Dynamic Pricing as a Service for E-Commerce Applications

Wim van der Zijden
University of Twente
PO Box 217, 7500 AE Enschede
The Netherlands
w.vanderzijden@student.utwente.nl

ABSTRACT
E-commerce is one of the few sectors in the current Dutch economy that is still experiencing substantial growth. This growing industry faces two key challenges: international expansion and the phasing out of warehouses. Dynamic pricing can aid in meeting those challenges. However, this is currently only available as tailor made solutions for companies with big IT development budgets. In this paper we research the possibility of offering dynamic pricing functionalities to web shops as a service, in line with the service oriented architecture (SOA) paradigm. We focus on e-commerce retailers that sell non-perishable goods in plenty supply. We introduce a categorization of dynamic pricing strategies with a three-dimensional model. On this we base a design of a SOA which we analyse using theoretical grounds and a practical case study. Two major problems have arisen from this analysis. First, a potentially negative impact on customer loyalty. Second, a potential problem with the data model for storing prices used by the underlying ERP system of the case study web shop. Based on these problems we had to conclude that we cannot confidently say that such a service is feasible.

Keywords
Dynamic pricing, time based pricing, e-commerce, web shop, online retail, cloud computing, service oriented architecture, SOA, loosely coupled systems, SaaS, SoaML.

1. INTRODUCTION
E-commerce is one of the few sectors in the current Dutch economy that is still thriving. In 2012 the e-commerce revenue in The Netherlands grew by 9% to nearly 10 billion Euros [14]. Together with Denmark and the UK, The Netherlands is amongst the frontrunners in e-commerce in the European Union [7].

In a developing market such as this one there is plenty room for innovation. There are two key challenges for e-commerce businesses:

1. Expanding internationally. Even though there is no obvious reason for a web shop to limit itself to the domestic market, only 9 percent of Europeans are purchasing goods from non-domestic retailers [9].

2. Larger catalogue, smaller stock. A huge potential benefit of a web shop in comparison with a traditional brick and mortar shop is making allowance for a much greater catalogue of products, since it is not necessary to physically present all the products. Actually, in an extreme situation, a web shop could operate without even having a warehouse at all.

An important aspect of both key challenges is the potential need for dynamic pricing. Dynamic pricing is a very promising way of maximizing profit and is already being used in several major markets such as airline tickets and Major League baseball games. Dynamic pricing can be defined as freely adjusting the price of your products in response to supply and demand conditions, even up to the level of the individual transaction [8].

To expand internationally, at least some extent of dynamic pricing is unavoidable. You may be dealing with different currencies, dynamic exchange rates, but the logistic cost will also play a role. For a seller of high margin luxury products, dynamic pricing will be a key factor in maximizing profits both in low and high-income countries.

For web shops that wish to capitalize on the possibility of small or even no warehouse capacity, dynamic pricing will also be of interest. In such cases, the web shop is no more than a channel between the manufacturer and the customer. Dynamic pricing could be one of its major competitive advantages, making sure that the price offered is competitive and yet profitable at all times.

1.1 Problem Statement
As mentioned, dynamic pricing is already in production in some major markets. The most well known applications are goods with a limited availability and a set utility date, such as airplane or theatre tickets. However, some big retailers like Amazon have also experimented with dynamic pricing [8].

The common denominator for all applications is that they involve major players that have big IT development budgets. This allows for tailor made solutions to their specific pricing needs. In order to make this functionality available to a wide range of web shops, a flexible dynamic pricing solution that can be integrated into existing web shops is needed.

To illustrate this goal, let's take a look at recent developments in online payments. It used to take considerable time and effort to build a payment gateway to enable customers to make online payments. Nowadays, several Payment Service Providers (PSP) have emerged. These can be easily integrated into any web shop.

This kind of architecture is also known as Service Oriented Architecture (SOA). In this architecture all functionalities are provided as independent services, specified in the business context and in terms of contracts between the provider and its clients. In this paradigm, the implementation details are irrelevant and of no concern for the client [9].
If an efficient and effective way can be found to apply SOA to the problem of dynamic pricing, this could be a major enhancement for e-commerce applications. This research will investigate the possibility of such a service.

