to find cracks for SAP on the market, it is very difficult, and when you find and hire them, you become poor as a company. The second factor is basically SAP's licensing model. You got a vendor lock-in. And for a company in the e-commerce sector where the collection of pennies is considered to be attractive, then this can definitely be an exclusion criterion. If you choose the approach for Single-Point of Truth, this is the right one. If you are now a company in the e-commerce sector, where you have a lot of different business lines, you would make yourself as much as possible dependent on this small team that works with this system. So everyone's waiting to get through this bottleneck. Now, this is not a scientific consideration, but rather a practical one.

Int. The architecture can not only be implemented with SAP. There could also be another ERP system as the core system. The orchestration functionality is not implemented in the SAP System. The application services, containers or system landscapes are described in such a way that they can be recorded as standardized materials in any ERP system so that the development effort in the ERP system would be mainly a customizing effort. The functionality I need is implemented in this production execution system.

Res. The other trend tends to go in the other direction, saying that basically, the ERP system is the system receiving from the preliminary systems. With all the disadvantages that come with it. That means you would use such an ERP system as a consolidation tool in such a distributed organization. Because then you can also make mergers and acquisitions quite easily. Basically, they use the ERP system as credit in the first step, and then they may use the ERP system as debit. If you look at how fast the company has grown in recent years, and you would choose this approach, from the central ERP system everything outward, you would have no chance to exist in the market, because such migration projects are very expensive and lengthy. And let's say the life cycle of the board members is getting shorter and shorter. The short-term IT strategies are often derived from this, because you are not a member of the company's executive board for thirty years, as you used to be, but rather we have a five-year cycle, and in the five years you have to achieve visible success.

Int. Let's get into the questionnaire now. And here's my first question: How do you assess the increasing standardization and automation in IT?

Res. As urgently needed and the right step.

Int. Then we come to these application services. The interview has two blocks. The first is application services and then, in particular, this application system architecture related to the system landscape, and in the first block, the first question is: The architecture proposes the reuse of configuration models between different application services provided for customers. Does this reuse increase overall efficiency?

Res. Yes.

Int. All operational tasks are recorded in configuration models. Expenses incurred can be directly allocated to these models. Is this helpful in determining the profitability of products?

Res. Yes.

Int. Application services are compiled from configuration models. The development of these models entails costs that can be attributed to the models. This makes it easy to determine the costs for an application system landscape before engineering. Is this determinability helpful?

RES. Yes.

INT. Can you imagine application services that cannot be implemented with architecture?

RES. I don't know of any.

INT. Service levels are used in the architecture to signal to the helpdesk which response times have been agreed or to quantify the agreed service availability. Is it sufficient looking at the current standard practice?

RES. No, not enough, but it is the first step.

INT. What would be the second step?

RES. Depends on the business. For example, if you are dealing with system checks, the service level alone is not decisive. Compliance would be a measurement criterion; customer satisfaction would be a measurement criterion. So a pure SLA view would be too technical.

INT. Two types of payroll accounting are addressed: pay-per-use and fixed price regardless of volume. Are these types of settlement sufficient?

RES. Post-billing and pre-billing are the words used by us. Instant billing is actually becoming more and more popular. If you are dealing with American providers, there is the possibility of instant charging if you want to. The trend towards instant billing. This is actually a mixture of both models. Pay-per-use is always traditionally at the end of the month. Post-billing would basically be a basic fee.

INT. Exactly, that would just be the two extremes that can be mapped with architecture.

RES. Right, one talks more and more about subscriptions and then about usage. These are such common terms which are becoming more and more important. Zurora is also active in this area. That's such an American provider of billing platforms that basically does what you do, but it's very strong in the modern world, but they have very good metrics that are also available in subscriptions. For your market analysis.

INT. In principle, two types of services are distinguished: Engineer-to-order and build-to-order. Are these two types sufficient?

RES. Yes, these are both extremes. There are still interim solutions.

INT. What would interim solutions be?

RES. For example, that you say that in terms of complexity, I can do this with an operator, with a developer, i. e. only a part of it. After all, these are the gradations. The business is always about time-to-market and of course the higher the automation level, the better it is. When the departments can work for themselves. At the moment we proceed as follows, if this procedure has to be done x times, then the whole thing is transferred as a function for the department so that in the future it can be carried out without any dependency on IT. We basically have four to five different levels that you will go through until it makes economic sense to automate this. Well, there are gradations. Something like buying apps from a provider and integrating them.

Int. Well, that could be realized in the engineer-to-order process. So buying and integrating would be especially important?

Res. Exactly, very important. You just mentioned the CMS. You used to build it yourself. Today, you buy this on the market and integrate it. Or quite typically in the environment of ERP systems, is a tax engine for tax calculations. There are five to six in the world; they're very good. You can easily integrate them into large ERP systems.

Int. Is the automated provision and operation of application services as described practicable and meaningful?

Res. Yes. And necessary.

Int. Okay, then we come to the information system architecture. The application system architecture of the information system provides an ERP system as the central leading system. Here, the application services are not only held as sales products but are represented in their complete structure. Does this complete storage make sense?

Res. Not in my opinion. Because separation of concerns speak against it. So there must be an adjustment of the systems. But not a single point of truth in the ERP system.

Int. Does this mean that the structure information does not have to be in the ERP system?

Res. It doesn't have to originate there. They can be stored in it and should also be stored there. But the origin and the change would not have to take place in the ERP system.

Int. So you could imagine, for example, that a developer somehow adjusts his configuration in a version management system and that it is then compared with the IT service management system. Ok, but having this structural information in the ERP system would make sense.

Res. Yes, alone for control. Controlling functions and checking functions are very important. If you have stored the information in too many distributed locations, the internal control system escapes you.

Int. What about modeling landscapes [with the ASL Modeller]?

Res. Here the [ASL Modeler] could also draw information from different sources. You would just go forward with mergers and acquisitions, and then you would buy a system, which the other daughters would like to have. You wouldn't be able to do that if you maintained everything in a central system. This can, of course, have an influence on your IT system structure. If you have too many bottlenecks in terms of development resources, you can shut down the entire corporation.

Int. Is this especially true for your company or for ASLPs?

Res. Well, the fintechs are doing the same thing now. You're losing this time-to-market thing. You have to develop a system like this evenly, and if a partner doesn't go along with it. Then you have your old system, your new system and you have to keep it in sync. Something like Docker helps a lot because you can maintain the old environment for the partner who doesn't move with you and suggest that the system is still running in an old environment, but it's actually embedded in the new environment.

INT. And classic ASLPs?

RES. So, Zuora is basically something like that, but they can only run in the cloud. On-premise doesn't work for them. There are also providers that can be hosted in their own data center. However, the respective data center operators must be very mature.

INT. When terminating application services, the customer can be provided with copies of the IaaS instances. Is this service sufficient upon termination of the contract or would it have to be extended?

RES. How is it provided?

INT. As a hard disk copy.

RES. No, we have had better experiences if one party has the source code deposited with a lawyer. If the supplier files for bankruptcy or closes the doors?

INT. Yes, simply when the customer decides to switch to another provider.

RES. In this case, we have had very good experiences with the fact that we get a kind of hit bundle. This means the deployed artifact plus the source code, so that in case of doubt if the business relationship is no longer valid, we can continue to work on the code ourselves. Including the code developed by the provider.

INT. And the data?

RES. 90% of the data is hosted on-premise. In other words, the data is already in our computer center and not at the partner's site. That's why the data would be available anyway. If your partner becomes insolvent, you want to ensure the continuity of your business. And you can only do that if you get the source code and deployable artifacts on a regular basis so that you can step in if necessary.

INT. You want to have the configuration models, for example?

RES. Yes, and providers are also willing to get involved. I don't know about SAP, but suppliers like Zuora are all ready to do so.

INT. The concept requires the entire infrastructure layer of the IT stack to be provided by IaaS software. Can this requirement be met?

RES. Yes.

INT. The application systems of the information architecture essentially comprise three types of application systems: Enterprise Management, IT Service Management, and

IT Service Production Systems. Security & Audit, Systems Engineering, and Monitoring & Analysis Systems are other systems that are not in the focus. Does this selection correspond to the most critical systems of ASLP?

RES. Yes.

INT. And then there are two general questions: Would you be an ASLP and would you have the means to implement the information system architecture?

RES. Yes, parts of it. Not completely. For me, the ERP system would not necessarily be the master for everything. The functions would be made up of individual systems, i. e. more modules, more in the direction of microservices.

INT. Does the proposed information system architecture increase the quality and efficiency of creating application system landscapes?

RES. Yes, absolutely.

C.6.2 Interview 2

Presentation is started and then is paused at slide 2.

RES. Processes are not standardized?

INT. Yes, also processes. But we will see this in the following. I define a production process that involves several phases.

Presentation is continued and finished.

INT. We would now have the opportunity to clarify some further questions.

RES. I have to recapitulate what you told me. It is about this: There is a service provider somewhere who wants to make systems available to some customers so that they can do their work on these systems. That is the basic requirement. Did I understand that correctly? Usually, you start and say: what do you want? We'll virtualize it. Aha, you need this and that, this size, of computing sizes, of RAM[2] sizes of whatever. And then an appropriate virtual system is put together, and then there are price lists, or not and then some prizes are played at dice. Here's what you've done. You have created an ERP system, in this case, an SAP system, where I can enter what I would like to use. This system is based on materials and configuration options. Well, that goes so far with old-school. I would say. I enter this, and the service is automatically selected, which position do I need for it, the price is selected, and it is automatically and, that I have not understood yet, in the technology [data center] this system environment is made available because it does not exist before that.

INT. Exactly.

RES. Uh-huh, that means they have an interface from their configuration system to the data center. There's physics [computing equipment] there. So this is virtualized, but

[2]Random-access memory (RAM)

at the end of the day, physics is what accommodates virtualization. But then you have a configuration tool where you can use this configuration, which you have just put together via the ERP system, where it is then made available to the customer.

Int. Exactly. There are different characteristics. On the one hand, this standardized case where I offer very standardized services, e. g. also via a webshop, to the customer configurable. Or very individual solutions where you sit together with the customer or the customer can draw something by himself to create more individual solutions. And the entire process, this creation process is also mapped in the ERP system, including project management, in order to cover the costs of developing such configuration models, because this is the only way necessary to provide such a service if you already have the application software. So, for example, if you buy SAP software, you only need the corresponding configuration models. These are created. They automate the normal operation of these applications.

Res. But this is only the case if they provide the virtual server, the operating system, i. e. the middleware. The application itself is not yet. Cause they don't even know what's on it.

Int. That's what I implied with [Covering the IT stack with configuration models]. The software configuration models can be arbitrarily complex. Of course, you can easily install packages, for example. For example, an OpenERP system can be installed. But you can of course also use the interfaces from SAP ERP via RFCs[3], via BAPIs[4] with which you can also possibly start any eCATTs[5] with which you can also start the application, of course only to a certain standardized degree, which you have already defined in advance. So only what you have previously defined in the configuration models can be done in the systems. But one could also imagine, for example, that these configuration models for SAP ERP take on certain customizing tasks in the application.

Res. But then, of course, they have to know that. And they must already have SAP ERP installed. I don't know of any case where we in our data center when we get new procedures, where we can automate that sort of thing. Because each procedure is different from the other. Well provided we offer SAP as an ERP system and the customer can configure it himself. You're right; we have the software. Some steps can be automated. But if I don't know what the customer wants to run on these machines, then I can't automate anything.

Int. That's right. If you first have to determine what the customer wants, and then if necessary also carry out an in-house development. Then, of course, it is not possible to automate this beforehand. Of course, this is only possible if you roll out the application service or parts of it to many customers. If, for example, you offer many different SAP ERP landscapes, which are similar to each other between customers,

[3]Remote functional call (RFC)
[4]Business application programming interface (BAPI)
[5]Extended computer aided test tool (eCATT)

but you provide many landscapes and of course also offer consulting services. But for me, the focus is now really on standardized system provisioning and operation.

Res. With us, this is already automated. Parameters are passed, and then the virtual machine is configured. But we have no connection to any ERP system. We don't have a billing from there either. It's all running parallel.

Int. I will start the interview now if you don't have any more questions. The starting question is: How do you assess the increasing standardization and automation in IT?

Res. Open answer? How do I rate this? It will increase or become more important. And it will become extremely important because of demographics and shrinking human resources.

Int. Are you nostalgic?

Res. Well, you have to see the reasons why. You don't do it that way because you like to do it, but there are reasons for it. First of all, you have to cut costs because everything is becoming more complex and expensive. Automation is one way of keeping costs down. On the other hand, there's something lacking in qualified personnel.

Int. I would then go into the questions now. That's three blocks of questions. Firstly, about the application services that can be provided, the information system architecture and two final questions. The first question of the first block would be now: The architecture suggests the reuse of configuration models between different application services provided to customers. Does this reuse increase overall efficiency? This is only about the general ASLP case, not your company in particular.

