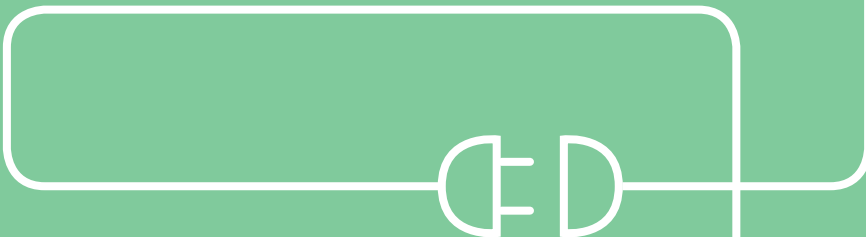


Fabian Aulkemeier

Pluggable Services

A Platform Architecture
for E-Commerce



PLUGGABLE SERVICES

A PLATFORM ARCHITECTURE FOR E-COMMERCE

Fabian Aulkemeier

Graduation committee

Chairman:

Prof. dr. Th.A.J. Toonen (University of Twente, The Netherlands)

Supervisors:

Prof.dr. J. van Hillegersberg (University of Twente, The Netherlands)

Prof.dr. M.-E. Iacob (University of Twente, The Netherlands)

Members:

Prof.dr. W.J.A.M. van den Heuvel (Tilburg University, The Netherlands)

Prof.dr.ir. S.L.J.M. de Leeuw (Nottingham Trent University, England, UK)

Prof.dr. C. Legner (University of Lausanne, Switzerland)

Prof.dr.ir. L.J.M. Nieuwenhuis (University of Twente, The Netherlands)

Prof.dr. R.J. Wieringa (University of Twente, The Netherlands)



The research was conducted as part of the CATeLOG project by the Dutch Institute for Advanced Logistics.

CTIT

CTIT Ph.D. Thesis Series No. 16-417

Centre for Telematics and Information Technology
P.O. Box 217, 7500 AE Enschede, The Netherlands.

ISSN: 1381-3617

ISBN: 978-90-365-4283-8

DOI: 10.3990/1.9789036542838

<https://dx.doi.org/10.3990/1.9789036542838>

Cover: Jana Kühl

Print: Ipskamp Printing

Copyright © 2017 Fabian Aulkemeier

No part of this thesis may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopy, recording or otherwise without permission of the author.

PLUGGABLE SERVICES

A PLATFORM ARCHITECTURE FOR E-COMMERCE

DISSERTATION

to obtain
the degree of doctor at the University of Twente,
on the authority of the rector magnificus
Prof.dr. T.T.M. Palstra,
on account of the decision of the graduation committee,
to be publicly defended
on Wednesday, April 12, 2017 at 14.45

by
Fabian Michael Aulkemeier
born on December 21, 1982
in Werneck, Germany

This dissertation has been approved by:

Prof.dr. J. van Hillegersberg

Prof.dr. M.-E. Iacob

Table of Contents

1	Introduction	1
1.1	Critical themes and research gap	5
1.2	Design research and research design	10
1.3	Dissertation outline	15
2	The State of the Art in E-Commerce Architectures	21
2.1	Research design	23
2.2	Objectives	23
2.3	Systematic literature review	27
2.4	Reference architecture	34
2.5	Validation	40
2.6	Conclusions	44
3	Measuring the Pluggability of Software	47
3.1	Quality models	48
3.2	Pluggability of services	52
3.3	Conclusions	55
4	A Platform-Based Return Registration Process	57
4.1	Platform architectures	58
4.2	A reference architecture for e-commerce service platforms	61
4.3	The return registration case	66
4.4	Validation	70
4.5	Conclusions	75
5	A Platform-Based Pluggable Trade Compliance Service	77
5.1	Preliminary considerations	78
5.2	A pluggable service platform	81
5.3	The trade compliance case	87
5.4	Conclusions	97

6	Analytics as a Service: A Pluggable Sales Forecasting Service	99
6.1	New sales forecasting module	100
6.2	Pluggable architecture	103
6.3	A pluggable sales forecasting service	106
6.4	Conclusions	109
7	Using Pluggable Services to Support IT-Driven Collaboration in Business Networks	111
7.1	Collaboration architectures	113
7.2	Service-oriented collaboration	116
7.3	Cross-selling architecture	118
7.4	Product evaluation	121
7.5	Conclusion	123
8	Conclusions	125
8.1	Anatomy of the proposed design	125
8.2	Limitations and future research	129
A	ArchiMate Metamodel	133
B	E-Commerce Business Model	134
C	Pluggability Instrument	135
D	Platform Prototype	136
D.1	Model and interface implementation	136
D.2	Administration interface	139
D.3	Platform client	142
E	Trade Compliance Service	144
	Bibliography	147
	Dutch Abstract	162

Chapter 1

Introduction

The recent growth of electronic commerce (e-commerce) has led to opportunities and challenges for existing and new stakeholders in the retail sector. Traditional retail distribution supplies brick and mortar stores with a limited and rigorously planned assortment of goods. E-commerce on the other hand, currently operates out of central warehouse locations, from where a larger amount and variety of goods can be offered to a more global customer base (Gunasekaran et al., 2002). As a consequence, customers profit from the greater choice of products, due to the decrease of geographic boundaries. Thus, stores and products with a niche existence are able to gain a stronger presence eventually leading to a *higher degree of competition in the retail market* (Brynjolfsson et al., 2006).

In order to stay competitive in the e-commerce market, a number of success factors should be taken into consideration (Feindt et al., 2002). While in many product domains pricing is a crucial factor (Rao et al., 2011; Wallace et al., 2004), shopping experience (content and convenience) and supply chain performance (control) play equally important roles, particularly for e-commerce endeavors in a start-up phase. A continuous growth of an online retail business depends on a strong partner network and ongoing process improvement. Furthermore, Feindt et al. (2002) stress the importance of integration with the information technology (IT) systems. In fact, most success factors depend to some extent on an elaborated and innovative IT infrastructure which can produce original frontend services and efficient handling of the order-fulfillment processes on the backend side.

Through the introduction of omnichannel retail, traditional retailers can make use of their existing stores, and gain a competitive advantage over so-called pure player e-commerce firms. The pure player business model has been a disruptor of the retail sector during the last years. However, Hudetz et al. (2016) argue that pure online shops are going to be eliminated entirely by omnichannel retailers within the next four years. Thus, destroying 90% of existing e-commerce businesses. The omnichannel model can be considered as a merger of the online and offline sales channels. A benefit of this integration is its capability to achieve better customer service, for instance by allowing buyers to order online and pick up goods in a physical store. Another potential advantage is the increased supply chain performance, for example by storing and shipping goods directly from a local store. Layo and Silver (2013) report that shipping out-of-inventory items directly from store increases revenue by 10-20%. Companies who manage to successfully implement a seamless

integration of online and offline channels are also thought profit from a higher customer loyalty in the highly competitive retail market (Wallace et al., 2004). To date, offline and online channels are often realized as distinct value chains with different underlying processes and static integration between systems. The merger of the two value chains in the prevailing static fashion will cause problems as soon as further sales channels such as mobile devices or kiosk systems will be introduced. A suitable e-commerce architecture should, therefore, be able to *flexibly integrate internal and external IT systems* and allow for plugging in new services, such as cloud applications, without affecting the overall architecture.

Next to the novel sales channels, competition in retail is growing through sales across national borders. While previously prevalent only in business-to-business (B2B) transactions, an increasing number of consumers buy from foreign vendors. The European Union (EU) advocates cross-border e-commerce and formulated the target of 20% of all EU citizen buying goods abroad by 2020 (European Commission, 2015). Global marketplaces such as Alibaba are currently among the most successful and fastest growing e-commerce firms (Smith, 2014). In order to limit the risk of cross-border e-commerce endeavors, the collaboration with an *increased number and new types of supply chain participants* is often desirable. Third party warehouses and fulfillment centers, the use of global marketplaces, and collaboration with specialized service providers help to reduce the investments required to tap new markets. Services for cross-border compliance, for example, help to cope with the requirements for global trade (Pincvision, 2012). Essentially, one business to consumer e-commerce transaction regularly involves a number of downstream business to business (B2B) e-commerce transactions. The market for corresponding B2B e-commerce services will grow significantly in the upcoming years (Soshkin, 2015). However, the current e-commerce architectures are based on static integration patterns and do not provide the right level of agility to meet the required level of flexible interoperation with external services (Lankhorst et al., 2012). Furthermore, the need for improved inter-organizational operation goes beyond the IT-related challenges. In fact, organizations need to find suitable business partners, enter a legal relationship, and maintain the complex agreements between partners. The concept of quick connect capability (QCC) has been established to describe the ability of organizations to engage in new partnerships in an efficient manner (van Heck and Vervest, 2007). The lack of QCC can be considered as a major obstacle in the service based business models (Koppius and van de Laak, 2009).

Another competitive factor in e-commerce is created by the *amount of information that becomes available when selling online*. Traditional retailers invest remarkable resources to collect consumer data, for example by introducing membership programs. With the online channel this information becomes available implicitly. The entirety of transactions is available in the system, regardless of whether a product gets ordered, sold, returned, or only clicked at in the online store. Retailers must exploit this information and use it to their advantage if they want to stay competitive. The challenge for retailers is the valorization of the available information, as its amount and diversity grow faster than the means of interpretation. The shift from

business intelligence based on pre-defined structured transaction data to the analysis of vast unstructured and scattered data (big data) is a challenge, even for large organizations with extensive budget (Tallon, 2013). Furthermore, instead of facilitating the extraction of critical information, novel e-commerce services introduce an increasing amount of data repositories. Thus, leading to a growing heterogeneity of information and the inability to delegate the complex task of data analytics to external services (Hashem et al., 2015).

We can conclude that the shift from offline to online retail as well as the recent trends of cross-border and omnichannel e-commerce require much more integrated IT support than current software packages offer. We can observe that to date, many successful online retailers consider themselves as IT companies. Thus, it is not surprising that Amazon, the most successful online retail business worldwide, also happened to become the largest IT infrastructure service provider (Leong et al., 2015). In fact, 50% percent of traditional retailers still hesitate to make the required investments into omnichannel activities (Guy, 2016). Especially smaller retailers are forced to decide, whether or not to heavily invest in their IT landscapes and technical expertise, in order to stay competitive (Hinton, 2014). Easily adoptable e-commerce solutions that allow for incremental growth would be desirable. Cloud solutions can help to outsource large amounts of IT development and efforts to operate the IT landscape. Thanks to their pay-by-use model they can reduce the investment risk (Armbrust et al., 2010). Despite these benefits, some issues originating from the stated innovation in e-commerce remain unsolved: 1) the need for more flexible integration of the services, 2) the requirement to facilitate the interoperation with business partners to outsource required services, such as warehousing and delivery, and 3) the inability to accurately derive the right information from the inhomogeneous data.

The use cases presented throughout this work are oriented towards the stated problem areas and will serve as a means to demonstrate the capabilities of a *novel architectural design*. In this work we define *IT architecture* as “the components of an IT infrastructure, the relation between the components and their environment, and the principles guiding their design” (Jen and Lee, 2000). All IT architecture serves a certain business purpose. The design and development of an IT architecture with the focus on the business strategy as well as the required business information and processes is considered as *enterprise architecture*. The enterprise architecture of online retailers is the main matter of this work. More precisely, research goals (RG) can be described as follows:

RG1 *Understand and describe the state of the art in architectures for online retail:*

We want to learn what are the current e-commerce and e-business components that are in place in online retail. We will investigate state-of-the-art reference models for online and offline retail. Based on the existing models and their shortcomings, we come up with a novel architectural reference model for online retail, including all the required e-commerce and e-business components.

RG2 *Identify e-commerce issues and opportunities that can be solved through innovative e-commerce services and provide working solutions:*

To gain competitive advantage, online retailers want to adopt innovative services that attract more customers and make the order fulfillment processes more efficient. The business cases considered throughout this work are based on the requirements of the consortium members collaborating in the CATeLOG project. They reflect the insights into key issues of the practitioners and propose working solutions for specific business requirements, that are not covered within state-of-the-art e-commerce offerings. Such services can be divided into three categories 1) services that provide a specific, novel application functionality to the retailer (IT services) 2) services that facilitate the collaboration with partners (collaboration services) 3) services with added value, such as logistics handling, which are integrate IT wise through the architecture (business services).

RG3 *Facilitate through design and implementation of a service-oriented IT architecture a) the use of services, to maximize their ease of adoption and b) the collaboration among e-commerce partners, through the use of collaborative services:*

The key contribution of this work lies in the IT architecture which aims at transforming state-of-the-art e-commerce platforms and services into a pluggable IT landscape. The goal is to allow retailers to adopt innovative services with fewer resources in terms of time and IT expertise. To ensure the achievement of this goal we establish the notion of *pluggability* as a quality characteristic for IT services and a corresponding instrument to evaluate our design artifacts.

With these points in mind, we have been investigating the state of the art in e-commerce but also studied the generic IT architectures and trends, such as cloud services. Therefore, many of the resulting design artifacts, prototypical implementations, and evaluations presented in this work may be valid for other business domains. In fact, we argue that many of the architectural principles presented in this work are of a generic nature and can be transferred easily. However, we stick to the presentation and evaluation in the context of online retail and invite the readers to come up with their own projections onto their respective field of interest.

The remainder of the introductory chapter is structured as follows. In Section 1.1 we elaborate on the core concepts of this work and point out the current research gaps regarding the aforementioned architectural goals. Section 1.2 describes the research design and establishes the methodological framework that was applied. Section 1.3 provides a preliminary summary and links this chapter to the remainder of the book.

1.1 Critical themes and research gap

Service-oriented architectures (SOA) have been conceived as an effective means to achieve dynamic processes through the composition of loosely coupled services (Merrifield et al., 2008). The loose coupling of services is a means to reduce dependencies among IT components. Thus, allowing to transform monoliths into microservice architectures (Lewis and Fowler, 2014). It allows the IT operators to maintain and evolve IT components independently and to extend the system's functionality by gradually adding new services with a bounded context. Software-as-a-service (SaaS) offerings are a particularly interesting option for such services as they allow faster adoption and better scalability (Armbrust et al., 2010; Waters, 2005). Eventually, the service-oriented approach bears the potential to give retailers the *freedom of choosing the right services independently of their complexity and the technical capabilities of the organization*.

However, a questionable aspect of SOA is the standardized, self-described service interface, which is supposed to facilitate service reuse and integration. As it turns out, the propagated ease of integration through automated service discovery and binding at runtime has been overestimated (Bachlechner et al., 2006). As we demonstrate in Chapter 4, the connection of the endpoints of various cloud services remains a very complex task. It does not approximate the level of simplicity the ready-to-use cloud service model was made up for. Consequently, replacing monolithic on-premises e-commerce systems with cloud services bears major drawbacks with regard to interoperability.

As it turns out, SOA middleware is mostly applied within organizations. So-called service repositories help large organizations to catalog service endpoints across internal domains. For inter-organizational endpoints, a new breed of middleware, so called API management products, are gaining in popularity (Clark, 2015). In fact, such software helps organizations to setup and control public gateways to their IT systems through developer portals and other components. The consumer of such programming interfaces requires the tools and skills to make use of the provided endpoints. Thus, the goal of *reducing the technical complexity of adopting novel IT and business services* remains a critical issue, even with the latest developments in the market.

1.1.1 Service platforms

The architectural pattern to potentially ease the adoption of services is the *service platform* (van Heck and Vervest, 2007). Two-sided platforms support service providers and service users at the same time. However, in many cases, such platforms are marginally perceived by the users. They act as 'invisible engines' enabling their clients to provide services (Evans et al., 2008). The model in Figure 1.1 illustrates the actors and relations in a two-sided platform ecosystem. The model makes use of the ArchiMate modeling notation for enterprise architecture (The Open Group, 2016). We make use of the notation throughout this work as it al-

lows to illustrate business and IT layer concepts, as well as their relationships (cf. Appendix A).

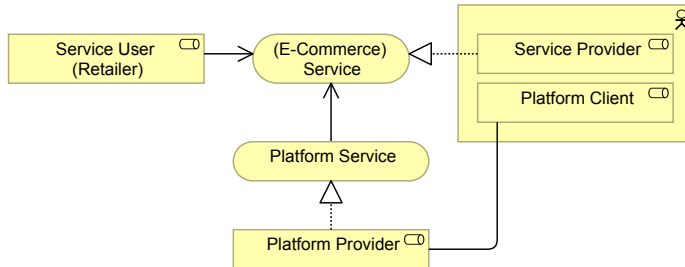


Figure 1.1: Roles and actors in a two-sided service platform setting

A service provider is a client (or partner) of the platform provider (or platform sponsor) and offers services to the users. In this work, we investigate the e-commerce services offered to retailers. Hence, we consider the e-commerce service provider as the client and the retailer as the user of the e-commerce platform. The e-commerce service makes use of the platform services which are implemented by the platform provider. The basic idea of a platform is the separation of stable and evolving components (Baldwin and Woodard, 2009). The platform provider is responsible for implementing long-term stable components of the system which will be used across services. That way, redundancy in functionality and data can be limited. Eventually, the service implementation and evolution, adoption, as well as cross-service collaboration gets simplified. Platforms following the described principles have been widely adopted in many domains such as mobile computing, geographic information systems, or gaming (Wareham et al., 2014).

1.1.2 Ecosystems of enterprise IT platforms

In the previous section, we highlighted the potential benefits of service platforms and illustrated them based on an example in mobile computing. In enterprise information systems, various examples for platforms exist. In the following, we outline some of the prevailing platforms and discuss their characteristics to relate them to the goals of the aspired platform architecture.

- *Marketplaces for enterprise SaaS offerings* such as the Oracle Cloud Marketplace¹ provide a means to help organizations during the first phase of service adoption to find and evaluate cloud services similar to ‘App Stores’ for mobile devices in the consumer space.

¹ Available at <https://cloud.oracle.com/marketplace>

Table 1.1: Comparison of selected enterprise IT platforms

Platform	SaaS marketplace	API/Web service directory	PaaS
Platform category	transaction platform	exchange platform	software platform
Client	SaaS provider	reusable service provider	cloud application implementers
Client benefits	helps to advertise and sell the services	helps to advertise the services	reduction of administrative tasks, improved scalability, reduced risk
User	business analyst	application developer	application user
User benefits	helps to analyze various SaaS offerings	find appropriate services at development time	improved user experience at runtime
Visibility	both client and user actively use the platform	both client and user actively use the platform	platform not visible to user

- *API or web service directories* such as ProgrammableWeb² provide a searchable index of publicly available reusable services. Instead of providing standalone solutions such services offer singular pieces of functionality such as geo-coding, currency converters, or routing. They are usually consumed within another application.
- *Platform-as-a-service (PaaS)* offerings facilitate development and deployment of enterprise application (Pivotal, 2015). Popular examples include the Amazon's Elastic Beanstalk, Heroku, IBM Bluemix, CloudFoundry, or Red Hat's OpenShift.

The first type of platform can be categorized as *transaction platform* according to (Sriram et al., 2014). Their purpose is to handling transactions between a service provider and the user. Accordingly, the second platform is an *exchange platform* helping service providers and users to find each other. The third example is a *software platform*. It acts as a foundation for service implementation. To further compare the three platforms Table 1.1 summarizes the actors and benefits of the different platforms.

We can observe that the various platforms support the client and user at discrete phases of the service implementation and adoption endeavors. Furthermore, we

² Available at <https://www.programmableweb.com>

can see that the platforms are geared towards discrete actors. Exchange and transaction platforms support the user and client to engage into a relationship and to conclude a contract. They do not support the setup and operation of a service. The software platform, on the other hand, focuses on deployment and operation of the service components but is somewhat invisible to the user and does not support him directly. For example, integrating the service into the IT landscape is out of scope of common PaaS solutions. Very specialized PaaS offerings for integration (iPaaS) start to emerge that facilitate service interoperation exclusively (Pezzini et al., 2014). In Chapter 4 we take a closer look at this type of platform and into its capabilities and limitations.

1.1.3 The missing platform

Pluggability is a software quality characteristic that we introduce in this work. It puts the needs of the service user into focus and is reflecting the recent trend of SaaS. Other software quality models such as agility, reusability, or interoperability are related but do not reflect the concept of services that abstracts the internal structure and behavior of a software component. Early quality models already include characteristics such as reusability or flexibility (Dromey, 1996; McCall et al., 1977). These characteristics reflect the ability of software implementers to reuse and change parts of the software but not the ability to reuse the same software product for a variety of use cases. Interoperability or configurability are quality characteristics that are more oriented towards the software user. Interoperability has become popular with the rise of web service technologies (Chung et al., 2003). It reflects the level of semantical standardization across systems and the possibility to engineer data exchanges between systems (Tolk and Muguira, 2003). Pluggability requires, but is not limited to, a high degree of interoperability. Service related concepts such as agility (Lankhorst et al., 2012) focus on the overall architecture and not individual services. Using pluggable service can eventually lead to a higher agility of the architecture and the enterprise. Chapter 3 is dedicated to presenting the pluggability model in detail.

A pluggable service platform architecture must be geared towards the needs of the service user and support him throughout the entire cycle of service use. Instead of focusing on the entire lifecycle of software use, the existing transactional and exchange platforms only focus on the early phase of service consumption. The existing software platforms are a step forward towards supporting the entire lifecycle of service use. However, their functionality is geared towards service allocation rather than service consumption (cf. Chapter 4). As a consequence, it supports the service provider (client) rather than the service user. In the following, we summarize the properties of the type of platform that we are missing in the current debate. Its goal is to facilitate the implementation of easy to adopt e-commerce services (RG3).

Platform category The intended solution can be considered as a software platform. In general, it should also bring together e-commerce service providers and retailers (exchange platform) and handle the contracts between them (transaction). However, as outlined previously, many of the existing platforms, such as service directories and marketplaces, offer support for service exchange and transaction. Thus, in this work, we focus on the characteristics of the platform with regard to service implementation, execution, and use.

Client The client of the intended platform is the e-commerce service provider. In contrast to existing platforms, the platform is geared towards domain-specific services. The limitation to a specific domain allows the platform provider to offer domain specific functionality. The existing platforms are not able to provide this kind of functionality due to their generic nature. In this work, we put a strong focus on the benefit of e-commerce specifics of the platform.

Client benefits The intended benefit for the client is the ability to deliver high-quality services in terms of pluggability. The platform should allow the service provider to focus on the implementation of novel e-commerce functionality. At the same time, he should be able to provide services that fit seamlessly into the remaining service landscape. Furthermore, it should allow the service provider to deliver services that enhance the collaboration within the business network.

User The retail company is the user of the intended platform. The user should be able to consume various domain specific services from different service providers. He should be able to adopt new services and exchange existing services with offers from other service providers.

User benefits By consuming multiple services through the platform the retailer's IT landscape is built on a core artifact which contains all the long term stable components. By relying on the platform individual e-commerce functionality can be adopted and exchanged in a pluggable fashion. The time to adopt novel functionality can be shortened, while the risks and efforts for the adoption of novel IT services, business services, and collaboration services decrease.

Visibility Existing software platforms are mostly visible to the client and act as invisible engines for the user. The intended platform should follow the same principle that most of its benefits are provided behind the scenes for the user. The user only gets in touch with the platform when it comes to adopting new services and handles contracts with service providers (exchange and transaction capabilities). For the service provider, on the other hand, the platform becomes a pivotal point for service implementation and delivery.

1.2 Design research and research design

Previously, we have presented current challenges in online retail and put forward the need for services that reflect the pluggability criteria of software quality. Furthermore, we discussed some of the issues of current architectural approaches and motivated the design and implementation of a platform architecture. The outcome of this work should be a tangible artifact with the intention to present and test a working solution. Eventually, our research provides novel architectural approaches and software components that can be reused in practice. Our pursuit of such intentions was based on the design science research paradigm which aims at designing, implementing, and testing artifacts in a context (Wieringa, 2014). Design science methods provide a more rigorous approach to building IT solutions compared to usual industry engineering processes. They require grounded reasoning of the design decisions and testing of the artifact's effectiveness based on concrete criteria. Various design science research methods (DSRM) have evolved over the years. In the following, we reflect some perspectives on design science and come up with a methodological framework that fits our research. Subsequently, we present a mapping of our contributions to the established framework.

Walls et al. (1992) argues that the goal of design science is the formulation of a design theory to make a design (prescription through synthesis) scientific (typically description or prediction through analysis). A design theory consists of 1) a design goal, 2) existing *theories* that drive the design, 3) the design *artifact*, as well as 4) a testable *product* that can be used to test the effectiveness of the artifact with regard to the goal. In the following, we map our research to these four elements.

In the previous section we introduced the *theoretical concepts* of enterprise architecture, service-orientation, platform architectures, and software quality that guided our design. Further theoretical underpinnings include reference modeling and model-driven development of prototypes. The *goal* of our design was outlined by formulating the research goals. In the subsequent section, we will formulate a number of design research question which specify the research goal. Finally, a differentiation is made between the *design artifact* which is represented through the architectural model and the *design product* which consists of multiple platform and e-commerce service prototypes that aim at validating the artifact.

1.2.1 Research questions

Previously we outlined the goals of our research. In this section, we specify the goals through the formulation of design research questions. Research questions are often considered as a crucial instrument. In fact, Gregor (2006) links the type of an information system (IS) theory to the attributes of the research questions. Other researchers stress the relevance of the problem statement in design science and the importance of the research goal over research questions. Wieringa (2009) proposes the nesting of the practical problem and the research question. According to Wieringa (2014) the 'practical problem solving delivers artifacts' and 'design sci-

ence research investigates properties of the artifacts' and thus provides answers to research questions. According to Walls et al. (1992) "explanatory theories tell 'what is', predictive theories tell 'what will be', and normative theories tell 'what should be', design theories tell 'how to/because.'" According to this categorization and the previously elaborated goals (G1-G3) of our research, we formulate the main design research question (MQ) as follows:

MQ: How to improve the state-of-the-art e-commerce architectures in order to facilitate pluggable services that help members of the e-commerce ecosystem to respond to current trends in their domain and to improve inter-organizational collaboration within their business network?

To answer the main question multiple sub-questions are formulated as follows:

SQ1: What is the state of the art in e-commerce architectures? (Analysis)

SQ2: How can the pluggability of services be operationalized? (Analysis and Design)

SQ3: What are the capabilities of existing e-commerce architectures in supporting pluggable services? (Prediction)

SQ4: How to facilitate service pluggability through a platform architecture? (Design)

SQ5: How to improve inter-organizational collaboration through pluggable services? (Design)

In Figure 1.2 the research questions are mapped onto the research goals that have been established previously.

1.2.2 Methodological framework

The methodological framework of this work relies on Peffers et al. (2007) and Hevner et al. (2004). Peffers et al. (2007) provide a nominal process model for conducting design science research in IS. The model guided us through the different phases of the design artifact and product construction cycle. Hevner et al. (2004) on the other hand provides a conceptual framework that helps us to make the contribution of our work more explicit and name the input and output during each design cycle in order to ensure a balance between theory and practice. In what follows, we start with describing the cycles of rigor and relevance and conclude with the description of the design cycle.

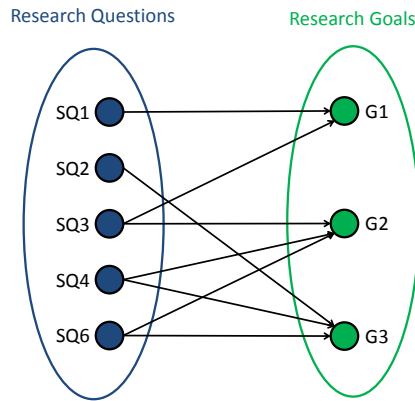


Figure 1.2: Mapping of research questions and research goals

Rigor and relevance

According to Hevner et al. (2004), design science “creates and evaluates IT artifacts intended to solve identified organizational problem”. The problem has to be *unsolved and important* on the one hand (relevance) and the artifact’s utility, quality, and efficiency have to be *rigorously evaluated* and its development process drawn from existing knowledge on the other hand (rigor). The framework in Figure 1.3 illustrates the incorporation of the two principles into our research.

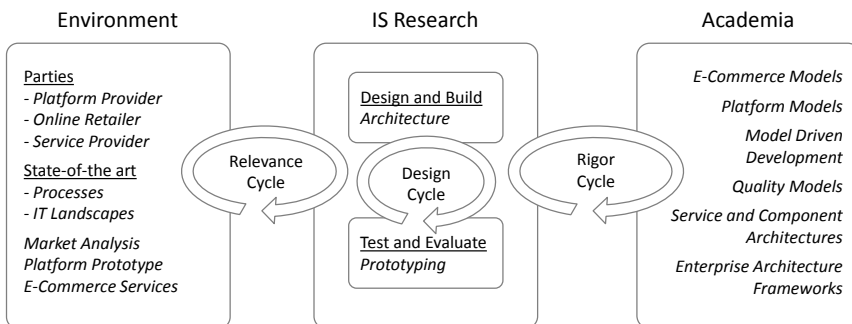


Figure 1.3: Instantiation of the IS research framework (Hevner et al., 2004)

The relevance of the design artifact is assessed by its contributions to the *environment*. The artifacts proposed in this work are based on the requirements of retailers, e-commerce service providers, and the platform provider. The *relevance cycle* indicates that the design evolves around existing processes, IT landscapes, and technical artifacts (IT systems, frameworks, platforms) of the organizations involved in the project or available on the market. Concrete contributions made to

practice include a market analysis on integration platforms and a resulting reference architecture in Chapter 4, a platform architecture and prototype in Chapter 5, as well as various e-commerce services presented in Chapter 4, Chapter 5, Chapter 6, and Chapter 7.

The second pillar in the IS research framework is the contribution to *academia*. The *rigor cycle* ensures that existing theory is reflected and new theoretical contributions are made explicit. In fact, the starting point of our architecture are the available state-of-the-art e-commerce process reference models and a novel architectural reference model, representing the result of a systematic literature review (Chapter 2). In the same fashion, we reflect current quality models and propose a novel quality model and instrument for pluggability (Chapter 3). The architectural models rely on enterprise architecture modeling frameworks and the actual architecture originates from the service-oriented thinking. Finally, the development of the design product is following a model-driven approach.

The idea behind the *design cycle* is to gradually improve the solution, by creating the new design artifact, based on the evaluation of the previous design product. Thus, our design starts with a state-of-the-art architecture (artifact) and the evaluation of the state-of-the-art prototype (product) in Chapter 4. The second design cycle reflects the shortcomings of the state-of-the-art architecture and prototype (Chapter 5). In the third design cycle (Chapter 7) we extend the objectives of the architecture and evaluate the product with regard to the collaborative capabilities of the service. In the following section, we describe the design cycle in more detail.

Design cycle

According to Peffers et al. (2007) and Wieringa (2009) the design cycle starts with one or two initial phases where the motivation and objectives for the design are defined. Subsequently, the design and development phase produces a design artifact that is used for the instantiation of the design product during the demonstration phase. The design product is then used for evaluation. The subsequent design cycle can either start with the reformulation of the design objectives or directly with the new design and development phase (Figure 1.4).

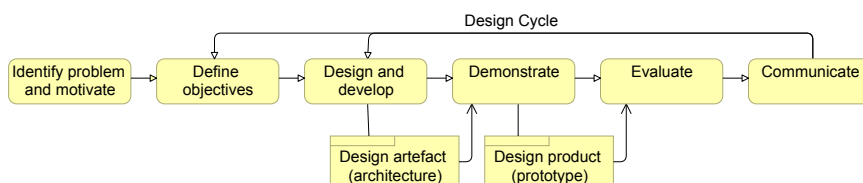


Figure 1.4: Six phases of the design science research cycle (Peffers et al., 2007)

The *motivation* for a design should be specific and the solution should provide a justified value (Peppers et al., 2007). The architectural design of the platform is motivated by the research goals and research questions that have been elaborated previously. Furthermore, in Chapter 2, we review the state of the art in e-commerce architectures and their capabilities which provides a further understanding of the limitations current architectures entail. By embedding the architecture in the e-commerce domain and the study of concrete e-commerce services we provide tangible use cases to underline the practical value of the design.

The *objectives* of the design are derived from the problems identified during the motivation phase and provide a means to operationalize the evaluation of the design. Verschuren and Hartog (2005) suggest a plan evaluation as a means to verify the consistency throughout the first phases of the design cycle and thus, to ensure that the objectives (requirements and assumptions) are valid sub-goals. The objectives for our design are based on the insights on software quality in Chapter 3. The actual design artifacts, that are put forward in the subsequent sections, should primarily support the quality criteria for pluggable services. The phase of objective definition is revisited once in Chapter 7 where the inter-organizational capabilities of platform-based services will be considered.

The *design and development* phase results in the construction of the *design artifact* which potentially meets the criteria defined during the objectives phase (design hypothesis). The development draws upon the insights from the previous design cycle. For example, our first design reflects the capabilities of current integration platforms (cf. Chapter 4). During its evaluation, we found that data management components could be beneficial to the design. Thus, we added it to the platform architecture of the subsequent design phase in Chapter 5.

The *demonstration* phase takes the design artifact as an input for the creation of a *design product*. The design product can be considered as the instantiation of the design artifact that demonstrates the feasibility of the design and facilitates the verification of the design hypothesis. Furthermore, we establish a business case during each demonstration phase which allows showing the practical relevance of the design.

The *evaluation* uses the design product to test the design hypothesis. According to Verschuren and Hartog (2005), product evaluation involves a measurement at two different points in time, once before and once after introducing the design product. For example, we compare the pluggability of services with and without the use of the platform. The evaluation of the product can be done through the use of an instrument or through logical reasoning (Peppers et al., 2007). In the first three cycles, we rely on the instrument of pluggability which we propose in Chapter 3. In the last design cycle, we discuss a business case to investigate the collaborative capabilities of the platform.

1.3 Dissertation outline

In the previous sections, we presented the goals for our research and established a number of research questions. Furthermore, we have put forward a methodological framework that relies on various design science research contributions. Table 1.2 summarizes the different concepts and maps the remaining chapters of this work to the established framework. In the following we discuss each chapter individually.

Chapter 2

The development of information systems often follows a model driven approach (Fettke and Loos, 2003). To understand the capabilities of current e-commerce systems various reference models in this field can be consulted and reused (Becker and Schutte, 2007; Frank and Lange, 2007). Reference models help software producers to implement standard software packages that can be used many organizations. At the same time the models help the adopters of software packages to analyze the fits and misfits with their current processes. Particularly small and medium sized companies struggle with the implementation of such comprehensive prepackaged solutions, due to the complexity of the systems and the underlying business model (Soh et al., 2000). A new paradigm to overcome these shortcomings and to improve system agility is “to design many business activities as Lego-like software components that can be easily put together and taken apart” (Merrifield et al., 2008). A suitable model to design such architectures, that we are going to use for the architecture of the platform, is the ArchiMate modeling language for enterprise architectures, which has been designed with the service-oriented paradigm in mind (Lankhorst et al., 2009). Based on a systematic literature review, we present an architectural reference model for online retail that encompasses all three layers of the ArchiMate metamodel. The reference architecture is evaluated by means of a case study of the IT landscape from a fulfillment center.

Chapter 3

Software quality models help organizations to assess the state of their application systems and have been subject to research for a couple of decades (McCall et al., 1977; Grady and Caswell, 1987; Dromey, 1996). However, the stakeholder of the existing quality models is mainly concerned with the internal structure of software components. With the shift towards cloud offerings, a different conception of quality arises. Current quality models consider resource owner and provider as a single entity. Thus, most of the quality characteristics covered in the models reflect the internal view of a software service (Ortega et al., 2003). To address the deficiency, Chapter 3 puts forward a new quality characteristic of pluggability that reflects the priorities of the service user as a distinct party. It implies six different phases of service consumption. We introduce the notion of pluggability to illustrate the aspired user experience of a service and propose a corresponding measurement instrument.

Table 1.2: Mapping of thesis to methodological framework

	Chapter 1	Chapter 2	Chapter 3	Chapter 4	Chapter 5	Chapter 6	Chapter 7
Method		Systematic literature review	Literature review	Market analysis, prototyping	Prototyping	Prototyping	Prototyping
Artifact		Reference architecture	Instrument	State-of-the-art architecture	Platform architecture	Pluggable service	Collaborative architecture
Evaluation		Business Case	Interview	Instrument	Instrument	Instrument	Business Case
Business case		Fulfillment center architecture		Goods return flow	Cross-border compliance	Sales forecasting	Cross-selling
DSRM cycle (Peffers et al., 2007)	Motivation	Motivation	Objectives	Design cycle 1	Design cycle 2	Design cycle 3	Design cycle 4
Research goal		RG1	RG3	RG1/RG2	RG2/RG3a	RG2/RG3a	RG2/RG3b
Research question		SQ1	SQ2	SQ3	SQ4	SQ4	SQ5
Relevance cycle (Hevner et al., 2004)	In: E-commerce consortium requirements		Out: Pluggability instrument	In: Existing platforms and services	In: E-commerce canonical data model	In: Platform architecture, PaaS offerings	In: Platform requirements
				Out: Return flow prototype	Out: Platform architecture and trade compliance service prototype	Out: Sales forecasting service prototype	Out: Collaboration platform architecture, canonical data model, cross-selling service prototype
Rigor cycle (Hevner et al., 2004)	In: Research methods and core concepts	In: E-commerce literature and reference models	In: Quality models and frameworks	In: Pluggability instrument		In: Statistical model and R module, pluggability instrument	
	Out: Research goals	Out: Architectural reference model	Out: Service lifecycle and pluggability model	Out: State of the art pluggability assessment		Out: Forecasting analytics service and evaluation	

The pluggability criteria are the major objectives for the design in the subsequent chapters and the instrument will be used as a means to evaluate the design.

Chapter 4

In the fourth chapter, we investigate the currently available e-commerce solutions and integration platforms in the market. The review supplements the state-of-the-art in e-commerce architectures from Chapter 2 with a perspective on available software products. We combine the findings of both chapters and propose a state-of-the-art platform architecture model for e-commerce which represents the first design artifact in the design cycle. The design is instantiated by means of a prototype which reflects an e-commerce returns handling scenario. The design product reveals the capabilities of currently available solutions and is evaluated by domain experts using the pluggability instrument established in Chapter 3. We conclude with a documentation of the shortcomings in current solutions and propose an extended reference architecture model.

Chapter 5

The extended reference architecture from Chapter 4 is taken as a basis for the second design cycle outlined in Chapter 5. The collaborative data management component of the extended model is reflected in the platform architecture in the form of a canonical data repository that allows the reuse of resources across services. Furthermore, the platform architecture implements a resource authorization process which is derived from similar mechanisms in social media. The implementation of the prototype is making use of existing software libraries available from social media and applies them in the enterprise information system context. For the evaluation of the design product a business case is established which reflects the cross-border e-commerce scenario. An existing global trade compliance service is transformed into a pluggable service. A comparison between the two services is carried out with regard to the six criteria of pluggability.

Chapter 6

In this chapter, we continue elaborating on the capabilities of the proposed platform architecture to support easy adoption of e-commerce services with relevance for cross-border and omnichannel retail. Business models that include ship-from-store offerings and globally distributed warehouse locations require a thorough stock allocation planning. Particularly in the fashion domain with short series lifecycles, excess stock or out of stock can lead to major losses. Sales prediction models support the task of stock allocation planning as they provide an approximation to future sales figures per channel and stock keeping units. In Chapter 6 we present a sales forecasting service which dates from the collaboration with our project's

research partner. Their state-of-the-art sales forecasting module has been transformed into a pluggable cloud service which relies on the platform architecture and makes use of the existing e-commerce resources. We propose the architecture for the service and present its implementation. We compare the pluggability of the sales forecasting module and the cloud-based sales forecasting service. We show that the previously required tailoring of solutions for data analytic tasks can be overcome and replaced with ready-to-use pluggable services.

Chapter 7

In the beginning of the final design cycle, we revisit the design objectives phase. We extend the focus from assessing the technical aspects of service pluggability to the quick connect capability among business partners. We hypothesize that the complex workings of B2B partnerships can be encapsulated into a collaboration service. Through the platform architecture, such service can be consumed by collaborating partners in a pluggable fashion. Thus, leading to increased quick connect capability of the platform clients and users. We demonstrate the idea based on a cross-selling service. Cross-selling is a popular means to attract new customers in the highly competitive market. We propose a service that allows to match products and refer customers among shops in an ad-hoc manner. The service handles the technical interoperability, the contract, and the settlement of commissions among cross-selling parties.

Synopsis

Figure 1.5 shows how each chapter is oriented towards the DSRM. In Chapter 2 we define the objectives through the construction of a quality model. In Chapter 5, Chapter 6, and Chapter 7 we present the iterations of design artifacts and design products. The design artifact is always drawn upon the design artifact of the previous phase (inheritance relationship). Furthermore, each new artifact takes as an input the findings of design product construction (generalization). The design product is based on the design artifact (instantiation) and the objectives for construction and evaluation. In Chapter 7 the objective definition phase is revisited and the design product gets evaluated based on the extended objectives.