1.2 Research Questions
The main research question is:

Can dynamic pricing functionality be offered to small to medium sized web shops, using the SOA paradigm?

The kind of web shop that we mean in this question is a typical retailer on the internet that sells non-perishable goods which are in plenty supply. This is a market that is not commonly associated with dynamic pricing. In order to answer this question, four steps must be taken: a categorization of dynamic pricing, an evaluation of which of these are applicable to such web shops, an assessment of the possible application of the SOA paradigm, and lastly, the practical application of a dynamic pricing service to a case study.

This leads to the following sub research questions:

1. What are some of the possible different dynamic pricing strategies and can they be categorized?

Dynamic pricing can take on many forms. One possible way of analysing different implementations is by looking at the scale of two dimensions: 'time' and 'customer'. For instance, the scale of 'time' could vary from monthly to every second. The scale of 'customer' could vary from national to individual.

2. Which of these strategies are useful for online retailers that sell non-perishable goods in plenty supply?

Some dynamic pricing strategies might actually be counterproductive. In the fall of 2000, Amazon experimented with dynamic pricing and started charging customers based on their perceived willingness to pay. However, this led to a storm of criticism and bad press and finally, a public statement from Amazon that they would stop using dynamic pricing [8]. To answer this sub question, we will suggest appropriate dynamic pricing strategies for this kind of web shop.

3. Can one or more appropriate strategies be implemented using the SOA paradigm?

Some dynamic pricing strategies may not be suitable for a service oriented architecture and must remain tailor made. Some of them might be highly suitable and relatively easy to implement. Using an appropriate modelling standard, we will design a SOA architecture for one or more of these strategies.

4. What, if any, problems would be caused by implementing this design into an existing legacy web shop and can these be overcome?

Without doing the actual implementation, we will analyse and evaluate the impact that implementing this design in an existing web shop could have. This will be a web shop that is build on an ERP system. This web shop is a good example of the kind of web shop that we are designing this service for, because most web shops of any significance will be using an ERP system or some other system closely resembling an ERP system. This is potentially problematic, because of the inherent complexity of such systems.

2. METHOD OF RESEARCH

Many scientific articles are available that describe an appropriate dynamic pricing model for a very specific situation. For instance, we found an article that specifically targets pricing strategies in a “two-tier multi-generation durable goods supply chain” [10]. Studying and categorizing such articles should provide insight into the input and output parameters of dynamic pricing algorithms, which will aid in answering sub question 1. To satisfy sub question 2, literature is needed on successful and unsuccessful implementations of dynamic pricing. Careful study of those should reveal the correlation between different types of e-commerce ventures and the appropriate dynamic pricing strategies.

Studying the work of prominent SOA author Thomas Erl will be necessary in order to be able to answer sub question 3. Most notably his book “SOA Design Patterns” should at least in part be studied. These, combined with alternative sources on the application of SOA and the research results of sub questions 1 and 2, should enable us to assess for some of the dynamic pricing strategies if and how they could be implemented in line with the SOA paradigm.

To be able to evaluate the research, we will conclude with a discussion of how the SOA architecture could be implemented into a case study web shop. For the case study an anonymised existing web shop will be used, which we will call “The Energy Saving Shop”. This web shop has been implemented as an extension to the ERP system Compiere and can be seen as exemplary for the type of legacy web shops that we want to provide this service for.

Another reason to choose this specific web shop is the author's involvement as lead developer for it. This will ensure that all
the necessary information about this specific implementation will be available at all times.

For the actual design we will follow the design methodology proposed by Peffers et al. [13] This methodology divides the design phase into six activities, as illustrated in Figure 1:

1. Problem identification and motivation
2. Definition of the objectives for a solution
3. Design and development
4. Demonstration
5. Evaluation
6. Communication

The entry point in this research is “Problem Centred Initiation”, because it is triggered by the need of web shops to employ flexible and affordable pricing functionalities.