Res. Yes.

Int. All operational tasks are recorded in configuration models. Expenses incurred can be directly allocated to these models. Is this helpful for determining the profitability of products, i. e. application services?

Res. Yes.

Int. Application services are compiled from configuration models. The development of these models entails costs that can be attributed to the models. This makes it easy to determine the costs for an application system landscape before engineering. Is this determinability helpful?

Res. Yes.

Int. Can you imagine application services that cannot be implemented with architecture? We have seen these three example services. Do you also have services that you would not be able to implement?

Res. It applies to software that is not preconfigured. That needs to be freshly installed.

Int. Service levels are used in the architecture to signal to the helpdesk which response times have been agreed or to quantify the agreed service availability. Is it sufficient to look at the current standard practice?

RES. Of course, I have the User Help Desk in SLAs. As far as reaction time is concerned. Possibly also concerning the solution time. But the SLAs also show how much memory I use and how quickly I have a recovery. Whether I have 99.8% availability and such things. This has nothing to do with the User Help Desk itself.

INT. Well, you would like to see more criteria.

RES. So availability is absolutely important.

INT. The agreed service availability with the customer, we have it with us. And then these reaction times, if any errors occur. How fast the service desk has to react.

RES. The recovery time is also partly important.

INT. Two types of payroll accounting are addressed: pay-per-use and fixed price regardless of volume. Are these types of settlement sufficient?

RES. There are only these two. I don't know a third possibility.

INT. In principle, two types of services are distinguished: Engineer-to-order and build-to-order. Are these two types sufficient?

RES. There is no other way.

INT. There would be some possibilities in between.

RES. Yes, of course. It always depends on the customer, with upper and lower limit. But from the principle, there are only these two possibilities. Either it is individual, or it is just standardized where it is clear how much effort has been calculated.

INT. Is the automated provision and operation of application services as described practicable and meaningful?

RES. Yes, it is.

INT. Then we come to the application system architecture. The first question is: the application system architecture of the information system provides an ERP system as the central leading system. Here, the application services are not only held as sales products but are represented in their complete structure. Does this complete storage make sense?

RES. Yes.

INT. When terminating application services, the customer can be provided with copies of the IaaS instances. Is this service sufficient upon termination of the contract or would it have to be extended?

RES. In my opinion, it is not sufficient because he cannot read it. It's no use to him. This has to be presented differently so that he can read the information in a formatted way.

INT. Ok, so it would have to be provided in CSV[6] dumps, for example.

RES. Yes, or old-school, microfiche or CDs[7] that work with index. As far as I'm concerned, if I have a booking list, then I can call up a booking list without having only the data. If I have an ERP, then I have to have all the data that has to be kept, I have to have it and be able to reproduce it. In the accounting department, I must have a list of bookings: Accounts receivable, accounts payable, open items and everything I have to keep for at least ten years. And some documents. It is quite possible that I have an archiving system or stored my invoices in the ERP system. In the future, when I operate with e-invoice, I will have XML files in the background anyway. These must also be archived and must be readable. And I don't just need a file with bits and bytes stored as they are. I just have to make it readable with an application.

INT. Of course, one could imagine that the customer would then start this image and extract the data himself.

RES. Yes, that works in theory. But there is no economic solution to this, I know that, too. But that's a problem for me. If I were a customer, I would pretend to need that and that kind of information, and I want it to be readable. If I no longer wish to operate the service.

INT. The concept requires the entire infrastructure layer of the IT stack to be provided by IaaS software. Can this requirement be met?

RES. I can't answer this question.

INT. The application systems of the information architecture essentially comprise three types of application systems: Enterprise Management, IT Service Management, and IT Service Production Systems. Security & Audit, Systems Engineering, and Monitoring & Analysis Systems are other systems that are not in the focus. Does this selection correspond to the most critical systems of ASLP?

RES. Yes.

INT. Ok, and now in general: If you were an ASLP and would have the appropriate means, would you implement the information system architecture?

RES. Yes. For very specific procedures. Our company doesn't have that. I can't say anything. Well, but such a generator, especially if it is also linked to an ERP system, where the order is then automatically generated. That'd be good. I know that our people in the datacenter do the calculation how expensive such a service, or only the provision of a virtual server, what this actually costs. They'd be happy to have a generator like that. Because they are also poking around in the fog and looking here and there and considering where they get the cost allocations. Because there are many departments and groups involved. This starts with the fact that the space rent must be there, that the frames must be there, that the climate is

[6]Comma-separated values (CSV)
[7]Compact disc (CD)

provided, that the chassis is provided. Where the physical servers end up.

INT. Does the proposed information system architecture increase the quality and efficiency of creating application system landscapes?

RES. Yes. If it is standardized, yes.

C.6.3 Interview 3

Presentation is started and finished.

INT. How do you assess the increasing standardization and automation in IT?

RES. Well, first of all... I like automation and standardization. That makes sense, too. They are in principle the means to stay on top of increasing demands and complexity. This is the second side of the coin. On the one hand, I have a lot more requirements with regard to what IT should do. There are always new things coming out, big topics like blockchain or Industry 4.0, whatever. Requirements for flexibility, agility or availability also arise from major social trends... In other words, from all criteria that can be applied in this way. So digitization is permeating life more and more. And in order to be able to make this tangible at all or to make it reproducible in some way, one always tries to automate and standardize. These are the shovels of IT to keep everything under control. I can't do without it. And it is in principle the daily struggle with the more. It's about keeping up. If I don't have standards and do everything individually, then this goes beyond the possibilities of the human mind and resources in any company or organization. Those who don't manage to automate that will either become incredibly expensive, or they won't be able to do what they are asked to do. And these are the two consequences if I can't keep up with standardization and automation. That's the point. This can also be seen in many examples. This is not only a technological issue but also an organizational one. This is diverse and not easy to do in IT organizations, but if you can't do that, you can't meet the requirements, or it's going to be incredibly expensive.

INT. Okay, that was the opening question. Then we now come to the application services. The first question is: the architecture suggests reusing configuration models between different application services provided to customers. Does this reuse increase overall efficiency?

RES. The reuse of configuration models. If the configurations are designed or flexible in terms of deployment so that they can be adapted to the various scenarios, then, of course, this is possible. It's just a question of how far this is given.

INT. So depending on how versatile the configuration models are.

RES. Basically, yes.

INT. All operational tasks are recorded in configuration models. Expenses incurred can be directly allocated to these models. Is this helpful in determining the profitability of products?

RES. Sure, that sounds logical to me. Profitability is how much I put in, how much I get out. And that is in principle a grasping of the investment. So what initial effort I have to create the model and then how much does it cost me to improve the model, increase the maturity level, optimize it in relation to how often I can use it and how often I benefit from it.

INT. Are there any configuration models in your company that can be reused?

RES. Yes, we have a configuration model, for example, in terms of deploying VMs from deployment scenarios. We have a deployment system where we can say template-based and parameterizable about a file, this is now an exclusive S4Hana system, that the configuration for the application server and for the database … no, not the database because it is not a VM. But everything that is standard in VM[8] deployment for the different deployment scenarios or for the different applications we deliver, we have for example such configuration models.

INT. Application services are compiled from configuration models. The development of these models entails costs that can be attributed to the models. This makes it easy to determine the costs for an application system landscape before engineering. Is this determinability helpful?

RES. Sure. It would definitely be. Because we also have the problem that the scenarios are becoming more complex. Or the variety of application variants increases and then the question of pricing is always the question. And if this is modularly structured and I can take a pricing from existing configurations, then that is of course good.

INT. Can you imagine application services that cannot be implemented with architecture?

RES. Well, the more difficult it gets, the higher the specialized requirements are. At the end of the day, I have to invest quite a lot in such configuration models and, for example, the systems I need for them. So the production execution system and the information systems. So this is all the more profitable, the more standardization I can accommodate. And whether or not I can accommodate this is something that is usually decided not by me, but by any customer who has requirements. And then according to how scalable it is, so a customer has the requirement or have n customers the requirement. The fewer variants and the more scaling, the more profitable it becomes, of course. Of course, the more efficiency benefits I get from such a system.

INT. Ok. Service levels are used in the architecture to signal to the help desk which response times have been agreed upon or to quantify the agreed service availability. Is it sufficient looking at the current standard practice?

RES. No, these are the two most important ones. Availability and response time are the two most important service levels. But in practice, of course, there is much more. The question is whether this will lead to more individualization and less

[8]Virtual machine (VM)

standardization at service levels. There is response time behavior. SLAs will then certainly also be recourse situations. And corresponding recovery situations. How quickly and in which case the system must be available again. Not only response times, but also time to solution. There's something like that. That is relative, it is very hard, very expensive because almost no one can guarantee that. There are many more, but these are the two most important ones. Yeah.

INT. Ok, then the next question: Two types of payroll accounting are addressed: pay-per-use and fixed price regardless of volume. Are these types of settlement sufficient?

RES. Well, I'll say, that's a matter of opinion as to whether that's sufficient. That's enough for my idea, but there are already plenty of examples of cloud service providers, so all you have to do is look at Amazon with their calculator, which integrates the wildest forms of incentive into such price models. There are many gradations, but in my opinion, this is sufficient to offer the two variants.

INT. In principle, two types of services are distinguished: Engineer-to-order and build-to-order. Are these two types sufficient?

RES. These are the two basic types. So either I have a fixed variance, which I can query standardized via such mass customization and then deliver automatically or I say ok there are special requirements. Or its requirements are not just as represented in zero and one; then I need exactly the such a process in which I have to discuss all configuration items. Of course, this is much more complex, and the question always consists of... I build up my own configurations from these results of the process and how complex it is. Will there ever be a second one that uses the same patterns again, or is it actually an individual process that then shuffles along standards. Of course, I have to be careful that in the Engineer-to-order process I have more new variance possibilities in the individual configuration models. The question is whether my automation can handle it, in the end, i. e. whether my Execution Order is still able to execute it just because I have introduced some new things or whether I need a new Execution Engine. That's where the variance can blow up my entire system. A normal cloud provider does not allow this. He has a variance. No matter how big it may be. It may still be varied, but the catalog. But he just won't let me define anything outside of the system.

INT. Okay, then the last question in this block. Is the automated provision and operation of application services as described above practicable and meaningful?

RES. Depends on who, of course.

INT. For the ASLPs.

RES. As I said, it makes more sense the more I automate things... For example, with desktop as a service, it's such a model where I can well imagine... So I think the model makes sense if I can handle an absolute majority of my things through this build to order process. If I can set up the system in such a way that I can run 90%

+ x percent of my cases over this build to order process, i. e. just run it in a highly automated way, then this makes sense. However, if I as a landscape provider build the system and more than 50% are always involved in engineering processes, I think it is no longer practicable. Then it's too much overhead. Of course, it also depends on the size of the company. Of course, it depends on the size and if this is completely different now.

INT. So an idea is of course that the solution designer also checks during the engineering process in the catalog to see if there are already corresponding models so that only non-existent models need to be newly implemented.

RES. Yes, but this, of course, presupposes a standardization of things. So a database can now be considered as a configuration on the higher level, but of course, it is already very complex. And whether my application, which should then use the application, does not have special requirements, e. g. with regard to the tablespaces, as far as I'm concerned. That's another one. The more powerful these configurations are, the more expert knowledge and know-how I need in these configurations. It's going to be expensive. If you look at the automated cloud service providers that already exist. If you look at web hosts, for example. So since ten years, there are such packages where I have a Wordpress or where I have a MySQL or such things. Many providers are able to provide such a set with a configuration with a corresponding user interface and access options depending on what packages I buy. The next level up is that I have some kind of web content system on MySQL, so I can use it more easily, where I don't have anything to do with MySQL. These are then always standards. Exactly this case. That it is highly scalable, that there are not only 10 users, but also thousands of users who use it. I can put a lot of know-how into it, that MySQL works everywhere or is configured in packages. And then it's just that there is hardly anything over the complexity of such web server applications. Well, it doesn't really go beyond that. Even with more complex database systems, it ends. I don't know, MongoDB or anything. But with complex application scenarios, however, this actually ends. You are relatively fast at Software as a service. Where they say ok, you've got no more to do with the whole thing downstairs, just use it. That's that ladder upstairs.

INT. Of course, it is also possible to provide an SAP system automatically.

RES. Okay, but what does that mean? The question at SAP does not stop at the SAP system. I can do that, too. But it's not that the SAP system is running. That's just when the first half step starts. Of course, I can look at an SAP system like a web application server and say ok; I'll start it up for you. An empty system or because of me it's an IDES[9] and do what you want, but that's not real life. All we're talking about is sandbox systems. Exclusively from trying or playing around. I can't get a transport out of there. What about licenses? Especially with such

[9]Internet demonstration and evaluation system (IDES)

complex industrial applications, someone tells me that I'll provide you with the ad hoc solution, I'll provide you with a game system where you start to learn. Which modules are used, what is the license, etc.?

