OTTO-VON-GUERICKE UNIVERSITY MAGDEBURG (GERMANY)

The Electronic Business Foundation

An Approach towards the Design of an E-Commerce Framework for Small and Medium Enterprises

Robert Neumann
(robert.neumann@densetsu.org)

Diplomarbeit
(Master Thesis)
March 23rd, 2009

Dipl.-Wirtsch.-Inf. Sebastian Günther (University of Magdeburg)
Prof. Dr.-Ing. habil. Georg Paul (University of Magdeburg)
Prof. Dr.-Ing. habil. Reiner R. Dumke (University of Magdeburg)

Prologue

Tokyo, Japan, 2nd of March, 2009, 19th floor of the Roppongi First Building: In an open-plan office, I can see myself surrounded by about 250 Japanese colleagues, all of them staring at their screens as they are doing it every day. This thesis has been an effort as I have never experienced it before and I am about to finish writing the last couple of lines. I like doing several things at the same time, but working eight hours a day as a process and application consultant at Mercedes-Benz Finance Japan and in the evening after having gotten home spending another two to three hours on my thesis bailed out even my last energy reserves. While being proud of the seven chapters I have filled with passion, I’m also happy that this effort will conclude very soon in setting the last full stop on this page.

For a long time already I have been passionate about the internet and its incredible power of bringing people from all around the world together. I am fascinated by how it created new ways of human interaction that, so I believe, eased and will continue easing our lives. Inspired by visionaries that have already realized what was considered by the grand majority as impossible, I started creating my own vision: To manage the “leap ahead” and finally manifest e-commerce in our everyday business interaction. With a focus upon electronic markets, this work is my first attempt to condense my thoughts and ideas for an enterprise that aims at equipping all businesses -no matter how small they are- with powerful e-commerce technology. By using this technology my hope is that small and medium enterprises will be able to provide better service to their customers as well as expose all their products over the internet.

I am especially happy that I have received the necessary academic support from my professor to research on a topic with many uncertainties and a large number of unknown variables. The freedom I was given has inspired me to also take things into consideration that on the first glance seemed to be very odd. But exploring this oddness and also understanding the opportunities it offers has resulted into several new attempts as they are described later in this text.

Finally, I am fortunate that I was given the chance to combine this effort with my passion for Japan. My doubts about whether I could manage to work fulltime and in addition write my thesis have steadily continued diminishing the closer I got to this day. In the end, however, I was only able to endure this endeavor due to the support I have received from home as well as the brilliant people I have worked with. Being grateful for my time here in Tokyo and all the impressions the city has left in my memories, it has come the time to finally set the last full stop.

Robert Neumann March, 2009 (Tokyo, Japan)

Acknowledgement

Having in some way or another committed to this work, my deep gratitude is directed to:

The loving one, Ksenia, with whose pure patience I endured this endeavor with great joy.

My wonderful parents for having supported me from home as well as my parents in law for having given me their "Podarok".

Mr. Sebastian Günther (Otto-von-Guericke University Magdeburg) for his excellent academic support. Your feedback has significantly determined the quality of this work.

Mr. Christoph Milde for proof-reading this thesis from his adopted country Sweden. You have been a close friend over all these years.

Prof. Georg Paul and Prof. Rainer Dumke (Otto-von-Guericke University Magdeburg) for having giving me the freedom to work on this topic and for having coordinated my activities from abroad.

Mrs. Yasuko Hamada (Vice President, Meguro UNESCO Association) for having taken care of maintaining my social balance during my time in Tokyo.

Mr. Shigeo and Mrs. Yukiko Katsuoka as well as Mrs. Mariko Doioka for having been a five star host family as well as incredible cooks.

The Mercedes-Benz Japan KG in their Roppongi Office for the beautiful view on Fuji-san.

And finally to you, Fuji-san, for having been an ever prevailing landmark that I could rely on whenever I got lost.

Table of Contents

Table of Abbreviations

AC	Accumulated Maintenance Cost
B2B	Business to Business
B2C	Business to Consumer
C2C	Consumer to Consumer
CRM	Customer Relationship Management
CSV	Comma-separated Value
EBF	Electronic Business Foundation
EBF-AF	EBF Application Foundation
EBF-IF	EBF Integration Foundation
EBF-PDL	EBF Product Description Language
EBF-PPM	EBF Product and Process Model
EBM	Electronic Business Model
ERP	Enterprise Resource Planning
ESB	Enterprise Service Bus
FAQ	Frequently Asked Questions
FEPP	Framework Engineering Principles and Practices
GUI	Graphical User Interface
IT	Information Technology
SaaS	Software-as-a-Service
SCM	Supply Chain Management
SME	Small and Medium Enterprises
SOA	Service-oriented Architecture
STEP	Standard for the Exchange of Product Data
UML	Unified Modeling Language
XML	Extensible Markup Language

1 Introduction

1.1 Motivation

The accomplishments of electronic commerce have enabled enterprises to merchandise and distribute their products and services across the whole world. Potential customers that are interested in products or services can research on prices and features on the supplier's website. They can get in touch with the supplier via internet communication channels and in most cases get immediate pricing information about offerings. Twenty four hours a day customers can purchase products and services, no matter where on the globe they currently are.

Many e-commerce solutions were developed to provide suppliers with a virtual storefront. Those e-commerce solutions are most often licensed from a software provider and hosted on a dedicated web server. The cost for licensing, customization and maintenance of the virtual storefront consumes a significant amount of the total revenue that is made with the e-business. Businesses with a solid revenue stream can more easily bring up those investments than businesses that operate upon very small profit margins and volumes. For Small and Medium Enterprises (SME), which are run by only a few individuals, it becomes literally impossible to afford an investment as such. Even though the investment amortized within a very short period of time, those businesses would not be able to gather the necessary funds. Certainly, SME could sidestep to a personal storefront on huge marketplaces, such as eBay or Amazon, but firstly such a general purpose marketplace might not provide all the functionality that is required and secondly does such a marketplace not integrate with other systems.

The SME Dilemma

While larger businesses already benefit from the developments that have taken place in the field of e-commerce, small businesses still seem to be ignored. Modern commercial concepts, such as one-to-one marketing and customer relationship-based business strategies (e.g. CRM) require business logic that can only be provided through IT. Also, an efficient communication with upstream or downstream partners of the same supply chain through integrated procurement or supply chain functionality (SCM) is only given in rare cases. Sporadically, SME have a website or a virtual storefront on which they can distribute their goods and services over the internet (Heins, 2008). At the current stage of the e-commerce evolution, the cost seems to be the most significant barrier for SME (Williams & Phillips, 1999) that hinders them from making use of modern e-commerce or electronic business concepts, including CRM and SCM.

The Cost Barrier

The technological isolation of the above described businesses is what most parts of this work will focus upon. The Goliath Vision that will be discussed in chapter six explains how SME are provided with e-commerce technology which, due to its enormous flexibility, can cater for any imaginable e-commerce scenario. Thereby, only a very low investment that should be affordable by the bulk of the businesses is required. Customization to the specific needs of an individual business, no matter what sort of products or services are dealt with, would be very easy to achieve and infrastructure not at all or only to a minimum extent required. Different businesses of different markets would use the same e-commerce solution, no matter whether they are local retailers, food marts, travel bureaus, dentists, pizza deliveries or hair cutters. By integrating all the e-commerce instances of the single businesses in one big portal, communication between each other and better alignment of transactions with business partners would become possible. The aggregation of all electronic businesses to one single entry point would allow for the unlocking of new services and experiences for customers which have never been seen before.

Goliath

A first conceptual design of the e-commerce technology that is required to enable the above vision represents the heart of this work. The specific needs of the smaller businesses determine the characteristics of a framework that is so flexible that it can cater for almost every kind of e-commerce transaction. In the same instance, the approach behind and the flexibility of the framework will have significant impact on the price at which it can be provided. With this work, the first step towards realizing the Goliath Vision is made.

1.2 Goal of this Work

Extracted from the above mission statement, the goal of this work is to provide answers to the following questions:

Chapter 1

- How was this work motivated?

Chapter 2

- What is e-commerce and what value does it generate?
- Which products and services are typically traded?
- What market making mechanisms do exist?

Chapter 3

- What are typical e-commerce applications?
- What are the shortcomings of existing e-commerce models?
- How would an e-commerce product and process model look like that is absolutely independent of the particular products and services being traded?

Chapter 4

- What are software frameworks and how do they facilitate software development?
- What needs to be considered when developing a framework?

Chapter 5

- How could the product and process model of chapter three be realized as a framework?
- What are the benefits of realizing the product and process model as a framework?
- How would the architecture of such an e-commerce framework look like?
- Which base technology could be leveraged to realize the framework?
- How could the product and process model be implemented by leveraging the base technology?

Chapter 6

- What new e-commerce experiences could the e-commerce framework unleash?
- How could the framework support the realization of the new experience?
- How could such an experience be managed project-wise?

Chapter 7

- Which conclusions could be made?
- Which topics could be the subject of future work?

Research Method

While attempting to answer the above questions, the argumentation that was employed does not rely upon empiricism. Instead, rationalism was chosen, whereby at many places arguments are based upon facts and opinions that were expressed and discussed in other, related sources (see bibliography). Including an empirical approach at this point in time and due to the limited timeframe of this work was not possible. Even though the influence of random variables is not negligible, the conclusions and results that were made would have to be validated in practice as part of follow-up projects. Future observations of the framework when it is applied could help to refine the proposed model and eliminate potentially existing flaws. Follow-up projects are already planned and will soon after the release of this work go through the specification phase. The Goliath Vision as briefed in the later part of this text represents an initial step towards the definition of one of those follow-up projects.

1.3 Chapter Outline

This work is divided into three parts. Part I aims at proposing a product and process model that is general enough to cater for any kind of e-commerce transaction. Prior to the development of such

Part I

a model, an introduction to the general concept of e-commerce will be given. It will be outlined which products are typically traded via e-commerce and what value electronic markets add to traditional ones. In addition to existing market making mechanisms, chapter two also features a mechanism that will be referred to as “Reverse Trading”. It will be shown in chapter three how the product and process model can be designed in a way that it is independent from the actual description of products and services that are traded.

Part II

Part II will - based on the insights gained from chapter two and three - try to provide a design proposal for an e-commerce framework that implements the product and process model. Thereby, the framework will have to provide an architecture that supports a strong componentization of its functionality while concurrently allowing for a smooth and seamless integration of its components. Ultimately, not only existing components should be easy to maintain, but also new components, even from external providers, should be easy to integrate. In the following, base technology upon which the framework could potentially be implemented is analyzed. A decision for one particular base technology provider will be sought by considering the Total Cost of Ownership (TCO) as well as the opportunity cost of the discussed alternatives. Based on the chosen technology, it will be shown how the product and process model could be turned into reality. Where appropriate, prototypes will be utilized to demonstrate the validity of concepts.

Part III

Part III will contain a discussion of the scope of a follow-up project that utilizes the designed framework in practice. By applying the framework to a case, valuable insights about the correctness of its assumptions are expected to be gained, whereby the identification of refinement points could help the framework to become more complete.

This text will close with a summary and a conclusion about the results that could be achieved. Finally, open questions will be called out and potential future steps will be discussed.

Quick Reviews

Throughout the text, so called Quick Reviews at the end of every section summarize what was covered. On the one hand, those Quick Reviews are meant to support the reader with manifesting the content of single sections. On the other hand, if the reader wishes to skip a section, he can do so and use the Quick Reviews to gain a quick understanding of the essential points of each section.

Part I: The EBF-PPM: A Domain-independent Product and Process Model for the Unified Catering of E-Commerce Scenarios

For many years e-commerce has been a heavily investigated discipline and the pervasiveness of digital store fronts of thousands of thousands of enterprises is the best evidence of its success. No matter whether electronics, clothes, flights or even cars, almost everything can nowadays be bought on the internet. Not only does electronic commerce allow in most cases for cheaper prices and faster service, but also enabled the development of totally new business and marketing strategies. One-to-one marketing (CRM), Supply Chain Management (SCM) and Enterprise Resource Planning (ERP) are only some examples that have changed the way in which enterprises conduct their daily business. The "digital firm" (Laudon & Laudon, 2005) has become reality and enterprises are more than ever relying on computer-assisted value generation.

Even though the evolution of e-commerce during the last twenty years has progressed at a more than rapid pace, this development seems to have by-passed SME to some extent. The number of small and independent businesses that conduct their daily operations without sophisticated IT system support is still significant (Heins, 2008). If at all those businesses are represented on the internet, they very often only have a small website that does not provide any more information than a general introduction of the firm and a listing of contact information. The advertisement and distribution of their products and services, however, still remains without any usage of sophisticated technology. The reason for that seems to become a little more obvious, when taking a look at the market of e-commerce software and vendors.

Many e-commerce solutions were developed to provide suppliers with an electronic value chain augmentation. Those e-commerce solutions are most often licensed from a software provider and hosted on a dedicated web server. The cost for licensing, customization and maintenance of the virtual storefront consumes a substantial amount of the total revenue that is made with the e-business. Businesses with a solid revenue stream can easier bring up those investments than businesses that operate upon very small profit margins and volumes. For SME that are run by only a few people it becomes literally impossible to afford an investment as such. Even though the investment could amortize within a very short period of time, those businesses would not be able to gather the necessary funds. Certainly, SME could sidestep to a personal storefront on mass marketplaces, such as eBay or Amazon, but firstly such a general purpose marketplace might not provide all the functionality that is required and secondly does such a marketplace not integrate with other systems.

By considering the monetary limitations and special conditions of SME, this work aims at stating a design proposal for an e-commerce framework that is affordable by each and every enterprise. Thereby, the framework has to be flexible enough to cater for the needs of each and every market segment. The core of this flexibility will come from a product and process model for which a design is proposed in chapter three. A previous introduction to the general concepts of e-commerce will help understanding the terminology in and rationale behind the following chapters.

2 Commerce and e-Commerce

2.1 Overview

In today's business, information systems and their applications play a major role. Not only did they help introducing new technologies, but also supported streamlining the processes within companies. Electronic commerce has become an instrument that is pervading our everyday life. The OECD (1997) describes electronic commerce (or e-commerce) as "commercial transactions occurring over open networks, such as the internet". Bichler (2001) states that those new information technologies "provide new opportunities and mechanisms to corporate or to compete" (Bichler, 2001), taking advantage of computer power, the enhanced communication options over the internet and the fact that millions of people can be simultaneously online. As many successful applications and platforms on the internet, such as eBay or Amazon, have shown, e-commerce radically altered business-to-business (B2B), business-to-consumer (B2C) as well as consumer-to-consumer (C2C) transactions. The old and well known Electronic Data Interchange (EDI) between businesses and suppliers, for example, has been enhanced by web-based front-ends for the placement of customer orders, systems for efficient customer response (e.g. CRM[1]) and systems that support the procurement process (e.g. SCM[2]).

e-Commerce

This chapter will first give a definition of the term "market" and compare it to its electronic equivalent. Paving the way for the understanding of the following chapters as well as the business rational behind the EBF, subsequently different types of electronic markets will be detailed and models for value generation in e-commerce applications discussed.

2.2 Classical vs. Electronic Markets

In classical economies, markets are usually referred to as virtual places where supply meets demand (Siebert, 1992). Even though a market can have a physical appearance, the term "virtual", in this case, encompasses even more than the traditional perception of a market being of a physical location where customers and suppliers come together to make business. The pervasiveness of trading platforms on the internet has proven that markets are not necessarily related to physical places anymore. Instead, the term "market" now describes a more abstract concept with one main function: bringing buyers and suppliers together or in other words matching the demand and the supply (Rensmann, 2007). Spulber (1999) adds four additional important functions of intermediaries (or markets): Price setting, providing liquidity and immediacy, searching and investigating as well as guaranteeing and monitoring. Bailey & Bakos (1997) derived four related roles from above mentioned functions. They say that markets aggregate buyer demand or seller products to achieve economies of scale to reduce "bargaining asymmetry". Secondly, markets protect buyers and sellers from the opportunistic behavior of

Markets

[1] CRM = Customer Relationship Management (Kramer, 2000)
[2] SCM = Supply Chain Management (Kramer, 2000)

other participants in a market by becoming an "agent of trust" (Bailey & Bakos, 1997). Markets furthermore facilitate the businesses by reducing operating costs and lastly they match buyers and sellers.

Market Equilibrium

In markets, suppliers usually offer their products or services to the buyers, whereby the price of the goods determines the quantity at which they are bought. So called "market determining forces" (Benjamin et al., 1986) determine the design of the market (e.g. competition, oligopoly, monopoly) and thus the price and the quantity at which goods and services are traded. Independently from those forces, buyers compare the offers of the different suppliers and choose the best one. The following graphic (Figure 1) demonstrates how (in competition) the demand of the buyers and the supply of the suppliers meet in a so called market equilibrium, the price-quantity combination that clears the market.

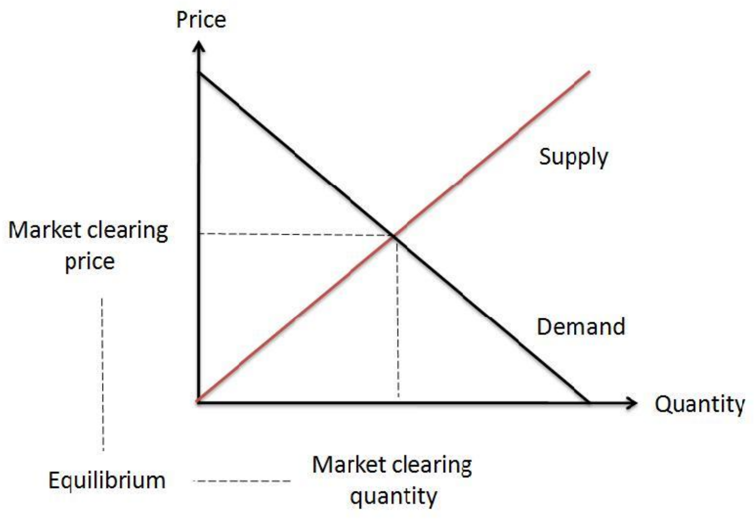


Figure 1: A Market Equilibrium with a price-quantity combination (Brickley et al., 2006).

Trust

The matching in the equilibrium establishes a contract between buyer and seller (Bichler, 2001). After both sides agreed upon a price and completed the deal, the product being sold needs to be transported to the buyer and the payment must be transferred. Referring to Bichler (2001), logistics and settlement require a certain level of trust which protects buyers and sellers. Trust, in this case, is often provided through the provider of the electronic marketplace, external service providers, such as Paypal or other third party organizations that issues a letter of credit or a rating of participants (Bichler, 2001). Furthermore, trust is established through general institutional infrastructure that specifies laws, rules and regulations to govern market transactions. Regulations, such as contract law, dispute resolution and intellectual property protection lie typically within the domain of governments (Bichler, 2001).

Markets vs. e-Markets

Even though electronic markets fulfill the same purpose as traditional markets, one of their distinctive features is that they do so by employing information and communication technology (Merz, 1999), such as the internet. Bakos (1992) defines an electronic market system as "an interorganizational information system that allows the participating buyers and sellers to exchange information about prices and product offerings".

Other fundamental differences of electronic markets to traditional markets are:

- **Transparency:** Electronic markets can be completely transparent, since the cost of search is marginal (Picot et al., 1996). Furthermore, market transparency is defined as "the ability of market participants to observe the information in the trading process (Bichler, 2001). Thereby, information can be related to current or past prices, offers, volumes and the identities and motivations of the markets participants.

- **Size:** Electronic markets are hypothetically not limited to regional borders, enabling the easy matching of partners from all over the world. Compared to traditional markets, this assumption significantly increases the number of potential trading partner. However, Bichler (2001) also states that the complexity of interaction may be heightened due to the fact that partners may be located in other countries, with a different culture and different trade customs as well as a different language.

- **Cost:** Transaction costs that evolve from the advertising, searching for trade partners and subsequent coordination are generally low due to the high degree of automation and the cheap connectivity to the internet (Wigand & Benjamin, 1993). Referring to Bichler (2001), in the early days of electronic commerce (with value added networks and EDI during the 1980s) switching costs for consumers were rather high due to significant setup costs for electronic transactions. This cost, however, has decreased as the internet and its related standards "homogenized the access channels" (Bichler, 2001).

Quick Review – Classical vs. Electronic Markets
The classical perception of markets is that they bring buyers and suppliers together and thus match demand and supply. Markets set prices, provide liquidity, monitor transactions and allow for search and investigation. Electronic markets basically fulfill the same function just that they are augmented by an enhanced transparency, a literally unlimited transaction volume and a very low transaction cost.

2.3 Economic Significance of Electronic Marketplaces

Market Coordination

After a general demarcation line between traditional and electronic markets was drawn, this section tries to explain the business rational behind electronic marketplaces from a macroscopic point of view. As it will be covered in detail later, the price of a good is the sum of its production cost, the transaction cost for market coordination and a margin the producer wants to earn. The framework that will be introduced later in this work, amongst others, aims at reducing the transaction cost by providing more efficient forms of market coordination. Reducing the transaction cost coherently reduces the overall price of the product at which the supplier can give an offer. If the price is lower, the supply curve hits the demand curve at a lower angle resulting in a new market equilibrium (Brickley et al., 2006) with a lower price and a higher quantity (Figure 2).

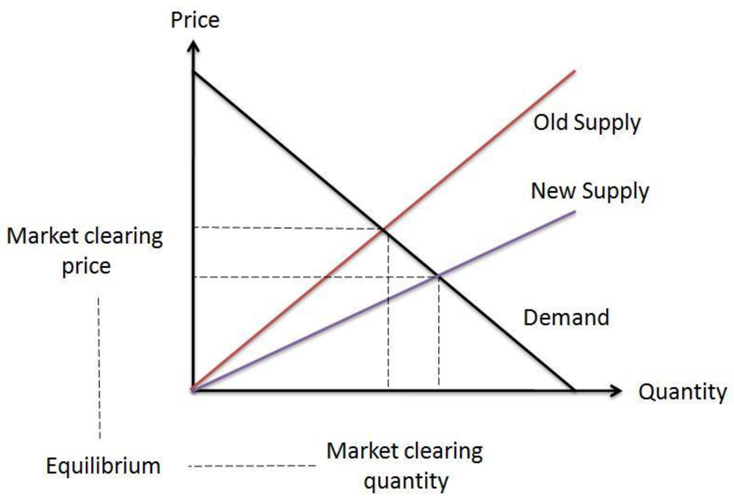


Figure 2: A new Market Equilibrium with lower price and higher quantity (Brickley et al., 2006).

Price - Quantity

Since in the new price-quantity combination a higher quantity could be gained by lowering the transaction cost, the margin of the supplier remains still the same. However, since the supplier sells more products now, his overall margin also increased. Additionally, more buyers that were interested in buying the product but were not willing to pay the old price are also able to acquire the product.

Welfare

To estimate the impact of the new supply function on the overall economy, usually a measure is utilized that is referred to as "welfare". Welfare is defined as the aggregation of the utility of single individuals or groups (Varian, 1992). Graphically seen, welfare is represented by the triangle between the demand and the supply functions (see Figure 3).

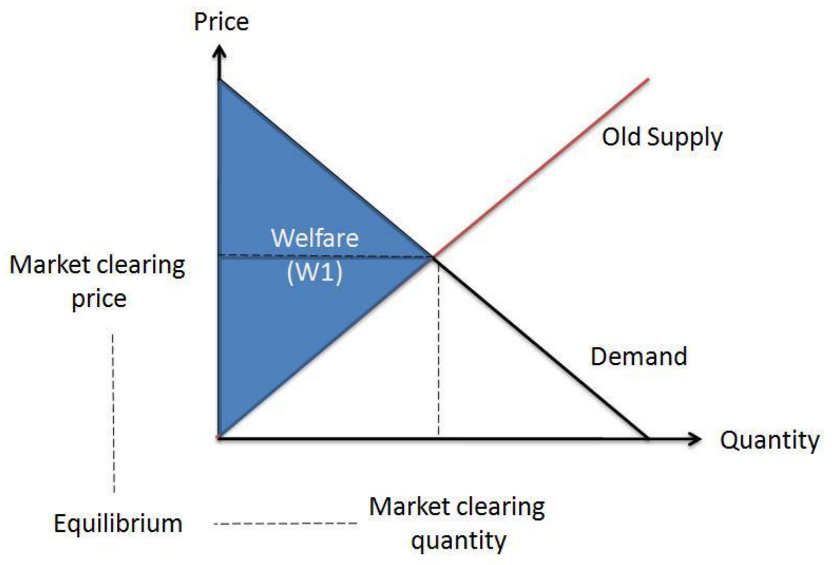


Figure 3: Welfare is defined as the triangle between the supply and the demand function (Varian, 1992).

If the supply function now tilts to the right as already shown in Figure 2, consequently the area of the triangle that defines the welfare in Figure 4 (W2) increases as compared to the old welfare (W1) in Figure 3. Hence, electronic trading platforms can theoretically help increasing the overall welfare of an economy by simply reducing the transaction cost.

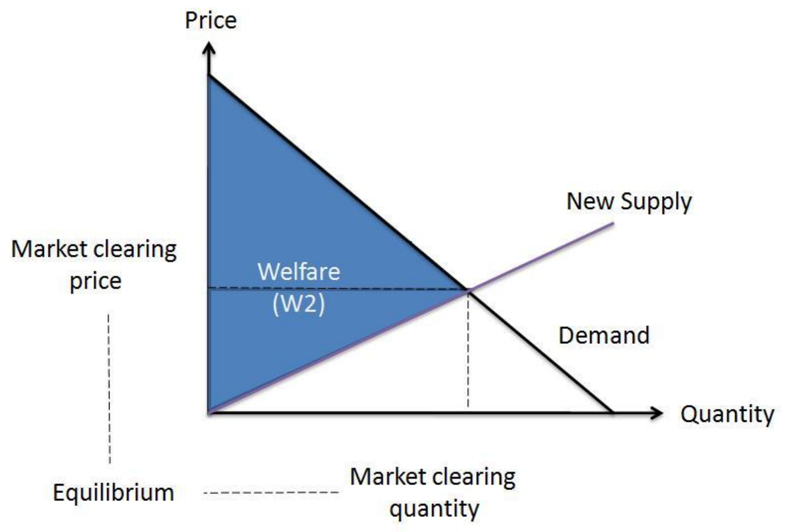


Figure 4: A lower price tilts the supply function to the right and thus increases the area that symbolizes the welfare (Varian, 1992).

Quick Review- Economic Significance of Electronic Markets

Electronic markets can help to provide more efficient forms of market coordination by lowering the transaction cost. The price for a good is determined by its production cost, the transaction cost and a margin for the producer. By simply reducing the transaction cost, the benefit of electronic markets lies in an overall lower price for the goods that are traded. Not only do the buyers benefit from lower prices, but also the suppliers as they can sell their products at a higher quantity. The result is in an increase in welfare.

2.4 Electronic Market Types

e-Market Types

As the last sections gave a definition of the term "electronic market" and demonstrated the potential economic impact of those, this section will discuss different types of electronic markets. Timmers (1999) refers to systems and applications that support Business-to-Business (B2B) transactions as B2B electronic commerce. He furthermore discriminates electronic commerce systems between electronic shops and electronic marketplaces, depending on the relation of buyers and sellers. In case of the buyer-seller relationship being one-to-many, the e-commerce application is considered an electronic shop (e-shop).

e-Market Classification

Electronic marketplaces have been classified in recent literature by using different criteria, such as product offerings, focus, openness and services provided (Oppel et al., 2001). Furthermore, they subdivide product offerings into the four categories: direct and indirect goods as well as services and capital goods. Direct goods tend to be industry-specific and are thus usually traded in industry-specific electronic B2B marketplaces, so called vertical marketplaces. Indirect goods, in contrast, are needed in numerous industries and can thus be found on horizontal marketplaces, which are not bound to servicing specific industries, but rather aim at fulfilling a specific function in an enterprise (Stearns Sgarioto, 2000).

Focus

The focus of an electronic marketplace falls into three categories, namely buyer-biased, seller-biased or neutral (Berryman, 1998). Biased marketplaces either focus upon buyers or sellers, depending on who gains the bigger advantage. Neutral marketplaces, on the other hand, are usually operated by third parties and favor neither buyers nor sellers.

Openness

The criterion openness differentiates electronic marketplaces into private and public. Whereas private marketplaces hold the right for the organizer to grant or refuse access to their system, public marketplaces do not underlie those restrictions and allow all users to perform transactions on them. Since private marketplaces are usually set up by larger enterprises or conglomerates, they are typically biased towards the side that set it up.

Transaction Phases

The services they offer to their users (EBM Services) are the last criterion Oppel et al. (2001) distinguish electronic marketplaces upon. Since the variety of possible services, however, is very large, Krähenmann (1994) utilizes the three phases of a transaction (information, negotiation, delivery) to find a useful frame for categorization (see Figure 5).

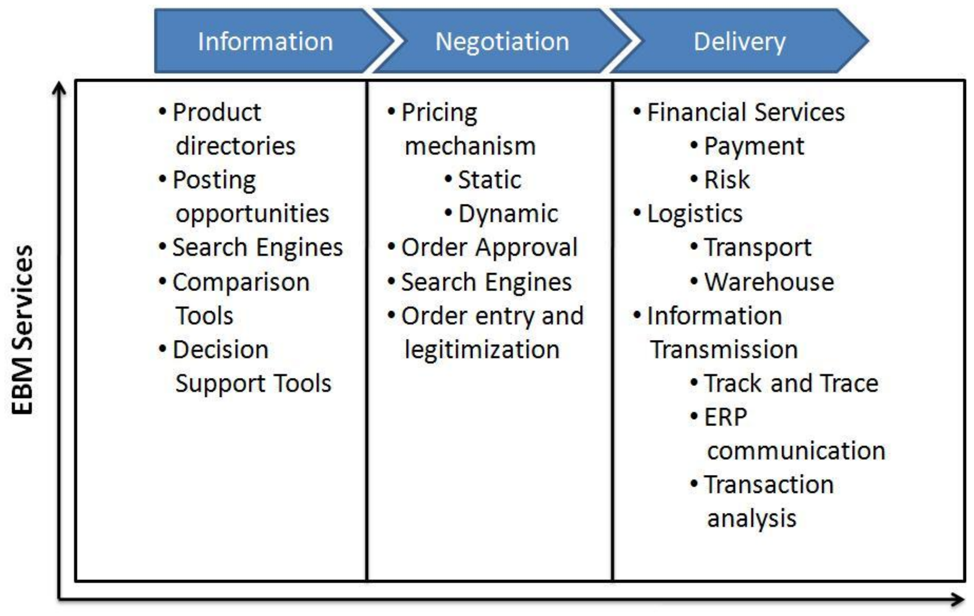


Figure 5: Transaction phases and their EBM Services (Krähenmann, 1994).

Quick Review – Electronic Market Types

Electronic Markets can be categorized into buyer-biased, seller-biased and neutral. Biased marketplaces focus upon who gains the bigger advantages. Neutral marketplaces do not favor any party. Furthermore, markets can be classified into private and public. The last criterion markets can be distinguished upon are the services they offer. In Kraehenmann's model, every service is assigned to a transaction phase.

2.5 Value Generation

Even though previous sections have already discussed value-related consequences of electronic marketplaces from a macroscopic perspective, this section will now put the focus upon a more microscopic analysis of how electronic commerce applications generate value.

On the buyers' side, the value of using electronic marketplaces lies in reducing purchasing cost. Purchasing cost compromises of the actual price of the product plus transaction cost. Transaction cost is defined by Williamson (1985) as "frictions" between the systems doing transactions. Malone et al. (1989) wrote in their article that electronic markets have an impact on transaction cost by "reducing the costs of negotiating and consummating deals and by helping buyers to find the best supplier" additionally to eliminating "paper handling and clerical work associated with making a purchase". Electronic marketplaces also reduce transaction cost by helping to find more qualified suppliers in a shorter time, which ultimately leads to strategic advantages. Strader & Shaw (2000) revealed that prices in electronic marketplaces are usually lower than in traditional markets, which might be an outcome of the better availability of information.

Value Generation

Although a number of business models for electronic B2B marketplaces exist with each of them implementing different ways of how value is generated, Oppel et al. (2001) suggest to distinguish into four prevalent models for value generation: Catalogue (or supply) consolidation, aggregation, market making and integration. Referring to Oppel et al. (2001), catalogue consolidation provides "value to the buyer by enabling one-stop shopping through the combination of catalogues from various suppliers". Aggregation means that demand within and across enterprises is combined to leverage buying power and reduce prices to, for example, establish buying consortia (Kaplan & Sawhney, 2000). Market making contains mechanisms that allow buyers and sellers to trade products, usually in a real-time pricing atmosphere (Oppel et al., 2001). The last model for value generation, integration, provides integration services by linking user back-end systems through data clearing (Kerrigan et al., 2001).

Value Generation Models

Even though each of the above described models for value generation can be used independently, Oppel et al. (2001) also state that every possible combination of the models is also valid.

Quick Review – Value Generation
"Catalogues" generate value by enabling one-stop shopping. The "Aggregation" model provides value by leveraging buying power and reducing prices. Finally, market making generates value by allowing buyers and sellers to trade products in real-time.

2.6 Transactions and Transaction Cost

After having discussed different value generation models above, the focus will now lie upon buyer-supplier transactions and it will be examined which electronic marketplace model is being used. Referring to Williamson (1985), transactions occur, when goods or services are transferred across a technologically separable interface. Thereby, buyers and sellers choose appropriate strategies for executing a transaction, depending on its characteristics (Oppel et al., 2001). Those strategies vary in the intensity of the relationship into which the transacting partners are engaged. A relationship can be defined as a "sequence of transactions between one supplier and one buyer which does not happen by chance (Plinke, 1997), but is in fact intended to continue for a longer time" (Gemünden, 1990).

Strategies

Referring to Bichler (2001), the price of a product can be considered as a combination of three elements: production costs, transaction costs and profit margin. Production cost takes into account the necessary physical or other primary processes to create and distribute the goods or services being produced. Transaction costs include the costs of gathering information, negotiating contracts and protecting against the risk of "opportunistic bargaining" (Bailey & Bakos, 1997). Lastly, the profit margin is what the producer earns (Bichler, 2001).

Product Pricing

Even though electronic commerce platforms cannot influence the production cost and the profit margin, they can help lowering the transactions cost and thus lowering the overall product price. Later in this work, a design for an electronic commerce platform will be proposed that is capable of integrating all possible products and services with each other in one market model. Thereby, the design of the Electronic Business Foundation (EBF) describes a unified representation of goods and services in the product model that is wrapped by a system of vertical and horizontal processes necessary to perform commercial transactions over the web (Figure 6).

The EBF

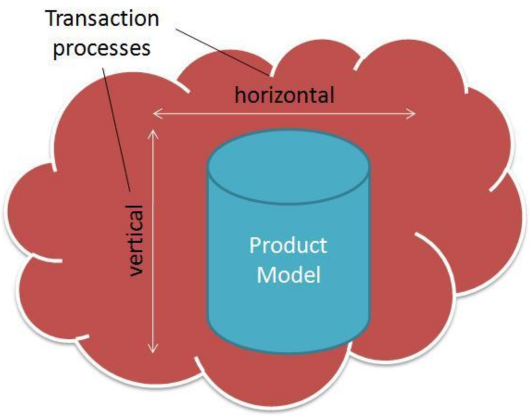


Figure 6: The EBF model consists of a generic product model that is able to describe arbitrary goods and services as well as transaction process that support the market making mechanism.

The power to integrate all possible products, such as normal goods and financial instruments, with each other in one marketplace can help to significantly lower the setup and maintenance cost for enterprises that want to trade heterogeneous products over one unified platform. Ultimately, the benefit of the enterprises would also reach the consumers, since the suppliers are now able to sell their items at lower prices resulting (from an economical perspective) into a "Pareto Improvement"[3] (Brickley et al., 2006).

Furthermore, the EBF allow for offering complementary, but structurally totally different products together in one electronic commerce solution. The author of this work believes that the EBF can not only help reducing setup, maintenance and transaction cost of enterprises, but could also serve as the foundation for totally new markets or trading experiences. A concept called "Reverse Trading", for example, describes how, in contrast to the traditional trading direction, buyers would advertise a product they want to buy and suppliers would apply to the buyers for a bargain. From the list of suppliers that are able to deliver the requested product, the buyer would pick the one he likes best or that offers the best price.

[3] The term "Pareto Improvement" origins from game theory research and stands for an improvement of an individual's utility (or situation) without negatively affecting the utility of other participating individuals.

Quick Review – Transaction and Transaction Cost
Even though electronic marketplaces cannot influence the production cost and the profit margin, they can help lowering the transaction cost. A lower transaction cost would also result in a lower total price for goods. Furthermore, by integrating all possible products with each other, the EBF could significantly help to lower the setup cost of the supplier and thus reach a broader customer spectrum.

2.7 Forward vs. Reverse Trading

Forward Trading

The contemporary understanding of how consumers find the product they are looking for and the supplier they want to bargain with has grown with the evolution of the human being as a very fixed and manifested pattern. If an individual is searching for a new item that it wants to buy, this pattern is best described through the following process:

1) Individual experiences a need for a product or service
2) Individual starts researching about the price of the product or service at various suppliers
3) Individual decides for the product or service of the supplier that it personally likes best (influenced by price etc.)
4) Individual enters a bargain with the supplier and obtains the product or service in exchange for a liability or funds

The above process is marked by a very strong pro-activity of the consumer towards the actual bargain. What that means is that regarding the supplier's side all he usually needs to do is offering and advertising the products or services that he wants to sell. The rest of the engagement will be covered by actions of the consumer: The consumer researches for products and suppliers (in magazines or on the internet), the consumer contacts the supplier and/or visits his shop and the consumer negotiates the price with the supplier, since it is only in his interest to spend as less money as possible.

The remainder of this section will in addition to the old and manifested trading pattern introduce a new one that describes the opposite – the pro-activity focus upon the supply side. Hereafter, the manifested trading pattern will be referred to as "Forward Trading", whereby the contracting pattern will be referenced as "Reverse Trading".

Reverse Trading

The concept of Reverse Trading proposes that in order to find the product or service as well as the supplier an individual wants to bargain with all the individual needs to do is announce or advertise its interest in the product or service on, for example, an electronic platform. Consequently, suppliers that are able to deliver the requested product or service receive a note from the system about a new bargaining opportunity and apply at the consumer for a bargain by giving a quote at which they are willing to sell the product or service. Finally, the consumer decides (from the list of suppliers that have applied for a bargain) for his favorite supplier that might even offer the best price. Lastly, the payment is completed and the product or service is delivered. The process could contain the following:

1) Individual experiences a need for a product or service
2) Individual opens a request for a product or service on, for example, an electronic platform
3) Suppliers apply at the individual for a bargain by providing a quote
4) Individual enters a bargain with its favorite supplier and obtains the product or service in exchange for a liability or funds

From a customer's perspective, Reverse Trading allows for a totally new trading experience. Individuals that wanted to buy a product would not need to spend time on researching about products and suppliers anymore, but could rather simply announce that they have a demand to be fulfilled. The remaining action would lie on the side of the suppliers that could potentially fulfill this demand. The key benefit about Reverse Trading is that customers would not need to understand the market behind the product or service they are interested in anymore in order to be able to evaluate the appropriateness of offers and prices. While the suppliers are competing with each other for a bargain with the customer, the customer would benefit from this competition by picking the supplier that offers the lowest price (or the best service). They would hence leave the price making and negotiation as it occurs in Forward Trading to the competing suppliers.

The Electronic Business Foundation, due to its openness and flexibility, will support Forward Trading as well as Reverse Trading. Thereby, the EBF makes use of both models (from a process perspective) being very similar. Just the roles of customers and suppliers are upside-down.

Quick Review – Forward vs. Reverse Trading

While traditional Forward Trading typically shows a strong pro-activity on the consumer side, Reverse Trading describes a shift in that towards the supplier's side. Thereby, the consumers would merely need to announce a demand on an electronic platform while suppliers that could potentially fulfill this demand would apply for a bargain at the consumer. The key benefit for consumers would lie in them not necessarily having to understand the market anymore in order to get the best price and the best product. This would be achieved automatically by the suppliers competing with each other for the bargain.

2.8 Traded Products

After different pricing mechanisms and directions of trade have been discussed, it will next be shown how electronic marketplaces influence buyer-supplier relationships.

Search Goods

Oppel et al. (2001) hypothesized that the value generating models (presented earlier) work best in purchasing situations "where the market governance model has been preferred" (Oppel et al., 2001). They further suspect that "the electronic marketplaces so far have not been able to attract business that is usually conducted in a more relational manner for several reasons" (Oppel et al., 2001). The earlier discussed internet-based value generation mechanisms tend to favor situations in which companies would revert to market mechanisms for coordinating their transactions. This results from the fact that buyers that are conducting transactions via an electronic marketplace need to be able to purchase the product without seeing it. Not seeing the product generates a risk for the buyer, which he is likely to only take for products that are easy to describe or standardized and have low asset specificity (Benjamin & Wigand, 1995). Those products that are characterized by the fact that the decisions of their purchase can be based on search qualities, which can be completely evaluated before the transaction, are called "search goods" (Adler, 1996). Oppel et al. (2001) state that considering those attributes, search goods "describe transactions, which are marked by low transaction cost and thus favor market coordination anyhow (as opposed to, for example, hierarchical coordination (Benjamin et al., 1986). Koppelmann et al. (2001) add that cheap and normal products lend themselves more to electronic procurement than innovative or specialized products.

Methods of Communication

Although above argumentation might have been true for the times the respective works were authored, it is part of this thesis to suggest a reevaluation of the situation. It will later be shown how utilizing modern information and communication technologies, even more specific and innovative products can successfully be traded on electronic marketplaces. The system that is to be designed will augment a pure textual product description with more intuitive methods of description (e.g. an image gallery). Ebay has already shown how significantly image galleries can support the perception and understanding of a product by a customer explicitly targeting the visual sense. An even more interactive way of virtual product description could contain a 360° tour around the product that gives the user the feeling to move around the item. Finally, a video about the product could demonstrate how it is used with a narrator providing a much higher detail of information than a pure textual description. A more in-deep analysis of various methods, such as the face-to-face shop, that target a more natural support of product data will be covered in chapter three.

Quick Review – Traded Products
Search goods are goods which are characterized by the fact that a decision for their purchase can be completely evaluated before the transaction is committed.

As their asset specificity is very low, goods that are traded on electronic marketplaces are typically search goods. Since this perception originated during the 90's, it is also the goal of this work to show that, due to the development that has taken place in the domain of e-commerce, it is not valid any longer.