For the actual modelling of the SOA implementation we will use the Service oriented architecture Modelling Language (SoaML) as specified by the Object Management Group [12].

3. LITERATURE REVIEW

Important work for our research can be divided into the following categories:

1. Service oriented architecture
2. Dynamic pricing algorithms.
3. Practical applications of dynamic pricing.

As far as we know, no attempt has been made to combine these fields into the research outlined in this paper.

3.1 Service Oriented Architecture

Perhaps somewhat hyped, SOA remains a very strong architectural concept. It is widely recognized as a technological step forward, for instance by Albrecht et al. [1]. The most notable SOA author is Thomas Erl. His definition of SOA is as follows:

Service oriented architecture represents an architectural model that aims to enhance the agility and cost-effectiveness of an enterprise while reducing the burden of IT on the overall organization. It accomplishes this by positioning services as the primary means through which solution logic is represented. [6]

According to Erl, there are eight important design principles that need to be accounted for when designing a service. These principles are illustrated in Figure 2. These principles will be explained and used to evaluate our design in section 5.4.

Furthermore, Erl defines four base characteristics that any SOA implementation should inhabit [6]:

1. Business-Driven: the technology is aligned with the business architecture.
2. Vendor-Neutral: different vendor technologies can be combined and replaced.
3. Enterprise-Centric: services are positioned as enterprise resources, which means that the logic is made available beyond specific implementation boundaries and designed in accordance with established design principles and enterprise standards.
4. Composition-Centric: it must be possible to pull services into a variety of composition designs, even if this was not an explicit requirement when they were designed.

3.2 Dynamic Pricing Algorithms

For our research, an interesting aspect of the service oriented architecture is that the consumer of a service does not need to worry themselves with the implementation of the service. It can be treated as a black box.

The algorithms to use for the actual implementation of the service fall out of the scope of this research. We merely concern ourselves with the feasibility of offering such a service, regardless of the exact algorithm to be used. To get some idea of the kinds of algorithms used, we did a short literature review and found an abundance of mathematically oriented papers on dynamic pricing algorithms.

For instance, Jia et al. [10] study the problem of dynamic pricing in a market where two specific conditions are met: the merchandise is durable and multi-generation, and the supply chain consists of one manufacturer and one retailer. This problem revolves around how to determine the best entry price for a new generation of the product and how to lower this price over time until it has become obsolete. Another example is Xiong et al. [17], who consider the problem of dynamic pricing for perishable products without using historic data.

A third interesting source is the dissertation by Den Boer on self-learning dynamic pricing algorithms [3]. One of his conclusions is that sometimes a trade-off must be made between short term and long term interests. An algorithm that always suggests the best price based on the current statistical estimates may negatively affect its learning curve. The reason for this lies in the fact that the process of learning from parameter values and the process of determining prices are two highly interdependent processes.

However, as mentioned, an in-depth study of these algorithms is not the main concern of this paper. What we do need to concern ourselves with, is the input and output that the algorithms should give. The output should be a price or a discount. The important input parameters seem to involve data...
on three entities: customer, product and time. The detail of the provided information is what determines the strategy used. In section 4.2 we further entertain this idea into a three-dimensional model for dynamic pricing strategies.

3.3 Practical Applications

What do we know about successful and unsuccessful applications of dynamic pricing? Garbarino et al. [8] show that in some situations it may be detrimental for your customer loyalty to apply dynamic pricing. For airline companies it is generally accepted that dynamic pricing is used, but for retailers this may be trickier. A perceived application of dynamic pricing by Amazon in the fall of 2000 led to a storm of criticism. In this case, Amazon was forced to issue a public statement in which they declared to stop using dynamic pricing.

The key difference between these two markets is that airlines deal with perishable goods in limited supply. It is easy to see that if only one ticket is left for a plane that departs next month, the airline company should ask a high price. Similarly, if there are 50 seats left and the plane departs tomorrow morning, it is clear that they should be offered at a low price. For online retailers that sell non-perishable goods in plenty supply, this is not clear at all. Therefore, in this market the consumer is likely to perceive dynamic pricing as unfair and applying it might negatively affect customer loyalty.