INT. Yes, good, but you can imagine that the modules will then be made available in a configurable way. The software configuration models can also be active at higher levels. For example, you can already carry out a lot of Customizing activities for BAPIs RFCs.

RES. But it quickly becomes so individual. Nobody would automate that. That's being asked twice. This is simply too individual for automation.

INT. Okay, let's get to the next block. The application system architecture of the information system provides an ERP system as the central leading system. Here, the application services are not only held as sales products but are represented in their complete structure. Does this complete storage make sense?

RES. Yes, I can understand that this is how you do it. That also seems to make sense. I think, if this will stand up to real life, then the decision is made... So if I put the system like this on, then that means in a process... the question is always who builds it... is that the question? So if I have only sales and finance people at my ERP system, it's too small for them, so they couldn't live with it. All right, I'm assuming it's automated. So some customer orders this. The question is, how does that work? So what's the order? Who designed the order. So with Build-to-Order, there is a standard form. And from this, everything in the system is automatically turned together. Ok, at the moment when it comes to engineer-to-order I have someone who wants to put it together in the system. And there's the question: is this service engineer or the architect sitting at the ERP system or is he sitting at another system? I can imagine that some of them are not at the ERP system, but that the ERP people really are the people who only aggregate on a higher level. So in principle, I also believe that if you save all of this, it makes it easier to analyze it, if I can, so to speak, show my capacity utilization, if I can also map the corresponding procurement processes. So it makes sense to me, too.

INT. Ok. When terminating application services, the customer can be provided with copies of the IaaS instances. Is this service sufficient upon termination of the contract or would it have to be extended?

RES. That goes even further then what is currently offered. So normally that's how it is when I use a service. Be it VM-based or software based. The service is gone. It's rare that I still have access to anything in the follow-up. You don't really find that. But on the contrary, these service providers actually close, as I said before, whichever level, is actually always the responsibility of the customer and for that something secured or wants to reuse it, that he himself is responsible for downloading it. Be it that he's doing a VM backup, be it that he's getting things out of some APIs. So with software as a service, it is usually the case that APIs are available to make backups or local backups. If there is. Perfect is when it doesn't exist. Then there's a vendor lock-in, and I'll never get it out of there. So that's more than usual. That's unusual today. This is extra.

Int. Ok, The concept requires the entire infrastructure layer of the IT stack to be provided by IaaS software. Can this requirement be met?

Res. Yes.

Int. The application systems of the information system architecture essentially comprise three types of application systems: Enterprise Management, IT Service Management, and IT Service Production Systems. Security & Audit, Systems Engineering, and Monitoring & Analysis Systems are other systems that are not in the focus. Does this selection correspond to the most critical systems of ASLP?

Res. Yes, I wouldn't think of much more than that now.

Int. Okay, let's get back to the general block. Would you be an ASLP and would you have the means to implement the information system architecture?

Res. Yes, something like that would be introduced, yes.

Int. Ok, something like that?

Res. Well, I think. The triangle of the three systems. That's logical, that's the way it is. The question I think is, on the one hand, how do I integrate these systems with each other, because, for example, if I use Solution Manager as ITSM, then I have Hyper-V as a service production system, for example. So the three kinds of systems are there, and those are the ones I need. The distances between the systems are relatively long. And that is the question which one I consider to be the leading system for me. I think that's a philosophical question. So if I see this from a commercial point of view, you always see the ERP system as the leading system. If I see this from an internal IT point of view, it's probably more like the IT production system that is leading for me. So this is such a philosophy question which is important for me and where I start. When I compare this in our company, we have just introduced the ERP system. We have the service desk. There is no de-facto integration yet. And I'm telling you, this execution system is not one thing. It is not the execution, but the production system, which is what many systems are. Take Amazon, for example. They have exactly such systems. When an order comes in, the individual systems are activated. These are exactly the same systems as in your concept. The question is - and this is always the point, I design it from top to bottom or from bottom to top. The goal is to have a desktop as a service, with the variants of what I have for a Windows version, etc. and then I start thinking about it and build it up. Or is there always a service-connected first and then we consider how to grow and get a structure. But in principle, such a network is needed. But it is probably true that each of these landscape service providers has a different character or a different focus. I think you'll find a lot of opinions there.

INT. Does the proposed information system architecture increase the quality and efficiency of creating application system landscapes?

RES. Yes, if I do this, then I am more standardized and generally have higher documentation... and traceability is always such a point. Especially when I have implemented ITSM and the processes that make everything comprehensible, which is just such a quality criterion. Or make appropriate audits possible, or release processes enable such a thing for changes.

Bibliography

Adam, M. T. P., Gimpel, H., Maedche, A., and Riedl, R. (2017). Design blueprint for stress-sensitive adaptive enterprise systems. Business & Information Systems Engineering, 59(4):277–291. (Cited on page 103.)

Ahmad, S., Schroeder, R. G., and Mallick, D. N. (2010). The relationship among modularity, functional coordination, and mass customization: Implications for competitivenes. European Journal of Innovation Management, 13(1):46–61. (Cited on pages 4, 57, and 78.)

Akkermans, H. and Van der Horst, H. (2002). Managing IT infrastructure standardisation in the networked manufacturing firm. International Journal of Production Economics, 75(1):213–228. (Cited on page 1.)

Alt, R., Auth, G., and Kögler, C. (2017). Innovationsorientiertes IT-Management – Eine Fallstudie zur DevOps-Umsetzung bei T-Systems MMS. HMD – Praxis der Wirtschaftsinformatik, 54(2):216–229. (Cited on pages 30, 31, 32, 44, 53, 54, and 57.)

Alter, S. (2011). Metamodel for service design and service innovation: Integrating service activities, service systems, and value constellations. In Galletta, D. F. and Liang, T.-P. (eds.), Proceedings of the 32nd International Conference on Information Systems, Shanghai, China, pages 1–20. (Cited on pages 27, 35, and 44.)

Andenmatten, M. (2017). IT4IT$^{\text{TM}}$ – das agile Betriebskonzept der IT der Zukunft. HMD – Praxis der Wirtschaftsinformatik, 54(2):261–274. (Cited on pages 17, 18, and 20.)

Andersen-Gott, M., Ghinea, G., and Bygstad, B. (2012). Why do commercial companies contribute to open source software? International Journal of Information Management, 32(2):106–117. (Cited on pages 1, 15, 32, and 44.)

Anderson, E., Lam, L.-l., Eschinger, C., Cournoyer, S., Correia, J. M., Wurster, L. F., Contu, R., Biscotti, F., Liu, V. K., Eid, T., Pang, C., Swinehart, H. H., Yeates, M., Petri, G., and Bell, W. (2013). Forecast overview: Public cloud services, worldwide, 2011-2016, 4q12 update. Technical Report G00247462, Gartner, Inc. (Cited on pages 3 and 113.)

Anderson, P. (1994). Towards a high-level machine configuration system. In Proceedings of the 8th Large Installation System Administration Conference, Berkeley, CA, USA, pages 19–26. (Cited on pages 14 and 39.)

Apel, S., Batory, D., Kästner, C., and Saake, G. (2013). Feature-oriented software product lines. Springer, Heidelberg et al. (Cited on pages 38 and 39.)

Arcangeli, J.-P., Boujbel, R., and Leriche, S. (2015). Automatic deployment of distributed software systems: Definitions and state of the art. Journal of Systems and Software, 103(1):198–218. (Cited on page 38.)

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M. (2010). A view of cloud computing. Communications of the ACM, 53(4):50–58. (Cited on pages 21, 35, and 85.)

Ballestrero, S., Batraneanu, S. M., Brasolin, F., Contescu, C., Girolamo, A. D., Lee, C. J., Astigarraga, M. E. P., Scannicchio, D. A., Twomey, M. S., and Zaytsev, A. (2014). Design and performance of the virtualization platform for offline computing on the ATLAS TDAQ farm. Journal of Physics: Conference Series, 513(3):1–5. (Cited on page 39.)

Bandara, W., Furtmueller, E., Gorbacheva, E., Miskon, S., and Beekhuyzen, J. (2015). Achieving rigor in literature reviews: Insights from qualitative data analysis and tool-support. Communications of the Association for Information Systems, 37(1):154–204. (Cited on page 10.)

Baun, C., Kunze, M., Nimis, J., and Tai, S. (2011). Cloud computing: Web-basierte dynamische IT-services. Springer, Berlin and Heidelberg. (Cited on pages 16 and 21.)

Beck, K. (2002). Test-driven development: By example. Addison-Wesley, Boston. (Cited on pages 30 and 34.)

Becker, J., Poeppelbuss, J., Venker, D., and Schwarze, L. (2011). Industrialisierung von IT-Dienstleistungen: Anwendung industrieller Konzepte und deren Auswirkungen aus Sicht von IT-Dienstleistern. In Bernstein, A. and Schwalbe, G. (eds.), Tagungsband der 10. Internationale Tagung Wirtschaftsinformatik, Zurich, Switzerland, pages 345–354. (Cited on pages 2, 15, 20, 21, 43, and 56.)

Becker, J. and Schütte, R. (2004). Handelsinformationssysteme. Redline Wirtschaft, Frankfurt am Main. (Cited on pages 24 and 43.)

Beimborn, D., Joachim, N., and Weitzel, T. (2012). Do service-oriented IT architectures facilitate business process outsourcing? Zeitschrift für Betriebswirtschaft, 82(4):77–108. (Cited on pages 2 and 35.)

Bell, S. C. and Orzen, M. A. (2010). Lean IT: Enabling and sustaining your lean transformation. Taylor & Francis Group, New York. (Cited on page 2.)

Berger, T. G. (2005). Konzeption und Management von Service-Level-Agreements für IT-

Dienstleistungen. PhD thesis, Technische Universität Darmstadt. (Cited on page 69.)

Bergeron, B. (2002). Essentials of shared services. Wiley, Hoboken, NJ, USA. (Cited on pages 13 and 21.)

Bernstein, D. (2014a). Cloud Foundry aims to become the OpenStack of PaaS. IEEE Cloud Computing, 1(2):57–60. (Cited on page 63.)

Bernstein, D. (2014b). Containers and cloud: From LXC to Docker to Kubernetes. IEEE Cloud Computing, 1(3):81–84. (Cited on pages 4, 35, 36, 42, 47, 60, and 62.)

Betz, C. T. (2011). Architecture & patterns for IT. Morgan Kaufmann, Waltham, MA, USA. (Cited on pages 2, 3, 18, 19, 53, and 154.)

Beyer, B., Jones, C., Petoff, J., and Murphy, N. (2016). Site reliability engineering: How Google runs production systems. O'Reilly Media, Beijing et al. (Cited on pages 32, 33, 36, 37, 44, 57, 58, and 88.)

Binz, T., Breitenbücher, U., Kopp, O., and Leymann, F. (2014). TOSCA: portable automated deployment and management of cloud applications. In Bouguettaya, A., Sheng, Q. Z., and Daniel, F. (eds.), Advanced Web Services, pages 527–549. Springer, New York. (Cited on pages 42, 47, and 63.)

BMC, Inc. (2017). Remedy AR system API and integration interfaces overview (C, Java, .NET, web services, email, Ruby, Jython, VB, direct SQL). URL https://communities.bmc.com/docs/DOC17512. Last accessed: March 15, 2017. (Cited on page 20.)

Boehm, B. (1988). A spiral model of software development and enhancement. Computer, 21(5):61–72. (Cited on pages 29, 30, and 82.)

Böhm, M., Koleva, G., Leimeister, S., Riedl, C., and Krcmar, H. (2010). Towards a generic value network for cloud computing. In Altmann, J. and Rana, O. F. (eds.), Proceedings of the International Workshop on Grid Economics and Business Models: Economics of Grids, Clouds, Systems, and Services, Ischia, Italy, pages 129–140. (Cited on page 16.)

Böhmann, T., Junginger, M., and Krcmar, H. (2003). Modular service architectures: a concept and method for engineering IT services. In Sprague, R. H. (ed.), Proceedings of the 36th Annual Hawaii International Conference on System Sciences, Big Island, HI, USA, pages 1–10. (Cited on page 27.)

Bordeleau, F. (2014). Model-based engineering: A new era based on papyrus and open source tooling. In Bordelau, F., Dingel, J., Gerard, S., and Voss, S. (eds.), Proceedings of the 1st Workshop on Open Source Software for Model Driven Engineering, Valencia, Spain, pages 2–8. (Cited on page 87.)

Bosse, S., Splieth, M., and Turowski, K. (2016). Multi-objective optimization of IT service availability and costs. Reliability Engineering & System Safety, 147(1):142–155. (Cited on page 58.)

Botta-Genoulaz, V. and Millet, P. (2006). An investigation into the use of ERP systems in the service sector. International Journal of Production Economics, 99(1):202–221. (Cited on pages 3, 26, 27, and 44.)

Botta-Genoulaz, V., Millet, P.-A., and Grabot, B. (2005). A survey on the recent research literature on ERP systems. Computers in Industry, 56(6):510–522. (Cited on pages 23 and 24.)