2.9 Summary

This chapter aimed at providing a general introduction to the basic principles of electronic commerce. While showing the differences between classical and electronic marketplaces, it was discussed which benefits the latter ones can offer. Electronic marketplaces were categorized after several criteria, whereby value generation-related aspects were also taken into account. It was shown how electronic marketplaces can contribute towards increasing a society's welfare by lowering the transaction cost of trade.

A new way of organizing trade was introduced and referred to as Reverse Trading. The benefit of Reverse Trading over traditional Forward Trading is that consumers do not need to understand the market in order to find the best goods at the best prices. This is guaranteed due to the nature of Reverse Trading as suppliers compete with each other for a bargain with the consumer.

The following chapter three will contain an in-deep analysis of requirements and methods to support product data of any kind in order to be able to represent arbitrary goods and services in one unified model. This model should then serve as conceptual foundation for a platform on the internet that is so flexible that appropriate customization can transform it into an online shop for a retailer, an auction platform for consumer-to-consumer transaction or a platform that allows for trading financial instruments, such as bonds, futures or even consumer-to-consumer credits.

In addition to the product model that enables this flexibility, the transaction processes will augment the traditional trading experience by providing stronger services to make the electronic business more trust-worthy and useful. For very specific goods or services that are not easy-to-trade, electronic face-to-face shops, for example, could establish a video conference between the buyer and the seller allowing the virtual customer (buyer) to ask questions to the virtual shop clerk (supplier) over a life-stream. For items that have a strong requirement for visual perception, 3D or video tours around the item could help providing better information to the customer. Data mining tools could support the automatic generation of a more intuitive and comprehensive product description or FAQ section from, for example, user inquiries and hence improve the overall information flow between buyer and supplier. Ultimately, it will be the goal of the transaction process modeling to show that the statement of Oppel et al. (2001) that only very simple and general products are traded on electronic marketplaces cannot stand a reevaluation any longer.

3 Towards a Unified Product and Process Model

3.1 Overview

After chapter two has given an introduction to electronic commerce and has outlined its fundamental concepts, this chapter is focused upon the design of a product and process model that is flexible enough to even support Reverse Trading (see 2.7 Forward vs. Reverse Trading). Additionally, this model (later the EBF Product and Process Model) is not only tailored towards enabling transactions for products of a certain type, such as end-consumer goods. The model introduced in this chapter will rather be flexible enough to represent transactions on products of any kind, no matter whether these are normal end-consumer goods, financial instruments, insurances or dentist appointments. Ultimately, it will be part of this chapter to prove that the statement of Oppel et al. (2001) cannot stand a reevaluation any longer. Oppel et al. (2001) say that only very simple and general products are traded on electronic marketplaces. The EBF will make an attempt to show that this might not be valid anymore.

This chapter will start with an overview of existing electronic marketplaces and will try to identify common and special properties of those. In the next instance, the requirements analysis will formalize the specific requirements to the product as well as to the process model, which will later be designed within separate sections. Those sections will put the focus upon the implementation of the formal requirements stated in the requirements analysis. The last section will discuss how to augment the process model with an integrated security mechanism.

3.2 An Overview of Existing E-Marketplaces

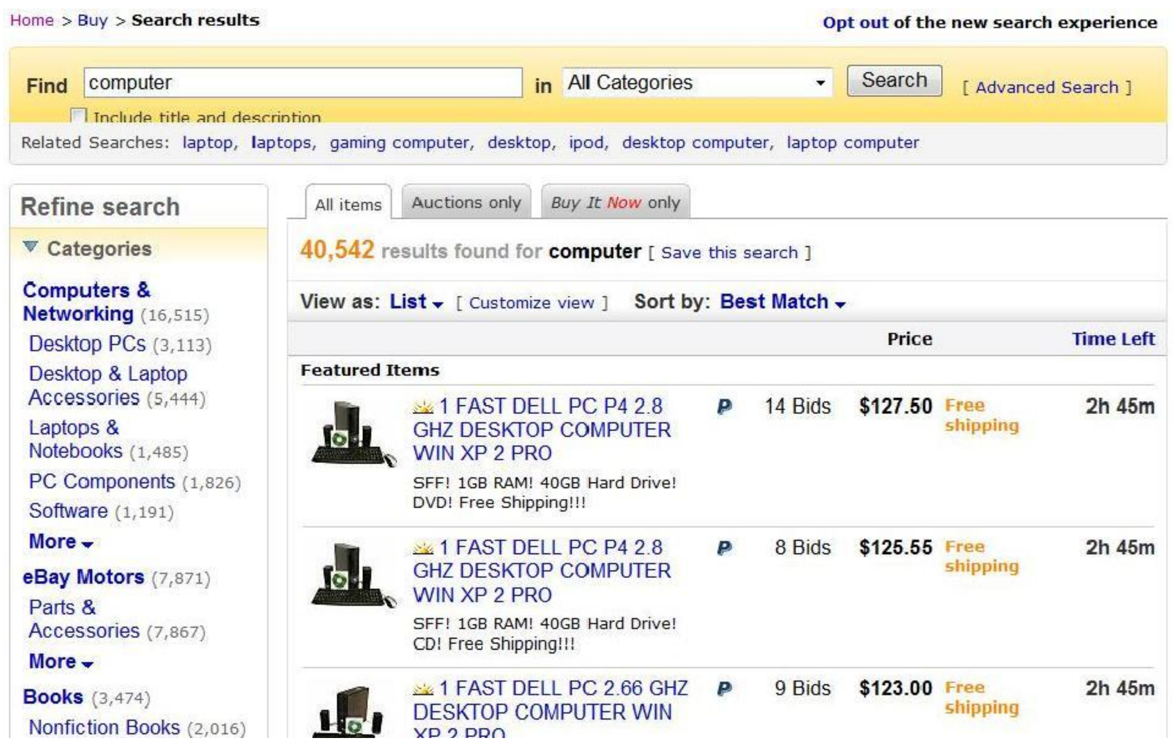
The goal of this work is to find a generic e-Commerce application that is able to trade all possible products and services, whether on an auction-based or a shop-based platform. The next step will be to examine a few selected already existing e-Commerce platforms for their functionality and processes.

eBay

Ebay is an online auction platform where end-consumers can buy or sell goods from other end-consumers (C2C). In order to be able to trade on eBay, a new user account has to be created. Afterwards, the user is free to buy or sell products as he wishes. Ebay charges a service fee on the supplier, when a new offering was made and again when an offering was sold. The buyers on the other side stay free of any eBay charge.

When creating a new offer, the supplier has the possibility to appropriately categorize the item, customize the values for a predefined set of attributes, add a textual product description and also add a picture for the item or even a whole picture gallery. He can further specify the start date and the duration of the offer as well as set a starting price. If the supplier wishes to not sell the product in an auction-based manner, he can set a fixed price at which the item is to be sold. In addition to the product-specific information, the supplier can also specify logistic-related information, such as the transportation cost or estimated shipping time. After the product information was inserted, the offer is launched to the eBay marketplace where buyers can start bidding on it.

To find the product they are looking for, the buyers can either enter a search term in the eBay search engine or browse through a hierarchy of categories. Once the item was found (what usually doesn't take more than a few clicks), the buyer has the possibility to ask the supplier questions via email. If the buyer decided on whether to participate in the auction, he can enter his maximal bid or buy the item directly at a specified fixed price, if such an option exists (see following graphic).


Home > Buy > **Search results**

Opt out of the new search experience

Find computer **in** All Categories Search [Advanced Search]

Include title and description

Related Searches: laptop, laptops, gaming computer, desktop, ipod, desktop computer, laptop computer

Refine search

Categories

Computers & Networking (16,515)
Desktop PCs (3,113)
Desktop & Laptop Accessories (5,444)
Laptops & Notebooks (1,485)
PC Components (1,826)
Software (1,191)
More

eBay Motors (7,871)
Parts & Accessories (7,867)
More

Books (3,474)
Nonfiction Books (2,016)

All items | Auctions only | *Buy It Now* only

40,542 results found for **computer** [Save this search]

View as: **List** [Customize view] Sort by: **Best Match**

Featured Items

		Price		Time Left
1 FAST DELL PC P4 2.8 GHZ DESKTOP COMPUTER WIN XP 2 PRO SFF! 1GB RAM! 40GB Hard Drive! DVD! Free Shipping!!!	14 Bids	$127.50	Free shipping	2h 45m
1 FAST DELL PC P4 2.8 GHZ DESKTOP COMPUTER WIN XP 2 PRO SFF! 1GB RAM! 40GB Hard Drive! CD! Free Shipping!!!	8 Bids	$125.55	Free shipping	2h 45m
1 FAST DELL PC 2.66 GHZ DESKTOP COMPUTER WIN XP 2 PRO	9 Bids	$123.00	Free shipping	2h 45m

Figure 7: An auction for PCs on eBay.

In the next step, eBay will initiate the engagement between the supplier and the buyer by sending both of them a so called order confirmation email. The buyer will now have to complete the payment by choosing from a set of payment options. He can either pay by bank withdrawal, Paypal[4] or by picking up the item directly from the supplier.

Even though eBay has a very rudimentary fraud protection system, it does not offer by default any trust services. In case of a fraud incident, the buyer can rather apply at eBay for receiving back a certain part of the amount that was initially paid. Additionally, buyers have the possibility to evaluate suppliers after the transaction and thus influence their overall reputation.

[4] Paypal is an online service provider that allows for payments on the internet by charging/refunding the according bank accounts of the buyer/supplier.

Items that are allowed for trade on eBay are generally items of a real physical shape (concrete goods). Thus, it is not possible to trade more abstract products, such as financial instruments or stocks. Also, service offerings cannot be posted.

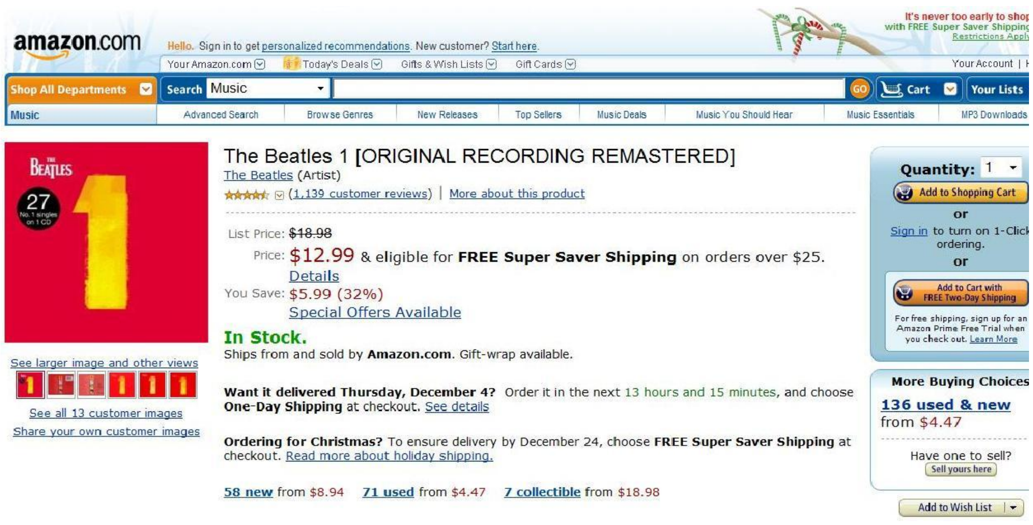
The communication between buyer and supplier is limited to the exchange of email messages. Options that would further support the establishment of a trust relationship between buyer and supplier, such as text chat or video conference, are missing. The product description also does not allow for showing embedded videos of the product.

In addition to the fixed pricing mechanism, goods on eBay can be traded in auctions. This auction, however, is always a normal forward auction where the one buyer wins who has offered the highest price. Other auction types, such as Dutch auctions or Sealed first-price auctions (Krishna, 2002) are not supported.

Amazon

Another online marketplace is **Amazon**. Headquartered in Seattle (USA), Amazon originally started as a virtual storefront for books. They quickly extended their core business to music CDs and video DVDs and since recently also offer renting and downloading whole movies over the internet.

In addition to this core business, Amazon offers the functionality of their platform to be used by other, external vendors to create their own online shop on the so called Amazon Marketplace. Using the Marketplace API[5], vendors can customize the categorization and the description of the product, provide several images as well as set the price and detail the shipping cost. As opposed to eBay, it is possible to embed videos in the product description that might sometimes provide better information to visualize the nature of the product than just plain text. Additionally, products can be evaluated by customers for their quality.



Hello. Sign in to get personalized recommendations. New customer? Start here.

The Beatles 1 [ORIGINAL RECORDING REMASTERED]
The Beatles (Artist)
(1,139 customer reviews) | More about this product

List Price: ~~$18.98~~
Price: $12.99 & eligible for **FREE Super Saver Shipping** on orders over $25. Details
You Save: $5.99 (32%)
Special Offers Available

In Stock.
Ships from and sold by **Amazon.com**. Gift-wrap available.

Want it delivered Thursday, December 4? Order it in the next 13 hours and 15 minutes, and choose **One-Day Shipping** at checkout. See details

Ordering for Christmas? To ensure delivery by December 24, choose **FREE Super Saver Shipping** at checkout. Read more about holiday shipping.

58 new from $8.94 71 used from $4.47 7 collectible from $18.98

See larger image and other views
See all 13 customer images
Share your own customer images

or
Sign in to turn on 1-Click ordering.
or
For free shipping, sign up for an Amazon Prime Free Trial when you check out. Learn More

More Buying Choices
136 used & new
from $4.47
Have one to sell?

Figure 8: The Amazon web interface.

[5] API stands for Application Programming Interface.

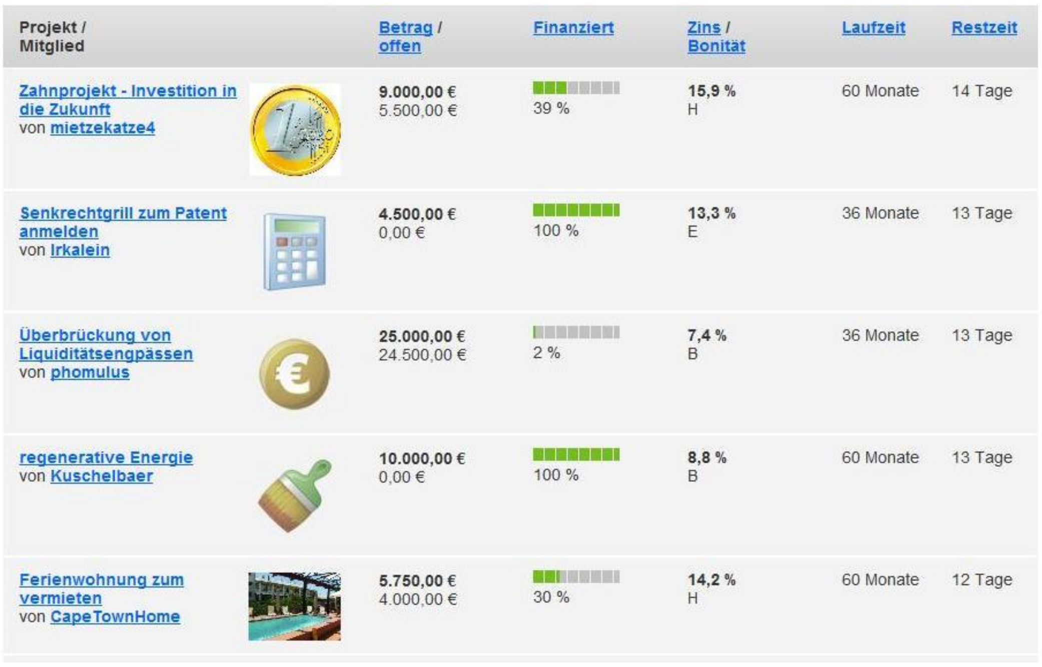
As compared to eBay, Amazon does not allow for an auction on a product, but rather offers all products at a fixed price. Third-party vendors pay a service charge to Amazon for being allowed to use their API and host the product descriptions on the Amazon servers. In addition to that, the products of third-party vendors are integrated with the Amazon search engine, so that it can happen that if a user searches for a book on “the history of coffee”, the search results contain an espresso maker.

External vendors benefit from the integration of their Marketplace shop into the Amazon payment system. Once a product was bought by a customer, the payment system handles the rest of the order approval and money transfer process.

Traded products on Amazon are, similar to eBay, restricted to real goods. Hence, services or financial instruments and credits cannot be traded. Fraud protection on Amazon is restricted to an evaluation of the supplier by the buyer, whereby this evaluation will be shown next to the name of the supplier in form of a number of stars. The communication between the buyer and the supplier is as simple as on eBay and also limited to the exchange of email messages.

Smava

The last electronic marketplace that is going to be observed is **Smava**. Smava is a newly founded online platform where people can offer and find consumer-to-consumer credits. Thereby, the potential delinquents can create a new project and can offer a contract period as well as an interest rate on the project sum. Creditors who are interested in financing the project can then decide with how much money they want to participate (see Figure 9).



Projekt / Mitglied	Betrag / offen	Finanziert	Zins / Bonität	Laufzeit	Restzeit
Zahnprojekt - Investition in die Zukunft von mietzekatze4	9.000,00 € 5.500,00 €	39 %	15,9 % H	60 Monate	14 Tage
Senkrechtgrill zum Patent anmelden von Irkalein	4.500,00 € 0,00 €	100 %	13,3 % E	36 Monate	13 Tage
Überbrückung von Liquiditätsengpässen von phomulus	25.000,00 € 24.500,00 €	2 %	7,4 % B	36 Monate	13 Tage
regenerative Energie von Kuschelbaer	10.000,00 € 0,00 €	100 %	8,8 % B	60 Monate	13 Tage
Ferienwohnung zum vermieten von CapeTownHome	5.750,00 € 4.000,00 €	30 %	14,2 % H	60 Monate	12 Tage

Figure 9: Smava’s consumer-to-consumer credits.

Once the offer has officially ended, the platform will take care of transferring the partial sums of the creditor to the bank account of the delinquent.

Smava tries to control the risk to the creditors of financing a project from which they will never get their money back by proposing to create a portfolio rather than putting all money in one project. Also, it will validate the liquidity and the credit history of all its members through so called Schufa[6]-checks.

A possible feature augmentation that could help to increase the trust into the platform could be a video conference that is held between the delinquent and the creditors before the financing project starts. This would give the creditors the opportunity to ask live questions and thus better understand the project and it would help the delinquents to attract more investors. Apart from that, having seen and talked to the person one is lending money to would certainly have beneficial outcomes with respect to trust and trustworthiness.

Quick Review – An Overview of Existing E-Marketplaces

The following Table 1 summarizes the above introduced electronic marketplaces eBay, Amazon and Smava by comparing them to a list of common features.

	Ebay	Amazon	Smava
Traded Items	goods	goods	money
Auction	yes	no	yes
Auction Type	forward	n/a	forward
Reverse Trade	no	no	no
Communication			
E-Mail	yes	yes	yes
live chat	no	no	no
video-conferencin	no	no	no
Fraud protection	yes/basic	yes/basic	yes/basic
Payment system	yes	yes	yes

Table 1: Properties of existing e-commerce platforms.

3.2 Requirements Analysis

After an overview of three existing e-commerce platforms and a high-level view of their properties were given, it will now be attempted to derive requirements for a more general product and process model that is capable of representing all possible goods and services in a unified manner. Thus, the product and process model should be able to reflect the trading of normal goods (similar to eBay and Amazon), but it should also be capable of enabling the trade with services (e.g. plumbing or cleaning service) as well as trade with more abstract goods, such as financial instruments, money or stocks (similar to Smava.de or Onvista.com).

[6] Schufa is a German institution that offers information about the credit history of potential delinquents to creditors.

In addition to the properties listed in Table 1, the new product and process model should also be able to customize the direction of trade (see 2.7 Forward vs. Reverse Trading). Furthermore, it should support various pricing mechanisms, such as fixed or dynamic (auction) pricing.

To be able to provide a better understanding for both, the product model and the transaction processes will be covered separately in this chapter.

Product Model Requirements

The modeling of a general product model that can easily store each and every product description, no matter whether it is a concrete or an abstract product, will be covered in 3.3 The Product Model. Regarding Table 1, the product model will contain the fundamental data description that is necessary to perform any transaction process. The product model does not have a direct relation towards supporting the properties that were summarized in Table 1 as they seem to be too high-level. However, the next layer on top of the product model, the transaction processes, will consider the Table 1 properties in a more specific manner and finally implement them. In order to achieve the general goal of being able to store each and every product description in the product model, it in particular will have to consider the following points:

Flexibility: The product model must be flexible enough to store each and every possible product, no matter whether the product has a concrete physical appearance or is of a more abstract kind.

Customizability: Product attributes should not be statically defined in the product model itself, but should rather be specifiable, when a new product is created from the product model.

Structure: The product model should be able to structure product attributes into hierarchies.

Size: The number of product attributes should not be limited.

Complexity: The product model should be very simple, so that is easy to edit by non-professionals and also easy to augment by, for example, a graphical representation.

Extensibility: The product model should be extensible in a way that, for example, new product models can be derived from existing ones.

Browsability: The product model should allow for external browsing operations, so that search operations through a product’s whole hierarchy of attributes can be implemented.

Dynamicity: The product model should allow for assigning static values to a product attribute field, but should also be capable of acquiring the value on demand or allow for updating or modifying it.

Process Model Requirements

The modeling of the transaction processes that wrap the product model and represent the actual processes that are involved in a trading transaction will be part of 3.4.Thereby, the process model will have to consider the following points:

Completeness: The process model should capture all relevant processes that are participating in a transaction, but on a rather general level.

Extensibility: The processes in the process model should be extensible, so that specific behavior can be added to a process, but from the outside the behavior of the process itself still remains the same.

Direction: The direction of the transaction should be considered as part of the process model. The process model should support both Forward and Reverse Trading.

Communication: The process model should provide a good communication infrastructure that is able to leverage modern information and communication technology.

3.3 The Product Model

The previous section formulated in detail the requirements to the product model as well as the model of transaction processes which will later serve as the basis for the design of a unified framework. This section will cover the theoretical development of a product model that aims at incorporating all the requirements of the Requirements Analysis.

Existing Standards

But prior to the development of the product model an answer to the following concern is given: Why can existing standards, such as STEP, not be used or enhanced, rather than developing a product model from scratch? The answer lies in the nature of standards itself. While standards, such as STEP (Pratt, 2002) typically aim at describing a standardized way of executing processes or structuring data (in this case exchanging product data), the product model that is to be designed in this chapter will have to fulfill a different task. Instead of describing patterns of how to structure data and communicate it to other systems, the target product model rather gives the applicant a tool to define a product by himself. Using a product description language the user can define a product that includes only those properties and property handlers of a product that are really relevant. Standards that deal with product definition and modeling, contrarily, require a customization of pre-defined properties in a list of fields and thus, no matter how comprehensive, always set the product modeling a certain limit. Having this limited flexibility, due to the goal of the product model to be able to reflect each and every product or service, is hence not acceptable.

Product Description Language

As it will be detailed shortly, the description of a product in the targeted product model is done in a product description language, which in reality is an XML dialect. Even though other approaches towards the design of a product description language would have been possible (e.g. CSV = Comma Separated Value), the markup language XML was chosen. The reason for that is that the product model, due to the mentioned hierarchies in the Browsability requirement, perfectly fits the target domain of XML (Evjen et al., 2007): the description of structured data.

Product Model and Product Description

The general assumption which the product model underlies is that it consists of a set of product descriptions. A product description reflects the product that is to be described and consists of product attributes which are ordered in a hierarchy. All product attributes together and the hierarchy they are ordered in give a complete product description. Furthermore, new product descriptions can be derived from existing ones. The following Figure 10 demonstrates the relation between the terms "product model", "product description" and "attribute".

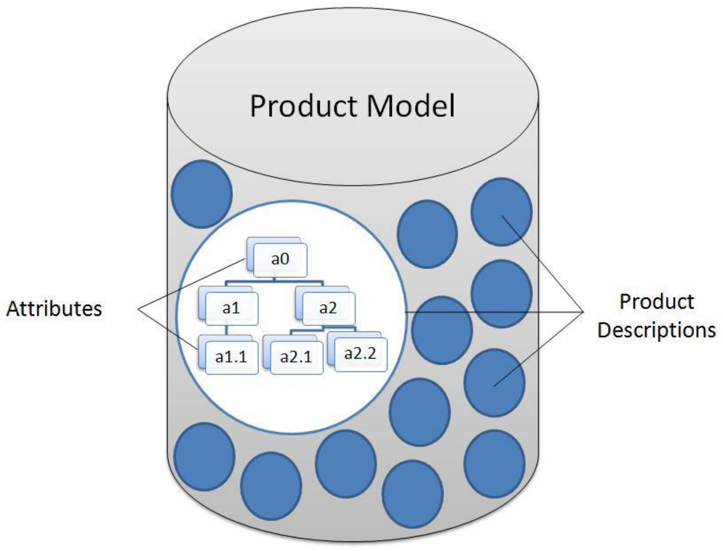


Figure 10: The complete product model consists of a set of product descriptions which are defined by a hierarchy of attributes.

To be able to formally express a product description, the remainder of this section will cover the design of a product description language (PDL). The design of the PDL utilizes and extends existing standards while aiming at providing the ability to express a product description in a natural and easy to understand way. Furthermore, it encompasses all the requirements that were defined in the Requirements Analysis.

Attributes

In the first instance and as the starting point for every later refinement, the PDL needs a way to express the name and value of a single attribute. Therefore, an attribute tag that serves the purpose of naming an attribute as well as storing its value is introduced:

```
<attribute>
      value
</attribute>
```

In above schema, the string "attribute" refers to the name of the attribute, whereby "value" stores its actual value. With the introduction of the attribute tag, it is already possible to fully describe one attribute of a product. Utilizing the example of a car, an attribute describing the make of the car could look like this:

```
<make>
      BMW
</make>
```

By introducing a rule that allows for adding an unlimited number of attributes to a product description, it is now possible to store more than only one attribute:

```
<attribute1>
      value1
</attribute1>
<attribute2>
      value2
</attribute2>
<attribute3>
      value3
</attribute3>
```

Continuing the example of the car, a more detailed product description could contain the following:

```
<make>
      BMW
</make>
<cost>
      100.000
</cost>
<color>
      blue
</color>
```

Again, the number of attributes in a product description is not limited - a property that refers to the Size requirement of the Requirements Analysis section.

Attribute Hierarchies

Until now, the product model does not allow for the structuring of attributes, neither does it allow for the description of dependencies between them. In above versions of the product model all attributes lie on the same hierarchical level, which makes it difficult to describe dependencies. To solve this problem, the next version allows for attributes being put into a hierarchy, which would result into the following schema:

```
<attribute1>
      value1
      <attribute1.1>
            value1.1
      </attribute1.1>
</attribute1>
<attribute2>
      value2
```

```
		<attribute2.1>
			value2.1
		</attribute2.1>
		<attribute2.2>
			value2.2
		</attribute2.2>
	</attribute2>
	<attribute3>
		value3
		<attribute3.1>
			value3.1
			<attribute3.1.1>
				value3.1.1
			</attribute3.1.1>
		</attribute3.1>
	</attribute3>
```

A concrete instance of the above PDL schema could contain the following car data:

```
	<make>
		BMW
		<type>
			M6
		</type>
	</make>
	<cost>
		100.000
	</cost>
	<color>
		blue
		<rims_color>
			black
		</rims_color>
		<brake_color>
			red
		</brake_color>
	</color>
	<engine>
		V12 Bi-Turbo
		<power>
			500 HP
			<at_rounds_per_minute>
				5000
			</at_rounds_per_minute>
		</power>
	</engine>
```

The above description expresses that the car is of the make “BMW”, whereby the make attribute now contains an additional attribute storing the type of the BMW (“M6”). The color attribute also contains two additional attributes, whereby one describes the color of the rims (“blue”) and one describes the color of the brakes (“red”). The engine attribute is new and subdivided into two more levels of hierarchy. Apart from the value “V12 Bi-Turbo”, the engine attribute contains an attribute power (“500 HP”). The power attribute itself again contains a next-level attribute at_rounds_per_minute (5000) that details the rounds per minute at which the power is unleashed.

Goal Assessment

The latest version of the product model is nothing else than an application of the XML language features that is able to describe a product as a set of structured attributes (Evjen et al., 2007). Even though very simple, it already fulfills most of the requirements that were stated in the Requirements Analysis:

Flexibility: Since the product model does not define any structure or field to be set in the product description, but rather allows for adding the name of an attribute and its value, it is theoretically possible to store each and every product as long as the product can be completely subdivided into attributes (properties). Thereby, it is of no matter, whether the product that is to be described is a normal good, a service or a financial instrument.

Customizability: Since the PDL allows for adding any arbitrary attribute, each and every production description and hence the whole product model is completely customizable.

Structure: The PDL allows for adding hierarchically ordered attributes to a product description with arbitrary depth. From another perspective, the product description is arrangable in a tree-like structure.

Size: The number of attributes in the product description is unlimited. So is the depth of the hierarchy the attributes are arranged in.

Complexity: Since the PDL is an XML dialect, product descriptions are easy to read and to edit. Adding a graphical user interface for editing product descriptions hence does not require any advanced technological know-how. From an implementation point of view, existing XML libraries and APIs could certainly support the development of a graphical user interface.

Extensibility: Since the general approach towards the product model is object-oriented, there are no limits in deriving new product models from existing ones. Even though the detailed procedure of how to derive new product models from existing ones would depend on the actual implementation, generally said the rules of the object-oriented programming paradigm are applicable.

Browsability: Browsing through the product model means browsing through all the product descriptions it contains. Individual product descriptions themselves are again browsable, while the attribute (name) and value are strings that can be read and compared.

The only not yet covered required property to the product model is dynamicity:

Dynamicity: The product model should allow for assigning static values to a product attribute field, but should also be capable of acquiring the value on demand or allow for updating or modifying it.

Attribute Handlers

To allow for an attribute within a product description to be dynamically loaded, acquired from other sources or modified, the product model needs to be augmented by a language feature that is capable of expressing each and every dynamic behavior of single attributes. A new tag set is added to the existing PDL that wraps the beginning and the end of a so called attribute handler.

```
<!attribute_handler>
…CODE…
</!attribut_handler>
```

Between those two tags code that performs the desired operations on the attribute can be placed. It will be shown in later chapters how adding real code between the attribute handler tags performs technically every action or command that is expressible with today's modern programming languages. For the reason of having an abstract product description, it will for now be refrained from demonstrating an example using real code. Instead, pseudo code will be utilized to visualize the attribute handler in action.

```
<cost>
      100.000
      <!attribute_handler>
      GetCostInCurrency( currency )
      {
            If(  currency = "USD")
                  return GetAttributeValue() * 1.33;

            If( currency = "JPY")
                  return GetAttributeValue() * 125;
      }
      </!attribute_handler>
</cost>
```

The example of the attribute handler above demonstrates how the attribute tag next to its static value (100.000) also contains a description of how to obtain the cost in a certain currency. By querying the product model for the cost attribute, the currency parameter of the "GetCostInCurrency()" attribute handler identifies in which currency the cost is to be returned. In case of USD (US Dollars), the attribute handler will return the static attribute value (100.000) provided by "GetAttributeValue()" times the exchange rate of 1.33. In case of the cost for the vehicle being required in Japanese Yen (JPY), the static value would be taken times 125. In above example, the base currency is implicitly assumed as EUR (Euro).

With above attribute handler tag in the PDL, it is now possible to add an arbitrary set of attribute handlers to an attribute that, when called explicitly, will perform certain actions on the attribute. However, it would certainly also be of use, if it was possible to register an attribute handler as default attribute handler which is called, when the value of an attribute is accessed from within the product description (for a visual representation of this process please refer to Figure 11).

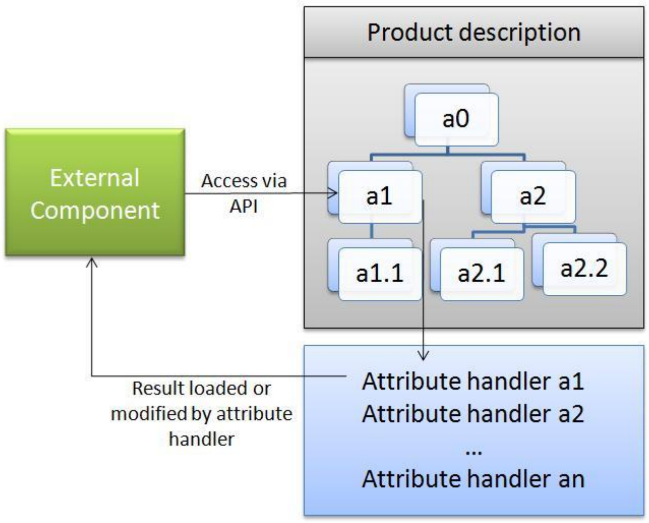


Figure 11: An external component accesses an attribute in a product description. The attribute's default attribute handler is invoked and its result returned.

Default Attribute Handlers

Therefore, another tag "attribute_handler_default" is added to the PDL. The following demonstrates its structure:

```
<!attribute_handler_default>
…CODE…
</!attribute_handler_default>
```

An example of a default attribute handler that on access acquires the latest price from the dealer's database over the internet could look like this in pseudo code:

```
<!attribute_handler_default>
GetPrice()
{
      GetLatestPriceFromTheDealerOverTheInternet();
}
</!attribute_handler_default>
```

The default attribute handler behaves the same way as normal attribute handlers do and is also explicitly callable. However, in contrast to the normal attribute handlers, the default attribute handler is called in background, when the value of an attribute is accessed. This dynamic behavior allows for attributes being acquired from an external source at the time the value is requested.

With the extension of the PDL by the attribute handler logic, the last requirement "dynamicity" is now also fulfilled.

Dynamicity: The product model allows for assigning static values to a product attribute field, but is also capable of acquiring the value of an attribute on demand (default attribute handler) and furthermore allows for updating or modifying it (normal attribute handlers).

The following Table 2 summarizes the requirements of the Requirements Analysis:

Requirement	Assessment
Flexibility	✓
Customizability	✓
Structure	✓
Size	✓
Complextiy	✓
Extensibility	✓
Browsability	✓
Dynamicity	✓

Table 2: Requirements Checklist for the product model.

Quick Review – The Product Model

The EBF Product Description Language (PDL) features an XML dialect that allows for segmenting a product description into attributes. Those attributes are expressed as a name/value combination and can be hierarchically ordered. Furthermore, the static content of attributes can be augmented with dynamic behavior by defining attribute handlers. Attribute handlers can be implemented in a modern object-oriented programming language and describe any transformation of the value of an attribute. The concrete implementation of the EBF Product Model, which will use a real programming language, will be part of chapter five.

3.4 The Transaction Processes

After the previous section has covered the design and development of the product model, this section is going to give a detailed description of the processes that participate in an online transaction. Due to the complexity of the model, this section is further broken down into two subsections, one to model the first phase in the process model and another one to model the second phase. Prior to the actual design, however, a discussion and evaluation of an existing process model for e-commerce is given. Potential drawbacks of the discussed process model will be added as requirements to the EBF Process Model, which is to be designed.

Transaction Phases

Chapter two discussed the basics of commerce and e-commerce and also stated already an attempt to classify a transaction on an electronic market into the individual phases “information”, “negotiation” and “delivery” (Krähenmann, 1994). According to Krähenmann (1994), each of the phases is supported by a certain set of EBM services (see Figure 12).

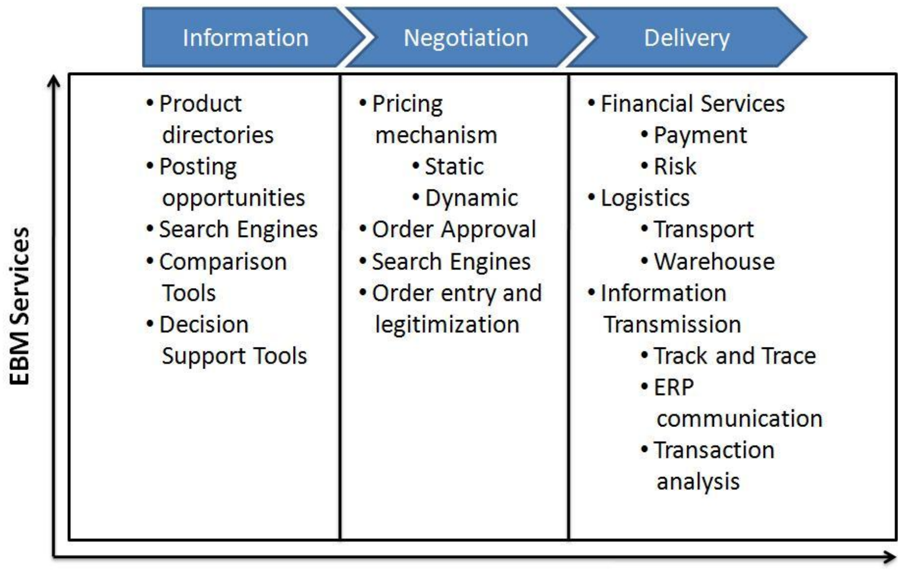


Figure 12: The Krähenmann process model (Krähenmann, 1994).

Information gathering in the information phase, for example, could include browsing through product directories or looking up products in search engines. Decision making could be based on the support of decision support and comparison tools that help the user choosing the alternative that apparently maximizes his utility.

Once the optimal alternative was found, according to Krähenmann's opinion, the negotiation phase begins with the determination of the price for the product. While online shops often set the price statically and do not leave much room for negotiations, online auction platforms allow for a more dynamic pricing mechanism which as discussed in chapter two (2.3 Economic Significance of Electronic Marketplaces) is likely to result in higher market efficiency. After the buyer has decided for a product and won it in an auction or bought it directly from an online shop, the supplier has to approve and legitimize the order (Krähenmann, 1994).

Dynamic vs. Fixed Pricing

The transaction of the Krähenmann model closes with the delivery phase. While the buyer has to ensure that the negotiated amount gets transferred to the supplier, the supplier has to set up the logistics and send the item to the buyer. Finally, a tracking code of the package could be handed over to the buyer and the supplier's internal information systems (ERP, SCM, CRM) need to be updated or perform post-transaction operations (Krähenmann, 1994).

Delivery Phase

Even though Krähenmann adds that the list of EBM services by all means is not complete, the author of this work raises the following points of criticism:

Points of Criticism

Flexibility: Since Krähenmann's model appears to be a sequential model, the individual phases cannot be run through iteratively. That means that the sequence of first making a decision for a product in the information phase and then negotiating about the price does not allow for a reevaluation of the picked alternative in comparison to other existing alternatives. Decoupling the phases from each other and also allowing iterations over them would result in a more realistic model.

Completeness: The structure of Krähenmann's model and the naming of the EBM services validate the assumption that the products traded are normal goods with a real physical appearance. This restricts the model from being applicable to more abstract products, such as services and financial instruments, as well. A warehousing service, for example, is not required when bonds and futures are traded on capital markets. A risk monitoring module, however, would be of a much a higher value in this case.

Detail: The perspective of Krähenmann's model on an online transaction seems to be too high-level. Some of the EBM services could clearly be considered as separate phases in the transaction process. Adding more detail to the rest of the model would certainly result in a more comprehensive representation of a transaction.

Direction: Chapter two (2.7 Forward vs. Reverse Trading) introduced a new concept of how transactions and trade are generally approached. In the traditional way of trading (referred to as Forward Trading) the buyer gathers a certain amount of offers from suppliers that could potentially deliver the item of interest and typically decides for the supplier that offers the lowest price. As opposed to that, Reverse Trading encompasses that the buyer simply announces his product of interest and the suppliers then apply at the buyer for the bargain by providing a quote. The distinction into Forward and Reverse Trading is called direction. Since Krähenmann's proposal assumes the transactions to be traditional and thus forward directed, his model consequently does not consider Reverse-traded transaction processes.

Security: Krähenmann's model does not consider any processes or EBM services that ensure that the transaction is correctly completed, neither does it detail mechanism that protect the supplier as well as the buyer from fraud.

Resulting from the above points of criticism it will be part of the remainder of this chapter to find a more detailed as well as more general model that captures the processes participating in an online transaction. Figure 13 gives a very high-level view of the process model that is going to be described in more detail shortly.

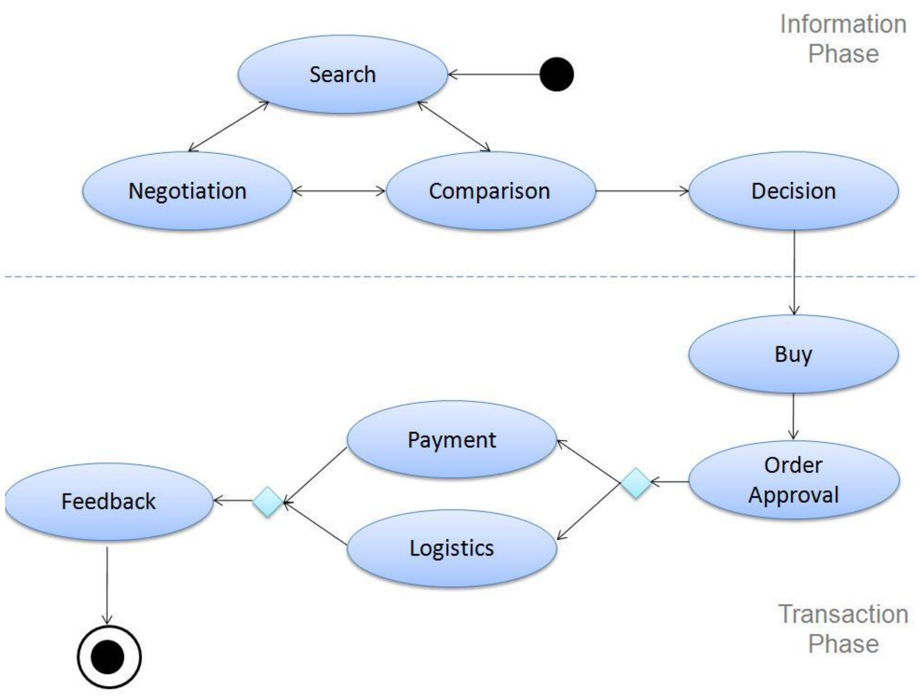


Figure 13: The EBF Process Model at a glance.

The EBF Process Model

As opposed to Krähenmann’s process model, above process model (hereafter “EBF Process Model”) distinguishes only into two phases, the information and the transaction phase, rather than three. Each of the two phases, however, consist of several individual processes that interact via interfaces with each other. The EBF Process Model aims at implementing the requirements of the Requirements Analysis (e.g. completeness, extensibility, direction, communication) as well as providing solutions for other points of criticism that were elaborated on Krähenmann’s process model (e.g. detail, security). It starts with the search process in the information phase and ends with the feedback process in the transaction phase.