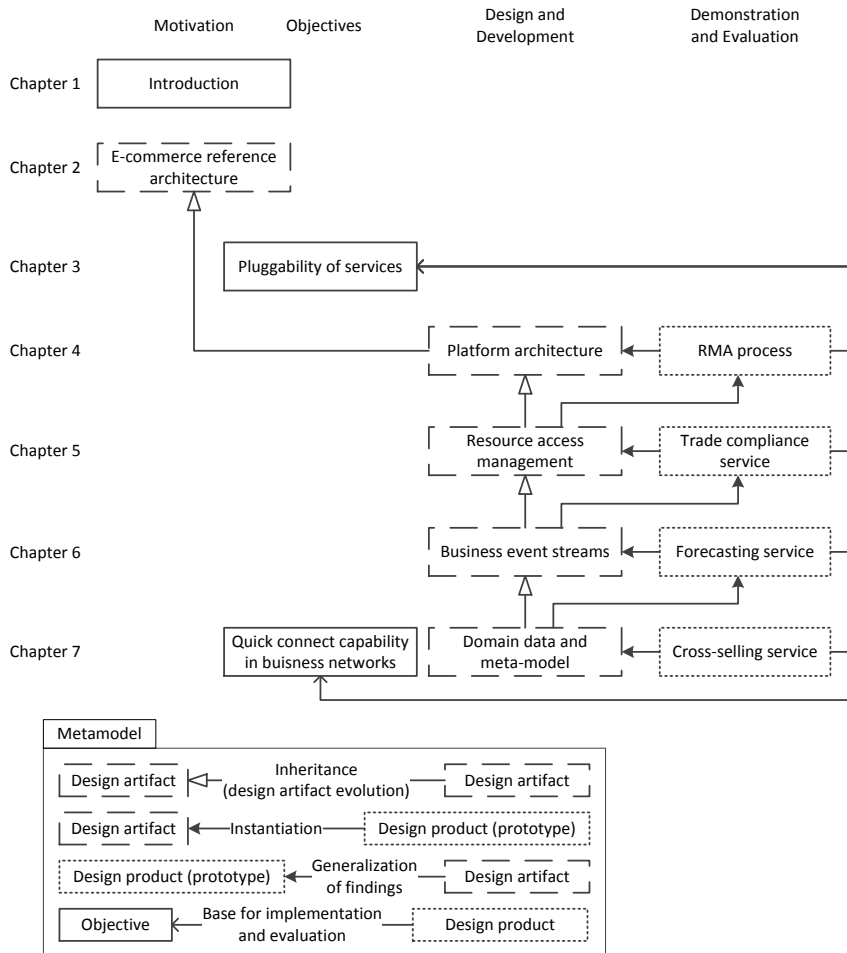


Figure 1.5: Application of the design cycle

Chapter 2

The State of the Art in E-Commerce Architectures

The coordination of processes is made possible for every e-commerce company by different software components with specific functionality. A single e-commerce transaction involves many software components that provide different services such as searching and browsing products, initiating a transaction, or payment of an order. Often, these components are bundled into a single software application which makes it challenging to change or add services to the existing landscape. However, in recent years the lack of flexibility of the monolithic software platforms has been addressed through a new software development paradigm, namely the service-oriented architecture approach (Merrifield et al., 2008).

Similar to the growth of the online market, the sheer amount of approaches and technologies has increased as well. This makes the development of an aligned, change responsive, and efficient software architecture more difficult. Especially small retailers, that want to establish an e-commerce channel along with their traditional business, struggle to oversee the large number of available products and architectures, from both the functional and technical expertise points of view. The emergence of technologies, such as cloud computing, and approaches like service-orientation, complement and compensate traditional applications. Nevertheless, they still require a profound understanding of many technical and architectural aspects.

The change and the variety of the users' demands is another source of challenge. We can observe a shift from the established web applications to fully-functioning mobile application in the last years. With this development, offline and online shopping experience will merge and bring new challenges and opportunities for retailers. According to the European Commission (2015), 20% of the EU citizenship is expected to buy goods from abroad in 2020. Following the internationalization, additional challenges and competition for e-commerce companies arise. These challenges range from supply, delivery to the payment issues. According to Weinfurter et al. (2013), the dropout rate because of the mode of payment varies between 1% and 88%. Given the fact that the preference for different modes of payment differs significantly per country (e.g., the Dutch prefer their national solution iDEAL (Blauw Research and GfK Retail and Technology, 2012) while Germans prefer to purchase on account (Weinfurter et al., 2013)), satisfying the customer is an on-

going challenge.

The emerging challenges can be summarized, as lack of:

- flexibility of the existing, monolithic systems.
- insight into all application and technology components due to the lack of existing reference architectures.
- knowledge regarding the impact of emerging technologies on current architectures.
- integration between online and offline channels.

The concept of service-orientation should overcome the mentioned challenges as it makes the previously bundled services explicit. It gives insights into what components existing architectures are built upon. It also allows to plan and implement new technologies and facilitates the integration with other, previously separated, systems. To come up with a suitable service-oriented reference model for e-commerce, an overview of the entire architectural building blocks and offerings is required. It allows making the right choices when designing an e-commerce architecture and is therefore put forward as research goal of this chapter. In the remainder of this chapter, we discuss currently used e-commerce architectures and derive best architectural practices from them. The main contribution of this chapter is an exhaustive study of the literature on e-commerce architectures. It is complemented by an E-Commerce Reference Architecture (ERA) which extends the known models with a concrete catalog of architectural artifacts and their relationships. The proposed architecture gives an overview of the functional building blocks which are required to fully support the execution of an e-commerce transaction and all other related activities (e.g., fulfillment, customer support, returns, etc.). It should be noted that, for now, we are not concerned with the exact implementation of the aforementioned building blocks.

The amount of literature on e-commerce is vast and the relevant information is scattered. Additionally, the main themes of this study, such as platform, architecture, or service tend to be overloaded as they are applied distinctively across the different sub-domains of information system research: architecture can refer to business or enterprise architecture, to application architecture in the context of software engineering but also to infrastructure architectures, such as networking or even hardware. As a consequence, the chosen approach for this study has to be extensive to the extent that no relevant publications are overseen and strictly selective to limit the scope to the research questions. Thus, we carried out a systematic literature review as proposed by Kitchenham (2004). The method encompasses two distinct phases. First, a search strategy is put forward and subsequently, a study selection is carried out. We applied a double coding strategy to minimize subjectivity during the selection phase. Our data extraction goes beyond the usual categorization of literature and themes. Instead, we are going to derive a reference model to

summarize our findings. For the introduction of such a new reference architecture, we made use of the design science research methodology from Peffers et al. (2007).

2.1 Research design

This chapter combines a literature study, which aims at identifying the current state of the art in e-commerce architectures with the development of a reference architecture. For this purpose, a *multi-method* approach is applied consisting of a *systematic literature review* nested into a *design science* research cycle.

Figure 2.1 visualizes the process we followed. The design science research methodology described by Peffers et al. (2007) includes the following phases: 1) problem identification 2) definition of objectives 3) the development of an artifact and 4) its validation. The artifact that is proposed in this chapter is an architectural model that outlines all the building blocks for an e-commerce solution and is based on the architectures proposed in current academic contributions. The list of relevant literature and its categorization is presented as an intermediary artifact prior to the development of the architecture.

To systematically analyze the existing literature, we follow the approach of Kitchenham (2004). This means that the process incorporates three steps: study selection, study qualification, and data extraction. The study selection is further divided into the definition of search criteria, the collection of the papers, and the application of the selection criteria.

The remainder of this chapter is structured according to the described research process. In the next section, we elaborate on the objectives of the research and the nature of its contribution. In Section 2.3 we describe the literature review process in depth and explore the qualitative and quantitative aspects of the literature list. In Section 2.4 we come up with the reference architecture. In Section 2.5 we are evaluating the architecture and, at the same time, the state of the art in e-commerce architectures.

2.2 Objectives

The goal of a reference model is to provide a working solution for a class of common design problems. To achieve this goal, reference models adhere to three characteristics, namely covering best practices for solving problems in practice, being universally valid and thus applicable to a class of problems, as well as being reusable and customizable or adaptable to different implementation scenarios (Fettke and Loos, 2007). The presented architecture can be used equally by scholars and practitioners to further investigate the topic or to build solutions based on the aggregated knowledge on e-commerce architectures. As the architecture is derived from the current e-commerce research, it also gives insights into which aspects are overrepresented or underrepresented in the current papers in the field of e-commerce. Fettke and Loos (2007) propose a framework to approach reference modeling by looking at

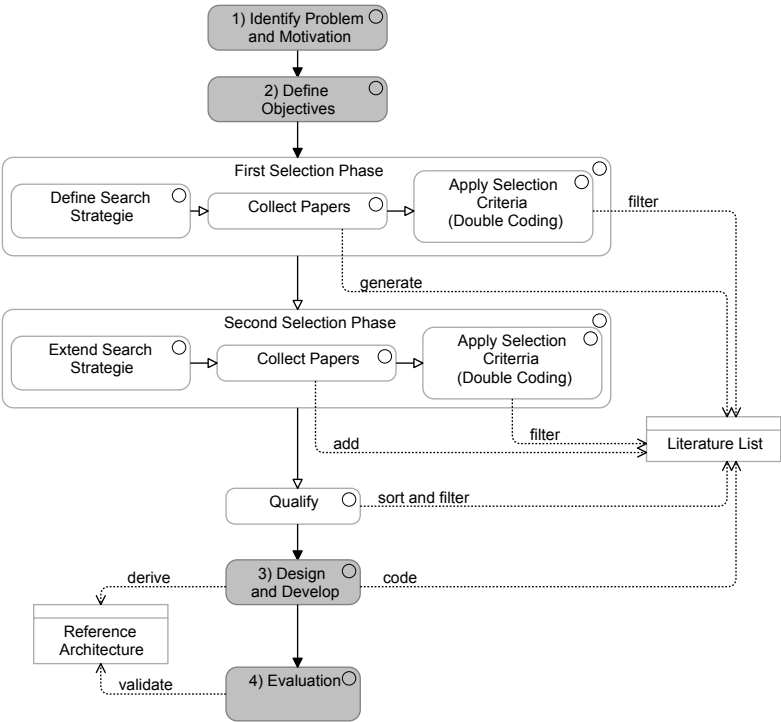


Figure 2.1: Mixed method approach

the context, existing reference models, and the modeling language. In the remainder of this section, we use this framework to outline the objectives for the e-commerce reference architecture model.

2.2.1 Context

The context of the reference model covers the technical, organizational, and economic scenario in which the modeling process is embedded. According to the motivation for our research, we focus on an e-commerce scenario in a retail context, where the business model of the retailer is to buy and store a large quantity of goods and resell them in smaller quantities (cf. Appendix B). This e-commerce process generally does not include the production of goods but it can include certain activities of product finishing or customization. It usually includes additional services before, during, and after the selling of goods. The process includes business-to-customer (B2C) as well as business-to-business (B2B) e-commerce transactions, between the e-commerce company, its suppliers, partners, and customers.

In this work, we are focusing on an enterprise architecture that reflects business processes and the required IT resources. The latter should be identified in a service-oriented manner to allow the users of the model to emancipate themselves from the application centered thinking when designing their landscape. Neither the implementation of the services nor how they might be bundled in various application systems are important. The system owner should be able to choose and exchange the IT services according to the business needs and overcome the limitations of monolithic business applications. We are going to cover internal business services, and more importantly, services that allow communication with business partners such as suppliers and logistic service providers.

2.2.2 Existing models

Several related reference models can be found in the scientific literature. The models focus on the business process of an e-commerce enterprise and do not present a comprehensive architecture for e-commerce. In the following, we present four prominent models and discuss their limitations with regard to the context of this study as well as their potential reuse in the architectural model.

The H-Model for retail from Becker and Schutte (2007) provides a domain-specific reference architecture for retail enterprises. The model describes eleven tasks within the three major groups of procurement, storage, and distribution (Table 2.1). Different views allow the extension or simplification of the model according to the business requirements. The model is based on the Architecture of Integrated Information Systems (ARIS) framework and contains three viewpoints which are business functions, processes, and static data. The model originates from a traditional retail business model and omits some functional tasks that are specific to e-commerce.

Frank and Lange (2004) and Frank (2004) establish a library of 85 business process models divided into nine business functions. Table 2.1 contains a mapping of these business functions to the functional tasks of the H-Model.

Burt and Sparks (2003) investigate the impact of e-commerce on the retail process. Their procedural model is used as a framework to compare activities, ownership, costs, and efficiency in an e-commerce setting against traditional retail. According to this model, the retail process comprises five business functions, which we mapped onto the aforementioned models in Table 2.1. Their view on the retail process is in line with the 8 business functions identified by Gunasekaran and Ngai (2004) for a similar purpose.

Some other reference models such as Croxton (2003) exist which provide a specific focus on parts of the overall business process, such as order fulfillment and supply chain matters. These reference processes might be relevant on a more detailed level of an ERA.

In none of the above-mentioned models return handling is considered as a primary business function. However, in online retail return handling is a critical activity, as the volume of returned goods is much higher compared to the traditional

Table 2.1: Business functions in the existing reference models

Becker and Schutte (2007)	Frank and Lange (2004)	Burt and Sparks (2003)	Gunasekaran and Ngai (2004)
Contracting	Procurement	Sourcing	Supplier development
Order management	Procurement	Sourcing	Purchasing
Goods receipt	Reception	—	—
Invoice auditing	Payment	—	—
Accounts payable	Payment	—	—
Warehousing	—	Stockholding	Warehousing
Marketing	Pre-sales communication	Marketing	Marketing
Marketing	Customer relations	Branding	Design
Selling	Initiation and pricing	Customer selection (pick and pay)	Sales
Good issue	Order processing	Distribution	Distribution
Billing	Order processing	Distribution	Distribution
Accounts Receivable	Collection	—	—
Return Handling (in goods issue)	—	—	—
—	Customer service	—	—

retail (Banjo, 2013). A proper return handling strategy is essential for online retailers in order to limit the expenses caused by the high volumes. Furthermore, in three of the models, an explicitly defined customer service business function is missing. The online transaction lacks a personal contact as it is present in a brick and mortar store. Thus, in case of individual inquiries or exceptions, online customers need to be provided with a telephone hotline and e-mail help center.

Apart from the minor functional gaps, existing retail models do not cover the information and technology layers which are essential for an enterprise architecture. Thus, building an information systems based on the available models will eventually lead to solutions that support either one specific business function or the end to end process. These are in most cases either a number of horizontally oriented systems, each supporting singular business functions, or monolithic sys-

tems that are not able to support the integration of specific application services. To overcome these limitations, a reference architecture model should include all the required IT services which can be reused across business functions. Such a model will eventually lead to an open and agiler architecture.

2.2.3 Metamodel

The modeling language of a reference model is crucial because it defines the formal framework and viewpoints to describe the system. The H-Model and the E-MEMO model presented previously are using the ARIS framework and the process modeling language MEMO-OrgML respectively. To facilitate the shift from these process centric modeling frameworks to a more comprehensive architectural view of the information system, the presented ERA is specified using the ArchiMate language (The Open Group, 2016) which was developed to model all the aspects of complex organizations. A description of the ArchiMate metamodel can be found in Appendix A.

2.3 Systematic literature review

Lee et al. (2011) provide an overview of e-commerce related literature studies. According to their work, existing literature reviews belong to either of the following two categories: 1.) generic reviews identifying themes and methods in e-commerce across disciplines (Clarke, 2000; Lee et al., 2007; Ngai and Wat, 2002; Wareham et al., 2005) or 2.) studies focusing on one specific aspect of e-commerce, namely economic theories (Kauffman and Walden, 2001), marketing (Schibrowsky et al., 2007), customer relation and satisfaction (Chen et al., 2008; Romano Jr. and Fjermestad, 2003), adoption (Chitura et al., 2008), payment (Dahlberg et al., 2008) and marketplaces (Wang et al., 2008). Considering this categorization, our contribution falls into the second category of studies aggregating existing knowledge from literature with regard to a specific theme. More specifically, our focus lies on the *identification of building blocks for e-commerce architectures* discussed in recent e-commerce studies. The aggregated knowledge will allow us to develop a state-of-the-art e-commerce reference architecture, based on the research of the last decade. In this section, we describe the strategy and results of the research process.

2.3.1 Data sources and search strategy

In order to capture a maximum number of relevant studies, we carried out two selection cycles. The search query for the first selection cycle was developed collectively by three researchers. The search query of the second selection phase reflects synonyms and related search terms that have identified during the first selection phase. The following search query was defined for the first selection phase:

(E-commerce OR electronic commerce) AND
(architecture OR SOA OR EA OR reference model OR platform)

Due to the generic nature of some of the used search terms and to increase the ratio of relevant contributions we limited the search to the title and abstract of the publications. We also limited the search to publications that were published within the last ten years as we want to investigate the state of the art. The literature review was carried out in 2013. The year 2003 can be considered as an adequate time horizon, as the modern web (web 2.0 and web services) were introduced around that time. To obtain scientific literature matching the search query, we used six electronic indexing sources. Due to the high number of duplicates during the first selection cycle, we conclude that adding more indexing sources to the list will not significantly increase the number of relevant papers. The indexing sources we used are:

1. Web of Science (<http://apps.webofknowledge.com>)
2. CiteseerX (<http://citeseerx.ist.psu.edu>)
3. ScienceDirect (<http://www.sciencedirect.com>)
4. IEEEExplore (<http://ieeexplore.ieee.org>)
5. ACM Digital Library (<http://dl.acm.org>)
6. Springer Link (<http://link.springer.com>)

Some of the above-mentioned indexing services allow the direct entry of the search query, while some require multiple searches to achieve the same. After omitting the duplicates and scanning the titles 880 documents were taken into the selection phase described in the subsequent section.

After selecting the papers, the search terms have been extended to cover synonyms and related terms that turned out to occur often in the abstract of relevant publications. In the second selection phase the following query was applied:

(E-commerce or ecommerce or electronic commerce) AND
(component or model or process or
framework or platform or service or cloud)

After pre-selection and removal of duplicates, 3292 additional papers entered the selection phase.

2.3.2 Study selection and study quality assessment

To select relevant publications from the large list of documents we filtered them based on their title, abstract and available meta-information by using the following inclusive criteria:

- The document is published in a journal, conference, workshop, as a technical report, thesis or book (chapter) to ensure that search results meet certain academic quality standards and have been peer reviewed.
- The document proposes an architecture for e-commerce scenarios or discusses design principles or best practices.

All papers have been scanned independently by two researchers to increase the objectivity of the results. After applying the above filtering criteria a list of 864 papers has been generated. To further limit the amount of papers considered for this study the papers have been rated with scores from 1-5 by scanning the full content of the paper:

- 1 Point: Has no relationship to e-commerce architecture at all
- 2 Points: Discusses one aspect of e-commerce architecture
- 3 Points: Discusses several aspects of e-commerce architecture
- 4 Points: Proposes a partial architecture
- 5 Points: Proposes a full architecture

The results of the scoring of both researchers have been then merged: papers where the scores differed by more than one point have been further discussed by two researchers resulting in an agreed upon adjustment of the score. All papers with an average score of more than 3.5 are included in the final literature list resulting in a total of 48 papers (Albrecht et al., 2005; Baghdadi, 2004, 2005; Baida et al., 2003; Baquero et al., 2012; Basu and Muylle, 2003; Chang and Wang, 2010; Chen et al., 2007; Chen and Su, 2011; Chou and Lee, 2008; Dorn et al., 2007; El Ayadi, 2011; Esswein et al., 2004; Feipeng and Qibei, 2010; Fragidis and Tarabanis, 2008; Frank, 2004; Ganguly and Bhattacharyya, 2011; Gao et al., 2009; Hashemi et al., 2006; Hernandez et al., 2005; Hu et al., 2009; Iqbal et al., 2013; Kim et al., 2003; Jiang and Song, 2010; Jiao and Helander, 2006; Kim and Smari, 2005; Lackermair, 2011; Lan et al., 2008; Li and Dong, 2010; Liu and Hwang, 2004; Liu et al., 2005; Medjahed et al., 2003; Mohamed et al., 2010; PengLing and Mian, 2010; Sabki et al., 2004; Sun et al., 2012; Tan, 2011; Tenenbaum and Khare, 2005; Thomas et al., 2008; Wan et al., 2013; Wan and Huang, 2008; Wen, 2007; Xu et al., 2010; Yang et al., 2007; Ying and Dayong, 2005; Zhang et al., 2009; Zheng et al., 2009a,b)

The literature review has been carried out based on the papers of this final selection. A coding technique has been applied in order to extract categories from the paper (Saldaña, 2012). By using in-vivo codes two researchers extracted the relevant architectural concepts from the papers and later grouped them into semantically identical categories. This way we maximize objectivity while identifying the architectural building blocks and minimize the effect of projecting the researchers' assumptions on e-commerce architectures when applying predefined criteria to the sample. In the next section, we are going to present the results of the coding.

2.3.3 Results

The purpose of this section is to give an extensive overview of the academic research on e-commerce architectures since 2003. To do so, a qualitative and a quantitative analysis have been carried out. We deem both analyses as necessary because the quantitative analysis helps us to get a general overview of the trends and topics in academic research on e-commerce architectures, while the qualitative analysis gives insights into the content of the papers and forms the base for the identification of artifacts of the reference model. Four different perspectives are reflected to classify and analyze the literature.

In the first step, the papers were analyzed from a technological point of view. This dimension describes the underlying software technology or the framework used to design the solution. Both are included, since not all papers design a technology oriented architecture, but may take other perspectives such as knowledge, stakeholders, or process perspectives. By doing this, we provide an overview of the used technology and extract relevant information for the reference model. Secondly, different types of artifacts have been proposed by the various authors (e.g., an architecture description, model, or a framework). This dimension aims at classifying the selected papers according to the type and nature of the proposed artifact. The third dimension is concerned with application functionality, i.e. the functional building blocks (e.g., the application services) of the proposed solutions. According to the ArchiMate metamodel, these could be components, services, application functions, objects, etc. of the proposed solutions. Extracting them from the selected literature provides us with an overview of the most relevant and most discussed functional building blocks which will be further used as the basis for the application layer of the reference model. Similarly, the fourth dimension is concerned with technological building blocks of the proposed solution such as devices, nodes, system software and infrastructure functions. These form the basis for ERA's technology layer.

The analysis of the papers in scope with respect to the used technology (Figure 2.2) revealed the extensive use of service-oriented technologies in e-commerce. Out of the 48 papers, 28 are related to service-oriented architecture. The second most frequently mentioned technology is component-based with six papers. Three of the analyzed papers discuss agent-based architectures. Two papers have focused on processes and another two on workflow architectures. It should be noted that papers which include more than one technology (e.g. Chen et al., 2007) are only represented once depending on the main technology. Two papers (knowledge-based and stakeholder-viewpoint) take a unique viewpoint and use approaches which are normally outside the scope of enterprise architectures. Four papers do not mention any technology and propose a generic and high-level architecture. In such case, a mapping to a specific technology was not possible.

In the selected articles, authors referred to technologies in relation to the development of different artifacts (Figure 2.3). 22 papers proposed an architecture, in which they define how services or components interact with each other, and how

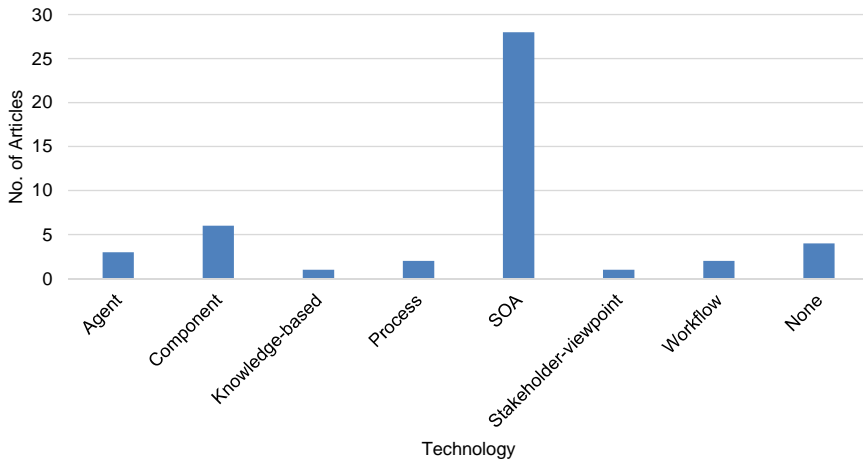


Figure 2.2: Number of articles per technology

they are layered. Five papers analyzed commonalities of architectures (namely by proposing reference architectures). In terms of frequency, the architecture papers are followed by eight model papers and three reference model papers, which propose specific designs. The five framework papers focus on the role of the services, components, etc.

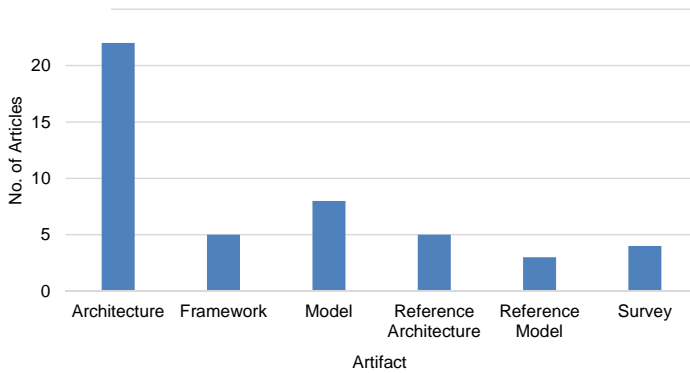


Figure 2.3: Number of articles per artifact

When analyzing the selected articles from a functional perspective, one can see a great variety of the implemented functions. A total of 85 functional features is proposed. The 85 features are mentioned in 37 articles (10 articles do not propose any functionality). Table 2.2 shows the functions mentioned in at least two papers. These are 19 functions (22.4% of the total amount of functions). Looking at the most frequently analyzed features (occurring in more than 3 articles) one gets an overview of the main application layer concepts supported by e-commerce solutions. Especially order (18 articles) and payment (13 articles) functions are popular and analyzed by many authors. While most of these functional building blocks are common to traditional shops as well, search and product information are rather technical and specific to the electronic channel.

Categorizing the articles into functional categories across the supply chain gives additional insights (Figure 2.4). While only seven of thirty-seven articles cover the upstream supply-chain activities (procurement), 36 of 37 articles analyze the downstream activities (distribution). The only paper which discusses functional aspects but leaves out distributive parts is Chou and Lee (2008), with a focus on customer service and integration. 16 articles discuss customer service related functions such as customer relationship management. Three articles discuss warehouse related topics such as stock management. 22 articles analyze functions which do not fit into this scheme. These functions vary from business related topics, such as general audits, to technical topics, such as certificates or central processing.

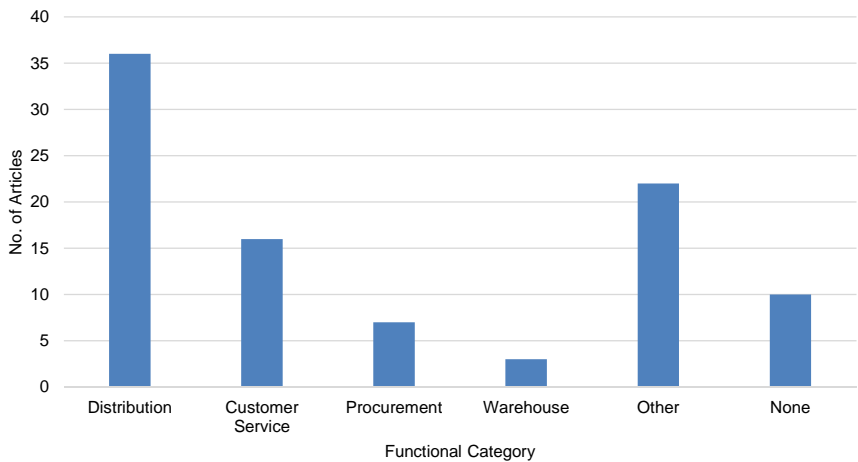


Figure 2.4: Number of articles per functional category

Besides a main technology, we also identified specific technology related concepts discussed in the selected papers. Those concepts form the basis for the technology layer of the reference architecture. 15 different concepts have been covered in the papers, which is significantly less than the functional concepts we discussed

Table 2.2: Publications per application layer concept

\sum	Application Concept	Articles
18	Order	Baghdadi (2004, 2005); Chen et al. (2007); Dorn et al. (2007); Gao et al. (2009); Iqbal et al. (2013); Jiao and Helander (2006); Lackermair (2011); Lan et al. (2008); Li and Dong (2010); Liu and Hwang (2004); Liu et al. (2005); Sabki et al. (2004); Tan (2011); Thomas et al. (2008); Wan et al. (2013); Wen (2007); Yang et al. (2007)
12	Payment	Basu and Muylle (2003); Feipeng and Qibei (2010); Iqbal et al. (2013); Kim et al. (2003); Jiang and Song (2010); Lackermair (2011); Lan et al. (2008); Li and Dong (2010); Liu and Hwang (2004); Liu et al. (2005); Medjahed et al. (2003); PengLing and Mian (2010); Zheng et al. (2009b)
8	Logistics	Basu and Muylle (2003); Feipeng and Qibei (2010); Jiang and Song (2010); Lackermair (2011); Li and Dong (2010); Tenenbaum and Khare (2005); Ying and Dayong (2005); Zheng et al. (2009a)
6	Finance	Mohamed et al. (2010); Sabki et al. (2004); Tenenbaum and Khare (2005); Wen (2007); Xu et al. (2010); Zhang et al. (2009)
5	Product Information	Chen et al. (2007); Lackermair (2011); Li and Dong (2010); Liu and Hwang (2004); Sabki et al. (2004)
	Negotiation	Baquero et al. (2012); Kim et al. (2003); Liu and Hwang (2004); Medjahed et al. (2003); Wan et al. (2013)
4	Procurement	Baghdadi (2004); El Ayadi (2011); Mohamed et al. (2010); Tenenbaum and Khare (2005)
	Search	Basu and Muylle (2003); Li and Dong (2010); Liu and Hwang (2004); Tenenbaum and Khare (2005)
3	Authentication	Basu and Muylle (2003); Feipeng and Qibei (2010); Yang et al. (2007)
	Production	Frank (2004); Jiao and Helander (2006); Liu et al. (2005)
	Sales	Frank (2004); Tan (2011); Zheng et al. (2009a)
	Shipping	Lackermair (2011); Liu et al. (2005); Medjahed et al. (2003)
2	Bank	Yang et al. (2007); Zheng et al. (2009a)
	Billing	Kim et al. (2003); Medjahed et al. (2003)
	Contract	Baquero et al. (2012); Fragidis and Tarabanis (2008)
	Customer service	Chou and Lee (2008); Tan (2011)
	Fulfillment	El Ayadi (2011); Mohamed et al. (2010)
	Product	Lan et al. (2008); PengLing and Mian (2010)
	Purchase	Baghdadi (2005); Medjahed et al. (2003)
	Sourcing	Frank (2004); Tenenbaum and Khare (2005)
10	None	Albrecht et al. (2005); Baida et al. (2003); Esswein et al. (2004); Ganguly and Bhat-tacharyya (2011); Hashemi et al. (2006); Hernandez et al. (2005); Hu et al. (2009); Kim and Smari (2005); Sun et al. (2012); Wan and Huang (2008)

earlier. Also, the consensus between the different papers is higher in this respect as only three concepts have been found in less than 2 papers. The results are presented in Table 2.3.

2.4 Reference architecture

Reference models are a means to facilitate the process of information system design by providing enterprises with a reusable and adaptable blueprint for a class of domains. It helps to validate current solutions and supports the development and selection of new applications (Fettke and Loos, 2007). Especially organizations with limited resources can profit from such models (Frank, 2004). Their benefits are lower costs, less expertise needed, and less time to design the architecture. A model discussed and accepted by the e-commerce community is likely to lead to less error prone architectures as it reduces the risk of failures or gaps.

In this section, based on the four reference models discussed in Section 2.2.2 and the conclusions of the literature review, we propose a new extended ERA. The main differences between our model, on one hand and the existing models, on the other hand, reside in the scope and in comprehensiveness: all existing reference models are process models, while ours is a reference architecture model that goes beyond business process specifications. More precisely, in this section, we cover the concepts belonging to the business, application, and technology layers. This includes the application services, that enable business processes, and the technology services, that are used internally to allow the provisioning of the application services.

The approach we followed in designing the new reference architecture is based on the methodological foundation described in Section 2.2.3. The business layer of the ERA is based on the context and the existing process models presented in the same section. The application and technology layers are then derived from the findings of the literature review in Section 2.3.3. Figure 2.5 shows the three layers of the ERA as well as the relationships within and across the layers. In the following we are going to elaborate on each layer individually.

2.4.1 External business services

On the business layer, from an external behavior viewpoint and according to the context of our model, and e-commerce business provides three types of service, namely pre-trade, trade, and post-trade services (Liu and Hwang, 2004). A more in-depth investigation of activities and tools for external services in e-commerce is provided by Singh (2002). In the online retail scenario pre-trade services provide product information and encompass all activities which allow the end customer to gather information and consultancy about available products, eventually leading to a buying decision.

Table 2.3: Publication per technology layer concept

\sum	Technology Concept	Articles
29	Integration and message based data exchange	Albrecht et al. (2005); Baghdadi (2004, 2005); Baquero et al. (2012); Basu and Muylle (2003); Chen et al. (2007); Chou and Lee (2008); Dorn et al. (2007); El Ayadi (2011); Feipeng and Qibei (2010); Fragidis and Tarabanis (2008); Gao et al. (2009); Hernandez et al. (2005); Iqbal et al. (2013); Jiao and Helander (2006); Liu and Hwang (2004); Lackermair (2011); Lan et al. (2008); Liu et al. (2005); Medjahed et al. (2003); Mohamed et al. (2010); Sabki et al. (2004); Sun et al. (2012); Tenenbaum and Khare (2005); Thomas et al. (2008); Wan and Huang (2008); Yang et al. (2007); Ying and Dayong (2005); Zheng et al. (2009a)
21	Frontend Applications/Web Portal	Albrecht et al. (2005); Basu and Muylle (2003); Chang and Wang (2010); Chen and Su (2011); Chou and Lee (2008); Feipeng and Qibei (2010); Ganguly and Bhattacharyya (2011); Gao et al. (2009); Iqbal et al. (2013); Jiang and Song (2010); Jiao and Helander (2006); Kim and Smari (2005); Lackermair (2011); Lan et al. (2008); Li and Dong (2010); Liu and Hwang (2004); Liu et al. (2005); PengLing and Mian (2010); Wen (2007); Yang et al. (2007); Zheng et al. (2009b)
	Database	Baghdadi (2004); Chang and Wang (2010); Chou and Lee (2008); El Ayadi (2011); Feipeng and Qibei (2010); Gao et al. (2009); Iqbal et al. (2013); Jiang and Song (2010); Jiao and Helander (2006); Kim and Smari (2005); Lackermair (2011); Lan et al. (2008); Liu and Hwang (2004); Liu et al. (2005); Medjahed et al. (2003); PengLing and Mian (2010); Sabki et al. (2004); Tan (2011); Wen (2007); Yang et al. (2007); Zheng et al. (2009a)
10	Workflow/Process Engine	Frank (2004); Kim and Smari (2005); Lan et al. (2008); Liu et al. (2005); Medjahed et al. (2003); Mohamed et al. (2010); Sabki et al. (2004); Thomas et al. (2008); Wan and Huang (2008); Zheng et al. (2009a)
	Backend Applications	Baghdadi (2004); Chou and Lee (2008); El Ayadi (2011); Ganguly and Bhattacharyya (2011); Gao et al. (2009); Iqbal et al. (2013); Lackermair (2011); Medjahed et al. (2003); PengLing and Mian (2010); Yang et al. (2007)
6	Network Infrastructure	Baida et al. (2003); Feipeng and Qibei (2010); Kim and Smari (2005); PengLing and Mian (2010); Sabki et al. (2004); Zheng et al. (2009b)
4	Process and Service Monitoring and Analytics	Hernandez et al. (2005); Lan et al. (2008); Medjahed et al. (2003); Zheng et al. (2009a)
3	Server Hardware/Virtualization	Baida et al. (2003); PengLing and Mian (2010); Tan (2011)
	Security/User Management	Feipeng and Qibei (2010); Ganguly and Bhattacharyya (2011); PengLing and Mian (2010)
2	E-Mail	Chang and Wang (2010); Iqbal et al. (2013)
	Operating System/Virtualization	Lackermair (2011); PengLing and Mian (2010)
	Business Rule Engine	Liu et al. (2005); Wen (2007)
1	Printing	Iqbal et al. (2013)
	Legacy Systems	Chen et al. (2007)[13]
	Cloud-Based Systems	Sun et al. (2012)
6	None	Esswein et al. (2004); Frank (2004); Hashemi et al. (2006); Kim et al. (2003); Wan et al. (2013); Zhang et al. (2009)

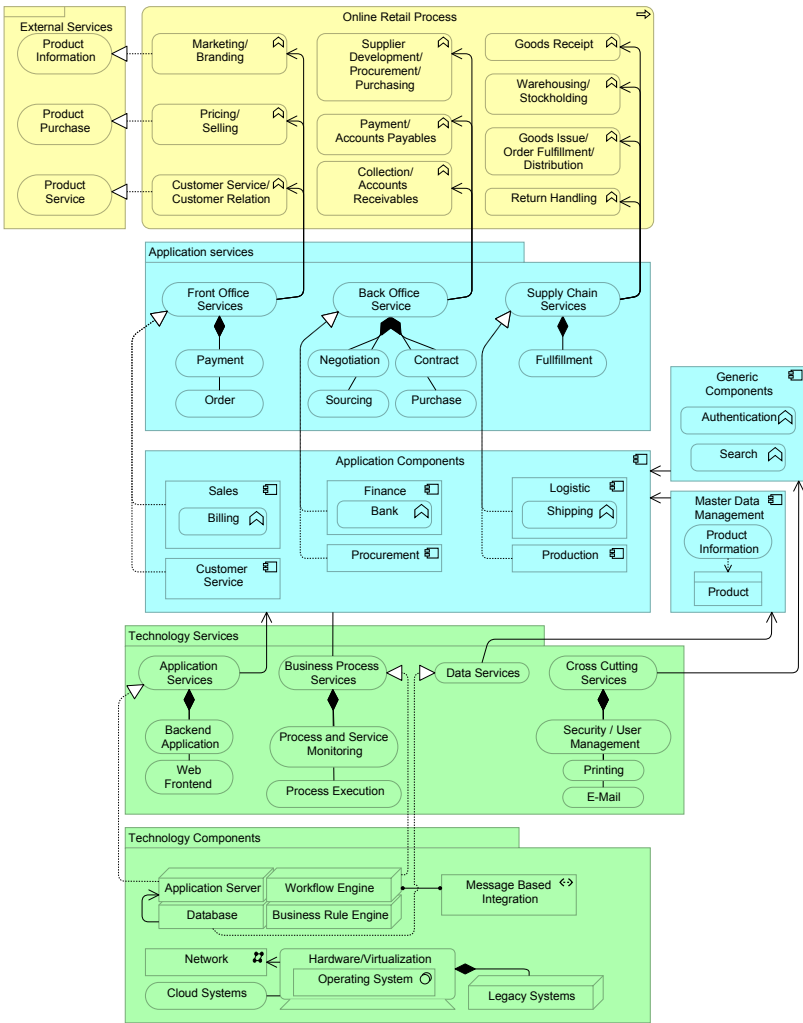


Figure 2.5: Extended e-commerce reference architecture

Product Information Services are services offered by the e-commerce company prior to the sales transaction, such as:

- searching for products to get specifications of the product and the price.
- getting subjective information on the products, such as reviews from other customers or the retailer
- generating product recommendation based on customer profile information, especially on past purchases

Product Purchase Services are the services which allow the customer to purchase the product. Those are:

- order entry allowing the customer to close a buying contract.
- payment allowing the customer to settle purchases.
- fulfillment of the contract concerning the home delivery service of products to the customer.

Post Trade Services are all the services which take place after the purchase, including:

- warranty processing services such as repair or exchange of defect products.
- support services, e.g. troubleshooting problems or software updates.
- return services for damaged products or reverse transaction of purchased goods.

2.4.2 Online retail process

Besides the high-level external business services, an internal view of the business is provided which is based on the existing reference models presented in Section 2.2.2. This is realized by building clusters of identical or closely related tasks such as Warehousing identified by Becker and Schutte (2007) and Stockholding proposed by Burt and Sparks (2003). A complete set of ten business functions has been included in the ERA.

The model proposed in Becker and Schutte (2007) is categorizing the business functions into procurement, warehousing, and distribution related tasks. The other three models do not propose any categorization of business functions. In our model, we position on the left side the business functions that are directly contributing to external customer services, including 1) marketing and branding which includes all tasks to attract and inform customers, 2) pricing and selling which includes all tasks of the actual sales transactions, and 3) customer services, which includes the tasks of communicating and responding to the customers inquiries.

On the right side of the model the internal logistic value chain is presented and includes: 1) the receipt of ordered goods 2) stockholding 3) distribution of stored goods and 4) handling of return shipments.

Further business functions include back office activities such as financial matters, for example, payment, collection, and accounting. Finally, supplier development, assortment planning and procurement can be considered as the main strategic business functions.