Brandon, C. (2016). 6 problems with container technology. URL https://storageos.com/6-problems-with-container-technology-in-the-enterprise/. Last accessed: October 16, 2017. (Cited on page 62.)

Breiter, G. and Behrendt, M. (2009). Life cycle and characteristics of services in the world of cloud computing. IBM Journal of Research and Development, 53(4):3:1–3:8. (Cited on page 81.)

Brenner, M., Schaaf, T., and Scherer, A. (2009). Towards an information model for ITIL and ISO/IEC 20000 processes. In Raz, D., Schulzrinne, H., and Kim, Y.-T. (eds.), Proceedings of the 11th IFIP/IEEE International Symposium on Integrated Network Management, Long Island, NY, USA, pages 113–116. (Cited on pages 17, 19, and 44.)

Brooks, F. P. (1995). The mythical man-month. Addison-Wesley, Boston et al. (Cited on page 134.)

Bumgardner, V. C. (2016). OpenStack in action. Manning Publications, Shelter Island, NY, USA. (Cited on pages 60 and 73.)

Cannon, D. (2011a). ITIL® service operation. The Stationery Office, Norwich, UK. (Cited on pages 50 and 90.)

Cannon, D. (2011b). ITIL® service strategy. The Stationery Office, Norwich, UK. (Cited on pages 14, 15, 17, 56, 68, 70, and 83.)

Cardoso, J., Bostrom, R. P., and Sheth, A. (2004). Workflow management systems and ERP systems: Differences, commonalities, and applications. Information Technology and Management, 5(3):319–338. (Cited on pages 34 and 145.)

Chang, M., He, J., and Castro-Leon, E. (2006). Service-orientation in the computing infrastructure. In Chi, C.-H., Bastani, F., and Xue, X. (eds.), Proceedings of the 2nd IEEE International Symposium on Service-Oriented System Engineering, Shanghai, China, pages 27–33. (Cited on page 4.)

Charland, A. and Leroux, B. (2011). Mobile application development: web vs. native. Communications of the ACM, 54(5):49–53. (Cited on page 1.)

Chen, I. J. and Popovich, K. (2003). Understanding customer relationship management (CRM) people, process and technology. Business Process Management Journal, 9(5):672–688. (Cited on page 23.)

Chen, R., Kraemer, K. L., and Sharma, P. (2009). Google: The world's first information utility? Business & Information Systems Engineering, 1(1):53–61. (Cited on pages 32 and 44.)

Chlebusch, A. (2016). Microservice Architekturen – eine systematische Literaturanalyse. Bachelor thesis, Otto-von-Guericke-Universität Magdeburg. (Cited on page 33.)

Conger, S. (2010). From the special issue editor: Servitizing IT. Information Systems Management, 27(2):100–102. (Cited on page 1.)

Currie, W. L. and Seltsikas, P. (2001). Exploring the supply-side of IT outsourcing: Evaluating the emerging role of application service providers. European Journal of Information Systems, 10(3):123–134. (Cited on pages 3, 16, 17, 43, and 59.)

Cusumano, M. (2010). Cloud computing and saas as new computing platforms. Communications of the ACM, 53(4):27–29. (Cited on page 1.)

Dargahi, J. and Najarian, S. (2004). Human tactile perception as a standard for artificial tactile sensing – a review. The International Journal of Medical Robotics and Computer Assisted Surgery, 1(1):23–35. (Cited on page 1.)

Davenport, T. H. (2000). Mission critical: Realizing the promise of enterprise systems. Harvard Business School Press, Boston. (Cited on pages 23 and 154.)

Debois, P. (2011). DevOps: A software revolution in the making? Cutter IT Journal, 24(8):3–5. (Cited on page 32.)

Deboosere, L., Vankeirsbilck, B., Simoens, P., Turck, F. D., Dhoedt, B., and Demeester, P. (2012). Cloud-based desktop services for thin clients. IEEE Internet Computing, 16(6):60–67. (Cited on page 111.)

Delaet, T., Joosen, W., and Van Brabant, B. (2010). A survey of system configuration tools. In van Drunen, R. (ed.), Proceedings of the 24th Large Installation System Administration Conference, San Jose, CA, USA, pages 1–14. (Cited on pages ix, 4, 14, 38, 39, 40, 45, 72, 130, 134, and 140.)

Dernbecher, S., Beck, R., and Toenker, M. (2013). Cloudifying desktops – a taxonomy for desktop virtualization. In Shim, J., Hwang, Y., and Petter, S. (eds.), Proceedings of the 19th Americas Conference on Information Systems, Chicago, IL, USA, pages 1–9. (Cited on page 111.)

Desai, N., Bradshaw, R., and Lueninghoener, C. (2006). Directing change using bcfg2. In LeFebvre, W. (ed.), Proceedings of the 20th Large Installation System Administration Conference, Washington, DC, USA, pages 215–220. (Cited on page 97.)

Dijkstra, E. (1968). The structure of the "THE" multiprogramming system. Communications of the ACM, 11(5):341–346. (Cited on page 34.)

Dikaiakos, M. D., Katsaros, D., Mehra, P., Pallis, G., and Vakali, A. (2009). Cloud computing: Distributed internet computing for IT and scientific research. IEEE Internet computing, 13(5):10–13. (Cited on page 89.)

Disterer, G. (2009). ISO 20000 for IT. Business & Information Systems Engineering, 1(6):463–467. (Cited on pages 13 and 17.)

Docker, Inc. (2017). Compatibility matrix. URL https://success.docker.com/Policies/Compatibility_Matrix. Last accessed: October 16, 2017. (Cited on page 62.)

Dudek, S., Uebernickel, F., and Brenner, W. (2012). Beherrschung von Vielfalt bei IT-dienstleistungen: Adaption und Einsatz der Variantenkonfiguration. In Böhmann, T., Knackstedt, R., Leimeister, J. M., Nüttgens, M., and Thomas, O. (eds.), Tagungsband der Teilkonferenz im Rahmen der Multi-Konferenz Wirtschaftsinformatik: Service Engineering & Management, Norderstedt, pages 27–40. (Cited on pages 26, 28, 29, and 47.)

Dybå, T. and Dingsøyr, T. (2009). What do we know about agile software development? IEEE Software, 26(5):6–9. (Cited on page 18.)

Ebert, C., Gallardo, G., Hernantes, J., and Serrano, N. (2016). DevOps. IEEE Software, 33(3):94–100. (Cited on pages ix, 18, 31, 32, 38, 44, and 120.)

Ebert, N. (2009). Produktionsplanung und -steuerung bei IT-Dienstleistern. PhD thesis, Universität St. Gallen. (Cited on pages 26, 28, 29, 47, 77, 103, and 144.)

Ebert, N., Uebernickel, F., Hochstein, A., and Brenner, W. (2007). A service model for the development of management systems for IT-enabled services. In Hoxmeier, J. A. and Hayne, S. (eds.), Proceedings of the 13th Americas Conference on Information Systems, Keystone, CO, USA, pages 455–462. (Cited on pages 26, 27, 44, and 68.)

Eden, R., Sedera, D., and Tan, F. B. (2012). Archival analysis of enterprise resource planning systems: The current state and future directions. In Huang, M.-H., Piccoli, G., and Sambamurthy, V. (eds.), Proceedings of the International Conference on Information Systems, Orlando, FL, USA, pages 1–19. (Cited on page 24.)

Egyedi, T. (2001). Strategies for de facto compatibility: Standardization, proprietary and open source approaches to Java. Knowledge, Technology & Policy, 14(2):113–128. (Cited on page 1.)

Eilam, T., Kalantar, M. H., Konstantinou, A. V., Pacifici, G., Pershing, J., and Agrawal, A. (2006). Managing the configuration complexity of distributed applications in internet data centers. IEEE Communications Magazine, 44(3):166–177. (Cited on pages 3 and 4.)

Eisenhardt, K. M. (1989). Building theories from case study research. Academy of Management Review, 14(4):532–550. (Cited on pages 8 and 103.)

Eisenhardt, K. M. and Graebner, M. E. (2007). Theory building from cases: Opportunities and challenges. Academy of Management Journal, 50(1):25–32. (Cited on page 106.)

Erbes, J., Nezhad, H. R. M., and Graupner, S. (2012). The future of enterprise IT in the cloud. Computer, 45(5):66–72. (Cited on pages 2 and 13.)

Esteves, J. and Bohórquez, V. W. (2007). An updated ERP systems annotated bibliography: 2001-2005. Communications of the Association for Information Systems, 19(1):386–446. (Cited on page 24.)

Esteves, J. and Pastor, J. (2001). Enterprise resource planning systems research: an annotated bibliography. Communications of the Association for Information Systems, 7(1):1–52. (Cited on page 24.)

European Commission (2005). The new SME definition: User guide and model declaration. Publications Office of the European Union, Luxembourg. (Cited on pages 9, 54, and 158.)

European Commission (2015). Broadband coverage in Europe 2015 – mapping progress towards the coverage objectives of the Digital Agenda. Publications Office of the European Union, Luxembourg. (Cited on page 36.)

Evans, A. and Kent, S. (1999). Core meta-modelling semantics of UML: The pUML approach. In France, R. and Rumpe, B. (eds.), Proceedings of the 2nd International Conference on the Unified Modeling Language: Beyond the Standard, Fort Collins, CO, USA, pages 140–155. (Cited on page 113.)

Everett, G. and McLeod, R. (2007). Software testing: Testing across the entire software development life cycle. Wiley, Hoboken, NJ, USA. (Cited on page 88.)

Färber, F., Cha, S. K., Primsch, J., Bornhövd, C., Sigg, S., and Lehner, W. (2011). SAP HANA database: data management for modern business applications. ACM SIGMOD Record, 40(4):45–51. (Cited on page 3.)

Fehling, C., Leymann, F., Retter, R., Schupeck, W., and Arbitter, P. (2014). Cloud computing patterns: Fundamentals to design, build, and manage cloud applications. Springer, Wien et al. (Cited on pages 36 and 37.)

Feitelson, D. G., Frachtenberg, E., and Beck, K. L. (2013). Development and deployment at Facebook. IEEE Internet Computing, 17(4):8–17. (Cited on page 30.)

Felter, W., Ferreira, A., Rajamony, R., and Rubio, J. (2014). An updated performance comparison of virtual machines and linux containers. Technical Report RC25482 (AUS1407-001), July, IBM Research Division. (Cited on page 62.)

Fettke, P. (2016). Client-server-architektur. In Gronau, N., Becker, J., Kliewer, N., Leimeister, M., and Overhage, S. (eds.), Enzyklopädie der Wirtschaftsinformatik – Online Lexikon. GITO Verlag, http://www.enzyklopaedie-der-wirtschaftsinformatik.de. (Cited on page 34.)

Fettke, P. and vom Brocke, J. (2016). Referenzmodell. In Gronau, N., Becker, J., Kliewer, N., Leimeister, M., and Overhage, S. (eds.), Enzyklopädie der Wirtschaftsinformatik – Online Lexikon. GITO Verlag, http://www.enzyklopaedie-der-wirtschaftsinformatik.de. (Cited on pages 24 and 43.)

Fink, A. (2014). Monolithisches IT-system. In Gronau, N., Becker, J., Kliewer, N., Leimeister, M., and Overhage, S. (eds.), Enzyklopädie der Wirtschaftsinformatik – Online Lexikon. GITO Verlag, http://www.enzyklopaedie-der-wirtschaftsinformatik.de. (Cited on pages 33 and 34.)

Finney, S. and Corbett, M. (2007). ERP implementation: a compilation and analysis of critical success factors. Business Process Management Journal, 13(3):329–347. (Cited on page 103.)

Fischer, J., Knepper, R., Standish, M., Stewart, C. A., Alvord, R., Lifka, D., Hallock, B., and Hazlewood, V. (2014). Methods for creating XSEDE compatible clusters. In Lathrop, S. (ed.), Proceedings of the Annual Conference on Extreme Science and Engineering Discovery Environment, pages 1–5. (Cited on page 38.)

Fischer, T. A., Hirschheim, R., and George, B. (2012). Governance in outsourcing relationships - the role of information technologies. In Huang, M.-H., Piccoli, G., and Sambamurthy, V. (eds.), Proceedings of the International Conference on Information Systems, Orlando, FL, USA, pages 1–9. (Cited on page 18.)

Fitzgerald, B. (2006). The transformation of open source software. MIS Quarterly, 30(3):587–598. (Cited on page 1.)

Fletcher, C., Williams, D. P., and Wurster, L. F. (2016). Magic quadrant for application release automation. Technical Report G00302195, Gartner, Inc. (Cited on page 72.)

Fowler, M. P. (2003). Patterns of enterprise application architecture. Addison-Wesley, Boston. (Cited on page 66.)

Frank, U. (2014). Multi-perspective enterprise modeling: foundational concepts, prospects and future research challenges. Software & Systems Modeling, 13(3):941–962. (Cited on page 27.)