H-Processes vs. V-Processes

All the processes (search, negotiation, comparison, decision, buy, order approval, payment, delivery and feedback) will further be detailed into sub processes. The sub processes will be referred to as vertical processes (v-processes) while the actual top level processes will be referred to as horizontal processes (h-processes). While the role of the h-processes is to capture and structure the process flow of an electronic commerce transaction as well as establishing the data environment[7] for the v-process, the v-processes themselves perform the actual work. If, for example, during a search process an individual enters a keyword into a search form, the individual does this as part of a horizontal search process. The search h-process will before routing the call to an appropriate v-process ensure that the v-process has access to all required data and afterwards initiate the vertical “drill-down” process. The v-process then queries the product descriptions in the product model for their attributes, matches them with the search criteria and returns the plain result of the search back to the overlaying h-process. The h-process will

[7] Means ensuring that the v-processes have access to all the data they need in order to be able to execute.

transform the result of the v-process into a format that allows interfacing with other h-processes and also implement functionality for reporting the search result back to a user interface (UI). The h-processes will be formally described in simple mathematical equations. Even though other methods of formalizing process descriptions could have been used as well (e.g. natural language notation, UML), the mathematical notation provides the best overview and quickly shows the in and out interfaces of the process. Later, it will be demonstrated that the interfaces of the h-processes can be derived from their formal process descriptions. Due to restrictions size-wise and time-wise, a formal description of the v-processes, however, cannot be given. The following graphic depicts how h-processes and v-processes relate to each other.

Notation

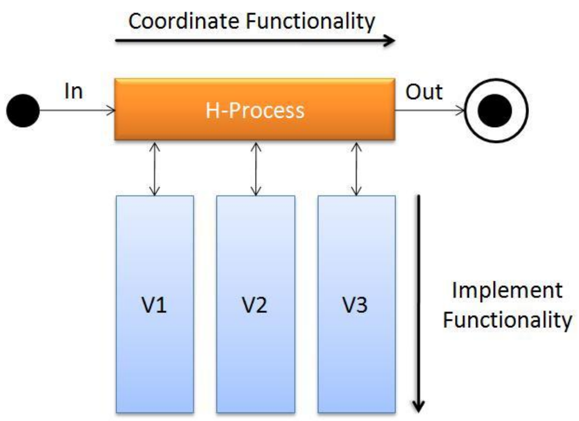


Figure 14: H-processes are the means of interaction with other h-processes as well as coordinate the belonging v-processes.

The first phase in the EBF Process Model is the information phase. During the information phase, an individual that wants to buy a product in order to fulfill a certain demand performs all the necessary research on the market to be able to decide for one alternative out of a set of alternatives. As it will be shown in this section, the processes in the information phase always operate on top of the product model, which was designed in the previous section. Thereby, it does not matter, whether a search query is run, the price of a product is negotiated or whether several products are compared with each other in order to make a decision. Those operations are always performed against the product model. In other words, the role of the product model is to function as the data and information source for the process model's information phase.

Information Phase

In order to clarify which available products on an electronic commerce system would serve the demand fulfillment, individuals typically start with entering keywords into a search engine or browsing through product catalogues or online shops. The outcome of the search process is a list of alternatives an individual would consider to buy. However, even though the list is not yet ranked, all alternatives in the list in some way or another would serve the intended purpose. The following equation states a formal description of the search process (s-process):

Search Process

Search (s)

$$s = f(\sigma) = \{\vec{p}\}$$

The process s is a function of σ and returns a vector $\vec{p}$. σ stands for a function that describes the search criteria the individual applies when searching for a product, while the vector $\vec{p}$ represents the set of alternatives the individual would consider to buy. The process description above already contains the interface description of the s-process. The interface description consists of in- and out-parameters, whereby the in-parameters are represented by the parameter list of the function f and the out-parameters are given by the return set (in curly brackets). Hence, the interface of the s-process is represented by the following:

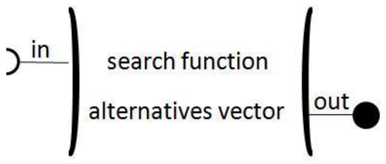


Figure 15: Interface description for the s-process.

Search v-Processes

The s-process is an h-process and thus does not directly define the actual search functionality. The search functionality is rather implemented by the search v-processes.

Search Engine (s.se-process): The search engine v-process implements the search in all product descriptions of the product model for an attribute or a value that matches the search string (in this case σ). The return set of the s.se-process is a set of product descriptions that match the search string.

Browser (s.br-process): The browser allows for browsing through all product descriptions in the product model and therefore provides a way of making each attribute-value combination accessible. The return set of the s.br-process is a set of products an individual has determined to be of further interest as it appears to the individual that those products could potentially fulfill its need. In this case, σ is represented by the implicit browsing behavior of the individual.

Similarity Matcher (s.sm-process): The similarity matcher searches for products that are similar to the product the process was provided with. The s.sm-process tries to find a matching score for an existing product description with others in the product model. With respect to the similarity matcher, σ contains the similarity search criteria.

The above s.se-, s.br- and s.sm- processes can be called in any combination or sequence as they will escalate their results to the steering s-process. This enables scenarios in which an individual, for example, first runs a search (s.se), then browses through the product description of a specific alternative (s.br) that was part of the s.se-process result set and finally wants to find a similar product to one alternative (s.sm).

Once the s-process has completed and a set of alternatives was determined, the search process moves over to either the comparison or the negotiation process. The rationale behind this ambiguity is also one of the advantages of the EBF Process Model over the Krähenmann model. If an individual, for example, received already initial (pricing) quotes for the alternatives $\vec{\mathrm{p}}$ it has found, it can start comparing the prices of the individual alternatives with each other (comparison process) considering the degree to which they serve the intended purpose. However, if the individual does not yet know about the prices of the alternatives in $\vec{\mathrm{p}}$, it should first get some quotes from the supplier, which is part of the negotiation process.

Negotiation Process

In the latter case, the formal description for the negotiation process is reflected by the following equation:

Negotiation (n)

$$n = f(\vec{\mathrm{p}}) = \{\, \vec{q}\, \}$$

The process n is a function of $\vec{\mathrm{p}}$ and returns a new vector $\vec{\mathrm{q}}$ that contains the priced or repriced alternatives. Not only does the negotiation process capture the initial pricing of the alternatives, but it also reflects a repricing in later negotiations with the supplier. If (in a fixed-price scenario) the individual finds that another alternative p2 is better priced than the current alternative p1, the individual will most likely get back to the supplier of p1 and ask for an adjustment of the price. In contrast to a fixed pricing mechanism, the price could also be determined dynamically in an auction-like system. Even though the process would formally seen remain the same, the negotiations would now more or less take place between the individuals that are interested in winning the auction. The interface description for the n-process can be derived from the process description (similar to the s-process).

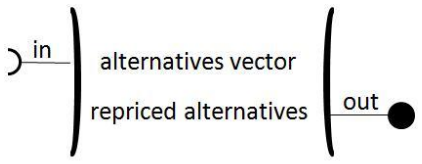


Figure 16: Interface description for the n-process.

Negotiation v-Processes

As shown in the example of the search process, the n-process also wraps a certain amount of v-processes:

Communication (n.co-process): The negotiation process not only consists of the actual negotiation, but is also supported by a communication v-process. The n.co-process reflects the communication ways between the supplier and the buyer, which could have the concrete forms email, chat and video. As it was already indicated in section 3.2, most contemporary e-marketplaces are very limited in the way they allow the participants of a transaction to communicate. To improve the communication and thus ultimately increase the level of trust at which transactions are committed on e-commerce platforms, the EBF Process Model could offer support for electronic messaging via an integrated emailing system, live chat and video chat as well as an appointment scheduling system.

Scheduling Engine (n.se-process): The scheduling engine allows for setting up a time between the buyer and the supplier that might be used for live or video chat or even for a direct meeting at, for example, the suppliers place.

Bidding Engine (n.be-process): The n.be-process enables the participants of a transaction to negotiate about prices, which can be of particular use, when buyers want to acquire a certain quantity of a product from the supplier. Clearly, for large quantities the supplier will be able to offer the buyer a better price – the n.be-process implements those scenarios. More commonly, the bidding engine will be used in electronic auctions, where buyers decide the price of a product between themselves. In addition to normal auctions (e.g. eBay), the EBF n.be-process will also support other types of auctions, such as Dutch auctions or Sealed First-price auctions (Krishna, 2002).

FAQ Engine (n.fq-process): The FAQ engine helps the supplier of a product or service to generate a catalog of answers from inquiries potential buyers have sent via the emailing system. Thereby, the engine suggests adding a new question-answer pair to the FAQ catalogue, but also allows for manually editing the catalog. The n.fq-process can be of a significant benefit to the supplier as it helps him understanding which information potential buyers are really interested in.

All the v-processes above support the establishment of trust between the buyers and the suppliers as well as the pricing of a product. Furthermore, those processes allow for an increased efficiency of communication, not only between the parties of a transaction, but also of the product description itself. The above mentioned FAQ engine, for example, helps the supplier to understand the information needs that potential buyers are interested in and generates this information sufficiently quick without having too much work on the supplier's side. The steering n-process finalizes the negotiation by providing a set of priced or repriced alternatives.

Comparison Process

The second process next to the negotiation process that can be invoked by the search process is the comparison. After the individual has decided for a set of alternatives and knows about the prices, it will try to rank them depending on the extent to which they serve their intended purpose. A formal process description is given by the following equation:

Comparison (c)

$$c = f(\vec{p}, \mu) = \{\vec{r}\}$$

The process c is a function of the vector of alternatives $\vec{p}$ (determined during the search process s) and a utility function μ. The utility function μ describes the individual's personal utility function (considering its predilections und disinclinations) and when applied to an alternative p results into a utility vector $\vec{r}$ for the alternatives $\vec{p}$. The outcome of the comparison process c will be a cardinal or ordinal ranking of the alternatives in $\vec{p}$ considering the utility each alternative provides with respect to its price. Ideally, the ranking is unambiguous (means transitive and complete[8]) which makes it very easy for the individual to decide for the alternative with the highest value. However, if one alternative generates a higher utility in this scenario and another alternative generates a higher value in another scenario, then the ranking becomes more ambiguous and it is not clear to the individual which alternative to choose. In this case, the individual might proceed with another round of negotiations, so that the price might determine the result, or might return to the search process and investigate on more alternatives. Again, the interface for the c-process is derivable from the formal process description.

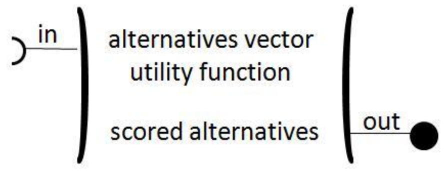


Figure 17: Interface description for the c-process.

Comparison v-Processes

The involved v-processes operate directly on product descriptions while providing support for the following:

> **Product Opposition (c.po-process):** The c.po-process allows for comparing attributes of one product description with the attributes of another product description. Typically, this could be visualized by highlighting those attributes that are similar (or even equal) in both product descriptions while distinguishing those that do not match with other attributes. The c.po-process supports individuals in preparing a decision by trying to make the relevance of a product for the need fulfillment more transparent considering its price and other attributes, such as its age.

The steering c-process returns with a ranked list of product descriptions.

Decision Process

If the efforts of the individual after a sequence of search, negotiation and comparison actions finally result in an unambiguously ranked vector of alternatives $\vec{p}$, the individual can pick the alternative p with the highest utility and make a decision.

[8] For further elaboration on those two terms, please refer to Brickley et al. (2006).

Decision (d)

$$d = f(\vec{\mathrm{p}}) = \{ p \}$$

The process d thereby describes a function of the alternatives vector $\vec{\mathrm{p}}$ and returns the alternative p with the highest utility value. The interface description for the d-process is given by the following Figure 18:

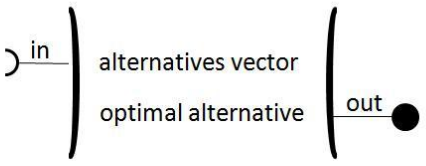


Figure 18: Interface description for the d-process.

Decision v-Processes

As the decision process basically just supports picking the alternative with the highest utility, the number of involved v-processes is rather small.

> **Decision Supporter (d.ds-process):** The decision supporter suggests an alternative from a set of alternatives to be chosen by the individual as it apparently shows the highest utility value.

Once the d-process has ended and the individual has chosen an alternative, the information phase concludes and the buy process in the transaction phase is invoked. The following graphic summarizes the information phase of the EBF Process Model.

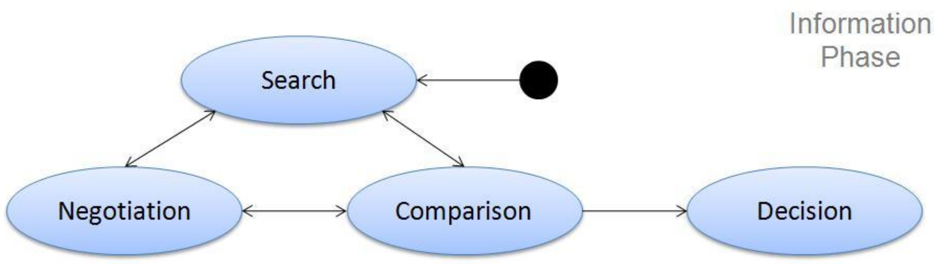


Figure 19: The information phase of the EBF Process Model.

The decoupling of the information phase from the transaction phase also separates the processes that interact with the product model from those that do not show any product-related interaction anymore. As the transaction phase is purely focused upon fulfilling the conditions of the deal, means ensuring that the supplier receives his funds and the buyer his products or services, no further interaction with the process model is required.

Krähenmann Model Augmentation

In the beginning of this section another already existing process model (Krähenmann, 1994) was introduced and weak spots were analyzed. As those weak spots make a part of the Requirements Analysis to the EBF Product Model that is being designed, this section in particular addressed the following points of criticism to the Krähenmann model.

Flexibility

Flexibility: While the Krähenmann model was very static and strictly described the order in which the transaction phases have to be run through, the EBF Process Model is absolutely flexible. While researching on products and trying to make a decision for one alternative, the EBF Process Model allows an individual to jump back and forward in the search, negotiation and comparison process. These jumps occur on something that from now on is referred to as "Information Bus" (see Figure 20). The individual could, for example, first search for a product, then compare it with others and afterwards try to renegotiate the price with the supplier. If the requested price doesn't satisfy the individual, it could start over with the search process again or investigate on similar products. This flexibility was not given by the Krähenmann model.

Information Bus

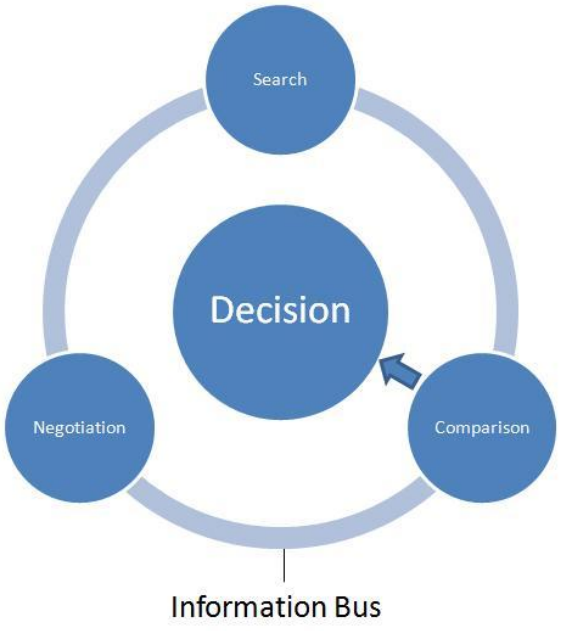


Figure 20: An individual in the EBF Process Model's information phase moves on the Information Bus from one process to another. The information phase ends, when after a comparison the individual is able to make a decision.

Direction

Direction: Chapter two introduced the concept of Reverse Trading as opposed to the traditional way of trading. As Reverse Trading only covers the aspect of how to match buyers with suppliers and find a price, similar to traditional Forward Trading it only takes place in the information phase. Even though above explanations about process descriptions so far only took into consideration the normal Forward Trading scenario, they were designed in a way that they are capable of also supporting Reverse Trading scenarios. This is possible, because in case of Reverse Trading, the suppliers and the buyers simply turn the tables and take the opposite role. The operations on the product model as well as the processes themselves remain the same. Assuming that instead of the suppliers launching product offerings on a platform, now buyers launch product inquiries, the search process would start with the suppliers searching for the buyers whose demand they could potentially fulfill (in a more sophisticated scenario, a platform would inform suppliers automatically about buyers who have entered an inquiry they would be able to serve). As there might be several suppliers interested in a potential deal with the customer, they start bidding on the price of the product at which they are willing to sell, using exactly the same v-processes that the previously mentioned n-process contains. The comparison process might be entered, when suppliers have identified a list of potential customers, but cannot fulfill their aggregated demand. The v-processes in the c-process could than assist the supplier with identifying the buyers that appear to be the most lucrative ones.

Before in the next section the second phase of the process model will be modeled, a pre-assessment of the implementation of the requirements to the process model (see Requirements Analysis) is given in Table 3.

Requirement	Completion
Completeness	
Extensibility	
Direction	✓
Communication	
Flexibility	✓
Detail	
Security	

Table 3: Preliminary requirements checklist for the EBF Process Model.

Transaction Phase

After it was previously shown how the EBF Process Model supports individuals with making decisions during the information phase, it will now be covered the processes that are involved in the second phase of the EBF Process Model, the transaction phase.

The information phase concluded with a decision that was made by the individual for a product. As part of the buy process, this decision is now transformed into a contract. For the first time, the engagement between the buyer and the supplier becomes serious as both have committed certain duties on their sides: The buyer to transfer the total amount of the transaction volume to the suppliers account and the supplier to deliver the product(s) to the buyer within a timeframe that was agreed upon (or was part of the product description).

Buy Process

Typically, during the buy process, the buyer specifies the amount of products he wants to buy, passes product parameters (e.g. size of a business suit), if applicable, and enters the shipping information into the system. The following equation represents a formal description of the buy process:

Buy (b)

$$b = f(\mathrm{p}, \mathrm{n}, \vec{\mathrm{v}}, \mathrm{s}) = \{\, c \,\}$$

Thereby, the process b is a function of p (the product that is to be bought), n (the number of products that are to be bought), $\vec{\mathrm{v}}$ (the parameter vector for the product) and s (the shipping information). The b-process compiles this data into a contract c between the buyer and the supplier. The interface description can (similar to the processes in the information phase) again be derived from the formal process description above.

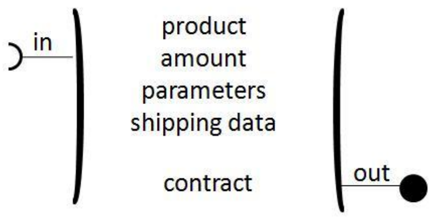


Figure 21: Interface description for the b-process.

In above process description, one could add that with every transaction it is only possible to buy one or many products of a certain product description. If, however, those processes are virtualized again and internally assembled to one big order, a transaction could also contain several products with different product descriptions.

Buy v-Processes

The v-processes that are involved in the buy process are given in the following:

Order Capturer (b.oc-process): The b.oc-process captures the product and amount of the product an individual wants to buy as well as the parameters for the product.

Shipping Capturer (b.sc-process): The b.sc-process captures the shipping data s of the product that was bought.

The above b.oc- and b.sc-processes need both to be called and are not optional as many other v-process that were discussed in other parts of the design section. The shipping capturer process, however, might become superfluous, if the shipping data of the buyer is already captured or stored elsewhere. The information that was assembled by the v-process will be compiled to a contract and passed over to the next process.

Order Approval Process

The contract generated by the buy process is forwarded to the next process, the order approval process. The order approval process gives the supplier the possibility to confirm that he is able to deliver the amount that is stated in the contract within the specified timeframe. The order approval process can also be used to verify the integrity of the buyer, which might be of particular interest when financial instruments are traded. If either the supplier cannot deliver the requested quantity of the product or the buyer failed the integrity test, the contract can be canceled here. In other cases, the order will be approved and forwarded to the payment system.

The following again stands a formal process description, this time for the order approval process:

Order Approval (o)

$$o = f(c) = \{r\}$$

The process o is a function of c, the contract that was filed during the last b-process. It returns an order approval result r, which would typically contain "approved" or "not approved". If the order is not approved (either due to the supplier not being able to deliver the order quantity or due to the buyer not having passed the integrity test), the process flow ends here. The interface description derived from the formal description would look like this:

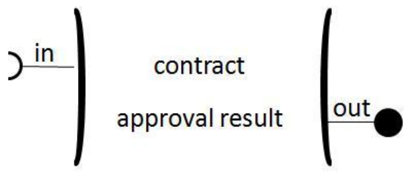


Figure 22: Interface description for the o-process.

If the order approval failed, the process flow would gracefully end and communicate the result as well as the circumstances of the failed order approval to the buyer. One could imagine a possible start-over in the transaction process model at this point.

Order Approval v-Processes

The o-process consists of the following v-processes:

Delivery Confirmation (o.dc-process): The delivery confirmation process confirms that the supplier is really able to deliver the ordered quantity of the product. Thereby, it also needs to be considered the customization, which results from the parameters that were captured during the buy process.

Integrity Confirmation (o.ic-process): The integrity confirmation process makes sure that a transaction with the customer does not involve any unknown risk for the supplier by, for example, browsing through black lists or hiring background agencies.

The results of the o.dc- and o.ic-process together determine the final order approval result. If the order was approved, the process flow is handed over to the next stage. According to Figure 13, the process flow now splits into two concurrent streams. The rationale behind that is that in order to maintain a high degree of flexibility, the EBF Process Model must be able to represent scenarios in which the payment and the logistics are independent from each other. In traditional e-commerce, the product is usually shipped, once the payment was confirmed and the funds arrived in the supplier's bank account. However, also the opposite might be applicable, if, for example, an individual gives a loan (supplier) to another individual (buyer). By paying out the loan amount, the supplier would first deliver the product. After the contract time has ended, the buyer then would pay for the product, which includes the lent money plus an interest. Therefore, the

EBF Process Model splits the process flow after the order approval process and thus allows for a concurrent payment and delivery process.

Payment Process

In the payment process, the buyer transfers the amount that was agreed upon in the contract to the dealer's bank account (or pays cash in case of a personal pickup). All necessary data for the payment is contained in the contract already.

The formal description for the payment process is given by the following equation:

Payment (p)

$$p = f(\mathrm{c}) = \{\, pc \,\}$$

The process p is a function of c (the contract) and returns information about the payment confirmation (pc). The interface description for the payment process would look like this:

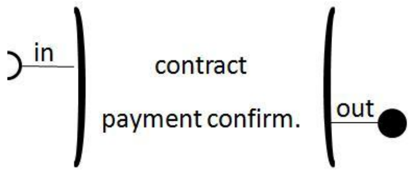


Figure 23: Interface description for the p-process.

Payment v-Processes

Typically, the v-processes for the payment process are handled by or outsourced to either an external service provider, such as Paypal, or a financial institution. In case of using Paypal, the whole implementation of the payment process is outsourced from the EBF Process Model. The steering p-process simply needs to create an invoice from the contract and later check the payment confirmation that was sent by Paypal. Using a financial institution instead might seem similar from a high-level perspective, but the administrative overhead in the p-process would be significantly higher. The p-process consists of the following v-processes:

Invoice Generation (p.ig-process): The p.ig-process generates an invoice from the contract and sends it to the buyer.

Payment Confirmation (p.pc-process): The p.pc-process monitors the transaction and confirms that the payment was really done.

Logistics Process

The second process that runs concurrent to the payment process is the logistics process. The process description for the logistics process is represented by the following equation:

Logistics (l)

$$l = f(\mathrm{c}) = \{\, s \,\}$$

The process l is a function of c (the contract that was generated in the buy process) and returns a shipping status s. Typically, the shipping status contains a package tracking number for the buyer to online track the shipment. In case of a personal pickup by the customer, it could also contain the information that the product was handed over. The interface description for the l-process is shown by the following Figure 24:

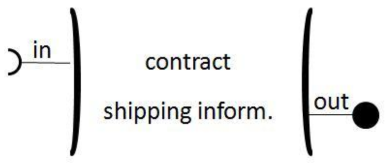


Figure 24: Interface description for the l-process.

Due to the complexity of its nature, in most e-commerce scenarios the logistics process is outsourced to specialized logistics enterprises (e.g. DHL, FEDEX). As those enterprises deal with a large amount of customers, they are able to offer the same service at a much lower price than companies trying to have their own logistics setup.

Logistics v-Processes

The logistics' v-processes consist of the following:

> **Internal Logistics (l.il-process):** The internal logistics process includes all actions involved in getting the ordered goods from the internal warehouse or storage to the l.lp- or l.pp-process. The internal logistics process therefore is also a reasonable candidate for establishing a connection to another supply chain management process.
>
> **Logistics Provider (l.lp-process):** The logistics provider process is a hull process that allows the integration of an external logistics provider. However, if a supplier would like to maintain his own logistics facility, this could also be placed in the l-lp-process.
>
> **Personal Pickup (l.pp-process):** The personal pickup process reflects the scenario of the product(s) being picked up by the buyer in person.

Feedback Process

Once the payment was received by the supplier and the product was delivered to the buyer, all conditions of the contract were formally fulfilled. The EBF Process Model shows this by merging both the logistics and the payment process together again into the feedback process. However, the feedback process does not have an impact on the fulfillment of the contract, but rather allows for post-contractual communication between the buyer and the supplier. On an online auction platform, feedback typically includes an evaluation of the supplier by the buyer. However, it could also be as simple as the buyer just sending an email to the supplier communicating his opinion about what he thinks could be done to improve the service. The last process description, this time for the feedback process, consists of the following:

Feedback (f)

$$f = f(\mathrm{c}) = \{\, i \,\}$$

Relating to each contract and thus to every individual transaction, the feedback process is a function of c (the contract). The return value is the feedback information i that was gathered and could typically consist of an evaluation or a post-contractual email. The interface description of the feedback process would look like this:

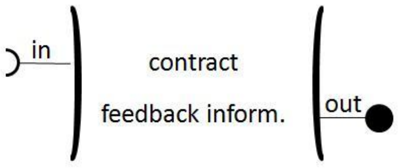


Figure 25: Interface description for the f-process.

Feedback v-Processes

The f-process consists of the following v-processes:

Evaluation (f.ev-process): The valuation process captures and stores information about the quality of the transaction.

Post-contractual Communication (f.pc-process): The f.pc-process allows for a final communication between buyer and supplier after the formal conditions of the contract were met (or not met).

The transaction phase concludes with the feedback process and thus the overall EBF Process Model ends as well. The following graphic summarizes the individual processes of the transaction phase.

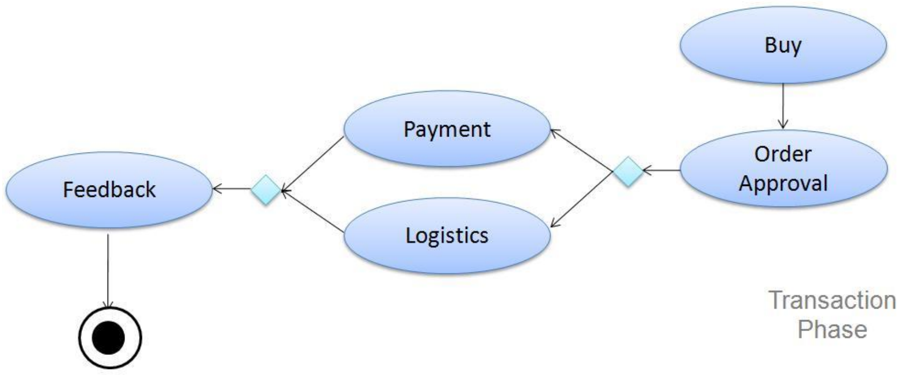


Figure 26: The transaction phase of the EBF Process Model.

Goal Assessment

In the beginning of this chapter, the requirements to the EBF Process Model were formalized as part of the Requirements Analysis. Points the EBF Process Model would need to consider were completeness, extensibility, direction and communication. Derived from the assessment of an existing e-commerce process model (Krähenmann, 1994), the points flexibility, detail and security were added as requirements. Given the approach that was presented throughout this section, the design of the EBF Process Model implements the following criteria:

Completeness: From the perspective of an e-commerce platform, the EBF Process Model appears to implement all relevant processes that are participating in an online transaction. The Krähenmann model met this criterion also already, but from a more distanced point of view.

Extensibility: Processes within the EBF Process Model are extensible to a very high extent. While the interfaces of a process (described in the interface description) are considered to remain the same throughout most extension efforts, the v-processes can be altered and also completely substituted or deleted. Adding new v-process is also possible, as long as they escalate their results to the steering h-process. Via the h-process's interfaces the communication to other processes is established. If a modification of an h-process' interfaces, however, should become necessary, the matching interface of related processes would most likely also need to be changed.

Direction: The EBF Process Model by nature is able to support traditional Forward Trading as well as Reverse Trading transactions (see Table 3 in the information phase modeling).

Communication: During the information phase, the EBF Process Model supports communication as part of the negotiation process. Unless an individual has not decided for a product or a set of products, it can from any process always return to the negotiation process and seek for the communication with the supplier. Once the individual has made a decision, all further communication between the buyer and the supplier will be handled by the EBF processes. The same is true for Reverse Trading scenarios, just that suppliers and buyers turn the tables. If necessary, one could maybe also think of adding a distinct communication process that spans across the whole EBF Process Model. However, at the current stage, it appeared that there was no need for such a distinct communication channel as an EBF-steered communication throughout the transaction phase seemed to be sufficient.

Flexibility: While Krähenmann's model was static in the way that it described in which order the individual processes would be run through, the EBF Process Model implements a higher degree of flexibility. As long as interaction between a (potential) buyer and a supplier is involved, the EBF Process Model merely described which individual processes need to be run through, but it did not determine the order neither did it state how often those processes are entered.

Detail: The point of criticism that the Krähenmann model is too high-level was addressed in the EBF Process Model by first of all splitting up an online transaction into information and transaction phase. Furthermore, several h-processes were added to those two phases with each of them containing a formal process description as well as a description of the interfaces. Lastly, the h-processes were subdivided into v-processes and their general purpose was explained.

Security: Security has not been addressed yet and will be part of the next section.

Requirement	Completion
Completeness	✓
Extensibility	✓
Direction	✓
Communication	✓
Flexibility	✓
Detail	✓
Security	

Table 4: Preliminary requirements checklist for the EBF Process Model.

Quick Review – The Transaction Processes

The EBF Process Model distinguishes into the two phases information and transaction. The information phase covers all the processes that support the decision making of an individual. Thereby, those processes can be run through in any order and repeated as often as necessary for making an unambiguous decision. This dynamicity is described in a theoretical concept that is referred to as Information Bus. The transaction phase covers those processes that are involved in the order processing, once an individual decided for an alternative in the information phase. Thereby, the main task of the transaction phase processes is to ensure that a contract between buyer and supplier is established and that the conditions of this contract are fulfilled. The EBF processes consist of h-processes and v-processes. H-processes provide the interfaces to other adjacent processes, whereby v-processes implement functionality. Formal process descriptions for h-processes can be used to derive their interfaces. Furthermore, every h-process type contains a proposal for possible v-processes.

3.5 Security Augmentation

So far, the EBF Process Model has captured all for an online transaction relevant processes and has also shown in what sense those processes relate to each other. However, the question of how the process model actually ensures that the overall transaction was completed to the conditions that were stated in the contract and also how fraud is prevented was not yet discussed.

One instance of the security that is more or less independent from the process model was already mentioned throughout the previous section. During the order approval process, the supplier not only verifies that he is able to deliver the customization and the quantity of the product that the customer bought. Typically, the order approval process is also used to ensure the validity of the buyer's background. If, for example, a credit shall be given to an individual via an e-commerce solution, the credit supplier would first gather a certain amount of information about the individual's credit history. Based on this data, the credit supplier could then either approve the order and grant the credit or he could cancel the order, if the individual was ranked as too risky.

Order Approval

Apart from those security aspects that require the involvement of certain logic on the supplier's side, the EBF Process Model furthermore formalizes a way of how security can be dealt with. Referring to Figure 27, a real security concern only lies in the transaction phase of the EBF Process Model. This also makes sense as during the information phase the buy and the supply side have not agreed upon any contract yet.

Security

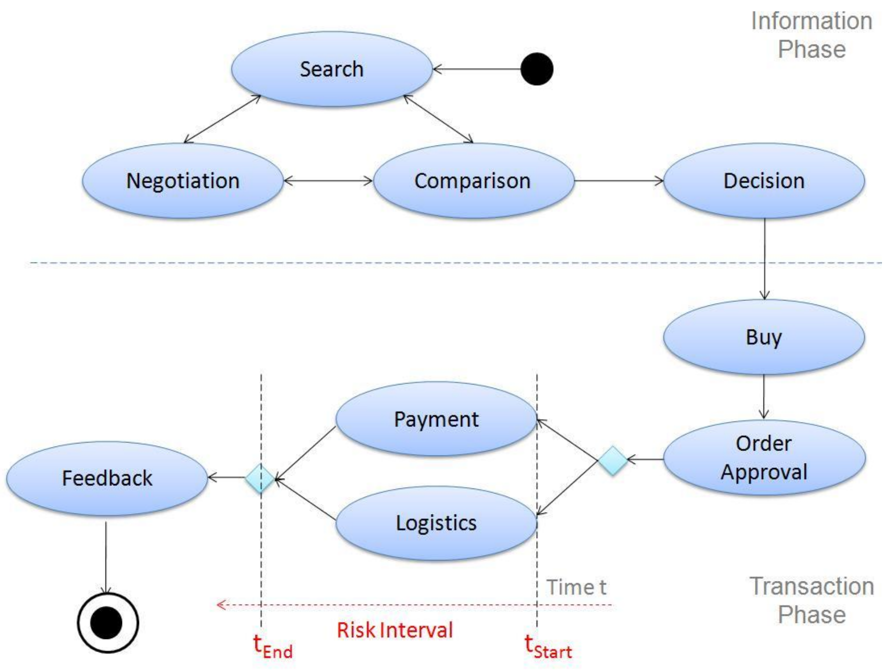


Figure 27: The risk interval in the transaction phase of the EBF Process Model.

In contrast to the information phase, a critical moment from a security point of view arises when the payment and the logistics processes are not executed concurrently. If, for instance, the payment process is started prior to the logistics process, which organ ensures that the supplier really delivers the product as it was stated in the contract? Figure 28 demonstrates the impact of the risk interval, if payment and logistics are not executed at the same time. Thereby, only the relevant part of the transaction phase is reflected.

Risk Interval

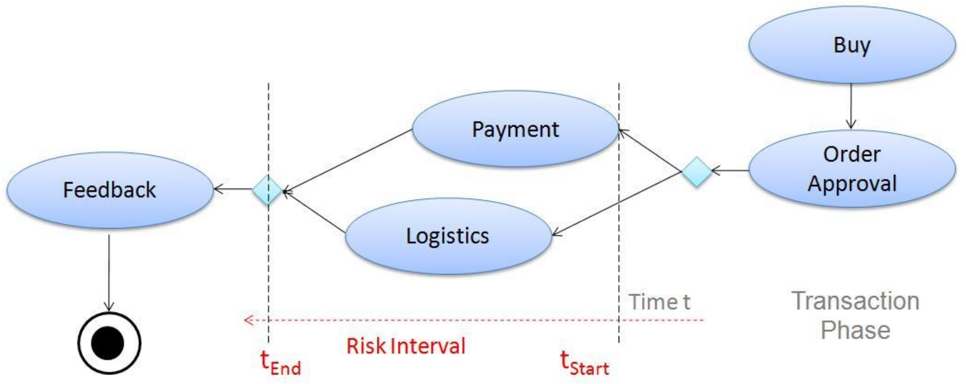


Figure 28: The risk interval with a delayed logistics process.

As it can be seen in Figure 28, a delay in the logistics process expands the risk interval (in this case for the customer). That means, the longer the delay between the two processes payment and logistics, the higher the risk for either the supplier or the buyer, depending on which process is executed first. If, for example, a credit supplier wants to give a loan to a debtor (buyer), above risk interval logic would result in a higher risk for the credit supplier, the further the payback date for the debtor in the future. If the debtor was a company that wanted to invest a credit into new machines, who ensures the credit supplier that after, for example, two years the company doesn't go bankrupt and hence the credit supplier has to depreciate his credit amount? Many, even more practical examples are considerable, but the rationale behind the risk interval still remains the same. To solve this problem and remove the ambiguity involved in the concurrency of payment and logistics, the EBF Process Model introduces a new process, the risk process. For the purpose of simplicity, the buy, order approval as well as feedback processes will not further be considered in the following Figure 29.

Risk Process

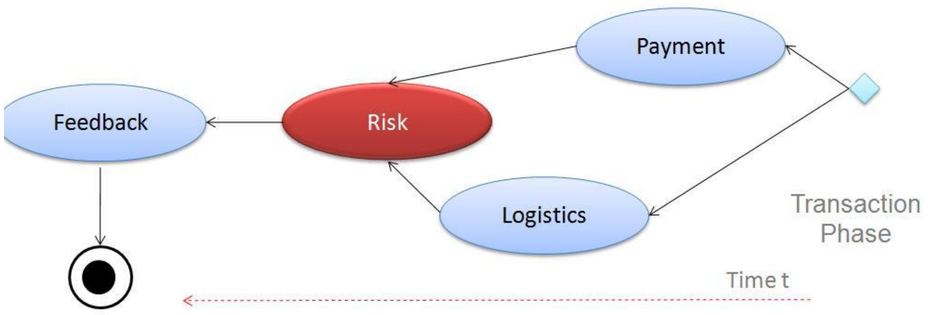


Figure 29: The risk process removes the ambiguity between the payment and the logistics processes as it waits for both to complete before continuing.

The task of the newly introduced risk process is to streamline the payment and the logistics processes. Thereby, streamlining can occur in a variety of ways depending on the actual implementation of the risk process. First, the risk process could act as a man-in-the-middle that in case of an eBay-like or Amazon-like scenario would be the instance of trust. After having received both the funds from the payment process and the products from the delivery process the risk

Instance of Trust

process would re-distribute those to their intended owners. Other than that, in case of an online platform that deals with credits, the risk process could act as insurance for the credit supplier. If, for instance, the debtor was not able to pay back the credit, the insurance involved in the risk process would serve as the payment process, transferring the whole credit sum back to the supplier's bank account (see Figure 30).

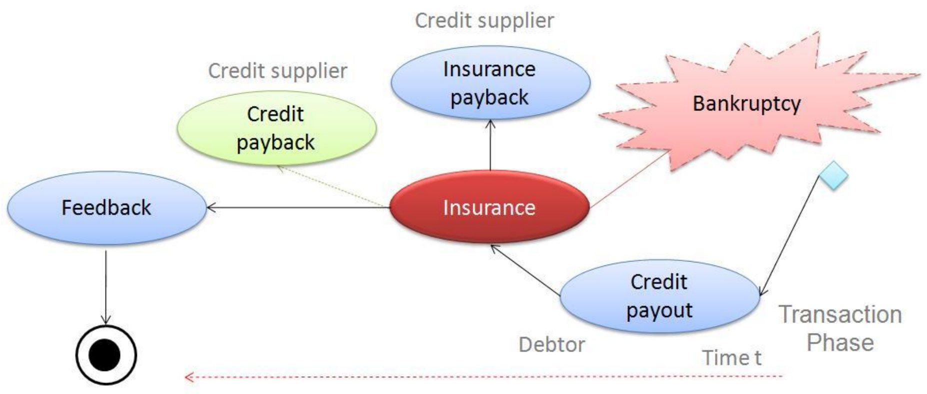


Figure 30: The risk process (here insurance) ensures that the interests of both credit supplier and debtor are maintained. In case of the debtor not being able to pay back the credit sum, the insurance pays the amount back to the credit supplier.

Above scenario of course involves a premium for the insurance which takes over the risk of the transaction. This premium might be added to the total that was agreed upon in the initial contract.

Risk Process Interface

Due to the high number of possible applications for the risk process, no formal process description (similar to the previous section) can be given at this point. Also, it would not be clear how the interface for the risk process (r-process) would look like. Previously mentioned examples, however, already tried to indicate of what kind involved v-processes could possibly be.

With the addition of the risk process, the elaborations on the EBF Process Model conclude. The final version of the EBF Process Model that integrates the risk process is depicted in Figure 31.

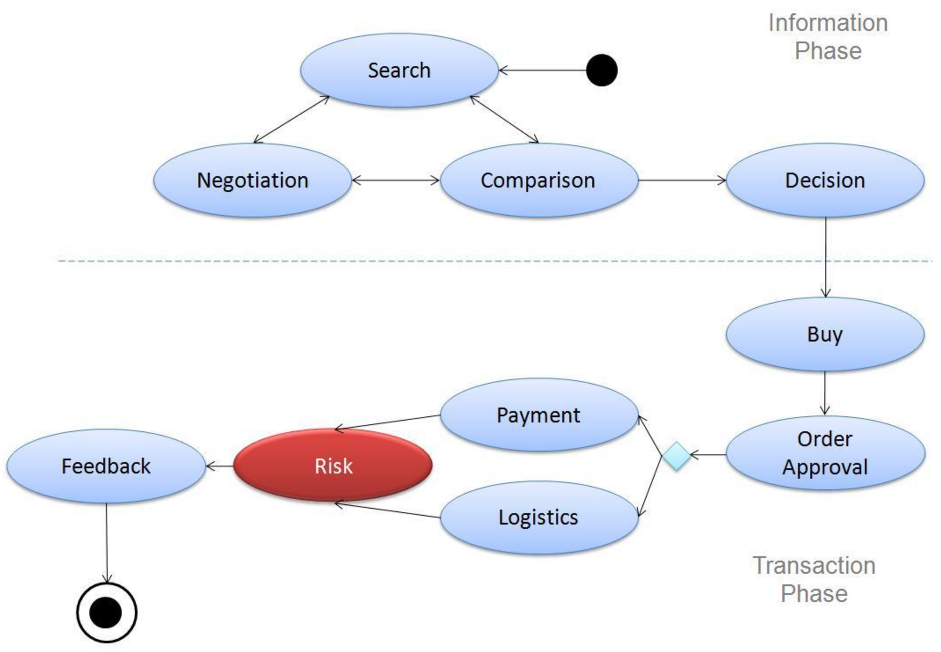


Figure 31: The final EBF Process Model.

Lastly, a final assessment of the implemented requirements stated in the Requirements Analysis shall be given in Table 5:

Requirement	Completion
Completeness	✓
Extensibility	✓
Direction	✓
Communication	✓
Flexibility	✓
Detail	✓
Security	✓

Table 5: Final requirements checklist for the EBF Process Model.