One way to mitigate this negative effect is proposed by Weisstein et al. [16]. By using different price framing methods the comparability of prices is decreased and with it, the perceived unfairness. For example, a price difference between € 90,- and € 85,- could be framed as a price difference between € 120,- with 25% off and € 85,- without a discount. According to the authors, this works because most people will not make the effort to figure out which of these is actually the better deal. However, this strategy seems unethical and is therefore probably not a viable long term solution.

The most important thing to learn from these studies is that a company should apply some caution when aiming to start a dynamic pricing project. If dynamic pricing cannot be made transparent without hurting customer loyalty, it should probably not be implemented at all.

4. CATEGORISATION

4.1 Strategic Model

Kannan et al. [11] provide a table that compares types of dynamic pricing of e-commerce for consumer goods and services. A summary of this table is included as Table 1.

Table 1: Dynamic pricing strategies. Modified from [11]

<table>
<thead>
<tr>
<th>Type of dynamic pricing</th>
<th>Transaction Time</th>
<th>Availability of price information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Posted Pricing</td>
<td>Minutes</td>
<td>Price bots available</td>
</tr>
<tr>
<td>Auction Pricing</td>
<td>Hours</td>
<td>Can be gathered with effort</td>
</tr>
<tr>
<td>Bundle Pricing</td>
<td>Minutes</td>
<td>Comparison is complicated</td>
</tr>
</tbody>
</table>

For our purposes, we are only concerned with the posted pricing model. The Auction model could not be used with a service that sets a price, because it is the customer that sets the price by bidding. The third type is also not feasible as a service, because bundling is an inherently customized marketing technique that is not readily implementable as an algorithm.

Kannan et al. [11] describe a suggestion for implementing dynamic posted prices. They can be made dynamic in two directions: time and customer. In an e-commerce environment the posted prices can be changed much more frequently than in conventional channels. However, these posted prices will still be the same for every customer. To differentiate between customers, Kannan et al. describe the use of e-coupons. Dependant on their behaviour, customers can be offered dynamic coupons which face discounts differ over time as well as across different customers.

This can be seen as an elegant implementation of the idea of price framing as described in section 3.3. The posted price is the same for everyone, but customers can get individual non-transparent discounts. This seems like a viable long term solution, unlike the more naïve implementation of price framing that is based on the perceived inability of most customers to figure out which is the better deal: €120,- with 25% off or €85,- without a discount.

4.2 Dimensional model

We propose the definition of dynamic posted prices as the combination of the following actions:

1. Applying an algorithm that takes contextual variables for a purchase into consideration, resulting in multiplying the base price with a certain factor.
2. Adjusting said base price.

The contextual variables can be understood in three dimensions:

Customer: Determines to what extent customer-specific input variables are used in the algorithm.

Product: Determines to what extent product-specific input variables are used in the algorithm.

Time: The frequency of changing the base price.

Applying this yields a cubic of dynamic pricing strategies as shown in Figure 3. The three scales express the amount of detail that is provided. By setting the low boundaries on Customer Group, Daily Changes and Product Category, we have implicitly defined dynamic pricing to be all strategies that take at least this amount of detail into consideration. So for instance all strategies that are based on weekly price updates or global customer precision (i.e. no customer differentiation at all), are considered to be outside of the spectrum that we call dynamic pricing.

As can be seen, we have defined four basic pricing strategies: precise, extreme, moderate and fast. All four of these do not explicitly assume a product-detail, so the analysis below will go for both product category precision and single product precision. Whether it is interesting to differentiate on the product level will be largely dependant on the number of products offered. When offering tens of thousands of products, it will probably be best to use product category-precision. When marketing only a few key products, it would be more obvious to use single product-precision.
When prices change every day at customer group level, we say that there is a moderate degree of dynamic pricing. It is possible to change them manually, but automating it might yield a big reduction in costs.