Frank, U., Heise, D., Kattenstroth, H., Ferguson, D. F., Hadar, E., and Waschke, M. G. (2009). ITML: A domain-specific modeling language for supporting business driven IT management. In Rossi, M., Sprinkle, J., Gray, J., and Tolvanen, J.-P. (eds.), Proceedings of the 9th OOPSLA workshop on domain-specific modeling, Orlando, USA, pages 1–8. (Cited on pages 27 and 40.)

Frechette, S. (2011). Model based enterprise for manufacturing. In Duffie, N. A. (ed.), Proceedings of the 44th CIRP International Conference on Manufacturing Systems, Madison, WI, USA, pages 1–6. (Cited on page 2.)

Fuchs, L. and Pernul, G. (2013). Qualitätssicherung im Identity- und Access Management. HMD – Praxis der Wirtschaftsinformatik, 50(1):88–97. (Cited on pages 95 and 97.)

Fuchs, L., Pernul, G., and Sandhu, R. (2011). Roles in information security – a survey and classification of the research area. Computers & Security, 30(8):748–769. (Cited on page 97.)

Gacek, C. and Arief, B. (2004). The many meanings of open source. IEEE Software, 21(1):34–40. (Cited on pages 32 and 44.)

Gartner (2017a). IT glossary: Enterprise applications. URL https://www.gartner.com/it-glossary/enterprise-applications. Last accessed: March 9, 2017. (Cited on page 23.)

Gartner (2017b). IT glossary: ITSSM tools (IT service support management tools). URL http://www.gartner.com/it-glossary. Last accessed: March 9, 2017. (Cited on page 19.)

Gartner (2017c). Newsroom: Gartner says a massive shift to hybrid infrastructure services is underway. URL https://www.gartner.com/newsroom/id/3666917. Last accessed: March 9, 2017. (Cited on page 3.)

Garud, R. and Kumaraswamy, A. (1993). Changing competitive dynamics in network industries: An exploration of sun microsystems' open systems strategy. Strategic Management Journal, 14(5):351–369. (Cited on page 1.)

Gholami, M. F., Daneshgar, F., Beydoun, G., and Rabhi, F. (2017). Challenges in migrating legacy software systems to the cloud - an empirical study. Information Systems, 67(1):100–113. (Cited on page 62.)

Glohr, C., Kellermann, J., and Dörnemann, H. (2014). The IT factory: A vision of standardization and automation. In Abolhassan, F. (ed.), The Road to a Modern IT Factory: Industrialization – Automation – Optimization, pages 101–109. Springer-Verlag, Berlin and Heidelberg. (Cited on pages 2, 41, and 47.)

Gmach, D., Rolia, J., Cherkasova, L., and Kemper, A. (2007). Workload analysis and demand prediction of enterprise data center applications. In Breternitz, M. (ed.), Proceedings of the 10th International Symposium on Workload Characterization, Boston, MA, USA, pages 171–180. (Cited on page 96.)

Goeken, M. and Alter, S. (2009). Towards conceptual metamodeling of IT governance frameworks approach – use – benefits. In Sprague, R. H. (ed.), Proceedings of the 42nd Annual Hawaii International Conference on System Sciences, Big Island, HI, USA, pages 1–10. (Cited on pages 17 and 44.)

Goettsch, N. and Tosse, T. (2013). Digitale produktentwicklung und fertigung: Von CAD und CAM zu PLM. In Kief, H. B. and Roschiwal, H. A. (eds.), CNC-Handbuch 2013 / 2014, pages 562–579. Hanser, Munich. (Cited on pages 24 and 58.)

Goldratt, E. M. (2004). THE GOAL: A process of ongoing improvement. The North River Press, Great Barrington, MA, USA. (Cited on page 93.)

Gómez, J. C. M. (2012). Serviceorientierte architektur. In Gronau, N., Becker, J., Kliewer, N., Leimeister, M., and Overhage, S. (eds.), Enzyklopädie der Wirtschaftsinformatik – Online Lexikon. GITO Verlag, http://www.enzyklopaedie-der-wirtschaftsinformatik.de. (Cited on page 34.)

Grant, R. M. (1996). Prospering in dynamically-competitive environments: Organizational capability as knowledge integration. Organization Science, 7(4):375–387. (Cited on page 2.)

Gregor, S. (2006). The nature of theory in information systems. MIS Quarterly, 30(3):611–642. (Cited on page 10.)

Gronau, N. (2010). Enterprise Resource Planning – Architektur, Funktionen und Management von ERP-Systemen. Oldenburg, Munich. (Cited on pages 24, 25, 43, 50, 75, 85, 95, 117, and 118.)

Grossniklaus, M. and Norrie, M. C. (2002). Information concepts for content management. In Huang, B., Ling, T. W., Mohania, M., Ng, W. K., Wen, J.-R., and Gupta, S. K. (eds.), Proceedings of the 3rd International Conference on Web Information Systems Engineering (Workshops), Singapore, pages 150–159. (Cited on page 112.)

Guay, M., Pang, C., Hestermann, C., and Montgomery, N. (2015). Magic quadrant for single-instance ERP for product-centric midmarket companies. Technical Report

G00272540, Gartner, Inc. (Cited on page 74.)

Gutenberg, E. (1983). Grundlagen der Betriebswirtschaftslehre: Die Produktion. Erster Band. Springer-Verlag, Berlin et al. (Cited on page 1.)

Hall, R., Pauls, K., McCulloch, S., and Savage, D. (2011). OSGi in action: Creating modular applications in Java. Manning, Stamford, CT, USA. (Cited on page 38.)

Hanappi, O., Hummer, W., and Dustdar, S. (2016). Asserting reliable convergence for configuration management scripts. In Visser, E. (ed.), Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications, pages 328–343. (Cited on pages 40, 41, 63, and 105.)

Harrison, A. and van Hoek, R. (2008). Logistics management and strategy: Competing through the supply chain. Prentice Hall Financial Times, Harlow, UK. (Cited on pages 2, 25, 43, and 151.)

Hendricks, E. C. and Hartmann, T. C. (1979). Evolution of a virtual machine subsystem. IBM Systems Journal, 18(1):111–142. (Cited on page 23.)

Herden, S. (2013). Model-Driven-Configuration-Management: Ein modellgetriebener Ansatz für das Konfigurationsmanagement von IT-Systemlandschaften. Springer Vieweg, Wiesbaden. (Cited on pages 40, 45, 58, and 88.)

Hernantes, J., Gallardo, G., and Serrano, N. (2015). IT infrastructure-monitoring tools. IEEE Software, 32(4):88–93. (Cited on page 95.)

Hess, T., Loos, P., Buxmann, P., Erek, K., Frank, U., Gallmann, J., Gersch, M., Zarnekow, R., and Zencke, P. (2012). ICT providers: A relevant topic for business and information systems engineering? Business & Information Systems Engineering, 5(6):367–373. (Cited on pages 13 and 153.)

Hevner, A., March, S., Park, J., and Ram, S. (2004). Design science in information systems research. MIS Quarterly, 28(1):75–105. (Cited on pages ix, 1, 6, 7, 54, 103, 104, 105, 106, and 139.)

Hintsch, J. (2013). ERP for the IT service industry: A structured literature review. In Shim, J., Hwang, Y., and Petter, S. (eds.), Proceedings of the 19th Americas Conference on Information Systems, Chicago, IL, USA, pages 1–9. (Cited on pages 5, 10, 25, 26, 47, and 106.)

Hintsch, J., Görling, C., and Turowski, K. (2015a). Modularization of software as a service products: A case study of the configuration management tool Puppet. In Hinkelmann, K. and Thönssen, B. (eds.), Proceedings of the 2015 International Conference on Enterprise Systems, Basel, Switzerland, pages 184–191. (Cited on page 4.)

Hintsch, J., Görling, C., and Turowski, K. (2016a). A review of the literature on configuration management tools. In Andrade, A. D. and Seymour, L. (eds.), Proceedings of the International Conference on Information Resources Management: Digital Emancipation in a Networked Society, Cape Town, South Africa, pages 1–12. (Cited on pages 6, 11, 38, 39, 45, 72, and 106.)

Hintsch, J., Khan, A., Siegling, A., and Turowski, K. (2017). Application software in cloud-ready data centers: A survey. In Marx Gómez, J., Mora, M., Raisinghani, M. S., Nebel, W., and O'Connor, R. V. (eds.), Engineering and Management of Data Centers: An IT Service Management Approach, pages 261–288. Springer International Publishing, Cham. (Cited on pages 6 and 94.)

Hintsch, J., Kramer, F., Jamous, N., and Turowski, K. (2016b). The application system landscapes of IT service providers: A multi case study. In Li, G. and Yu, Y. (eds.), Proceedings of the 4th International Conference on Enterprise Systems, Melbourne, Australia, pages 122–131. (Cited on pages 6, 9, 18, 20, 48, 49, 50, 51, 106, 151, 154, and 159.)

Hintsch, J., Kramer, F., and Turowski, K. (2015b). ERP systems' usage in the german IT service industry: An exploratory multi-case study. In Hallé, S., Mayer, W., Ghose, A. K., and Grossmann, G. (eds.), Proceedings of the 19th International Enterprise Distributed Object Computing Conference, Adelaide, Australia, pages 169–178. (Cited on pages 6, 9, 18, 19, 53, 106, and 159.)

Hintsch, J., Kramer, F., and Turowski, K. (2018). An information system architecture for build- and engineer-to-order production of application services. Information Systems and e-Business Management, online. (Cited on pages 6 and 159.)

Hintsch, J., Schrödl, H., Scheruhn, H.-J., and Turowski, K. (2015c). Industrialization in cloud computing with enterprise systems: Order-to-cash automation for SaaS products. In Thomas, O. and Teuteberg, F. (eds.), Tagungsband der 12. Internationale Tagung Wirtschaftsinformatik, Münster, Germany, pages 61–75. (Cited on pages 3, 6, and 118.)

Hintsch, J. and Turowski, K. (2013). Towards implementing IT service management in an ERP for the IT service industry. In Grabis, J., Kirikova, M., Zdravkovic, J., and Stirna, J. (eds.), Proceedings of the 6th IFIP WG 8.1 Working Conference on the Practice of Enterprise Modeling, Short Papers, Riga, Latvia, pages 83–94. (Cited on pages 6, 11, 17, 44, and 106.)

Hochstein, A., Ebert, N., Übernickel, F., and Brenner, W. (2007). IT-Industrialisierung: Was ist das? Computerwoche, 15(3):5. (Cited on pages 2 and 20.)

Hochstein, A. and Uebernickel, F. (2006). Operations management and IS: Using the SCOR-model to source make and deliver IS services. In Rodríguez-Abitia, G. and B.,

I. A. (eds.), Proceedings of the 12th Americas Conference on Information Systems, Acapulco, México, pages 32–39. (Cited on pages 15, 82, and 110.)

Hochstein, A., Zarnekow, R., and Brenner, W. (2005). ITIL as common practice reference model for IT service management: formal assessment and implications for practice. In Cheung, W. K. and Hsu, J. (eds.), Proceedings of the International Conference on e-Technology, e-Commerce and e-Service, Hong Kong, China, pages 704–710. (Cited on page 17.)

Hoffmann, B. (2010). Hostvirtualisierung – Vergleich der Konzepte und Produkte. In Helmbrecht, U., Teege, G., and Stelte, B. (eds.), Virtualisierung: Techniken und sicherheitsorientierte Anwendungen – Bericht 2010-04, pages 7–32. Universität der Bundeswehr, Munich. (Cited on pages 35 and 62.)

Hofmann, G. R. (2009). Interview with Bettina Uhlich on 'IT controlling'. Business & Information Systems Engineering, 1(3):266–268. (Cited on page 2.)

Hori, K., Yoshihara, K., and Horiuchi, H. (2007). Customer equipment configuration manager for managed network service providers. In Rodosek, G. D. and Aschenbrenner, E. (eds.), Proceedings of the 10th IFIP/IEEE International Symposium on Integrated Network Management, Munich, Germany, pages 516–526. (Cited on page 39.)

Hsu, P.-F., Ray, S., and Li-Hsieh, Y.-Y. (2014). Examining cloud computing adoption intention, pricing mechanism, and deployment model. International Journal of Information Management, 34(4):474 – 488. (Cited on pages 21, 22, 43, and 149.)

Humble, J. and Farley, D. (2010). Continuous delivery: Reliable software releases through build, test, and deployment automation. Addison-Wesley, Boston. (Cited on pages 18, 30, 58, 88, and 120.)

Hunnebeck, L. (2011). ITIL® service design. The Stationery Office, Norwich, UK. (Cited on page 69.)

ISO/IEC (2011). 20000: Information technology – service management – part 1: Service management system requirements. Second edition. (Cited on page 2.)

ISO/IEC/IEEE (2011). 42010: Systems and software engineering – architecture description. First edition. (Cited on pages 8, 33, 51, 107, 108, 136, and 151.)