Quick Review – Security Augmentation

The EBF Process Model features an explicit security process that is used to remove the ambiguity between not synchronized payment and logistics processes. By augmenting a transaction with instances of trust (e.g. man-in-the-middle, insurance), the EBF Process Model allows for a more efficient communication and processing of transactions.

3.6 Summary

Throughout this chapter, a proposal for a product and process model that should be flexible enough to cover almost every kind of e-commerce transaction was given. The EBF Product and

Process Model was designed in a way that it is not bound to the trade with a particular kind of goods.

The open structure of its Product Description Language allows for the definition of almost every good and service that is imaginable. Furthermore, its powerful attribute handler feature allows for product properties or attributes that cannot be defined at design time to be defined, acquired or modified at runtime. Totally transparently to the user this attribute handler logic can also be applied, when attribute values are accessed. This makes it possible to, for example, always consider the latest exchange rates for prices on goods that are shipped internationally.

The process model of the EBF was subdivided into two phases: the information and the transaction phase. While the processes in the information phase support market making by providing powerful tools for search, negotiation, comparison and decision, it does not explicitly state from which direction those processes are run through. In traditional Forward Trading scenarios, a potential buyer could use the information phase processes to find information about products and suppliers in a particular market as well as engage in communication and negotiation with the various suppliers. In a Reverse Trading scenario, this initiative could be taken by the suppliers searching for individuals that have started an opening. They could search for demand they could potentially fulfill and communicate and negotiate with potential customers using exactly the same processes as in Forward Trading. Thereby, the market making that includes the communication, negotiation and comparison takes place on a so called information bus, which is exited once a decision was made.

The transaction phase of the EBF Process Model took care of routing all transaction-relevant information through both sides (buyer and supplier). The pay, order approval, payment, logistic and feedback processes covered the administration of a contract as well as the bilateral fulfillment of the contract conditions. Lastly, a distinct risk process augmented the transaction phase with the possibility to integrate security mechanisms for both the supplier as well as the buyer. An intermediate man-in-the-middle or an insurance, for example, could help preventing buyers and suppliers from fraud.

The development of the EBF Product and Process Model was driven by a set of requirements (stated in the beginning of this chapter) and inspired by the comparison with another already existing process model for e-commerce (Krähenmann, 1994). Thereby, the achievement of those requirements was periodically evaluated and a prospect of still open goals was given.

Recalling chapter two, the design of the EBF Product and Process Model also pursued the goal to prove that the statement that only very simple and general goods can be traded on electronic markets (Oppel et al., 2001) cannot stand a reevaluation any longer. The power of the EBF Product and Process Model is given by its flexibility. While being able to represent almost every arbitrary goods or services, its attribute handler logic allows for modifying the product data at every time dynamically, if necessary absolutely transparently from the user. Hence, even complicated products or services that depend on dynamic data can be defined. With the v-processes implemented in the information phase, modern communication technologies, such as text chat or video conferencing, are leveraged to provide a better level of trust between the buyer and the supplier.

Part II: The EBF-AF: An Application Foundation to Support General-purpose E-Commerce Scenarios

While the purpose of this work is to propose a design for a new system that is light-weight and very flexible, the business justification that drives this effort can be found in the huge amount of SME that still run their daily operations without sophisticated IT system support. According to the opinion of the author of this work, the reason for that lies in the complexity and cost of existing e-commerce solutions that mostly target larger enterprises and also are focused upon one specific product or service domain. By proposing a design that is capable of capturing electronic commerce with products that were originally considered as too complex to be traded over the internet and concurrently leveraging cutting edge technologies, the development of the EBF aims at providing a general purpose e-commerce framework with a price that is affordable by SME.

In Part I, an introduction to the basics of electronic commerce was given and drawbacks and limitations of existing e-commerce models were discussed. Based on this, the requirements for an enhanced and novel e-commerce model that should later serve as the conceptual basis for an IT system were assembled. Thereby, the model should not only be capable of reflecting a huge variety of product descriptions (with products of very different natures), but also capture and describe the whole e-commerce process chain.

In Part II, the result of the elaborations conducted in previous sections – the EBF Product and Process Model (EBF-PPM) – will be applied to the design of a real application framework. The reason why the EBF is going to be implemented as a framework rather than a standalone application can be found in the various beneficial aspects frameworks exhibit. Amongst others, the strengths of frameworks lie in the ability to reuse them. Thereby, not only source code is reused, but also the analysis of the application domain that was performed prior to the framework development. Being able to leverage the domain analysis can noticeably lower the risk of software projects derived from the EBF as the results of a complete domain analysis are already incorporated within the EBF itself.

The characteristic of frameworks to be reusable has a strong impact on economies of software development (Froehlich et al., 1997). Not only could the EBF as a framework help to significantly shorten the development time of derived e-commerce applications, but it would also simplify the maintenance of those applications. In case of a fix becoming necessary to a part of the EBF-AF, all derived applications would automatically benefit from this improvement - at most a recompilation of the derived application might become necessary.

The above mentioned “economies of frameworks” could turn out to be a great contributor towards not only reducing the development cost of the EBF, but also towards reducing the cost for maintenance and extension development of derived applications. In this sense, implementing the EBF as an application framework rather than a customizable standalone application seems to be one step towards meeting the main design objective.

Chapter four will therefore, prior to the actual design of the framework, provide a solid foundation of framework engineering principles and practices. Seeking for a high quality of the design and its incorporated architecture from the very beginning and thus easing later maintenance and extension efforts, those principles and practices will be applied in chapter five. The core of the EBF framework will be the later introduced EBF Integration Foundation (EBF-IF), which thanks to its Spawn Point logic allows for an easy extension of the framework and thus a high degree of flexibility.

4 Foundations of Framework Engineering

4.1 Overview

This chapter introduces basic ideas and concepts of software frameworks and engineering. The bulk of this chapter represents a recompilation and reinterpretation of results that were gained from the author's Bachelor Thesis "Maintaining Object-oriented Component Frameworks – A Case Study of the MFC" (Neumann et al., 2008). However, please note that several sections, in order to serve the topic of this work, might have been altered and put into a different context.

The covered clarification of terms and principles will serve as a foundation for the understanding of later chapters. Starting with the definition of the term framework and followed by an introduction to concepts and characteristics of frameworks, it will be shown how frameworks are classified and what framework domains are. Framework benefits and difficulties will be opposed to each other as well as an economic evaluation of the value frameworks can add to the software development process will be given. Finally, framework development techniques as well as the role of components will be highlighted.

4.2 General Framework Concepts

The Term Framework

The term "framework" reflects a basic conceptual structure that solves or addresses certain complex issues. At the same time, the conceptual structure consists of a sequential set of directions which lead to a solution of the problem. Furthermore, the conceptual structure must be generic enough to be applied to various problems that are possibly spread throughout different problem domains. Problem domains, thereby, incorporate an abstract set of problems that is general enough to unify certain problems within one group (Gangopadhyay & Mitra, 1995). Figure 32 depicts how a sequence of specific directions out of a set of directions together with a specific problem out of a problem domain produces a solution.

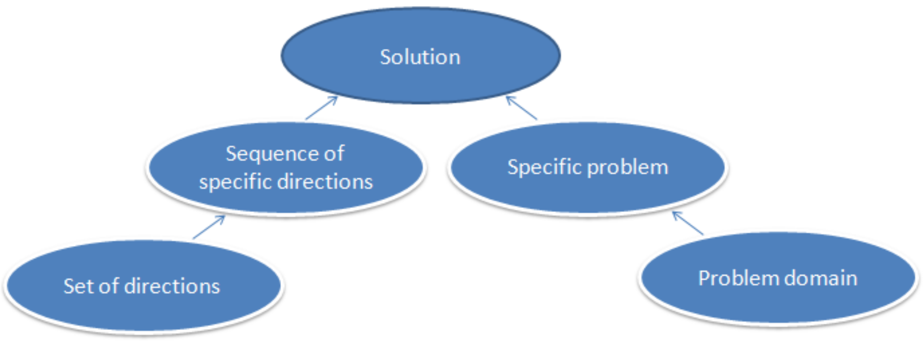


Figure 32: Specific directions applied to a specific problem result in a solution.

"A [software] framework is a skeleton of an application which can be customized by application developers" (Lücke, 2005). Taligent Inc. (1995) defines a framework as "a set of prefabricated software building blocks which programmers can use or customize for specific computing solutions". Thereby, Taligent Inc. (1995) emphasizes that a framework captures the programming expertise necessary to solve a particular class of problems. Companies can purchase or license frameworks to obtain such problem-solving expertise without having to develop it independently. Hence, frameworks provide developers with solutions for problem domains and help them to better maintain those solutions. On top of that, frameworks provide a well-designed and thought out infrastructure, so that when new pieces are created, they can be substituted with minimal impact on the other pieces in the framework (Nelson, 1994). Figure 33 illustrates this with B being substituted by B' in the framework. The application A, B and C that are built on top of the framework automatically and transparently for the applications themselves incorporate the substitution of B by B'.

Framework Definition

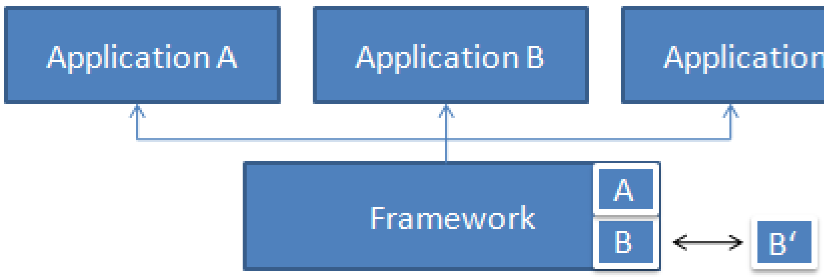


Figure 33: Part B is substituted by a new B' and automatically propagated through the framework to the application extensions.

Lücke (2005) defines a framework as "a reusable design that is described by a set of abstract classes as well as the interaction of instances of these classes". For more detail on abstract classes, please refer to the following excurse.

Abstract Classes

Excurse - Abstract Classes

Abstract classes are classes that declare a set of virtual methods, but do not provide an implementation, means a definition, for these methods. As a result, abstract classes cannot be instantiated through an object, but are rather used as interface classes to objects of subclasses. These subclasses usually define implementations for the base classes' virtual methods and thus become concrete classes that are instantiable by a constructor. On the other hand, subclasses can again be abstract by either not providing implementations for all virtual methods of the base classes or by defining new virtual methods on their own.

Even though the above definitions attempt to characterize the concept of software frameworks by highlighting the key elements and properties of such, they cannot provide an absolutely clear understanding of the concept of software frameworks. Though, all of them share the idea of two basic aspects: First, a framework provides the users with a certain set of functionality and secondly, frameworks allow the users to customize or modify this functionality. However, since frameworks, due to their very nature, try to abstract from a specific problem with a specific solution towards a more generic solution for various specific problems, it is common sense that there can only be an attempt to find a common understanding of how to consider software frameworks rather than a concrete definition.

Framework Architecture

Frameworks, in general, consist of two different parts: The framework core and the framework library. The framework core not only serves as a basis for all interaction between the framework and later applications built from the framework respectively, it also defines the generic structure and behavior of the framework (Sparks et al., 1996). The framework core usually encompasses an ensemble of abstract classes creating an interfacing methodology with applications or with other framework-intern concrete specializations of these abstract classes. An example could be a file system with one abstract class defining the general interface to several file system implementations. While every concrete file system implementation has its own internal representation of data and ways of handling it, the read and write part of each file system always remains the same to the client. The abstract interface introduces a layer of transparency to the client that treats all file system implementations as they were the same and thus hides internal implementation details. Apart from that, the framework core can also contain public concrete classes that are meant to be used as every normal class or in combination with other parts of the framework.

Framework Library

The framework library, on the other hand, consists of extensions to the framework core with concrete components that can be used directly by applications developed from the framework (Froehlich et al., 1997). In most cases, only a certain set of framework library functionality is used within an application. However, the framework library is the part of the framework that addresses the criteria of completion, which will be introduced later, by adding concrete and immediately usable functionality to the mostly abstract core. Therefore, the framework library generally provides solutions for more scenarios than a single application can cover. Figure 34 depicts the two parts of a framework that were introduced in this paragraph.

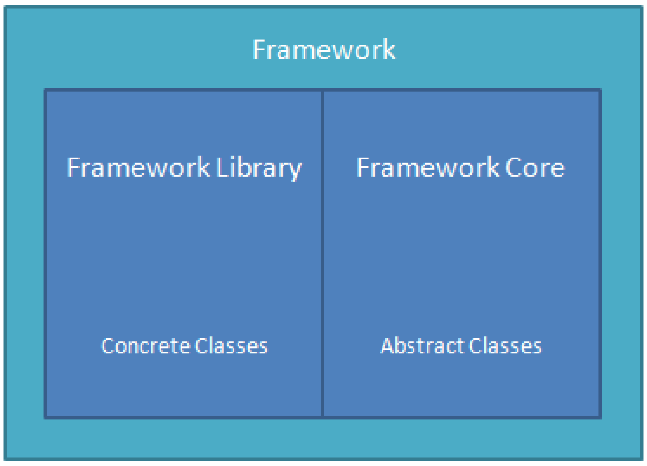


Figure 34: A framework consists of a core with abstract classes and a library with concrete classes.

Application Extension

From a framework engineering perspective, the whole application is the aggregate of the framework core, the used framework library functionality and any application specific extension made (Froehlich et al., 1997). Thereby, the application extension is the additional functional part that the developer has to add on top of the framework, so that the application can perform its specific task. The following graphic illustrates how framework core, framework library and application extension form the whole application.

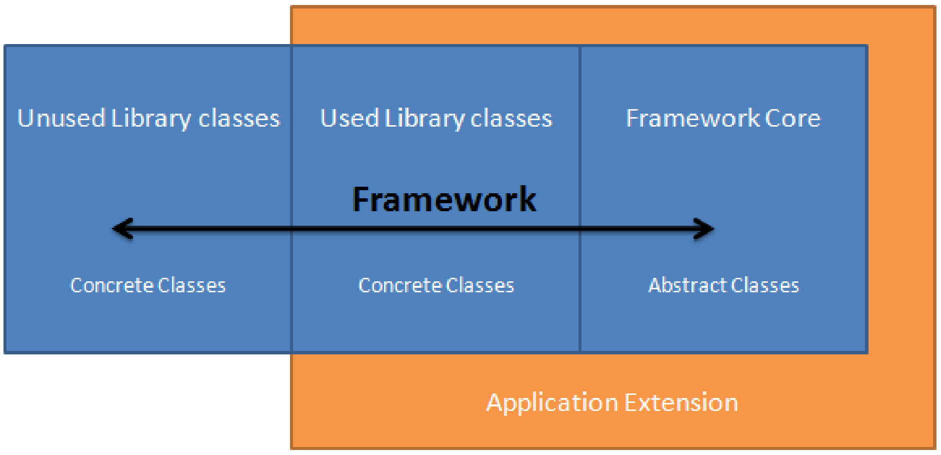


Figure 35: A complete application consists of the framework core, the used library classes as well as the application extension.

EBF Core and Library

Regarding the EBF, its structure from a high level perspective could be distinguished into a part that assembles all external functionality provided by, for instance, external service providers or through extensions. The other part would provide the infrastructure for this functionality and ensure sufficient communication and interaction between the functional components. While all external functionality together could be considered as the framework library, the infrastructure part of the framework in combination with the product and process model (EBF-PPM) could typically reflect the framework core (see Figure 36).

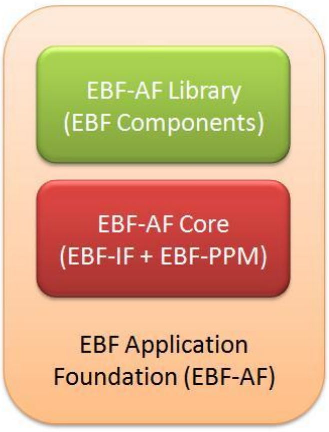


Figure 36: The EBF library and core.

Quick Review – General Framework Concepts

This section has shown that, even though there are some attempts, there can hardly be a concrete definition of the term "software framework". However, all definition attempts somehow share the following two aspects: First, frameworks provide users with functionality and secondly frameworks allow users to customize this functionality. Furthermore, the parts of a framework, namely core and library, were discussed and the EBF functionality from a very high level distinguished into those two.

4.3 Characteristics of Frameworks

When software developers started using frameworks, they did this because of several beneficial characteristics they incorporate. Application engineers noticed that those characteristics help raising the productivity of their everyday work. Instead of troubling with reoccurring problems, such as memory management or file access, they were able to concentrate on the concrete task their application had to perform. This section highlights common characteristics of software frameworks and how these characteristics contribute towards an improved software development experience.

Framework Characteristics

As shown in 4.2 General Framework Concep, the term "software framework" refers to a high-level design that is abstract enough to be applicable to various problems of an application field. On top of that, frameworks offer functionality in form of already partially implemented classes. Both the design and the functionality of a framework incorporate the following key aspects:

Reusability means that software and ideas are developed once and then used to solve multiple problems. Using and reusing frameworks typically leads to an enhanced productivity since applications can now be built on top of already existing solutions for generic problems (Koskimies & Mossenback, 1995). Furthermore, frameworks have usually passed numerous iterative phases of refinement and testing, so that software that is developed from a framework typically shows an increased reliability and quality. The reusability aspect refers to the code as well as the design of the framework, since application extensions not only make use of the already existing code, but are also bound to the general design concept of the framework.

Reusability

Ease of Use encompasses the application developer's ability to use the framework (Froehlich et al., 1997). The framework should be easy to understand and facilitate the development of applications. In general, badly designed frameworks will not be used, if it is hard for the user to understand them and so will fail to meet considerable utilization. In order to develop easy-to-use frameworks, the interactions between the application extension and the framework should be simple and consistent.

Ease of Use

Extensibility means that new components or properties can be added easily to the framework. In order to be truly useful, frameworks not only have to be easy to use, but also easy to extend. Extension typically is achieved by deriving from existing classes (inheritance) or adding customized components to the framework (composition).

Extensibility

Flexibility describes a framework's ability to be used in more than one context (Froehlich et al., 1997). The more problems the framework can be applied to, the higher the problem domain coverage. With a higher problem domain coverage (means the framework provides more solutions to problems within one single problem domain) also the framework's flexibility increases. If frameworks can be applied across multiple domains, their flexibility is especially high. Very flexible frameworks are reused more often than frameworks with a lower degree of flexibility. On the other hand, very flexible frameworks are likely to have a lower ease of use, since flexibility mostly is achieved through abstraction. More abstraction is generally connected to more concrete functionality that needs to be added by the user before the framework can perform its task. Thus, framework design always raises the question of how to find the balance between flexibility and ease of use. Finding this balance can become a tough engineering activity, but is important since neither absolutely inflexible nor very hard-to-use frameworks will appeal to future users (Koskimies & Mossenback, 1995).

Flexibility

Completeness refers to a framework's ability to cover all possible variations of a problem. Since even the best frameworks can never provide solutions to all possible problems in arbitrary detail, it makes it consequentially impossible for frameworks to be complete. However, a certain degree of completeness can be achieved and is referred to as "relative completeness" (Froehlich et al., 1997). Relative completeness encompasses default implementations for the abstractions within a framework, so that these abstractions do

Completeness

not necessarily have to be implemented by the user. If the user wishes to hook in specific functionality for an abstraction, he can do so by overriding the default implementations. On the other hand, if a user does not want to trouble with implementing abstractions that are not meaningful to the solution of his particular problem, then the default implementations for the abstractions might be sufficient.

Consistency

Consistency is a characteristic that reflects that the rules and conventions which determine the framework are followed throughout the whole framework without exception. Consistency in frameworks speeds up the developer's understanding of the framework and helps to reduce errors in its use (Koskimies & Mossenback, 1995). Consistent frameworks always follow the same interface conventions and class structures as well as the same concepts for naming variables, functions and classes.

Additional Framework Characteristics

In addition to the characteristics that were detailed by Froehlich et al. (1997), Lücke (2005) identifies another three basic characteristics of software frameworks: Reuse of Analysis, Hot Spots and Hook Methods and Inversion of Control.

Reuse of Analysis

Reuse of Analysis: Before applications are developed, a detailed description of the problems that the application tries to solve needs to be identified. This process is referred to as Application Domain Analysis. An application domain for financial software, for example, might include portfolio evaluation, an automatic transaction mechanism for stocks, as well as a forecasting model. The results of the application domain analysis include the width of the application domain as well as the application domain's stability. Software that serves the needs of a wide application domain is applicable to more problems compared to software that covers only a narrow application domain. If the second result of the Application Domain Analysis, the stability aspect, turned out to be low, then sudden changes within the application domain itself are likely. In this case, software with a narrow application domain is more likely not to apply to the application domain anymore as compared to software with a broader application domain. So, when application developers perform feasibility studies of frameworks, they implicitly make use of the software framework domain analysis. They will decide for the framework that turns out to best fit the targeted problem domain and provides an appropriate degree of stability towards changes.

Hot Spots and Hook Methods

Hot Spots and Hook Methods: Two other characteristics that are typical for frameworks are "Hot Spots" and "Hook Methods" that represent the customization aspect of the framework. Hot Spots, sometimes also referred to as Hinges, are the particular places in a framework that need to be or can be customized by the framework users. From a developer's perspective, Hot Spots also represent the areas within a framework where placing hooks for the user is beneficial. A networking framework, for example, demonstrates that one Hot Spot can assemble many hooks. In this case, the Hot Spot concentrates on the remote connection management of the networking engine. A user could register a callback function with the framework to notify his application, when another computer tries to open a remote connection. Additionally, the developer might want to specify a hook method that is called, once a remote computer left the network session and closed the connection. A third hook could be installed to notify the application of a refused client due to a limitation in the number of parallel open connections. Hook Methods are understood as the means to perform this customization and mostly encompass virtual functions on a class that need to be or can be overridden by the user in order to fit the framework into the desired behavior. In contrast to Hot Spots, so called Frozen Spots capture the common parts across applications. Frozen Spots are fully implemented by the framework and typically have no hooks associated with them.

Inversion of Control

Inversion of Control describes the property of software frameworks to not reveal the flow of control to the application extension. Frameworks rather keep the flow of control internally while user code or class libraries are called by the framework itself. This is an important demarcation criterion between software frameworks and class libraries: In contrast to software frameworks, code that is assembled in classes as part of a class library is executed by the calling environment. Frameworks on the other hand rather represent those calling environments (Froehlich et al., 1997). Figure 37 summarizes above explained differentiation into the flow of control of frameworks and libraries.

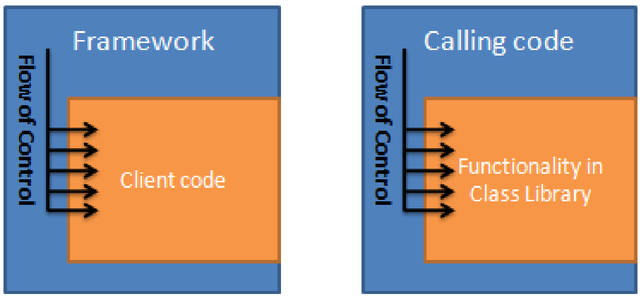


Figure 37: Left a framework's flow of control, right a class library's flow of control: The figure depicts that a framework calls functionality from the user code and thus maintains the flow of control, whereas the functionality of a class library is called by the user's code and thus does not have any influence on the flow of control.

Quick Review - Characteristics of Frameworks

This section gave an overview of the framework characteristics reusability, ease of use, extensibility, flexibility, completeness, consistency, reuse of analysis, Hot Spots and Hook Methods, as well as inversion of control.

4.4 Classification of Frameworks

As already indicated in earlier sections, it is essential for frameworks to be structured generally enough to be applicable to a large domain of problems. In order to achieve high flexibility and thus high domain coverage, the functionality the framework provides is kept as abstract as possible. The more abstract a framework is, the higher its flexibility. On the other hand, higher flexibility usually affects the completeness aspect negatively (see previous section). This section opposes the flexibility and completeness aspects in the means of methods to customize frameworks by discriminating them into two basic classes.

As proposed by Adair (1995), frameworks are distinguished into architectural-driven and data-driven frameworks.

Whitebox Frameworks

Architectural-driven frameworks, also referred to as "Whitebox frameworks", are considered to be still in the initial phase and thus not yet stable. Covered earlier in this text, this means that unstable frameworks are more likely to change due to evolving requirements. Whitebox frameworks do not yet identify stable Hot Spots, which means that it is not yet clearly defined at which place the framework needs to be customized regarding the different application domains. As a result of that, users of Whitebox frameworks have to acquaint themselves with the inner structure of the framework in order to identify the places they need to target for their problem-specific customizations. In most cases, the specialization of Whitebox frameworks is done through inheritance. New functionality is added by creating subclasses of classes that already exist within the framework. This makes it easy for the user to extend the framework and hook in more specific functionality. With respect to architectural-driven frameworks, the flexibility clearly dominates the completeness aspect.

Blackbox Frameworks

Data-driven frameworks, also referred to as Blackbox frameworks, are more sophisticated and mature than Whitebox frameworks and define clear interfaces for their Hot Spots. Furthermore, data-driven frameworks often come with components which are easily hooked up with the framework. These components are combined to achieve the same customization that is done through inheritance in Whitebox frameworks. Selection and composition of already existing components tends to be a lot easier to the user than inheritance from the framework (Adair, 1995). Thus, Blackbox frameworks show an increased ease of use towards Whitebox frameworks. Furthermore, data-driven frameworks tend to show a higher degree of completeness than flexibility as compared to architectural-driven frameworks. The increased stability of the hot spots as well as the components within the framework library provide ready-made solutions for a larger set of problems, but on the other hand are harder to modify and extend. Figure 38 opposes architectural-driven and data-driven frameworks to each other.

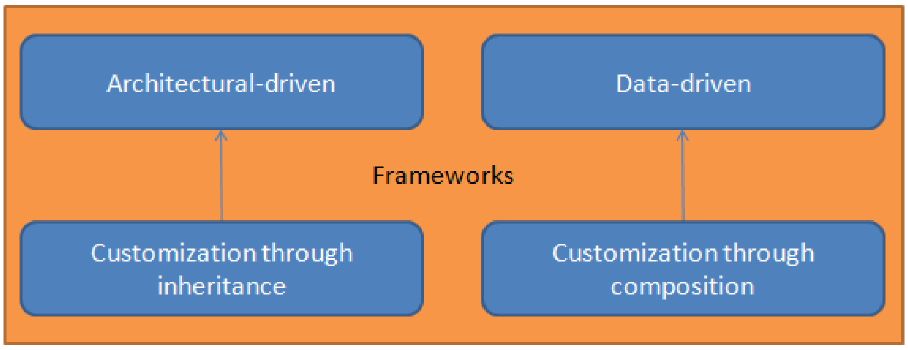

Architectural-driven
Data-driven
Frameworks
Customization through inheritance
Customization through composition

Figure 38: In architectural-driven frameworks customization is achieved through inheritance, whereas in data-driven frameworks customization is achieved through composition.

Framework Evolution

Generally, frameworks start out as Whitebox frameworks that rely upon inheritance. As the domain is better understood, more concrete support classes are developed and the framework evolves to use more composition, mutating into a Blackbox framework. Due to the unstable nature of their Hot Spots, Whitebox frameworks require a more in-depth knowledge from the user than Blackbox frameworks. Typically, a framework will have elements of both Whitebox and Blackbox frameworks rather than be clearly one or the other (Lücke, 2005).

Architectural Hybrid

Applying the Whitebox/Blackbox concept to the EBF does reveal that the EBF can neither be clearly categorized as a Whitebox nor as a Blackbox framework. On the one hand, the EBF Application Framework should be ready for use without any coding involved on the applicant's side (e.g. an e-store or online auction provider). Therefore, the EBF would need to consist of components that, when linked together, provide all basic functionality that is necessary for running e-commerce in general. This Blackbox characteristic might cater for the needs of most of the EBF applicants. However, in case of more severe customization or even extension development becoming necessary, the EBF should also facilitate those efforts through modern object-oriented methods, such as the derivation of classes or Polymorphism[9]. As those aspects tend to lie more within the domain of Whitebox frameworks, the EBF apparently will also have to implement architectural-driven characteristics of those. Hence, the EBF Application Foundation will face the challenge to unite characteristics of Blackbox as well as Whitebox frameworks and thus, from an engineering point of view, could be considered as an architectural hybrid.

In addition to Adair's classification of frameworks, Froehlich et al. (1997) introduced further demarcation criteria that distinguished frameworks based on the scope.

The scope is understood as the area the framework is applicable to. Further classification of frameworks by their scope results into application framework and domain frameworks.

> **Application frameworks** encompass functionality that can be applied horizontally across problem domains. In other terms this means that application frameworks typically

[9] For details on Polymorphism, please refer to (Neumann et al., 2009).

provide functionality for problems of various problem domains. The solutions application frameworks offer are mostly common to a wide variety of problems. Graphical User Interface (GUI) frameworks, for example, find appliance in all applications that need to interact with the user via the screen and stretch horizontally across the domain. The horizontal nature of application frameworks is depicted in Figure 39.

Application Frameworks

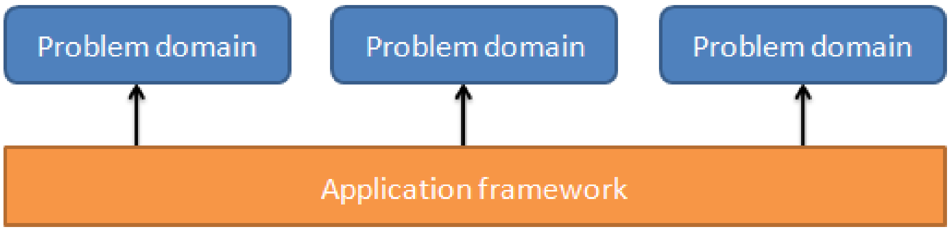


Figure 39: An application framework spans horizontally among several problem domains.

Domain frameworks, on the other hand, offer vertical functionality for a particular problem domain including problems of a similar nature. Operating systems, for example, all offer fundamental memory and process management functionality to other applications that are built on top of the framework. Support frameworks encompass basic system level functionality upon which other frameworks or applications can build. The vertical characteristic of a domain framework is depicted in Figure 40.

Domain Frameworks

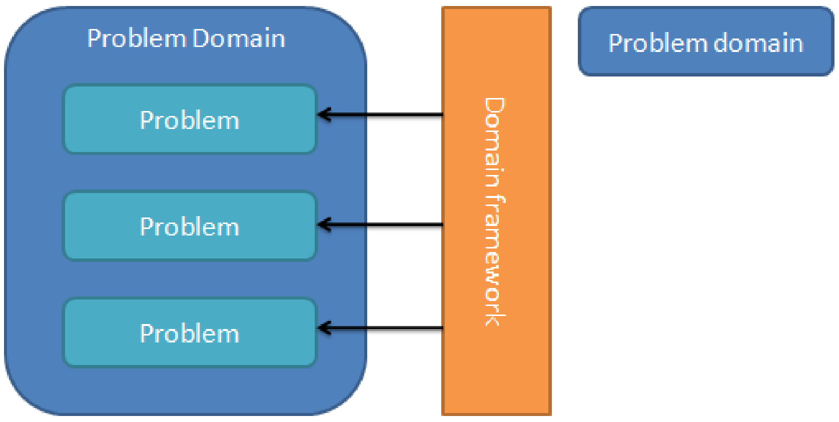


Figure 40: A domain framework spans vertically among one single problem domain.

Since the EBF would specifically cater for the domain of electronic commerce and would rather be unlikely to be applied across other domains, it seems reasonable to declare the EBF as a domain framework. Calling the EBF an application framework, on the other hand, would mean that it would be applicable to problems across multiple domains, which clearly would not be the case.

EBF Domain

Please note that the EBF throughout this work will often be referred to as application framework (in the broader sense), which is not to be confused with the explanations of this section. The term

"EBF Application Framework" rather entitles the power of the framework to serve as an entire application, but to also be customizable and extendable.

Quick Review – Classification of Frameworks

This section showed how frameworks can be distinguished into architectural and data-driven frameworks. While architectural-driven frameworks do not yet identify stable Hot Spots, data-driven frameworks define clear interfaces for their hooks. Another way of classifying frameworks is by their scope into application frameworks and domain frameworks. By discussing the relevance of those principles with respect to the EBF, it turned out that 1) it would rather be an architectural hybrid than a clear Whitebox or Blackbox framework and 2) the EBF, due to its focus upon e-commerce, would be a domain framework.

4.5 Benefits of and Difficulties with Frameworks

After the previous sections have given a profound overview of the characteristics of frameworks, this section will deduce beneficial aspects from the individual characteristics. As it was already discussed earlier, frameworks support the application development process by providing standardized solutions to reoccurring problems. They capture and leverage the expertise of domain experts in a software component that can be included by other application programs. Thus, application developers can focus on providing a solution for the actual problem their software addresses rather than troubling with secondary problems, such as the implementation of I/O operations or mathematical functionality. The use of frameworks can result in a dramatically shortened development time with fewer lines of code. This is because several common aspects of the application extensions are already captured by the framework. Also, the effort required for maintaining applications can be significantly reduced as multiple applications are built on top of one framework (Froehlich et al., 1997). The result of that is the following: In case of modifications or fixes being made to the framework, application extensions implicitly benefit from the changes since those are automatically propagated through the framework. As application extensions that are built on top of a framework follow the same design and share the same code base, frameworks provide consistency and therefore a better integration across platforms. Lücke (2005) claims in his article that frameworks can trivialize the application development to first find a fitting framework, to then learn this framework and finally to customize it in order to satisfy the specific demands of the application extension.

Framework Benefits

Above all, using frameworks is related to two major aspects: Reuse and Quality. Thereby, not only the implementation of a system, but also the design of the system is reused. Since the design of successful frameworks has already proven to be efficient and has run through an in-depth testing and refining process, it forms a quality base for developing new applications. Thus, once a

Reuse and Quality

framework has been developed, the problem domain has already been analyzed and a working design and implementation have been produced (Arrango et al., 1991).

Difficulties with Frameworks

Concerning the development of software frameworks, Froehlich et al. (1997) state that the building cost for a good object-oriented framework is significantly increased compared to single applications. A large amount of time is spent on defining and refining the abstraction of the framework in cyclic phases. Additionally, framework designers and developers have to incorporate shifting problem domains into their thinking and keep their framework flexible enough to adopt future changes. If the problem domain changes and the framework fails to adopt these changes, it becomes less valuable to the community of developers. In general, frameworks built for a stable problem domain tend to live longer than frameworks created for rather unstable problem domains. Parallel to changes within the framework due to emerging requirements, the impacts of the framework's evolutionary transcend on depending applications must be taken into account. If the architecture of any framework interfaces change, then upgrading the applications built from the framework might be a costly procedure. Contrarily, if the applications are not upgraded to newer versions of the framework, the advantage of having a common code base gets lost (Froehlich et al., 1997).

Learning Frameworks

Another major concern about the use of frameworks is that the learning of the framework, which is a necessary step prior to the development of the application extension, is most often a time consuming process. Covering a general aspect of the framework learning curve, the 90-10 rule states that the progression of the curve reveals the following peculiarity: 90% of the application development is done in 10% of the time, whereby the remaining 10% require 90% of the overall development time (Froehlich et al., 1997). In the beginning of the learning phase users can quickly solve easy problems since they are utilizing ready-made components of the framework library that address general reoccurring problems. As soon as more difficult and more specific problems need to be solved, whereby no ready-made components yet exist, the users will find themselves lacking the understanding of how to achieve the goals using the framework. They therefore will have to learn more in detail about the internals of the framework and understand how to extend the built-in functionality or modify the framework's existing behavior to serve their needs. They will have to acquaint themselves with the framework's Hot Spots and understand which Hook Methods to override in order to inject their user-defined code into the framework. The following illustration depicts above explained interrelation in the framework learning curve (Werfel et al., 2005).

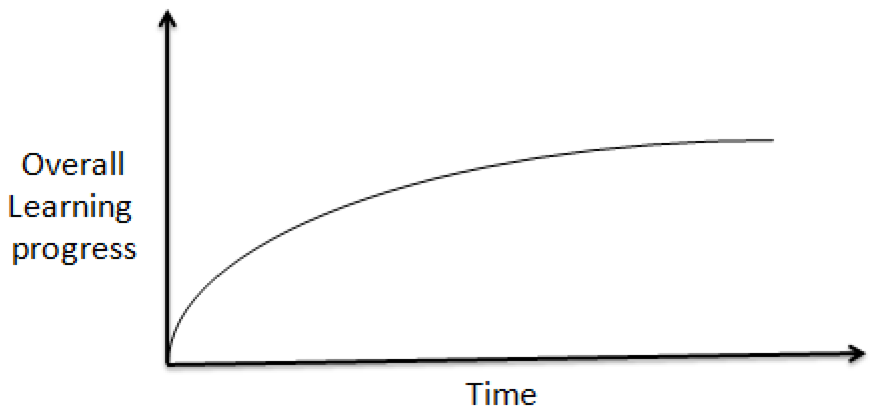


Figure 41: The framework learning curve: In the early phase, the user's knowledge about the framework increases quickly. Over time, the knowledge gain becomes more and more marginal.

Quick Review - Benefits and Difficulties with Frameworks

Frameworks provide standardized solutions to reoccurring problems, capture expertise of domain experts and allow for applications having a shortened development time. Furthermore, frameworks can reduce the effort that is necessary for maintenance and provide a consistent design as well as better cross-platform integration. Regarding the difficulties with frameworks, they show not only a significantly increased building cost, but they also suffer, depending on their flexibility, from shifting problem domains. On the application extension's side, difficulties evolve from the process of identifying the right framework and the subsequent learning that needs to be done.

4.6 Economic Significance of Frameworks

Due to the repetitive refinement cycles during the development phase and the intensive testing that is necessary to ensure the framework's quality, a framework needs more time to be developed than a normal application. Furthermore, the clients that use the framework also need a certain amount of time to learn the framework before they can start writing their application extensions. Therefore, frameworks should be considered as long-term investments whereby the benefits gained from the development of the framework are not necessarily immediate. First, frameworks require more effort to be build and learned compared to normal applications. Since application extensions are developed on top of frameworks and the users usually rely on the framework as being a bug-free foundation for their software development projects, frameworks must run through significantly more intensive and repetitive testing. Additionally, the future success of the framework is not only dependent on the usability and flexibility towards various problem scenarios, but also on the reliability the framework provides. Frameworks with significant bugs and flaws will not be accepted by the clients and thus are predetermined to fail.

Economical Drawbacks

Another concern about the usage of frameworks is that debugging the application extension can require a significantly increased amount of time and money as compared to fixing the bug on the framework's side. In frameworks which are not open source difficulties emerge due to the developer lacking the ability to understand what's going on within the framework and just being presented the results of his method call. For example, a non-open source framework might appear to the user as a magical black box where he merely specifies the input for and gets back the result of the operation without understanding the actual transition from input to output. If the developer does not understand the internals of the method he called, he might also not know in which format the input is expected to be specified. Open source frameworks on the other hand enable the user to read the source files, but the high complexity of the source code, poorly commented instruction lines, as well as thin documentation might turn out to be an obstacle when trying to reconstruct how to correctly call functions and pass arguments. Figure 42 and Figure 43 give an example for how non-open source frameworks can hinder the users in their understanding of the framework.

Open Source vs. Non-Open Source

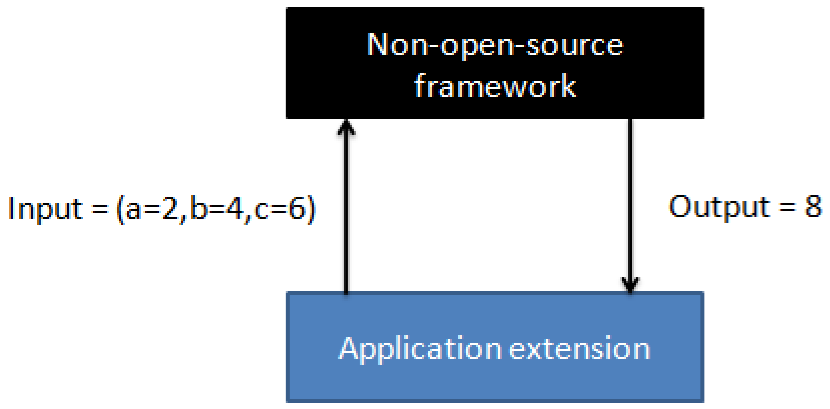


Figure 42: Non-open-source frameworks do not provide the user with the understanding of how input is transformed into output.

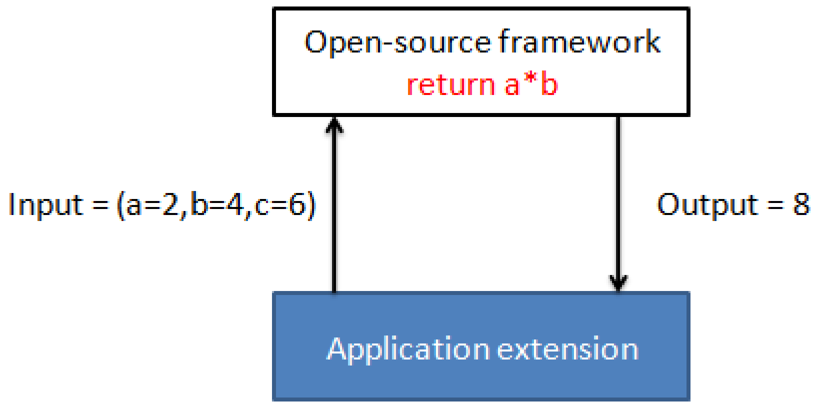


Figure 43: Open-source frameworks on the other hand allow the user to understand the internal process.

Framework Documentation

Further difficulties can evolve from the circumstance that frameworks require detailed documentation including use cases and examples. As mentioned earlier, framework users will hardly have the chance to understand the internals of the source code, regardless of the framework being open source or not. Therefore, it is even more important to provide the user with all information that is necessary to operate the framework in form of a detailed documentation and coding examples. Identifying and gathering the documentation requirements as well as specific attitudes of the target group makes the creation of documentation a costly and timely process.

Framework Migration

The last cost driver that shall be discussed arises from the continuance of maintenance and support for the adoption of changes within the problem domain the framework services. Frameworks that will not respond to changing requirements or provide fixes for bugs will soon or later be rejected by their clients and substituted by alternatives. In general companies are only able to discontinue the servicing of their framework without losing their clients, if they can or at least intend to push their clients towards a successor of the older version or an alternative. Since the framework developers will have to provide support to their clients for migrating away from the old framework towards a possible successor, another considerable cost driver evolves at this place (Froehlich et al., 1997).