2.4.3 Application services and components

The application layer describes the software artifacts used by the internal business users, customers, and partners, to achieve their business goals. Active structure elements of this layer are application components which are self-contained units of functionality. They include, among others, behavioral elements such as application functions, and services. Passive structure elements are data objects.

As a result of the literature survey in Section 2.3.3 we have identified 19 different concepts that are mapped onto the different element types of the ArchiMate language. More precisely, six main application components have been identified, namely Procurement, Sales, Customer Service, Logistic, Finance, and Production. The latter has only limited applicability to e-commerce (cf. Section 2.2.1), which is confirmed by the limited contained functions and services discussed in the literature. Components such as Sales and Procurement provide a couple of application services. However, the application functions implementing those services have not been discussed in the literature. Although in general e-commerce literature provides an overview of the main application components, we get little insight into the specific details of these components.

Besides the main application components, other generic components have been mentioned in the literature, such as Authentication, and Search. These components are characterized by their universal applicability across the business specific applications. We argue that the list of such generic components could be extended with Communication, and Collaboration (including documentation), Policy Automation, Business Rules, or Data Analysis and Reporting, to name just a few.

Another building block contains the Product Information service based on the actual product data. We put these components separately as we think they do not belong to one specific application component but are shared resources across specific application services and functions. Furthermore, Product Information can be considered as just one out of several other shared resources in the e-commerce landscape. Trading Community data (customers, suppliers, and partners) and Sites are other examples of master data that are used across application components.

A mapping of business functions to the application components has been carried out to identify gaps between the two layer models, and to find out if there are any business functions that are not supported by the application layer of the model or vice versa. Figure 2.6 shows the result of this exercise. As mentioned in Section 2.2.1 production is not a critical business activity in the context of this study as

we mainly deal with the retailing of finished goods. Only finishing or customization steps are considered as production activities in this context. Nevertheless, production components have been mentioned in several e-commerce studies and, therefore, we consider them relevant during sales and order fulfillment. The procurement component is used on the supply side, i.e. when purchasing, receiving, and paying goods. The sales component is critical throughout all other business functions that deal with the actual marketing, selling, and distribution activities. We also consider it for warehousing and receivables, as stock levels and reverse sales transactions should be reported to the actual sales data. Customer service is relevant in pre-sales or marketing activities, during sales, and to provide after sales service or returns. The logistic component is used for inbound, outbound, stockholding, and returns handling. Finally, the finance component handles accounting such as receivables and payables. It should also be able to enable monetary aspects of sales and reverse transactions.

Application Component Business Function	Procurement	Production	Sales	Cust. Service	Logistic	Finance
Purchasing / Procurement	X					
Goods Receipt	X				X	
Payment / Payables	X					X
Warehousing / Stockholding			X		X	
Marketing / Branding			X			
Pricing / Selling		X	X			X
Goods Issue / Distribution		X	X		X	
Collection / Receivables			X			X
Return Handling			X	X	X	X
Customer Service				X		

Figure 2.6: Mapping of ERA's business functions and application components

2.4.4 Technology services and components

To complete the architectural model, a mapping of the concepts in Table 2.3 to the technology layer of the ArchiMate metamodel has been carried out. Despite this classification, these concepts are connected to each other.

Hardware and network infrastructures act as the fundamental basis for other concepts. Supported by these infrastructures, technological advances emerge; these advances started by Legacy Systems, which are a special type of systems that is characterized by its use of outdated technologies and its potential replacement; and continued by Cloud Systems, which can be considered as an alternative to on-site sys-

tems, and as a special type of virtualized hardware infrastructure. Each has been discussed only in one study (Chen et al., 2007; Sun et al., 2012).

Furthermore, four main active structure elements in the technology layer are discussed throughout the reviewed papers, which are Application Server, Database, Workflow and Business Rule Engine nodes. They provide the Web Frontend and Backend Application services as well as other services such as Process and Service Monitoring as well as User Management. Other services, such as Printing and E-Mail, have been mentioned in the literature without discussing the underlying components.

The most popular state-of-the-art method for integration of internal and external systems is message-based Integration, such as Electronic Data Interchange (EDI), web services, or other more specific message-oriented middleware. More recent approaches (e.g. Representational State Transfer) or traditional technologies such as file transfers have not been found in the e-commerce literature.

2.5 Validation

To validate the ERA we carried out a single case study as proposed by Wieringa (2014). The goal of the case study is to map the ERA to a real-world case in order to identify potential gaps in the model. The chosen case originates from the order-to-delivery practice of a major Dutch online retailer for media and books. The architecture of the various key processes was gathered during a series of workshops in cooperation with the head of the IT organization and the IT architect. In what follows, we are going to present the high-level architecture, including four key processes as well as the executing and contributing IT components.

2.5.1 Case study

The case study reflects four core processes in e-commerce and the required IT systems. The four processes are chosen with the goal of obtaining a comprehensive view of all IT systems. This means, adding another core process to the architecture will not lead to any changes in the application layer. Figure 2.7 shows an overview of processes and systems. It includes pre-sales activities in the form of sales campaigns, the actual sales transaction and the customer credit check, as well as the picking and packing of the orders. In the following we are referring to the different layers of the ERA for validation.

2.5.2 Online retail processes

The ERA contains ten different business functions, which contribute to the various processes. These are three front office business functions responsible for pre-sales, sales, and post-sales activities, three back office business functions, which can be further divided into strategic and operational functions, as well as four different supply chain functions.

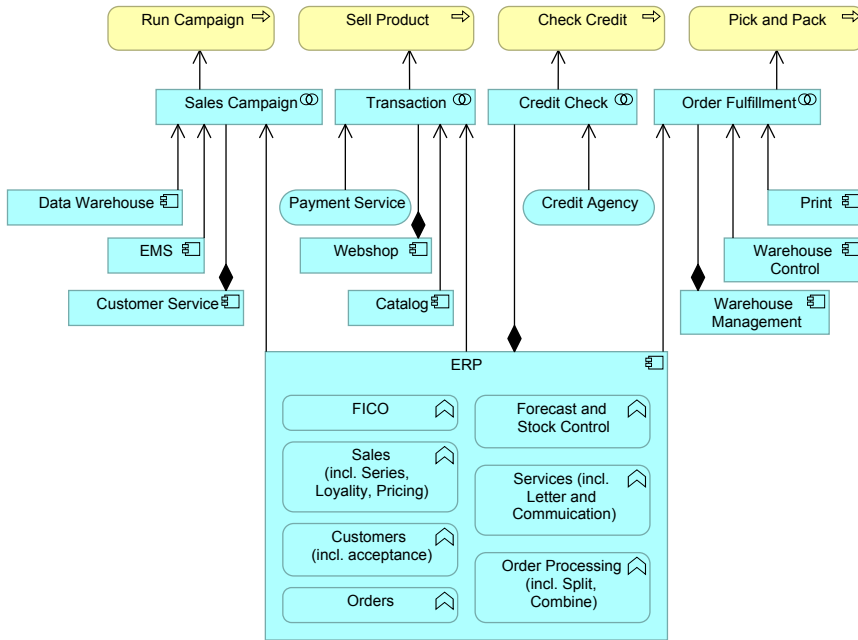


Figure 2.7: Processes and application systems

The four core processes of the case study can be unambiguously mapped onto these business functions. The goal of the run campaign process is to increase sales through marketing initiatives which can be special offers or other programs offered to a specific group of existing customers. The process encompasses the identification of the target customer group, selection of the customers, and the communication with the customer through various online channels. These pre-sales, front office activities are part of the ERA's marketing and branding business function. The sell product process allows the customer to place an order providing delivery and payment information, ending with the order confirmation. In terms of the ERA, it is a front office sales activity and belongs to the pricing and selling business function. The purpose of the credit check is to approve the payment method for the transaction. Based on the past payment behavior of the customer and the information provided by an external credit agency certain payment methods such as purchase on account will be granted or not. The process can be considered as a back office activity and is part of the collection and receivables business function. Finally, the pick and pack process is part of the order fulfillment business function, which is one of the four ERA's supply chain activities. This process encompasses the split and combine logic for orders with more than one product, box size determination, picking of products, and printing of invoices.

2.5.3 Application services

Each of the described business processes is supported by a collaboration of application components. Within the collaboration one main component manages the overall process execution while the other components deliver additional logic or data. For example, the sales transaction is handled by the online shop while the external payment service and the product catalog are providing additional payment functionality and product information. The provisioning of higher level services by composition of various services is reflected on the application service layer of the ERA. However, most of the literature proposes distinct, generic business process management services responsible for orchestrating the various business services. This approach can be beneficial to support the centralized management of the business process lifecycle including design, development, execution monitoring, and refinement, but has not been globally adopted in practice.

The application services covered by the ERA include front office, back office, and supply chain services. However, on a more detailed level, only particular services have been covered in the literature. While for example the back office services for negotiation, contracting, sourcing and purchase are covered, sales and accounting related back office services from the case study have not been found. The same applies for the supply chain services where the warehousing service related topics are not being discussed in the reviewed e-commerce literature. One can argue that those topics are not specific to e-commerce as warehouse management is a generic function. However, split and combine logic, box size calculation, and parcel registration require specialized functionality for e-commerce compared to other warehousing practices.

2.5.4 Application components

The ERA contains eight main components, providing the various application services. However, in the case study, nine components are used. In Table 2.4 we map the application components of the case study to those included in the ERA.

The most dominant application component in the case study is the enterprise resource planning (ERP) system which covers financial, procurement and sales functionality. It also is the de facto master data management system. While the e-commerce literature differentiates between these components, in practice, most functionality is bundled into one system and provided as modules. However, other components mentioned in the literature are more complex and are handled by a number of distinct systems. For example, the logistic component consists of a warehouse management component, on the one hand, and the warehouse control system, on the other hand. In the same way, the sales component consists of an online shop, a separate product catalog, and the sales module of the ERP system.

Table 2.4: Application components of the case study mapped to the ERA

Case study application components	ERA application component
Online shop	Sales
Catalog	Sales
ERP	Sales
ERP	Finance
ERP	Procurement
ERP	Master Data Management
Warehouse Control	Logistic
Warehouse Management	Logistic
Email Marketing Software (EMS)	Generic Components
Data Warehouse	Generic Components
Print	Generic Components
—	Production
Customer Service	Customer Service

2.5.5 Technology services and components

The ERA provides four different technology layer services. The application services and data services are consumed by the various business applications of the case study. As mentioned earlier the organization has not implemented a dedicated master data management system. Thus, there is no application layer counterpart to the technical data services. The same applies to the business process services where the company uses the built-in functionality of the applications to support the process lifecycle. Accordingly, the underlying technology components are less differentiated compared to the ERA. The applications rely on various application servers and database nodes. One exception is the customer service application, which is entirely cloud-based, and thus, has no footprint on the technical layer, a scenario which is also reflected in the ERA. However, the company operates advanced printing facilities, which are required for producing invoices, flyers, and leaflets, that are included in the shipments. This component has not been found in the e-commerce literature and is not part of the ERA.

2.6 Conclusions

In this chapter, we have shown how traditional, process-oriented e-commerce models can be extended to get insight into the application and technology layer services and components. We have studied relevant literature since 2003 and came up with an enterprise reference architecture, which is specified in the ArchiMate language. The reference architecture can be used for several purposes:

- It can be implemented in various ways. In particular, the implementation of a service-oriented e-commerce platform as we will present in the upcoming chapters.
- It can be used to validate existing solutions with regard to their comprehensiveness.
- The architecture also gives an overview of the state of the art in e-commerce research and points out gaps as mentioned in Section 2.5.

The proposed architecture is drawing exclusively on components exposed in the academic e-commerce literature and uses a well-established framework to structure them. The architecture can, thus, be considered as objective, as it excludes researchers' or any biased parties' assumptions on e-commerce architectures such as platform or software vendors. On the other hand, the architectures proposed in the literature might focus (due to topical trends or expected research value) on certain aspects and leave out some components that should belong to a comprehensive architecture. In the following, we are discussing each architectural layer in this respect.

The business layer of the architecture is based on several existing reference models. The most dominant model in this collection is the Retail-H from Becker and Schutte (2007). The other models are matching all, a subset, or a specialization of the business functions identified by the Retail-H. The only exception is the customer service function, which is only mentioned explicitly by the model by Frank (2004). This is due to the origin of the Retail-H model, in a brick and mortar setting, where individual customer communication is an inherent part of the sales process, which happens face to face. Frank's model, in contrast, focuses specifically on e-commerce and considers the fact that the sales process itself is automated, and individual communication requires a distinct practice, such as a call center.

The application layer of the ERA is based on the literature review and proposes six main business specific application components. Furthermore, master data and generic constructs can be found in the model. As mentioned before, the containing services can only be considered as a placeholder for further master data and generic services. Some of the main application components contain further services, especially sales and procurement applications are discussed in more detail in the e-commerce literature. Other applications, such as customer service or logistic, are not very elaborated. This might be due to their extensive applicability

in broader contexts than just e-commerce settings. Further investigation of these topics in specific literature for customer service and logistics could be helpful to analyze application components in detail. Nevertheless, from an enterprise architecture perspective, a good overview of the application layer is provided in the existing e-commerce literature.

The technical layer of the proposed reference architecture contains the most common middleware nodes, the underlying infrastructure, and the services used by the different application layer components. The implementation of all the services is not described in detail. For example, security and user management are described in various studies, but we learn very little about how these are implemented in an e-commerce context (for example what encryption techniques are applied or how users are managed across applications). The e-commerce literature is not going much into detail hereof, probably the research community assumes that the implementation of those technical services does not differ from those used in other domains. Similarly, to the application layer, going into the details about printing or user management systems could provide further insights.

The comprehensive view of the overall process and data has led to large end-to-end monolithic software solutions in the past. Nowadays companies are struggling to maintain and to replace those legacy applications with new systems or cloud offerings. The shift from modeling application functionality as self-contained services may help companies to build their systems in a way that pluggable components can be exchanged more easily with new offerings based on the business needs and without architectural constraints.

The surveyed literature helped us identify the most important functional and technical building blocks facilitating e-commerce operations. These findings helped us transform existing reference models and identify required services. The evaluation has shown that there is no gap between the systems covered in the case study and the reference model layers. In Chapter 4 we will revisit the presented reference architecture by analyzing existing solutions on the market.

Measuring the Pluggability of Software

As mentioned previously, the provisioning of hardware and software resources in terms of services is becoming increasingly popular. Improved scalability of hardware resources and faster adoption of applications are some of the benefits this IT sourcing model has to offer (Armbrust et al., 2010). Another advantage of outsourcing is the abstraction from technical complexity by shifting the responsibility for implementation and operation of systems from the user to a service provider. In the past, the IT user was the owner of an IT resource, but its role is now shifted to become merely the consumer of services, which lets him outsource all the related IT tasks. Accordingly, the use of SaaS becomes increasingly popular, as it enables the user to consume application services from a third party. Thus mitigating risks. Companies getting into financial difficulties because of exceeding the cost for re-engineering of large IT systems becomes an outdated issue (Scott, 1999).

While in general, we see promising developments in the domain of cloud computing, some issues can be observed that have not received much attention to date. The IT landscape of modern industry and service companies usually consist of dozens of different application systems. Processes span across applications and resources have to be shared or passed between the systems (Puschmann and Alt, 2001). While the service model allows the user to bypass intra-application complexity, the inter-application complexity remains and along with it the need for the labor to develop and maintain the various connecting artifacts. This becomes a critical issue as the task of integrating third party systems is more challenging than integrating on-premises, due to the limited knowledge and access to the internal structure and behavior of cloud solutions.

In this chapter, we elaborate on the capabilities of software services with respect to their pluggability. The outcome is a pluggability quality model and a measurement instrument. The novel concept of pluggability reflects the experience of the end user of a software service with regard to the complexity of its adoption and use. An analogy to a pluggable software service is the plug and play hardware introduced in the nineties. Modern hardware components such as USB devices work in a pluggable way as they allow the user to connect the resource to the existing environment and make its overall operation effortless. Different information systems and architectures may lead to different level of pluggability. Instead of focusing on

one specific architecture, we are going to present a quality model that allows evaluating service-oriented architectures with regard to the pluggability of the services and the conceivable platform. Subsequently, pluggability is utilized as the objective of the design cycles outlined in the following chapters. Thus, this chapter can be considered as the foundation of the pluggable service platform design.

In Section 3.1 we present existing software quality models and quality model development methods. In Section 3.2 the concepts of the new quality model for pluggability is presented, along with its operationalization.

3.1 Quality models

The quality of an IT systems describes the non-functional attributes that should be considered when implementing or adopting a solution (Lankhorst et al., 2012). In this section, we are going to present the common methods to define models for the quality of IT systems as well as popular models in the field. This analysis provides the base for the development of the pluggability model.

3.1.1 Method

Ortega et al. claim that the notion of quality in general and software quality in particular is elusive. The perception of quality depends on the stakeholder's perspective and needs. Therefore, they propose a five-step approach to building dedicated quality models, including the determination of high-level concepts, the identification of tangible properties that match the concepts, as well as the evaluation of the model (Ortega et al., 2003). Folmer (2012) uses a similar approach to develop a quality model in the field of semantic standards, starting with a high-level structure, measurable concepts are derived which are the base for the actual measures. We combined both approaches into a process that meets our requirements to develop the quality concept of pluggability. Figure 3.1 outlines the process.

During the conceptual phase, we analyzed existing quality models in order to identify gaps and outline the novel quality characteristic which reflects the needs of a service user as the stakeholder. The goal of the subsequent step was to operationalize the model by defining measurable concepts based on the goals of the quality characteristic and to derive a number of measures for each concept.

3.1.2 State of the art

Scholars have been developing quality models for software since the 60th. McCall et al. (1977) presented a mapping of software quality factors from 31 studies between 1968 and 1976 which resulted in a final list of 11 quality factors. Those factors represent a high-level view which is further detailed by derivation of criteria that drive the actual quality factor. To date, the most recognized quality model is the ISO/IEC 25010:2011 standard (formerly known as ISO/IEC 9126). It defines a similar set of factors, called quality characteristics, which are specified by

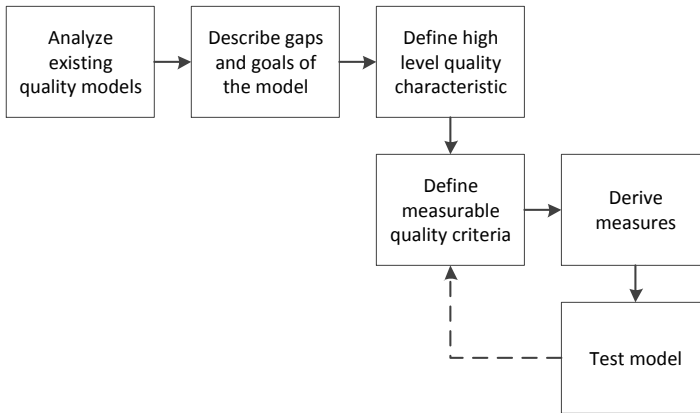


Figure 3.1: Approach followed for the quality model construction

sub-characteristics or attributes. Quality criteria and attributes respectively can be operationalized by defining corresponding measures. Within the described framework, pluggability can be considered as a quality factor or characteristic. Thus, the goal of this chapter is to define the criteria that drive pluggability and to find the appropriate measures for each criterion.

Ortega et al. (2003) has mapped the characteristics of various quality models, including the above mentioned and further popular models by Boehm (1988); Dromey (1996); Grady and Caswell (1987). Figure 3.2 shows the occurrence of various quality characteristics in the models. The most popular characteristics mentioned are *efficiency*, *reliability*, and *maintainability* followed by *portability*, *testability*, and *functionality*.

3.1.3 Perspectives on software quality

The existing models are applicable to software components rather than to a holistic architectural view. Furthermore, they focus on internal properties rather than on the external attributes of system components. Considering the service model described in the introduction and the concomitant shift from software ownership to service subscription, many of the quality attributes are more relevant to the service provider than to the service user. Out of the six most frequently mentioned quality characteristics only *reliability* and *functionality* are quality criteria which are significant for a party consuming the software as-a-service. The other four characteristics are mostly dealing with tasks related to providing and maintaining the service and are critical for the service provider in the first place. On the other hand, the characteristics which seem highly relevant for service users are less prominent among the different quality models. For example, *correctness* is a vital quality criterion for

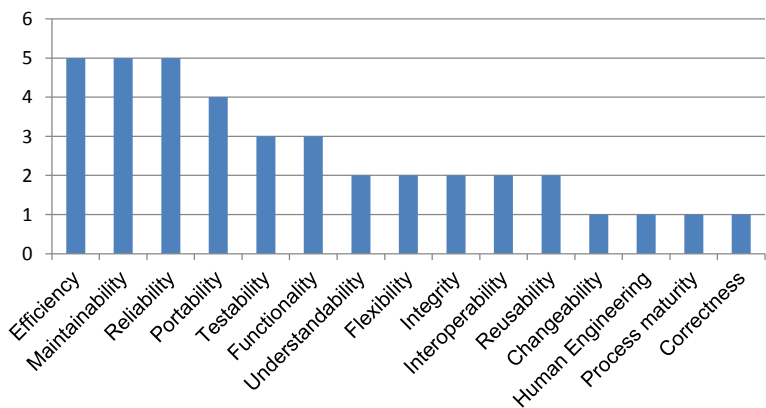


Figure 3.2: Occurrence of characteristics in different quality models (Ortega et al., 2003)

service consumers who generally do not have the ability to easily correct errors of a service implementation. However, *correctness* is only mentioned in one of the quality models (McCall et al., 1977).

We can summarize, that the practice of no longer developing in-house IT systems or deploy and customize pre-packaged solutions on-premises, leads to a changing perception of quality. While some of the common quality criteria become extraneous, new criteria gain in importance. Those criteria reflect the external, non-functional attributes of software services, which companies should assess prior to adopting new systems. It is important to note, that the external view does not claim to replace the existing quality models which are still highly relevant from a service provider standpoint. Instead, they offer a new perspective on software quality from the service user point of view. Furthermore, this perspective should also not be confused with the perception of the end user of the software which is mostly concerned with functionality and does not take into consideration the aspects of provisioning and integration of services (Erbes et al., 2012).

3.1.4 Related concepts

Beside the mentioned software quality models, new models evolve, which are related to the concept of pluggability. O'Brien et al. (2007) proposes nine quality attributes for services in a service-oriented architecture. While some of the criteria are known from the traditional software quality models, new ideas emerge, especially the focus on *security*, *reliability*, *performance*, and *availability*. Those factors go beyond the design and implementation of software artifacts and consider the deployment and hosting of the software component, which are characteristics that more inherent to a cloud product than to an on-premises IT system.

Some more specific quality characteristics evolve that reflect the service model and stress the architectural perspective on software quality. For example, the concept of agility assesses the quality of the overall system instead of the individual software components. According to Lankhorst et al. (2012) the following four criteria drive the agility of a system:

- Ease of making changes to the system
- Ease of rapidly deploying changes
- Ease of minimizing and dealing with effects of changes
- Ease of integrating a system with its environment

The benefit of defining the criteria for a certain quality factor such as agility is twofold. On the one hand, they help to make the concept of agility more tangible which is beneficial in communication and teaching. On the other hand, the tangible criterion ‘minimizing and dealing with effects of changes’ can be directly translated into a measure such as ‘does the servers require restart to reflect changes?’ or ‘does the restart of a single server causes a downtime?’. Thus, the criteria build a bridge between the high-level quality characteristic and its operationalization.

In the research field of smart business networks and inter-organizational collaboration, the concept of quick-connect capability (QCC) has emerged. The criteria for QCC are put forward by Koppius and van de Laak (2009) and encompass the entire lifecycle of inter-organizational communication and transactions including quick connect, quick complexity, quick disconnect, and low switching costs. As QCC refers to the task of adopting and integrating information system services it is related to the concept of pluggability. Koppius and van de Laak (2009) presents a measurement instrument for the QCC of an organization and claims that the establishment of process and communication standards within and across organizations leads to a higher QCC. van Heck and Vervest (2007) claim that besides agility, modularity of systems is an important factor to achieve a ‘plug and play’ collaboration among business network partners and refers to the successful adoption of modular systems in manufacturing. In recent versions of the ISO/IEC 25010:2011 quality standard modularity is considered as a sub-characteristic for maintainability. While the QCC considers high-level technical aspects such as the use of open standards, most of the discussion is about the business practice and challenges within business networks (Konsynski and Tiwana, 2005). In contrast, the notion of pluggability refers to the sourcing of IT services and the enabling architecture while it does not exclude its application for the integration of IT services within smart business networks.

To summarize, *we can define pluggability as a software quality characteristic that encompasses the quality criteria concerning a SaaS user and reflecting the need for technical skills and labor to adopt the software.* In the following section, we elaborate on the quality criteria for pluggability and the operationalization.

3.2 Pluggability of services

As described in the previous section, the development of a new quality model in a top-down manner usually starts with one or more quality characteristics in mind. According to the goal of the model and the identified gaps, the quality characteristic we put forward in this chapter covers the pluggability of a software service. The notion of pluggability describes the quality of a software component itself but also the way it is deployed, managed, and provided to the service consumer. The focus of the quality factor is broader than just the software artifact itself as it reflects the way the service provider facilitates its use. For example, the same open source software component can be adopted by two different service providers and offered as SaaS in a different manner. The decisions taken by the service provider impact the use of the service and lead to a different level of quality perceived by the service consumer. The focus on quality with respect to the implementation *and* provisioning is necessary as the user of a service has less possibility to compensate for adverse decisions in service design.

The pluggability of a service reflects the experienced ease throughout the lifecycle of service use in a specific context including faster and cheaper adoption as well as less required expertise. The context is relevant because the same service can have a different level of pluggability within an existing architectural landscape. As van Heck and Vervest (2007) suggest, the use of a shared platform facilitates the interaction in a business network. The use of a certain platform might, therefore, have a significant impact on the pluggability of individual services.

3.2.1 Lifecycle of service use

A pluggable service allows the user to reduce the effort in every aspect of service use, in the same way, plug and play hardware devices abstract technical complexity and allow the user to choose, deploy, configure, operate, and replace them without insights into their internal behavior and structure. To outline the concept of pluggability we discuss each step of the service adoption lifecycle.

Similar to the various quality models, most of the service lifecycle models focus on the service provider rather than on the service user perspective. Common service lifecycle models proposed by Iqbal et al. (2007) or Alter (2008) mention service design, service implementation, service publishing, and service operation as the main tasks. Sun et al. (2007) reflects the integration of SaaS solutions but leaves out other aspects of the service use lifecycle. In Figure 3.3 we present a model for a service adoption lifecycle that takes into account the phases of service consumption.

3.2.2 Criteria for pluggability

The six criteria for pluggability presented in this chapter are based on the lifecycle of service use. A pluggable service facilitates every phase of the cycle. Thus, reducing the required time and skills during each phase which eventually impact the cost of

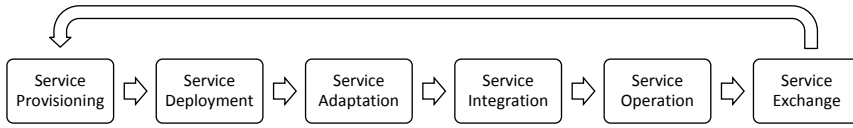


Figure 3.3: Lifecycle of service use

service adoption. In what follows, we describe the criteria for each phase of service adoption.

Ease of service provisioning (EOP): The goal of service provisioning is to discover potential services, compare the various available services, to assess individual services with regard to the business needs, and to enter into a contract with the service provider. The service provider can facilitate these tasks by listing the service in various service marketplaces, disclose all the relevant information publicly, including the terms of use, pricing, service levels, and documentation. Furthermore, the access to demo environments and self-service subscription can further facilitate the assessment and comparison of services.

Ease of service deployment (EOD): Individual services should be easy to install, learn, and test. By default, cloud services do not require technical testing and installation and thus, have an inherent advantage over traditional software components with regard to deployment. In any case, the service should support learning and functional testing through high quality and accessible documentation.

Ease of service adaptation (EOA): The service should be easy to adapt to the functional needs of the consumer. This includes the ability to configure and customize the service. Customizations have a higher level of technical complexity, as additional or deviating logic has to be implemented. Configuration, in contrast, leverages existing logic through setup. A pluggable service maximizes configurability while reducing the need for customizations.

Ease of service integration (EOI): Services should be able to communicate and share data mutually in order fulfill the overall business process. The construction of dedicated interfaces between services is labor intensive and should be supported by the service provider, for example through adapters or service platforms. Services should be able to share and exchange resources without other service quality criteria being affected, especially EOD, EOA, and EOE.

Ease of service operation (EOO): Service operation encompasses the long-term tasks to enable the continuous use of a service, namely maintenance, monitoring and customer support. Service providers can facilitate service operation by providing service level agreements for availability, bug fixing and change requests, as well as a suitable infrastructure such as call centers, a bug tracking systems, and support portals. In order to provide a single point of contact across services, a joint service infrastructure can further improve the EOO.

Ease of service exchange (EOE): Loose coupling is a fundamental principle of service-oriented architectures and requires services to act as independent units of computing. The goal is to facilitate the exchange of individual services. However,

loose coupling of services requires dedicated service orchestration which affects the EOI.

Especially the demand for EOI is somewhat conflicting with the other aspects. Integrating independent services usually requires major efforts which negatively impacts the deployment and adoption efforts. Achieving a high level of pluggability requires a balanced focus on all criteria. Figure 3.4 shows the final model of pluggability, including the main quality characteristic and the six quality criteria.

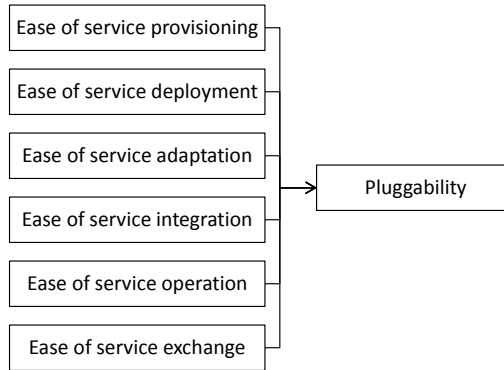


Figure 3.4: Model of the pluggability quality characteristic and its criteria

3.2.3 Operationalization

After the conceptualization presented in the previous section, we carried out the operationalization of the quality model. The goal was to come up with a measurement instrument that helps to evaluate the pluggability of a service. According to Bhattacharjee (2012) the operationalization of a construct includes the definition of indicators as well as attributes which represent the different values. The indicators are the empirical counterpart of the six theoretical constructs (quality criteria) and together form the variable for the pluggability quality characteristic. In the following, we present the reflective indicators which match the quality criteria.

Ease of provisioning What is the level of formal and organizational boundaries to gain insights into the functioning of the service and to assess its usefulness?

Ease of deployment What is the level of required resources to make a service operable including installation, testing, and management?

Ease of adaptation What is the level of required expertise to adapt a service to the functional needs of the consumer?

Ease of integration What is the level of required resources to integrate the service into the existing landscape?

Ease of operation What is the level of required resources to guarantee the target service level including monitoring, maintenance, and customer service?

Ease of exchange What is the level of required resources to exchange the service?

During the construction of the prototype presented in Chapter 4, various types of IT services and different alternatives for each service have been considered. During the lifecycle of service adoption, the six criteria of pluggability have been reflected in order to identify suitable attributes for each pluggability criterion. Table C.1 in Appendix C contains the final attributes for each indicator and the represented values of the instrument after its validation. The levels of measurement form an ordinary scale from -2 to 2 for quantifying the results. The median or mode is the central tendency for this type of scale (Bhattacharjee, 2012).

3.2.4 Face validity

The goal of validating the instrument is to ensure that the indicators and attributes reflect the theoretical constructs. In this work, we aim for validation of the design on the qualitative rather than on the statistical plane. Face validity is a theoretical assessment of the instrument which assesses if the indicator is a reasonable measure for the construct and often carried out by experts in the field. In order to achieve face validity, a group of experts assesses the instrument with regard to conceptual and semantic aspects.

A preliminary version of the instrument as well as the prototype outlined in Chapter 4 were presented consecutively to two scholars and two practitioners in the field of service-oriented architectures. The participants were asked to assess the prototype using the instrument and to explain their reasoning. Afterwards, they were asked to provide a feedback on the instrument with regard to the constructs of pluggability. The feedback of the interviewees on the prototype and the instrument was recorded. Semantic ambiguities in the instrument were resolved based on the input before the subsequent interview. The final result after four iterations is the pluggability instrument that can be found in Appendix C.

3.3 Conclusions

In this chapter, we introduced the novel quality characteristic of pluggability. The model complements existing quality models, which have a strong focus on the internal quality of software artifacts and the service provider as a stakeholder. The instrument may help companies adopting cloud services to assess the quality of the services within their landscape. For the service provider, the traditional quality models are still highly relevant. However, the presented model can also help service providers to reflect the service user viewpoint, eventually leading to improved customer satisfaction. By applying the model, shortcomings in the external quality of their services can be identified.

Within this work, the construction of the quality model represents the objective definition in the design cycle. In the upcoming chapters, pluggability is considered as the goal of the platform architecture and the e-commerce services.

A Platform-Based Return Registration Process

In Chapter 2, we identified the functional building blocks of e-commerce architectures by studying the state of the art in the academic literature. To provide retailers with the flexibility to achieve a working e-commerce solution by combining the functionality of various services, a strong service ecosystem is required. Open platforms are considered as a critical factor for establishing a strong service ecosystem (Benlian et al., 2015). Furthermore, to make the modular approach work, the increasing number of services have to stay manageable. Current IT services are mostly based on packaged applications and require significant resources to make them ready for the business needs of the user (O’Leary, 2000). The required efforts during each phase of adopting an IT service has been reflected in Chapter 3. The lack of current e-commerce architectures to support pluggable services is a potential obstacle for more flexible use of services in e-commerce. Thus, in this chapter, we want to gain insights into the capabilities of the state of the art in platform architectures to support pluggable services.

This chapter performs a synthesis of the ERA presented in Chapter 2 and the pluggability model in Chapter 3. Figure 4.1 illustrates the integration of the previous research results into the initial design science cycle. First, we carried out a review of available platforms in the market. A structured list of functionalities that are supported by current platform solutions is put forward. We incorporate the list into an architectural platform model and link it to the ERA. The resulting platform reference architecture summarizes the literature review and the market research on service platforms. We consider this architectural model as the first design artifact that goes into the cycle of demonstration and evaluation. We discuss the first design cycle in this chapter and conclude with a platform design that hypothetically enhances the pluggability of the state-of-the-art model. The design forms the basis for the next cycles of demonstration and evaluation in Chapter 5 and Chapter 6.

The remainder of this chapter is structured as follows. In Section 4.1 we review current service platforms and identify common features. In Section 4.2 we summarize the findings and come up with an architectural platform model that relates them to the ERA. In Section 4.3 we present the business case and a prototype for a platform-based return material authorization process. In Section 4.4 we present the evaluation of the prototype with regard to the pluggability criteria.

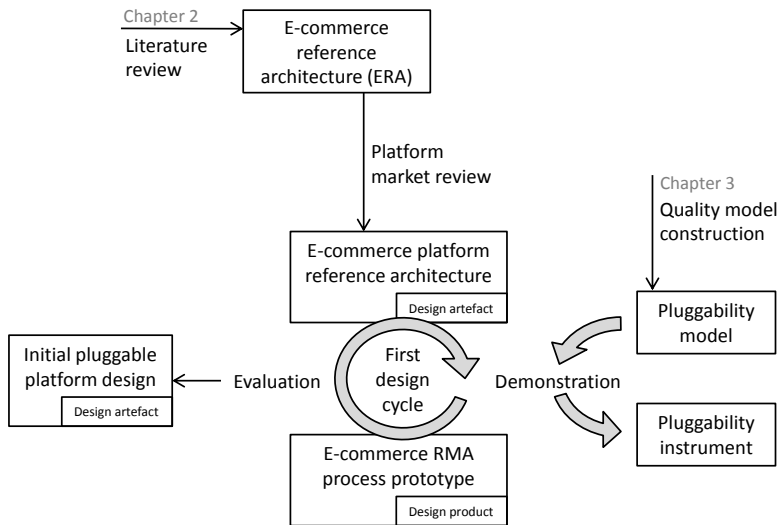


Figure 4.1: Synthesis of ERA and pluggability model in this chapter

4.1 Platform architectures

The goal of the platform architecture proposed in this chapter is to complement the ERA in Chapter 2 in order to reflect recent developments in service-oriented architectures. By taking the state of the art in service platforms as a starting point we were able to assess the current practice and point out shortcomings. In the following, we discuss the objectives and benefits of the service-oriented approach and outline the state of the art in service platforms.

4.1.1 Objectives and benefits

Existing reference models in the domain of e-commerce and retailing are generally business layer models (Frank, 2004; Becker and Schutte, 2007). They describe the entirety of functions and processes of the business model. Systems that are built based on these reference models tend to encompass all the primary business activities, often resulting in monolithic solutions. As mentioned earlier companies increasingly focus on single activities within the value chain. Providing end-to-end systems in the highly disaggregated business environment, with individual organizations that only cover parts of the value chain, leads to inefficient use of IT resources. Furthermore, monolithic systems are not built for collaboration with external systems, making the exchange of individual IT functionality and external business partners cumbersome.

The main objective of a service-oriented, modular architecture is therefore to support the construction of systems that go beyond what current, monolithic sys-

tems achieve with regard to flexibility in IT and business service adoption. It enables companies to integrate innovative IT services faster and helps them to connect with business partners with fewer efforts. The so-called quick connect capability (QCC) has been proposed by a number of authors (van Heck and Vervest, 2007; Koppius and van de Laak, 2009) and describes the capability of network partners to set up business collaboration with fewer efforts in less time. The authors claim that the decomposition of the system and modularity are required to achieve versatility. As van Heck and Vervest (2007) suggest, digital platforms improve interoperability and the QCC. The platform approach differentiates between the stable component in a system and the remaining components which evolve around it (Baldwin and Woodard, 2009). A major goal in constructing the platform architecture can, therefore, be achieved by identifying the stable components of the system to maximize pluggability of the remaining components.

The expected benefits of the increased pluggability through the service platform can be summarized as the ability of the platform user to source external and innovative IT services as well as to collaborate with external business partners more easily.

4.1.2 State of the art

To analyze the state of the art in service platforms we started by investigating the available e-commerce specific products and cloud services in the market. Based on our findings we extended the market research by looking at cloud integration platforms, which complement the functional components with the aspects of connectivity. Even vendors which are in favor of a lightweight, pluggable e-commerce service platform differentiate between core business functionality and connectivity components. Spreecommerce, one of the major cloud service providers for e-commerce, for example, differentiates between their core storefront product and their dedicated integration hub called wombat.

Similar to other enterprise application systems the e-commerce platform solution landscape has evolved from custom-made components to pre-packaged solutions. Pre-packaged e-commerce solutions provide the e-commerce specific functionality such as shopping cart, product catalog management, marketing tools, and payment (Humeau and Jung, 2013). By implementing a pre-packaged solution, it becomes easier for business owners to set up and launch their online store, resulting in a faster time to market. Despite the ease and functionality that e-commerce solutions offer, some challenges remain, that need to be solved by specialized pieces of functionality.

In Chapter 2, a systematic literature review was carried out, investigating on processes and architectures for online retailing. The results were validated with the online retail practice. According to the findings, the online retail process is comparable to existing reference processes in retailing. Furthermore, the IT landscape of online retailers is characterized by five major components, namely procurement, sales, service, logistic, and finance. In practice, most of these components are not

covered by the e-commerce system. As a result, additional components such as an ERP system and a warehouse management system (WMS) are in use.

We conclude that the state-of-the-art e-commerce platform is modular to some extent. However, the granularity of the services is limited to a small number large application systems that cover an extensive set of functionality. Furthermore, the interoperability of the functional components relies on middleware components which are supposed to increase the pluggability.

The most widely adopted solution by the e-commerce platforms under study for solving the enterprise integration issue is to rely on hard-wired web service integration. In this approach, each external service is connected to each online shop platform through the so-called “connectors” or “adapters”. If a connector is not available, some platforms also provide toolkits for users to develop their own application connectors. While this approach seems to work just fine, it will produce an inefficient point-to-point integration topology in the end. When the number of systems to integrate increases, the entire integration scheme will become highly complex, with a negative impact on scalability.

Besides, in near future, it is expected that cloud computing will gain more popularity with companies and organizations migrating their existing local systems to the cloud. Because of this situation, new integration scenarios emerge that involve both on-premises and cloud applications. It might become cumbersome to integrate systems of different nature like SaaS systems and legacy systems. Connections between SaaS applications are also challenging due to the diversity of data models and lack of standardization (Potočník and Juric, 2012). The increasing adoption of cloud computing brings novel ways to solve integration challenges. Traditional middleware and integration platforms could benefit from cloud computing technology by leveraging themselves as cloud integration platform (Kleeberg et al., 2014), commonly referred to as ‘Integration Platform as a Service’ (iPaaS), a term coined by Gartner. Their recent study evaluated and compared iPaaS providers (Pezzini et al., 2014). Another research by Ried (2014) assesses 14 vendors providing hybrid integration solutions, which in their description, comprise of four integration scenarios: on-premises integration, cloud-based integration, iPaaS, and API Management.