ISO/IEC/IEEE (2015). 15288: Systems and software engineering – system life cycle processes. First edition. (Cited on page 22.)

Iveroth, E., Westelius, A., Petri, C.-J., Olve, N.-G., Cöster, M., and Nilsson, F. (2013). How to differentiate by price: Proposal for a five-dimensional model. European Management Journal, 31(2):109–123. (Cited on pages 13, 21, 22, 56, 74, 89, and 110.)

Jennings, B. and Stadler, R. (2014). Resource management in clouds: Survey and research challenges. Journal of Network and Systems Management, 23(3):567–619. (Cited on page 28.)

Jørgensen, M., Dybå, T., and Kitchenham, B. (2005). Teaching evidence-based software engineering to university students. In Lanubile, F. and Seaman, C. (eds.), Proceedings of the 11th IEEE International Software Metrics Symposium, Como, Italy, pages 24–31. (Cited on page 10.)

Kaczmarek-Heß, M. and de Kinderen, S. (2017). A multilevel model of IT platforms for the needs of enterprise IT landscape analyses. Business & Information Systems Engineering, 59(5):315–329. (Cited on page 40.)

Kandogan, E., Haber, E., Barrett, R., Cypher, A., Maglio, P., and Zhao, H. (2005). A1: End-user programming for web-based system administration. In Baudisch, P., Czerwinski, M., and Olsen, D. (eds.), Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology, New York, NY, USA, pages 211–220. (Cited on page 40.)

Kappelman, L., McLean, E., Johnson, V., and Gerhart, N. (2014). The 2014 SIM IT key issues and trends study. MIS Quarterly Executive, 13(4):237–263. (Cited on pages 16, 43, and 59.)

Kastensson, Å. (2014). Developing lightweight concepts in the automotive industry: Taking on the environmental challenge with the SåNätt project. Journal of Cleaner Production, 66(1):337–346. (Cited on page 3.)

Keller, A. and Ludwig, H. (2003). The WSLA framework: Specifying and monitoring service level agreements for web services. Journal of Network and Systems Management, 11(1):57–81. (Cited on pages 69 and 70.)

Kern, T., Kreijger, J., and Willcocks, L. (2002). Exploring ASP as sourcing strategy: Theoretical perspectives, propositions for practice. The Journal of Strategic Information Systems, 11(2):153–177. (Cited on page 9.)

Kim, G., Behr, K., and Spafford, K. (2013). The Phoenix project: A novel about IT, DevOps, and helping your business win. IT Revolution Press, Portland. (Cited on page 2.)

Kirschnick, J., Calero, J. M. A., Wilcock, L., and Edwards, N. (2010). Toward an architecture for the automated provisioning of cloud services. IEEE Communications Magazine, 48(12):124–131. (Cited on pages 41, 47, and 103.)

Kitchenham, B. (2007). Guidelines for performing systematic literature reviews in software engineering: Version 2.3. Technical Report EBSE-2007-01, July, Keele University and

University of Durham. (Cited on page 10.)

Klaus, H., Rosemann, M., and Gable, G. (2000). What is ERP? Information Systems Frontiers, 2(2):141–162. (Cited on pages 2, 9, 23, 33, and 112.)

Koukis, V., Venetsanopoulos, C., and Koziris, N. (2013). õkeanos: Building a cloud, cluster by cluster. IEEE Internet Computing, 17(3):67–71. (Cited on page 37.)

Kræmmergaard, P. and Rose, J. (2002). Managerial competences for ERP journeys. Information Systems Frontiers, 4(2):199–211. (Cited on page 112.)

Kratzke, N. and Quint, P.-C. (2017). Understanding cloud-native applications after 10 years of cloud computing - a systematic mapping study. Journal of Systems and Software, 126(1):1–16. (Cited on pages 36, 37, and 62.)

Kromrey, H. and Strübing, J. (2009). Empirische Sozialforschung: Modelle und Methoden der standardisierten Datenerhebung und Datenauswertung. UTB: Lucius & Lucius, Stuttgart. (Cited on page 139.)

Kvale, S. and Flick, U. (2007). Doing interviews. SAGE Publications, Los Angeles. (Cited on page 9.)

Labes, S., Hanner, N., and Zarnekow, R. (2017). Successful business model types of cloud providers. Business & Information Systems Engineering, 59(4):223–233. (Cited on pages 16 and 43.)

Lacity, M. C., Khan, S., Yan, A., and Willcocks, L. P. (2010). A review of the IT outsourcing empirical literature and future research directions. Journal of Information Technology, 25(4):395–433. (Cited on pages 13, 16, 21, 43, and 59.)

Lainhart, J. W., Oliver, D. J., Andrews, P. G., Antonsson, E. J., Babb, S. A., De Haes, S., Harrison, P., Heschl, J., Johnson, R. D., Pols, E. H., Poole, V. R., and Rafeq, A. (2012). Cobit® 5: A business framework for the governance and management of enterprise IT. ISACA, Rolling Meadows, IL, USA. (Cited on pages 2, 17, and 94.)

Lange, L. (2007). Why ITIL rules. United Business Media, London. (Cited on page 18.)

Laplante, P. A., Zhang, J., and Voas, J. (2008). What's in a name? distinguishing between SaaS and SOA. IT Professional, 10(3):46–50. (Cited on page 34.)

Laudon, K. C. and Laudon, J. P. (2005). Management information systems: Managing the digital firm. Prentice Hall, Upper Saddle River, NJ, USA. (Cited on pages 22 and 23.)

Lebrecht, A. (1991). Die anwendung des CIM-konzeptes auf den DV-betrieb, dargestellt am beispiel der produktionsplanung und -steuerung. In Schwichtenberg, G. (ed.), Tagungsband des 9. GI-Fachgespräch über Rechenzentren: Organisation und Betrieb von

Informationssystemen, Dortmund, Germany, pages 167–189. (Cited on pages 28 and 47.)

Lenhard, J. (2016). Portability of process-aware and service-oriented software. phdthesis, Universität Bamberg. (Cited on page 42.)

Lewis, J. and Fowler, M. (2014). Microservices: a definition of this new architectural term. URL https://martinfowler.com/articles/microservices.html. Last accessed: March 6, 2018. (Cited on pages 36, 37, 44, 54, and 62.)

Leymann, F., Fehling, C., Wagner, S., and Wettinger, J. (2016). Native cloud applications: Why virtual machines, images and containers miss the point! In Cardoso, J., Ferguson, D., Méndez Muñoz, V., and Helfert, M. (eds.), Proceedings of the 6th International Conference on Cloud Computing and Service Science, Rome, Italy, pages 7–15. (Cited on pages 36 and 62.)

Liang, H., Xue, Y., Boulton, W. R., and Byrd, T. A. (2004). Why western vendors don't dominate China's ERP market. Communications of the ACM, 47(7):69–72. (Cited on page 2.)

Lloyd, V. (2011). ITIL® continual service improvement. The Stationery Office, Norwich, UK. (Cited on pages 2, 3, 18, and 53.)

Löffler, M. and Reinshagen, F. (2014). From project to product orientation. In Abolhassan, F. (ed.), The Road to a Modern IT Factory: Industrialization – Automation – Optimization, pages 43–48. Springer-Verlag, Berlin and Heidelberg. (Cited on page 37.)

Logsdon, S. (2015). ansible-roles. URL https://github.com/slogsdon/ansible-roles/blob/master/wordpress/defaults/main.yml. Last accessed: October 6, 2017. (Cited on page 72.)

Magherusan-Stanciu, C., Sebestyen-Pal, A., Cebuc, E., Sebestyen-Pal, G., and Dadarlat, V. (2011). Grid system installation, management and monitoring application. In Dadarlat, V. T. and Mundani, R. P. (eds.), Proceedings of the 10th International Symposium on Parallel and Distributed Computing, Cluj Napoca, Romania, pages 25–32. (Cited on page 60.)

Manco, F., Lupu, C., Schmidt, F., Mendes, J., Kuenzer, S., Sati, S., Yasukata, K., Raiciu, C., and Huici, F. (2017). My VM is lighter (and safer) than your container. In Alvisi, L. and Chen, P. (eds.), Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, pages 218–233. (Cited on page 62.)

Mansfield-Devine, S. (2014). Not coping with change. Network Security, 2014(8):14–17. (Cited on pages 15 and 20.)

Marrone, M., Gacenga, F., Cater-Steel, A., and Kolbe, L. M. (2014). IT service management: A cross-national study of ITIL adoption. Communications of the Association for Information Systems, 34(49):865–893. (Cited on pages 2, 13, 17, 18, 43, and 90.)

Mastelic, T., García, A. G., and Brandic, I. (2016). Towards uniform management of multi-layered cloud services by applying model-driven development. Journal of Systems and Software, 121(1):358–371. (Cited on pages 36, 41, 47, and 103.)

McIlroy, M. (1969). Mass produced software components. In Naur, P. and Randell, B. (eds.), Proceedings of the NATO Science Commitee Conference on Software Engineering, Garmisch, Germany, pages 138–155. (Cited on page 34.)

Mell, P. and Grance, T. (2011). The NIST definition of cloud computing. Technical Report 800-145, National Institute of Standards and Technology. (Cited on pages 4, 16, 21, 22, 35, and 96.)

Menzel, M., Klems, M., Le, H. A., and Tai, S. (2013). A configuration crawler for virtual appliances in compute clouds. In Campbell, R., Lei, H., and Markl, V. (eds.), Proceedings of the 2013 International Conference on Cloud Engineering, Redwood City, CA, USA, pages 201–209. (Cited on page 40.)

Michel, R. (1998). IT services take center stage. Manufacturing Systems, 16(9):44–50. (Cited on page 103.)

Mietzner, R., Leymann, F., and Unger, T. (2011). Horizontal and vertical combination of multi-tenancy patterns in service-oriented applications. Enterprise Information Systems, 5(1):59–77. (Cited on page 81.)

Mobus, G. E. and Kalton, M. C. (2015). Systems engineering. In Mobus, G. E. and Kalton, M. C. (eds.), Principles of Systems Science, pages 699–731. Springer, New York et al. (Cited on page 94.)

Møller, C. (2005). ERP II: a conceptual framework for next-generation enterprise systems? Journal of Enterprise Information Management, 18(4):483–497. (Cited on page 23.)

Momoh, A., Roy, R., and Shehab, E. (2010). Challenges in enterprise resource planning implementation: State of the art. Business Process Management Journal, 16(4):537–565. (Cited on pages 25, 43, 56, and 75.)

Naur, P. (1969). Programming by action clusters. BIT Numerical Mathematics, 9(3):250–258. (Cited on pages 29 and 34.)

Nelson, L. E., Staten, J., and Williamson, K. (2014). State of cloud platform standards: Q1 2014 – OpenStack steps forward to become the new de facto model. URL https://www.forrester.com/report/State+Of+Cloud+Platform+Standards+Q1+%202014/-/E-RES112621. Last accessed: July 30, 2014. (Cited on pages 42 and 72.)

New Relic, Inc. (2015). DevOps without measurement is a fail: How to measure and track the 5 critical drivers of DevOps success. URL https://try.newrelic.com/rs/412-MZS-894/images/AWS_New%20Relic_Measuring%20DevOps%20Success_eBook_062217.pdf. Last accessed: March 6, 2018. (Cited on pages 31, 48, and 53.)

Ng, F., Nag, S., ling Lam, L., Dharmasthira, Y., Eschinger, C., Anderson, R. P., Tornbohm, C., Roth, C., Tramacere, G., Blackmore, D., Wurster, L. F., Contu, R., Biscotti, F., Pang, C., Singh, T., Swinehart, H. H., Montgomery, N., Dominy, M., Petri, G., Kandaswamy, R., Palanca, T., Hare, J., Woodward, A., and Corriveau, J. (2017). Forecast: Public cloud services, worldwide, 2015-2021, 2017 update. Technical Report G00247462, Gartner, Inc. (Cited on pages 22 and 43.)

Ngai, E., Law, C., and Wat, F. (2008). Examining the critical success factors in the adoption of enterprise resource planning. Computers in Industry, 59(6):548–564. (Cited on pages 25 and 43.)

Nielsen, T., Iversen, C., and Bonnet, P. (2011). Private cloud configuration with MetaConfig. In Feig, E., Pu, C., and Goscinski, A. M. (eds.), Proceedings of the International Conference on Cloud Computing, Washington, DC, USA, pages 508–515. (Cited on page 40.)

OASIS (2015). TOSCA simple profile in YAML. Version 1.0 – public review draft 01. (Cited on pages 41 and 47.)

Oestereich, B. and Scheithauer, A. (2012). Analyse und Design mit der UML 2.5: Objektorientierte Softwareentwicklung. Oldenbourg Wissenschaftsverlag, Munich. (Cited on pages 67 and 123.)

OpenStack Foundation (2011). Hypervisor support matrix. URL https://wiki.openstack.org/wiki/HypervisorSupportMatrix. Last accessed: June 15, 2017. (Cited on pages 60 and 75.)