Economical Benefits

Early in its life, a framework will probably require more cost-intensive routine maintenance to fix initial bugs and respond to more frequent client requests (Figure 44:a). Over time, however, the cost for maintenance and support will drop and remain at a level that is sufficient to allow for updates and necessary changes to the framework (Figure 44:b). With the replacement of an existing framework by a successor the maintenance and support cost curve rises again (Figure 44:c) before it will finally drop to zero. A maintenance and support cost curve that reached zero stands for the migration away from the framework towards an alternative has completed and the support and maintenance for the framework has been discontinued entirely (Figure 44:d). The following illustration depicts the maintenance and support cost curve associated with frameworks.

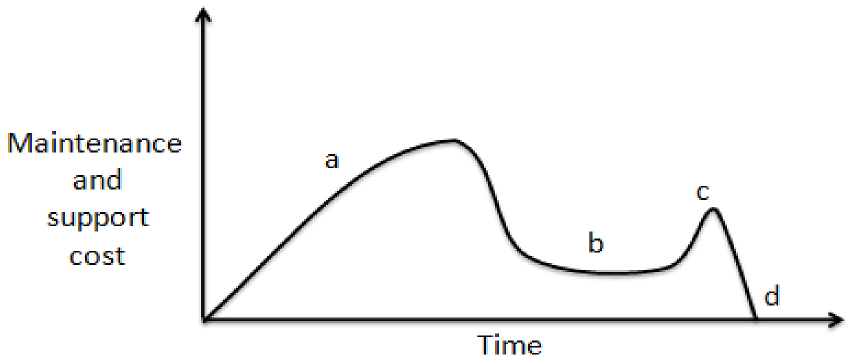


Figure 44: Due to initial bug fixing and refinement, the maintenance and support cost curve rises in the early phase of the framework (a) and afterwards falls down to a point that is sufficient to keep the support alive (b). With the discontinuance of the framework, additional effort for migrating away from the old version to its successor becomes necessary (c) until the cost curve finally drops to zero (d).

As discussed earlier, one of the key benefits software frameworks offer are standardized solutions to reoccurring problems. This results in frameworks enabling application developers to concentrate on their actual application rather than to trouble with the implementation of low-level support functionality. Another key benefit of applications that are built from frameworks is that changes to the frameworks are automatically propagated to all dependent application extensions which makes bug fixing and extension programming a lot easier and less expensive (Froehlich et al., 1997). However, in order to realize those benefits the organization has to commit resources to support the earlier identified cost drivers. Organizations must be able to assist clients with their issues and to respond to their problems and requests. This can be achieved through maintaining mailing lists, news groups as well as support hotlines or community forums.

Propagation of Changes

After all, over the lifetime of a framework, the cost of supporting it becomes a benefit. The cost of supporting one framework with three dependent applications will be less than the cost of supporting three independent applications with duplicate code. In general the following rule applies: The more applications that use a framework, the bigger the savings on cost. In Figure 45 and Figure 46, the measure of the accumulated maintenance cost (AC) illustrates this circumstance. In the long run, using frameworks can reduce the accumulated maintenance cost of applications since all applications are based on one common code base. Changes to the code base would result in the same changes to all depending applications, which makes it easier and less cost-intensive to propagate fixes and updates. Figure 45 and Figure 46 demonstrate how the accumulated maintenance cost of single applications behaves in contrast to the accumulated maintenance cost of applications built from one common framework.

Accumulated Maintenance

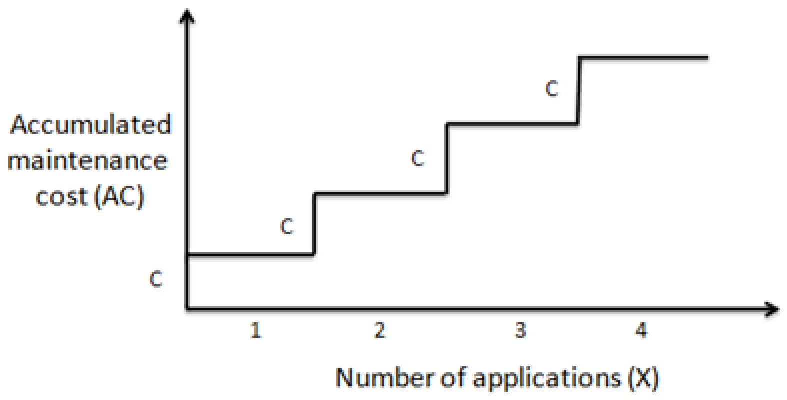


Figure 45: Provided the same changes are required by every application: The overall cost for maintaining independent applications is the aggregate of the single maintenance costs of the applications c or a multiple of the single maintenance cost (AC = X*C).

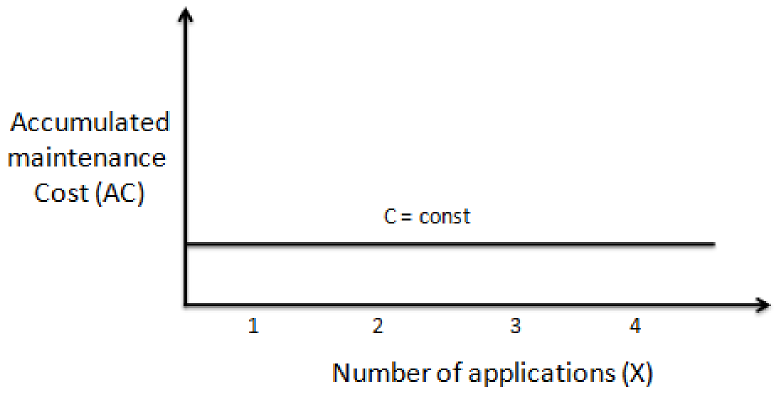


Figure 46: Provided the same changes are required by every application: The overall cost for maintaining applications built from one common framework is constant and exactly the cost that refers to the maintenance of the framework (AC=C).

Quick Review - Economic Significance of Frameworks

Frameworks tend to require more effort to be built and learned than usual applications. Cost drivers evolve from the fact that frameworks require detailed documentation and sufficient maintenance and support. Also, later migration towards newer frameworks must be supported, if the goal is to retain the client community. On the application extension's side, a higher debug complexity might bind an increased amount of resources. The utilization of software frameworks encompasses a set of economical benefits. When a framework is upgraded or maintained, those changes are automatically propagated to the application extensions that were built from the framework. In contrast to multiple single applications that were not built from a framework, the effort that is necessary for maintaining those single applications would be a multiple of the maintenance effort necessary for the maintaining the framework.

4.7 Framework Development Techniques

Frameworks show certain invariants and common aspects regarding their development that need to be considered. This section details those invariants which encompass abstract and concrete classes, composition and inheritance, Hot Spots and Frozen Spots, Design Patterns and Architectural Patterns, Framework Engineering Principles and Practices as well as Component Orientation.

Abstract and Concrete Classes

The framework core often consists of a set of abstract classes that embody the basic architecture and interaction of the framework. The abstract classes should be flexible and extensible since they capture the interactions between the framework's elements, generally in form of hook methods. The number of core classes is usually very limited as it is the goal of the framework design to provide clear and manageable means of interaction. Concrete classes that usually contain very specific solutions to specific problems are added to the framework library and thereby aim at enhancing the framework's completeness. In most cases, the concrete classes of the framework library are specializations of the abstract classes within the framework core supporting more specific problem scenarios. Even though frameworks can never be absolutely complete, a good framework library supports the framework through reaching a level of relative completeness by supplying ready-made components for common scenarios. If certain parts of the framework library are not needed in an application, they can simply be excluded.

Composition vs. Inheritance

In order to allow for hooking into the framework, composition and inheritance are considered to be the two main ways (Adair, 1995). By nature, composition of already existing components tends to be easier than deriving from classes and typically encompasses the use of parameterized types or callback functions. A class, for instance, that needs to be customized will have parameters that are to be filled in by the user. One of the benefits of composition over inheritance is that the user does not need an in-depth knowledge of how particular components operate. Furthermore, composition allows the components to be changed dynamically at runtime which is hardly possible using inheritance. In case of changes in the framework's behavior becoming necessary, instantiating objects of classes that support the desired behavior and passing these objects back to the framework will be enough to achieve behavioral modification. Composition in combination with a large number of concrete classes within the framework library makes adapting the framework a lot easier and reduces this process to simply choosing the right component. Developers then just need to understand which existing component can perform the desired functionality.

When to Use What?

Inheritance on the other side describes the procedure of specializing methods from abstract classes. Compared to composition, it requires a considerable understanding of the abstract classes and their interactions with other classes. As a result of this, inheritance is likely to be more difficult and error prone. The advantage of inheritance over composition is that a high degree of flexibility is sustained and extensibility is incorporated. Application developers can add functionality to subclasses of any existing class which is not easily accommodated with composition. In many cases, inheritance hooks have the form of abstract classes that provide default methods which can be overridden by the child class.

Composition is best used when interfaces and the means of how to use the framework are well defined, whereas inheritance caters for scenarios where the full spectrum of the functionality cannot be anticipated. Moreover, composition forces conformance to specific interfaces and functionality which cannot be altered so easily (Adair, 1995).

Hot Spots and Frozen Spots

Hot Spots represent the parts of the framework that require user interaction and are considered as the means of customization. Accordingly, Frozen Spots encompass the parts of the framework that are equal to every application and thus do not require any interaction on the client's side.

Every Hot Spot will likely contain several hooks associated with the Hot Spot as they describe how specific changes can be made.

In general, three main ways exist in which frameworks can be modified using hooks (Froehlich et al., 1997).

Framework Modification

1. **Providing pre-built components** within the library that are used directly or passed back to the framework to achieve the desired customization.
2. **Providing parameterized classes or patterns** which are filled in by the user.
3. **Deriving subclasses from existing framework classes** and adding new functionality when it is difficult to anticipate how certain Hot Spots are used.

Flexibility vs. Ease of Use

Concerning the desired degree of a Hot Spot's flexibility, it must be remarked that flexibility and ease of use relate indirect proportionally to each other. This means that an increase in flexibility will result in a decrease in the ease of use. The reason for that is simply that an increased flexibility can only be achieved by further abstracting the framework from its targeted problem domain. But the more abstract the framework becomes, the less concrete functionality for specific problems is available. Thus, the flexibility aspect and the ease of use have to be balanced in a way that promises the highest possible success. This balance, however, is hard to find and can often only be approximated. Another significant aspect of customization performed through Hook Methods is that with regards to the changeability of the framework's behavior at runtime, composition is preferred to inheritance if the runtime behavior of a framework changes. As it is hardly possible to generate code at runtime in a way that is efficient, inheritance in most cases is less applicable to address changing runtime behavior (Froehlich et al., 1997).

Design Patterns

Design Patterns capture the expertise of object-oriented software developers by describing a solution to a common design problem that has worked in the past in several different applications. The description of the solution encompasses details about benefits and drawbacks of the pattern and perhaps also names alternatives. With respect to frameworks, Design Patterns are particularly useful for designing Hot Spots - the parts of the framework that determine its flexibility. As Design Patterns have excelled to be a best practice providing reusable solutions to generic problems, Design Patterns can help enhancing the flexibility and extensibility of a framework's Hot Spots and thus of the whole framework (Gamma et al., 1995).

Gang of Four

Even though Design Patterns are available for a long time, the credit goes to Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides who, known as the Gang of Four (GoF), together have fundamentally shaped our today's understanding of Design Patterns. The GoF classified Design Patterns based on two criteria: The first criterion is referred to as the purpose with the dimensions creational, structural and behavioral, whereby the second criterion is named scope which is further sub classified into class and object (Gamma et al., 1995). The graphic given below shows the Design Pattern space after GoF.

		Purpose		
		Creational	**Structural**	**Behavioral**
Scope	**Class**	Factory Method	Adapter (class)	Interpreter Template Method
	Object	Abstract Factory Builder Prototype Singleton	Adapter (object) Bridge Composite Decorator Façade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Momento Observer State Strategy Visitor

Figure 47: The classification of Design Patterns after GoF: Creational Patterns talk about Object creation, Structural Patterns deal with Object and Class composition while Behavioral Patterns are concerned with Object interaction and distribution of responsibilities.

Another classification of Design Patterns was done in Froehlich's work on object-oriented frameworks. He identifies three levels of design patterns: Architectural Patterns, Design Patterns in a more specific sense and Idioms (Froehlich et al., 1997).

Architectural Patterns

Architectural Patterns describe a general high-level architecture for applications and are closely related to architectural styles. Many patterns incorporate a particular style rather than defining an own, distinct style. Exposing expertise in software architecture, they identify specific components and connectors that are used to describe functional relationships between the components. In an object-oriented paradigm, for instance, objects communicate with each other through object methods which, in this particular case, represent the connectors between the objects.

FEPP

Nemann et al. (2009) highlighted in their work about the reengineering of deprecated component frameworks the importance of a well though-out and clean framework architecture with respect to future maintenance and extension development. According to Neumann et al. (2009), the life of a framework does by far not end after the framework development has completed. The subsequent phases of application, maintenance and discontinuance would every now and then require certain investments from the EBF provider, if an adoption of changes to the framework domain is anticipated. Otherwise, the EBF framework would soon not be applicable anymore and thus not be used.

Avoiding vs. Enabling Maintenance

The two concepts of "Avoiding Maintenance" and "Enabling Maintenance" (Neumann et al., 2009) describe how an implementation of the Framework Key Characteristics (FKC), such as reusability, ease of use, extensibility, flexibility, completeness and consistency, can help to cut down on the complexity and thus on the cost of future framework maintenance and extension development. A high pervasiveness of the FKC can be achieved by implementing what the authors refer to as "Framework Engineering Principles and Practices" (FEPP). Design Patterns, for example, provide solutions to reoccurring software design problems that have proven to work in practice (Gamma et al., 1995). They could support the development of the EBF framework by improving the flexibility and enhancability of the framework's Hot Spots. Standard Conformity -yet another FEPP- could help to improve the consistency of the overall EBF architecture and code. Incorporating standards would not only improve the framework quality (standards go hand in hand with principles and best practices), but also could help improving the flexibility of the EBF (Neumann et al., 2009).

Quick Review - Framework Development Techniques

After abstract classes were distinguished from concrete classes, composition was opposed to inheritance. Composition tends to be easier to use, whereby inheritance provides a higher degree of flexibility. Hot Spots were identified as the means of customization, whereas Frozen Spots represent the framework's invariant parts. Design patterns and architectural patterns were introduced to be particularly useful for designing a framework's Hot Spots. Furthermore, it was shown how considering the FEPP throughout the EBF design and implementation can help to dramatically cut down on the cost of future maintenance and extension efforts.

4.8 Characteristics of Component Orientation

Components

Component coupling and communication has been a highly investigated research domain during the last two decades. Several attempts for defining the concept of components were made. Szyperski (1996), for example, defined a component as "a unit of composition with contractually specified interfaces and explicit context dependencies". Pereira and Price (2002) add that "a component can be implemented using various technologies", thus being absolutely independent from a certain paradigm or programming language. The essence of above definitions is that any software artifact that provides an interface can be considered as a component. Thereby, an interface is "a collection of service access points (Figure 48), each of them including a semantic specification" (Bosch, 1997).

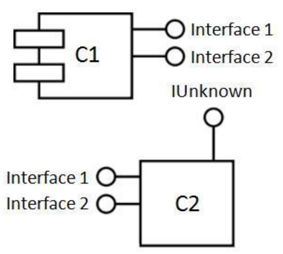


Figure 48: Components (C1, C2) with their access points (Interface 1, Interface 2) and a not defined access point (IUnknown).

Benefits of Component Orientation

The benefits of component-oriented design can be found in an enhanced language and platform interoperability as well as a separation of interfaces and implementations (Eskelin, 1999). Existing and newly constructed components can be deployed to clients and servers to build flexible and reusable solutions. This flexibility allows organizations to quickly adapt to changes in their business and thus enable an overall enhanced customer satisfaction. Regarding an organization's efficiency, component orientation can help to dramatically cut down on the cost for maintenance and extension development by simply replacing old components or adding new ones. Ultimately, component orientation picks up the ideas and guidelines of object orientation, but rather applies them to a more macroscopic view of things. All of the above described criteria in one way or another include the requirements to the EBF framework, which were formulated at the beginning of this chapter. Not only because of this high coverage, but also because it has evolved as a de facto standard in systems engineering, the EBF-AF will follow a component-oriented approach.

Component Interaction Patterns

Software systems are often composed of several components, each of which can be considered as a computational entity that realizes a particular function (An & Peters, 2003). Components interact with each other by communicating through their interfaces. Guaranteeing for a seamless communication in a system consisting of custom and pre-built components, however, can be difficult because of hidden dependencies, complex interactions and obscure design. Eskelin (1999) described in his work five patterns that ease the assembly of components to communicate, collaborate and coordinate to get their respective tasks done. In the following some of those component interaction patterns will be briefly summarized[10], whereby the significance of Design Patterns in general can be reviewed in 4.7.

- **Third-Party Binding:** A common problem in distributed systems is that components operating in one environment need to communicate with components operating in another. Even when in the same component environment, often incompatibilities evolve as the result of type incompatibilities. Furthermore, while steering the component design towards lose coupling, a side effect is that components are designed to be autonomous

[10] Please note that the list of patterns to enable component interaction and communication by all means is not complete. Component Glue, for example, represents another solution to achieve the same, but tends to be more proprietary.

Third-party Binding

and do not interact with their peers. In other scenarios, data needs to be joined together from multiple data sources and sent to one component for processing. Data retrieved from databases, legacy data or from other components often previously needs to be combined before sending it to a component, because this component was not initially designed to do so (Eskelin, 1999).

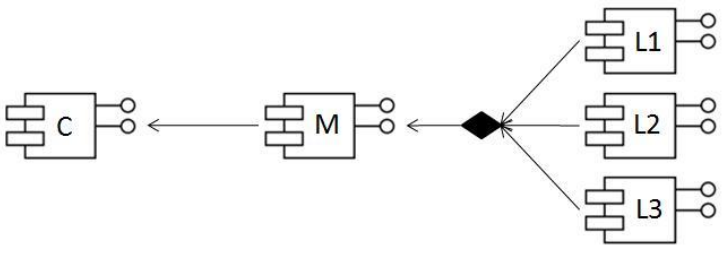


Figure 49: Data from legacy systems (L1, L2, L3) is combined by a mediator component (M) and prepared for input into C.

One solution to connect incompatible components in situations like these is to build a third component acting as a mediator or adapter that implements the interfaces required by the client and performs a translation to the interface the server implements (Eskelin, 1999). In Figure 49, a mediator component M receives input from three legacy components L1, L2 and L3. M was specifically built to be able to interact with the legacy components as well as the target component C, thus representing some sort of bridge. M transforms the input from L1, L2 and L3 into C's target format and sends it via its interface to C.

Abstract Interaction

Abstract Interactions: Describing the way in which components interact is not always straightforward and often only insufficiently supported by component vendors. Since documentation is often generated by tools which by no means are capable of extracting sufficient information about the component interaction from the source code, component applicants or assemblers are likely to notice incompatibilities earliest when they try to integrate those components with each other. While the best description of a component's way of interaction can be found in the source code itself, vendors often hesitate to distribute the source code together with the binaries.

Interaction Protocols

The solution to this problem is to represent interaction protocols with abstract interfaces that define them. A component's dependency on its environment can be reduced by defining interaction protocols between components separately from the components themselves (Eskelin, 1999). The interactions are to be specified in terms of abstract interfaces and additional components to be implemented to communicate with each other through them (Eskelin, 1999).

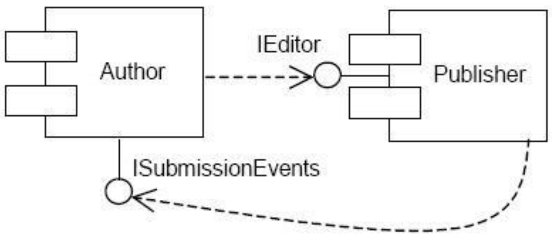
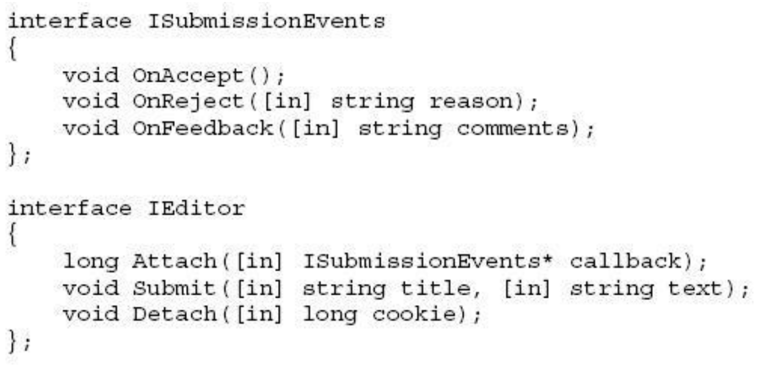


Figure 50: Abstract interaction between an author and an editor (Eskelin, 1999).

In above Figure 50, an abstract interaction between an author and a publisher is described. The author connects, binds and submits text to the editor via the IEditor interface. The publisher sends events (e.g. acceptance, rejection, iterative feedback) back to the author via the ISubmissionEvents interface. The interaction protocol between the author and the publisher is given by the following:



```
interface ISubmissionEvents
{
    void OnAccept();
    void OnReject([in] string reason);
    void OnFeedback([in] string comments);
};

interface IEditor
{
    long Attach([in] ISubmissionEvents* callback);
    void Submit([in] string title, [in] string text);
    void Detach([in] long cookie);
};
```

Figure 51: Interaction protocol between author and editor (Eskelin, 1999).

Component Bus

- **Component Bus:** Systems that consist of components with many interdependencies can behave unpredictably or fail to operate altogether, if explicit bindings they depend upon are not established properly or connections are lost (Eskelin, 1999). Furthermore, it can become very difficult to coordinate the process of instantiating and binding components, when there are many dependencies between them. When, for example, components have an explicit interface they depend upon in each of the other components, the number of dependencies is N * (N – 1), with N representing the number of components involved.

Dependencies

Dependencies can negatively impact the ability to plug in components and just let them perform their specific task, when it's required to do so. In distributed systems, as the EBF-AF strives to be one, where load balancing is performed in reaction to unexpected usage volumes, components might need to be migrated to other servers or new ones need to be instantiated to handle the extra volume. In those situations, hard-coded dependencies can cause performance bottlenecks or failures that are extremely difficult to find (Eskelin, 1999).

One possible solution aims at the centralization of all dependencies in some sort of directory component. Each time a component wants to interact with other components, it queries the directory component for the respective names and establishes a connection. However, in case of a connection failure without warning, from the point of failure until the connection is reestablished and data is updated, data and notifications from the failed connection won't occur and integrity is seriously threatened. Without a more flexible approach that maintains consistency, reliability seems to be virtually impossible.

Dynamic Information Routing

The component bus pattern represents a solution to above described problems by allowing components to communicate indirectly with each other, through a so called information bus (Eskelin, 1999). This information bus manages the routing of information between participating components, from those that produce information to those that consume it. According to Eskelin (1999), routing of information is managed dynamically, as each participant can attach and detach to and from the bus without affecting the integrity of others. By attaching to the bus, components register interest in the information they require. When one particular component places information onto the bus, it is delivered to those participants that registered interest. Not only the connections, but also the data passed around are centralized. Instead of having N * (N – 1) dependencies between communicating components, now only 2 * N dependencies are to be managed using the bus (Eskelin, 1999).

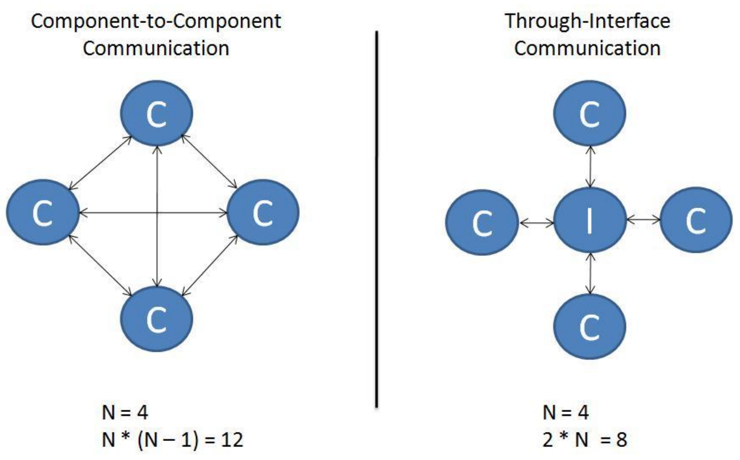

Component-to-Component Communication
C
C
C
C
N = 4
N * (N – 1) = 12
Through-Interface Communication
C
C
I
C
C
N = 4
2 * N = 8

Figure 52: Component-to-Component communication vs. through-interface (bus) communication.

EBF Component Bus

Regarding the EBF, out of the above introduced component interaction patterns the component bus seems to be the most interesting one. While third-party binding and abstract interaction are often applied as solutions for integrating monolithic components with each other, the component bus requires all participating components to implement certain interfaces. Hence, the component bus pattern is more likely to be applied in the course of from-the-scratch development and aims at getting things right from the very beginning. As the EBF will be designed from scratch, the component bus pattern becomes eligible for implementation. It clearly stands the cleaner solution for the problem of integrating components with each other as compared to abstract interaction or third-party binding.

Quick Review – Characteristics of Component Orientation
Component orientation reveals its benefits in the development of larger software systems. Breaking down a system's architecture into several logical parts enhances the maintainability as well as the reuse of functionality. However, when separating functional parts from other functional parts, the problem of how to let those parts communicate with each other evolves. The component interaction patterns "abstract interaction", "third-party binding" as well as "component bus" aim at reintegrating the previously split components. With respect to the EBF, the component bus pattern seems to be the best fit as it not only describes the cleanest solution, but also can only be realized when systems are built from scratch.

4.9 Summary

Designing the EBF as an application framework rather than a stand-alone application would reveal many benefits, which would become especially interesting when looking at maintenance and extension. Applications that are developed from the EBF, for instance, would automatically benefit from fixes or modifications that happened to the framework.

When trying to classify the EBF as either a Blackbox or a Whitebox framework, it turned out that it would rather become an architectural hybrid as it combines characteristics of both. Furthermore, the EBF, due to the nature of its business (e-commerce), was identified as a typical domain framework.

From a high-level perspective, the functionality of the EBF was distinguished into core and library. Thereby, the core would encompass parts that provide core functionality as well as parts that represent the infrastructure for the interaction of all functional units. The library would consist mostly of external functionality or extensions that are embedded in the EBF.

The FEPP were introduced as principles and practices that can help to significantly cut down on the complexity of future maintenance and extension efforts. Aiming at "Avoiding Maintenance" and "Enabling Maintenance", the design of the EBF will incorporate the FEPP to a high extent from the very beginning.

Lastly, it was shown which benefits a strict componentization of the EBF would unleash. In order to enable an efficient communication and integration of EBF components with each other, the component bus pattern was proposed for consideration in the EBF design.

Chapter five will apply the insights gained from the elaborations in this chapter to the design of the EBF Application Foundation.

5 The EBF as a Framework

5.1 Overview

This chapter is going to analyze methods for implementing the EBF Product and Process Model (EBF-PPM), which was developed in chapter three. It will be shown how the EBF-PPM is going to be wrapped by a framework that carries the name "EBF Application Framework" (EBF-AF). While demonstrating their practical relevance, many concepts of framework engineering that were introduced in chapter four will be picked up again and significantly guide the design of the EBF-AF and its architecture.

Amongst others, the EBF-AF will have to provide answers to the challenging question of how to integrate single components with each other and how to let them communicate. The architecture of the EBF-AF will play a crucial role regarding how easily extensibility and maintainability can be achieved. While having to cater for all the functional requirements that evolved from the product and process modeling, the EBF-AF will have to ensure a high degree of flexibility. Changes in the domain must be adoptable quickly and efficiently by adding a new component or modifying an existing one. The architecture of the EBF-AF will have to facilitate those changes without exposing an impact on other components or parts of the framework.

The sections contained in this chapter gradually refine the EBF-AF design and clarify concerns that are related to its feasibility. Following a top-down approach, the elaborations begin with the general architecture of the EBF-AF, then focus upon finding a platform for the realization of this architecture and finally conclude in the concrete implementation of the product and process model. Section 5.2 Requirements Analysis will formulate requirements to the EBF-AF from a business perspective and from a technical perspective. Thereby, those requirements are partially related to the requirements of the EBF Product and Process Modeling covered in chapter three. Section 5.3 will detail a proposal for how to arrange and distribute all bits and pieces of the EBF-AF in a modern enterprise architecture. One specialty of the EBF-AF will be its Integration Foundation (EBF-IF) that is introduced in section 5.4 and proposes how to provide a solid way for letting individual EBF components communicate with each other. After the general architecture of the EBF was defined, section 5.5 will attempt to identify and decide for base technology upon which the EBF-AF is going to be implemented. While trying to utilize an objective decision criterion, the Total Cost of Ownership (TCO) as well as the opportunity cost of existing alternatives will be taken into account. Finally, section 5.6 and 5.7, based on the chosen technology, will detail suggestions for how to implement the EBF Product and Process Model in reality. Thereby, the last two sections at several places become very technical and detailed and sometimes argue from one specific direction only. However, certainly also other approaches towards realizing the EBF-PPM might have been possible. For those readers who do not have a special interest in the technical realization but want to focus more upon the theoretical concept behind the EBF-AF, section 5.6 and 5.7 are designed in a way that they can be easily skipped.

5.2 Requirements Analysis

For the purpose of being easier to understand, the requirements to the EBF-AF are split into business requirements and technical requirements. Business requirements are requirements that are mostly business-driven and reflect a key success driver towards achieving the business goal. Technical requirements on the other hand define certain technical constraints to the framework that is going to be designed and only indirectly influence the business goals. This sub categorization, however, does not have any further purpose throughout this chapter than providing a better overview.

Business requirements:

Full feature support of the EBF Product and Process Model (EBF-PPM): Chapter three discussed a design for a product and process model that is general enough to be able to reflect any arbitrary e-commerce transaction. This product and process model that was referred to as the EBF-PPM will in this chapter serve as a foundation for the design of an application framework. Thereby, the design, amongst others, will have to provide implementations for the following points or incorporate them in one way or another:

The EBF Product Model: The implementation of the EBF Product Model (see 3.3 The Product Model) will be the enabler for the processing and management of product-related data. All the requirements to the product model that were formulated in section 3.2 Requirements Analysis will of course also apply to the implementation of the product model.

The EBF Process Model: The implementation of the EBF Process Model (see section 3.4 and 3.5) will add the process flow of an e-commerce transaction and its functionality to the framework.

Product Domain independence: Chapter two discussed the importance of product domain independence for the EBF-AF. The goal of the implementation of the EBF-PPM is to be able to reflect product descriptions for every arbitrary type of product, may it be a good, a service or a financial instrument.

Low Total Cost of Ownership (TCO): Due to the nature of the business and the primary target group of the EBF not being able to afford huge investments into IT, the overall cost of the EBF from procurement to operation and maintenance needs to be kept at a minimum. This overall cost is referred to as Total Cost of Ownership (TCO). Two components of the TCO are the development cost and the cost of operation (more details on TCO in 5.5 Base Technology Analysis for the EBF-AF).

Low Cost of Development: As the EBF is primarily -but not exclusively- targeting the support of e-commerce for SME and those businesses by nature can only afford relatively small investments, the implementation cost of the EBF-AF on the applicant's side needs to be kept as low as possible. As, however, this implementation cost would to a high extent be determined by the expenses for the EBF-AF, the total development cost of the framework by itself needs to be low. This can be achieved by leveraging existing standards and technologies as well as targeting a possibly large domain and number of applicants.

Low Cost of Operation: But not only has the cost for an investment into the EBF-AF to be small. Almost more important is that the cost of operating the framework is also low. In order to reduce the permanent expenses required for running the EBF-AF, one of the framework's basic rationales is to represent itself as SaaS – Software-as-a-Service (SIIA, 2006). This means that the EBF-AF will support a business model in which the customer does not need to provide any infrastructure for running the framework, but rather agrees upon a hosting contract, whereby the framework provider hosts the EBF-AF on remotely administered infrastructure, while the applicant uses a web front-end to access the EBF-AF over the internet. Thus, the only piece of infrastructure that is required on the applicant's side is a standard PC with an internet browser and a connection to the internet. This model not only has the advantage of the applicant saving on cost for the infrastructure, but also ensures the security and integrity of his data and makes maintenance significantly easier. Of course, if the applicant wishes to have the EBF-AF running on his own infrastructure (e.g. due to privacy concerns or concerns related to the confidentiality of information), an (offline) version of the EBF-AF would also be supported.

Integratability: Even though the EBF-AF tries to provide a complete solution for any kind of e-commerce, no matter whether an e-store or an auction system, the framework must be capable of being integrated with other third-party systems. An applicant who is interested in the EBF-AF might already have an ERP system in which he tracks his product data or a CRM solution which he uses for one-to-one marketing (Sunil et al., 2006). In such scenarios, the EBF-AF would need to be capable of acquiring the product data for its product model from the ERP system and also update the CRM system with information on which products were recently bought and by which customer. Therefore, the EBF-AF has to provide certain interfaces to communicate with third-party systems.

Technical requirements:

Platform Independence: As the EBF-AF targets a huge variety of different applicants with sometimes very different businesses, it must not be dependent on a particular platform. Since the market targeted cannot always afford excessive investments into software and infrastructure, it is even more likely that the EBF-AF will need to be able to run on a free or open source platform. Therefore, a carrier for the EBF-AF must be found that can easily be hosted on any platform or operating system.

Object Orientation: In coordination with other requirements, such as "easy-to-access extension development and maintenance", and not at least because object orientation has evolved as a de-facto standard in software engineering, the functionality of the EBF-AF will be strictly encapsulated and componentized.

Lose Component Coupling: To support the SaaS business model that was mentioned earlier already and to furthermore maintain a very high degree of flexibility within the framework architecture, the EBF-AF components are to be loosely coupled. Lose coupling supports the replacement of existing components with later versions of the same or the addition of new components.

Scalability: Even though the EBF-AF is meant to support SME in the first place, maybe even larger enterprises could potentially benefit from the application foundation as well. To furthermore support businesses with a high transaction volume, such as auction-based platforms, the overall architecture of the EBF-AF requires a high degree of scalability. Thereby, not only databases, but also the application framework itself as well as the web front-end need to be able to adjust to varying load rates and possibly implement certain load balancing mechanisms.

Physical Distributability: The SaaS theme together with the "scalability" and the "lose component" requirements form the need for a high degree of physical distributability of the whole application framework. This means that all the components in the EBF-AF should be able to run on separate platforms while communicating with each other over the internet or a LAN.

Easy-to-access Extension Development: As Neumann et al. (2009) outlined the crucial role of an architecture that easily allows for extension, the design of the EBF-AF of course should also implement their Framework Engineering Principles and Practices (FEPP). According to Neumann et al. (2009), those FEPP cannot only help to avoid maintenance, but furthermore also play a key role in enabling maintenance and extension development. A good framework architecture enables extension development in a way that significantly cuts down on time and effort as compared to a very complicated architecture with many interdependencies between classes and components. Last but not least, the success of the EBF Application Foundation will depend on how well the architecture allows for an adoption of changes to the EBF's problem domain (details on framework domains can be reviewed in chapter four).

Easy-to-access Maintenance: In line with the previously discussed need for an easy-to-access extension development, an easy-to-access maintenance can also be achieved through an implementation of the FEPP Neumann et al. (2009) proposed. "Avoiding maintenance" and "enabling maintenance" start with the design of the EBF-AF following the theme "The better the design the better the implementation".

5.3 The EBF Application Foundation Architecture

After the requirements to the EBF-AF were detailed above, the elaborations on a design proposal for a framework that implements those requirements can begin. This section will address the question of how the EBF framework will be deployed and used. Traditionally, internet-based applications relied upon a client-server model. In this model, a client (in many cases simply a web browser) connects to a server and queries the server for the processing of data. The server, in the second instance, sends back (or "serves") the result to the client and the transaction is complete (Renzel & Keller, 1997).

Client/Server

Since the EBF-AF represents an application framework for electronic commerce over the internet, some sort of client-server architecture seems to be involved as well. Traders, for example, could log into the framework using a standard web browser on their client machine, while the actual framework is hosted by a service provider on a designated remote server.

However, implementing a client-server architecture could restrict the framework in its flexibility to adopt to varying loads as client-server typically shows difficulties with allocating new resources at runtime (Orfali et al., 1994). As this restriction stands in contrast with the scalability requirement, a more sophisticated way is to be found. Furthermore, a good componentization of the business logic would not be supported by a client-server approach, which might negatively impact later maintenance and extension efforts (Neumann et al., 2009).

Three Tier Architecture

During the 90s, an architectural pattern evolved as successor of the client-server paradigm that solved the previously mentioned problem of not being able to scale. By splitting the functionality of an application into a presentation tier, an application tier and a data tier, adding more processing power to an application has become as simple as just launching new application servers and adding them to an application server pool (Rautenstrauch & Schulze, 2002). Thus, in times of high traffic load on an application, the size of the application server pool can be increased, whereby in times of less traffic, the number of application servers can be reduced (Hernandez et al., 2005).

The three-tier architecture does not only have the advantage of being able to scale, but it also keeps the application logic and the manipulation logic for the data that the application works upon separate from each other. The advantage of a centralized data repository is that it is easier to maintain and to coordinate (Heuer & Saake, 2000). Ultimately, the "easy-to-access maintenance and extension development" requirement that was formalized in the requirements analysis should gain a clear benefit from this separation as application developers can concentrate on maintaining the application layer. The support for the database and the data manipulation logic would be provided by another instance.

Wrapping the EBF-PPM

As earlier indicated in chapter three, the EBF-AF is meant to fulfill the purpose of wrapping the EBF Product and Process Model (EBF-PPM) into an application framework while providing an architecture that is capable of catering for all the requirements stated in the Requirements Analysis. In addition, the EBF-AF will have to provide a certain set of support functionality (e.g. user management) as well as interfaces for the import and export of data and should be able to easily adopt extensions in the form of new components.

A possible implementation of those requirements using a three-tier architectural approach could look as shown in Figure 53:

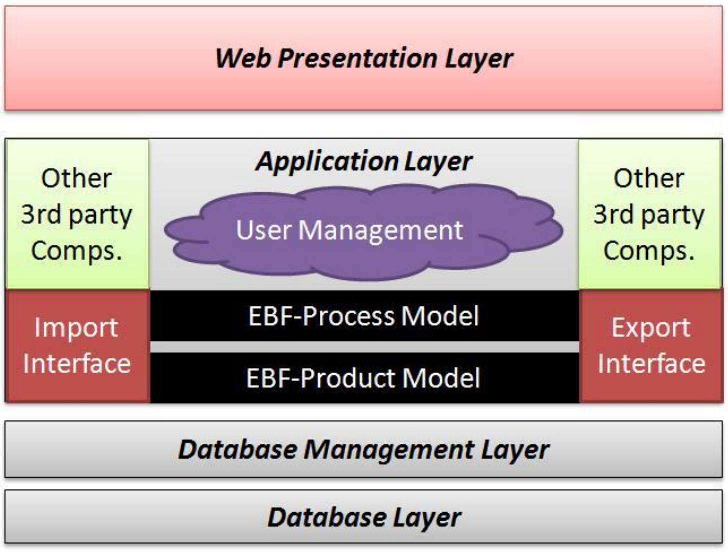


Figure 53: The EBF Application Foundation in a modern three-tier architecture.

EBF-AF Stack

Figure 53 depicts that the product model would function as the bottom most instance in the application layer. It would serve as a foundation for all the operations within the application layer. Based on top of the EBF-Product Model would be the EBF-Process Model, which would represent a container for all transaction-related processes and actions. Additional third party components could also be linked to the product model or even extend the process model. Data import and data export would be guaranteed through import and export interfaces. Finally, a user-management component would operate within the application layer and could become an integral part for rights and access management.

In Figure 53, all the EBF-AF business logic is concentrated in the application layer. Even though it is a part of the three-tier architecture, the web presentation layer would merely have the form of a standard web interface to the business logic of the application framework. It of course could be customized by other parties to fit their individual requirements. Thereby, no big programming expertise should be necessary to modify the standard web interface, but this task should rather be reducible to a matter of simply adding or removing certain (HTML) tags in a template. The database layer also would not directly relate to the application framework itself as it merely would encompass the functionality for the modification of data as well as the data storage. If no existing standard solution could be leveraged here, the development of the database layer would become a more or less one-time investment and should never be the subject of any customization initiatives. All customization and thus all adoption to the different needs of the individual framework applicants should be achievable through the application layer itself. However, having the database layer separate would certainly ease backup and data maintenance efforts significantly.

Focus upon Application Layer

Even though the EBF-AF primarily targets small businesses, it could as well happen that one of those businesses wants to run an online auction platform, similar to eBay. In this case the expected workload on the EBF-AF could potentially become very high. On the other hand, the EBF-AF could be extended in a way that adds a component which implements time consuming business logic. Therefore and not to block the EBF framework applicant from building scalable applications, it would be useful, if the application layer could automatically adapt to varying traffic loads. This mechanism, which is also referred to as “load balancing”, assumes that the application layer is distributable. However, distributing or in other words “clustering” the application layer of the EBF-AF to several application servers would require the presence of a master server that firstly would accept all queries from the web clients and secondly would forward individual queries to the application server with the lowest current work load. Referring to the nature of its functionality, such a master server is also called “dispatcher” (Hernandez et al., 2005). Figure 54 demonstrates how the initial application layer shown in Figure 53 is projected onto the “EBF. App. Server” and how the new application layer would contain an “EBF App. Server Dispatcher”.