4.1.3 Common platform services

According to the previously mentioned objectives, we extracted the common features that iPaaS vendors typically offer in their platform. In the recent times, vendors started to incorporate API Management capabilities into their platform in addition to SOA Governance to deliver a complete solution to take care of both REST and web services. Both SOA Governance and API Management functionality can be considered as essential to enable pluggability of services. SOA Governance and API Management share the same underlying architectural design principle, which is service-oriented design. Both aim at governing and managing the service lifecycle including design, implementation, publication, operation, maintenance, and

retirement of services and APIs (Malinverno et al., 2013). SOA governance technologies, however, have been around for several years and almost reached maturity. SOA governance covers a wide range of functions including but not limited to policy enforcement, security, service contract, compliance, service level agreement (SLA), lifecycle management, service registry and repository (Schepers et al., 2008). On the other hand, although API Management comprises similar building blocks as SOA Governance, it incorporates some distinct functionality (Maler and Hammond, 2013). It can be said that the fundamental difference of API and SOA lies in their orientation of service consumption. In general terms, SOA is geared towards service consumption within an organization while APIs, due to their openness, can be used both internally and externally. As a consequence, some additional components, such as enterprise gateway, security, developer portal, and service billing need to be available in API Management. The study outlined in (Paramartha, 2014) was part of the CATeLOG research project and presents a market analysis of cloud integration platforms. It groups the services offered by these platforms in two categories. *Meta-services* on the one hand facilitate the access and use of provided services. *Process services* on the other hand offer additional features to enable process execution across the integrated services. The service framework meta-services are presented in Table 4.1 and process framework services are presented in Table 4.2.

4.2 A reference architecture for e-commerce service platforms

As mentioned in the previous sections, a popular way to source functionality in modern enterprise architectures are outsourced cloud applications, offered by third-party service providers. Retailers that want to add or replace such services have to be able to integrate them into their current system landscape. The idea behind the service platform architectures is to give users the possibility to integrate e-commerce services into the existing environment with a minimal amount of resources in terms of time and expertise. The platform should allow supply-chain partners to share their services, execute inter-organizational processes and work on resources collaboratively, eventually resulting in an open and agile e-commerce business network. Such inter-organizational integration platforms have some distinct requirements compared to systems internally deployed and used within one organization or only available to a closed business consortium (van Hillegersberg et al., 2012), especially if the platform aims to act as a one-stop shop to source IT services. In this section, we are going to present a reference architecture for a service platform, which incorporates the findings on state of the art in e-commerce and cloud integration platforms from the previous section. This initial state of the art service platform acts as a base for gradual enhancement of the service pluggability in the subsequent design cycles.

Table 4.1: Service framework meta-services

Service framework metaservice	Features
Developer portal	In the developer portal, companies should provide relevant and comprehensive aspects of their APIs such as API documentation, policy, terms and agreement, testing environment (sandbox or real), or API versioning.
Enterprise gateway	Management of the interaction between the API and external API consumers.
Policy enforcement and management	Management of both, design time and runtime policies of services. Design time policies are concerned with aspects such as design guidelines or security mechanism while run-time policies are concerned with operational environment and requirements that must be met by the service at runtime.
Security	The difference between security in SOA Governance and API Management is that in SOA Governance, the organization administers internal and known users while API Management handles external and unknown users. API Security manages additional aspects like authorization and authentication, API Key management, as well as Identity and Credential Management.
Service analytic and reporting	Exploration of insightful traffic analytics and reports of API activities with respect to developers account, application, or services as well as observation of the overall API usage and trends.
Service level agreement	Management of service levels as stated in SLA contract, service evaluation as well as fees for consuming the service and fines in case of contract violation.
Service lifecycle management	Managing the design, development, and delivery of individual services in a SOA. The tasks include change management procedure, service registration and even deciding on service granularity.
Service metering and billing	Monitor and measure service usage as the basis for billing and calculation for the service consumers. Also, the service performance can be monitored regularly.
Service registry and repository	The catalog of services and management of their publication. Definition of taxonomies of the published services allowing consumers to find suitable services to their needs. While the Service Registry only contains service references, the Service Repository is the actual holder of documentation, policies and, metadata about the versioning of the service.

Table 4.2: Process framework services

Process framework service	Features
Development and lifecycle management platform services	Manages service integration process flows throughout their lifecycle including modeling, development, configuration, testing, and deployment.
Integration platform services	Consists of aspects that ensure seamless integration flow both at design time (service orchestration) and runtime (process execution). These aspects include but are not limited to: Message transformation and routing, an Integrated Development Environment (IDE), adapters, flow management, protocol conversion, service virtualization, and security federation.
Monitoring, management, and administration platform services	Takes care of deployment and administration of integration flows, monitor their execution and manage their behavior. Covers several aspects such as technical and business activity monitoring, logging and tracking, as well error resolution.

4.2.1 Framework

According to Baldwin and Woodard (2009) a platform can be considered as “a set of stable components that supports variety and evolvability in a system by constraining the linkages among the other components”. In our case, the components are e-commerce services that, in combination with the platform, compose a working information system for e-commerce businesses. The goal of such a platform is to improve the pluggability of the services to support variety and evolvability of the used services and the overall system.

The service platform has three stakeholders, namely the service provider, the platform provider, and the service consumer, which is the company running an e-commerce business. To illustrate our architecture, the ArchiMate modeling language has been used (Lankhorst et al., 2009). It provides concepts on business, application and technology layer to model enterprise architectures (cf. Appendix A). As we are dealing with an inter-organizational architecture we choose a viewpoint with the three business actors as high-level concepts. All further concepts are assigned to either of these actors. We focus on the application layer components that support the e-commerce process at hand. Figure 4.2 illustrates the view of the architecture, including a metamodel of the relevant concepts. The details of each business actor are discussed in the following.

4.2.2 E-commerce company

The e-commerce company is the actor selling goods partially or exclusively over the online channel. On the business layer, the online retail process consists of pre-trade, trade, and post-trade activities (Liu and Hwang, 2004). Internally the actor implements eight different business functions which have been identified by different authors in (Gunasekaran et al., 2002; Burt and Sparks, 2003; Becker and Schutte,

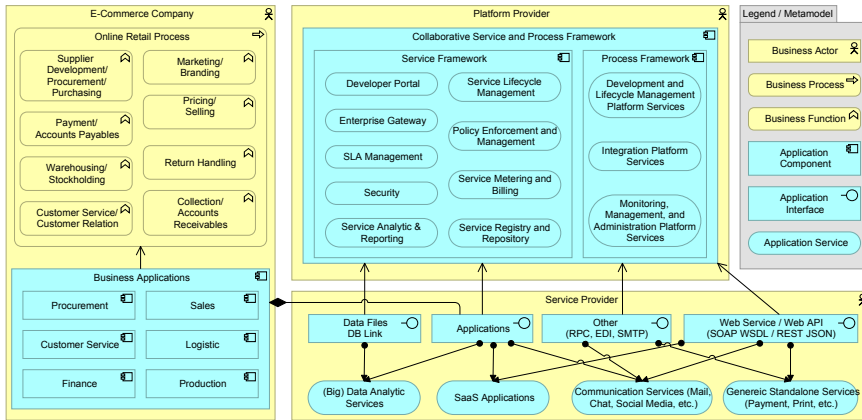


Figure 4.2: ArchiMate model of a common service platform

2007; Frank and Lange, 2007) and consolidated Chapter 2. In that chapter, we also presented the six application layer components implemented by most online retailers. Depending on the details of the company's background, different legacy components will be implemented on the application layer. A retailer coming from an offline channel business, with a number of brick and mortar stores, will likely have an ERP system to manage its operations. In that case, a variety of components will be bundled into the ERP system. When introducing an online channel, the retailer will add an online shop component to the landscape which allows customers to browse and order goods online. The order fulfillment and other back office activities will be carried out by the ERP system. Thus, in this case, the e-commerce platform consists of a lightweight online shop and the ERP system. A pure online retailer, on the other hand, might implement a more comprehensive e-commerce platform as covered in Section 4.1.2. These platforms not only provide an online shop but also a rich set of back office functionality. Depending on the complexity and size of the business, an ERP component might not be present at all. All these application components can be either operated on-premises or as SaaS solutions provided in form of web applications.

The presented architecture for the retailer business actor can be considered as the current state of the art and does not introduce any new concepts. In that sense, it is a starting point for the use of a pluggable service platform. The architecture should allow for a gradual transition from the current, monolithic landscape to a cloud service architecture. It should be possible to add services to that landscape and successively shift new and existing functionality from internal systems to the cloud.

4.2.3 Service provider

The service provider can either offer pure IT services or be a supply chain partner that allocates business services. Both service types must be integrated on the information system level for seamless process execution. The actual service provided can contain either additional components that internal systems or business functions do not cover or functions that should be outsourced for strategic reasons. The actual services, as well as their granularity, are too diverse to provide a comprehensive list of potential services. Essentially, it should be possible to integrate any kind of services through the architecture. However, a more important aspect is to obtain a comprehensive picture of potential service interfaces the platform needs to support. Four different services interfaces have been identified.

- Message-based integration can be realized through modern web services or web APIs that communicate over HTTP and can be consumed with state-of-the-art integration tools and techniques. This kind of interface is suitable for standalone services such as payment services, address verifications, customer or credit inquiries but also to access or populate resources of SaaS applications and social media services in a programmable manner.
- Another interface type is based on more specialized protocols that can be considered as an older technique to integrate services. Despite their higher complexity and technical dependencies, these protocols are still widely used to integrate legacy systems or communication services such as mail and chat but will not be implemented by modern SaaS applications.
- Web applications are generally used as user interfaces in SaaS or social media services as well as in analytical services and reporting in form of dashboards.
- On the backend analytical services will be integrated using an interface type that allows the exchange of large amounts of data. Message-based integration would produce too much overhead and is therefore not suitable for such scenarios which involve large to big data sets. Instead, the integration will be based on data extraction, transfer, and loading (ETL) or through database links.

4.2.4 Platform provider

The service platform acts as an intermediary between the retailer and the service provider. The goal of the platform is twofold: it should allow retailers to source IT services and to collaborate with supply chain partners. It provides a service framework that allows provisioning and consumption of services and a process framework to implement service-oriented process flows. Both the meta-services of the service framework and the services of the process framework are based on the findings in Section 4.1.3.

4.3 The return registration case

Based on the proposed reference architecture in the previous section, we demonstrate the implementation of a service-oriented process through realization of a prototype for a specific e-commerce case. The goal of the prototypical implementation was to assess the state of the art in e-commerce services and integration platforms. The prototype development gave insights into the feasibility of a process, based on a loose set of e-commerce related services and integration platforms. In the subsequent phase, the level of pluggability of the services in the resulting system was determined.

4.3.1 Business case and solution design

As the existing reference models for retail do not cover return handling processes (cf. Chapter 2), it can be assumed that retail systems based on such models are not designed to handle return shipments efficiently. This might cause problems for multi-channel retailers that are facing high volumes of returns, especially in the fashion sector (Banjo, 2013). In the following, we first describe a business case which covers a part of the overall return handling process. We then discuss the services used to implement the process and present the architecture of the solution.

In the scenario, an end customer should be able to register a return online. Through a web page referenced by the online shop, the user can select his order and retrieve information on the items contained in that order. The customer chooses the items and the amount he would like to return, specifies the reason for the return, and optionally adds a comment. The return request is transferred to the retailer who authorizes the return of the material (RMA). Subsequently, the return is planned by registering the expected goods and assigned to the appropriate return center. Finally, the shipper is contacted by e-mail to inform him about the approval, providing the link to a return label for print and possible drop off points based on the customer address. With that information, the end customer can prepare the goods for shipment and transfer it to the drop-off point.

Four application components are required in the scenario, each provided by individual service providers. They include 1) a SaaS solution, that allows customers to register their return shipments (such as provided by 12return.nl) 2) a generic standalone web service from a logistic service provider (LSP) that allows to register and pay shipments and to obtain the required documents (such as intraship.de) 3) a workflow task list that allows back office staff to approve and reject requests and 4) an e-mail service provider (ESP) that delivers high volume customer communication services (such as tripolis.com). A description of the services is provided in Table 4.3.

Figure 4.3 shows the overall architecture of the solution. The service platform executes the collaborative flows that make use of the various services to provide the business functions with the required functionality. The model shows that each service relies on the collaborative data resources required to fulfill the service. The

Table 4.3: Services used in the business case

Service	Description
Return registration	A web application that handles return shipments. It allows end customers to request a return of goods through the web interface. The user has the possibility to look up recent orders and select individual goods for return
Parcel registration	The parcel delivery registration service allows to register parcel shipments, print parcel badges and schedule parcel pick up. In the case study scenario, it is used to issue parcel label to the end customer
Task list	A task list application assigns a list of tasks to each user which they have to act upon. This can be approvals, responses or other action items. Such task lists are often integrated into workflow systems and have advanced features such as task forwarding, task escalation or holiday calendar integration. In this case, the task list is used to assign approval notifications for the requested returns
E-mail transmission	The e-mail service is used to send outgoing e-mails to the customer including the approval and parcel label for the return shipment

return registration SaaS solution requires information about the orders made by the customer in the past. The same applies for the LSP and the ESP that require information on the customer. We can observe that having these resources available as part of the platform would allow the allocation and exchange of services to the overall process in a more flexible way.

4.3.2 Prototype

The services in our example are implemented using diverse technologies, are distributed among different environments, and use various protocols to communicate with other systems. In the following, we provide an overview over the different services and platforms that have been used for prototype construction.

For parcel delivery registration, many carriers offer web services to register shipments and generate parcel labels. Among these, DHL is among the leaders regarding easy adoption of their web services, offering a developer portal and well-documented services. However, the interface to register parcels seems very large with 225 required data fields and another 173 optional data fields, exposing a lot of internals of the system which the user has to comprehend before getting the service to work. Other services such as shipcloud.io or postmaster.io evolve and facilitate the integration of various logistic service providers. Their REST APIs only have 16

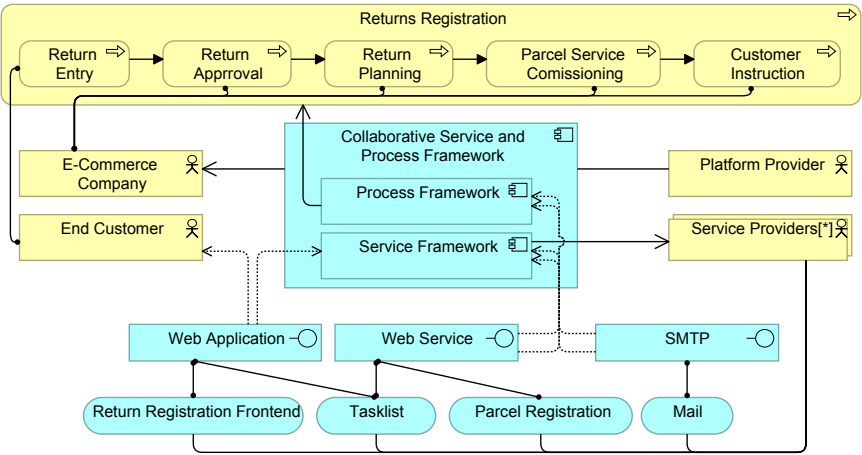


Figure 4.3: Architecture of the returns registration process

data fields to achieve the same shipment label generation. The time to integrate and exchange parcel services could be reduced from 2 days to half an hour by using the interfaces of these broker services. Also, switching between different carriers during runtime by requesting quotes and selecting the cheapest offer is becoming easier as the brokers cover a wide range of parcel services through the same interface.

The web application frontend for return registration is a custom-made lightweight single page application (SPA) realized with common web application technologies, namely HTML, JS and CSS. Figure 4.4 shows the browser fronted of the application. It allows the user to enter his order number and to choose the returns from the contained items. He can further specify the return reason and type of processing. All resources including orders, customers, and metadata are retrieved from the cloud database service running on a remote backend.

As e-mail service, the prototype uses Gmail which has an SMTP interface. In the domain of marketing communication, more specialized B2B e-mail service providers exist, such as mailchimp.com or tripolis.com which offer business-specific services like e-mail templates, analytics, and different integration endpoints. However, these advanced features were not relevant for the business case and we did not find the pluggability changing significantly by using a different interface.

The prototype uses the Questetra BPM Suite for the task list as it is one of the few BPM suites, that provides a cloud environment for the development of the processes and offers an API to integrate with other services. The screenshot in Figure 4.5 shows the task list of a user, listing the assigned tasks that are awaiting action, including the pending returns, waiting for approval.

The key component of the solution is the integration platform containing a service framework to plug the different services together and a process framework to execute the business processes. According to Pezzini et al. (2014), three of these

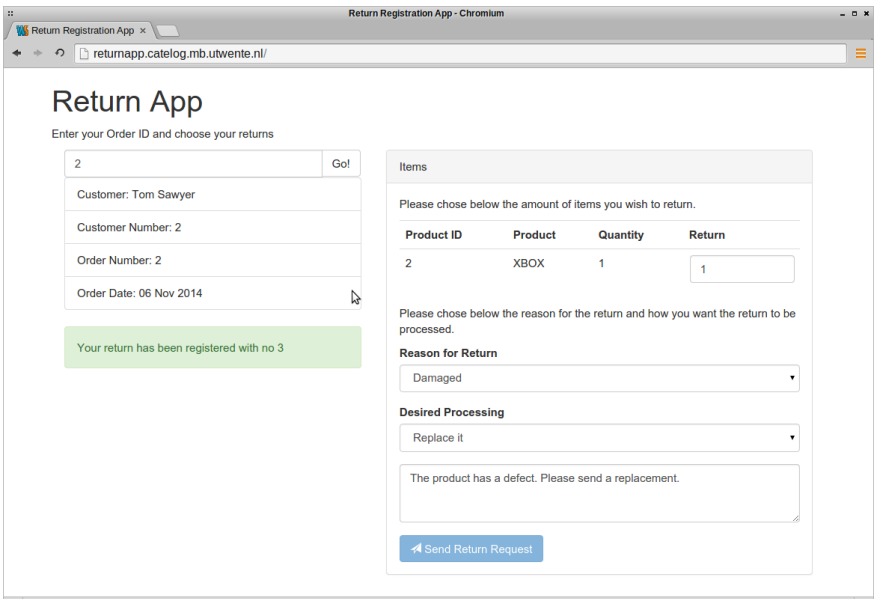


Figure 4.4: Return registration app frontend

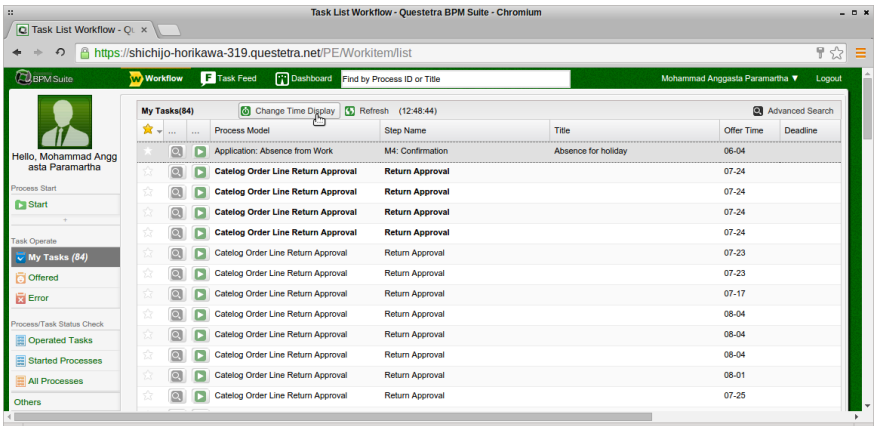


Figure 4.5: Task list application with pending approvals

services stand out in the market with regard to their completeness. For the prototype, we choose the Mulesoft CloudHub platform as it is was the most accessible in terms of documentation and subscription, which is one criterion for pluggability (cf. ‘ease of service provisioning’ in Chapter 3). Figure 4.6 shows how messages are routed and transformed between endpoints using the example of parcel label generation response and outgoing e-mail.

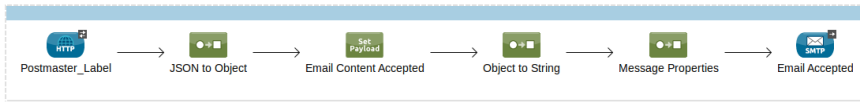


Figure 4.6: Service composition using the example of the integration of parcel and e-mail services

During prototype development, we had to introduce another component that contains data about order, customer and product information in a cloud database. These resources must be available throughout the different services used in the process. In a real world scenario, this information will be scattered across order management, customer relationship, and product catalogs or, in case of a more domain data-driven architecture, be stored in appropriate master data management (MDM) systems. The cloud integration platform could access the database through the built-in adapters. However, we decided to introduce a layer of business logic, built into the component, which exposes the data through a REST interface. For the database and business logic we choose the cloud application platform heroku and the lightweight web application framework flask, however, the same could be achieved with any other platform and web framework on the market. Suitable, e-commerce MDM cloud services were not found.

4.4 Validation

The goal of the prototyped process is to test the feasibility of implementing solutions, based on a set of distinct IT services, using the reference architecture and state-of-the-art cloud integration platforms. Furthermore, the main purpose of the prototype is to evaluate the pluggability of the resulting solution. In this section, we are going to validate the practicability and utility of the architecture based on the prototypic implementation as well as the support of available cloud integration platforms towards the goal of a pluggable system.

4.4.1 Observations

In order to measure the pluggability of the five services, the instrument was applied individually by the developer of the prototype, two scholars, and two practitioners from a Dutch iPaaS provider. The prototype was presented to the participants as well as evaluated against the predefined levels for pluggability. The graph in Figure 4.7 shows the mode score for each pluggability criterion for each of the five services.

The service with the lowest overall pluggability is the return registration frontend. It is a custom-built service which is hosted on-premises. The cloud database is a custom-built solution but deployed to a PaaS environment, which seems to be beneficial for the overall pluggability. In the mid-level of the overall pluggability is the task list solution, which is a PaaS that follows a model driven approach to

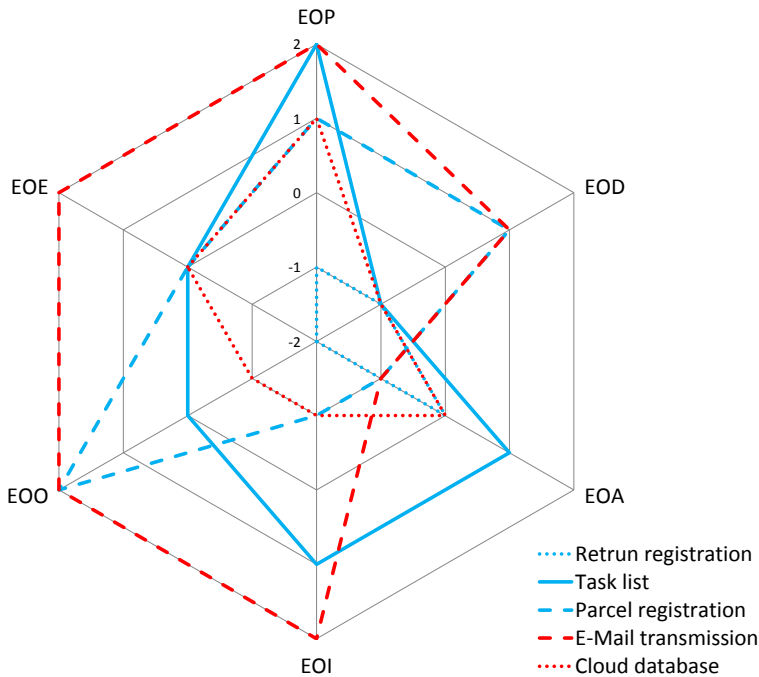


Figure 4.7: Score per pluggability criterion

process implementation. Its non-coding approach seems to be beneficial for the overall pluggability. Finally, the e-mail transmission and parcel registration can be considered as SaaS solutions and exhibit the highest level of pluggability.

The EOD is high for the parcel delivery service and the e-mail service as these services are installed and maintained entirely by the service providers. The EOD is low in contrast for the return registration frontend and the cloud database as the deployment and management are carried out by the retailer. The workflow solution is operated by the service provider but the actual processes running on the platform have to be developed, tested, and monitored by the service user.

For the workflow task list application, the EOA is high, because the used BPM service provides a model-driven approach to implementing business logic. The user has a very high flexibility to adopt the service to his needs without having to customize the system. The return registration frontend and the cloud database are adaptable but need technical expertise to implement customizations. The parcel registration service also has a low level for EOA because it delivers specialized services which are not adaptable. The e-mail service is flexible to the extent that the content of the e-mail is concerned.

All four services of the prototype which are based on cloud platforms have a high EOP. This indicates that cloud service providers, in general, are doing a good

job in documenting and providing potential users with resources for assessing their services. The low EOP for the custom developed return registration service, however, is difficult to explain. We think that the concept of provisioning might not be applicable to custom developments as the task of a fit gap analysis does not have to be carried out in that case. The provisioning is then happening on the technical level which might explain the low EOP.

The EOO is low for the return registration frontend because it requires monitoring of the application and the underlying infrastructure as well as support and maintenance of the application. The use of PaaS for the cloud database and workflow increases the EOO as basic infrastructure is handled by the service provider. However, application maintenance and support is still the responsibility of the service user. The other two services have a high EOO as the entire service operation is outsourced.

The e-mail service has a high EOE as the protocol used by the e-mail service provider is highly standardized and used by any other service provider. Therefore, the migration to another service has minimal impact. The exchange of the parcel delivery service with another service can be achieved relatively easy. We have carried out this exercise by changing the postmaster.io service with shipcloud.io which is only a matter of changing a couple of fields and the service endpoint. Exchanging the cloud database or the workflow system impacts the entire architecture which leads to a medium EOE. The return registration frontend can be exchanged without impact on any other service in the architecture. However, it is relying on other services such as the cloud database to retrieve order and customer information. Introducing another service for this purpose requires it to be adapted to the interface of these services or the service bus respectively.

The EOI is the criterion with the lowest score across all services. The return registration service, the parcel registration, as well as the cloud database have a very low EOI due to the complexity of their interfaces. Each of the services requires thorough design, build, and maintenance of custom interfaces to connect to the other services. While the task list and e-mail service require the same, their interfaces are considered more simplistic so that the EOI is higher.

4.4.2 Improvements on the state of the art cloud in integration platforms

In the previous section, we investigated to what extent the services and the architecture of the prototype adhere to the criteria of pluggability. We discovered, that the ease of integration is the criterion current services and integration platforms lack the most. We see the reason for this shortcoming in the complexity of the task of developing and maintaining the interfaces between the various software services.

Service implementers can easily gain a good understanding of the reference processes, data models, and use cases required to implement a certain application component. It allows them to deliver services that can be used in a wide variety of organizations. However, service providers have no insights into the environments

in which the services eventually operate, which may be the explanation of the lack in delivering the appropriate integration artifacts. Furthermore, these system landscapes vary across the different potential service users which poses another obstacle to delivering such artifacts.

To address this issue, we propose to adjust the architecture of the current integration platforms to facilitate the task of delivering pre-integrated services. As with cloud software services that release the user from struggling with the underlying technology, a suitable integration platform allows the users to reduce their workload in service integration from customization to configuration. Figure 4.8 shows the different milestones of an on-premises solution compared to a SaaS offering. The deployment of the artifact happens prior to the subscription of the service by the user. Thus, the deployment becomes part of the implementation carried out by the provider. As we have shown in this chapter the tedious task of integration happens after the subscription to a common SaaS offering. Pre-integrated SaaS offerings consider the integration as a duty of the service provider and release the user of that task as well.

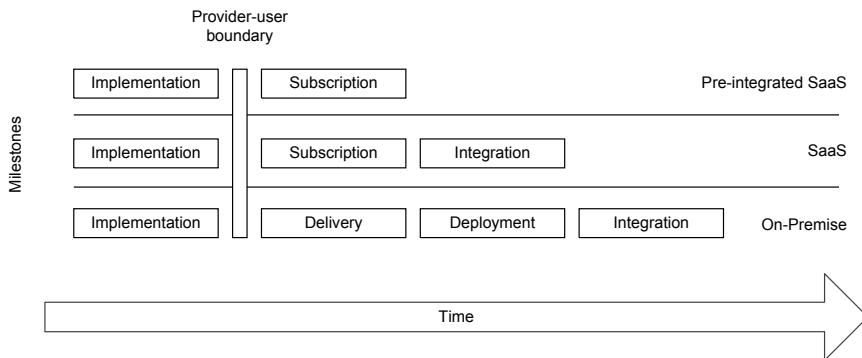


Figure 4.8: Pre-integration of SaaS offerings

We argue, that in an application landscape with evolving functionality the most crucial stable component is the domain data used across the system. This data is the same throughout the different services (integration can be seen as the task of transferring data from one system to another) and different generations of services (migration can be seen as the task of transferring data from one system generation to its successor). However, this aspect is ignored by existing integration platforms.

As a result, to the lack of current platforms and the indicated integration challenges, we claim that the business data should be part of the platform rather than the individual services. The integration platform becomes a domain specific artifact, including a canonical data model (Hohpe and Woolf, 2003) and the required services to help service providers to ship pre-integrated services. Giessmann and Legner (2016) describe application-centric platforms as a distinct type of PaaS. Services

evolve as add-ons to a core application component and can be considered as pre-integrated. Such application-centric platforms that deliver the core e-commerce functionality and allow for pluggable add-ons are not available to date. In Figure 4.9 we show an extended version of the state-of-the-art model that follows a similar approach and adds an e-commerce specific collaborative data management component to the platform. By introducing this concept, we shift the component that is most critical for the integration of systems from the service to the platform. This component handles the canonical data that is used and provides an interface to the resources. In the context of this work, individual e-commerce services evolve around the platform. They can operate by default on the data service and thus, allow their users to avoid the implementation of integration artifacts. Furthermore, it is possible for the processes to access the canonical data through the same interface, which allows for integrating legacy services that are not participating in the use of the data service.

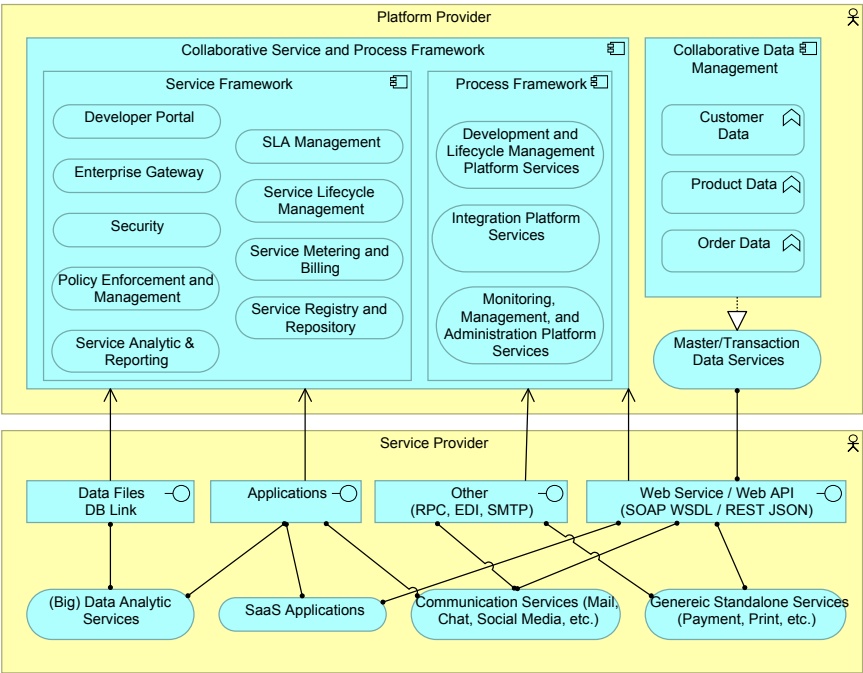


Figure 4.9: Extension of the common service platform model

4.5 Conclusions

In this chapter, we have shown how the IT service industry currently approaches the issue of plugging software services into existing environments. The presented reference model was implemented based on a simplified real-world scenario. Using the pluggability assessment model we found the ease of integration one of the main challenges service users face today. The derived assessment instrument can help to investigate on the pluggability of services in practice and research.

Furthermore, we proposed an extended reference model that can improve the pluggability and is in line with the platform concept. The extension consists of a canonical data management component containing the data that has to be shared among the various services. However, the introduction of such a component will have consequences with regard to the handling of the shared data. At this point, various scenarios are possible how existing and new services deal with the centralized data repository. While new services can interact directly with the data services, the link leads to a strong dependency between service and platform. Furthermore, the availability of platform compatible e-commerce services will be limited unless the platform is gaining strong support from service providers. Finally, the adoption of the platform requires integration of existing services and thus a strong commitment and initial investment from the e-commerce company.

The construction of the collaborative data management component itself should be the subject of the subsequent chapters. First, the canonical data model will be further specified based on the various existing reference models in the field. Second, the data access level must be defined to allow the various partners to work collaboratively and assure protection of sensitive information at the same time. The goal is to design these components that will undergo prototypical implementation and evaluation.

A Platform-Based Pluggable Trade Compliance Service

With specialized IT services that go beyond the features standard e-commerce packages provide, retailers can improve service quality and information quality, two of the main drivers for trust and customer loyalty (Yoon and Kim, 2009). As mentioned previously, cloud services are a promising approach to adopt specific functionality and to build an IT landscape with increased interoperability capabilities (Lewis and Fowler, 2014).

Despite the potential benefits of the service based approach, the adoption of new services can be cumbersome. In the previous chapter, we have shown that existing e-commerce services do not support their own adoption sufficiently. As Kephart and Chess (2003) state, the complexity of a system is higher than the complexity of the entirety of services. Thus, system complexity depends on the ability of the services to manage their own operation within the system and to reduce the requirements for system integrators to handle the complexity originating from a growing amount of services. Today's cloud services are a step forward to the easy adoption of individual pieces of functionality. However, as Martens et al. (2012) point out, the required resources for adopting cloud services are often underestimated, as the need for evaluation, implementation, configuration, and integration gets neglected.

Platforms have been propagated as a means to increase the quick connect capability within a network (van Heck and Vervest, 2007). The provisioning of services through a platform means to bundle the reoccurring resources and features across application components and to make them accessible for reuse. Using that pattern, the platform contains the stable components and supports the variety and evolvability of the other services by constraining their linkages (Baldwin and Woodard, 2009). Eventually, it can help retailers to decrease the efforts related to service adoption and allow service providers to offer services that are easier to consume. However, the available cloud platforms do not reflect all stable components within service systems. As we pointed out, domain specific data is often replicated across e-commerce services with a negative impact on pluggability. Thus, we proposed an extension of the existing integration platform model that reflects domain information and makes it available across services.

The goal of this chapter is to investigate whether the proposed model helps to

increase the pluggability of e-commerce services. The main contribution of the chapter lies in the elaboration of the collaborative data management component (cf. Section 4.4.2). The design and demonstration put forward represent the second design cycle of the DSRM framing this thesis. To this end, we come up with an ArchiMate model of the platform. The architectural model can be considered as a design artifact to solve the problem of low pluggability provided by current e-commerce services and platforms. The technical design draws upon recent technologies and protocols, originating from the social media and mobile service industry, and applies them in an enterprise information system context.

To evaluate the design artifact, we propose the business case of an e-commerce company that plans to get involved in cross-border selling to increase its customer base. A key hindrance for cross-border e-commerce endeavors lies in the diversity of legal trade regulations across countries (Accenture, 2011). Based on a prototypical implementation of the platform, we present a pluggable service for tax compliance in cross-border retail. The service helps retailers to calculate the correct foreign tax and can be adopted in a pluggable manner. The evaluation is carried out through qualitative comparison of the pluggability of prevalent application components in e-commerce with the platform-based service.

The chapter is further structured as follows. Section 5.1 contains an assessment of current e-commerce systems and their pluggability. In Section 5.2 we present the platform architecture and describe how it contributes to improved pluggability in the context of social media services. The prototypical implementation of the architecture is presented and evaluated in Section 5.3. In Section 5.4 we conclude with additional benefits and potential improvements of the platform.

5.1 Preliminary considerations

The adoption of IT services is often following the same pattern. The six phases of the service adoption lifecycle introduced in Chapter 3 occur in fairly every change process treating the introduction, update, or replacement of IT services. The adoption life cycle is used as a framework to assess the pluggability of a software component. In the following, we are going to discuss the current practice in IT service adoption with regard to the six phases. Subsequently, we are going to discuss the shortcomings in pluggability of current IT services, which emerges from the required resources during each phase of service adoption. The goal of the discussion is to pave the way for a new design that embraces these factors and reflects the proposed collaborative data component proposed in Chapter 4.

Current e-commerce architectures consist of a small set of large systems. More precisely, in most cases, they contain a shop frontend, an ERP system for back office tasks, and a WMS system for fulfillment logistics (Paramartha, 2014). Each of these application systems covers a large amount of functionality. Such monolithic systems are closely tied to the operational processes of the companies owning them, which makes it hard to detach them (Mandal and Gunasekaran, 2003). Fur-

thermore, these systems operate as one entity, thus consisting of tightly coupled modules, which are difficult to replace with atomic pieces of functionality, encapsulated in other services (Gattiker and Goodhue, 2004). In the following we discuss the six phases of the service adoption lifecycle, using the example of the shop frontend system. The front-office solutions for e-commerce contain a large set of functionality related to the selling of goods through the online channel. This encompasses product information management (PIM), storefront management, customer account management, marketing and promotions, analytics and reporting, and sometimes many more (Humeau and Jung, 2013).

Service provisioning

During the service provisioning phase, potential services are being evaluated and compared. Instruments exist to analyze the fits and misfits of the software package (Wu et al., 2007). Accordingly, the assessment of software packages can be divided into multiple sub-phases and can take a year or more. In general, the more complex the system and the higher its impact on the business, the higher the risk, and the more critical service provisioning becomes. Finer grained software components have less impact and thus entail less risk, which makes the provisioning more straight forward. Furthermore, the amount of available information on the software component can be considered as an important factor during service provisioning. Out of the six largest store frontend solutions mentioned by Humeau and Jung (2013), four prohibit the access to the product documentation. In most of the cases, only marketing brochures are available and detailed information about the software and can only be received through a sales or partner contact, making an unbiased assessment of the software particularly difficult. Furthermore, the pricing of the product licenses is, in many cases, not transparent. The availability of structured product information and pricing would increase the comparability of the software. The existing open source shop frontends have the advantage, that their documentation is openly available.

Service deployment

The prevailing shop frontend solutions are pre-packaged applications. However, a number of SaaS offerings started to emerge in the field. Packaged applications are usually deployed on-premises or on a cloud infrastructure by the service user. They require the installation and setup of a database and an application server, or web-server. Furthermore, security mechanisms must be in place such as firewalls and encryption of traffic for sales transactions. Depending on the amount of traffic and service level, failover and load balancing solutions might be required to maximize the availability of the system. Cloud systems, on the other hand, increase the ease of service deployment significantly as most of these tasks are handled by the service provider (Waters, 2005).

Service adaptation

In the adaptation of an application service, we can distinguish between configuration and customization. While the configuration allows a user to set up the system according to his needs, a customization requires changes in the source code or scripting. In general, the ability of the system to cover a maximum number of business scenarios through configuration decreases the need for customization, which in turn increases the ease of adaptation. A SaaS solution should aim for a high degree of configurability because customizations are difficult to realize by the user of an external service.

Service integration

The shop frontend has to share various resources with other application systems. For example, customer and product data must be shared with the ERP system and orders must be persisted into the order management module. Product stock and availability have to be shared in real time with the warehousing system, which is challenging for many e-commerce companies. Erroneous stock information in the shop will eventually result in a poor order fulfillment performance, which has a negative impact on customer satisfaction and loyalty (Rao et al., 2011). However, the current approach to enterprise application integration is a mix of messaging and scheduled batch processing techniques (Iqbal et al., 2007). For both, packaged and cloud shop frontends engineering of integration artifacts is required. Thus, the ease of integration can be considered as low in both cases.

Service operation

The service operation phase encompasses all the long term activities the application system requires. These can be a major cost factor, as they require permanent human resources in a traditional, non-cloud setting. These activities include a multi-level support, a service desk, and the maintenance of a knowledge base. Furthermore, the maintenance of the system is required to enhance the service and to apply patches, especially concerning security issues. If no support agreement with the vendor is in place, which might be the case for open source solutions, the maintenance might also encompass bug fixing. Also, major release updates might be necessary at the end of the software's support cycle. Such updates can have a high impact and require regression testing, re-engineering of customizations and integration artifacts, and re-import of the setup. In case of a SaaS front-office solutions, most of this tasks are outsourced to the service provider. However, the first level support might remain on site to establish a single point of contact for all IT related incidents. Furthermore, as discussed above, the integration of the shop is most of the times a custom IT artifact. Its maintenance and monitoring must be handled by the user, regardless of the fact that the shop might be a SaaS solution.

Service exchange

The exchange of the e-commerce frontend might impact the overall architecture, depending on what data is stored within the system and accessed by other applications. The replacement of the service requires an impact analysis: integrations will have to be reengineered and changes might be necessary on other systems. For example, changes in the sales process will need to be reflected in the backend systems.