OpenStack Foundation (2012). Heat: OpenStack orchestration. URL https://wiki.openstack.org/wiki/Heat. Last accessed: June 15, 2017. (Cited on pages 42, 47, 62, and 83.)

OpenStack Foundation (2015). Heat-translator. URL https://wiki.openstack.org/wiki/Heat-Translator. Last accessed: February 15, 2018. (Cited on page 42.)

OpenStack Foundation (2017). Companies supporting the OpenStack Foundation. URL https://www.openstack.org/foundation/companies/. Last accessed: March 7, 2017. (Cited on page 72.)

OpenStack Foundation (2018). Virtual machine image guide. URL https://docs.openstack.

org/image-guide/obtain-images.html. Last accessed: January 31, 2018. (Cited on page 72.)

Otto, B. and Schmidt, A. (2010). Enterprise master data architecture: Design decisions and options. In Talburt, J. R., Koronios, A., and Su, Y. (eds.), Proceedings of the 15th International Conference on Information Quality, Little Rock, AR, USA, pages 147–159. (Cited on page 52.)

Owens, D. (2010). Securing elasticity in the cloud. Communications of the ACM, 53(6):46–51. (Cited on pages 13, 21, 28, 37, and 96.)

Pahl, C. (2015). Containerization and the PaaS cloud. IEEE Cloud Computing, 2(3):24–31. (Cited on pages 35, 36, and 113.)

Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. Communications of the ACM, 15(12):1053–1058. (Cited on page 34.)

Peffers, K., Tuunanen, T., Rothenberger, M. A., and Chatterjee, S. (2008). A design science research methodology for information systems research. Journal of Management Information Systems, 24(3):45–77. (Cited on pages 6, 7, 11, and 106.)

Perez, S. (2014). Amazon WorkSpaces, Amazon's cloud desktop service, launches to public along with new sync client. URL http://tcrn.ch/2I32zMV. Last accessed: November 30, 2017. (Cited on page 111.)

Pilgram, U. and Vogedes, A. (2012). Ein Geschäftssystem für ICT-Dienstleister nach industriellen Maßstäben. HMD – Praxis der Wirtschaftsinformatik, 49(5):103—112. (Cited on pages 26, 29, and 47.)

Pink Elephant (2017). PinkVERIFY^{TM} 2011 toolsets. URL https://www.pinkelephant.com/en-us/PinkVERIFY/PinkVERIFYToolsets. Last accessed: March 9, 2017. (Cited on pages 18, 20, and 90.)

Pinnow, A. (2009). Das Rechenzentrum als Produktionsstätte für IT-Dienstleistungen - Kapazitätswirtschaft in virtualisierten Rechenzentren. PhD thesis, Otto-von-Guericke-Universität Magdeburg. (Cited on pages 28 and 47.)

Pop, D., Neagul, M., and Petcu, D. (2014). On cloud deployment of digital preservation environments. In Buchanan, G., Klein, M., Rauber, A., and Cunningham, S. J. (eds.), Proceedings of the 14th IEEE/ACM Joint Conference on Digital Libraries, London, UK, pages 443–444. (Cited on page 39.)

Porter, M. E. (1980). Competitive strategy: Techniques for analyzing industries and competitors. The Free Press, New York. (Cited on page 52.)

Porter, M. E. (1985). Competitive advantage: Creating and sustaining superior performance. The Free Press, New York. (Cited on pages 19, 66, and 81.)

Praeg, C.-P. and Spath, D. (2008). Perspectives of IT-Service Quality Management: A Concept for Life Cycle Based Quality Management of IT-Services. In Cater-Steel, A. (ed.), Information Technology Governance and Service Management: Frameworks and Adaptations, pages 381–407. Information Science Reference, Hershey, NY, USA. (Cited on page 81.)

Prahalad, C. K., Hamel, G., et al. (1990). The core competence of the corporation. Harvard Business Review, 68(3):79–91. (Cited on page 2.)

Prat, N., Comyn-Wattiau, I., and Akoka, J. (2015). A taxonomy of evaluation methods for information systems artifacts. Journal of Management Information Systems, 32(3):229–267. (Cited on pages 105 and 107.)

Prodan, R. and Ostermann, S. (2009). A survey and taxonomy of infrastructure as a service and web hosting cloud providers. In Lu, P. (ed.), Proceedings of the 10th IEEE/ACM International Conference on Grid Computing, Banff, AB, Canada, pages 17–25. (Cited on page 3.)

Puppet, Inc. (2017). puppetlabs/mysql. URL https://forge.puppet.com/puppetlabs/mysql. Last accessed: October 6, 2017. (Cited on page 72.)

Puppet, Inc. (2018). Module fundamentals. URL https://puppet.com/docs/puppet/4.8/modules_fundamentals.html#module-layout. Last accessed: March 6, 2018. (Cited on page 173.)

Rance, S. (2011). ITIL® service transition. The Stationery Office, Norwich, UK. (Cited on pages 18, 20, 38, and 95.)

Rautenstrauch, C. and Schulze, T. (2003). Informatik für Wirtschaftswissenschaftler und Wirtschaftsinformatiker. Springer, Berlin and Heidelberg. (Cited on pages 23 and 34.)

Richter, C. and Schaaf, T. (2011). A maturity model for tool landscapes of IT service providers. In Agoulmine, N., Bartolini, C., Pfeifer, T., and O'Sullivan, D. (eds.), Proceedings of the 12th IFIP/IEEE International Symposium on Integrated Network Management, Dublin, Ireland, pages 1050–1057. (Cited on pages 2 and 27.)

Riemer, K. and Ahlemann, F. (2001). Application service providing – erfahrungsbericht aus sicht eines providers. In Buhl, H. U., Huther, A., and Reitwiesner, B. (eds.), Tagungsband der 5. Internationale Tagung Wirtschaftsinformatik, Augsburg, Germany, pages 743–756. (Cited on pages 3, 16, 17, 43, 57, and 59.)

Rimal, B. P., Jukan, A., Katsaros, D., and Goeleven, Y. (2011). Architectural requirements for cloud computing systems: An enterprise cloud approach. Journal of Grid Computing, 9(1):3–26. (Cited on page 81.)

Romero, D. and Vernadat, F. (2016). Enterprise information systems state of the art: past, present and future trends. Computers in Industry, 79(1):3–13. (Cited on pages 2 and 34.)

Royce, W. W. (1970). Managing the development of large software systems. In Proceedings of the IEEE Western Electronic Show and Convention: Technical Papers, Los Angeles, CA, USA, pages 328–338. (Cited on page 29.)

Ruscio, D. D. and Pelliccione, P. (2014). Simulating upgrades of complex systems: The case of free and open source software. Information and Software Technology, 56(4):438–462. (Cited on page 40.)

Saenz de Ugarte, B., Artiba, A., and Pellerin, R. (2009). Manufacturing execution system – a literature review. Production Planning and Control, 20(6):525–539. (Cited on pages 74 and 75.)

SaltStack, Inc. (2017). SALT.STATES.FILE. URL https://docs.saltstack.com/en/latest/ref/states/all/salt.states.file.html. Last accessed: October 6, 2017. (Cited on page 72.)

SAP SE (2013). Maintaining characteristics for configuration. URL https://help.sap.com/saphelp_erp60_sp/helpdata/en/a1/e7ba53422bb54ce10000000a174cb4/frameset.htm. Last accessed: June 15, 2017. (Cited on page 79.)

SAP SE (2014a). Building block configuration guide: Connectivity SAP ERP. SAP ERP 6.07, M11, October. (Cited on page 119.)

SAP SE (2014b). Building block configuration guide: Engineer-to-order (ETO)-project assembly. EHP7 for SAP ERP 6.0, 240, February. (Cited on page 119.)

SAP SE (2015a). SAP benchmark glossary. URL http://global.sap.com/campaigns/benchmark/bob_glossary.epx$#$s. Last accessed: May 2, 2016. (Cited on page 119.)

SAP SE (2015b). The value of SAP Solution Manager 7.1! URL https://de.scribd.com/document/293143515/SAP-Solution-Manager-7-1. Last accessed: June 19, 2017. (Cited on page 20.)

SAP SE (2016). SAP documentation: ABAP workbench tools. URL https://help.sap.com/saphelp_nw73ehp1/helpdata/en/ef/d94b78ebf811d295b100a0c94260a5/frameset.htm. Last accessed: October 9, 2017. (Cited on page 74.)

SAP SE (2017a). Installation of SAP systems with unattended mode. SAP Note 950619. (Cited on page 123.)

SAP SE (2017b). SAP documentation: Creating returns. URL https://help.sap.com/saphelp_erp60_sp/helpdata/en/8f/65b65334e6b54ce10000000a174cb4/frameset.htm. Last accessed: October 9, 2017. (Cited on page 80.)

SAP SE (2017c). SAP documentation: Quantity contract. URL https://help.sap.com/saphelp_erp60_sp/helpdata/en/4a/65b65334e6b54ce10000000a174cb4/frameset.htm. Last accessed: October 10, 2017. (Cited on page 80.)

Scheer, A.-W. (1997). Wirtschafsinformatik: Referenzmodelle für industrielle Geschäftsprozesse. Springer-Verlag, Berlin and Heidelberg. (Cited on pages 2, 24, 43, 76, and 78.)

Schlichter, B. and Kraemmergaard, P. (2010). A comprehensive literature review of the ERP research field over a decade. Journal of Enterprise Information Management, 23(4):486–520. (Cited on pages 13 and 24.)

Schneider, S. and Sunyaev, A. (2016). Determinant factors of cloud-sourcing decisions: reflecting on the IT outsourcing literature in the era of cloud computing. Journal of Information Technology, 31(1):1–31. (Cited on pages 16, 43, and 59.)

Schröder, K. and Pilgram, U. (2010). Industrialized IT - Idee und Realität. In Heinrich, H., Johannsen, A., and Bohne, D. (eds.), Tagungsband zum 9. Berlin-Brandenburger SAP-Forum der Fachhochschule Brandenburg: Business Software Trends in Ausbildung und Praxis, Brandenburg an der Havel, Brandenburg, pages 73–96. (Cited on page 53.)

Schryen, G. (2015). Writing qualitative IS literature reviews: Guidelines for synthesis, interpretation and guidance of research. Communications of the Association for Information Systems, 37(1):286–325. (Cited on page 10.)

ServiceNow, Inc. (2015a). Finance service management. URL https://www.servicenow.com/products/finance-service-management.html. Last accessed: November 24, 2015. (Cited on page 20.)

ServiceNow, Inc. (2015b). Product documentation: List of available integrations. URL http://bit.ly/2rNd2Tl. Last accessed: March 15, 2017. (Cited on page 20.)

Seuring, S. and Müller, M. (2008). From a literature review to a conceptual framework for sustainable supply chain management. Journal of Cleaner Production, 16(15):1699–1710. (Cited on pages 10 and 26.)

Sheikh, K. (2003). Manufacturing resource planning (MRP II): with introduction to ERP, SCM and CRM. McGraw-Hill, New York. (Cited on page 76.)

Siedersleben, J. (2007). SOA revisited: Komponentenorientierung bei Systemlandschaften. Wirtschaftsinformatik, 49(1):110–117. (Cited on page 34.)

Simon, A., Schoeman, P., and Sohal, A. S. (2010). Prioritised best practices in a ratified consulting services maturity model for ERP consulting. Journal of Enterprise Information Management, 23(1):100–124. (Cited on pages 50 and 74.)

Skene, J., Raimondi, F., and Emmerich, W. (2010). Service-level agreements for electronic services. IEEE Transactions on Software Engineering, 36(2):288–304. (Cited on pages 68, 69, and 70.)

Soltesz, S., Pötzl, H., Fiuczynski, M. E., Bavier, A., and Peterson, L. (2007). Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors. In Ferreira, P., Gross, T. R., and Veiga, L. (eds.), Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems, Lisbon, Portugal, pages 275–287. (Cited on page 36.)

Song, T., Wang, J., Wu, J., Ma, R., Liang, A., Gu, T., and Qi, Z. (2017). FastDesk: A remote desktop virtualization system for multi-tenant. Future Generation Computer Systems, 81(1):478–491. (Cited on page 111.)

Sonnenberg, C. and vom Brocke, J. (2012). Evaluations in the science of the artificial – reconsidering the build-evaluate pattern in design science research. In Peffers, K., Rothenberger, M., and Kuechler, B. (eds.), Proceedings of the 7th International Conference of Design Science Research in Information Systems: Advances in Theory and Practice, Las Vegas, NV, USA, pages 381–397. (Cited on pages xiii, 11, 103, 104, 106, 139, 151, and 154.)

Spinellis, D. (2005). Version control systems. IEEE Software, 22(5):108–109. (Cited on page 73.)

Spinellis, D. (2012). Don't install software by hand. IEEE Software, 29(4):86–87. (Cited on pages 18, 30, 58, 72, 88, and 135.)