Framework Clustering

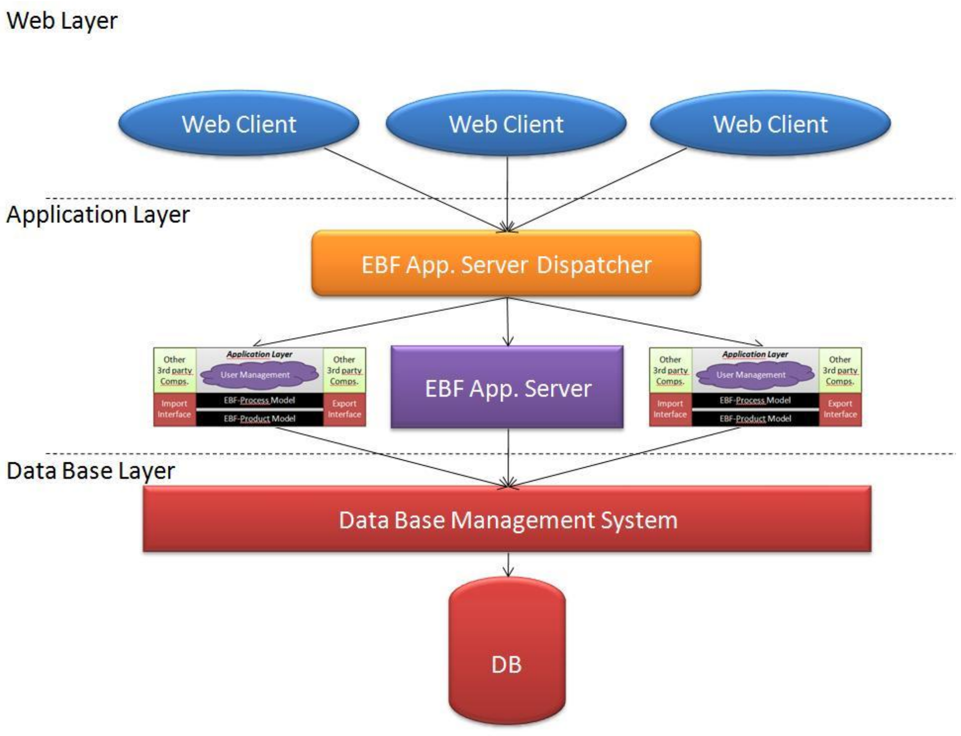


Figure 54: The EBF Application Layer clustered amongst several application servers and managed by a dispatcher.

Dispatcher

While the EBF App. Server Dispatcher organizes the assignment of EBF App. Servers to web client inquiries, each and every app server can independently communicate with the database management system. To synchronize the concurrency of the application servers and thus guarantee the correctness and integrity of the data, the database management system needs to apply additional logic. While one application server, for example, reads or writes to the database, other application servers must be hindered from doing so at the same time. Otherwise, the consistency of the data cannot be ensured. Therefore, the respective parts of the database are typically locked as long as one application server operates on it while the requests of other application servers are enqueued and put on hold until the database was unlocked again. This procedure assumes the existence of a designated Enqueue and Lock Server in the Database Layer/ Database Management System (Hernandez et al., 2005). As the main subject of this chapter, however, is the design of the EBF-AF and its application layer while taking into account that there is already a huge variety of Database Management Systems existing (Oracle[11], Microsoft SQL Server[12], IBM DB2[13]) that exactly provide this functionality, above explanations on the database layer cannot exceed the size of a brief mentioning.

[11] http://www.oracle.com
[12] http://www.microsoft.com
[13] http://www.ibm.com

Goal Assessment

After general architectural questions were discussed throughout the last section, it seems reasonable to attempt a pre-assessment of which requirements were addressed so far:

Full feature support of the EBF Product and Process Model (EBF-PPM): As the most fundamental requirement, the EBF-AF implements the EBF Product and Process Model (EBF-PPM) that was developed throughout chapter three in its full complexity. The implementation is based upon a framework-like component architecture and deployed on a so called application server. Furthermore, the framework core of the EBF-AF was augmented by a dispatcher and a user management component.

Physical Distributability: The EBF-AF is deployed on an application server which is part of a three-tier architecture (presentation, application, database). This separation allows for a logical as well as physical distribution of the individual tiers onto several computers and furthermore ensures that multiple EBF-AF instances can run on different hardware.

Scalability: The implementation of the "physical distributability" requirement also allows for scalability of the framework's likely bottleneck – the application layer. One of the resulting key benefits is that now, depending on the work load on the framework, a so called dispatcher can dynamically increase computing power by adding additional application servers to the application server pool. This mechanism is commonly referred to as "load balancing" (Hernandez et al., 2005).

Quick Review – The EBF Application Foundation Architecture

In order to keep data logic separate from business logic, the EBF Application Foundation to will implement a three-tier architecture consisting of a presentation layer, an application layer and a data layer. Furthermore, the separation of presentation from business logic allows for load balancing in times of varying traffic on the framework. Thereby, a so called dispatcher has the task to monitor the current load on all EBF-AF application servers and distribute incoming requests in a way that an even capacity utilization is achieved. Since the database layer could be simply represented by a standard database management system and the presentation layer merely encompasses the preparation of visual output for the user, hereafter the actual focus of the design of the EBF-AF will lie upon the application layer that implements the business logic.

5.4 The EBF Integration Foundation

The last section described the general architecture of the EBF Application Framework, whereby one of the key points was the deployment of the EBF-AF on multiple application servers. Being able to distribute the framework, amongst others, allowed for the implementation of a high scalability. Throughout this section, the previously gained macroscopic view is going to be further refined into a component-based view. Figure 53 showed of which components the EBF-AF consists and into which layers the framework can be logically subdivided. It was further indicated in chapter four that the EBF would consist of a framework library and a framework core (see Figure 36). The library reflected a logical container for all third-party modules while the core encompasses the EBF-PPM components as well as the actual infrastructure that enables a smooth interoperation and intercommunication between all components. The latter mentioned infrastructure in the proceedings of this work will be referred to as EBF Integration Foundation (EBF-IF). The naming of the EBF-IF evolved from its so crucial function to the framework of seamlessly but loosely integrating all the EBF components.

EBF Component Bus

Chapter four already contained an introduction to the world of component engineering and showed what value component orientation can add to software development projects. A set of component interaction patterns was discussed, whereby the component bus pattern turned out to potentially be the most interesting approach for the EBF. The following elaborations on the EBF Component Bus refer to the realization of the EBF-AF application tier, shown in the initial and rather macroscopic architectural proposal of Figure 53. The EBF Component Bus will serve the role of being the foundation for the integration and communication of all EBF components within the application tier.

While third-party binding and abstract interactions were discussed as possible alternatives to a component bus, the latter one shows clear advantages over the competing patterns. In the end, the benefits regarding dependency management and an increased flexibility of the component bus pattern clearly indicated a possibly high coverage of requirements (see section 5.2).

The EBF Component Bus -as the technical implementation of the EBF Integration Foundation- in the first place behaves like a normal component bus. EBF components can (dynamically) attach to and detach from the bus, while the bus manages the routing of messages and data between the individual components.

Product and Process Model

Even though from a technical perspective treated the same way as each and every other EBF component, two of them play a major role. The product model represents the container that not only stores the product data, but also allows access to and management of this data via certain interfaces. Without the product model, all the other components could not fulfill their tasks as they would simply lack the fundamental product data to operate upon. For this reason and to emphasize its fundamental role for the EBF-IF, the product model is placed at the lower end of the T-like EBF component bus (see Figure 55). Another justification for the T-structure of the bus is that from a data abstraction point of view, the product model is the component that is lowest and nearest to the database which stores the product data. All other components (process model and third party components) are only able to access the product data via the product model. For this reason, the degree of data abstraction increases, the further on the bus the components are away from the product model (Figure 55).

T-Bus

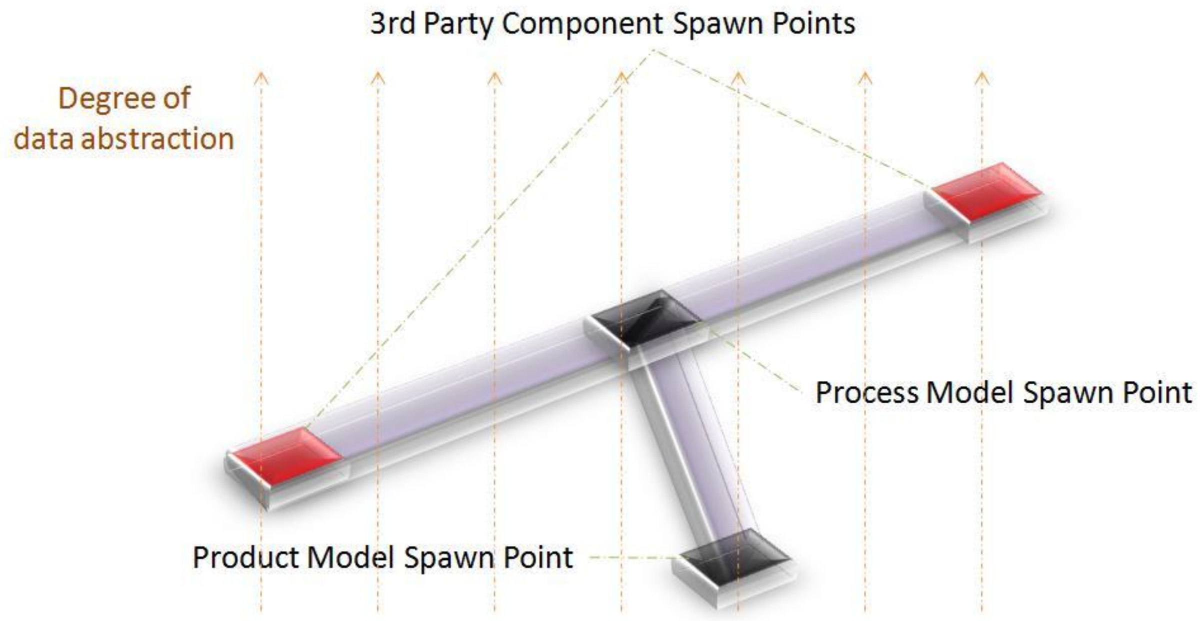

3rd Party Component Spawn Points
Degree of data abstraction
Process Model Spawn Point
Product Model Spawn Point

Figure 55: The EBF-AF Component Bus and its Spawn Points.

To actually attach a component, the EBF-AF Component Bus defines what is called “Spawn Points” (in Figure 55 the red and black rectangles). Spawn Points represent the places on the bus, where components can be “spawned” referring to a metaphor of the instantiation of a component. In order to be able to be spawned, a component needs to implement the EBF Component Bus interface[14], a set of conditions and rules which a component needs to follow, when joining the bus. This way, it is guaranteed that each component can post and receive messages in the same way and thus a seamless component communication is possible.

Spawn Points

By taking a closer look at Figure 55, it can be seen that Spawn Points are distinguished into red and black. This classification goes back to the distinction of the EBF Application Framework into core and library. While the black Spawn Points (meant to host the EBF-PPM) together with the component bus itself represent the framework core, the red Spawn Points are intended to host

Red and Black

[14] Please note that it would by far exceed the timely frame of this work to provide a detailed description of the bus interface and the component binding mechanism. Furthermore, it will be shown throughout the next section that already existing standards can be leveraged to implement the EBF Component Bus and that those standards sufficiently abstract from the component binding and interface implementation process.

framework library components, such as third party components or possible future extension components (for more detail, please review section 5.3 The EBF Application Foundation Architecture).

Hosting Components

As simple as described in the previous explanations on the component bus pattern, EBF components attach to and detach from the EBF Component Bus. The following graphic demonstrates the EBF Component Bus in a state where it is hosting the EBF Product Model, the Process Model as well as a third party ERP[15] and a CRM[16] component.

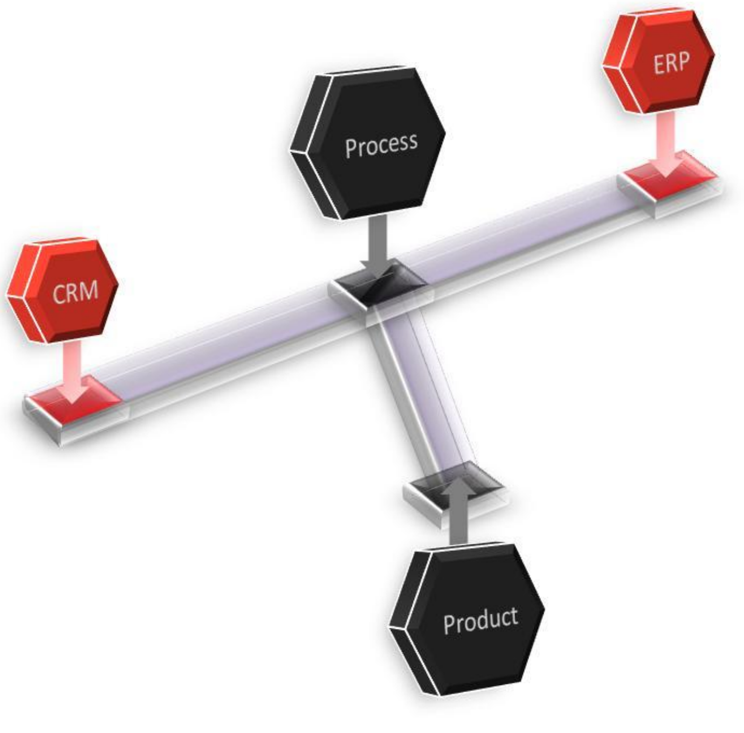


Figure 56: The EBF-AF Component Bus with components spawned on the Spawn Points.

Changing Colors

Despite the fact that Figure 55 only shows two red Spawn Points, of course the number of those spawn points is virtually unlimited. It might also be worth a thought to change the color of a yet red Spawn Point to black, which would literally indicate the transition of a former third party component to the EBF framework core.

So far, it was shown how the EBF Component Bus integrates individual components with each other by guaranteeing a loose coupling and a seamless communication. But how are whole third party systems that consist of numerous other components connected to the EBF framework? One solution could be to trait those systems simply as third party components and require them to

[15] Enterprise Resource Planning: Might be used to add business intelligence and procurement functionality to the EBF.

[16] Customer Relationship Management: Might be used to add functionality for a more customer-centric marketing strategy to the EBF.

connect to the EBF Component Bus by implementing the respective interface. In this case the component bus could be used as it is and no further additions would be necessary. However, the unilateralism of this more or less pull-based approach requires some customization on the side of the system that is to be connected. Also, if the other system already used its own component bus and instead might require the EBF to connect to its own bus, above approach tends to not provide a good solution. In this case, it seems like again some sort of third party binding or this time bus-to-bus integration would become necessary.

Integrating Third Parties

A better way that avoids the complexity of the above proposed solutions is to add an import and an export interface to the EBF Component Bus. Via the export interface, third party systems would be able to import all the data from the EBF that is available on the bus in a structured and standardized manner. Furthermore, the EBF could import data from external systems via its import interface. The figure below summarizes the architecture of the EBF Component Bus.

Export and Import Interfaces

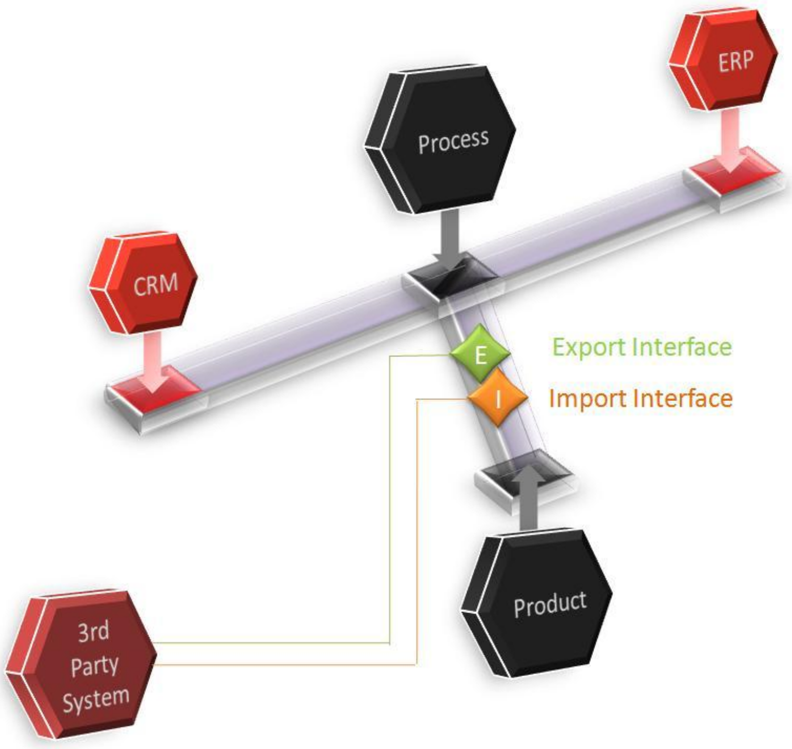


Figure 57: Export and Import Interfaces on the EBF Component Bus.

Even though the EBF-AF Component Bus describes a theoretical concept of how to integrate components with each other, it is not yet clear how this concept could be turned into reality. A recently and also frequently debated hype in the world of enterprise software engineering that could help with answering this question carries the name Service Oriented Architecture (SOA). An SOA describes an architectural pattern for the integration of enterprise services with the goal of achieving a relatively high degree of flexibility in the system while concurrently allowing all services to seamlessly communicate with each other (Müller, 2008).

SOA Augmentation

Service vs. Component

So far, it was only discussed how a strict componentization of the EBF-AF can help achieving the requirement goals. The terminology of a service, however, has not been mentioned before. But what exactly is the difference between a service and a component and how can SOA contribute towards meeting the requirements of the EBF Application Framework? The answer for the first part of the question might lie in the different perspectives at which both a service and a component look at the same thing. In order to find a suitable answer for the second part, however, a more in-deep understanding for SOA needs to be provided.

Defining SOA

According to Nickull et. al. (2007), SOA "is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains and implemented using various technology stacks".

SOA Services

It seems like SOA describes an architectural concept that focuses upon a service rather than a component or a class. While architectural concepts prior to SOA always focused upon the technical perspective on an enterprise system, SOA propagates the concentration on the business side of a system. Therefore, in SOA the business and thus its processes mandate which solution is necessary to achieve the company goals. Services represent the implementation of a process, are exposed through components and can be assembled through a workflow module (Nickull et al., 2007). Several services can be combined or "orchestrated" to together provide an implementation of another service (Müller, 2008). However, a single component could also be enough to implement a service. Coming back to the initial question, the difference between a service and a component apparently lies in the perspective. While a service (from a business perspective) has the task to support a business process, a component (from a technical perspective) has the task to implement the service.

Merging Two Perspectives

While from a macroscopic point of view the earlier proposal for how to implement the requirements to the EBF-AF based on a three tier architecture, the microscopic perspective on how to realize the application tier in particular focused upon the component bus pattern. But how could a combination of macroscopic and microscopic view be appropriately interpreted? The result of joining those two views would give a kind of three-tier architecture that is highly componentized and uses a component bus at the base level of the application tier to enable a smooth integration and communication of its components.

Even though the earlier elaborations on the component bus draw a general picture of how to let the EBF-AF components interact, it is still not yet sufficiently clarified how those components will actually collaborate. An answer to this question could come from SOA.

Adobe's reference architecture for SOA (Figure 58) contains three tiers: the client tier, the service tier and the resource tier. The client tier with its services controller, date/state management, security container, virtual machine, rendering and media as well as communication services reflects all the services of a SOA client application that are necessary to communicate with service-tier services, which implement the actual business logic (Nickull et al., 2007). The service tier with its parts service invocation layer, registry repository, service container and core services as well as service provider interface groups together what is necessary for administrating and running services. The bottom most tier, the resource tier, represents an assembly of EIS, databases, directories, ECM repositories, message queues and legacy systems (Nickull et al., 2007). Hence, it groups the resources together on which the business logic implementing services operate.

Reference Architecture

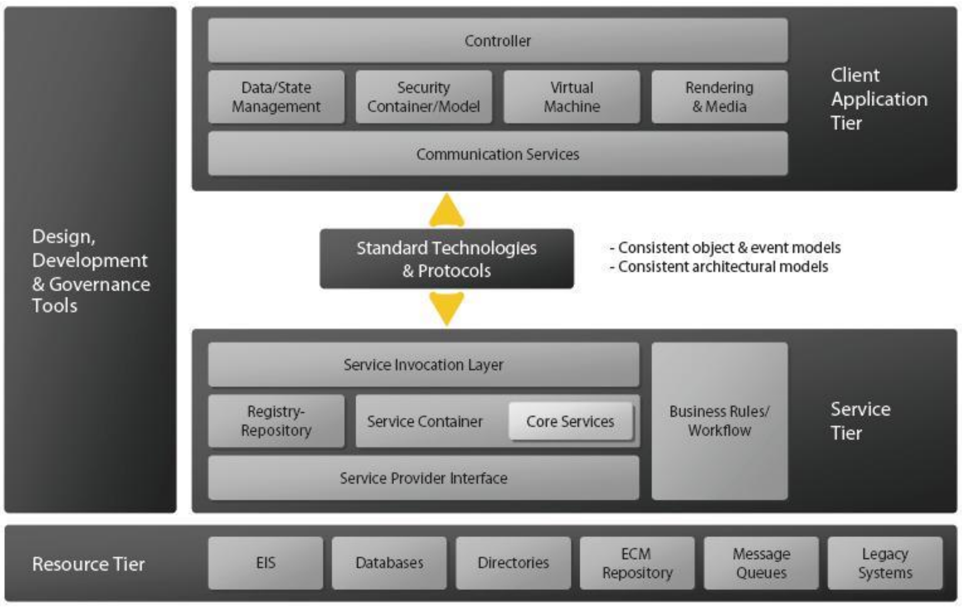


Figure 58: Adobe's SOA Reference Architecture (Nickull et al., 2007).

Condensing the tier descriptions shown in Figure 58 a little bit results in the client tier being a layer that supports end-user interaction, the service tier being the layer that hosts the business logic and the resource tier representing the data source. By accordingly reviewing Figure 55, it is revealed that those tiers are very similar to the presentation layer, the application layer and the database layer of a three-tier architecture. Does this conclusion mean that SOA is a service-oriented three-tier architecture or maybe even an enhancement of it? At least it seems to be worth a consideration as though three-tier defines the separation of presentation, business logic and data storage, it is very vague regarding the seamless but loose integration of the components in the individual layers.

SOA vs. Three Tier

Getting back to the further enhancement of the EBF-AF architecture, SOA made the next step and provided a solution for efficient component interaction, while permanently aiming at a lose coupling of components. Using SOA, IT systems are enhanced by a more flexible and dynamic nature, while concurrently being able to quickly adapt to changes in the business processes. By providing a standardized wrapper for components (in an SOA an XML-based wrapper) and allowing for those components to be accessible over the web via an indexed name, they turned into services.

SOA for the EBF-AF

But how does his help the design of the EBF Integration Foundation? Without having it called out explicitly, the three tier-based design proposal for the EBF Application Framework with the EBF Component Bus as enabler for a loosely coupled but seamless component interaction already turned into an SOA. Thereby, the component bus routes XML messages between the EBF components, which, by adding an XML-based SOA wrapper around them, turn into services. Finally, the EBF Component Bus turns into a service bus, which maps to the concept of a SOA Enterprise Service Bus (ESB).

Recalling the Dispatcher

The final bit that is left to be added is the placement of the EBF App. Server Dispatcher (review Figure 55). In 5.3 The EBF Application Foundation Architecture, it was discussed how the EBF App. Server Dispatcher functions as a naming, routing as well as load balancing processor for the EBF Application servers. In an SOA, the dispatcher would typically be placed in the service invocation layer or registry repository (Figure 58). The SOA-augmented picture of the EBF-AF seems to be complete. The full SOA EBF-AF stack is shown in the following graphic:

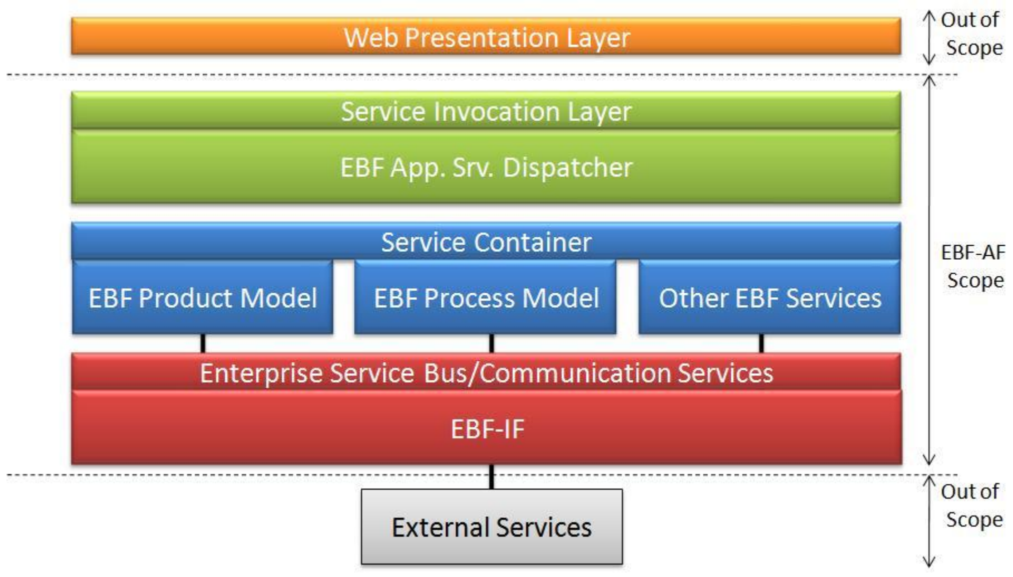


Figure 59: The EBF-AF stack in an SOA.

Goal Assessment

Before moving on to the next section and to furthermore not let the requirements get out of sight, an interim assessment of those seems to be appropriate.

Lose Component Coupling: By introducing an architecture that suggests letting components communicate via an Enterprise Service Bus (EBF-IF), components can be very easily attached to and detached from the EBF-AF. Explicit dependencies between components are avoided by letting them communicate via message exchange. In terms of a service-oriented architecture, the EBF components have mutated into services with the EBF-IF functioning as service integration foundation. Using the EBF-IF, even web services of external service providers can be seamlessly integrated with the rest of the EBF-AF and at anytime unplugged from the framework.

Easy Maintenance and Extension: Earlier in this section, it was indicated how maintenance or extension efforts that are easy to access can possibly be driven at a lower intensity. Thus they would allow for saving on resources and being more cost-efficient. It was also shown that by aiming at a possibly high coverage of the FEPP in the framework design, firstly the need for maintenance and extension can be lowered and secondly access towards maintenance can be simplified. Additionally, the incorporation of the SOA pattern in the EBF-AF architecture supports a loosely coupled interaction of EBF services while making them easy to exchange with newer versions.

Integratability: With the introduction of the EBF Integration Foundation, EBF services can communicate in an efficient but loosely coupled manner. New services can be added to the EBF-AF by simply plugging them into the EBF-IF. Furthermore, existing services can be removed without affecting the integrity of other EBF services. One of the key benefits of the EBF-IF is that it does not distinguish between local EBF services and remote services that are hosted by other service providers. Instead, the EBF-IF treats them similar and hence truly enables a seamless integration experience.

Quick Review – The EBF Integration Foundation
By adding the concept of SOA to the design of the EBF Application Foundation, the component architecture with its component bus turned into a service architecture with a service bus. The advantage of a service architecture over a component architecture lies within the implementation of a standardized way for achieving a loose but seamless coupling of component functionality that is accessible over the internet. The Adobe SOA Reference Architecture describes the individual parts that enable this type of service interaction. The introduction of SOA, however, has not invalidated the rationale behind the EBF Component Bus and the three tier architecture that were elaborated upon earlier. Instead, they were augmented while still being prevalent as integral parts within the final EBF-AF architecture.

5.5 Base Technology Analysis for the EBF-AF

The last section's "Assessment of Goal Achievements" has indicated that, so far, all but two requirements were addressed and implemented. Until now, all explanations and elaborations on the design of the EBF Application Framework were of a theoretical dimension without actually detailing a concrete method of realization. This is good on the one hand as it shows that the design of the EBF-AF is generic and abstract enough to be independent from base technology, concrete platform software and middleware. On the other hand, it leaves the question open how the design of the EBF-AF is going to be implemented in reality.

By performing an evaluation of established industry development platforms, which are produced and maintained by large enterprises and consortia, this section stands a proposal for the concrete realization of the EBF-AF. However, the number of relevant development platforms is restricted to two, namely J2EE and .NET, as currently in SOA-driven software engineering no real alternatives to those exist. At the end of this section, a decision for one of the two technology platforms will be made. Last but not least, it will be briefly discussed which parts of the EBF-AF would map to which parts of the technology platform and how a realization of the EBF-AF could be achieved.

SOA Technologies

As one of the most critical denominators for the decision that is to be found, the Total Cost of Ownership (TCO) plays a major role. When deciding for a base technology, it is often not enough to just consider the plain cost of capital investment into this base technology (e.g. licensing). Even more important is a) the cost of development that is to a fairly high extent determined by the chosen base technology and b) the maintenance cost for the developed software that is also significantly influenced by this choice (refer to 4.6 Economic Significance of Frameworks). The Total Cost of Ownership brings all those cost positions together, thus providing a figure upon which relatively unambiguous decisions for initial capital investments (e.g. base technology) can be made. Bullinger et al. (1997) define TCO as “a figure that helps enterprises to understand the […] cost that is related to the usage of complex client-server systems. Thereby, the whole software lifecycle of [a system] from the installation to the discontinuance is considered”.

Total Cost of Ownership

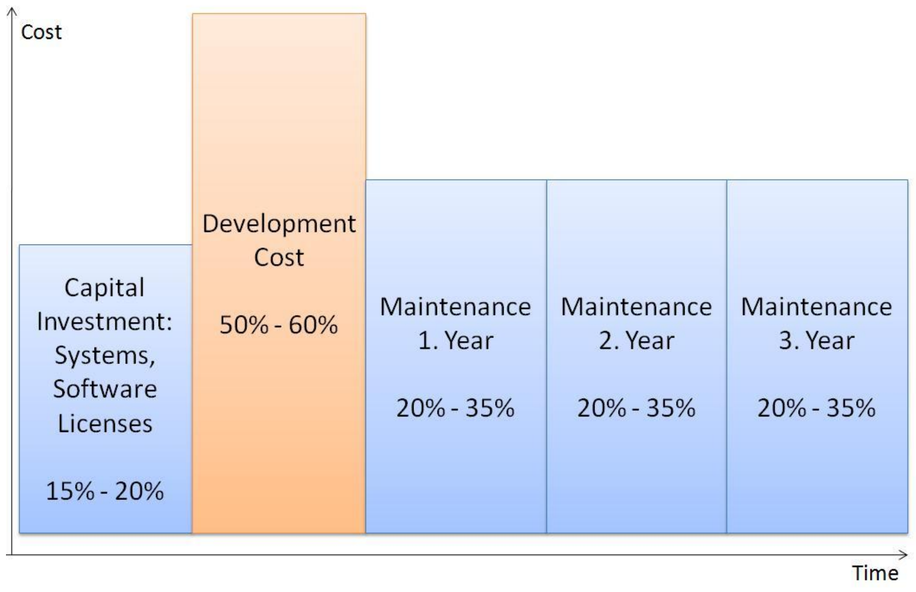


Figure 60: Typical cost distribution of big development projects (Göldi & Stuker, 2003).

As shown above, in case of a commercial realization of the EBF, the TCO for the EBF framework would be determined by the capital investment, the actual development cost of the system as well as expenditures for maintenance and extension development. Since the EBF would target a huge number of applicants in small and medium enterprises, this TCO could hypothetically be

EBF TCO

divided by the number of customers in order to determine the price for the framework that would at least cover for the Total Cost of Ownership.

It is debatable, whether it makes sense to also introduce the TCO terminology on the applicant or customer side, and not only on the side of the EBF-AF provider. If so, the client side TCO would also need to take into account the cost for the infrastructure that is necessary to host an EBF application. One of the proposals that was discussed earlier is to reduce the applicant's infrastructure expenditures to a minimum by marketing the EBF as SaaS. In this approach, the applicant's fully customized EBF application would run on a remote server hosted by the EBF provider, whereby the EBF applicant on the client side would merely require a standard web browser and an internet setup to connect to the EBF. Hence, the applicant's infrastructure expenditures would become marginal and as a result of that the TCO for the EBF provider would equal the number of EBF-AF applicants times the TCO of an individual applicant.

EBF SaaS

When looking at possible base technologies for the EBF-AF that could be used to implement the EBF SOA stack, the only two real options are .Net and J2EE. Technology-wise both .Net and J2EE may not seem to differ a lot. The most significant demarcation criterion between both, however, is that J2EE is platform independent but limited to the Java programming language, while .Net is language independent but de facto bound to the Windows platform (Vawter & Roman, 2001)

.Net vs. J2EE

Both Sun's J2EE and Microsoft's .NET provide runtime mechanisms that insulate software developers from particular dependencies. In addition to a web service-oriented XML layer of indirection between platforms, languages, and enterprise architectures, J2EE and .NET facilitate language-level intermediation via the Java Runtime Environment (JRE) and the Common Language Runtime (CLR) respectively (Vawter & Roman, 2001).

Language-level Intermediation

J2EE offers several features that accelerate time-to-market which are not found in .NET. For example, "beans" that are able to maintain their state enable developers to write less code and not worry about persistency, resulting in a higher degree of rapid application development. This not only makes applications leaner, but also database independent and thus easier to maintain (Vawter & Roman, 2001).

J2EE

Microsoft's.NET on the other hand offers a variety of time-to-market features that are not found in J2EE as well. Most notably, ASP.NET is client device independent and allows for portability of user interfaces across platforms. Microsoft also offers Queued Components which are superior to Message-driven Beans in J2EE. On top of that, Microsoft also provides business process management and e-Commerce capabilities, which are available in some J2EE implementations but not all.

.Net

Vawter & Roman (2001) conclude that "the ability to achieve rapid application development offered by both J2EE and .NET is definitely not equal. It is, however, comparable". They argue that

Not Equal, but Comparable

feature differences are minor and it is very difficult to make a compelling argument either way. Making platform decisions based upon those minor differences might be misleading as there are larger business issues at hand that easier determine the platform choice. The following Table 6 exposes .Net to J2EE and tries to summarize their characteristics by value.

Feature	J2EE	.Net
Type of technology	Standard	Product
Middleware vendors	30+	Microsoft
Interpreter	JRE	CLR
Dynamic Web Pages	JSP	ASP.Net
Middle-tier components	EJB	.Net Managed Components
Database access	JDBC, SQL/J	ADO.Net
SOAP, WSDL, UDDI	Yes	Yes
Implicit middleware (load-balancing, etc.)	Yes	Yes
Language Support	Java	VB, J#, C/C++, C#

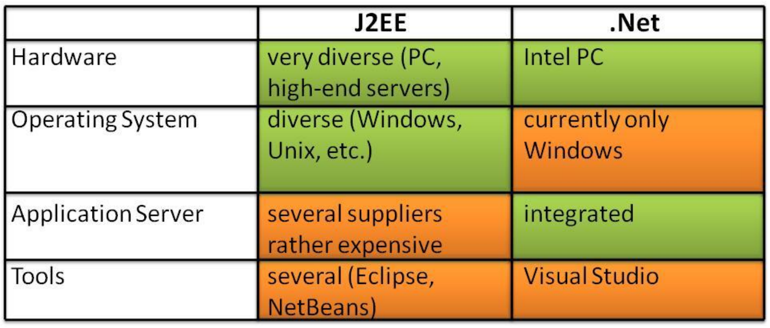
Table 6: Understanding .Net and J2EE by analogy.

TCO-based Decision Making

Since it seems like there is no significant technological pro or contra of each that could facilitate a decision towards one of the both discussed alternatives, it is time to recall the Total Cost of Ownership (TCO), introduced in the beginning of this section. In addition to considering the TCO, the decision will be influenced by the opportunity cost, which is the cost that evolves from lost market share or revenue opportunities by neglecting one alternative in favor of the other (Horngren et al., 2008).

Capital Investment

Figure 60 has demonstrated that in case of the development of an enterprise application (or framework), as the EBF-AF is one, the TCO consists of capital investment cost, development cost and maintenance cost for each year the application is running. According to Göldi & Stuker (2003), the capital investment cost for both .Net and J2EE takes the following aspects into account.



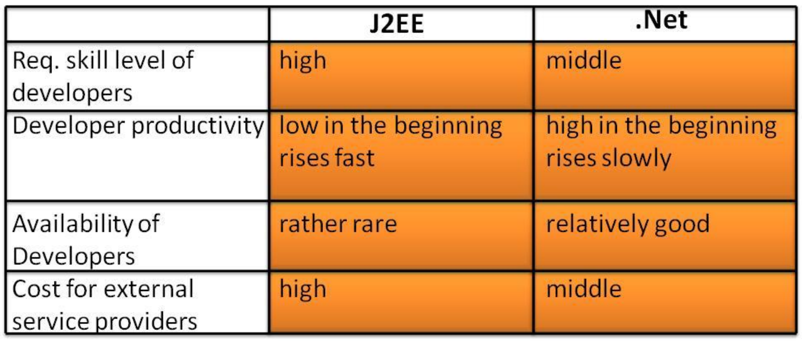
	J2EE	.Net
Hardware	🟩 very diverse (PC, high-end servers)	🟩 Intel PC
Operating System	🟩 diverse (Windows, Unix, etc.)	🟧 currently only Windows
Application Server	🟧 several suppliers rather expensive	🟩 integrated
Tools	🟧 several (Eclipse, NetBeans)	🟧 Visual Studio

Figure 61: Capital Investment of .Net vs. J2EE (Göldi & Stuker, 2003).

The above four criteria hardware, operating system, application server and tools are exposed by both .Net and J2EE in slightly different ways. However, it seems like there is no definite advantage of one technology over the other. A possible cost driver, however, could arise on the J2EE side through the remarkably more expensive application server licenses, if non-open source software, such as IBM or Oracle, is used (Göldi & Stuker, 2003). Those application servers, in some cases, might also require very demanding and special hardware. As opposed to that, the JBOSS application server is distributed under an open source license (hence for free) and runs on a normal desktop operating system, such as Microsoft Windows. However, using open source software might reveal a negative cost impact on the overall project in later stages of the application's lifecycle, if, for example, vendor support is required.

Development Cost

Regarding the second component of the TCO, Göldi & Stuker (2003) state that the development of J2EE applications by trend causes the higher cost. Acquainting oneself with .Net is easier and less expensive and especially smaller projects are faster achieved with .Net. For more complex and larger projects, however, the development cost for .Net applications typically quickly exceeds the one of a J2EE-based alternative. Göldi & Stuker (2003) explain this with the higher required skill level for J2EE developers and thus the potentially higher cost for labor as compared to .Net developers. While the developer productivity for J2EE in the beginning rather succumbs to the productivity of .Net developers, it rises very quickly and soon even exceeds the productivity of .Net developers. In addition to this, Göldi & Stuker (2003) state that by trend it has been easier to find .Net developers than J2EE developers. Also, the cost for external service providers in case of J2EE exceeds the cost for comparable .Net service providers.



	J2EE	.Net
Req. skill level of developers	high	middle
Developer productivity	low in the beginning rises fast	high in the beginning rises slowly
Availability of Developers	rather rare	relatively good
Cost for external service providers	high	middle

Figure 62: Development Cost of .Net vs. J2EE (Göldi & Stuker, 2003).

Maintenance Cost

The third and last component of the Total Cost of Ownership is the maintenance cost. Due to changes in the domain or errors found in the application, investments into an application might become necessary, even after the initial development has completed. Göldi & Stuker (2003) have the opinion that with respect to extendibility and reuse -two of the Framework Engineering Principles and Practices (FEPP) - J2EE dominates .Net. They argue that this is partially due to the longer time J2EE is in the market. Also, Microsoft systems in the past were known as not as stable as other UNIX systems or UNIX derivatives. However, Microsoft's recent and heavy investments into this market segment (e.g. the 2008 announced Red Dog platform) as well as Microsoft's significantly increased quality initiatives might even sooner than expected turn the tables.

	J2EE	.Net
System Stability	middle – high	low - middle
Ease of Maintenance	complex, not always optimal	often pragmatic architecture
Reusability of Code	high	traditionally low
Extensibility	high	traditionally low

Figure 63: Cost of .Net vs. J2EE (Göldi & Stuker, 2003).

TCO Summary

After Capital Investment, Development and Maintenance Cost were discussed as TCO components, the following points summarize the previous TCO-related elaborations on J2EE and .Net:

- **J2EE**
  - For larger systems with a long scheduled runtime
  - TCO benefits in the long run, due to lower maintenance cost and a higher potential reuse
- **.Net**
  - For small systems with a short amortization period
  - Cost for maintenance in the long run higher

Opportunity Cost

In addition to the Total Cost of Ownership, another significant denominator is given by the opportunity cost. Opportunity cost was earlier defined as the cost that evolves from lost market share or revenue opportunities by neglecting the one alternative in favor of the other. What that means with respect to the EBF-AF is that by deciding for one platform over the other, it might happen that certain potential framework applicants might not be able to use the framework anymore. In case of .Net being installed on a local machine on the applicant's side, the framework applicant must use Windows. This means that organizations running operating systems other than Windows (e.g. Unix, Linux) would not be able to have their own local installation of the EBF-AF. Instead, they would rather have to go for the SaaS (Software-as-a-Service) way and license an EBF-AF instance that is hosted remotely. This, however, might fail due to concerns revolving around privacy and security of organization and customer data. In such case, organizations might hesitate to host their confidential or classified data on hardware that is run and accessible by third party service providers.

Also, choosing .Net as a platform for the EBF-AF might conflict with the requirement of targeting a minimal setup and operational cost on the client side. As it was elaborated upon earlier, the primary target group of the EBF-AF are SME and those business might prefer (in accordance with above explained security concerns) to host their own instance of the EBF-AF, but on an open source operating system, such as Linux, that does not require any monetary investment.

Implementation Platform

Concluding from previous explanations, the decision for the implementation platform of the EBF-AF is made in favor of J2EE, due to the following attributes J2EE exposes:

- Presumably lower TCO in the long run
- Eventually, better maintenance and reusability
- No platform lock-in
- No noticeable opportunity cost
- No necessary expenditures for operating system (Unix/Linux) and platform licensing (JBoss)

On the other hand, choosing J2EE trades the following advantages of .Net:

- “All-from-one-vendor” solution
- Highly sophisticated support (Quick Assist) and documentation (MSDN)
- Language independency

EBF-AF SOA Stack

With J2EE being chosen as the platform to implement the EBF-AF upon, it seems appropriate to give a prospect of how in particular the EBF-AF (J2EE) architecture might look like. Figure 64 demonstrates how the architecture shown in Figure 59 is being mapped to elements of the J2EE standard. The EBF Appl. Server Dispatcher, for instance, is mapped to the J2EE Service Registry. The J2EE Service Registry, similar to the concept of a dispatcher provides information about which services are available. It will later be detailed how the service invocation layer in many implementations of the J2EE standard also provides fail-over and load-balancing functionality. This means that the EBF-AF would not need to implement an own load-balancing mechanism and could instead simply leverage the J2EE application server’s load-balancing and fail-over functionality. Furthermore, applying J2EE to the EBF Product Model, the Process Model as well as other EBF services would map those to Enterprise Java Beans (EJB) that run in a so called J2EE container. The EBF Integration Foundation, which was introduced earlier as the instance of integration and communication across EBF components or services in J2EE, would be reflected by a J2EE Enterprise Service Bus (ESB). A J2EE ESB could also be used to seamlessly integrate external services with internal EBF services.