From the investigation of the IT service lifecycle, using the example of the shop frontend, we can conclude that the adoption of application services for e-commerce is a resource-intensive endeavor in terms of time, expertise, and cost, resulting in a low pluggability. Online retailers that want to adopt new features to keep up with competitors have to accept high investments or wait until their software vendor introduces a similar feature in the next release. In the previous chapter, we found that SaaS solutions, in general, can improve the pluggability of a service. However, especially the issues in service integration, operation, and exchange cannot be solved by merely switching from an on-premises solution to a cloud equivalent. We argue that, to improve the pluggability of services, it is not enough to optimize the service itself. In fact, it is required that the platform allows service providers to build services that can be consumed in a more pluggable way. van Heck and Vervest (2007) claim that a superordinate platform helps to improve the capability of systems to quickly connect and act as a unit.

5.2 A pluggable service platform

In the previous section, we described the different phases of service adoption and pointed out the low pluggability of IT services for e-commerce in current approaches. In this section, we propose a platform architecture to increase the pluggability of individual services and the overall flexibility of the IT landscape.

5.2.1 Service platforms

The prevailing architecture for e-commerce is the information silo where the application system consists of a database, an application backend and an application frontend (Figure 5.1). The object-oriented domain model is mapped to a relational data model and the data of the application system is persisted to a dedicated database (Fowler et al., 2002). Eventually, the e-commerce company applies a number of self-contained systems. The need for collaboration between applications is ignored during their implementation and has to be solved by the user.

In other domains, IT services rely on a shared backend which encapsulates the common components across all services. The mobile industry applies this platform approach extensively. As a consequence, developers of mobile applications can rely on a number of platforms for marketplaces, gaming backends, user profiles, advertising, and geo-services. These platforms help service providers to reduce im-

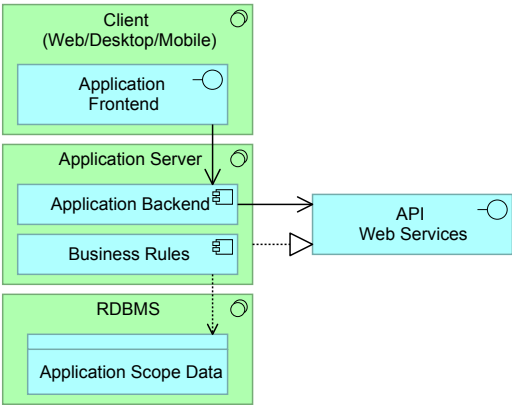


Figure 5.1: Common application architecture

plementation efforts, increase the collaboration between services, and enhance the end-user experience. As the platforms allow access to shared resources, the replication of data is often superfluous, thus decreasing data redundancy and increasing data quality. Furthermore, the platform design allows service providers to reuse common components and concentrate on the core functionality allowing them to offer finer-grained services.

In the previous chapter, we motivated the introduction of a collaborative data component where the canonical business data gets aggregated by the platform and shared across services. The e-commerce platform we propose in Figure 5.2 encapsulates the canonical data model (CDM) and the business rules for e-commerce (Hohpe and Woolf, 2003). Furthermore, the platform facilitates the definition of access rules which allows the e-commerce companies to grant access to their resources to the various services on an entity level. Finally, the application model extensions allow individual services to extend the data model. This helps to keep the CDM simple but versatile.

5.2.2 Platform architecture

In order to outline the platform architecture, we present an ArchiMate model which specifies IT and business concepts as well as their relationships (cf. Appendix A). The ArchiMate notation allows the visualization of components and actors involved and the description of the mechanism of interaction with the platform.

Figure 5.3 shows the ArchiMate model including the roles, platform components, and processes. The architecture includes three roles, namely the platform provider, the platform user, and the client who offers a service through the platform. There is no preference as to which actors should take which role. It is thus possible, that an e-commerce service provider is also running the platform. The platform itself contains four components and has two different interfaces.

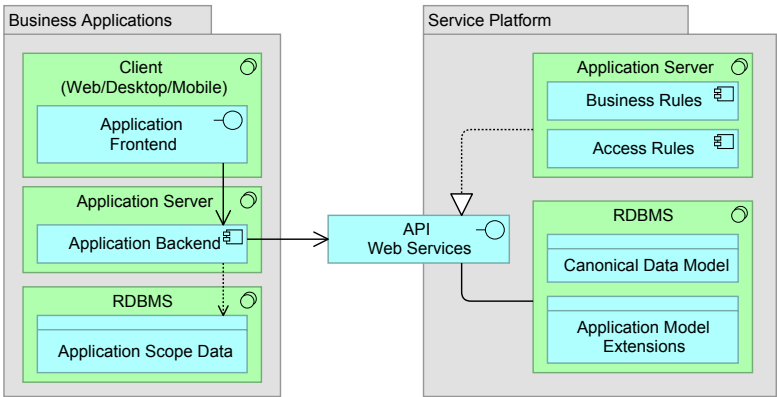


Figure 5.2: Segregation of service specific and shared resources

The multiple application components of the platform rely on two different data object types. On the one hand, a resource set data object is assigned to each user of the platform. It contains the core business data for e-commerce such as orders, customer data, and product information. On the other hand, the metadata contains extensions to the core data as well as the permission which grants access to the various clients.

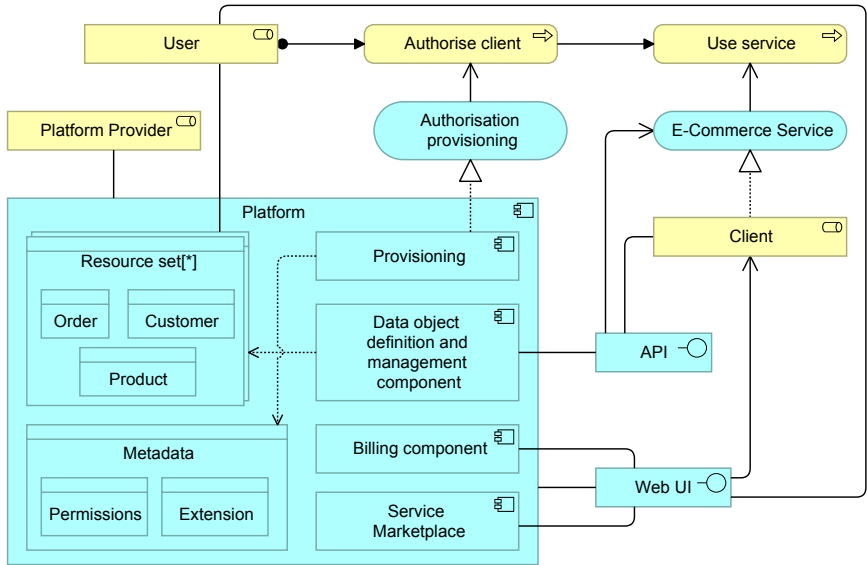


Figure 5.3: E-commerce platform architecture

The platform components accommodate functionality to look up services in a marketplace and to handle the payment of services. Furthermore, the client is able to look up and manipulate the business data as well as the setup of the metadata required for its service. Finally, the provisioning component handles the resource access authorization which is described in detail in Section 5.3.1.

Canonical data model

The resource set is one of the core components of the platform and is based on a CDM of the business domain. According to Saltor et al. (1991), expressiveness is a critical success factor for CDMs: ‘A CDM must have an expressiveness equal or greater than any of the native models of the component DBs that are going to inter-operate, in order to capture the semantics already expressed with the native models. Moreover, it should support additional semantics made explicit thru a semantic enrichment process.’

To adapt the cycle proposed by Saltor et al. (1991), we started from a native model of one service component and extend the model by successively adding more services. We stopped the enrichment cycle after the implementation of the platform-based end-to-end material return process covered in Chapter 4.

Figure 5.4 shows the result of the data model after four iterations of enrichment. The process was started with the service to register return shipments which requires access to orders, order lines, and customer information. The second service is the backend service for return merchandise authorization (RMA) processing and extends the data model with product information and returns (including return types, status types, and return reason types). The last two iterations for reverse logistic and customer communication rely on the same data and do not require any additional information. The enrichment of the data can be continued in the same way by adding more services of other e-commerce processes.

Metadata model

The heterogeneity of components relying on the CDM can be addressed through meta information (Busse et al., 1999). Therefore, we introduced two types of metadata in the architecture, client related metadata and infrastructure related metadata. Figure 5.5 shows the metadata model.

The main purpose for the client related metadata are custom data model extensions. It allows the client components to define additional attributes for each resource on the platform. The definition of new resource attributes is carried out declaratively, at design time, through the web UI. For example, a client application for temperature controlled transport planning can enrich the product entity with information on ambient thresholds. This information would be too specific for the CDM. The described approach dissolves the rigid structure of the data model and it can be adjusted to any use case within the context of the platform. Furthermore,

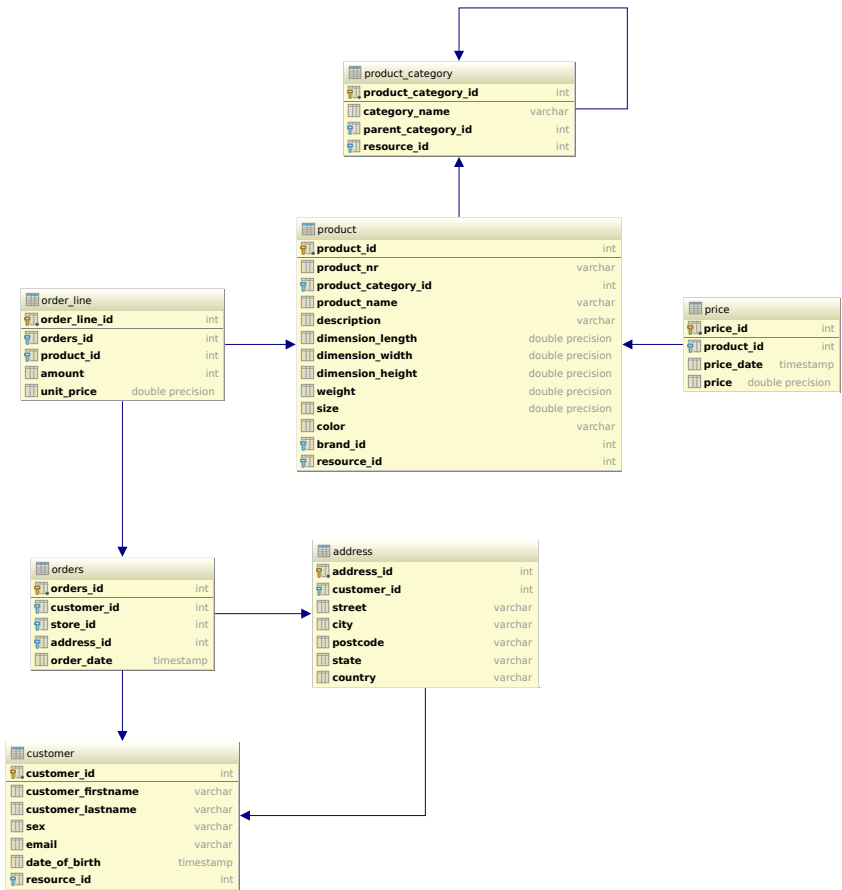


Figure 5.4: Platform architecture - canonical data model

the client can determine whether the scope of the additional attributes should be private or shared with other clients.

The infrastructure related metadata resolves various requirements related to data ownership and mutual data access. It allows configuring visibility and access of data among business partners. Through the use of access tokens, as in state of the art in social media platforms, users can grant platform clients the access to their resources (Hardt, 2012).

Interfaces

The platform provides two interfaces. A web user interface (web UI), which is used to access the marketplace and billing component. The core data can be exposed

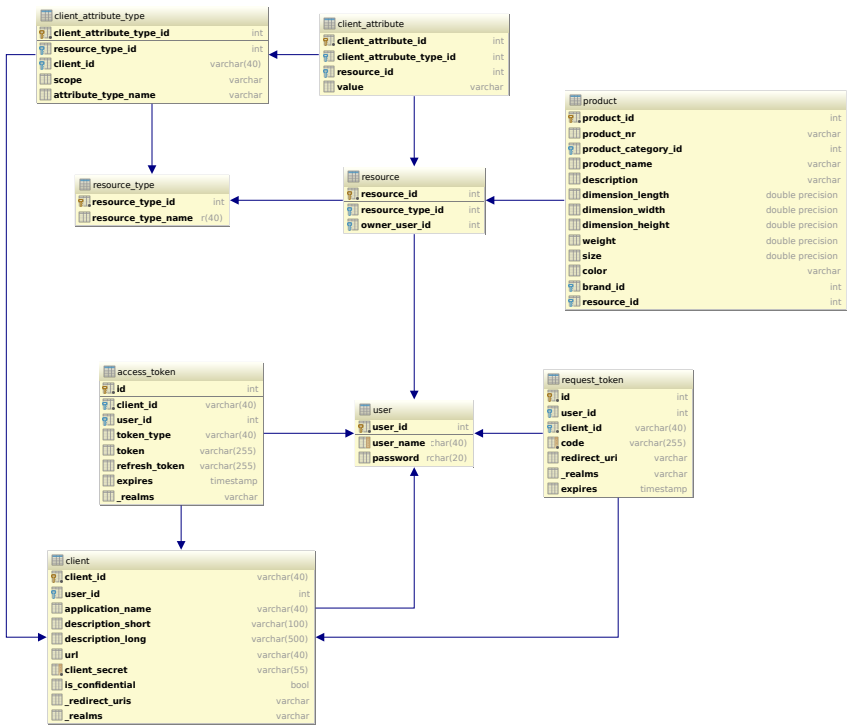


Figure 5.5: Platform architecture - metadata model

through the web UI to facilitate administration tasks on the core data. The computing interface, which is a REST API, consists of two endpoints for client authorization and access, as well as for a number of endpoints to access the business resources. Table 5.1 contains the description of the endpoints and the supported methods. The resource endpoints are available for every resource of the CDM and allow access to single resources or entire collections with the specification of filters.

The web UI of the platform prototype is shown in Appendix D.2 and has the four following main functions which allow the platform users to maintain resources, clients, and services.

- The documentation offers access to the user documentation and the documentation for service providers, including API documentation.
- The service administration allows the user to edit service subscriptions, search and subscribe to new services, as well as to retrieve billing and payment information for services.
- As the user can be service consumer and provider at the same time, the client administration allows the user to set up new clients which can be subscribed

Table 5.1: Platform API endpoints

Endpoint	Method	Description
/authorize	PUT GET	Redirect a user to the web page for authorizing the access to his resources.
/token	GET	Exchange and refresh tokens which give access to the resources.
/resource	GET PUT POST DELETE	Access and modify resources. Single resources can be accessed by their ID. Collections of resources can be filtered through parameters. Every request requires an access token. Additional attributes which are defined in the metamodel by the client are seamlessly integrated into the response.

by other users. It includes the setup of custom attributes and retrieval of subscriber information including invoicing.

- The business resource administration allows users to access their resource set. Users can view, update, and delete resources as well as metadata such as additional attributes created by clients.

5.3 The trade compliance case

In this section, we describe the design and implementation of a prototype, based on the described platform architecture. The purpose of the prototype was to gain insights into the feasibility of an e-commerce service, based on the platform, and to assess its pluggability. First, we describe the business case, the state of the art, as well as the platform-based solution. Subsequently, we present an evaluation of the solution using the pluggability model.

5.3.1 Business case

For the prototype, the business case of a cross-border trade compliance service was chosen. The capability of the service goes beyond what current e-commerce solutions provide in terms of supporting tax and customs compliance for cross-border e-commerce. The architectural limitations of the current landscapes do not allow online shops to manage legal regulation for cross-border selling in straightforward fashion. By addressing this challenge through a pluggable service, we can demonstrate not only the advantages of the architecture for the adoption of services but also its benefits in embracing novel business opportunities.

The goal of the service is to provide real-time updates of orders with international shipment, customs, and value added taxes (VAT) cost. That way, the buyer



gets accurate information on total costs of their online purchases from foreign retailers. The customer is able to receive information about all the costs related to the cross-border transaction. Furthermore, from a retailer perspective, the correct application of tax regulations often depends not only on a single transaction but on the total value of goods sold to a specific country within the fiscal period (European Commission, 2016). Thus, the service has to keep track of all cross-border transactions per country and apply the correct regulations.

Electronic Mall

Best prices and service.

2 Items

Verify your order

Product	Quantity	Price	Total
 <div>15 inch laptop by HP Status: In Stock</div>	1	1,439.90 €	1,439.90 €
 <div>Laptop charger by HP Status: In Stock</div>	1	59.50 €	59.50 €

Name	John Smith	Subtotal	1499.40 €
Street	Random Street 15	Shipping	6.05 €
Postcode	12345	VAT	314.88 €
City	San Francisco	Customs	36.29 €
Country	United States		
Total		1856.62 €	

Continue Shopping

Place order

Figure 5.6: Checkout with tax information

5.3.2 Current situation

The prototype is based on a working web service that was made available through the cooperation with a global trade service provider. The web service encapsulates the logic required for calculating foreign VAT that needs to be declared if selling abroad. In order to get a tax request, the corresponding product needs to be populated to the service beforehand. Listing E.1 in Appendix E shows the operations available from the WSDL interface of the service. The addProduct operation allows populating product master data to the trade compliance service which is required to derive the appropriate tax from the product category. The service also provides

a web application that allows maintaining the product information manually. The `getRequest` operation allows to request the actual VAT, customs, and shipping cost for an order containing one or more products.

The reference architecture proposed by the service provider is shown in Figure 5.7. The model illustrates the generic use of the service within a common e-commerce architecture. The core component of a state-of-the-art e-commerce architecture is a packaged e-commerce solution that encompasses product information management, customer management, and a storefront that allows customers to order products. To integrate the tax compliance solution into the architecture, a custom tax calculation component is proposed which is implemented as a customization of the e-commerce package. The customization handles the population of new products to the tax calculation service and carries out the tax calculation in real time during the order process through invocation of the remote web service.

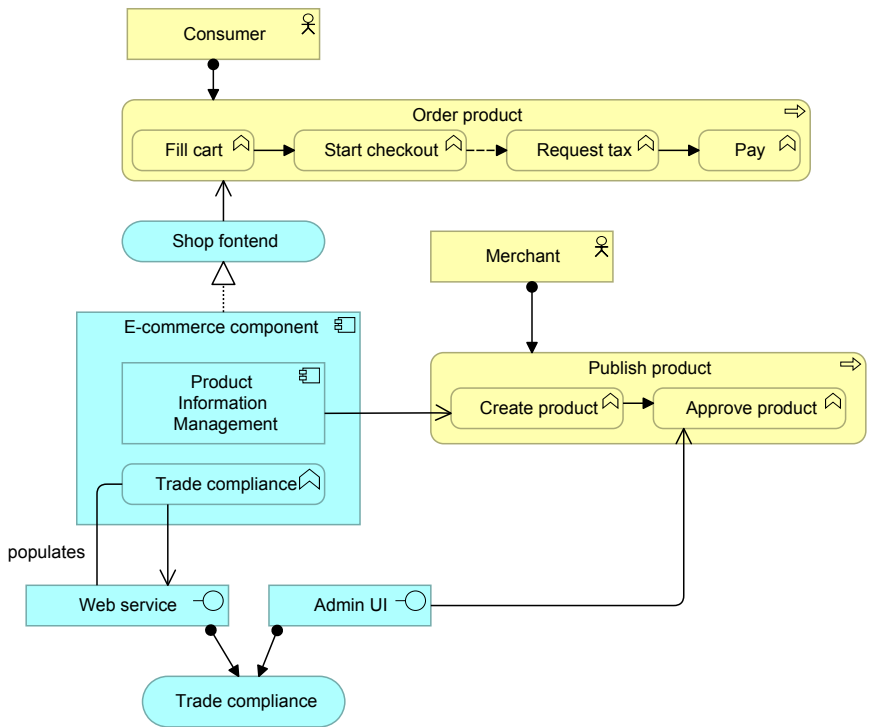


Figure 5.7: Original reference architecture for the tax compliance service

The provided web service takes as an input a message as shown in listing E.2 in Appendix E. The example contains two invoice lines for an order shipped to the united states. The payload includes the product number which has to be published to the service beforehand, the current price of the product, the amount for each

order line, currency, shipping cost, as well as the shipping address. Listing E.3 in Appendix E contains the corresponding response of the same message exchange. It provides detailed VAT information per order line, including the applicable VAT rates based on the product master data and the shipping destination. Furthermore, it contains the overall VAT, customs, and shipping cost for the order. The web service interface will be reused in the platform-based solution which is described in the following.

This approach has a couple of limitations, resulting in a limited pluggability of the service. The ease of deployment and ease of adaptation are limited by the requirement for a custom tax compliance component. In order to make the solution work, additional logic has to be implemented and deployed inside the e-commerce component. The adaptation requires technical knowledge of the components and also requires cycles of technical and functional testing. Furthermore, the ease of operation is affected as the custom component might not be supported by the e-commerce component provider. Finally, the exchange of the e-commerce component requires re-engineering of the custom tax compliance component.

5.3.3 Platform-based solution

Figure 5.8 shows the architectural model of the platform-based solution for the pluggable tax compliance service. The proposed solution dissolves the monolithic reference architecture and distributes the business logic across different pluggable microservices. It contains a service for the online shop that handles the sales transaction. A product information management (PIM) service allows the administration of products. Finally, the tax service handles the automatic population of new products to the tax calculation service and triggers the computation of foreign VAT and customs in real time.

In order to adopt the various microservices as pluggable services, the architecture earmarks them as clients of the platform. More precisely, domain data such as product information is present within the resource set of the platform user and shared across the services. There is no requirement for using a specific service for PIM or the online store provided that they rely on the platform.

The solution supports two process flows which rely on the two web service operations of the trade compliance service:

- The *publish product* flow handles the maintenance of product master data. The user utilizes the PIM client application to maintain product data. The trade compliance service subscribes to consequent changes in the product data on the resource set. In order to do so, the platform provides a streaming API as an addition to the usual request API. The difference between the two interfaces lies in the integration pattern between the platform and the client. While the request API implements the *request and reply* pattern, the streaming API reflects a *publish and subscribe* pattern (Hohpe and Woolf, 2003). Using the streaming API allows clients to act upon business events in real time. In

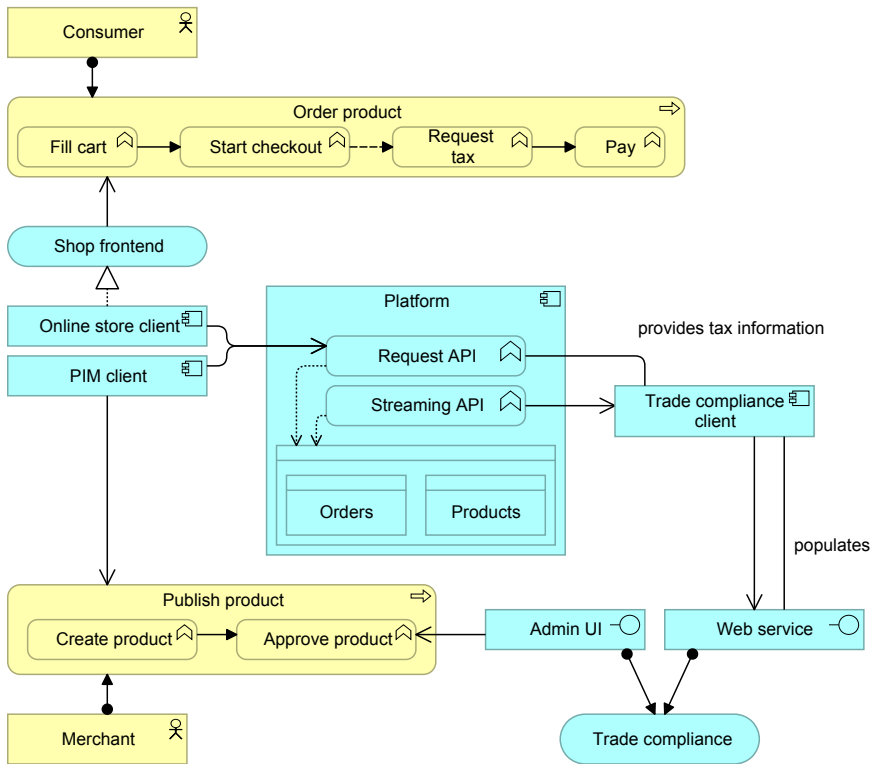


Figure 5.8: Proposed architecture

this case, a new product in the platform causes the trade compliance client to pick up the product information and publish changes to the trade compliance service through its web interface.

- The *order products* flow makes use of the same streaming API. The goal is to allow foreign customers to place orders which get updated in real time with the cross-border transaction cost. The crucial part of the implementation is that the additional costs need to get injected into the order during the checkout procedure. The trade compliance component picks up the initial order creation event and publishes the additional costs (cf. Figure 5.6). As a consequence, the online store client has to reflect changes that have been done after order creation. In order to do so, the platform has to support service hooks on public resource transactions. Such hooks allow services to block and release resources after subscribed events have been triggered. The diagram in Figure 5.9 shows the sequence of actions among the platform and the clients. As a general rule, it is necessary that clients have to consider the user hooks during implementation and re-query resources after public operations.

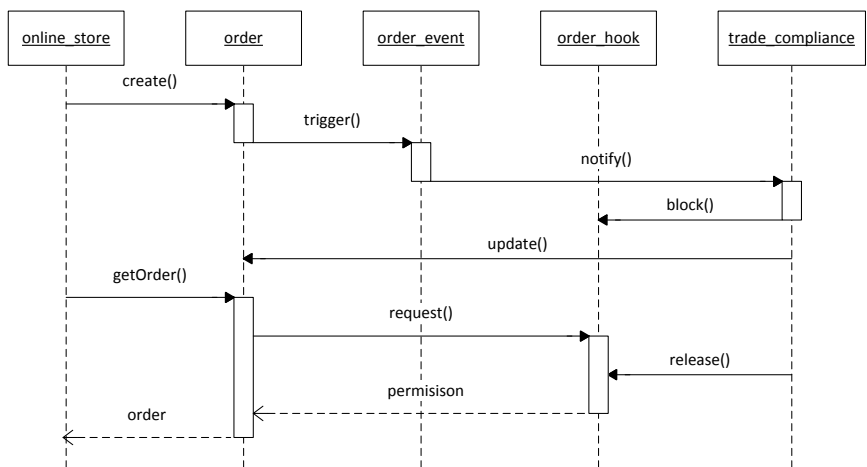


Figure 5.9: Order creation sequence

For the purpose of resource authorization, the platform implements the OAuth protocol (Hardt, 2012), which is commonly used in social media services in order to authenticate a user towards a third-party service and to share the user’s resources with the service. After the user visits the home page of the service, and chooses to subscribe, he is redirected to a page on the platform, providing the client information and the required scope of resource access. The user is then presented with a screen, as shown in Figure 5.10, where he can authorize the delegated resource access to the trade compliance service. After confirmation, the user is redirected to the service, together with an access token which allows the service from now on to access the user resources, according to the scope of the grant. Details about the platform implementation can be found in Appendix D.1.

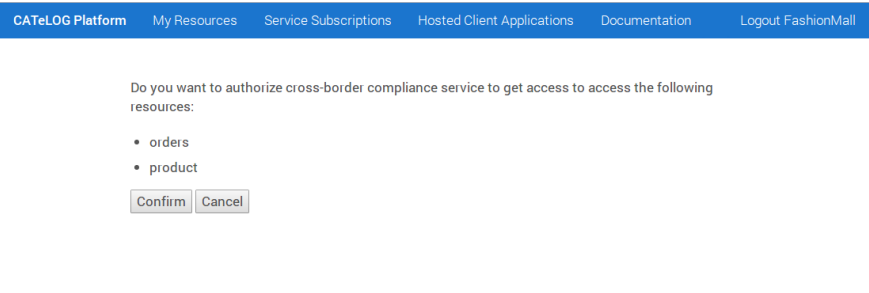


Figure 5.10: Resource authorization for trade compliance client

While the OAuth was originally designed for cross-site authentication and uni-

fied user accounts, it turns out that it is compatible with the mechanisms in enterprise application service collaboration. The only difference consists in the amount and granularity of access scopes. While in social media a limited number of resources, such as username and e-mail, are shared with the third-party service, the e-commerce platform has a large amount of resources (such as order, products, categories, customers, and returns) that can be shared. Furthermore, a distinction between read and write access of these entities has to be made when defining the scopes. However, it is possible to reuse the existing OAuth provider libraries. For the prototype, OAuthLib has been used (Gazit, 2012).

The second challenge, in addition to the delegation of resource access, is the implementation of services based on the platform API instead of a dedicated database. For the case of the prototype, all three clients rely on the data from the platform. While current architectures would aim for a local replication of the data, the goal of the platform architecture is to facilitate the online access to the business resources. Accordingly, the data access layer of the service has to be able to act as a web API client to request and send the resources directly from and to the platform. As current full stack web application frameworks expect the use of a dedicated database, the implementation of a platform-based service cannot be achieved with most of the available solutions. Some dedicated frameworks exist that facilitate the consumption of web services and REST APIs, which can be used to reduce the implementation efforts. For the prototype, we rely on RestTemplate to consume the platform API (Lui et al., 2011). However, the shortcoming of supporting technologies for the platform architecture makes the development of the service more cumbersome. Detail on the implementation of the platform client can be found in Appendix D.3

Another drawback of the platform architecture is the client performance. More precisely, the centralized, remote data repository increases the delay on the client side as the information must be transferred over multiple nodes. While current enterprise application architectures consist of a frontend and a backend, the communication with the platform has to be considered as another source of latency. Nevertheless, the latest innovations in the field stress the importance of rich clients, including client side caching and execution in combination with asynchronous communication with the backend. These mechanisms supersede the delays on the server and the platform, such that the extended backend will have minimal impact on the user experience. In the prototype, we do not address these technologies, however, we think it is important to consider this issue in the future.

As we can see from the architectural models in Figure 5.7 and Figure 5.8, from a user perspective, both architectures share the same business layer. However, the benefit of the platform solution relies in the adoption of the individual services. The use of the platform and platform-based services by the retailer offers potential benefits in terms of pluggability, and thus a shorter time to market. In the following section, we are going to elaborate on those benefits by evaluating the platform with regard to the pluggability criteria.

5.3.4 Evaluation

In this section, we present the evaluation of the cross-border compliance service with regard to its pluggability. We rely on the framework presented in Chapter 3, which is based on the lifecycle of service adoption and use. Figure 5.11 shows the results of the application of the instrument. Each phase of the cycle is being discussed individually, referring to the corresponding service lifecycle phase.

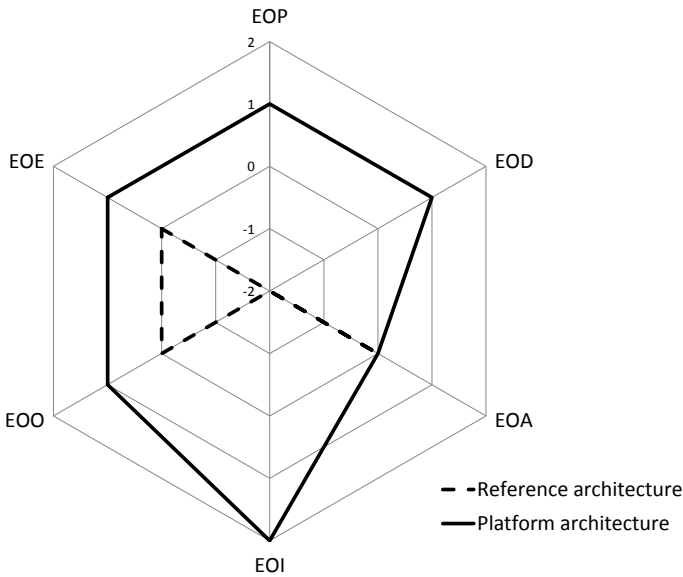


Figure 5.11: Pluggability of reference architecture and platform-based solution

Ease of service provisioning

The ease of service provisioning depends on how much the service provider supports the goal of making the capabilities of the solution transparent, and allows the user to assess the fit of the solution to the business needs. The existing tax compliance offering is only available upon request and requires individual contact with the sales department of the service provider. It is not possible to discover or learn anything about the service unless a business relationship with the service provider is established. Detailed information on the service capabilities is not available. The EOP of the current tax compliance solution is therefore on the low end. The advantage of the cloud solution is that it allows the service provider to relatively easy issue a demonstration environment for the customers to get hands-on experience, and assess the service. The trade compliance service does not require a high investment in the beginning for licensing and deployment. As the cost for the service arises

only from the subscription fee, the investment is distributed evenly over the time of use. Furthermore, the price of a cloud service is better predictable as it will not generate unexpected costs for product implementation and deployment.

The platform architecture aims to facilitate the provisioning of finer-grained services compared to the current service offerings. The trade compliance service adds a very specific functionality to the landscape which has a very limited impact on the overall system. Thus, there is no need for a long cycle of product assessment, a more comprehensive service would require. The user can try out the service in order to investigate fits and misfits. Compared to the reference architecture of the original trade compliance service (Figure 5.7), this does not require laborious customization of the online store. Furthermore, the marketplace component of the platform provides structured information on the service and its pricing, making it easier for the potential user to evaluate its adoption. We can conclude that the platform-based solution has major advantages for the user during the provisioning phase and thus a higher EOP.

Ease of service deployment

As mentioned in Section 5.1, the ease of deployment is a distinctive feature of cloud services in general. The trade compliance service can be used without the need for additional hardware or software artifacts. The platform facilitates the subscription to the service by implementing the OAuth protocol. The client has specified the required resource access in advance, during the service implementation. As a consequence, the deployment of the service happens in three steps, the subscription to the service, the authentication with the platform, and the authorization of the resource access. The next task of the user is to configure the service according to his needs which can be already considered as service adaptation. The EOD can, therefore, be considered as high. In comparison, the original reference architecture requires the user to implement the required tax compliance component, including its technical testing which leads to a low EOD.

Ease of service adaptation

Functional setup of the service and functional testing are the main tasks during service adaptation. The support of the platform-based solution does not go beyond what the original reference architecture has to offer. The administration interface of the tax compliance service allows the user to configure the service to his needs. However, the service shares a common drawback with other cloud services. The user will not be able to customize the system beyond the functionality anticipated by the service implementer. Therefore, both solutions have a medium EOA. If the service provider fails or denies to implement certain features, the user will not be able to customize the service beyond the provided functionality, even if he had the technical expertise to do so. However, to satisfy the customer needs, the cloud service providers are getting forced to improve configurability. The platform can be

a means of collecting change request from the various users. Furthermore, the advantage of the platform architecture in contrast to self-contained SaaS solutions is that multiple services of the same type can be used. For example, a user has the possibility to maintain the product data with different services because the underlying data is the same. Instead of adapting one service, the user always has the possibility to choose the right service according to the current needs.

Ease of service integration

The most significant difference between the original and the proposed architecture is the way in which the services interoperate from a user perspective. While both solutions rely on the same web service interface, the reference architecture delegates the task of integration to the service user, resulting in a very low EOI. The platform-based approach aims at a pre-built integration. With the unified data model of the platform, the integration of the services is embedded in the architecture. More precisely, we have a separation of data and functionality, through the federated information system (Busse et al., 1999), which solves the integration problem and leads to a high EOI.

Ease of service operation

As mentioned in Section 5.1, there are two types of service operation related tasks, namely, user support and service maintenance. The maintenance of the trade compliance service, including bug fixing and release updates, is managed by the service provider, in both the reference and the platform-based architecture. However, the required customization of the e-commerce component must be maintained by the service user, resulting in a lower EOO. The user support is also provided by the trade compliance service in both cases. However, if the user relies on the platform-based microservices for the PIM and online store, he will have to communicate with each service provider individually, which is not a good practice, as IT service management (ITSM) should aim for a single point of contact (Iqbal et al., 2007). In fact, we argue that the platform service provider is the suitable role to fulfill the first level support, coordinating tickets between users and clients. Hence, the platform model should be extended, and go beyond the provisioning of the technical infrastructure, and should include ITSM-related tasks.

Ease of service exchange

In terms of service exchange, ending the service subscription to the trade compliance service has no impact when using the platform-based solution. In the reference architecture, the ending of service subscription requires the re-engineering of the customization in order to avoid unwanted errors during the checkout. As a result, we obtain a higher EOE with the platform-based architecture.

5.4 Conclusions

In this chapter, we proposed a platform architecture to improve the pluggability of e-commerce services. We found that cloud services have an advantage over traditional application components with regard to pluggability, particularly during service deployment and operation. Furthermore, we found that the platform approach brings additional benefits with regard to service pluggability, which are not addressed by the prevailing cloud computing offerings. More precisely, service integration and exchange can be facilitated through the platform. The marketplace feature may provide additional advantages during service provisioning for users that require a structured overview of comparable services (Agrawal et al., 2013).

During prototype evaluation, we identified a couple of potential topics for further research around the platform architecture and service implementation. Namely, the impact on service performance and latency due to the additional node, and the potential of client side caching to overcome delays. Furthermore, the integration on the client side, i.e., the support of portlets and widgets through the platform, is another potential feature of the platform which will be elaborated in Chapter 7. Other non-technical extensions of the platform include the first level support of the various services as well as bug tracking and change request management by the platform provider.

Analytics as a Service: A Pluggable Sales Forecasting Service

The expertise of data analytics specialists or data scientists has become a critical success factor for organizations to understand and react to their environment. The shortage of skilled professionals and the resulting high cost causes a deficit of such experts in many domains (Davenport and Patil, 2012). As a consequence, notably small and medium enterprises (SME) often miss the potential that lies in unexploited information. In e-commerce, sales forecasting is one example of such critical analyses. Online retailers, that are able to compute reliable forecasts, based on existing sales transactions, can reduce losses caused by out of stock or non-selling items. Especially in short series product lifecycle fields such as fashion, it is crucial to have accurate figures on upcoming sales even before production.

Cloud computing in general and SaaS in particular are popular solutions among SMEs to share the costs for IT service development and operation (Danaiaata and Hurbean, 2010). Therefore, the task of data analytics for product sales forecasting is a promising application for the new cloud service model.

As figured out in Chapter 2 and Chapter 3, in the current system landscape of most online retailers, transactional data is scattered across various application system components and has to be preprocessed before it may be used. Data preprocessing consists of data cleaning, record selection, summarization, denormalization, variable creation, and coding. It is considered as the most time-consuming task in data analytics projects (Ordonez, 2011). However, with the prevailing architectures, collecting and cleansing data from various sources is a very system specific task and therefore difficult to implement as a reusable cloud service.

In this chapter, we build upon the platform architecture that has been established in the previous two chapters. After focusing on transactional services in the previous chapters, the forecasting service presented in this chapter is a means of demonstrating the utility of the platform for business analytics. To come up with a pluggable solution for sales predictions, we have combined the work on platform architectures with the joint research project on sales forecasting. Relying on state-of-the-art sales forecasting techniques, we present a new pluggable sales forecasting service as client of the platform architecture. The research goal was to design and develop a sales forecasting cloud service to allow SMEs to make use of advances in data analytics techniques.

In Section 6.1 we present the current research in sales forecasting and present a forecasting module which is the core component of the solution. In Section 6.2 we discuss the forecasting module with regard to the criteria of service pluggability and present the architecture for its transformation into a pluggable service. In Section 6.3 we present the prototype of a sales forecasting cloud service and evaluate its pluggability.

6.1 New sales forecasting module

New product sales forecasts are valuable for managers in supporting important decisions in operations planning (Cohen et al., 2000). For example, managers need to know how sales will evolve in the future in order to determine purchasing quantities and inventory. The number of new product introductions has been increasing over the past decade. As a consequence, managers need to perform new product sales forecasting tasks more frequently than in the past.

Despite the importance and prevalence of new product forecasts, these forecasts are seldom accurate. Kahn (2002) found that new product forecast accuracy was 58%, based on interviews with managers. Moreover, there are numerous approaches to forecasting new product sales, and these approaches may perform differently under different circumstances. Hence, managers would benefit from a tool that can incorporate multiple approaches to forecast new product sales and pick the most accurate approach.

In what follows, we briefly discuss new product forecasting approaches and describe the development and implementation of the new product sales forecasting module.

6.1.1 New product sales forecasting approaches

Goodwin et al. (2014) distinguish three categories of new product forecasting approaches: managerial judgement, judgement by potential customers and formal models. Managerial judgement relies on managers providing estimates of sales expected, typically using experience. Judgement by potential customers may involve for example expert panels. Formal models make quantitative projections using mathematical formulations of relationships between relevant variables. Since our goal is to provide a forecasting tool that is modular and can be used by multiple clients, we focus on formal models.

There are numerous formal models that can be used to forecast new product sales. These formal models come from both Statistics and Machine Learning areas. Some models have roots in marketing or economic theory, while others are purely data-driven. We will use a commonly used statistical model based on Marketing theory (the Bass model), and one more data-driven model (latent-class regression).

Originally proposed by Bass (2004), the Bass model and its derivatives are widely used in Marketing to characterize the lifecycle of a product. In order to forecast new

product sales before launch, the parameters of the Bass curve for the new product can be estimated using analogy by considering ‘similar’ products introduced in the past.