Sprecher der WKWI und GI-FB WI (2008). WI-Orientierungslisten. Wirtschaftsinformatik, 50(2):155–163. (Cited on page 26.)

Spring, J. (2011a). Monitoring cloud computing by layer, part 1. IEEE Security & Privacy, 9(2):66–68. (Cited on page 95.)

Spring, J. (2011b). Monitoring cloud computing by layer, part 2. IEEE Security & Privacy, 9(3):52–55. (Cited on page 95.)

Staehr, L., Shanks, G., and Seddon, P. B. (2012). An explanatory framework for achieving business benefits from ERP systems. Journal of the Association for Information Systems, 13(6):424. (Cited on page 83.)

Stahlknecht, P. and Hasenkamp, U. (2005). Einführung in die Wirtschafsinformatik. Springer, Berlin et al. (Cited on pages 5, 22, 23, 34, and 69.)

Stallings, W. (2017). Overview of cloud computing. In Vacca, J. R. (ed.), Cloud Computing Security: Foundations and Challenges, pages 13–29. Taylor & Francis Group, Boca Raton, FL, USA. (Cited on pages ix and 21.)

Stevenson, M., Hendry, L. C., and Kingsman, B. G. (2005). A review of production planning and control: the applicability of key concepts to the make-to-order industry. International Journal of Production Research, 43(5):869–898. (Cited on page 4.)

Supply Chain Council (2010). SCOR: Supply Chain Operations Reference model. Version 10.0. (Cited on page 83.)

Susarla, A., Barua, A., and Whinston, A. B. (2003). Understanding the service component of application service provision: An empirical analysis of satisfaction with ASP services. MIS Quarterly, 27(1):91–123. (Cited on page 3.)

Talligent (2016). 2016 state of OpenStack report. URL http://talligent.com/wp-content/uploads/2016/03/2016-State-of-OpenStack-Report.pdf. Last accessed: March 7, 2017. (Cited on page 72.)

Talwar, V., Milojicic, D., Wu, Q., Pu, C., Yan, W., and Jung, G. (2005). Approaches for service deployment. IEEE Internet Computing, 9(2):70–80. (Cited on pages 14, 31, 40, 56, 57, 60, 63, 129, and 140.)

Tarhan, A. and Yilmaz, S. G. (2014). Systematic analyses and comparison of development performance and product quality of incremental process and agile process. Information and Software Technology, 56(5):477–494. (Cited on pages 30 and 58.)

Teubner, A. and Remfert, C. (2017). Giving IT services a theoretical backing. In Yamamoto, S. (ed.), Proceedings of the 19th International Conference Human Interface and the Management of Information: Information, Knowledge and Interaction Design, Vancouver, BC, Canada, pages 448–468. (Cited on pages 13, 15, 27, and 57.)

The HFT Guy (2016). Docker in production: A history of failure. URL https://thehftguy.com/2016/11/01/docker-in-production-an-history-of-failure/. Last accessed: October 16, 2017. (Cited on page 62.)

The Open Group (2017). The Open Group IT4IT$^{\mathrm{TM}}$ reference architecture. Version 2.1. (Cited on pages 2, 17, 19, and 45.)

Thomas, D. R. E. (1978). Strategy is different in service businesses. Harvard Business Review, 56(7):158–165. (Cited on page 2.)

Turner, J. A., Bikson, T. K., Lyytinen, K., Mathiassen, L., and Orlikowski, W. (1991).

Relevance versus rigor in information systems research: an issue of quality. In Nissen, H.-E., Klein, H. K., and Hirschheim, R. (eds.), Proceedings of the Working Conference on the Information Systems Research Arena, Copenhagen, Denmark, pages 1–33. (Cited on page 106.)

Turowski, K. (2014). Fachkomponenten. In Gronau, N., Becker, J., Kliewer, N., Leimeister, M., and Overhage, S. (eds.), Enzyklopädie der Wirtschaftsinformatik – Online Lexikon. GITO Verlag, http://www.enzyklopaedie-der-wirtschaftsinformatik.de. (Cited on page 34.)

Übernickel, F., Bravo-Sánchez, C., Zarnekow, R., and Brenner, W. (2006). IS service-engineering: A process model for the development of IS services. In Irani, Z., Sarikas, O. D., Llopis, J., Gonzalez, R., and Gasco, J. (eds.), Proceedings of the European and Mediterranean Conference on Information Systems, Costa Blanca, Spain, pages 1–8. (Cited on page 27.)

Valiente, M.-C., Garcia-Barriocanal, E., and Sicilia, M.-A. (2012). Applying an ontology approach to IT service management for business-IT integration. Knowledge-Based Systems, 28(1):76–87. (Cited on pages 17 and 44.)

Van den Berg, J. P. (2007). Integral warehouse management: The next generation in transparency, collaboration and warehouse management systems. Management Outlook Publications, Utrecht. (Cited on pages 71 and 76.)

Vanbrabant, B., Delaet, T., and Joosen, W. (2009). Federated access control and workflow enforcement in systems configuration. In Moskowitz, A. (ed.), Proceedings of the 23rd Large Installation System Administration Conference, Berkeley, CA, USA, pages 10–10. (Cited on page 40.)

Velamuri, V. K., Neyer, A.-K., and Möslein, K. M. (2011). Hybrid value creation: a systematic review of an evolving research area. Journal für Betriebswirtschaft, 61(1):3–35. (Cited on page 154.)

Vogedes, A. (2011). Ansatz eines Kapazitätsmanagements für die Erbringung von IT-Dienstleistungen. phdthesis, Universität St. Gallen. (Cited on pages 26, 28, 29, and 47.)

Walter, S. M., Böhmann, T., and Krcmar, H. (2007). Industrialisierung der IT - Grundlagen, Merkmale und Ausprägungen eines Trends. HMD – Praxis der Wirtschaftsinformatik, 44(4):6–16. (Cited on pages 14 and 15.)

Ward, J., Daniel, E., and Peppard, J. (2008). Building better business cases for IT investments. MIS Quarterly Executive, 7(1):1–15. (Cited on page 85.)

Webster, J. and Watson, R. T. (2002). Analyzing the past to prepare for the future: Writing a literature review. MIS Quarterly, 26(2):xiii–xxiii. (Cited on page 10.)

Weidner, S. (2012). Interactive learning - teaching IT project management using an explorative role play. In Abramowicz, W., Domingue, J., and Wecel, K. (eds.), Proceedings of the Business Information Systems: International Workshops and Future Internet Symposium, Vilnius, Lithuania, pages 198–207. (Cited on page 118.)

Weske, M. (2012). Business process management. Springer, Heidelberg et al. (Cited on pages ix, 22, 65, 66, 67, and 68.)

Wettinger, J., Andrikopoulos, V., Leymann, F., and Strauch, S. (2016). Middleware-oriented deployment automation for cloud applications. IEEE Transactions on Cloud Computing, online. (Cited on pages 4, 5, 37, 42, 45, 57, 59, 60, 62, 63, and 72.)

Wettinger, J., Behrendt, M., Binz, T., Breitenbücher, U., Breiter, G., Leymann, F., Moser, S., Schwertle, I., and Spatzier, T. (2013). Integrating configuration management with model-driven cloud management based on tosca. In Desprez, F., Ferguson, D., Hadar, E., and Leymann, F. (eds.), Proceedings of the 3rd International Conference on Cloud Computing and Service Science, Aachen, Germany, pages 437–446. (Cited on pages 40, 45, and 103.)

Wettinger, J., Breitenbcher, U., and Leymann, F. (2014). Compensation-based vs. convergent deployment automation for services operated in the cloud. In Franch, X., Ghose, A., Lewis, G., and Bhiri, S. (eds.), Proceedings of the 12th International Conference on Service-Oriented Computing, Paris, France, pages 336–350. (Cited on pages 41 and 63.)

Wikipedia (2003). Citrix systems. URL https://en.wikipedia.org/wiki/Citrix_Systems. Last accessed: November 30, 2017. (Cited on page 111.)

Wilde, T. and Hess, T. (2007). Forschungsmethoden der Wirtschaftsinformatik. Wirtschaftsinformatik, 49(4):280–287. (Cited on page 106.)

Willcocks, L. and Fitzgerald, G. (1993). Market as opportunity? Case studies in outsourcing information technology and services. The Journal of Strategic Information Systems, 2(3):223–242. (Cited on page 9.)

Willis, J. (2010). What DevOps means to me. URL https://blog.chef.io/2010/07/16/what-devops-means-to-me/. Last accessed: February 10, 2018. (Cited on pages 30, 31, 32, and 44.)

Winter, R. and Fischer, R. (2007). Essential layers, artifacts, and dependencies of enterprise architecture. Journal of Enterprise Architecture, 3(2):1–12. (Cited on pages 33 and 66.)

Wittgreffe, J., Trollope, C., and Midwinter, T. (2006). The next generation of systems to support corporate grade ICT products and solutions. BT Technology Journal, 24(4):93–112. (Cited on pages 26, 44, and 50.)

WKWI (1994). Profil der Wirtschaftsinformatik, Ausführungen der Wissenschaftlichen Komission der Wirtschaftsinformatik. Wirtschaftsinformatik, 36(1):80–81. (Cited on page 1.)

Wu, M.-S., Huang, S.-J., and Chen, L.-W. (2011). The preparedness of critical success factors of IT service management and its effect on performance. The Service Industries Journal, 31(8):1219–1235. (Cited on pages 2 and 17.)

Wuhib, F. Z., Yanggratoke, R., and Stadler, R. (2013). Allocating compute and network resources under management objectives in large-scale clouds. Journal of Network and Systems Management, 23(1):111–136. (Cited on page 28.)

Yang, H. and Tate, M. (2012). A descriptive literature review and classification of cloud computing research. Communications of the Association for Information Systems, 31(1):35–60. (Cited on pages 21 and 35.)

Yazici, A., Mishra, A., and Kontogiorgis, P. (2015). IT service management (ITSM) education and research: Global view. International Journal of Engineering Education, 31(4):1071–1080. (Cited on page 2.)

Zarnekow, R. (2007). Produktionsmanagement von IT-Dienstleistungen: Grundlagen, Aufgaben und Prozesse. Springer, Berlin and Heidelberg. (Cited on pages 2, 15, 27, 28, 29, 45, 47, 56, 68, and 83.)

Zarnekow, R. and Brenner, W. (2003). A product-based information management approach. In Ciborra, Claudio U.and Mercurio, R., de Marco, M., Martinez, M., and Carignani, A. (eds.), Proceedings of the 11th European Conference on Information Systems, Naples, Italy, pages 2251–2263. (Cited on pages 15, 37, and 68.)

Zarnekow, R., Brenner, W., and Pilgram, U. (2006). Integrated information management: Applying successful industrial concepts in IT. Springer, Berlin and Heidelberg. (Cited on pages 2 and 15.)

Zarnekow, R., Scheeg, J., and Brenner, W. (2004). Untersuchung der Lebenszykluskosten von IT-anwendungen. Wirtschaftsinformatik, 46. (Cited on page 129.)

Zhu, L., Xu, D., Tran, A., Xu, X., Bass, L., Weber, I., and Dwarakanathan, S. (2015). Achieving reliable high frequency releases in cloud environments. IEEE Software, 32(2):73–80. (Cited on pages 41, 61, and 93.)

Zolnowski, A., Schmitt, A. K., and Böhmann, T. (2011). Understanding the impact of remote service technology on service business models in manufacturing: from improving after-sales services to building service ecosystems. In Tuunainen, V. K., Rossi, M., and Nandhakumar, J. (eds.), Proceedings of the 19th European Conference on Information Systems, Helsinki, Finland, pages 1–13. (Cited on page 79.)

Ehrenerklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; verwendete fremde und eigene Quellen sind als solche kenntlich gemacht. Insbesondere habe ich nicht die Hilfe eines kommerziellen Promotionsberaters in Anspruch genommen. Dritte haben von mir weder unmittelbar noch mittelbar geldwerte Leistungen für Arbeiten erhalten, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen.

Ich habe insbesondere nicht wissentlich:

- Ergebnisse erfunden oder widersprüchliche Ergebnisse verschwiegen,
- statistische Verfahren absichtlich missbraucht, um Daten in ungerechtfertigter Weise zu interpretieren,
- fremde Ergebnisse oder Veröffentlichungen plagiiert,
- fremde Forschungsergebnisse verzerrt wiedergegeben.

Mir ist bekannt, dass Verstöße gegen das Urheberrecht Unterlassungs- und Schadensersatzansprüche des Urhebers sowie eine strafrechtliche Ahndung durch die Strafverfolgungsbehörden begründen kann. Die Arbeit wurde bisher weder im Inland noch im Ausland in gleicher oder ähnlicher Form als Dissertation eingereicht und ist als Ganzes auch noch nicht veröffentlicht.

Magdeburg, den 2. Juli 2018

. . . .
Johannes Hintsch, M. Sc.