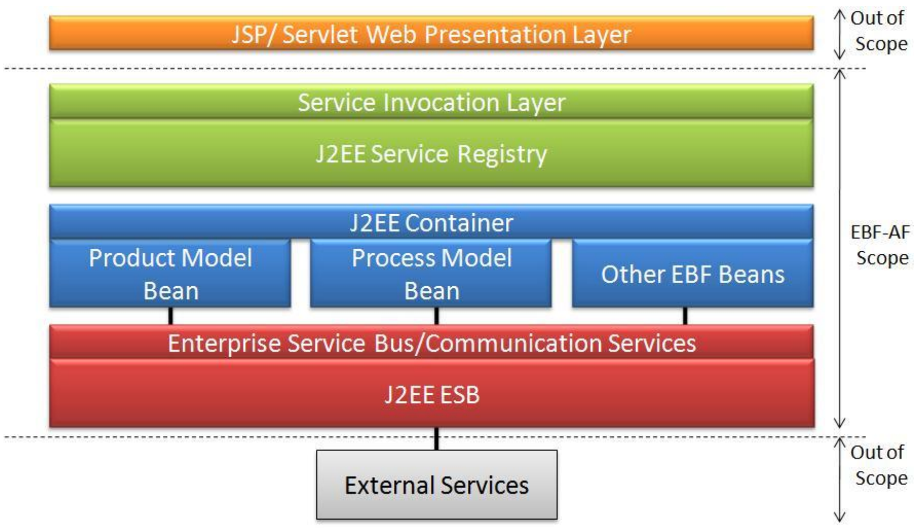


Figure 64: The EBF-AF SOA Stack running on a J2EE architecture.

EBF JBoss Stack

One of the key reasons for J2EE having been chosen over .Net was that there is no investment cost for an open source J2EE implementation, such as JBoss. While the name JBoss actually stands for an application server that implements the J2EE standard, over the years a variety of other products were added to its project portfolio. Figure 65 shows how the Web Presentation Layer of the EBF could be realized purely by using JBoss technologies, such as JBoss Web, JBoss Ajax4jfs or JBoss Rich Faces. It furthermore shows that the service invocation layer, which serves as web service registry as well as load-balancing and fail-over managing instance, is being implemented purely by the JBoss Application Server. Also running within a container of the application server, the EBF Product Model, the EBF Process Model as well as other EBF components or services would be implemented through Enterprise Java Beans. The EBF Integration Foundation (EBF-IF), which in Figure 64 mapped to a standard J2EE Enterprise Service Bus (ESB), in terms of JBoss technology would be represented by a JBoss ESB with enhanced communication services being provided by JBoss Messaging. In order to give a complete picture, the SOA stack of the EBF-AF as shown in Figure 65 also exposes an item that was not yet mentioned so far: A J2EE Workflow Engine implemented by the JBoss jBPM module.

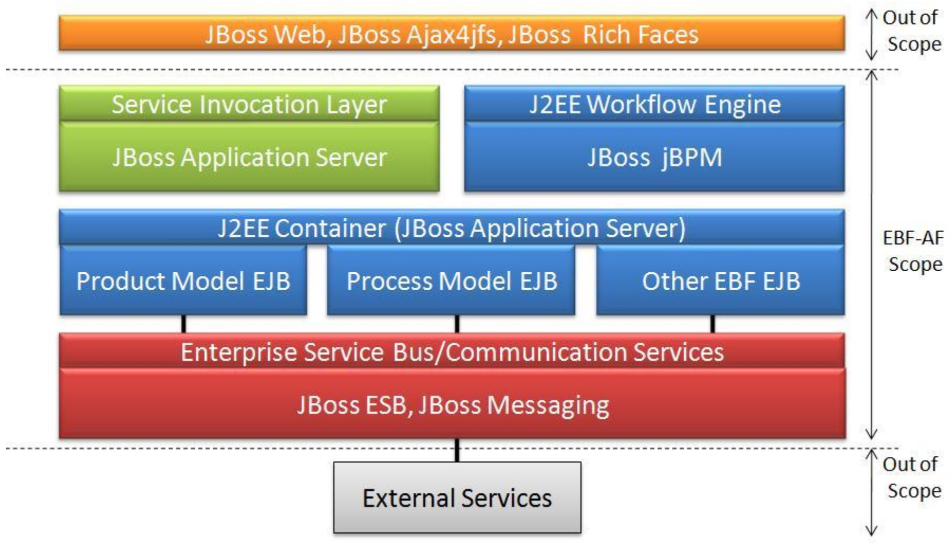


Figure 65: EBF-AF SOA Stack running on JBoss technology.

Goal Assessment

With the EBF-AF SOA stack being complete, the still outstanding goals shall be assessed.

- **Low TCO:** The key for being able to provide a low TCO in the EBF-AF is its web-based SaaS approach. All EBF-AF applicants would need for their daily operations is a standard desktop computer that has access to the internet via a web browser. No additional investments into infrastructure and no expenditures for maintenance would be required. The capital investment part as well as the development part of the EBF-AF TCO is minimized by utilizing open standards and technologies.
- **Object Orientation:** With the base technology being J2EE, the programming language used for the implementation of the EBF-AF is Java. Java is highly object-oriented and allows for componentized and better to maintain code. Hence, not only the design of the EBF itself is componentized where possible, but also the implementation of the components would strictly follow an object-oriented paradigm. All in all, the object-oriented approach that was chosen in the EBF-AF should help to significantly lower the development and more importantly the maintenance cost.
- **Platform Independence:** When looking at the opportunity cost of both technology alternatives, platform independence played a major role. By having chosen J2EE over .Net, the EBF-AF is totally platform independent and can be run on any operating system. Hence, if need be due to, for example, security or privacy reasons, all businesses are able to host the EBF-AF by themselves, not matter what operating system or middleware they use.

Quick Review – Base Technology Analysis for the EBF-AF

A low Total Cost of Ownership (TCO) is not only a requirement to the EBF, but even more importantly one of the key success drivers towards achieving the mission of providing a low-cost e-commerce platform for SME. The TCO of the EBF-AF would be determined by the capital investment, the development cost and the maintenance cost for the period the EBF-AF would be in operation. While the capital investment reflects quite a significant part of the overall TCO, capital investment-related decisions should more importantly take into account the cost for development and investment, which in most cases would together stand the larger portion of the TCO. Having compared .Net and J2EE with each other, no significant differences, neither technology-wise nor TCO-wise, could be identified. Both technologies facilitate enterprise application development while offering all in all similar functionality. However, since the targeted EBF-AF applicants cannot afford a comparably high investment into IT and infrastructure, an open source-based solution would have been preferable. Adding the criteria of opportunity cost to the technology evaluation, which assumed that potential EBF-AF applicants that are running their operations upon another operating system than Microsoft Windows could potentially hesitate to go for the EBF-AF, resulted into a decision for J2EE. The initial SOA-based EBF architecture was mapped to a J2EE stack, based on solutions provided by the open source J2EE vendor JBoss.

5.6 Implementing the EBF Product Model

Chapter three covered the theoretical development of the EBF-PPM, whereby the design of both the product and the process model took place in separate sub sections. It was shown how by using an XML dialect that defines the syntax of the product description language (EBF-PDL), a very flexible product model could be proposed. This was possible due to the EBF-PDL being generally independent of semantic aspects. It was furthermore discussed that the overall product model was merely an aggregation of its individual product descriptions. Product descriptions on the other hand fully described a product by segmenting a product's properties into attributes and their values (for an example see Figure 66). Thereby, both the naming of the attributes as well as the values were variables in the product description language.

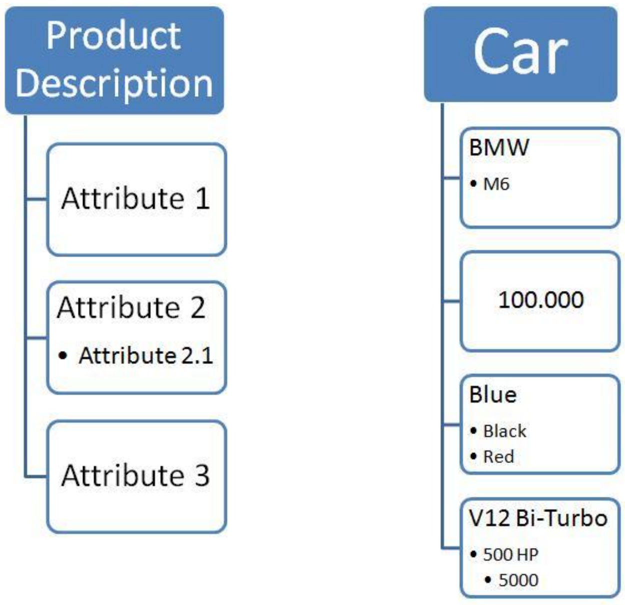


Figure 66: Product descriptions consist of hierarchically ordered attributes (left in general and right applied to the car example).

Recalling the Car Example

By modeling the example of the car and its properties, such as make, cost and color, a first attempt to validate the EBF-PDL in practice could be made. The car example EBF-PDL description as of section 3.3 The Product Model contained the following:

```
<make>
      BMW
      <type>
            M6
      </type>
</make>
<cost>
      100.000
</cost>
<color>
      blue
      <rims_color>
            Black
      </rims_color>
      <brake_color>
      red
      </brake_color>
</color>
<engine>
      V12 Bi-Turbo
      <power>
            500 HP
            <at_rounds_per_minute>
                  5000
            </at_rounds_per_minute>
      </power>
</engine>
```

J2EE and Java

After the beginning of this chapter tried to find an architecture that would fit the special needs of the EBF Application Framework, the previous section discussed two alternatives (J2EE and .NET) that could be used as technological foundation for this architecture. Even though both technologies are in principal very similar, by focusing upon the Total Cost of Ownership (TCO) as well as opportunity cost, a decision in favor of J2EE could be made. Leveraging J2EE's capabilities, however, is bound to the Java programming language as the language of implementation for the EBF Application Framework.

In this second call to the EBF Product Model, a prototype is being discussed that demonstrates how Java can be used to implement the EBF Product Model.

From EBF-PDL to Java

Using the EBF-PDL, framework applicants on a relatively easy-to-understand level can customize their own product descriptions and add them to the product model. However, since the product model component is implemented in Java and runs as a Java Bean within a J2EE container, it is at this point in time not yet clear how an EBF-PDL product description can be accessed from within the product model itself. Please refer to Figure 67 for a visual representation of the problem.

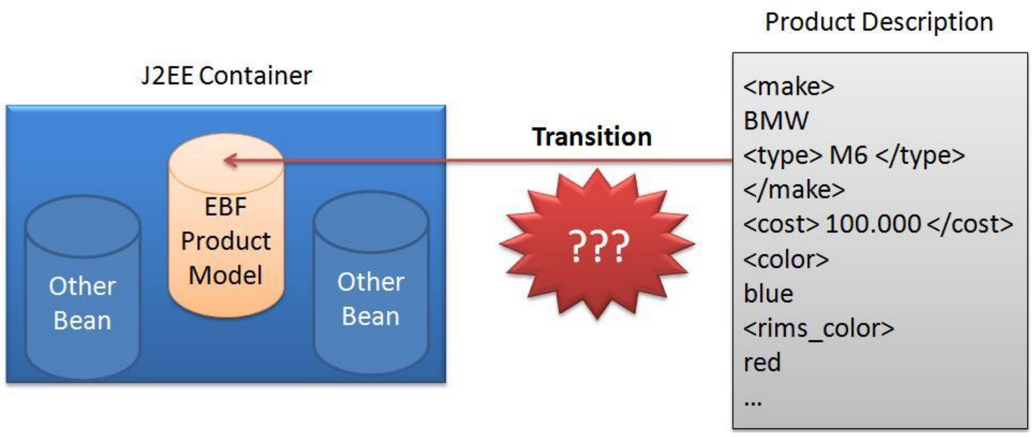


Figure 67: The transition from a Product Description to a Java class in a J2EE Bean is yet not defined.

Javac

The proposal that is presented in the remainder of this section tries to close the above shown transitioning gap by introducing a compiler that generates a Java class (.java file) from an EBF-PDL description, compiles this class with the standard Java compiler (javac) into a Java bytecode representation (.class) and hooks this bytecode file into the EBF Product Model component. Thereby, the above process must be executable at runtime and thus a new product description addable, even when the EBF Framework is running.

EBF-PDL Compiler

THE EBF-PDL Compiler translates an EBF product description into a Java class. Thereby, the compiler first of all checks, whether the syntax of the provided EBF-PDL description is correct and valid. If so, it generates a new Java file which contains a Java class. This class serves as the Java representation of the production description, but contains exactly the same information as the product description. Figure 68 shows how the compiler performs a mapping of the product description information from the format stored in the product description file (source format) to the format of a Java class (target format).

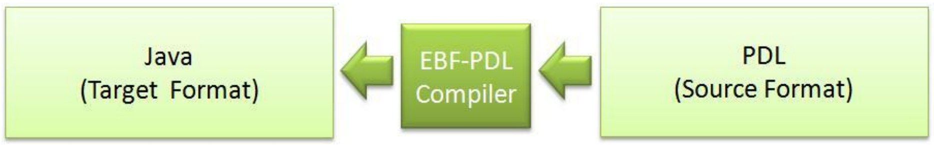


Figure 68: The EBF-PDL Compiler as a black box that maps the source format (PDL) into the target format (Java).

As a result of the elaborations on the EBF Product Model and the EBF-PDL in chapter three, the source format is already well known. However, the target format was not yet discussed and will be the subject of the following paragraphs.

Product Description Tree

By taking a close look at the PDL car example, due to the hierarchical organization of attributes in the XML description, a graphical representation of the hierarchy could potentially result into a tree-like shape (see Figure 69). In the end, a tree is nothing else than a hierarchy (Massengill & Jackson, 2006). The following graphic demonstrates the EBF-PDL example of the car mapped onto a tree.

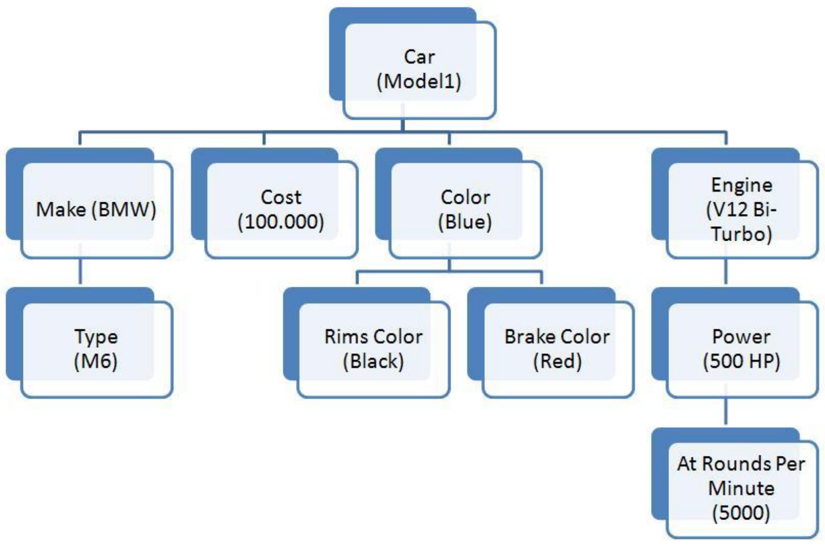


Figure 69: Product Description tree of the PDL car example.

Trees in computer science are referred to as hierarchical data structures that consist of a root node, child nodes as well as leaf nodes (Ranta-Eskola, 2001). In above example, the root node would be “Car”, an example for a child node would be “Color” and a leaf node would be given by “At Rounds Per Minute”. However, since the categorization with respect to the EBF into leaf nodes does not seem to be of a particular benefit, this terminology will not be subject of the ongoing elaborations[17].

Root and Childe Nodes

In the EBF product description above in Figure 69, the root node stands for the entry point of being able to access further child nodes and carries merely a product model-wide unique name that serves as identifier or reference for the particular product description. From the root node, all other child nodes within the product description are accessible either directly (as direct child nodes) or indirectly (via another child node). Being to a high extent similar in organization and function, the data structures behind the root node and the child node are basically the same. Thus, creating a hierarchy is merely a matter of recursively adding additional nodes as child nodes to other child nodes. Hence, a child node (e.g. in Figure 69 "Color") somehow needs to maintain its own list of child nodes (e.g. "Rims Color", "Brake Color") and also provide its parent node (e.g. "Car") with a way to access its child nodes as well as the actual value they store (e.g. "Blue").

Product Description

The EBF Product Model prototype[18] that was created to serve the above purpose distinguishes into a "ProductDescription" node (root node) and a "ProductNode" (child node). By using the object-oriented method of Derivation, the ProductNode could be simplified to merely being a specialization of the ProductDescriptionNode. The following UML class diagram (Figure 70) provides an overview of the Java class behind the ProductDescription node and demonstrates its interface functions and internal data structures.

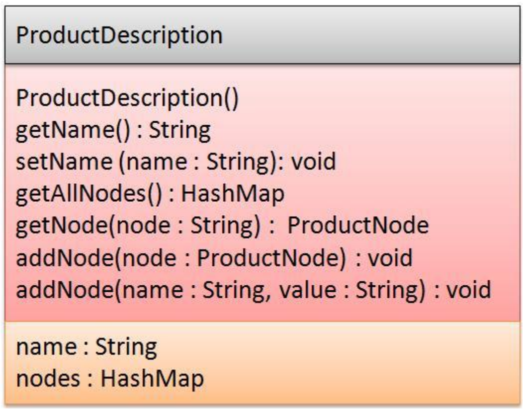

ProductDescription
ProductDescription()
getName() : String
setName (name : String): void
getAllNodes() : HashMap
getNode(node : String) : ProductNode
addNode(node : ProductNode) : void
addNode(name : String, value : String) : void
name : String
nodes : HashMap

Figure 70: The root node class "ProductDescription" of the EBF Product Model in a UML class diagram.

Interface Functions:

- **ProductDescription:** Constructor necessary to initialize the object.
- **getName:** Returns the name of the ProductDescription root node (e.g. in the car example "Car").
- **setName:** Sets the name of the ProductDescription node (e.g. "Car").

[17] Leaf Nodes are only of a particular interest, if, for example, the depth of a tree is to be determined.
[18] The source code of the prototype can be found in the Appendix of this work.

- **getAllNodes:** Returns a HashMap of all the child nodes.
- **getNode:** Returns a single child node, which is identified by the name of the child node.
- **addNode:** Adds a new ProductNode to the HashMap of child nodes. The parameter list can either contain an already initialized instance of a ProductNode or the parameters (name, value) that are necessary to initialize a new ProductNode.

Internal Data Structures:

- **name:** Name of the node.
- **nodes:** List of child nodes.

Product Node

The UML class diagram for the ProductNode class looks accordingly. Please note that the ProductNode class is a derivative of the ProductDescription class and therefore, even though not explicitly notated in the class diagram, derives all of the interface member functions from its base class ProductDescription.

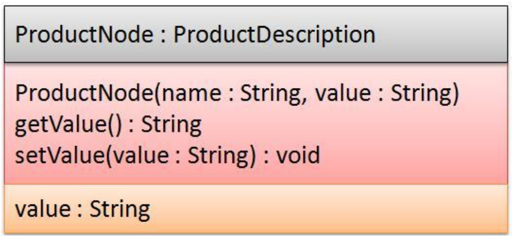

ProductNode : ProductDescription
ProductNode(name : String, value : String)
getValue() : String
setValue(value : String) : void
value : String

Figure 71: The child node class "ProductNode" in a UML class diagram.

Interface Functions

- **ProductNode:** Constructor necessary to initialize the object.
- **getValue:** Returns the value of the ProductNode (e.g. in the car example "BMW" for the "Make").
- **setValue:** Sets the value of the ProductNode (e.g. "BMW").

Internal Data Structures

- **value:** Value of the node.

Compilation Process Model

Getting back to the red bang in Figure 67, how do those two classes relate to an EBF-PDL file and how can they be used to load a .pdl file at runtime into the EBF Product Model? To allow for product descriptions being compiled dynamically and added at runtime to the EBF Product Model, the Java specialty Reflections can be leveraged. Reflections represent a way of loading java bytecode (.class files) at runtime and linking it to the execution environment (Gosling et al., 2005). Using Java Reflections, instances of classes that are not known at compile time can be created simply by announcing the path and name of the .class file to the Java Runtime Environment (JRE) from where the calling code is executed. Therefore, the EBF-PDL compiler could create a java class from the .pdl file, compile the class with the Java compiler (javac) and add the resulting bytecode file (.class) to the classpath of the EBF Product Model Bean. A more detailed perspective on above described is given by the following procedure:

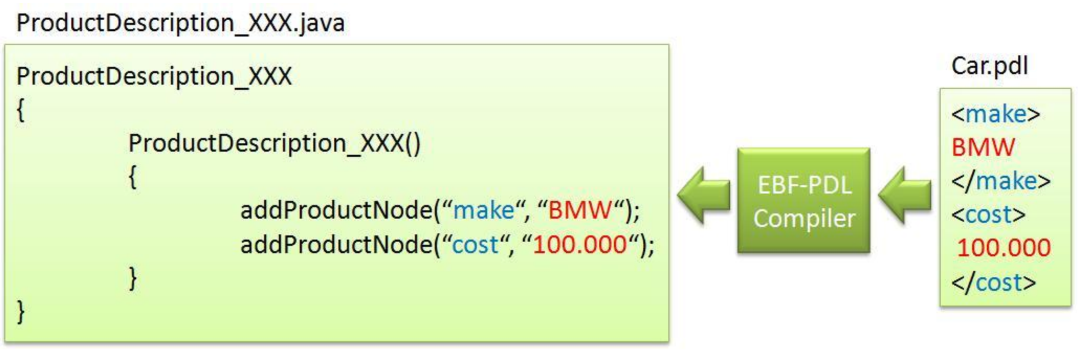
EBF-PDL Compiler Process

(It is strictly recommended to have a brief look at the source code of the EBF Product Model prototype that can be found in the appendix of this work)

EBF-PDL Compiler Process

1) Verify the integrity and correctness of the .pdl file's syntax.
2) EBF-PDL compiler creates a new .java file that contains the targeted product description:
   - The name of the new class must be unique and could be reflected by a combination of the string "ProductDescription" and the root node of the product description tree (e.g. in the car example: "ProductDescription_Car").
   - The new class ProductDescription_XXX must extend the existing EBF-PPM class ProductDescription.
3) Constructor ("ProductDescription_XXX()") is added to the class ProductDescription_XXX:
   - Within the constructor, the product nodes are added to the product description. Therefore, the EBF-PDL compiler for each attribute in the PDL description needs to give a call to addProductNode (see Figure 72) in the constructor that adds the child ProductNodes. The name and the value of the ProductNodes must be extracted from the PDL description.

The above explained procedure for the transformation of a PDL description (stored in a .pdl file) into a Java class (stored in a .java file) is visually summarized in Figure 72:


ProductDescription_XXX.java

```
ProductDescription_XXX
{
        ProductDescription_XXX()
        {
                addProductNode("make", "BMW");
                addProductNode("cost", "100.000");
        }
}
```

Car.pdl

```
<make>
BMW
</make>
<cost>
 100.000
</cost>
```

Figure 72: The EBF-PDL Compiler as a black box between the PDL description and the Java class.

At this point, it should be remarked that in case of the attributes in the PDL file having more than one hierarchy level, the procedure of transforming the product description into a Java class is not that simple anymore. Instead, the EBF-PDL compiler would need to start initializing distinct ProductNode instances beginning from the most bottom level attributes in the product description and recursively adding those ProductNodes to the next higher level ProductNodes. *Hierarchy Levels* This procedure continues until finally the addProductNode function can be called for the top most ProductNode (as depicted above).

4) Compile the ProductDescription_XXX Java class:

   - Invoke the java compiler (javac) and specify the compilation parameters, such as name and classpath of the ProductDescription_XXX.java.

5) Deploy the bytecode files:

   - Copy the .class files that were generated by the Java compiler to the classpath of the EBF Product Model Bean.

6) Application from within the EBF Product Model:

   - The class that contains the product description is now within the scope of the EBF Product Model Bean and can be dynamically invoked using another Reflection. To instantiate the product description, all the reflection needs to know is the class name (e.g. ProductDescription_XXX) and, if applicable, an attribute that is to be processed.

The EBF-PDL Compiler process, based on above explanations, is summarized in the following UML activity diagram (Figure 73). Please note that a more in-depth view on the “Construct Product Description” item was already given in Figure 72 and as part of point three in the EBF-PDL Compiler Process.

EBF-PDL Compiler Activity Diagram

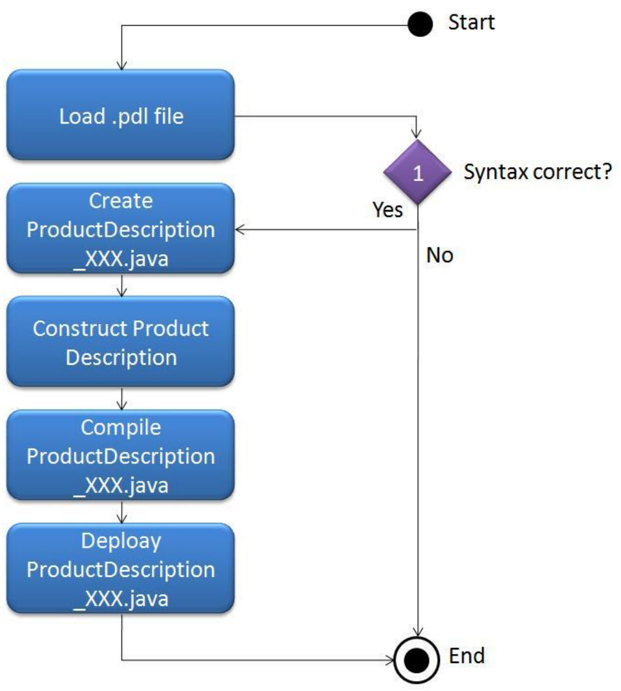

Start
Load .pdl file
1
Syntax correct?
Yes
No
Create ProductDescription _XXX.java
Construct Product Description
Compile ProductDescription _XXX.java
Deploay ProductDescription _XXX.java
End

Figure 73: A UML activity diagram for the EBF-PDL Compiler process.

One of the requirements to the EBF Application Framework was the “Full feature support of the EBF Product and Process Model” (see section 5.1). While the EBF Process Model is going to be the subject of the next section, one of the key features of the EBF Product Model was not covered so far.

Implementing Attribute Handlers

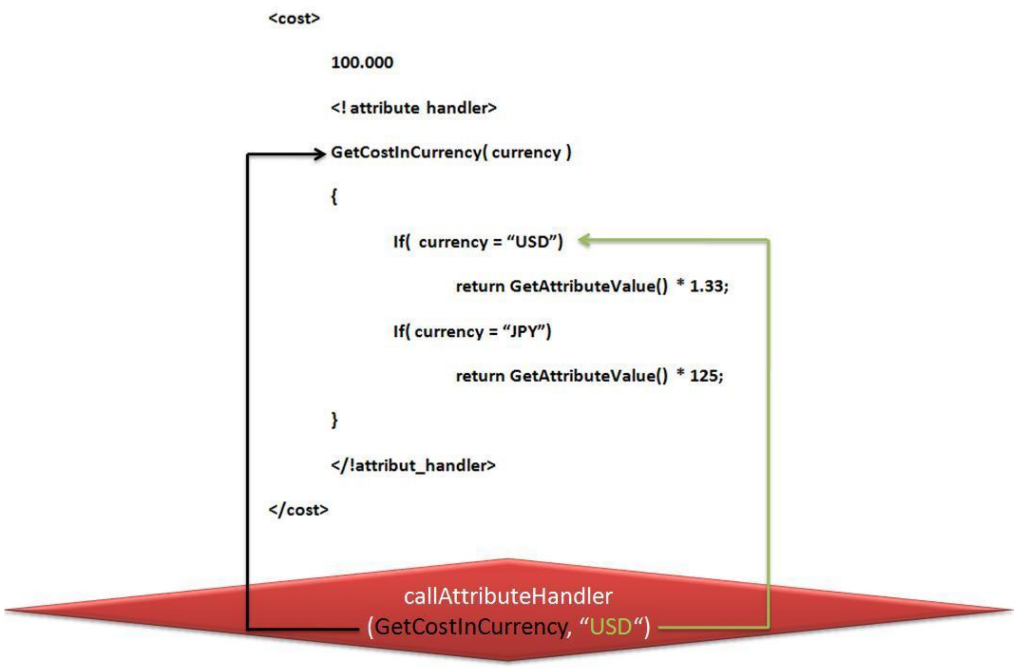
To allow attributes within a product description to be dynamically loaded, acquired from other sources or modified at runtime, the second part of section 3.3 The Product Model proposed an enhancement to the EBF Product Description Language that was capable of expressing each and every dynamic behavior of single attributes. This dynamic behavior was achieved through the introduction of a new attribute handler tag (“<!attribute_handler>”). By adding this tag to an attribute definition, certain operations and modification logics on the attribute could be achieved. Thereby, the language in which the attribute handler is actually to be implemented in, due to the decision made in section 5.5 Base Technology Analysis for the EBF-AF, is restricted to Java. On the other hand, for the implementation of the attribute handler, no new scripting or programming logic needs to be added to the product model. Instead, the whole Java language feature spectrum can be used, since behind the scenes the code for the attribute handler will simply be inserted into the respective ProductNode and compiled with the Java compiler (javac). Hence, syntax validation and error check completely remains within javac. Referring back to chapter three, the attribute handler for the “cost” attribute in the car example looked like the following:

```
<cost>
      100.000
      <!attribute handler>
      GetCostInCurrency( currency )
      {
            If( currency = "USD" )
                  return GetAttributeValue() * 1.33;

            If( currency = "JPY")
                  return GetAttributeValue() * 125;
      }
      </!attribut_handler>
</cost>
```

Extending the existing EBF-PDL Compiler Process model to be able to consider attribute handlers as well once again requires the utilization of the so powerful Java language feature Reflections. Before the compiler is able to just insert the java code for the attribute handler in the respective ProductNode, a certain set of infrastructural changes needs to be done to the product model. The ProductNode class is extended by a new method "callAttributeHandler". CallAttributeHandler takes two parameters: one for the name of the attribute handler that is to be called and another one for a list of parameters that are to be marshaled to the attribute handler (see Figure 74).

callAttributeHandler



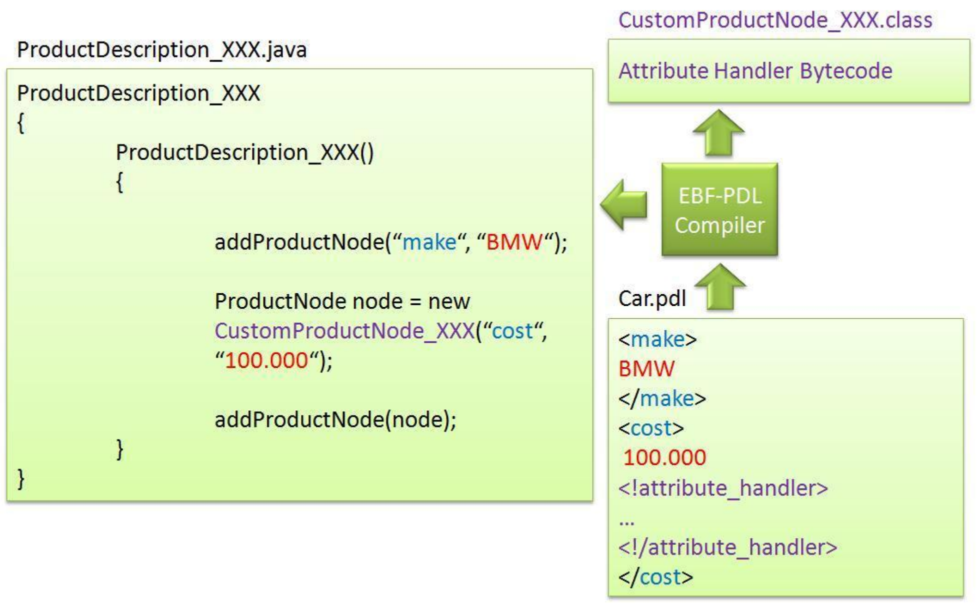
```
<cost>
      100.000
      <! attribute handler>
      GetCostInCurrency( currency )
      {
            If( currency = "USD")
                  return GetAttributeValue() * 1.33;
            If( currency = "JPY")
                  return GetAttributeValue() * 125;
      }
      </!attribut_handler>
</cost>
```

Figure 74: Binding of the function callAttributeHandler to the attribute handler code in the PDL file.

The other necessary change that needs to be conducted is that for those attributes with an attribute handler, the compiler cannot simply call addProductNode as shown in Figure 72 anymore. Instead, the EBF-PDL compiler first of all needs to derive a new CustomProductNode_XXX from ProductNode and place the attribute handlers in the body of the CustomProductNode_XXX class.

Custom Product Node

Furthermore, where formerly addProductNode was called, a new ("reflected") instance of the CurstomProductNode_XXX needs to be created. This custom product node instance, however, can in the second step simply be passed as a parameter to addProductNode. The javac compiler again takes care of correctly binding the EBF-PPM classes (ProductDescription, ProductNode) together with the custom classes (ProductDescription_XXX, CustomProductNode_XXX). The result of those changes is visualized in the following Figure 75:


CustomProductNode_XXX.class
Attribute Handler Bytecode
EBF-PDL Compiler
ProductDescription_XXX.java
Car.pdl

```
ProductDescription_XXX
{
        ProductDescription_XXX()
        {
                addProductNode("make", "BMW");

                ProductNode node = new
                CustomProductNode_XXX("cost",
                "100.000");

                addProductNode(node);
        }
}
```

```
<make>
BMW
</make>
<cost>
 100.000
<!attribute_handler>
...
<!/attribute_handler>
</cost>
```

Figure 75: The EBF-PDL Compiler in addition to the previous version now also creates a CustomProductNode_XXX.class, which is bound into the EBF-PPM via a Java Reflection.

Within the EBF-PPM, a reflected CustomProductNode instance of the product description can be queried for the attribute handlers it provides. Alternatively, if the desired attribute handler of a CustomProductNode can be announced through the EBF Product Model to the ProductDescription_XXX, it can be called straight ahead.

Implementing Default Attribute Handlers

The last point that is going to be covered throughout these elaborations on the implementation of the EBF Product Model refers to the logic of default attribute handlers. The rationale behind default attribute handlers was that by querying an attribute for its value, it might happen that this value first of all might need to be acquired from an external source or certain operations might need to be applied to it. As those operations should be performed under the cover and invisible from the framework applicant, the product model, when an attribute is queried for its value:

a) needs to check, whether a default attribute handler was set up for this attribute and

b) invoke the default attribute handler

before returning the actual attribute value. In order to achieve this behavior, the attribute handler infrastructure that was previously introduced can be leveraged. The only change that needs to be done to the EBF Product Model again concerns the ProductNode class. A registerDefaultAttributeHandler function is added that stores the name of a normal attribute handler in an internal variable assuming that the function for the attribute handler really exists within the class. Since, after all, default attribute handlers are also just normal attribute handlers, the EBF-PDL compiler needs to derive a CustomProductNode_XXX class from ProductNode and place the default attribute handler code within the body of the class. This needs to be done for each attribute that contains a default attribute handler. Secondly, in the constructor of the CustomProductNode_XXX, the EBF-PDL compiler needs to place a call to registerDefaultAttributeHandler and provide the name of the respective attribute handler as a parameter to this function. The related changes do not affect the ProductDescription_XXX at all, but instead entirely concentrate upon the CustomProductNode_XXX class. The class ProductNode is furthermore enhanced by a getValueEX function, which is the default attribute handler-enabled version of the getValue function. In case of a default attribute handler existing for an attribute node, getValueEx executes this attribute handler code and then returns the value. If no default attribute handler was registered, however, getValueEx simply falls back to getValue, which returns the plain and static attribute value. In order to facilitate the understanding of the previous explanations, it is also recommended here to have a look at the prototype source code in the appendix.

Enhancements to the Prototype

To give a complete picture of the final ProductNode class, an updated version of the UML class diagram is provided in the following:

Enhanced ProductNode

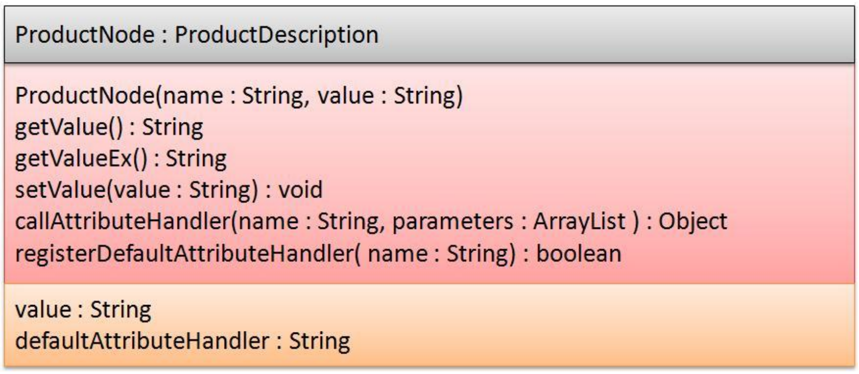

ProductNode : ProductDescription
ProductNode(name : String, value : String)
getValue() : String
getValueEx() : String
setValue(value : String) : void
callAttributeHandler(name : String, parameters : ArrayList) : Object
registerDefaultAttributeHandler(name : String) : boolean
value : String
defaultAttributeHandler : String

Figure 76: Final UML Class Diagram of the EBF-PPM ProductNode class.

Interface Functions

- **ProductNode:** Constructor necessary to initialize the object.
- **getValue:** Returns the value of the ProductNode (e.g. in the car example “BMW” for the “Make”).
- **getValueEx:** Before returning the value of the ProductNode (e.g. in the car example “BMW” for the “Make”), getValueEx checks, whether a default attribute handler was registered for the attribute and, if so, invokes it.

- **setValue:** Sets the value of the ProductNode (e.g. "BMW").
- **callAttributeHandler:** Invokes an attribute handler on an attribute, if the attribute handler specified in the name exists. Furthermore, marshals the parameters provided to the actual attribute handler.
- **registerDefaultAttributeHandler:** Declares one particular attribute handler of the attribute to be the default attribute handler that is invoked, when an access attempt to the attribute value is made via getValueEx.

Internal Data Structures

- **value:** Value of the node.
- **defaultAttributeHandler:** registered name of the default attribute handler function.

Independently of those modifications being made to the EBF-PDL compiler, the EBF-PDL Compiler Process model as shown in Figure 73 still remains the same. Merely the internals of the "Construct Product Description" process item change, whereby the modifications can be reviewed in Figure 75.

Quick Review – Implementing the EBF Product Model

An approach for making EBF-PDL descriptions accessible from within the EBF Product Model aimed at introducing an EBF-PDL/Java compiler as an interim step. The so called EBF-PDL compiler would translate product descriptions into Java classes and afterwards invoke the javac compiler on the generated classes. The compiler would for every product description create a tree-based data structure that can be linked to the EBF Product Model dynamically at runtime. Thereby, one of the key aspects was that the EBF-PDL compiler would also know how to handle the attribute handler logic that stands for the EBF-PPM's enormous flexibility. Of course, the compilation process would be performed in background and invisible from the EBF-AF applicant. In a more advanced and user-friendly scenario, the editing of the EBF-PDL file could further be abstracted via a product description editor which would store product descriptions invisible from the user in XML-based PDL files.

5.7 Implementing the EBF Process Model

In addition to the EBF Product Model, chapter three introduced a design for a process model that operates on top of the product model. Thereby, the goal of the process modeling was to capture all steps that are involved in an e-commerce transaction and expose them in a consistent, but easy to extend manner. The EBF Process Model detailed the participating e-commerce processes into h-processes and v-processes. While the role of the h-processes was to capture and structure the process flow of an electronic commerce transaction as well as ensuring that the v-process had access to all data resources they required for execution, the v-processes themselves implemented the actual logic to perform a request. After execution, the v-process would report the result back to the h-process, which for compatibility reasons would then bring the result into the interfacing format that was defined in the formal process description (refer to section 3.4).

Sampling the Search

Similar to the previous elaborations on the technical realization of the EBF Product Model, it will be the subject of this section to propose a way of implementing the EBF Process Model by building on top of a J2EE stack. Since the methodology that was used for describing the transaction processes was the same for every EBF process, it is considered to be sufficient to explanatorily highlight the implementation of one process only. Implementing other transaction processes would follow exactly the same pattern and use exactly the same technology. Therefore, the remainder of this section focuses upon the technical realization of the search process – a part of the Information Bus participating in the information phase (review section 3.4).

Recalling the s-process

In chapter three, the search process (notated: s-process) served the purpose of clarifying which products on an electronic commerce platform an individual would actually take into consideration for a possible demand fulfillment. Its interfaces were defined by the search function σ and the return vector $\vec{p}$ that contained all the alternatives that matched the search. The formal description of the s-process in chapter three was given by the following:

$$s = f(\sigma) = \{\,\vec{p}\,\}$$

Exposing v-Processes

Furthermore, the s-process was implemented by the three v-processes s.se (search engine), s.br (browser) and s.sm (similarity matcher). Since it is the responsibility of an h-process to expose the functionality of its v-processes in a certain way to the outside world, regarding the s-process this would mean that the functionality of the s.se, s.br and s.sm processes would somehow need to be made accessible to the user. This could happen via a web interface to the s-process in which the user explicitly chooses the search v-process he would like to invoke. Also, via the web interface, the user could provide the parameters for the search processes. These parameters would be passed to the s-process in form of the search function σ. In the case of s.se, for instance, σ would contain the parameters of the search - typically a string of terms the product model is to be queried for. For s.br, σ would rather be represented by the implicit browsing behavior of the user and thus would have a more abstract meaning. For s.sm, σ would describe for which product similar products are to be found and how the similarity[19] is determined.

[19] For an introduction to the topic of semantic similarity, refer to (Neumann, 2008).

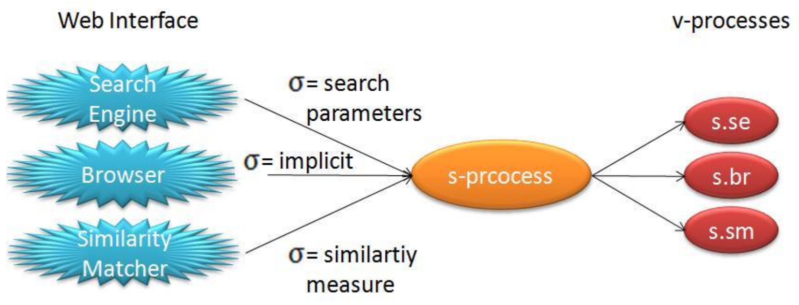


Figure 77: A web interface exposes the v-processes s.se, s.br and s.sm via the s-process. The v-process parameters are marshaled as **σ**.