We furthermore consider a latent-class Poisson regression model (Wedel et al., 1993) in combination with concomitant variables (Grün and Leisch, 2008). The advantage of this model is that it estimates the two models simultaneously: one model for clustering product lifecycles and one model for assigning cluster probabilities to each instance based on concomitant variables.

6.1.2 Algorithms and implementation

We build our forecasting module using the R software (R Development Core Team, 2014). It has a few advantages, including the fact that it is free, open source, and contains a large number of statistical packages that facilitate rapid model development.

Before the sales forecasting module can be utilized, it is very important to supply clean data to the module in the format that it requires. In our application (cf. 6.1.4), data had to be aggregated, cleaned, and new features had to be derived. Some of these procedures require domain-specific knowledge. It can therefore be expected that data preprocessing may encompass different procedures for different clients. R data management libraries, such as *plyr* (Wickham, 2011) or *caret* (Kuhn and Johnson, 2013), may greatly facilitate the development of client specific data handling procedures.

There are three key functions in the sales forecasting module: 1) model tuning 2) best model selection and 3) forecasting. Model tuning refers to finding the best model-specific setting. For example, in the case of latent-class regression, model tuning implies selecting the number of clusters that minimizes cross-validation error. Selecting among models is done based on cross-validation errors - we select the model with the least error. This model is used to perform the forecasting task. For more details on model tuning and selection, we refer the reader to Kuhn and Johnson (2013).

Both forecasting models described in Section 6.1.1 were implemented in R using the *stats* (R Development Core Team, 2014) and the *flexmix* (Grün and Leisch, 2008) packages.

6.1.3 Case description

We demonstrate the functionality and performance of our forecasting module by using sales data of a large Dutch apparel retailer. In apparel retailing, assortments are renewed at least two times per year, and new item introductions are common.

Our data includes monthly sales of 43 brands in the period between 05-02-2009 and 21-02-2013. A brand can have multiple collections, styles, colors and sizes. We aggregate across these variables to arrive at brand sales data. We observe average prices, discounts, inventory levels and number of unique stock-keeping units

within each brand. Each brand is characterized by its functionality (an internal company classification variable) and average non-discounted price. Sales series of four selected brands are shown in Figure 6.1

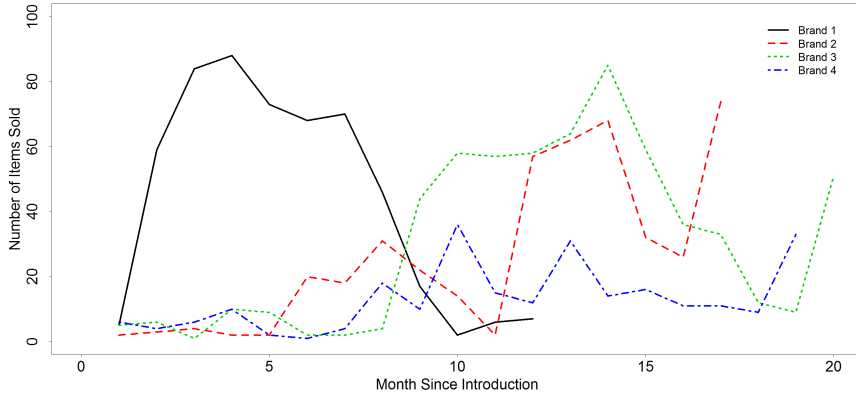


Figure 6.1: Sales series of selected brands

We can see that brands exhibit different lifecycle patterns and that sales peak at different moments.

6.1.4 Capabilities and output of the new product forecasting module

Our objective is to forecast sales of new brands in a given category prior to their launch. We split the data into a training set and a test set, where training set data include observations until 01-01-2011, and the test set includes sales data on new products that are launched after this date. The splitting procedure resulted in 22 brands in the training set and 21 brands in the test set. We tune the models and select the best-performing model on the training set, and evaluate forecasting performance on the test set. Therefore, we imitate a real-life scenario of a pre-launch forecast.

Training set results

We used leave-one-out cross-validation to tune and evaluate the Bass model and the latent class regression. For the Bass model, we first fit the Bass curve to each brand separately and obtain the parameters. Next, to predict the sales curve of a new brand, we use its attributes to compute the “distance” between the current brand and all brands in the training set. We use n closest brands and computed the average p , q and m values. Next, based on these average values, we predict the sales of the new brand. Hence, n , the number of closest brands to consider, is our tuning parameter. For the latent-class regression, we used leave-one-out cross-validation to fit the model with k different clusters. Hence, k is the tuning parameter for the

Table 6.1: Performance on test set

Months since introduction	MAPE Bass model	MAPE latent class regression
1	1119	158
2	1036	115
3	3886	156
4	2037	99
5	1005	78
6	2405	100
7	4751	261
8	900	157
9	2122	103
10	1695	131
11	360	153
12	449	217

latent-class regression model. The optimal configuration, giving the lowest mean absolute percentage error (MAPE) turned out to be $n = 2$ and $k = 3$.

We compare the MAPEs of both models with the best configurations using the t-test. We reject the null hypothesis that the mean performance of the Bass model is better than that of latent-class model with p-value of 0.01. Thus, we expect the latent-class model to perform better on the test set.

Forecasting performance

We use both models to predict the sales of new products in the test set. Table 6.1 provides MAPEs, aggregated across brands, for each month since new brand introduction. It is important to note that the results described in this section are preliminary and should be interpreted with care.

We can see that the latent class regression performs better than the Bass model. This is due to the fact that it incorporates decision variables (pricing, discounts, and stock levels). The forecast errors are rather high for both models. This is possibly due to the fact that we do not have data on many brand attributes, and it is difficult to establish similarity between brands based on current attributes alone. Figure 6.2 provides several plots with actual and forecast sales for several brands.

6.2 Pluggable architecture

In Section 6.1 we have shown how past sales transactions can help retailers to predict future sales. However, to obtain a complete, ready to use sales forecasting cloud service, architectural questions arise and will be discussed in the following. We rely

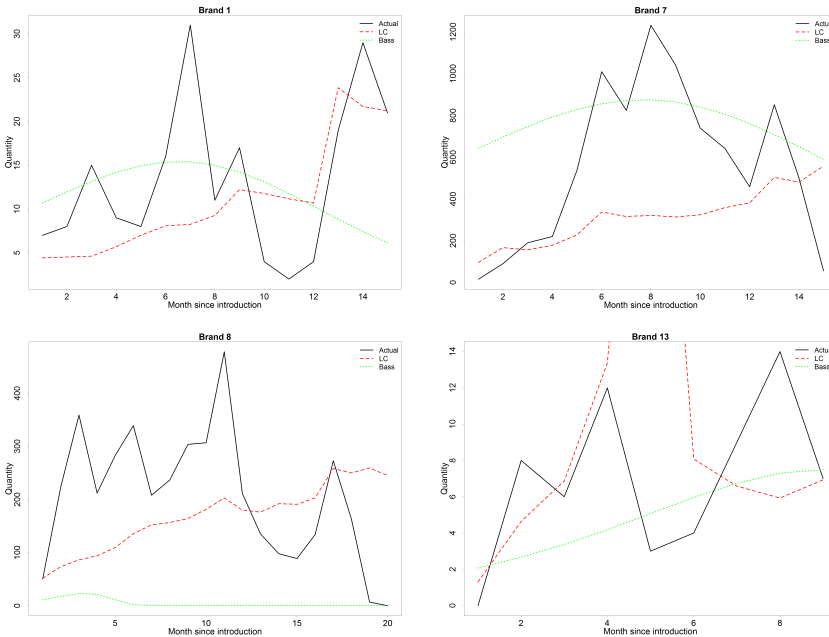


Figure 6.2: Forecast vs. actual sales for selected items

on the concept of pluggability, which will help us to understand the issues of the forecasting module and later on, to evaluate the solution proposed in this chapter. We demonstrate the use of the platform architecture which forms the base of the implemented prototype in Section 6.3.

6.2.1 Pluggability

In Chapter 3 we introduced the six criteria for pluggability. In order to transform the core forecasting module into a pluggable cloud service, all the criteria have to be addressed. In the following, we are discussing the forecasting module with regard to the criteria.

Ease of Provisioning (EOP) is the ability of the service to support the user in selecting a suitable service and to anticipate the costs, efforts, and benefits of its use. In case of the forecasting module, it is not possible to oversee the capabilities of the module from a business user perspective. Thus, it is difficult to predict the costs, associated with the transformation of the core module into a ready to use business application.

Ease of Deployment (EOD) means to minimize the efforts for installing the service, including the allocation of hardware and system software resources. If the forecasting module would be distributed as is, the user would have to deploy suit-

able hardware and software as well as to provide suitable application components in order to support the end user.

Ease of Adaptation (EOA) has two different aspects. The adaptation through configuration by a business user as well as the adaptation of the service by technical experts. The forecasting service offers a maximum flexibility for software developers to reuse the forecasting functionality. However, it does not support a configuration by a non-technical business user.

Ease of Integration (EOI) describes the capability of a service to interact with other IT components. The data gathering and preprocessing tasks mentioned earlier can be considered as aspects of integration. The forecasting module requires the preprocessed data as input and does not support the user with gathering and cleansing the data from other services.

Ease of Operation (EOO) encompasses all continuous tasks after setting up the service. Maintenance of services includes, for example, bug fixing, functional enhancements, security updates, and end user support. While bug fixes and enhancements could be distributed in an automatic fashion, the maintenance of potential additional application components and end user support needs to be carried out by the consumer.

Ease of Exchange (EOE) of a service often relates to the dependencies with other components. If services depend on each other, the process of exchange is getting more complex. Services, such as the forecasting module, that only serve reporting purposes can usually be removed without affecting the rest of the landscape.

It is apparent that the forecasting module does not cover all quality criteria of a pluggable service. Thus, in the following section, we present an architecture for a pluggable forecasting service with the module at its core.

6.2.2 Architecture

To support retailers with the adoption of innovative e-commerce services, we introduced the pluggable service platform in the previous chapters. At its core, the platform contains a canonical data model (CDM) to share e-commerce related information across services. Furthermore, it provides an application programming interface (API) to give service providers access to the shared resources. It allows the platform clients to implement e-commerce services in a federated fashion (Busse et al., 1999). The CDM and the federated nature of the platform help to reduce the efforts in data gathering and preprocessing required for the forecasting service.

The architecture of the forecasting service and the interaction with the platform is shown in Figure 6.3. The forecasting service component has four application functions. It interacts with the platform through the same API as transactional services such as order management, online store, product information management. It provides a wrapper around the core forecasting module, which transforms the data from the platform into the suitable format, triggers the prediction module generation, and requests individual forecasts. The subscription mechanism uses the standard authentication flow provided by the platform. It allows potential users to

subscribe to the service by entering its platform credentials and granting the service access to the shared resources. The web application allows the user to configure the service and displays the output of the forecasts. Finally, the scheduler is available for long running jobs. As the generation of the prediction models usually takes a longer time, it has to be referred to the background. Users can schedule the model generation periodically or on demand.

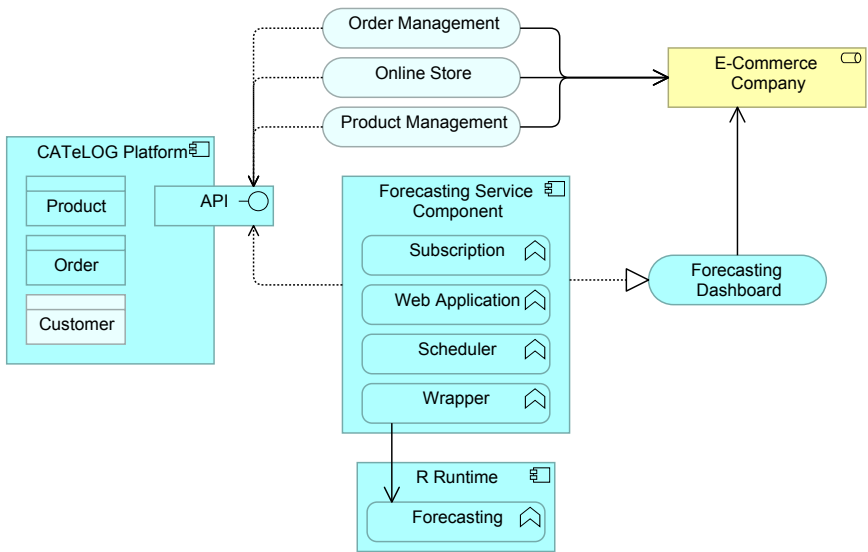


Figure 6.3: Forecasting service architecture

6.3 A pluggable sales forecasting service

In order to evaluate the architecture, we created a prototype of the forecasting service based on the platform. In the following we provide a description of the prototype and evaluate the pluggability of the implemented forecasting service.

6.3.1 Prototype

The prototype has been developed using standard web application technologies, an SQL database for storing data within and outside the scope of the platform, as well as various common libraries for web APIs, OAuth authentication flow between the platform and the service, interfacing the R module, and job scheduling.

Figure 6.4 shows the web application user interface. The user has the option to choose between various forecasts and to schedule the prediction model and forecast generation jobs.

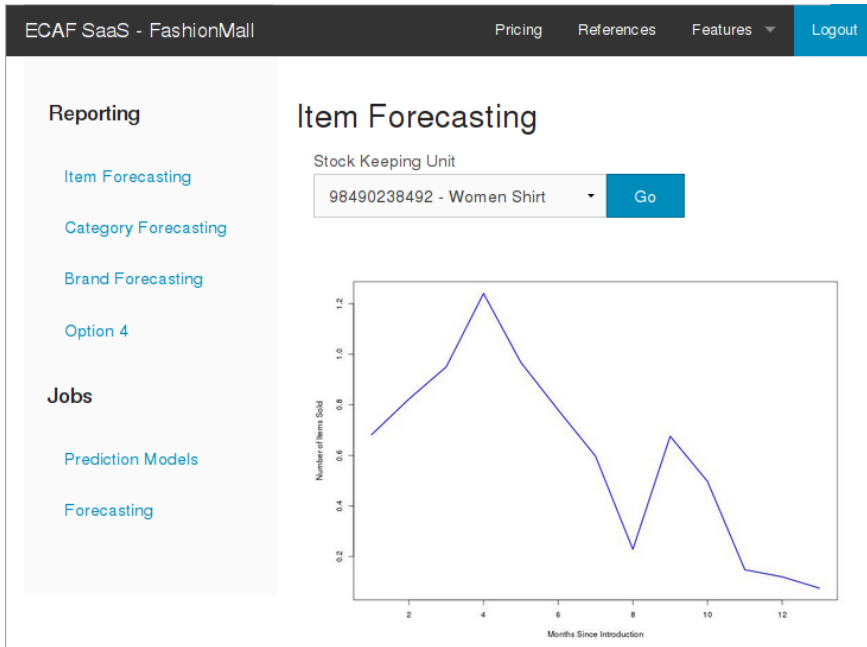


Figure 6.4: “E-Commerce analytics and forecasting (ECAf) SaaS” web application.

6.3.2 Evaluation

The goal of the architectural design and prototype was to transform the state-of-the-art forecasting module into a pluggable cloud service. According to prevailing design science research methodologies (Peffer et al., 2007), the design and demonstration of a design artifact should be evaluated by observing if the artifact that provides a solution to the design goals. In Figure 6.5 the outcome of the applied the pluggability instrument is shown. We compare the pluggability of the implemented service with the pluggability of the forecasting module in Section 6.2.1. The results are discussed in the following.

EOP The forecasting module has an average EOP. The information on the module functionality can be hosted publicly and assessed by the service user. There is no possibility to get hands on a working solution unless a custom forecasting application is getting implemented based on the module. By introducing the service with the forecasting module at its core, we can achieve a higher abstraction of service. While the forecasting module is providing functionality, the service is providing business value (Haesen et al., 2008). Thus, it gets easier for the potential user to map the service features to the business requirements, eventually improving the EOP. Furthermore, if the service is offered as a platform-based artifact, it is possible to discover and compare the

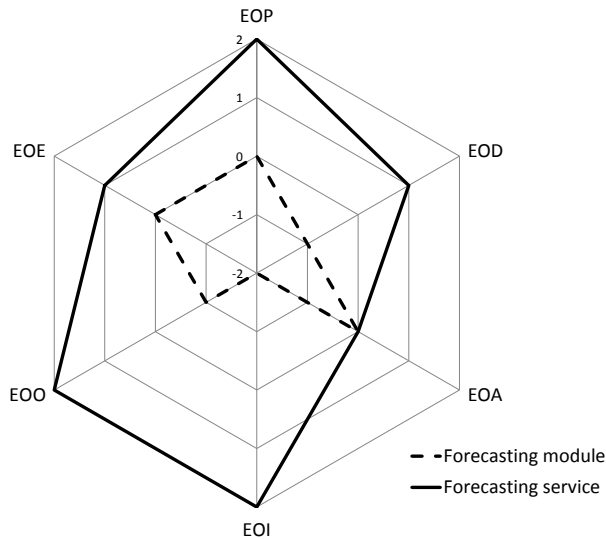


Figure 6.5: Pluggability of forecasting module and forecasting service

services in a marketplace fashion. This could further improve the EOP of the service over the plain module.

EOD As the forecasting service is cloud-based, the user does not need to deploy any software or hardware resulting in a high EOD. The service offers a subscription by using the platform credentials. After subscribing to the service, the user can go over to the adaptation phase. The forecasting module, on the other hand, has to be deployed by the user on-premises or by using a cloud platform. The requirements for the platform are relatively low compared to other artifacts (with the R runtime as a minimum requirement) resulting in a low to medium EOD.

EOA Similar to the aspects of the provisioning phase, the service also supports the user in terms of service adaptation. The configuration and setup can be done within the web interface, resulting in a higher EOA. However, the possibilities of service customization through developers are very limited, resulting in less flexibility in adaptation. This is a common disadvantage of cloud services compared to tailored or packaged on-premises solutions. Due to the inflexibility of the configuration, the EOA for the forecasting service could be negatively impacted. The forecasting module, on the other hand, has a high flexibility in adaptation but requires the technical expertise, which also limits the EOA.

- EOI** The EOI depends on the work that is necessary to connect the service with other components. In order to operate accurately, the forecasting module requires a preprocessed data set that contains the right data fields and records. In current e-commerce architectures, sales transactions and product information are stored across different systems such as product information management, order management, or online shop frontend. Making the forecasting module work in such heterogeneous environments is a major obstacle and is the most time-consuming task in its adoption. As a consequence, the EOI for the forecasting module is very low. By relying on the platform architecture, the provider of the forecasting service can pre-integrate the service with the CDM. Thus, the service user does not have any additional tasks with regard to service integration and the EOI could be improved significantly by using the platform.
- EOO** By making the service cloud-based, the service operation is shifted from the user to the service provider. The user requires practically no additional resources to maintain service operation, resulting in a high EOO. The forecasting module requires a custom solution on top of the core functionality which leaves problem resolution, manual updates, and user support to the user.
- EOE** As mentioned previously the EOE has limited relevance for reporting services. The EOE is slightly easier with the cloud solution if it is replaced with a comparable service. The impact on the remaining IT landscape is relatively low in both cases.

6.4 Conclusions

In this chapter, we have shown how a sales forecasting cloud service can be designed and implemented based on a state-of-the-art forecasting module. Furthermore, we verified the pluggability of the prototype with regard to the six criteria. It was shown that the pluggability of the service exceeds the pluggability of the plain forecasting module, and offers the user a solution that is easy to adopt. We can conclude that the capability of the platform presented in the previous chapters go beyond transactional services and provide the same benefits for business analytics. The solution can be particularly interesting for SMEs that do not have the resources for a comparable on-premises solution. However, it is required that the platform is in place and an ecosystem of services and service providers has been established.

Using Pluggable Services to Support IT-Driven Collaboration in Business Networks

Organizations are getting increasingly specialized in their contribution within value chains. Tasks that were previously handled within a company are forwarded to service providers and are nowadays often spread across a number of different parties, forming a business service network (Tenenbaum and Khare, 2005). To achieve agile business processes in this setting the network and its participants have to improve their capability to quickly connect (and disconnect) with each other (van Heck and Vervest, 2007). The time and cost required to engage with new business partners are core aspects of the emerging concept of quick connect capability (QCC) (Koppius and van de Laak, 2009; Merrifield et al., 2008). From an IT perspective, the QCC depends on the capability of applications and of the IT architecture to reduce the effort to adopt new services. The limitation of the current IT systems is often considered as a reason for the limited agility in business networks (van Hilleegersberg et al., 2012). More specifically, the application components involved in the collaborative processes are not constructed with interoperability in mind.

Various authors propose a platform-based approach to facilitate inter-organizational collaboration (van Heck and Vervest, 2007; van Hilleegersberg et al., 2012). The existing platforms facilitate the task of linking the endpoints of systems within and across organizational boundaries. However, as outlined in Chapter 4, the current practice still requires the implementation of dedicated integration artifacts that contain the business logic and model transformations required for system interaction. Such artifacts link two or more systems and get deployed onto one of the systems or the integration platform, which then acts as a mediator between systems. The approach entails a number of limitations:

- A dedicated collaboration artifact needs to be built for each and every system involved in the collaboration.
- One integration artifact usually focuses only on one specific business process or use case.
- Changes in the business processes or integrated systems usually require re-engineering of integration artifacts.

- Design decisions and its required changes of the integrated system can lead to conflicts between the business partners.
- The responsibility for the integration artifact and the resources required for its operation can lead to conflicts between the business partners.

The overhead of building dedicated integration artifacts can lead to delays in establishing working business collaboration. Furthermore, the requirement for building integration artifacts can stress the relationship between partners. We can conclude that the practice of building integration artifacts on top of the existing information systems leads to inflexible collaboration, despite the use of existing integration platforms.

To achieve a higher level of interoperability among the systems involved in a business to business transaction, we propose the use of collaborative services based on the platform we outlined in the previous chapters. The difference between an integration artifact and a collaborative service is the higher level of pluggability. In other words, the collaborative service can be consumed out of the box while an integration artifact requires tailoring to a specific combination of existing information systems. The benefit is the ability of the network partners to engage into a business collaboration in an ad-hoc manner. Thus, leading to an increased quick connect capability within the network.

To the same extent as the electronic channel enables improved collaboration with partners, it bears the risks of decreased customer loyalty. On the one hand, new technologies allow faster engagement in relationships. On the other hand, customers that previously stick to a single vendor can switch between suppliers with more ease. As a consequence, in the e-commerce market, the acquisition of new customers plays an increasingly important role (Srinivasan et al., 2002). A popular instrument for customer acquisition relies on a collaboration among e-commerce firms in which one firm refers customers during a sales transaction to a product of its partner. The mechanism is known as cross-selling. Vendors that manage to establish cross-selling partnerships with other firms can profit from their customer base and get incentives for referring their own customers. To demonstrate the collaborative capability of the platform architecture, we provide the example of a platform-based collaborative cross-selling service that allows e-commerce firms to establish such partnerships in an ad-hoc fashion. It allows not only a mediation between two online retailers but provides a ready-to-use solution for the entire cross-selling lifecycle, including product matching, referral, and settlement of commission fees.

In this chapter, we revisit the object definition phase of the design science research cycle. We shift the focus from the previous goal to provide a platform that supports pluggability of e-commerce services to a platform that supports the quick connect capability of partners in the e-commerce network. Figure 7.1 shows the five steps of the DSRM and maps them to the generic stages of DS proposed by Verschuren and Hartog (2005). The structural specification of this work is represented

by an architectural model and the implementation is represented by a prototype. While the evaluation of these two artifacts is a vital step in the method, the gained knowledge, as pointed out by Cross (2006), essentially resides in the two artifacts which are the product of a design process. This differs from the paradigm in empirical research where the collection of data and its formal evaluation is the key factor to gain insights into certain phenomena.

In the next section, we are going to outline the state of the art in collaboration architectures and set the goals for an architecture that addresses the mentioned limitations. Afterward, we discuss the requirements for the architecture which will also be used to evaluate the final prototype. The functional specifications and the implementation are the subjects of the subsequent two sections. In the final sections, we evaluate the proposed artifacts and provide the conclusions and pointers to future research.

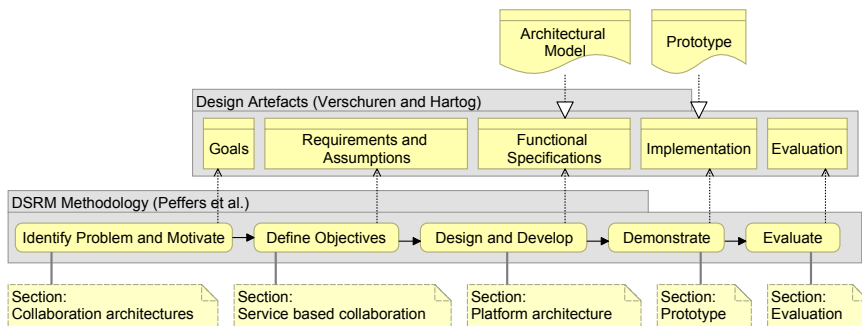


Figure 7.1: Research design

7.1 Collaboration architectures

To assess the prevailing inter-organizational IT architectures, we present a review of the state of the art in collaboration architectures. Based on the identified limitations imposed by the prevailing IT architectures in the literature new ways in collaboration can be developed. According to Verschuren and Hartog (2005) the problem identification and motivation in DS should result in the formulation of a small set of goals, which we outline in the second part of this section.

Since the early days of electronic, inter-organizational collaboration and trading, the heterogeneity of information systems has been identified as a major challenge. Particularly the incompatible representation of the relevant information and semantic differences in the systems of trading partners are major obstacles and reduce the expected benefits of IT-driven collaboration (Lincke and Schmid, 1998). To overcome these issues various approaches based on the core concepts of mediation (Wiederhold, 1992) and federation (Busse et al., 1999) of information systems

have been proposed. The common goal of these approaches lies in the introduction of standards that help to cope with the issue of heterogeneity.

7.1.1 State of the art

The goal of mediation is to transform data from various sources and to make them accessible in a unified structure. Handschuh et al. (1997) and Lincke and Schmid (1998) for example, propose various models for mediating product catalogs to help business partners to obtain a standardized view on product information and thus, facilitate transactions. A discussion whether the mediation should be handled by an intermediary or not has been raised by Sen and King (2003). The benefits of intermediaries according to Palmer and Johnston (1996) are additional security and virtual marketplaces that help to initiate new partnerships. Mediation handled by so called brokers is particularly interesting for markets with a large number of participants as they can add trust and reduce the transaction costs (Bichler et al., 1998).

The implementation of mediating technologies such as electronic data interchange (EDI), requires a stable and long-term relationship between partners. Reimers (2001) claims, that Extensible Markup Language (XML) technologies, due to their self-descriptive nature, are more suitable for ad-hoc connection of services. Standardize message formats are available, such as EDIFACT or more recent XML-based formats. However, such data standards are limited to the exchange of transaction information and not suitable to support an integrated architecture for agile collaboration (Vujasinovic et al., 2010).

Web services have been propagated more recently to solve the problem of static communication patterns between dedicated systems (Papazoglou, 2003). The use of common internet protocols provides some advantages over the use of proprietary business-to-business (B2B) networks. However, the most promising mechanisms of SOA, such as automatic service discovery and consumption through Universal Description, Discovery and Integration (UDDI), have not found their way into practice to date. The use of web services by default does not change the inflexibility of the prevailing communication pattern: Integration artifacts have to be built including mapping of data, serialization and deserialization of messages, interface testing, monitoring, and maintenance. Another more sophisticated approach to reducing the required engineering efforts to connect web service endpoints are semantic web services. Their goal is to make machines understand the ontology of the data and let them act in a more autonomous fashion. However, to date, semantic web services do not find a wide adoption beyond academic studies or prototypical implementation and have not yet solved the problems of agility in inter-organizational collaboration.

A term that has also evolved more recently in the domain of enterprise application integration is the canonical data model (CDM) (Hohpe and Woolf, 2003). Originating from a messaging paradigm, it supports the idea of increased interoperability by applying unified data models across collaboration partners. However, the use of CDMs has not been widely reflected in research to date. In particular,

a validated CDM development method is lacking. Other domains provide concrete construction methods for unified data models such as for data analytics in the research field of data warehousing (Prat et al., 2006) or ontologies in the field of semantic systems (Aussenac-Gilles et al., 2000). However, no concrete methods exist that describe how to build a unified data model for transactional systems.

To date, the concept of the unified data model can be found in many ERP systems, as they rely on a centralized database that is used across business domains. Instead of providing distinct applications per business function, a monolithic solution is provided around one central database (Kumar and van Hilleberg, 2000). The modules provided for various business functions across an enterprise share the same underlying database, thus avoiding the need for integration mechanisms across autonomous applications. As the ERP systems are set to cover the value chain within enterprises, they are not suitable to overcome the heterogeneity in inter-organizational settings. ERP systems that are shared across organizational boundaries usually use a multi-tenancy architecture which isolates the resources of potential business partners (Bezemer et al., 2010).

7.1.2 Goals

We can summarize that modern inter-organizational collaboration within business networks is mostly realized through dedicated intermediated or point-to-point integration artifacts. It allows organizations to exchange the required information for a specific business transaction. The concept of mediating CDMs can only be found in ERP systems within organizations and has been neglected in the recent debate on inter-organizational interoperability.

In order to propose the requirements for a new design, Verschuren and Hartog (2005) claim that a number of *goals* have to be formulated as a means to evaluate the approach. In our case, we aim at an improvement of the QCC of organization within a business network. Koppius and van de Laak (2009) state that the QCC consists of the three factors *quick connect*, *quick complexity*, and *quick disconnect*. A suitable solution to support a pluggable inter-organizational collaboration should therefore contribute to the following goals:

- G1: Individual services can be adopted quickly (quick connect).
- G2: Complex inter-organizational functionality can be handled using appropriate collaboration services (quick complexity).
- G3: Disconnection of individual services or partners will not affect any remaining services or collaborations (quick disconnect).

As mentioned earlier, in this chapter we want to reflect the business case of two online retailers with a complementing assortment who want to establish a cross-selling collaboration. An online shop selling garden tools, for example, can collaborate with an online florist to attract its customers. The florist is then getting a

commission from the tool shop for a referred customer and vice versa. The transaction of the commissions, as well as the matching of products, are handled by an intermediary, the cross-selling service provider.

From an IT perspective, the two retailers should be able to connect their product offerings by the adoption of a collaborative cross-selling service. The service should integrate the systems in an ad-hoc manner with the existing product catalog of the two business partners. Furthermore, changes of other services that both partners use (for example the introduction of a new product information management (PIM) system) should not affect the cross-selling activities handled by the cross-selling service.

7.2 Service-oriented collaboration

While Peffers et al. (2007) propose to define the objectives of a solution that rationally infer from the goals, the more rigorous DS process of Verschuren and Hartog (2005) propose a two-step approach. The first step aims at defining a set of requirements. The second step is to carry out a plan evaluation. In the following, we come up with a set of requirements for an architecture that meets the goals described in the previous section. Subsequently, we carry out a plan evaluation which provides grounded reasoning for the contribution of each requirement to the different goals.

7.2.1 Requirements

In the previous sections, we motivated the use of a mediating CDM to achieve the goal of an improved QCC. However, the goal of connecting various services using a common information system has become unpopular as it bears the risk to create a tight coupling between services, which in turn impedes the agility and evolution of individual services (Evans, 2004). To address this issue, we aim at a separation of the evolving components in the system from the constant components. A separation of what evolves from what stays unchanged is usually addressed through the introduction of a platform. By applying the platform design pattern, it should be possible to introduce a common information system across services without affecting their evolvability. In the following, we are elaborating on the requirements of a platform embracing a mediating CDM.

The initial design task for the platform is to find a domain model which is suitable as a CDM. In the context of data warehousing, Prat et al. (2006) differentiate between two approaches for unified data model construction: 1) the data model results from combining the source systems and 2) the model is based on the requirements for its use. Due to the unpredictability of the systems collaborating in the network as well as their data needs, the first method seems less viable. In the case of analytical systems, the goal of using the model is to meet the reporting needs of the decision support system. The equivalent goal in a transactional system is to facilitate the execution of processes that will be supported by the platform and the

collaboration services. Thus, the completeness of the data model with regard to a reference process model of the domain is crucial. However, the goal to support a maximum number of use cases bears the risk to result in a system which becomes too complex to adopt. The previously mentioned EDIFACT standard, for example, can be considered as a system that is difficult to comply to due to its high complexity. Therefore, we consider a good balance between completeness and simplicity as an important factor for model construction.

A means to overcome the conflicting goals of completeness and simplicity is extensibility. In a paper on configurability of cloud solutions Nitu (2009) notes that 'there will be some unique features in the database of each tenant' and proposes a data model which 'meets the most common requirements of the tenants with an option to add the tenant's specific data requirements like adding additional fields to a table'. This is in line with a general trend towards less rigorously structured data repositories. NoSQL databases, for example, make use of the flexible structures instead of a static relational database scheme. The possibility to define ad-hoc structures in addition to the given data model is a means to increase the usability and acceptance of the CDM. Concretely, we propose to stick to the core entities and a limited number of attributes in the definition of the data model and leave the possibility to define additional attributes on each entity to the user of the system.

We can summarize the requirements for the architecture as follows:

- R1: Guaranteeing agility by separating stable from evolving components.
- R2: Define a core domain model suitable for data federation among services.
- R3: Find a balance of completeness and simplicity through extensibility in the domain model.
- R4: Allow intermediaries to offer collaborative services.
- R5: Allow business partners to discover new services and business partners.
- R6: Provide a means of granting intermediaries access to shared resources.

7.2.2 Plan evaluation

According to Verschuren and Hartog (2005) a plan evaluation should be carried out in order to ensure that the requirements are valid sub-goals and contribute to achieving the overall goal of the design. In the following, we describe how the design requirements contribute to each of the design goals. An overview of the mapping is shown in Figure 7.2.

The use of shared resources (R1, R2) has various benefits which contribute to the goal of improved collaboration (G1, G2). First, the reduced efforts in mapping data models of distributed data repositories. Furthermore, data quality increases due to the limited redundancy of data across systems. Finally, a unified resource

repository gives the opportunity for advanced business monitoring and exception handling by defining business rules on the enterprise or network-wide data. Certain events can then raise an exception which can be handled by a service or control tower. Additionally, normal business events can be handled on a more global scale and enable real-time processing instead of scheduled events. This is in line with the current trends in social media where the pattern changes from request and reply to real-time data streaming APIs (Mason, 2011). The separation of agile and stable components (R1) is also required to limit the effects on the overall system when replacing outdated components (G3).

A robust data model is required (R3) in order to guarantee a long-term use for future services (G3) and to support a maximum number of business scenarios (Saltor et al., 1991) such as inter-organizational collaboration services (G2). According to Palmer and Johnston (1996), these collaborations should be handled by intermediaries (R4). The adoption of new services (G1) can be further improved by the provisioning of discovery services (R5) as pointed out by Bichler et al. (1998). Finally, the intermediary needs to get the possibility to gain access to the resources of the business partners (R6) to allow for inter-organizational services with QCC (G1, G2).

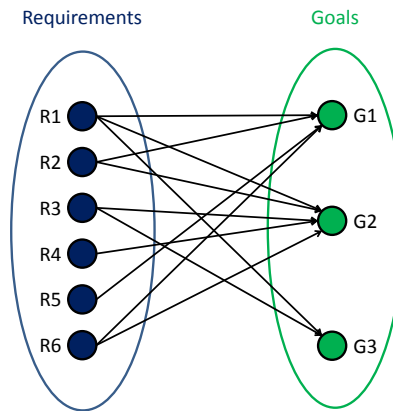


Figure 7.2: Mapping of the design requirements and design goals

7.3 Cross-selling architecture

In design science, the specification of the design artifact is based on the requirements and forms the basis for its implementation (Verschuren and Hartog, 2005). In our case, the functional specification is the architecture that fulfills the requirements to achieve the goal of improved QCC among business partners. As our architecture contains domain specific elements, and to make the architecture more

tangible, we use the e-commerce cross-selling case within the architectural description. Again, the ArchiMate notation was used to present an architectural model (cf. Appendix A).

7.3.1 Ecosystem

A couple of approaches and architectures exist that incorporate a similar mechanism of collaboration. Kleeberg et al. (2014) present various cloud integration scenarios. Among the more visionary solutions, the authors mention ‘EAI-in-the-cloud’ which ‘leads to a situation, where more enterprise resources are being exposed to off-premises access or moved to the cloud. This situation opens novel opportunities for supporting B2B-transactions. [...] A straightforward example is cross-enterprise data sharing by means of a common cloud-based data store.’ Włodarczyk et al. (2009) propagate the idea of an industrial cloud which ‘should be controlled by an organization in form of e.g. special interest group’. However, no architecture or solution is presented which goes beyond the high-level vision of such a solution.

In Figure 7.3 the high-level ecosystem of the architecture contains the three main roles, namely a platform provider, a (platform) client which implements and offers platform-based business services, and finally a (service) user. An example of a business service would be the cross-selling service we discussed earlier. In that case, the two actors of tool shop and flower store represent the users. The platform provider role can either be taken by an independent party or one of the service providers. It is thinkable that one of the dominant marketplace service providers such as Amazon or eBay offer such a platform and allow other service providers to plug in their services. A strong dominance and trust in the e-commerce market may attract more service providers and will probably have a higher chance to achieve a critical mass of services and users (Evans and Schmalensee, 2010). In the model, we can further see that, according to the mediated approach discussed earlier, the platform contains the CDM containing, for example, product information. Following the reference model for federated systems (Sheth and Larson, 1990; Busse et al., 1999), the business service makes use of the platform’s CDM and only controls the component data itself. In our case, the matchmaking data required for cross-selling is an example for component data.

7.3.2 Services and components

Figure 7.4 shows the components and services of the cross-selling service architecture. In this case, the platform has three users, the two online shops, and a cross-selling service provider. The cross-selling service implements a platform client that facilitates cross-selling transactions. For that purpose, both shops grant the client access to their product resources. Furthermore, the cross-selling service is implemented as a web application which is based on the platform and allows both shops to match their own products with the products of their partners. A flower shop

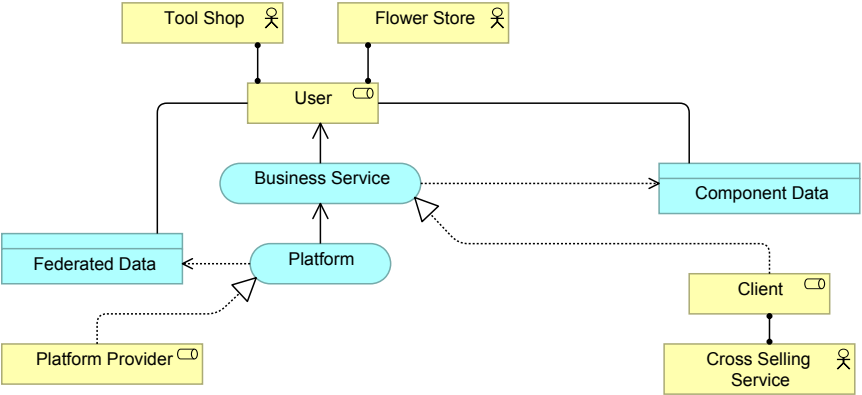


Figure 7.3: Architecture ecosystem

can, for example, match rose trees with pruners and mowers with grass seeds. If the shop for garden tools integrates an UI widget provided by the cross-selling service into the online shop, it will automatically show a reference to the grass seeds when a customer picks the mower.

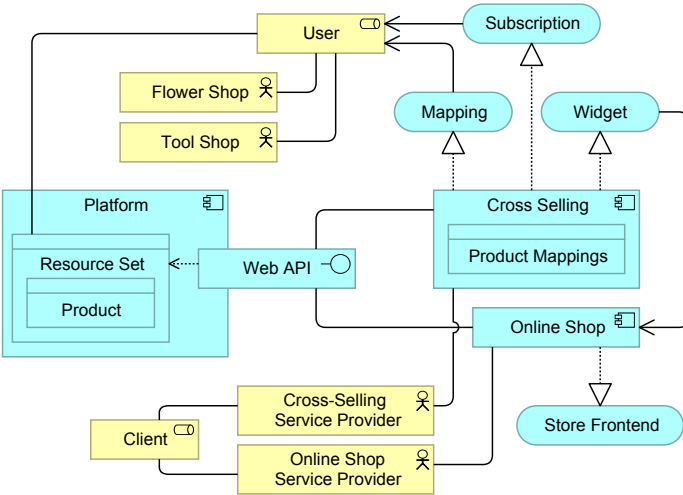


Figure 7.4: Cross-selling components and services

7.4 Product evaluation

The instantiation of the design allows to evaluating the feasibility and the effectiveness of the structural specifications. The instantiation can be a realization (immaterial artifact) or a materialization, can meet all or only parts of the specifications, or be a mere mock-up (Verschuren and Hartog, 2005). In our case, we aim at a complete realization of the platform in order to be able to evaluate its ability to support the implementation of services that help to achieve QCC goal. In the previous sections, we have outlined the requirements, structural specification, and the implementation of a collaborative platform. During plan evaluation, we have already discussed how the requirements of the platform match the overall goals related to improved quick connect capability among business network partners. The DSRM by Peffers et al. (2007) uses the instantiation of the artifact to observe to what extent it supports the objectives. Similarly, Verschuren and Hartog (2005) ask for a product evaluation in order to assess the goal achievement. To evaluate the platform's capability to allow QCC among business partners, we used the cross-selling business case and implemented the cross-selling service and verify if it allows the retailers to enter into a dedicated business relationship without prior agreement. In the remainder of this section, we first present the prototype of the platform-based cross-selling service and then assess its effectiveness towards the three goals G1-G3.