Irrespective of which v-process is going to be invoked by the user via the s-process, in any case s would need to marshal σ to the involved v-process and ensure that the result of the v-process execution is reported back to the calling environment as the alternatives vector $\vec{p}$ (Figure 77).

Implementing the s-Process

By leveraging J2EE's SOA possibilities, the h-processes as well as the v-processes would be reflected by loosely coupled services running independently from each other in a managed container as Enterprise Java Beans (EJB). Those beans would announce their existence to the registry repository of the J2EE container, from where they would be accessible via their defined interface to all other services.

Adding v-Processes

In the concrete case of the EBF's s-process (itself an EJB), it could query the registry repository for which v-processes are available and which parameters they require. Having obtained this kind of information (more precisely the web service description), the s-process could present the user a list of available search functionality and let him choose whatever kind of search he wants to do. The big benefit of this approach simply lies in its flexibility. Without changing the general structure of the s-process, v-processes can be added to an h-process by registering a new web service (EJB) with the registry repository (J2EE container). On the other hand, web services of external service providers could be added to the existing EBF Process Model by simply connecting them via the EBF-IF Enterprise Service Bus with the EBF environment.

Integrating Third-party Processes

Integrating existing web services of third-party service providers with an own delimited SOA-based enterprise system is one of the most appealing strengths of the service-oriented architecture concept. The EBF-AF, by leveraging SOA technology to a very high extent, opens the possibility for special value generation through the integration of other external web services. They would be integrated via the EBF Integration Foundation running as a JBoss ESB. Thereby, the ESB would ensure that the external web service registers with the registry repository in the JBoss application server and thus announces its existence to the rest of the EBF-AF services.

One of those external web services the EBF-AF could include in its initial design is the Microsoft .Net Passport[20] service. Figure 53 detailed a user management component as part of

[20] Also known as Microsoft Windows Live ID.

the EBF Application Framework which was intended to govern the functional domain of access and rights management throughout all EBF components. Microsoft .Net Passport represents a powerful and zero-cost "Single Sign-In" (SSI)[21] online solution that can be used from any kind of application to provide an integrated and cross-component user management experience. Microsoft furthermore provides a Self Development Kit (SDK) for its .Net Passport service that in detail describes how the .Net Passport service could be linked to the EBF Integration Foundation and how EBF services could then communicate with .Net Passport to obtain user-related log-on data.

.Net Passport

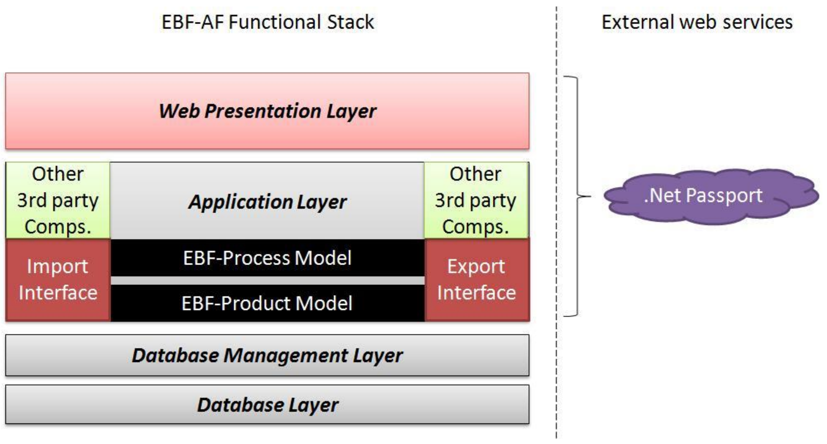


Figure 78: Integrating the .Net Passport web service with the EBF-AF as a central rights and access management component.

Another external web service that could be leveraged in the EBF Process Model refers to the payment process (p-process) in the transaction phase. Instead of implementing the whole payment process for the transaction phase, the web service Paypal could be added to the EBF-AF as an external service seamlessly interacting with the rest of the process chain. Paypal could be used to generate invoices for customers (see chapter three: p.ig v-process) and to receive a payment confirmation (see chapter three: p.pc v-process), while the payment data would come from upstream EBF processes. Figure 79 demonstrates how Paypal could simply replace the payment process in the EBF Process Model. Hence, the payment process would be outsourced to an external web service.

Paypal

[21] SSI describes a concept where users, no matter which application they are using, only log in once to a centralized rights and access management service.

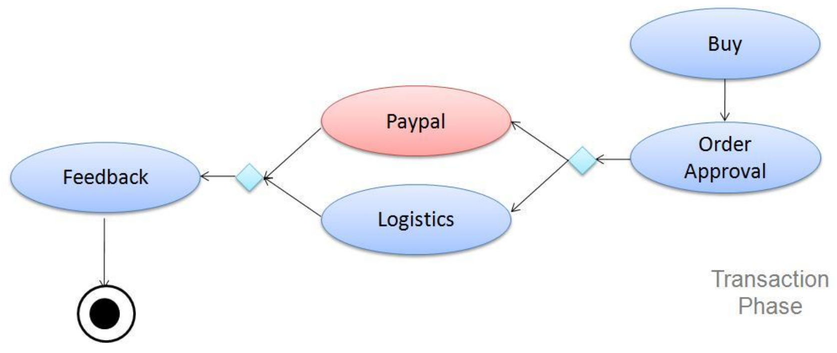


Figure 79: Embedding an external web service as the payment process instance in the EBF Process Model.

Quick Review – Implementing the EBF Process Model

By leveraging SOA-based technologies, the EBF Process Model architecture that was initially designed came for free. H-processes as well as v-processes can be implemented as web services and announce their existence to the application server (internal web services) or the EBF Integration Foundation (external web services). The h-processes would serve the role of being the domain entry point for, for example, search. While exposing an interface to the user of which search options exist, the search h-process would firstly accept the search parameters entered by the user and then marshal them to the v-process that implements the desired search method, such as keyword search, browse or similarity matching. After completion of the v-process, the search h-process would format the results of the v-process to fit the interface with adjacent processes or to be reportable to the user.

5.8 Summary

This chapter started out with a statement of the requirements to an application framework (EBF-AF) that could support the EBF-PPM modeled in chapter three. Thereby, all the requirements in one way or another aimed at minimizing the TCO of the EBF-AF in order to be affordable by SME.

In the following, the framework concepts that were introduced in chapter four were applied to the EBF-AF. The EBF-AF was subdivided into framework core and library, whereby the core would implement the EBF-PPM as well as provide mechanisms for adding library components to the framework.

The elaborations on the EBF-AF's architecture were based on the assumption that a traditional client/server architecture would not be sufficient to provide modern enterprise functionality, such as load-balancing and fail-over. Aiming at a low TCO while considering the themes "Avoiding Maintenance" and "Enabling Maintenance" (Neumann et al., 2009), the EBF-AF would need to aim at keeping business logic separate from database and presentation logic. This resulted into an initial three-tier architecture proposal for the EBF-AF functionality stack.

One of the key matters of success that would allow the EBF-AF for achieving all its goals was the way in which its functionality is distributed: through componentization. Therefore, the EBF-IF revealed a component bus architecture that had to support an easy and seamless integration of EBF components with other EBF components as well as external web services. One characteristic of the EBF-IF was its Component Spawn Point metaphor. Red Spawn Points represented places on the bus for internal components to be attached, whereby black Spawn Points were meant for components of external providers to connect. By augmenting the EBF-AF architecture with SOA, many parts of the framework’s architecture could be achieved for free. The EBF-IF, for example, perfectly matched to an Enterprise Service Bus in an SOA. In the course of that, a complete SOA-based stack for the EBF Application Framework was presented.

In the next step, the EBF-AF SOA stack had to be turned into reality. An analysis of the two main existing SOA technologies .Net and J2EE that could be leveraged to implement the EBF revealed that both from a functional as well as TCO point of view do not differ very much. However, a decision in favor of J2EE could be made based on its potentially lower opportunity cost.

Having decided for Java as the programming language in which the EBF-AF would be realized opened the way for the conceptual implementation of the Product as well as the Process Model. While chapter three already introduced the syntax for the EBF-PDL, it was still open how to transform the PDL syntax into Java and hence make the content of a PDL file accessible from within the product model. This gap was filled in by the EBF-PDL compiler, whereby a prototype tried to provide evidence for the validity of its concepts. The compilation process of the EBF-PDL compiler was formalized in a process diagram and the individual steps exemplarily highlighted. Since the EBF-PDL features Java code to be added to the XML product description, one point of criticism could include that the chosen approach could potentially have negative effects on maintenance and error handling. However, by providing a graphical EBF-PDL editor that allows for editing product descriptions visually and without writing any XML, the concerns expressed above might diminish.

By having JBoss as middleware for the EBF-AF, most parts of the EBF-AF infrastructure came for free. The EBF-IF, for example, that aims at letting components interact seamlessly with each other could completely be mapped to JBoss’ Enterprise Service Bus. Hence, by using the ESB, the EBF Product and Process Model could benefit from a very high lever. Basically, the entire infrastructure for exposing h-processes and v-processes was already given, so that the elaborations on the EBF Process Model could really focus upon how h-processes interact with v-processes. By sampling the s-process of the transaction phase, it was demonstrated how h-processes can be implemented in general. It was shown how the s-process wraps the search-related v-processes and exposes them to the framework applicant via a web interface. Lastly, it was briefed how the integration of other, external web services, such as Microsoft .Net Passport or Paypal, can be achieved similarly to the integration of normal EBF components.

The next chapter, chapter six, will introduce a vision in which the EBF-AF demonstrates its practical relevance. By leveraging the EBF-AF, the vision aims at providing SME with state-of-the-art e-commerce and e-business technology and thus to finally benefit from the development that has taken place in this sector. One driver that will significantly determine the success of this enterprise will be the EBF-AF's low TCO characteristic.

Part III: Goliath – The Digital City

For a too long time already, small and medium sized businesses have been hiding from the electronically connected world. As information became a seamless vehicle utilized by many, the way enterprises offer and act on markets has changed fundamentally. Every year, e-sales and e-marketing rose despite all crises. A new generation of internet-based educated people is emerging. For them, the web and all its services and offered goods are the primary way of shopping and acquiring knowledge. One can only hypothesize what this means for traditional printed media. Maybe the last generation of book readers will vanish soon. Conducting business has become a question of how enterprises can best adapt to this new world.

Several studies have shown that traditional retailers who started exposing their offerings on the internet could increase their sales as new and more people could be reached. Of course, this would be a preferable situation for all businesses. But what hinders the engagement of those that have not turned to e-commerce yet? Explanations might be found in the high cost for necessary infrastructure to host electronic platforms and services as well as the knowledge of how to effectively market offerings via the new type of media. Since SME have only a very limited budget, they usually cannot invest the necessary time and money to manage a successful entry into the world of e-business. The goal of the EBF is to ultimately scatter those barriers.

The EBF as they were introduced in this work represent a flexible framework for e-commerce. They enable the seamless offering of goods and services from companies, thus widening the market. From an applicant perspective, the EBF includes the computerized foundation, the knowledge and the conceptual marketplace that is necessary to run e-commerce. With SaaS, customers can step into this field by paying only a small amount of money. Extensions to the core business are only minimal, since merely the resources to maintain the electronic offerings and to handle the delivery are required. On the buyer side, the EBF enables Reverse Trading, a more convenient way of matching buyer requirements with offerings. Through a uniform description of offers, buyers can choose according to their preferences without over-investing time that would be required to compare offers from different, scattered marketplaces. This increases the overall sales volume, lowers barrier entries and potentially boosts welfare.

The final part of this work will discuss a vision which states how SME could benefit from the new possibilities the EBF offers to them. Thereby, the elaborations contained in chapter six are to be understood more as a prospect of possible future trends and tendencies rather than a concrete plan or strategy. Many topics are still unclear and actual results and insights about the feasibility of the assumptions that were made are expected to be gained from a prototype which could possibly be launched in the near future.

6 The Goliath Vision

6.1 Goliath Mission Statement

The accomplishments of electronic commerce and modern logistics have enabled enterprises to merchandise and distribute their products and services across the whole world overcoming even the furthest distances. Potential customers that are interested in a product or a service can research on prices and features on the supplier's website. They can get in touch with the supplier via internet communication channels and in most cases get immediate pricing information about the supplier's offerings. 24 hours a day customers can purchase products and services, no matter where on the globe they currently are.

Many e-commerce solutions were developed to provide suppliers with a virtual storefront. Those e-commerce solutions are most often licensed from a software provider and hosted on a dedicated web server. The cost for licensing, customization and maintenance of the virtual storefront consumes a significant amount of the total revenue that is made with the e-business. Businesses with a solid revenue stream can easier bring up those investments than businesses that operate upon very small profit margins and volumes. For SME that are run by, for example, one to five individuals it becomes literally impossible to afford an investment as such. Even though the investment amortized within a very short period of time, those businesses would not be able to gather the necessary funds. Certainly, SME could sidestep to a personal storefront on mass marketplaces, such as eBay or Amazon, but firstly such general purpose marketplaces might not provide all the functionality that is required and secondly do such marketplaces not integrate with other systems. In case of a small import/export business desiring do reflect currency-related exchange rates (FX rates) in the prices of its goods, its electronic storefront requires some sort of smarts to obtain FX rates from a financial market, such as Onvista[22], and include it in its pricing. Another business might maintain its product catalogue already electronically and might want to integrate its storefront with it. Again, a general purpose online market does not provide solutions for this kind of scenario.

The SME Dilemma

While larger businesses already benefit from the developments that have taken place in the field of e-commerce, small businesses still seem to remain excluded (Heins, 2008). Modern commercial concepts, such as one-to-one marketing and a customer relationship-based (CRM) business strategy require business logic that can only be provided through IT. Also, an efficient communication with upstream or downstream partners of the same supply chain through integrated procurement or supply chain functionality (SCM) is not given. Many SME still do not have a website or a virtual storefront on which they can distribute their goods and services over the internet (Heins, 2008). At the current stage of the evolution of e-commerce, the cost seems to

SME Cost Barrier

[22] http://www.onvista.de

be the most significant barrier for SME, which hinders them from making use of modern e-commerce or electronic business concepts, including CRM and SCM.

Picturing the Future

The technological isolation of SME in the above described current situation is exactly where the story behind Goliath picks up. The naming for Goliath refers to the size of a biblical giant, whereby his name pictures the dimension of the project. In the Goliath vision, SME are provided with e-commerce technology which, due to its enormous flexibility, can cater for any imaginable e-commerce scenario while requiring only a very low investment that is affordable by each and every business. Customization to the specific needs of an individual business, no matter what sort of products or services are dealt with, is very easy to achieve and infrastructure not at all or only to a minimum extent required. Different businesses of different markets use the same e-commerce solution, no matter whether they are local retailers, food marts, travel bureaus, dentists, pizza deliveries or hair cutters. By integrating all the e-commerce instances of the single businesses into one big portal, communication between each other and better alignment of transactions with business partners becomes possible. The aggregation of all electronic businesses to one single entry point would allow for the unlocking of new services and experiences for customers which have never been seen before.

Goliath Mission Statement

The goal of the Goliath Vision is to provide each and every business with the possibility to make use of modern e-commerce technology while combining them to one big virtual eco-system. Goliath's methodology to achieve this eager goal will be to provide an e-commerce suite that, due to its low cost, is accessible for each and every business while concurrently allowing for the seamless integration of business partners. Ultimately, the Goliath Vision contains the hope to in future unlock totally new and never seen end-consumer experiences, such as scheduling an appointment with a doctor over the internet, being able to track a waiting queue for, for example, government administration via a mobile phone or simply ordering comestible goods from the supermarket around the corner over the internet.

Quick Review – Goliath Mission Statement
The Goliath Vision describes an attempt to equip Small and Medium Enterprises (SME) with state-of-the-art e-Commerce technology, so that they can implement modern IT-supported business strategies (e.g. CRM, SCM).

6.2 The Goliath Scope

EBF as Key

Throughout this work, the specific requirements to the EBF Application Framework as prevailing leitmotivs over and over again drove design-related aspects and decisions. By focusing upon an architecture and deployment model that aims at keeping the TCO for framework applicants low as well as using existing standards and technologies, the EBF seems to be the key for turning Goliath into reality.

As it will be described later in the Goliath rollout plan, the Goliath Vision should start with an analysis of existing business and their processes. Once this analysis was performed and results were incorporated in the Vision, the realization can begin. Exposing all products and services a business offers via a virtual storefront on the internet should be the first step in the realization of Goliath. Therefore, businesses should be equipped with remotely hosted SaaS instances of the EBF-AF. Once the product and service portfolio of a business was understood and their properties modeled, product descriptions for each and every product or service can be implemented and added to the product model of the EBF-AF instance. It is furthermore assumed that in most cases no customization to the EBF Process Model would be required. This is because the goal of the EBF-PPM was to propose a standardized way of how e-commerce is conducted. As soon as the EBF-AF instance was customized with all product-relevant data, contact and payment as well as other administrative information, this instance is ready to go live. The customized product and process model would be exposed to potential customers via the EBF's web interface, which in most cases would be a virtual storefront. Thereby, the web interface is merely a template that visualizes the EBF-AF's API and thus can be easily customized to cater for individual design ideas. For customers, the EBF-AF instance would look like a normal online shop.

Leveraging the EBF

An approach towards starting the realization of the Goliath Vision could be to launch a pilot project in cooperation with a selected municipality. Cutting down the scope of Goliath would allow for an analysis of which businesses and service providers exist within the city. The existing businesses within a city could be equipped with an instance of the EBF while customizing the framework to be able to reflect an online version of their daily sales and service operations. From this pilot project detailed insights into the requirements and development of SME e-commerce could be gained. Standard customization models for the EBF and web interfaces for similar businesses could be developed and the benefits of a potential standardization of similar business processes even further leveraged. The theory prevalent in every chapter of this work would finally have a chance to be applied in practice.

Goliath – The Digital City

While the plan is to introduce the EBF to a major set of businesses and service providers in one selected municipality, one could also imagine this enterprise as the digitizing of a city. All products and services, irrespective of whether commodities, health services or other services, could be purchased by the residents of the pilot city over the internet. The daily food shopping, the appointment scheduling with a doctor or a hair cutter, the home delivery pizza order or the ordering of a taxi could be accomplished from a normal desktop computer or even a mobile phone.

Business Segmentation

The introduction of the EBF to a city could happen step by step, one business segment after another. Therefore, of course all business segments would firstly need to be identified. An initial and very high-level segmentation of businesses into vendors and service providers could contain the following:

- **Vendors**
    - *1st Level Vendors*
        - Deal with consumer items
        - E.g. food shops, food marts, shopping centers
    - Assumed functionality requirements
        - Exposure of product catalog
        - Online shopping with virtual storefront
        - Augmented Communication Channels
    - *2nd Level Vendors*
        - Deal with commodities
        - E.g. boutiques, electronic shops, etc.
    - Assumed functionality requirements
        - Exposure of product catalog
        - Online shopping with virtual storefront
        - Augmented Communication Channels

"Vendors" Category

The "vendors" category encompasses all businesses that trade real products. While 1st Level Vendors deal with everyday consumer items that are frequently bought by a large number of customers, 2nd Level Vendors deal with commodities and could encompass cloth boutiques or electronic shops. The target of Goliath for this segment is to digitalize those businesses' commerce while providing detailed information about the products offered. While the majority of those businesses is currently depending on the customers coming to their place, an area-wide coverage of Goliath/EBF technology could help realizing a vision in which customers could perform all their daily shopping via the internet. The integration of a later briefly mentioned logistics service provider could help distributing the orders to the customers while the customers would benefit from low shipping cost due to large scales of orders. The current communication channels (mostly telephone) could be enhanced with augmented communication channels, such as video telephony or live chat for answering customer inquiries.

"Service Providers" Category

Service providers on the other hand could be categorized as follows:

- **Service Providers**
  - *General Service Providers*
    - Offer different services
    - E.g. travel bureaus, agencies, taxis, home-moving
  - Assumed functionality requirements
    - Exposure of service catalog
    - Online shopping with virtual storefront
    - Set up appointments online
    - Online queue tracking via mobile phone
    - Augmented Communication Channels
  - *Health and Personal Service Providers*
    - Provide health-related services
    - Most often require customer to come to their location
    - E.g. doctors, dentists, hair cutters, beauty saloons
  - Assumed functionality requirements
    - Set up appointments online by picking from calendar
    - Online queue tracking via mobile phone

- Augmented Communication Channels

- *Financial Service Providers*
  - Provide financial services
  - E.g. banks, financial consultants, tax consultants
- Assumed functionality requirements
  - Set up appointments online by picking from calendar
  - Exposure of product catalog
  - Online shopping with virtual storefront
  - Augmented Communication Channels

- *Governmental Service Providers*
  - Offer administrative services
  - E.g. ward offices, board of trade, registry office
- Assumed functionality requirements
  - Set up appointments online by picking from a calendar
  - Online queue tracking via mobile phone

General Service Providers

General Service Providers, such as travel bureaus, agencies or taxis, offer "normal" services similarly to vendors trading commodities, just that their products do not have a concrete physical appearance. They would benefit most from a virtual storefront that exposes their services and makes them bookable via the internet as well as augmented communication channels, such as video telephony or live chat.

Health and Personal Service Providers

Health and Personal Service Providers encompass doctors, dentists, hair cutters and beauty salons. As those service providers rely on their customers to come to the place, Goliath or EBF technology could significantly help with scheduling appointments. A calendar, for example, could be exposed via the internet from where clients could pick a date to their convenience. Furthermore, if those service providers are likely to have their clients queue at their places, the EBF could provide a queue tracking service that would provide the client with live data about where in the queue he would currently be and how much longer he would have to wait. Consulting about medicine or a first diagnosis about the urgency of the client's condition could be conducted via augmented communication channels including Video Counseling with the doctor.

Financial Service Providers

Financial Service Providers are to a high extent similar to Normal Service Providers just that their security level might differ at certain places.

Governmental Service Providers

Governmental Service Providers, due to the nature of their business, behave similar to Health Service Providers. However, a virtual storefront could also help them to streamline the processes behind the services they offer (e.g. extension of passport).

Quick Review – The Goliath Scope
The key towards realizing the Goliath Vision is the EBF Application Framework. The specific properties of the framework cater for a powerful but low cost e-commerce platform that can be used by literally each and every kind of business. By equipping the businesses of a municipality with EBF instances, it will become possible to streamline goods and services of a whole city via one single access channel: the internet. By using the internet, so the vision says, the way everyday shopping is conducted nowadays might soon be augmented by new and never seen end-user experiences.

6.3 Goliath Rollout

Rollout Phases

While an initial attempt towards classifying the businesses that Goliath would be applied to was already given, a detailed realization or rollout plan was not yet formulated. Once the EBF are implemented, the rollout could happen in three phases: 1.) Preparation Phase, 2.) Implementation Phase and 3.) Follow-up Phase. While this chapter is merely a sketch of the future procedure, these phases basically detail the goals and concepts of Goliath to make the vision come true. Potential contents of individual phases are briefed in keywords as follows:

PI) Preparation Phase

1. Creation of Goliath Vision Statement
   a. Formulate the goal and the method
2. Embedding of the Goliath enterprise in an academic project
   a. Establish seriousness
3. Identification of suitable rollout cities
   a. Willingness of city administration to support project?
   b. Reasonable size of the city?
   c. Economical strength of the city?
4. Identification of partners and sponsors
   a. Propose Vision Statement to potential (industry) partners
   b. Get partners' buy-in
5. Decision for one rollout city
   a. Create a budget plan

P II) Implementation Phase

6. Analysis of existing businesses in the city
   a. Get business buy-in on Goliath
7. Segmentation of businesses
   a. Classification of businesses into rollout waves
8. Validation of the EBF being able to cater for all business needs
9. Stepwise implementation of the roll-out waves

PIII) Follow-up Phase

1. Refinement, extensions, new components and functionality

Quick Review – Goliath Rollout

The rollout of the Goliath Vision could happen in the three phases Preparation, Implementation and Follow-up. A detailed rollout plan, however, will have to be established in future efforts.

6.4 Summary

By sampling the Goliath Vision, it was shown how the EBF could be applied in practice to a project that would unlock end-consumer experiences, which in their richness were never seen before. Enabling all businesses within one city to offer their products and services to their already existing customers over the internet might even herald the next generation of e-commerce. For the daily food shopping, for example, consumers would not need to go to a food mart anymore, but could rather order everything via the virtual storefront of their favorite supplier. The integration of an external logistics provider into Goliath would allow for orders being promptly shipped to the customer while a big amount of order volume would keep the shipping cost low.

The key factor of success for the Goliath Vision would lie in the EBF's low cost (= low TCO) characteristic. With the EBF's SaaS approach, every business that owns a desktop PC with an internet connection could implement an own virtual storefront. The huge scale of potential EBF applicants would allow the EBF for being provided at a very low price.

While technologically everything seems possible, the biggest challenges for Goliath would presumably evolve from business itself. It might become very difficult for the project to get the cities' or communes' buy-in as well as to convince businesses of the benefit they could realize. To clarify involved risks and define the method, this chapter concluded in an initial high-level rollout plan for the Goliath Vision.

7 Conclusion

This work was motivated by the technological gap between state-of-the-art e-commerce as it is used in large enterprises and e-commerce as it is prevalent in SME. In most cases, businesses that are run by only a few individuals do not use any form of e-commerce. One reason for that could lie in customizable e-commerce solutions, which are provided by specialized vendors, simply being too expensive. On the other end, low-cost "mass e-commerce" solutions (e.g. eBay, Amazon Marketplace) are often too limited in their functionality. Even though desirable, virtual storefronts or online shops that expose products or services over the internet cannot be afforded by SME.

Closing the Gap

To close this technological gap, this work aimed at proposing an e-commerce framework that, due to its enormous flexibility, can cater for a huge variety of e-commerce transaction types and scenarios. In the same instance, the approach that was presented minimized the Total Cost of Ownership by leveraging free but standardized technology. Distributing the EBF Application Framework as SaaS made the framework not only affordable but also maintainable by SME.

Digitizing Cities

While being theoretically applicable to each and every kind of business, the EBF-AF could become the enabler for next generation e-commerce experiences. The Goliath Vision pictured a project in which all vendors and service providers of a pilot city would be equipped with a virtual storefront for their products. Step by step, all businesses within this city would be digitalized, no matter whether food shops, boutiques, health service providers or governmental administrations. Regarding the realization of the Goliath Vision, the characteristics of the EBF-AF made it be an optimal foundation for this enterprise.

Integrated Component Architecture

Amongst others, the EBF-AF featured a highly flexible and domain-independent product and process model. While implementing the design of the EBF-AF, special emphasize was put upon easy-to-access maintenance and extension. Both were achieved through a highly componentized architecture with components being loosely-coupled and integrated with each other via an Enterprise Service Bus. To allow for the EBF-AF being able to adapt to varying workloads, load balancing and scalability mechanisms ensured that EBF components are physically distributable.

Points of Criticism

Points of criticism to the EBF-AF could address the attribute handler logic that was featured in the EBF Product Model. The product model's high flexibility implemented by its attribute handler logic was only possible due to some kind of late binding of Java classes. Thereby, attribute handlers could be defined in a product description and would be compiled and dynamically linked to the EBF Product Model. Late binding, however, would also open the door for malicious code injection. Future elaborations on the EBF-AF could explicitly address security-related questions and also pick up this point again. Other concerns related to the proposed design revolved around XML being stored together with Java code in one product description file as this might potentially result into some sort of "Maintenance Hell". Future refinements to the EBF-AF could, for instance, feature a graphical editor that would expose a product description file to the EBF applicant in a way that it is easier to edit and to maintain.

Future Work

Other potential candidates for future work, which the text mentioned briefly but not in detail, include the database management layer of the EBF function stack. Future efforts would need to be put into the design and the implementation of this layer as sophisticated data access and management logic would be mission-critical for maintaining the integrity of physically distributed EBF components. Also, it was not yet detailed how the functionality of the EBF-AF would be accessible for the framework applicant via the web presentation layer. Ideally, the EBF-API would be exposed via plain HTML tags, so that EBF web applications could be created simply by editing HTML, but without writing any code.

Once the EBF made its way to the business applicant and the theories that were hypothesized throughout this work were validated in practice, adding components for CRM or SCM support would be amongst the next steps. But until then, the EBF will still have a long way to go.

Robert Neumann March, 2009 (Tokyo, Japan)

Appendix

The source code of the EBF Product Model prototype consists of main.java, ProductDescription.java, ProductNode.java and CustomProductNode.java.

main.java

```
import java.util.ArrayList;

public class main
{
	public static void main(String[] args)
	{
		ProductDescription ps = new ProductDescription("Car1");
		ps.addNode(new ProductNode("make", "BMW"));
		ps.getNode("make").addNode(new ProductNode("type",
		"M6"));

		ps.addNode( new CustomProductNode("price", "100" ));
		ps.addNode("color", "blue");

		System.out.println( ps.getNode("price").getValueEx() );
		System.out.println( ps.getNode("make").getNode("type").ge
		tValueEx() );

		ArrayList paramsGetPriceInCurrency = new ArrayList();
		paramsGetPriceInCurrency.add(new String("JPY"));

		System.out.println( ps.getNode("price").callAttributeHand
		ler("getPriceInCurrency", paramsGetPriceInCurrency) );

		System.out.println("EOA");
	}
}
```

ProductDescription.java

```
import java.util.HashMap;

public class ProductDescription
{
	public ProductDescription(String _strName)
	{
		setName(_strName);
		nodes = new HashMap();
	}

	public String getName()
	{
		return strName;
	}

	public void setName(String _strName)
	{
		strName = _strName;
	}

	public HashMap getAllNodes()
	{
		return nodes;
	}
```

```
        public ProductNode getNode(String _strName)
        {
                return (ProductNode) nodes.get(_strName);
        }

        public void addNode(ProductNode _pNode)
        {
                nodes.put(_pNode.getName(), _pNode);
        }

        public void addNode(String _strName, String _strValue)
        {
                nodes.put( _strName, new ProductNode(_strName,
                _strValue));
        }

        protected String strName;
        protected HashMap nodes;
}
```

ProductNode.java

```
import java.lang.reflect.Method;
import java.util.ArrayList;
import java.util.HashMap;

public class ProductNode extends ProductDescription
{
        public ProductNode(String _strName, String _strValue)
        {
                super( _strName );
                setValue(_strValue);
        }

        public String getValue()
        {
                return strValue;
        }

        public String getValueEx()
        {
                if( strDefaultAttributeHandler != null )
                        callAttributeHandler(strDefaultAttributeHandler,
                        new ArrayList() );

                return strValue;
        }

        public void setValue(String _strValue)
        {
                strValue = _strValue;
        }

        public HashMap getAllNodes()
        {
                return nodes;
        }

        public boolean registerDefaultAttributeHandler(String _
        strDefaultAttributeHandler)
        {
```

```
			strDefaultAttributeHandler = _strDefaultAttributeHandler;
			return true;
		}

		public Object callAttributeHandler(String _strHandler,
		ArrayList _params)
		{
			try
			{
				return
				findAttributeHandler(_strHandler).invoke(this, new
				Object[]{_params});
			}
			catch(Exception e)
			{
				return null;
			}
		}

		public Method findAttributeHandler(String strAttributeHandler)
		{
			Method[] methods = this.getClass().getMethods();

			for(int i = 0; i < methods.length; ++i)
			if(methods[i].getName().equals(strAttributeHandler))
					return methods[i];

			return null;
		}

		public String toString()
		{
			return getValue();
		}

		private String strValue;
		private String strDefaultAttributeHandler = null;
	}
```

CustomProductNode.java

```
	import java.util.ArrayList;

	public class CustomProductNode extends ProductNode
	{
		public CustomProductNode(String _strName, String _strValue )
		{
			super(_strName, _strValue);

			//>RegisterDefaultHandler
			registerDefaultAttributeHandler("foo");
			//<RegisterDefaultHandler
		}

		//>Handler
		public String getPriceInCurrency(ArrayList _params)
		{
			String strCurrency = (String) _params.get(0);

			if( strCurrency.equals("USD"))
			{
				Integer i = new Integer( getValue());
```

```
                        return new Double( i.intValue() * 1.33).toString();
                  }

                  if( strCurrency.equals("JPY"))
                  {
                        Integer i = new Integer( getValue());
                        return new Double( i.intValue() * 125).toString();
                  }

                  return null;
            }

            public void foo(ArrayList _params)
            {
                  Integer iPrice = new Integer( getValue() );

                  int iTemp = iPrice.intValue();
                  setValue( (new Integer(iTemp + 1)).toString() );
            }
            //<Handler
      }
```

Table of Figures

Bibliography

Adair, D. (1995). *Building Object-Oriented Frameworks.* AIXpert.

Adler, J. (1996). *Informationsökonomische Fundierung von Austauschprozessen. Eine nachfragerorientierte Analyse.* Gabler.

An, Z., & Peters, D. (2003). *On the Description of Communications Between Software Components with UML.* Memorial University of Newfoundland, Newfoundland.

Arrango, G., Pietro-Diaz, G., & Pietro-Diaz, R. (1991). *Domain Analysis Concepts and Research Directions.* IEEE Computer Society.

Bailey, J., & Bakos, Y. (1997). *An exploratory study of the emerging role of electronic intermediaries.* International Journal of Electronic Commerce.

Bakos, J. (1992). *A Strategic Analysis of Electronic Marketplaces.* MIS Quarterly (September).

Benjamin, I., & Wigand, R. (1995). *Electronic Markets and Virtual Value Chains on the Information.* Sloan Management Review.

Benjamin, R., Malone, T., & Yates, J. (1986). *Electronic Markets and Electronic Hierarchies: Effects of Information Technology on Market Structures and Corporate Strategies.* Massachusetts Institute of Technology, USA.

Berryman, K. (1998). *Electronic commerce: Three emerging strategies.* The McKinsey Quaterly.

Bichler, M. (2001). *The Future of e-Markets: Multidimensional Market Mechanisms .* Cambridge University Press.

Bosch, J. (1997). *Summary of the Second International Workshop on Component-Oriented Programming .* WCOP'97.

Brickley, J., Smith, C., & Zimmerman, J. (2006). *Managerial Economics & Organizational Architecture.* Mcgraw-Hill Higher Education.

Bullinger, H.-J., Groh, G., & Kopperger, D. (1997). *TCO.* University of Stuttgart, Germany.

Cearley, D., Fenn, J., & Plummer, D. (2005). *Gartner's Positions on the Five Hottest IT Topics and Trends in 2005.* Gartner Consulting Group.

Chappell, D. (2006). *Understanding .NET (2nd Edition).* Addison-Wesley Professional.

Durvasula, S., Kumar, A., Lamb, J., Mitchell, T., Oral, O., Pai, Y., et al. (2006). *SOA Practitioners' Guide Part 2: SOA Reference Architecture.* Combined Effort.

Eskelin, P. (1999). *Component Interaction Patterns.* Ernst & Young LLP.

Evjen, B., Sharkey, K., Thangarathinam, T., Kay, M., Vernet, A., & Ferguson, S. (2007). *Professional XML.* Wrox.

Froehlich, G., Hoover, H., Liu, L., & Sorenson, P. (1997). *Designing Object-Oriented Frameworks.* Universtiy of Alberta, Canada.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley.

Gangopadhyay, D., & Mitra, S. (1995). *Understanding Frameworks by Exploration of Examples.* Proceedings of 7th international Workshop on Computer Aided Software Engineering.

Gemünden, H. (1990). *Innnovationen in Geschäftsbeziehungen und Netzwerken.* University of Karlsruhe, Germany.

Göldi, A., & Stuker, J. (2003). *.NET vs. J2EE: In welche Plattform investieren?* Namics AG.

Gosling, J., Joy, B., & Bracha, G. (2005). *Java(TM) Language Specification, The (3rd Edition).* Addison Wesley.

Heins, E. (2008). *Handarbeit dominiert noch im Berichtswesen.* IS Report.

Hernandez, J., Keogh, J., & Martinez, F. (2005). *SAP R/3 Handbook.* Mcgraw-Hill Professional.

Heuer, A., & Saake, G. (2000). *Datenbanken Konzepte und Sprachen.* mitp.

Horngren, C., Datar, S., & Foster, G. (2008). *Cost Accounting: A Managerial Emphasis.* Prentice Hall International.

Kaplan, S., & Sawhney, M. (2000). *E-Hubs: The New B2B Marketplaces.* Harvard Business Review (May-June).

Kerrigan, R., Roegner, E., Swinford, D., & Zawada, C. (2001). *B2Basics.* The McKinsey Quarterly.

Koppelmann, U., Brodersen, K., & Volkmann, M. (2001). *Electronic Procurement im Beschaffungsmarketing.* WiSt.

Koskimies, K., & Mossenback, H. (1995). *Designing a Framework by Stepwise Generalization.* Proceedings of the 5th European Software Engineering Conference.

Krähenmann, N. (1994). *Ökonomische Gestaltungsanforderungen für die Entwicklung elektronischer Märkte.* Hochschule St. Gallen, Switzerland.

Kramer, T. (2000). *eBusiness: Strategies, Frameworks, Business Models, Networking, Security, ePayment, eProcurement, SCM, ERP, CRM, Case Study: Channel Conflicts.* GRIN Verlag.

Krishna, V. (2002). *Auction Theory.* Academic Press.

Laudon, K., & Laudon, J. (2005). *Management Information Systems. Managing the Digital Firm.* Prentice Hall International.

Lücke, T. (2005). *Frameworks.* University of Hanover, Germany.

Malone, T., Yates, J., & Benjamin, R. (1989). *The Logic of Electronic Markets.* Harvard Business Review (May-June).

Massengill, D., & Jackson, S. (2006). *SAS/AF – Building a Tree View Hierarchy with Drag and Drop.* SAS Institute Inc., USA.

Merz, M. (1999). *Elektronische Dienstemärkte: Modelle und Mechanismen des electronic commerce.* Springer.

Müller, A. (2008). *SOA - Entschlüsselt.* Hochschule für Technik und Wirtschaft Karlsruhe, Germany.

Nelson, C. (1994). *A Forum for Fitting the Task.* IEE Computer 27.

Neumann, R. (2008). *Semantic Similarity - An Introduction to Measures and Resources.* Otto-von-Guericke University Magdeburg, Germany.

Neumann, R., Günther, S., & Zenker, N. (2009). *Reengineering Deprecated Component Frameworks: A Case Study of the Microsoft Foundation Classes.* Otto-von-Guericke University, Magdeburg.

Neumann, R., Shoukry, A., Paul, G., & Zenker, N. (2008). *Maintaining Object-oriented Component Frameworks - A Case Study of the MFC (Bachelor Thesis).* Otto-von-Guericke University Magdeburg, Germany.

Nickull, D., Reitman, L., Ward, J., & Wilber, J. (2007). *Service Oriented Architecture (SOA) and Specialized Messaging Patterns.* Adobe Systems Incorporated, USA.

Oppel, K., Hartmann, E., Lingenfelder, M., & Gemuenden, H. (2001). *Electronic B2B Marketplaces - Impact on B2B Transactions and Relationships?* Philipps-University Marburg, Germany.

Orfali, R., Harkey, D., & Edwards, J. (1994). *Essential Client/Server Survival Guide.* Wiley.

Pereira, R., & Price, R. (2002). *Component Interface Pattern.* Federal University of Santa Catarina, Brazil & Federal University of Rio Grande do Sul, Brazil.

Picot, A., Bortenlänger, C., & Röhr, H. (1996). *Börsen im Wandel.* Knapp.

Plinke, W. (1997). *Grundlagen des Geschäftsbeziehungsmanagements.* Springer.

Pratt, M. (2002). *Introduction to ISO 10303 - the STEP Standard for Product Data Exchange.* National Institute of Standards and Technology, USA.

Ranta-Eskola, S. (2001). *Binary Space Partioning Trees and Polygon Removal in Real Time 3D Rendering.* Uppsala University, Sweden.

Rautenstrauch, C., & Schulze, T. (2002). *Informatik für Wirtschaftswissenschaftler und Wirtschaftsinformatiker.* Springer.

Rensmann, B. (2007). *Mediation Patterns in e^{3}value.* Tilburg University, Netherlands.

Renzel, K., & Keller, W. (1997). *Client/Server Architectures for Business Information Systems: A Pattern Language.* sd&m GmbH & Co. KG, Germany & EA Generali, Austria.

Research, M. &. (2000). *Coming into Focus: Using the lens of economic value.* McKinsey.

Siebert, H. (1992). *Einführung in die Volkswirtschaftslehre.* Kohlhammer.

SIIA. (2006). *Software As a Service: A Comprehensive Look at the Total Cost of Ownership of Software Applications.* Software and Information Industry Association.

Sparks, S., Benner, K., & Faris, C. (1996). *Managing Object-Oriented Framework Reuse.* IEE Computer.

Spulber, D. (1999). *Market Microstructure: Intermediaries and the Theory of the Firm.* Northwestern University, USA.

Stearns Sgarioto, M. (2000). *Internet-based trade exchanges.* Manufacturing Systems (December).

Strader, T., & Shaw, M. (2000). *Electronic Markets: Impact and Implications (in Handbook on Electronic Commerce).* Springer.

Stroustrup, B. (2000). *The C++ Programming Language: Special Edition.* Addison-Wesley Longman.

Sunil, M., Almirall, D., & Krishnan, M. (2006). *Do CRM Systems Cause One-to-One Marketing Effectiveness?* Institute of Mathematical Statistics (IMSTAT), USA.

Szyperski, C. (1996). *First International Workshop on Component-Oriented Programming.* WCOP'96.

Taligent. (1995). *The Power of Frameworks.* Addison-Wesley.

Thielscher, J. (2002). *J2EE und .Net zwei konkurrierende Standards?* Computerzeitung.

Timmers, P. (1999). *Electronic Commerce: Strategies and Models for Business-to-Business Trading.* John Wiley & Sons.

Varian, H. (1992). *Macroeconomic Analysis.* W. W. Norton.

Vawter, C., & Roman, E. (2001). *J2EE vs. Microsoft.NET - A comparison of building XML-based web services.* The Middleware Company.

Werfel, J., Xie, X., & Seung, H. (2005). *Learning curves for stochastic gradient descent in linear feedforward networks.* Massachusetts Institute of Technology, USA.

Wigand, R., & Benjamin, R. (1993). *Electronic Commerce: Effects on Electronic Markets.* Syracuse University, USA.

Williams, V., & Phillips, B. (1999). *E-Commerce - Small businesses venture online.* U.S. Small Business Administration, USA.

Williamson, O. (1985). *The economic institutions of capitalism.* The Free Press.

Statement of Autonomy

This is to confirm that this work was created autonomously by me, Robert Neumann, using only the declared sources of information and tools.

Robert Neumann March, 2009 (Tokyo, Japan)