The cross-selling service implements a platform client that facilitates cross-selling transactions. For that purpose, both shops grant the client access to their product, orders, and customer resources. The cross-selling service further offers a web application which is based on the shared data platform and allows both shops to match their own products with the products of their partners. The flower shop can, for example, match rose trees with pruners and mowers with grass seeds (Figure 7.5). Furthermore, an UI widget is provided by the cross-selling service which both users can integrate into their online shop (Figure 7.6). The widget will automatically show a reference to the grass seeds when a customer picks the mower. If the customer selects the product from the widgets, a cross-selling transaction is automatically created by the service.

The cross-selling service is used to evaluate the platform with regard to the three goals G1-G3. If both online shops rely on the platform to handle their resources, such as products and orders, the service provider for cross-selling does not need to map the data models as they share the same database schema. The same applies for the reduced efforts in engineering the integration. As both online shops not only share the same schema but also the physical database, no further implementation work is required to access the data of both organizations within the same functional component (e.g. the product referral UI widget).

The prototype was constructed by reflecting four different services of an e-commerce process. All the data model entities required for the cross-selling use case, namely products, orders, and customers are reflected in the CDM. This indicates, that the construction of a complete unified data model was successful with reasonable effort. The simplicity of the data model was achieved by limiting the

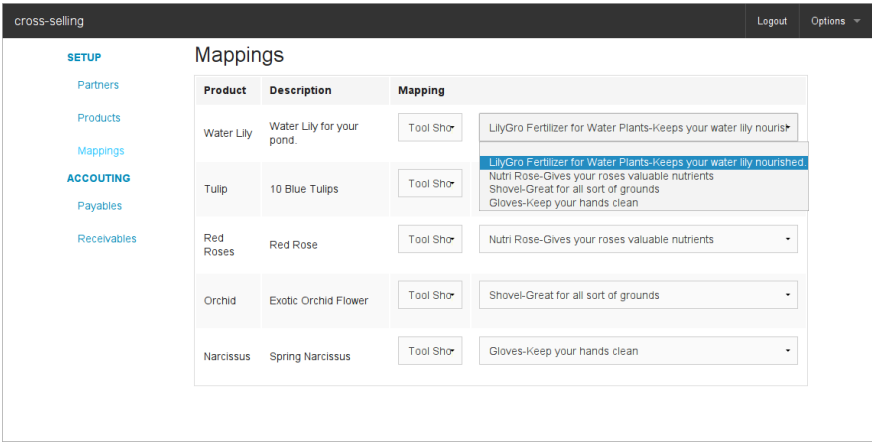


Figure 7.5: Platform client service

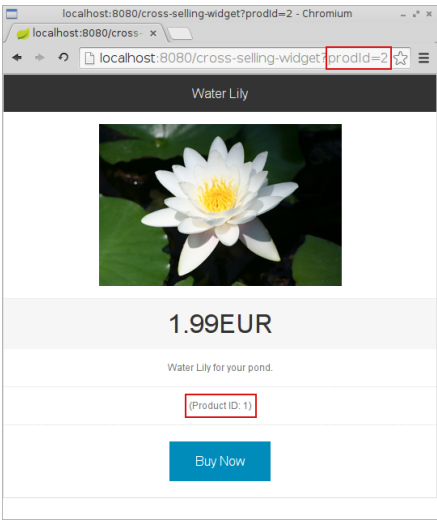


Figure 7.6: Platform client widget

attributes of the entities. Through the mechanism of extension, the cross-selling, the client is able to extend the product entity with cross-selling commission rates. The adoption of the service through the service users can be achieved with limited efforts (G1) due to the canonical data. However, other than in earlier federated architectures, the functionality of the cross-selling service can be remote from the CDM and thus, offered for example as a cloud service. It is thus easier to adopt then, for example, an additional module or plug-in for an existing system.

The benefit of business monitoring and exception handling relies on the unified

data model and the possibility to analyze real-time data and deliver reports in case of particular events. The cross-selling service can make use of this situation. For example, the cross-selling service can give the retailers insights into currently top-selling goods of their partners. During setup of matching products, the flower shop can see which mower is selling best in the last hours and match a special offer of grass seeds to that particular product. Such complex inter-organizational functionality (G2) can be built on the fly as all required data is available from the canonical data component. Another potential benefit of the CDM is the increased data quality. In solutions that rely on data from multiple repositories, the replication and synchronization of those sources are often a cause for poor data quality that grows over time when single updates are not populated properly into all the systems. As the cross-selling service relies on a single data source which also happens to be the base for other operational systems of the two shops, no data replication is required. Thus, there is no risk of decreasing data quality inside the cross-selling component.

Finally, the functionality of mission critical services and added value services, such as the cross-selling service, are kept separated in the architecture. Thus, it is possible to disconnect the cross-selling service without side effects on other functionality (G3). The tight coupling between services, that is often linked to shared databases, cannot be confirmed in this case as none of the services is owning the core data model. The fact of having to rely on the given core data model might, however, affect the flexibility of the service provider, due to the constraints implied by the platform.

7.5 Conclusion

In this chapter, we investigated on state of the art in collaboration architectures and pointed out the shortcomings in the current approaches. We argued that our platform with its federated architecture embracing a CDM can significantly improve the collaboration among systems and proposed an architecture for such a platform. The development of a platform-based service has shown that the expected goals can be achieved. In the era of web APIs and cloud computing, it is worth revisiting the debate on federated information system as it might have even more favorable outcomes. While this development is happening in other domains, such in content management systems through so-called content APIs hosted separately from websites, we don't see similar approaches in enterprise architectures.

According to Verschuren and Hartog (2005) we outline some of the 'favorable outcomes that have nothing to do with the design goal'. One benefit of the platform architecture is the facilitation of a business event which reports data updates in real time as shown in Chapter 5. The business event system can significantly change the way how individual services communicate with each other. Usually, a service has to be triggered in order to execute its behavior. If the cross-selling service wants to calculate the commissions that shops owe each other it needs to reflect all the orders since the previous billing. In the current architectures, the service will schedule the

request of this information and recalculate the figures once or twice per day. The platform architecture, however, can provide a business event system which allows services to subscribe to specific events. Every time a new order is created through cross-selling, an event will get triggered. The cross-selling service will reflect the update and recalculate the commissions in real-time. In many other cases, this mechanism can improve the accuracy of information regarding recent changes.

Like every IT solution the proposed architecture has some limitations and can only solve the challenges of system development to a certain extent (Brooks, 1987). Specifically, the service platform is putting a strong focus on system interoperability, which comes at the cost of major constraints for service development. In current systems, the ownership of the database schema is with the application component which offers high flexibility when designing and implementing functionality. Giving up this ownership to the platform will lead to more restrictions when building new services. Furthermore, the proposed approach requires the business network partners' strong commitment towards a particular platform although it might lead to higher risks.

The focus of this work lies in the platform architecture and its benefits. Future research has to focus on the services which have to be built with respect to the platform in order to achieve the benefits. This includes technical considerations, such as the support of application frameworks, the additional complexity of building distributed solutions, as well as side effects of the architecture, such as the impact on application performance. Furthermore, organizational factors of platform adoption have to be discussed. Besides the general skepticism of organizations to move business resources to cloud platforms, the willingness to share resources within a business network has to be investigated.

Conclusions

In the previous chapters, we have motivated and carried out the design and implementation of a platform for pluggable e-commerce services. To come up with a scientifically sound design that meets the practical needs, we made use of the DSRM proposed by (Peffer et al., 2007). Gregor and Jones (2007) provides a framework to express the outcome of design science research in IS and comes up with eight components a design contains. In the following section, we make use of this framework to recapitulate and discuss the design presented in this work and follow up on the research goals and research questions we established in Chapter 1. Subsequently, in Section 8.2 we outline the limitations of this work and motivate further research on the topic.

8.1 Anatomy of the proposed design

The framework of Gregor and Jones (2007) consists of eight components for expressing a design. In this section, we reflect our work with regard to these components. The goal is not to reformulate the entire design and to establish a design theory as Gregor (2006) propose. The purpose is rather to provide the reader with a structured overview of the contribution of this work and to show that it contains all the required components of a scientifically sound design. The *constructs* and *principles of form and function* components have been summarized in one section for narrative purposes. The *principles of implementation* component has been disregarded in this work. The component refers to the design process as one of the two pillars of a design theory proposed by Walls et al. (1992). However, we follow the argumentation of Gregor (2006) that the instantiation of the design as a proof-of-concept does not require the specification of a method. In this work, we followed this thinking and did not come up with a method to implement the platform architecture.

Purpose and scope

The purpose of our design is to provide an architecture of a pluggable service platform which facilitates the adoption of e-commerce functionality. The concrete requirements for a facilitated adoption have been presented in Chapter 3 in the form of a quality model.

The motivation for presenting the design was to allow small and medium size retailers to adopt novel functionality with limited technical knowledge (cf. RG3a Chapter 1). The second motivation is to leverage the potential of pluggable service for the use of collaboration among e-commerce partners in a quick-connect fashion (cf. research goal RG3b Chapter 1).

Justificatory knowledge

To come up with a design that meets its purpose and scope we rely on a variety of existing theories and approaches. Beside the instantiation of the design for the purpose of its evaluation, justificatory knowledge provides the reasoning that the design contributes towards the goals.

From theory in service-oriented computing, we know that, in general, IT functionality is easier to consume if it is provided as a service (Papazoglou, 2003). Thus, we followed a service-oriented approach in this work which relies on the principles of service-oriented design (Erl, 2008).

Furthermore, we took a model driven approach to developing the design and made use of the ArchiMate language (The Open Group, 2016). It allows to express, compare, and assess the consequences of design decisions. The practice of enterprise architecture modeling enables the analysis of effects within and across the three layers of the information system. A reference model (Fettke and Loos, 2007) was used as a base for the e-commerce architecture which is based on state-of-the-art process models in the field (Becker and Schutte, 2007; Frank and Lange, 2004; Burt and Sparks, 2003). We applied the method of systematic literature study (Kitchenham, 2004) to enrich the business layer with application and technology layer artifacts that have been discussed in the academic literature.

The objective for the design is the pluggability model which has been constructed with reference to existing quality models (McCall et al., 1977; Boehm, 1988; Dromey, 1996) and quality model construction methods (Ortega et al., 2003). The model borrows aspects of the agility notion as covered in (Lankhorst et al., 2012). The platform model was chosen as it bears the potential to enhance the agility of a system (Sriram et al., 2014; van Heck and Vervest, 2007).

Beside the theoretical underpinnings, we rely on working solutions from practice. The notion of pluggability refers to the popular concept of plug and play hardware and plugins for software components. The platform has been enhanced in an iterative fashion to reflect current trends in social media such as the OAuth protocol (Hardt, 2012) which inspired the use of shared resource access and authorization across services.

Constructs and principles of form and function

The description of constructs and principles of form and function are reflected in the architectural models. The structural concepts of the models (services, components, functions, etc.) can be considered as the constructs of the design. The

principles of form and functions are primarily represented by the relationships between the structural concepts. We provide a short summary of the most important structural concepts and relationships.

The core concepts of the design are the e-commerce services, the platform component, as well as the three main actors (service provider, client, and user) and their relationships which have been established in Chapter 1. In Chapter 2 we focused on the user (retailer) and outlined its ten required business functions, the application components, and technical artifacts in the reference model. Chapter 4 focuses on the service provider and the common functionality of cloud service platforms. The combination of both the retailer's and the platform provider's architecture form the initial pluggable e-commerce service platform. It was extended iteratively in the subsequent four chapters. The first extension is the need to manage the resource access and authorization for services. Next, we added web-based business event streams to the platform API and introduced the principle of user hooks for public resource updates. In Chapter 5, we provided more details on the required domain model and the metamodel that reflects the needs for access policies, scopes, and data model extensions.

Testable propositions

In Chapter 1 several research goals and related research question have been established. Various research artifacts and design decisions have been presented throughout this work to address the issue. They can be reformulated as testable design proposition (Gregor, 2006). In the following, we present the design proposition and relate them to the initial research questions.

In order to respond to **SQ1** we carried out a systematic literature review (cf. Chapter 2). The findings indicate that *the state of the art in e-commerce reference models does not reflect architectural aspects. Thus, a novel architectural reference model contributes to the decomposition of e-commerce systems into modular IT services.*

Regarding **SQ2** we presented a quality model in Chapter 3. Based on the quality model a pluggability instrument has been outlined which has been evaluated together with experts in the field. Based on our findings we confirm that *the presented quality criteria of pluggability provide a sound framework for evaluating the ease of adoption of a service.*

In Chapter 4 we investigated current e-commerce platforms to respond to **SQ3**. Findings indicate that *existing e-commerce architectures do not reflect the needs for pluggable services. As a consequence, existing platforms do not provide the best possible support for pluggable services.*

Various iterations of the platform design have been outlined to respond to the design research question **SQ4**. In Chapter 4 we proposed that *a domain specific service platform including a common e-commerce model allows to pre-integrate e-commerce services. Thus, facilitating the provisioning of services with improved pluggability.* Furthermore, we argued that *the extendibility of the platform model im-*

proves the flexibility for service providers. Thus, allowing to improve the versatility of pre-integrated, pluggable services and designed and tested the extensibility features built into the metamodel. In Chapter 5 we showed how (web-based) streaming of business events facilitates the decoupling of cloud services, operating on a common data model. In addition, we introduced user hooks on public resource updates to facilitate the interoperation among such decoupled services.

In response to **SQ5** we presented the cross-selling service in Chapter 7 and showed that *collaborative features can be provided as pluggable collaboration services in the same fashion as IT services, using the proposed platform architecture.* The case has furthermore shown that *such pluggable collaboration service can help to improve the quick connect capability between network partners.*

Artifact mutability

Gregor (2006) proposes that design artifacts are often mutable by nature and underlie a constant change to adapt to the needs. We argue that the design presented in this work has three aspects of mutability which we outline in the following.

1. In Chapter 4 we presented a platform reference architecture with the service framework, process framework, and the data management component at its core. We have stated that the service framework and process framework components are present in many available platform products today. As a consequence, we have focused on the domain specific data management across services and came up with specific needs for such a component during the design iterations. However, additional changes in the service framework and the process framework might be beneficial to support the new data management capabilities better. Furthermore, it is thinkable to evolve the platform beyond the three mentioned components.
2. In this work, we proposed a variety of e-commerce services to demonstrate and evaluate the application of the platform design. More precisely, the types of services we implemented include transactional, analytical, and collaboration services. It should be mentioned that the platform is not limited to the specific services and types of services that have been presented. The platform should be usable for other e-commerce services such as real world services for logistics which integrate IT wise with the platform.
3. As indicated in Chapter 1 many of the presented concepts, such as the data management component and the contained functionality, can be applicable for utterly different domains. A pluggable service platform for finance or health care is thinkable, following the same architectural building blocks. While some aspects of the platform architecture may require adjustment to the individual requirements of such domains, the core concepts, such as shared resource access and resource access authorization, can be reused.

Expository instantiation

The design artifact has been implemented in order to demonstrate its application which we call the design product. We have seen various alterations and additions to the design throughout the various design cycles presented in this work. The platform prototype has furthermore been used as a means to demonstrate the work to the research consortium partners. Precisely, the implementation of the design product contains the following building blocks:

- The *state-of-the-art platform part* is based on the *mulesoft cloud hub* (<https://www.mulesoft.com/>). It uses the service framework and process framework features. It was implemented in order to evaluate the pluggability of a service-oriented return authorization process (Chapter 4).
- The *collaborative data component* is implemented on top of a PaaS offering from heroku (<https://www.heroku.com/>). At its core, it contains a postgres SQL database, a python-based application layer, and a git-based deployment component. The data model and metamodel have been implemented in python using the SQLAlchemy object-relational mapping framework (<http://www.sqlalchemy.org/>).
- The *platform web interfaces* are based the flask web framework using flask-restless for the API part (<http://flask.pocoo.org/>). The administration interface of the platform has been built on Jinja2 and AngularJS for server and client web templating and ink (<http://ink.sapo.pt>) for the layout.
- The *resource authorization* is based on the OAuth protocol and has been built using the flask-oauthlib library. A comprehensive list of all used libraries for the platform can be found in Appendix D.1.

In addition to the platform prototype, it was necessary to design and implement various of business scenarios for supported e-commerce services. The services have been implemented to demonstrate the consequences for service providers and service users when using the platform and to assess the pluggability of the services. Some of the services have been implemented using the same technology stack as the platform itself. Other services such as the tax compliance service have been implemented using a Java stack to make use of the extensible features of the spring framework with regard to building REST clients. Another benefit of the service implementation was the ability to come up with working solutions in the course of collaboration with the consortium members of the research project.

8.2 Limitations and future research

Despite the insights and findings in this work, there are a few limitations that we were not able to overcome. In the following, we provide an overview of the shortcomings and point out opportunities for future research.

In the beginning of the research project, there was no plan to elaborate on the service adoption lifecycle and the pluggability model. However, as we did not encounter existing models and instruments that reflect the principles of pluggability, we had to come up with a dedicated model that helps us to formulate the objectives for the design and to evaluate the architecture. The development of a novel quality model is a considerable undertaking. Our goal was to progress this task to the extent that it allows us to compare the pluggability with and without the platform. Thus, the current state of the quality model is merely a byproduct and additional work could be carried out to make it stronger. Especially the lifecycle model would tolerate more grounding in the academic literature and could be subject to a systematic study of related lifecycle models.

The architectural model focuses on the features of the platform which are most relevant for achieving the goal of improved pluggability. Other components are already available in existing platforms and have not been discussed in detail. Accordingly, the implementation of the platform prototype is focusing on the parts that are not present in existing platforms. Furthermore, during prototyping, we focused on the aspects that are critical to proving the feasibility of the novel concepts. Completing the prototype to a working product would be necessary to test the design in practice. However, the completion of the prototype would consume considerable resources in terms of software engineering and is outside the scope of the study.

As a consequence, the prototype has not been tested in a real-world scenario. The application of the action design science paradigm by Sein et al. (2011) would be a beneficial addition to what has been achieved so far. The testing of the design in a real-world setting could reveal additional shortcomings of the design. Besides the completion of the entire platform, it requires a retailer that commits to the platform as well as a sufficient number of provided services. The presented design goes beyond the implementation of e-commerce functionality but aims for a higher pluggability of the services, which requires a strong commitment to the platform architecture from the involved actors. Furthermore, the current design is not open enough to cope with legacy applications and is therefore not suitable for testing single platform-based services. A concept for a migration path from existing e-commerce landscapes to the platform would be beneficial. The proposition of services that help to integrate existing e-commerce components with the platform is thinkable.

The services that have been presented in this work consist of a small set of examples that have been compiled in collaboration with the consortium members of the research project. It covers services that are critical to overcoming issues in cross-border retailing and omnichannel retailing and are lacking in the current e-commerce platforms. They do not cover the most basic services which are part of the reference architecture presented in Chapter 2. However, the goal of the platform is not only to provide innovative services but to allow small and medium size retailers profit from a holistic set of services that enable them to operate their entire business. It should allow them to keep pace with offering state-of-the-art services

that major retailers can only stem with extensive IT budgets. As a consequence, basic transactional services have to become part of the platform-based service offering as well.

To stay competitive in the e-commerce market, retailers have to bind customers across different channels, predict their shopping habits, and adopt services to the daily life of the consumer. Creative solutions and new services are also required for order fulfillment, to meet the evolving expectation of buyers. Appropriate e-commerce architectures constantly evolve and adopt new services to collaborate with partners and to handle the diverse needs in the increasingly global marketplaces. If the modular approach presented in this work will not be adopted in some way only large players with extensive budgets to deliver innovative end-to-end solution will be able to survive. Amazon, for example, recently came up with a bunch of new services to make their offering stronger. Their own delivery fleet allows strong integration and better order fulfillment performance. Their own offline stores extend their portfolio to get a stronger presence in the market. Their own hardware solutions like e-book readers and order buttons on household items enable ubiquitous commerce that transforms day-to-day shopping routines. Other retailers can only respond to such offerings in a collaborative effort, relying on shared services. Despite the fundamental paradigm shift of the e-commerce architecture presented in this work and the initial effort and commitment it takes to migrate to the new model, we argue that there are no future-proof alternatives at the moment.

Appendix A

ArchiMate Metamodel

The ArchMate language is used throughout the work in order to visualize different architectural aspects of the platform design. The purpose of the ArchiMate language is to provide an instrument for analyzing and describing enterprise architectures. Figure A.1 shows a simplified version of the ArchiMate core metamodel. The language distinguishes between three layers: the business layer, the application layer, and the technology layer. Within each layer, ArchiMate considers structural, behavioral, and informational aspects. It also identifies relationships between and within the layers. Each layer is a means to provide services to the next higher level, while the highest level provides business services to the customer. For a full description of the language, we refer to The Open Group (2016).

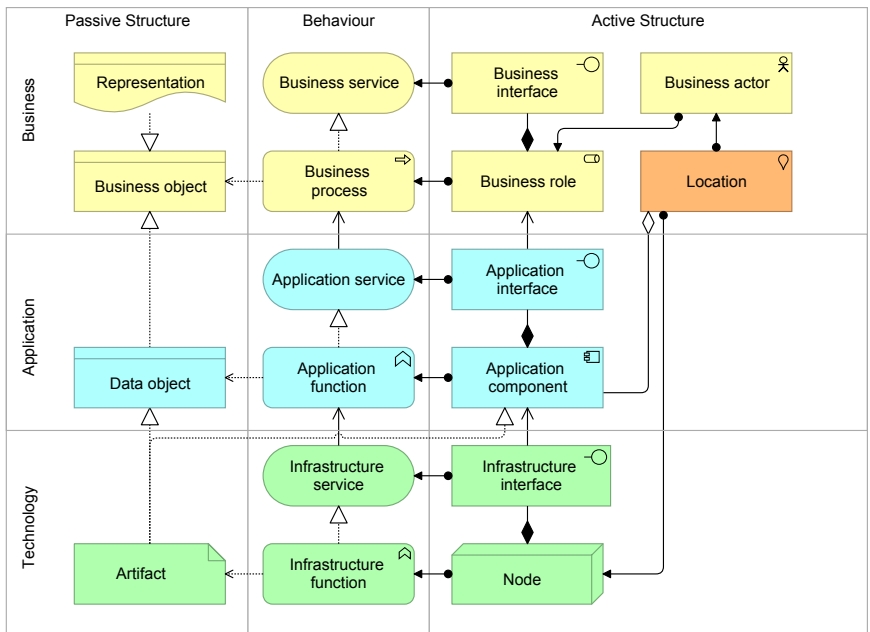


Figure A.1: Simplified ArchiMate metamodel

Appendix B

E-Commerce Business Model

Throughout this book, we discuss the requirements of innovative and easy to adopt e-commerce services. E-commerce is a broad term that refers to commercial transactions, supported by information technology. We apply the term e-commerce in a narrower sense, from an online retail perspective. To provide a better understanding of an e-commerce company from that perspective, a business model canvas is presented in Figure B.1. It is based on the work from Osterwalder and Pigneur (2010) which provides a framework for describing business models.

Key Partners Manufacturer and suppliers Warehouse and logistic service providers Internet service providers Platform service providers Software service providers	Key Activities Sales and marketing Supply chain management Inventory management Information system management Key Resources Supply chain network	Value Propositions Price Variety Convenience Speed Personalized services or goods	Customer Relationships Self service Helpdesk Channels Brick and mortar stores Online stores in web and mobile application	Customer Segments Niche market
Cost Structure Cost of goods Payroll Rent and utilities Logistics and warehousing Service subscription Adverstiment			Revenue Streams Sales of goods	

Figure B.1: Business model canvas for online retail

Appendix C

Pluggability Instrument

The quality characteristic of pluggability, conceptualized in Chapter 3, consists of six quality criteria. For each of these theoretical constructs, an indicator that allows to assess the criteria is defined. In Table C.1 the attributes that provide an anchor for the different values (low to high) of each indicator are shown. Face validity of the instrument was realized with a group of experts in the field of service platforms, consisting of both scholars and practitioners.

Table C.1: Attributes of the indicators for pluggability

	Low	Medium	High
EOP	Detailed information on the service is only available through individual contact with the service provider.	Documentation and information about the service can be requested and is made available according to a transparent process.	All required information including documentation, pricing and demos are openly available.
EOD	Technical expertise such as development or scripting is needed to make the service operable.	The deployment does not require any technical expertise but complex setup and configuration.	The service can be used straight away through subscription.
EOA	The service can hardly be adapted for use cases that have not been specified by the service provider.	The service can be adapted to any use case scenario but needs technical expertise to do so.	The service can be adapted to any possible use case through configuration or setup.
EOI	The integration of the service into the landscape requires coding or scripting.	The integration of the service into the landscape requires configuration or setup.	The service is automatically integrated into the landscape and requires no further action after deployment.
EOO	Monitoring, maintenance, and customer service have to be carried out by the service user.	Monitoring, maintenance, and customer service are partly handled by the service provider.	Monitoring, maintenance, and customer service are entirely handled by the service provider.
EOE	Exchanging the service impacts other services and requires development or scripting.	Exchanging the service impacts other services but can be handled through reconfiguration or setup.	Exchanging the service does not impact any other service.

Platform Prototype

D.1 Model and interface implementation

The platform prototype has been implemented using the Python programming language and a variety of open source frameworks and libraries. The python requirements file in Listing D.1 contains the python packages that are retrieved during the deployment using pip (<https://pip.pypa.io>).

Listing D.2 contains an example for an entity of the platform model (product) and three entities of the metamodel (user, client, resource). The model layer has been implemented using the SQLAlchemy ORM framework. User is the basic entity for any actor, i.e. a client actor is also a user. The client class has additional OAuth related attributes such as the redirect URIs or accessible resources (realms). Resource is the basic metamodel entity for any entity of the model, e.g. for every product, a resource of the type product will be instantiated.

Listing D.1: Platform prototype libraries

```
Flask==0.10.1
Flask-Admin==1.0.8
Flask-Login==0.2.11
Flask-OAuthlib==0.9.2
Flask-Restless==0.17.0
Flask-SQLAlchemy==1.0
Jinja2==2.8
MarkupSafe==0.23
SQLAlchemy==0.9.6
WTForms==2.0.1
Werkzeug==0.11.10
argparse==1.2.1
unicorn==19.0.0
itsdangerous==0.24
mimerender==0.5.4
oauthlib==1.1.2
psycogp2==2.5.3
python-dateutil==2.2
python-mimeparse==0.1.4
requests==2.10.0
requests-oauthlib==0.6.1
six==1.7.3
wsgiref==0.1.2
```

Model extensions and access privileges are handled on resource level.

Listing D.2: User, client, and resource entities (SQLAlchemy)

```

class User(db.Model):
    user_id = db.Column(db.Integer, primary_key=True)
    user_name = db.Column(db.String(40), unique=True)
    password = db.Column(db.String(20))

class Client(db.Model):
    user_id = db.Column(db.Integer, db.ForeignKey('user.user_id'))
    user = db.relationship('User')
    application_name = db.Column(db.String(40))
    description_short = db.Column(db.String(100))
    description_long = db.Column(db.String(500))
    url = db.Column(db.String(40))
    client_id = db.Column(db.String(40), primary_key=True)
    client_secret = db.Column(db.String(55), unique=True, index=True)
    ,
                                nullable=False)

    # public or confidential
    is_confidential = db.Column(db.Boolean)
    _redirect_uris = db.Column(db.Text)
    _realms = db.Column(db.Text)

    @property
    def client_type(self):
        if self.is_confidential:
            return 'confidential'
        return 'public'

    @property
    def redirect_uris(self):
        if self._redirect_uris:
            return self._redirect_uris.split()
        return []

    @property
    def default_redirect_uri(self):
        return self.redirect_uris[0]

    @property
    def default_scopes(self):
        if self._realms:
            return self._realms.split()
        return []

    def __init__(self, user, application_name, description_short,
                 _redirect_uris, _realms, client_id=None, client_secret=None):
        self.user = user
        self.application_name = application_name
        if client_id:
            self.client_id = client_id

```

```

    else:
        self.client_id = _generate_random_token(20)

    if client_secret:
        self.client_secret = client_secret
    else:
        self.client_secret = _generate_random_token(20)
    self.description_short = description_short
    self._redirect_uris = _redirect_uris
    self._realms = _realms

class Resource(db.Model):
    resource_id = db.Column(db.Integer, primary_key=True)
    resource_type_id = db.Column(db.Integer, db.ForeignKey('
        resource_type.resource_type_id'))
    resource_type = db.relationship("ResourceType")
    owner_user_id = db.Column(db.Integer, db.ForeignKey('user.
        user_id'))
    owner = db.relationship('User')

    def __init__(self, resource_type_name, owner):
        rt = db.session.query(ResourceType).filter_by(
            resource_type_name=resource_type_name).first()
        if rt is not None:
            self.resource_type = rt
        else:
            self.resource_type = ResourceType(resource_type_name)
        self.owner = owner

class Product(db.Model):
    product_id = db.Column(db.Integer, primary_key=True)
    product_nr = db.Column(db.String)
    product_category_id = db.Column(db.Integer, db.ForeignKey('
        product_category.product_category_id'))
    product_category = db.relationship('ProductCategory')
    product_name = db.Column(db.Unicode)
    description = db.Column(db.Unicode)
    dimension_length = db.Column(db.Float)
    dimension_width = db.Column(db.Float)
    dimension_height = db.Column(db.Float)
    weight = db.Column(db.Float)
    size = db.Column(db.Float)
    color = db.Column(db.String)

    brand_id = db.Column(db.Integer, db.ForeignKey('brand.brand_id')
        )
    brand = db.relationship('Brand')
    resource_id = db.Column(db.Integer, db.ForeignKey('resource.
        resource_id'))
    resource = db.relationship('Resource')

```

Platform access is mostly occurring by clients through the API. Listing D.3 shows the use of Flask-Restless to implement the product request API. Preprocessors are realized that verify access and results based on the configured permissions.

Listing D.3: Authorization and filtering of resources for product API using Flask-Restless and OAuthlib

```
def _filter_products(search_params=None, **kw):
    # token verification
    valid, req = oauth.verify_request(['product'])
    if not valid:
        raise ProcessingException(description='Unauthorized', code=401)
    # resources filter
    uid = req.access_token.user.user_id
    if search_params is None:
        return
    if 'filters' not in search_params:
        search_params['filters'] = []
    filt = dict(name='resource__owner_user_id', op='has', val=uid)
    search_params['filters'].append(filt)

prodcut_blueprint = manager.create_api(Product,
                                     methods=['GET'],
                                     preprocessors=dict(GET_MANY=[
                                         _filter_products]))
manager.create_api(Price, methods=['GET'])
```

D.2 Administration interface

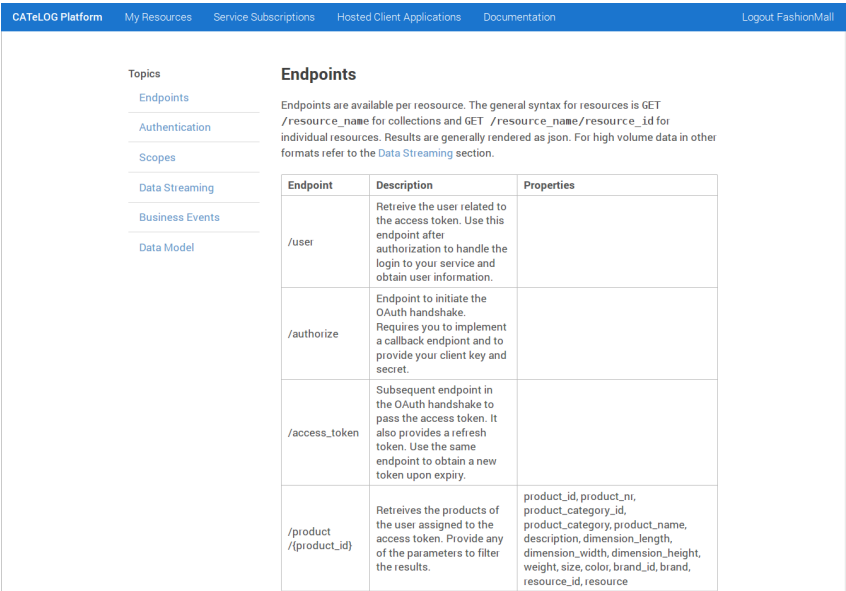


Figure D.1: Platform API documentation

CATeLOG Platform
My Resources
Service Subscriptions
Hosted Client Applications
Documentation
Logout FashionMall

Service Subscriptions

These are the services you subscribed to. You can see the service client and the resource access that was granted. Click on **Manage** to see the details of a subscription, change the subscription, or unsubscribe.

Name	Description	Client	Scopes	
CATeLOG Store	A pluggable store for short time to market ventures.	Cross	orders product	Manage

Available Services

This is a list of services that might be interesting for you. Click on **Details** to get more information about the service, service provider, pricing and more.

Name	Description	Client	Scopes	
Awesome Shop	A shop frontend to get you into the business.	E-Commerce Services Inc.	Orders, Customers, Products	Details
Bettershop	Still using Awesome Shop? Time for a change!	Next Generation Shops Inc.	Orders, Customers, Products	Details
Bookkeeping	An accounting app to keep track of the cash flow and to be compliant.	Bookkeepers Inc.	Orders, Products	Details
Marketplace invader	No one comes to your shop? Sell your stuff Amazon and eBay!	Market Checkers Inc.	Orders, Products	Details
X Border Compliance	Want to sell abroad? Not sure about regulations? We are!	pincvision	Orders, Customers	Details
Uber-PIM	Most advanced PIM service	E-Commerce Service Hackers	Products	Details
Adidas Shoe DB	Adidas sport shoe database. Allows you to select and import from the adidas shoe collection.	Adidas AG	Products	Details
Cross-Selling	Colaborate with partners and make your store take off in no time.	XBuis. Inc.	Products	Details
ECAF SaaS	Probably the most pluggable e-commerce analytics and forecasting service out there.	VU Amsterdam	Products, Orders	Details

Figure D.2: Platform service subscription

CATeLOG Platform
My Resources
Service Subscriptions
Hosted Client Applications
Documentation
Logout FashionMall

Hosted Services

Here you can find the services you are offering as client. Click on **Manage** to see details for each service, to get information about your customers and change the setup of the service.

Service Name	Scopes	Subscriptions	
GoB Annual Statements (German Legislation)	Orders, Products	3	Manage

Figure D.3: Platform hosted service administration

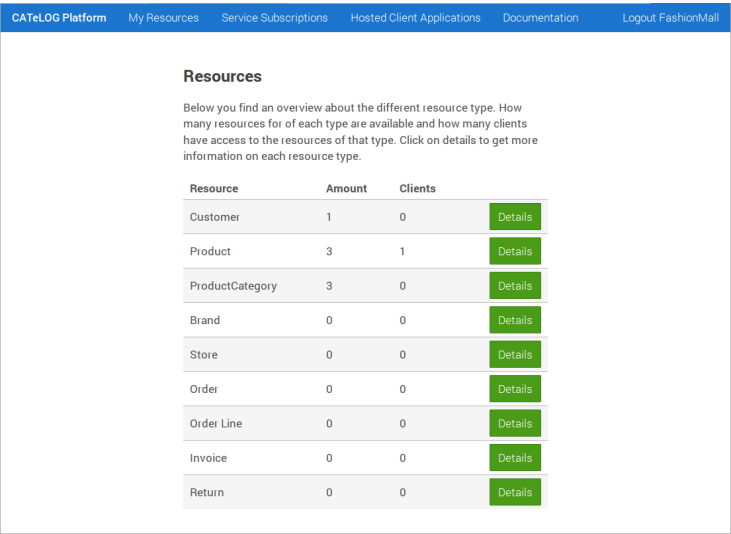


Figure D.4: Platform resource administration overview

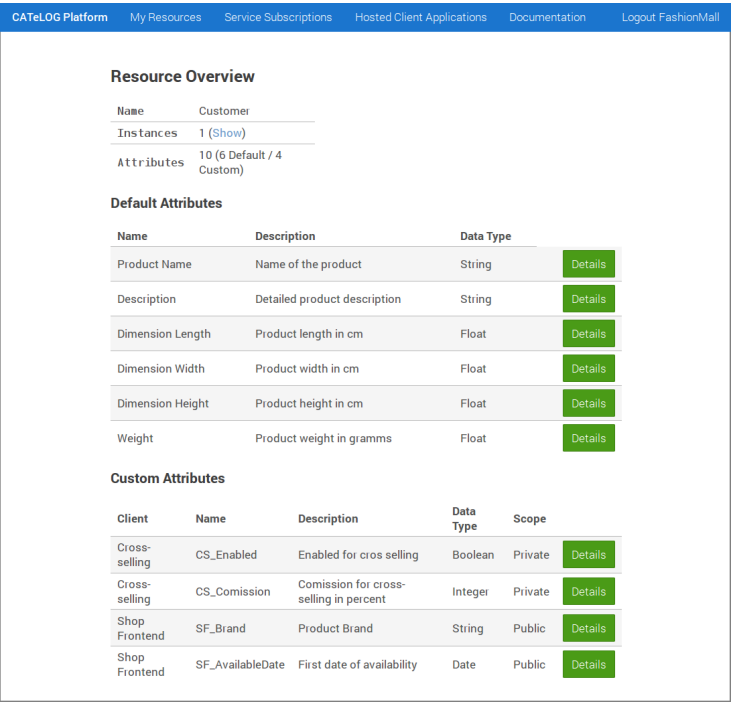


Figure D.5: Platform customer resource administration

D.3 Platform client

The e-commerce services (platform clients) can be implemented on any technology that support access to the web API. Listing D.4 shows the product model of a client that is instantiated during the unmarshalling of the API message. It is a plain old Java object (POJO) as part of a Spring application (<https://spring.io/>). The Gradle configuration for the client application with the libraries used is presented in Listing D.5. A simple example of accessing the platform API using Spring's Rest-Template is shown in Listing D.6.

Listing D.4: Product model of a platform client using Spring POJO

```
package app.model;

public class Product {
    private String product_id;
    private String product_category_id;
    private ProductCategory product_category;
    private String product_name;
    private String description;
    private String dimension_length;
    private String dimension_width;
    private String dimension_height;
    private String weight;
    private String resource_id;
    private Resource resource;
    private Price[] prices;
    //...
}
```

Listing D.5: Build file for platform client using Gradle

```
buildscript {
    repositories {
        mavenCentral()
    }
    dependencies {
        classpath("org.springframework.boot:spring-boot-gradle-
            plugin:1.2.3.RELEASE")
    }
}

apply plugin: 'java'
apply plugin: 'idea'
apply plugin: 'spring-boot'

jar {
    baseName = 'gs-serving-web-content'
    version = '0.1.0'
}

repositories {
```

```

    mavenCentral()
}

sourceCompatibility = 1.7
targetCompatibility = 1.7

dependencies {
    compile("org.springframework.boot:spring-boot-starter-thymeleaf"
    )
    compile("org.springframework.boot:spring-boot-starter")
    compile("com.fasterxml.jackson.core:jackson-databind")
    compile("org.springframework.boot:spring-boot-devtools")
    compile("org.springframework.boot:spring-boot-starter-security")
    compile('org.springframework.security.oauth:spring-security-
        oauth2')
    compile("org.springframework:spring-jdbc")
    compile("org.springframework:spring-web")
    compile("com.h2database:h2")
    testCompile("junit:junit")
}

task wrapper(type: Wrapper) {
    gradleVersion = '2.3'
}

```

Listing D.6: Example product lookup of platform client using RestTemplate

```

@RequestMapping(value="/cross-selling-widget")
public String crossSellingWidget(@RequestParam(value="prodId",
    required=false)String prodId,
                                Model model){

    Map<String, String> vars = new HashMap<String, String>();
    vars.put("prodId", crossProdId);

    HttpHeaders httpHeaders = new HttpHeaders();
    httpHeaders.add("Authorization", "Bearer_" + token);

    RestTemplate restTemplate = new RestTemplate();
    HttpEntity<String> request = new HttpEntity<String> (
        httpHeaders);
    ResponseEntity<Product> response = restTemplate.exchange(uri
        , HttpMethod.GET, request, Product.class, vars) ;

    Product product = response.getBody();
    model.addAttribute("product", product);

    return "cross_selling_widget";
}

```

Appendix E

Trade Compliance Service

The provided tax compliance service contains two operations for creating new products and for requesting taxes and customs in real time. Listing E.1 shows the segment of the Web Service's WSDL. An example for a tax and customs request is shown in Listing E.2 and the corresponding response is shown in Listing E.3.

Listing E.1: Tax compliance web service operations

```
<portType name="merchant_wsdlPortType">
  <operation name="getRequest">
    <documentation>
      Returns translated shoppingcart or Fault
    </documentation>
    <input message="tns:getRequestRequest"/>
    <output message="tns:getRequestResponse"/>
  </operation>
  <operation name="addProduct">
    <documentation>Returns fault or true</documentation>
    <input message="tns:addProductRequest"/>
    <output message="tns:addProductResponse"/>
  </operation>
</portType>
```

Listing E.2: Tax compliance web service request

```
<soapenv:Body>
  <mer:getRequest soapenv:encodingStyle=
    "http://schemas.xmlsoap.org/soap/encoding/">
    <merchantid xsi:type="xsd:string">CTM01</merchantid>
    <invoicelines xsi:type="urn:InvoiceLines"
      soapenc:arrayType="urn:InvoiceLine[]"
      xmlns:urn="urn:merchant_wsdl">
      <invoiceline>
        <line_currency>EUR</line_currency>
        <line_product>2</line_product>
        <line_category>02</line_category>
        <amount>59.5</amount>
      </invoiceline>
      <invoiceline>
        <line_currency>EUR</line_currency>
        <line_product>3</line_product>
        <line_category>02</line_category>
        <amount>1439.9</amount>
      </invoiceline>
```



```

</invoicelines>
<shipping>5.00</shipping>
<shipping_currency>EUR</shipping_currency>
<ship_to>us</ship_to>
<shippingaddress xsi:type="urn:ShippingAddress"
  xmlns:urn="urn:merchant_wdsl">
  <name>John Smith</name>
  <street>Random Street 15</street>
  <city>San Francisco</city>
  <postcode>12345</postcode>
</shippingaddress>
</mer:getRequest>
</soapenv:Body>

```

Listing E.3: Tax compliance web service response

```

<SOAP-ENV:Body>
  <ns1:getRequestResponse xmlns:ns1="merchant_wdsl">
    <return>
      <total_amount_payable xsi:type="xsd:string">
        1820.3300
      </total_amount_payable>
      <payable_currency xsi:type="xsd:string">EUR</payable_currency>
      <lines xsi:type="SOAP-ENC:Array"
        SOAP-ENC:arrayType="unnamed_struct_use_soapval[2]">
        <item>
          <line_total xsi:type="xsd:float">72</line_total>
          <line_base xsi:type="xsd:string">59.50</line_base>
          <line_vat xsi:type="xsd:string">12.50</line_vat>
          <line_currency xsi:type="xsd:string">EUR</line_currency>
          <line_product xsi:type="xsd:string">2</line_product>
          <vat_tarif xsi:type="xsd:string">21%</vat_tarif>
        </item>
        <item>
          <line_total xsi:type="xsd:float">1742.28</line_total>
          <line_base xsi:type="xsd:string">1439.90</line_base>
          <line_vat xsi:type="xsd:string">302.38</line_vat>
          <line_currency xsi:type="xsd:string">EUR</line_currency>
          <line_product xsi:type="xsd:string">3</line_product>
          <vat_tarif xsi:type="xsd:string">21%</vat_tarif>
        </item>
      </lines>
      <exchange_rate xsi:type="xsd:int">1</exchange_rate>
      <total_base_amount xsi:type="xsd:float">1499.4</
        total_base_amount>
      <total_vat_amount xsi:type="xsd:float">314.88</
        total_vat_amount>
      <shipping xsi:type="xsd:float">6.05</shipping>
      <total_customs_amount xsi:type="xsd:float">
        36.29
      </total_customs_amount>
    </return>
  </ns1:getRequestResponse>
</SOAP-ENV:Body>

```

Bibliography

- Accenture (2011). European Cross-border E-commerce: The Challenge of Achieving Profitable Growth. Technical report.
- Agrawal, M., Hariharan, G., Rao, H. R., and Kishore, R. (2013). Competition In Mediation Services: Modeling the Role of Expertise, Satisfaction, and Switching Costs. *Journal of Organizational Computing and Electronic Commerce*, 23(3):169–199.
- Albrecht, C. C., Dean, D. L., and Hansen, J. V. (2005). Marketplace and technology standards for B2B e-commerce: Progress, challenges, and the state of the art. *Information & Management*, 42(6):865–875.
- Alter, S. (2008). Service system fundamentals: Work system, value chain, and life cycle. *IBM Systems Journal*, 47(1):71–85.
- 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.
- Aussenac-Gilles, N., Biebow, B., and Szulman, S. (2000). Revisiting ontology design: A method based on corpus analysis. In *International Conference on Knowledge Engineering and Knowledge Management*, pages 172–188. Springer.
- Bachlechner, D., Siorpaes, K., Fensel, D., and Toma, I. (2006). Web service discovery-a reality check. In *3rd European Semantic Web Conference*, volume 308.
- Baghdadi, Y. (2004). ABBA: An architecture for deploying business-to-business electronic commerce applications. *Electronic Commerce Research and Applications*, 3(2):190–212.
- Baghdadi, Y. (2005). A web services-based business interactions manager to support electronic commerce applications. In *Proceedings of the 7th International Conference on Electronic Commerce*, pages 435–445. ACM.
- Baida, Z., de Bruin, H., and Gordijn, J. (2003). E-Business Cases Assessment: From Business Value to System Feasibility. *International Journal of Web Engineering and Technology*, 1(1):127–144.

- Baldwin, C. Y. and Woodard, C. J. (2009). The architecture of platforms: A unified view. In Gawer, A., editor, *Platforms, Markets and Innovation*, pages 19–44. Edward Elgar.
- Banjo, S. (2013). Rampant Returns Plague E-Retailers. *Wall Street Journal*.
- Baquero, A., Taylor, R. N., Baquero, A., and Taylor, R. (2012). A Multidimensional Evaluation of Integrative E-commerce Architectures. Technical report, Institute for Software Research, University of California, Irvine.
- Bass, F. M. (2004). Comments on “A New Product Growth for Model Consumer Durables The Bass Model”. *Management Science*, 50(12_supplement):1833–1840.
- Basu, A. and Myulle, S. (2003). Online support for commerce processes by web retailers. *Decision Support Systems*, 34(4):379–395.
- Becker, J. and Schutte, R. (2007). A reference model for retail enterprises. In Fettke, P. and Loos, P., editors, *Reference Modeling for Business Systems Analysis*, pages 182–205. IGI Global.
- Benlian, A., Hilkert, D., and Hess, T. (2015). How open is this platform? The meaning and measurement of platform openness from the complementors’ perspective. *Journal of Information Technology*, 30(3):209–228.
- Bezemer, C. P., Zaidman, A., Platzbeecker, B., Hurkmans, T., and Hart, A. . (2010). Enabling multi-tenancy: An industrial experience report. In *IEEE International Conference on Software Maintenance*.
- Bhattacharjee, A. (2012). Social science research: Principles, methods, and practices. *USF Tampa Library Open Access Collections*, Textbook 3.
- Bichler, M., Beam, C., and Segev, A. (1998). Services of a broker in electronic commerce transactions. *Electronic Markets*, 8(1):27–31.
- Blauw Research and GfK Retail and Technology (2012). Online betalen. Market analysis. Retrieved from <https://www.thuiswinkel.org/kennis/publicatie/27/online-betalen>.
- Boehm, B. (1988). A spiral model of software development and enhancement. *Computer*, 21(5):61–72.
- Brooks, F. P. (1987). No Silver Bullet: Essence and Accidents of Software Engineering. *Computer*, 20(4):10–19.
- Brynjolfsson, E., Hu, Y. J., and Smith, M. D. (2006). From niches to riches: Anatomy of the long tail. *Sloan Management Review*, 47(4):67–71.
- Burt, S. and Sparks, L. (2003). E-commerce and the retail process: A review. *Journal of Retailing and Consumer Services*, 10(5):275–286.

- Busse, S., Kutsche, R.-D., Leser, U., and Weber, H. (1999). *Federated Information Systems: Concepts, Terminology and Architectures*. Number 99-9 in Forschungsberichte des Fachbereichs Informatik. Technische Universität Berlin.
- Chang, S. and Wang, C.-W. (2010). Effectively Generating and Delivering Personalized Product Information: Adopting the Web 2.0 Approach. In *IEEE International Conference on Advanced Information Networking and Applications Workshops*, pages 401–406.
- Chen, M., Zhang, D., and Zhou, L. (2007). Empowering collaborative commerce with Web services enabled business process management systems. *Decision Support Systems*, 43(2):530–546.
- Chen, P.-K. and Su, C.-H. (2011). The e-business strategies fit on different supply chain integration structures. *African Journal of Business Management*, 5(16):7130–7141.
- Chen, Q., Rodgers, S., and He, Y. (2008). A Critical Review of the E-Satisfaction Literature. *American Behavioral Scientist*, 52(1):38–59.
- Chitura, T., Mupemhi, S., Dube, T., and Bolongkikit, J. (2008). Barriers to electronic commerce adoption in small and medium enterprises: A critical literature review. *Journal of Internet Banking and Commerce*, 13(2):1–13.
- Chou, T.-H. and Lee, Y.-M. (2008). Integrating E-services with a Telecommunication E-commerce Using EAI/SOA Technology. In *International Conference on Information Management, Innovation Management and Industrial Engineering*, volume 2, pages 385–388.
- Chung, J.-Y., Lin, K.-J., and Mathieu, R. (2003). Web services computing: advancing software interoperability. *Computer*, 36(10):35–37.
- Clark, K. J. (2015). Integration architecture: Comparing web APIs with service-oriented architecture and enterprise application integration. Technical report, IBM.
- Clarke, R. (2000). Appropriate research methods for electronic commerce. Xamax Consultancy. Retrieved from <http://www.rogerclarke.com/EC/ResMeth.html>.
- Cohen, M. A., Ho, T. H., and Matsuo, H. (2000). Operations Planning in the Presence of Innovation-Diffusion Dynamics. In Mahajan, V., Muller, E., and Wind, Y., editors, *New-Product Diffusion Models*, pages 237–262. Springer.
- Cross, N. (2006). *Designrly Ways of Knowing*. Springer.
- Croxton, K. L. (2003). The Order Fulfillment Process. *International Journal of Logistics Management*, 14(1):19–32.

- Dahlberg, T., Mallat, N., Ondrus, J., and Zmijewska, A. (2008). Past, present and future of mobile payments research: A literature review. *Electronic Commerce Research and Applications*, 7(2):165–181.
- Danaiaata, D. and Hurbean, C. (2010). SaaS–Better solution for small and medium-sized enterprises. In *Applied Economics, Business and Development*. WSEAS.
- Davenport, T. H. and Patil, D. J. (2012). Data Scientist: The Sexiest Job of the 21st Century. *Harvard Business Review*, (October 2012):70–76.
- Dorn, J., Grün, C., Werthner, H., and Zapletal, M. (2007). A Survey of B2B Methodologies and Technologies: From Business Models towards Deployment Artifacts. In *Hawaii International Conference on System Sciences*, pages 143a–143a.
- Dromey, R. G. (1996). Cornering the Chimera. *IEEE Software*, 13(1):33–43.
- El Ayadi, A. (2011). *Einsatz von SOA in E-Commerce Systemen*. Master thesis, HAW Hamburg.
- Erbes, J., Reza, H., Nezhad, M., and Graupner, S. (2012). From IT Providers to IT Service Brokers: The Future of Enterprise IT in the Cloud World. *Computer*, 99(2).
- Erl, T. (2008). *Soa: Principles of Service Design*. Prentice Hall.
- Esswein, W., Zumpe, D.-K. S., and Sunke, N. (2004). Identifying the quality of e-commerce reference models. In *International Conference on Electronic Commerce*, pages 288–295. ACM.
- European Commission (2015). Digital Agenda Scoreboard 2015. Digital Agenda Targets - Progress report. Retrieved from http://ec.europa.eu/newsroom/dae/document.cfm?action=display&doc_id=9969.
- European Commission (2016). VAT Thresholds applied by Member States. Retrieved from http://ec.europa.eu/taxation_customs/sites/taxation/files/resources/documents/taxation/vat/traders/vat_community/vat_in_ec_annexi.pdf.
- Evans, D. S., Hagiu, A., and Schmalensee, R. (2008). *Invisible Engines*. MIT Press.
- Evans, D. S. and Schmalensee, R. (2010). Failure to Launch: Critical Mass in Platform Businesses. *Review of Network Economics*, 9(4).
- Evans, E. (2004). *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley Professional.
- Feindt, S., Jeffcoate, J., and Chappell, C. (2002). Identifying Success Factors for Rapid Growth in SME E-commerce. *Small Business Economics*, 19(1):51–62.

- Feipeng, G. and Qibei, L. (2010). The Research and Implementation of Supply Chain Resource Integration Platform on Textile Industry. In *International Forum on Information Technology and Applications*, volume 2, pages 16–19.
- Fettke, D.-W.-I. P. and Loos, P. D. P. (2003). Model Driven Architecture (MDA). *Wirtschaftsinformatik*, 45(5):555–559.
- Fettke, P. and Loos, P. (2007). Perspectives on Reference Modeling. In Fettke, P. and Loos, P., editors, *Reference Modeling for Business Systems Analysis*, pages 1–21. IGI Global.
- Folmer, E. J. A. (2012). *Quality of Semantic Standards*. PhD thesis, University of Twente.
- Fowler, M., Rice, D., Foemmel, M., Hieatt, E., Mee, R., and Stafford, R. (2002). Mapping to Relational Databases. In *Patterns of Enterprise Application Architecture*, pages 35–50. Addison Wesley.
- Fragidis, G. and Tarabanis, K. (2008). An Extended SOA Model for Customer-Centric E-Commerce. In *IEEE International Conference on E-Business Engineering*, pages 771–775.
- Frank, U. (2004). E-MEMO: Referenzmodelle zur ökonomischen Realisierung leistungsfähiger Infrastrukturen für Electronic Commerce. *Wirtschaftsinformatik*, 46(5):373–381.
- Frank, U. and Lange, C. (2004). ECOMOD - Reference Business Processes and Strategies for E-Commerce. Online library, Research Group Enterprise Modelling - University Duisburg-Essen.
- Frank, U. and Lange, C. (2007). E-MEMO: A method to support the development of customized electronic commerce systems. *Information Systems and e-Business Management*, 5(2):93–116.
- Ganguly, D. and Bhattacharyya, S. (2011). Winning the Industrial Competitiveness with E-Commerce Adopting Component-Based Software Architecture. In Jin, D. and Lin, S., editors, *Advances in Computer Science, Intelligent System and Environment*, number 105 in *Advances in Intelligent and Soft Computing*, pages 69–75. Springer.
- Gao, H., Zhang, J., Povalej, R., and Stucky, W. (2009). Service-Oriented Modeling Method for the Development of an E-Commerce Platform. In *International Conference on E-Business and Information System Security*.
- Gattiker, T. F. and Goodhue, D. L. (2004). Understanding the local-level costs and benefits of ERP through organizational information processing theory. *Information & Management*, 41(4):431–443.

- Gazit, I. (2012). OAuthLib 0.7.2 documentation. Retrieved from <https://oauthlib.readthedocs.org/en/latest/index.html>.
- Giessmann, A. and Legner, C. (2016). Designing business models for cloud platforms. *Information Systems Journal*, 26(5):551–579.
- Goodwin, P., Meeran, S., and Dyussekeneva, K. (2014). The challenges of pre-launch forecasting of adoption time series for new durable products. *International Journal of Forecasting*, 30(4):1082–1097.
- Grady, R. B. and Caswell, D. L. (1987). *Software Metrics: Establishing a Company-Wide Program*. Prentice-Hall.
- Gregor, S. (2006). The nature of theory in information systems. *MIS Quarterly*, 30(3):611–642.
- Gregor, S. and Jones, D. (2007). The Anatomy of a Design Theory. *Journal of the Association for Information Systems*, 8(5):1.
- Grün, B. and Leisch, F. (2008). FlexMix version 2: Finite mixtures with concomitant variables and varying and constant parameters. *Journal of Statistical Software*, 28(4):1–35.
- Gunasekaran, A., Marri, H. B., McGaughey, R. E., and Nebhwani, M. D. (2002). E-commerce and its impact on operations management. *International Journal of Production Economics*, 75(1–2):185–197.
- Gunasekaran, A. and Ngai, E. (2004). Information systems in supply chain integration and management. *European Journal of Operational Research*, 159(2):269–295.
- Guy, S. (2016). Retailers are divided about big investments in tech spending. Retrieved from <https://www.internetretailer.com/2016/02/19/retailers-are-divided-about-big-investments-tech-spending>.
- Haesen, R., Snoeck, M., Lemahieu, W., and Poelmans, S. (2008). On the Definition of Service Granularity and Its Architectural Impact. In *Conference on Advanced Information Systems Engineering*, Lecture Notes in Computer Science, pages 375–389. Springer.
- Handschuh, S., Schmid, B. F., and Stanoevska-Slabeva, K. (1997). The Concept of a Mediating Electronic Product Catalog. *Electronic Markets*, 7(3):33–35.
- Hardt, D. (2012). The OAuth 2.0 Authorization Framework. Technical Standard. Retrieved from <https://tools.ietf.org/html/rfc6749>.
- Hashem, I. A. T., Yaqoob, I., Anuar, N. B., Mokhtar, S., Gani, A., and Ullah Khan, S. (2015). The rise of “big data” on cloud computing: Review and open research issues. *Information Systems*, 47:98–115.

- Hashemi, S., Razzazi, M., and Bahrami, A. (2006). ISRUP E-Service Framework for agile Enterprise Architecting. In *International Conference on Information Technology: New Generations*, pages 701–706.
- Hernandez, G., Hernandez, C., and Aguirre, J. (2005). BPIMS-WS: Brokering Architecture for Business Processes Integration in B2B E-Commerce. In *International Conference on Electronics, Communications and Computers*, pages 160–165.
- Hevner, A. R., March, S. T., Park, J., and Ram, S. (2004). Design science in information systems research. *MIS Quarterly*, 28(1):75–105.
- Hinton, J. (2014). Small businesses must commit time, resources for e-commerce to pay off. Retrieved from <https://mibiz.com/item/21961-small-businesses-must-commit-time-resources-for-e-commerce-to-pay-off>.
- Hohpe, G. and Woolf, B. (2003). *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Professional.
- Hu, X., Wang, X., Sun, L., and Xu, Z. (2009). A Real-Time Intelligent System for Order Processing in B2C E-Commerce. *International Journal of Innovative Computing Information and Control*, 5(11A):3691–3706.
- Hudetz, K., Mohr, N., Mertens, S., and Himmelreich, A. (2016). Erosion im Handel: Dramatische Veränderungen durch Digitalisierung bis 2020. Industry report. Institut für Handelsforschung Köln. Retrieved from <http://www.ifhkoeln.de/pressemitteilungen/details/der-handel-muss-sich-neu-erfinden-5-thesen-zur-zukunft-des-handels/>.
- Humeau, P. and Jung, M. (2013). Benchmark of e-Commerce solutions. Industry report. NBS Lab. Retrieved from <https://www.nbs-system.co.uk/nbs-lab>.
- Iqbal, M., Nieves, M., and Taylor, S. (2007). *Service Strategy*. ITIL. TSO.
- Iqbal, R., Shah, N., James, A., and Cichowicz, T. (2013). Integration, optimization and usability of enterprise applications. *Journal of Network and Computer Applications*, 36(6):1480–1488.
- Jen, L. and Lee, Y. (2000). Recommended Practice for Architectural Description of Software-Intensive Systems. Standard, IEEE.
- Jiang, C. and Song, W. (2010). An Online Third Party Payment Framework in E-commerce. In *International Conference on Advanced Computer Control*. IEEE.
- Jiao, J. R. and Helander, M. G. (2006). Development of an electronic configure-to-order platform for customized product development. *Computers in Industry*, 57(3):231–244.

- Kahn, K. B. (2002). An exploratory Investigation of new product forecasting practices. *Journal of Product Innovation Management*, 19(2):133–143.
- Kauffman, R. J. and Walden, E. A. (2001). Economics and Electronic Commerce: Survey and Directions for Research. *International Journal of Electronic Commerce*, 5(4):5–116.
- Kephart, J. O. and Chess, D. M. (2003). The vision of autonomic computing. *Computer*, 36(1):41–50.
- Kim, D. J., Agrawal, M., Jayaraman, B., and Rao, H. R. (2003). A comparison of B2B e-service solutions. *Communications of the ACM*, 46(12):317–324.
- Kim, S.-y. and Smari, W. (2005). On a collaborative commerce framework and architecture for next generation commerce. In *Proceedings of the 2005 International Symposium on Collaborative Technologies and Systems*, pages 282–289.
- Kitchenham, B. (2004). Procedures for Performing Systematic Reviews. Technical Report TR/SE-0401, Software Engineering Group Department of Computer Science Keele University.
- Kleeberg, M., Zirpins, C., and Kirchner, H. (2014). Information Systems Integration in the Cloud: Scenarios, Challenges and Technology Trends. In Brunetti, G., Feld, T., Heuser, L., Schnitter, J., and Webel, C., editors, *Future Business Software*, Progress in IS, pages 39–54. Springer.
- Konsynski, B. and Tiwana, A. (2005). Spontaneous Collaborative Networks. In Vervest, P. P., van Heck, P. E., Pau, P. L.-F., and Preiss, P. K., editors, *Smart Business Networks*, pages 75–89. Springer.
- Koppius, O. R. and van de Laak, A. J. (2009). The Quick-Connect Capability and Its Antecedents. In Vervest, P. H. M., van Liere, D. W., and Zheng, L., editors, *The Network Experience*, pages 267–284. Springer.
- Kuhn, M. and Johnson, K. (2013). *Applied Predictive Modeling*. Springer.
- Kumar, K. and van Hilleegersberg, J. (2000). Enterprise Resource Planning: Introduction. *Communications of the ACM*, 43(4):22–26.
- Lackermair, G. (2011). Hybrid cloud architectures for the online commerce. *Procedia Computer Science*, 3(2011):550–555.
- Lan, J., Liu, Y., and Chai, Y. (2008). A solution model for Service-oriented architecture. In *World Congress on Intelligent Control and Automation*, pages 4184–4189.
- Lankhorst, M. M., Proper, H. A., and Jonkers, H. (2009). The Architecture of the ArchiMate Language. In Halpin, T., Krogstie, J., Nurcan, S., Proper, E., Schmidt, R., Soffer, P., and Ukor, R., editors, *Enterprise, Business-Process and Information Systems Modeling*, number 29 in Lecture Notes in Business Information Processing, pages 367–380. Springer.

- Lankhorst, M. M., Zoet, M. M., Janssen, W. P. M., and Molnar, W. A. (2012). Agility. In Lankhorst, M., editor, *Agile Service Development*, The Enterprise Engineering Series, pages 17–40. Springer.
- Layo, F. and Silver, N. (2013). Ship-from-Store: If You Haven’t Started Yet, You’re Dangerously Behind. Industry report. Kurt Salmon Consulting. Retrieved from <http://www.kurtsalmon.com/uploads/Why%2BShip%2Bfrom%2BStore%2B130509VFSP.pdf>.
- Lee, S. M., Hwang, T., and Kim, J. (FAL 2007). An analysis of diversity in electronic commerce research. *International Journal of Electronic Commerce*, 12(1):31–67.
- Lee, S. M., Hwang, T., and Lee, D. H. (2011). Evolution of Research Areas, Themes, and Methods in Electronic Commerce. *Journal of Organizational Computing & Electronic Commerce*, 21(3):177–201.
- Leong, L., Toombs, D., and Gill, B. (2015). Magic Quadrant for Cloud Infrastructure as a Service, Worldwide. Market analysis G00265139, Gartner.
- Lewis, J. and Fowler, M. (2014). Microservices. Retrieved from <http://martinfowler.com/articles/microservices.html>.
- Li, M. and Dong, M. (2010). A Case Study: Liquor Enterprise E-Commerce Platform Construction. In *International Conference on E-Business and Information System Security*.
- Lincke, D.-M. and Schmid, B. (1998). Mediating Electronic Product Catalogs. *Communications of the ACM*, 41(7):86–88.
- Liu, D.-R. and Hwang, T.-F. (2004). An agent-based approach to flexible commerce in intermediary-centric electronic markets. *Journal of Network and Computer Applications*, 27(1):33–48.
- Liu, J., Zhang, S., and Hu, J. (2005). A case study of an inter-enterprise workflow-supported supply chain management system. *Information & Management*, 42(3):441–454.
- Lui, D. M., Gray, M., Chan, A., and Long, J. (2011). Spring Integration and Your Web Application. In *Pro Spring Integration*, pages 591–614. Apress.
- Maler, E. and Hammond, J. S. (2013). API Management Platforms, Q1 2013. Market analysis, Forrester.
- Malinverno, P., Plummer, D. C., and Van Huizen, G. (2013). Magic Quadrant for Application Services Governance. Market analysis G00247339, Gartner.
- Mandal, P. and Gunasekaran, A. (2003). Issues in implementing ERP: A case study. *European Journal of Operational Research*, 146(2):274–283.

- Martens, B., Walterbusch, M., and Teuteberg, F. (2012). Costing of Cloud Computing Services: A Total Cost of Ownership Approach. In *Hawaii International Conference on System Sciences*, pages 1563–1572.
- Mason, R. (2011). Real-time Web and Streaming APIs. Retrieved from <http://blogs.mulesoft.org/real-time-web-and-streaming-apis>.
- McCall, J. A., Richards, P. K., and Walters, G. F. (1977). Factors in software quality. Volume I. Concepts and Definitions of Software Quality. Technical report, General Electric Company.
- Medjahed, B., Benatallah, B., Bouguettaya, A., Ngu, A. H. H., and Elmagarmid, A. K. (2003). Business-to-business interactions: Issues and enabling technologies. *International Journal on Very Large Data Bases*, 12(1):59–85.
- Merrifield, R., Calhoun, J., and Stevens, D. (2008). The next revolution in productivity. *Harvard Business Review*, 86(6):72.
- Mohamed, U., Galal-Edeen, G., and El-Zoghbi, A. (2010). Building Integrated Oil and Gas B2B E-commerce Hub Architecture Based on SOA. In *International Conference on E-Education, E-Business, E-Management, and E-Learning*, pages 599–608.
- Ngai, E. and Wat, F. (2002). A literature review and classification of electronic commerce research. *Information & Management*, 39(5):415–429.
- Nitu (2009). Configurability in SaaS (Software As a Service) Applications. In *Proceedings of the 2nd India Software Engineering Conference*, pages 19–26.
- O’Brien, L., Merson, P., and Bass, L. (2007). Quality Attributes for Service-Oriented Architectures. In *International Workshop on Systems Development in SOA Environments*.
- O’Leary, D. E. (2000). *Enterprise Resource Planning Systems: Systems, Life Cycle, Electronic Commerce, and Risk*. Cambridge University Press.
- Ordonez, C. (2011). Data Set Preprocessing and Transformation in a Database System. *Intelligent Data Analysis*, 15(4):613–631.
- Ortega, M., Perez, M., and Rojas, T. (2003). Construction of a Systemic Quality Model for Evaluating a Software Product. *Software Quality Journal*, 11(3):219–242.
- Osterwalder, A. and Pigneur, Y. (2010). *Business model generation: a handbook for visionaries, game changers, and challengers*. John Wiley & Sons.
- Palmer, J. W. and Johnston, J. S. (1996). Business-to-business connectivity on the internet: EDI, intermediaries, and interorganizational dimensions. *Electronic Markets*, 6(2):3–6.

- Papazoglou, M. (2003). Service-oriented computing: Concepts, characteristics and directions. In *International Conference on Web Information Systems Engineering*, pages 3–12.
- Paramartha, M. A. (2014). *Design and Instantiation of Reference Architecture of Pluggable Service Platform in E-Commerce*. Master thesis, University of Twente.
- Peffer, K., Tuunanen, T., Rothenberger, M. A., and Chatterjee, S. (2007). A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, 24(3):45–77.
- PengLing and Mian, Z. (2010). Outsourcing E-Commerce Model and Platform System Structure. In *International Conference on Management of E-Commerce and E-Government*, pages 122–127.
- Pezzini, M., Natis, Y. V., Malinverno, P., Iijima, K., Thompson, J., and Thoo, E. (2014). Magic Quadrant for Enterprise Integration Platform as a Service.
- Pincvision (2012). Are you completely compliant? Industry report. Retrieved from <http://www.pincvision.com/brochure/?page=trade-compliance-scan>.
- Pivotal (2015). The Essential Elements of Enterprise PaaS. Industry report. Retrieved from <http://pivotal.io/platform/white-paper/enterprise-paas-essential-elements>.
- Potočník, M. and Juric, M. B. (2012). Integration of SaaS using IPaaS. In *Proceedings of the 1st International Conference on Cloud Assisted Services*, pages 35–41.
- Prat, N., Akoka, J., and Comyn-Wattiau, I. (2006). A UML-based data warehouse design method. *Decision Support Systems*, 42(3):1449–1473.
- Puschmann, T. and Alt, R. (2001). Enterprise application integration-the case of the Robert Bosch Group. In *Hawaii International Conference on System Sciences*.
- R Development Core Team (2014). *R: A Language and Environment for Statistical Computing*. The R Foundation for Statistical Computing.
- Rao, S., Griffis, S. E., and Goldsby, T. J. (2011). Failure to deliver? Linking online order fulfillment glitches with future purchase behavior. *Journal of Operations Management*, 29(7–8):692–703.
- Reimers, K. (2001). Standardizing the new e-business platform: Learning from the EDI experience. *Electronic Markets*, 11(4):231–237.
- Ried, S. (2014). Hybrid Integration, Q1 2014. Market analysis, Forrester.
- Romano Jr., N. C. R. and Fjermestad, J. (2003). Electronic Commerce Customer Relationship Management: A Research Agenda. *Information Technology and Management*, 4(2-3):233–258.

- Sabki, A., Ahmed, P. K., and Hardaker, G. (2004). Developing an e-commerce solution: A case study of TimeXtra. *Journal of Enterprise Information Management*, 17(5):388–401.
- Saldaña, J. (2012). *The Coding Manual for Qualitative Researchers*. Sage.
- Saltor, F., Castellanos, M., and García-Solaco, M. (1991). Suitability of Datamodels As Canonical Models for Federated Databases. *ACM SIGMOD Record*, 20(4):44–48.
- Schepers, T. G. J., Jacob, M. E., and Van Eck, P. A. T. (2008). A Lifecycle Approach to SOA Governance. In *ACM Symposium on Applied Computing*, pages 1055–1061.
- Schibrowsky, J. A., Peltier, J. W., and Nill, A. (2007). The state of internet marketing research: A review of the literature and future research directions. *European Journal of Marketing*, 41(7/8):722–733.
- Scott, J. (1999). The FoxMeyer Drugs' Bankruptcy: Was it a Failure of ERP? In *Americas Conference on Information Systems*, pages 223–225.
- Sein, M., Henfridsson, O., Purao, S., Rossi, M., and Lindgren, R. (2011). Action Design Research. *MIS Quarterly*, 35(1):37–56.
- Sen, R. and King, R. C. (2003). Revisit the Debate on Intermediation, Disintermediation and Reintermediation due to E-commerce. *Electronic Markets*, 13(2):153–162.
- Sheth, A. P. and Larson, J. A. (1990). Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3):183–236.
- Singh, M. (2002). E-services and their role in B2C e-commerce. *Managing Service Quality*, 12(6):434–446.
- Smith, C. (2014). These Charts Show How Alibaba Became The World's Largest E-Commerce Company. Retrieved from <http://www.businessinsider.de/these-charts-show-how-alibaba-became-the-worlds-largest-e-commerce-company-2014-5>.
- Soh, C., Kien, S. S., and Tay-Yap, J. (2000). Enterprise Resource Planning: Cultural Fits and Misfits: Is ERP a Universal Solution? *Communications of the ACM*, 43(4):47–51.
- Soshkin, M. (2015). Order Fulfillment Services in the US. Market analysis. Retrieved from <http://www.ibisworld.com/industry/order-fulfillment-services.html>.
- Srinivasan, S. S., Anderson, R., and Ponnayolu, K. (2002). Customer loyalty in e-commerce: An exploration of its antecedents and consequences. *Journal of Retailing*, 78(1):41–50.

- Sriram, S., Manchanda, P., Bravo, M. E., Chu, J., Ma, L., Song, M., Shriver, S., and Subramanian, U. (2014). Platforms: A multiplicity of research opportunities. *Marketing Letters*, 26(2):141–152.
- Sun, H., Liu, Y., Chai, Y., and Sun, X. (2012). A novel architecture towards trusted E-commerce cloud. In *International Conference on E-Learning and E-Technologies in Education*, pages 223–229.
- Sun, W., Zhang, K., Chen, S.-K., Zhang, X., and Liang, H. (2007). Software as a Service: An Integration Perspective. In *International Conference on Service-Oriented Computing*, Lecture Notes in Computer Science, pages 558–569. Springer.
- Tallon, P. P. (2013). Corporate governance of big data: Perspectives on value, risk, and cost. *Computer*, 46(6):32–38.
- Tan, H. (2011). Design of E-Commerce Platform Based on Supply Chain Management. In Qi, L., editor, *Information and Automation*, number 86 in Communications in Computer and Information Science, pages 494–500. Springer.
- Tenenbaum, J. M. and Khare, R. (2005). Business Services Networks: Delivering the promises of B2B. In *International Workshop on Business Services Networks*.
- The Open Group (2016). ArchiMate 3.0 Specification. Retrieved from <http://pubs.opengroup.org/architecture/archimate3-doc/>.
- Thomas, O., Leyking, K., and Dreifus, F. (2008). Using Process Models for the Design of Service-Oriented Architectures: Methodology and E-Commerce Case Study. In *Hawaii International Conference on System Sciences*.
- Tolk, A. and Mugira, J. A. (2003). The levels of conceptual interoperability model. In *Simulation Interoperability Workshop*.
- van Heck, E. and Vervest, P. (2007). Smart Business Networks: How the Network Wins. *Communications of the ACM*, 50(6):28–37.
- van Hilleberg, J., Moonen, H., and Dalmolen, S. (2012). Coordination as a Service to Enable Agile Business Networks. In Kotlarsky, J., Oshri, I., and Willcocks, L. P., editors, *The Dynamics of Global Sourcing. Perspectives and Practices*, number 130 in Lecture Notes in Business Information Processing, pages 164–174. Springer.
- Verschuren, P. and Hartog, R. (2005). Evaluation in Design-Oriented Research. *Quality and Quantity*, 39(6):733–762.
- Vujasinovic, M., Barkmeyer, E., Ivezic, N., and Marjanovic, Z. (2010). Interoperable Supply-Chain Applications: Message Metamodel-Based Semantic Reconciliation of B2B Messages. *International Journal of Cooperative Information Systems*, 19(01n02):31–69.

- Wallace, D. W., Giese, J. L., and Johnson, J. L. (2004). Customer retailer loyalty in the context of multiple channel strategies. *Journal of Retailing*, 80(4):249–263.
- Walls, J. G., Widmeyer, G. R., and El Sawy, O. A. (1992). Building an information system design theory for vigilant EIS. *Information Systems Research*, 3(1):36–59.
- Wan, K., Alagar, V., and Ibrahim, N. (2013). An Extended Service-Oriented Architecture for Consumer-Centric E-Commerce. *International Journal of Information and Communication Technology Research*, 3(1):74–101.
- Wan, X. and Huang, L. (2008). Research on e-Commerce Application Architecture Based on the Integration of Workflow and Agile Service. In *International Symposium on Electronic Commerce and Security*, pages 843–849.
- Wang, S., Zheng, S., Xu, L., Li, D., and Meng, H. (2008). A literature review of electronic marketplace research: Themes, theories and an integrative framework. *Information Systems Frontiers*, 10(5):555–571.
- Wareham, J., Fox, P. B., and Cano Giner, J. L. (2014). Technology Ecosystem Governance. *Organization Science*, 25(4):1195–1215.
- Wareham, J., Zheng, J. G., and Straub, D. (2005). Critical themes in electronic commerce research: A meta-analysis. *Journal of Information Technology*, 20(1):1–19.
- Waters, B. (2005). Software as a service: A look at the customer benefits. *Journal of Digital Asset Management*, 1(1):32–39.
- Wedel, M., Desarbo, W. S., Bult, J. R., and Ramaswamy, V. (1993). A latent class poisson regression model for heterogeneous count data. *Journal of Applied Economics*, 8(4):397–411.
- Weinfurtner, S., Wittmann, G., Stahl, E., Wittmann, M., and Pur, S. (2013). Erfolgsfaktor Payment. Market analysis. ibi research. Retrieved from https://ecommerce-leitfaden.de/download/studien/Studie_Erfolgsfaktor_Payment_2013.pdf.
- Wen, W. (2007). A knowledge-based intelligent electronic commerce system for selling agricultural products. *Computers and Electronics in Agriculture*, 57(1):33–46.
- Wickham, H. (2011). The Split-Apply-Combine Strategy for Data Analysis. *Journal of Statistical Software*, 40(1):1–29.
- Wiederhold, G. (1992). Mediators in the architecture of future information systems. *Computer*, 25(3):38–49.
- Wieringa, R. (2009). Design Science As Nested Problem Solving. In *International Conference on Design Science Research in Information Systems and Technology*.

- Wieringa, R. J. (2014). *Design Science Methodology for Information Systems and Software Engineering*. Springer.
- Wlodarczyk, T. W., Rong, C., and Thorsen, K. A. H. (2009). Industrial Cloud: Toward Inter-enterprise Integration. In *International Conference on Cloud Computing*, number 5931 in Lecture Notes in Computer Science, pages 460–471. Springer.
- Wu, J.-H., Shin, S.-S., and Heng, M. S. H. (2007). A methodology for ERP misfit analysis. *Information & Management*, 44(8):666–680.
- Xu, J., Benbasat, I., and Cenfetelli, R. (2010). Does Live Help Service Matter? An Empirical Test of the DeLone and McLean’s Extended Model in the E-Service Context. In *Hawaii International Conference on System Sciences*.
- Yang, X., Qiang, Z., Zhao, X., and Ling, Z. (2007). Research on Distributed E-Commerce System Architecture. In *ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*, volume 2, pages 821–825.
- Ying, W. and Dayong, S. (2005). Multi-agent framework for third party logistics in E-commerce. *Expert Systems with Applications*, 29(2):431–436.
- Yoon, C. and Kim, S. (2009). Developing the Causal Model of Online Store Success. *Journal of Organizational Computing and Electronic Commerce*, 19(4):265–284.
- Zhang, Q., Yang, J., Song, C., and Yu, Y. (2009). Research on the Architecture of Iron and Steel Logistics E-Commerce System. In *International Conference on Management and Service Science*.
- Zheng, H., Liu, L., and Li, Y. (2009a). An Information Fusion-Enabled Third Party E-Commerce Platform Based on SOA. *Journal of Software*, 4(1):50–57.
- Zheng, Q., Han, Y., Li, S., Dong, J., Yan, L., and Qin, J. (2009b). E-commerce Architecture and System Design. In Zheng, Q., editor, *Introduction to E-Commerce*, pages 271–303. Springer.

Dutch Abstract

Pluggable services

Een platformarchitectuur voor e-commerce

De huidige architecturen voor e-commerce voorzien niet in de juiste mate van flexibiliteit bij het toepassen van nieuwe services om een continue klanttevredenheid te bereiken en de prestaties van e-commerce te optimaliseren. De detailhandel die vertrouwt op kant-en-klare of op maat gemaakte software heeft moeite om het tempo bij te houden van nieuwe trends zoals grensoverschrijdende en omnichannel handel.

Een veelbelovende aanpak voor het overwinnen van veel obstakels bij de adoptie en integratie van software en voor het verbeteren van pluggability in servicegeoriënteerde architecturen zijn cloud-gebaseerde diensten. Dit proefschrift introduceert een nieuw kwaliteitsmodel dat binnen platformarchitecturen voor e-commerce het meten van de aansluitbaarheid van deze diensten mogelijk maakt. De voorgestelde platformarchitectuur is ontworpen volgens een design-science onderzoeksmethode en is geïntanceerd door middel van een platformprototype. Hierop is het kwaliteitsmodel toegepast om de pluggability van de diensten uit het prototype te meten en geleidelijk het architectuurmodel te verbeteren.

Er wordt een aantal pluggable diensten voor cross-selling, verkoopprognoses en trade-compliance gepresenteerd die de recente trends in e-commerce weerspiegelen. Deze diensten illustreren de functionaliteit van het platformprototype en demonstren hoe de architectuur de pluggability van diensten kan verhogen.



Current architectures for online retail do not provide the right level of flexibility in adopting new services to achieve continuous customer satisfaction and to optimize e-commerce performance. Retailers who rely on pre-packaged or custom-built software are struggling to keep the pace with new trends like cross-border or omni-channel commerce. A promising approach to overcome many software adoption obstacles and to improve pluggability in service-oriented architectures are cloud services.

This thesis introduces a novel quality model that allows to assess the pluggability of those services in common platform architectures for e-commerce. Following a design science methodology, a state of the art architectural model is instantiated by means of a platform prototype to which the quality model is applied in order to measure the prototype's service pluggability and gradually enhance the architectural model.

A number of pluggable services for cross-selling, sales forecasting, and trade compliance are presented that reflect the recent trends in online retail. The covered services permit to demonstrate the functionality of the platform prototype and to evaluate the architecture's capability to improve service pluggability.



ISBN 978-90-365-4283-8



9 789036 542838 >