When prices change virtually constantly at customer group level, we call it ‘Fast Dynamic Pricing’. It is not possible to do this manually, and it heavily relies on robust high-capacity IT infrastructure. Applications for this strategy are markets with highly volatile pricing, in which it is not possible or desirable to differentiate between different kinds of customers.

When the base price is re-evaluated every day, but the algorithms applied to it take individual customer variables into consideration, we say there is ‘Precise Dynamic Pricing’. The base price may be changed manually, but the algorithms must be very sophisticated and automated. These algorithms basically try to compute the ‘willingness to pay’ of each individual customer.

Because of the ethical implications discussed in section 3.3, this method may negatively affect customer loyalty if not implemented with care. For a typical retailer of non-perishable goods in plenty supply, the best way of implementing will be the one as discussed in section 4.1, by using individual dynamic e-coupons. The e-coupon strategy may also be used to help getting access to detailed customer information. Customers could be persuaded to give more information about themselves in order to qualify for e-coupons.

Extreme dynamic pricing combines both fast and precise dynamic pricing. One of the major challenges for this strategy will be developing the appropriate tools for constantly monitoring and evaluating the pricing, because it might be very obscure how effective the pricing is when prices are neither stable at a certain point in time, nor for a particular customer.

4.3 SOA Feasibility

Let’s assess the SOA feasibility of each strategy in the three-dimensional model.

In the fast dynamic pricing strategy, the service provider is constantly sending actual prices for all products, which are not stored by the client, since they are volatile data.

In the precise dynamic pricing strategy, the client sends a request for every customer that accesses the web shop. The service provider then sends a discount list for this customer. This data is possibly stored, since it may be useful for the rest of the day, depending on what exact change frequency is used.

So we see that either strategy is possible. The extreme dynamic pricing strategy might also be possible, but this will be very challenging. In this strategy, the service provider will be constantly sending volatile price lists for every active customer on the website. Both the client and the server will need a very high quality infrastructure.

Finally, we have the moderate dynamic pricing strategy. This strategy will be most suited for a pilot of dynamic pricing. In many aspects it is actually similar to how a lot of web shops that do not explicitly employ dynamic pricing already operate.

For instance, the case study web shop that we will study in more depth later on already makes use of customer groups, based on the channel that the customer used to reach the web shop. Prices are also changed on a regular basis, albeit certainly not daily.

5. DESIGN

5.1 Problem Identification

Both dynamic pricing and service oriented architecture are highly researched fields. However, no research was found that attempts to combine these two. The reason to combine them is to make dynamic pricing available as an affordable option for small to medium-sized web shops that sell non-perishable goods in plenty supply.

5.2 Solution Objectives

The available resources in time for this research do not allow the development of a working artefact that demonstrates dynamic pricing in a service oriented architecture. The objective for this design is a model that lays the groundwork for further design research to actually implement and prove the concept.

5.3 Design

For this design we decided to use a standard by the Object Modeling Group, SoaML [12]. To get acquainted with how to use this in practice we studied two tutorials, one by IBM [2] and one by Visual Paradigm, a developer of a SoaML modelling tool [15]. This is also the tool we used to create the diagrams.
The model shown in Figure 4 is a Service Participant Diagram. It shows the participants at a high level and the services that must be provided.

The chain starts at the customer. In this model two services are depicted that the retailer should at least offer: an authentication service and a browse catalogue service. Please note that the customer only consumes services and does not provide any. This ensures that anyone can use this service – the person using the service does not have to be known by the provider.

Whenever a query on the product catalogue is requested by the customer, the retailer will request one or more prices from the dynamic pricing service provider. Similar to the customer, the retailer only requests services from the dynamic pricing service provider and does not offer any. This ensures that no services will have to be implemented on the retailer's side in order to be able to use this service.

Whenever an order is placed, the dynamic pricing service provider should be notified, so that it can take the historic data into consideration when computing future prices.

Figure 5 shows in more detail what the services entail. They are modelled as interface, because they are one-way services. This means that the service provider must offer these operations, but the service consumer does not have to implement any specific operations.

![Diagram](image-url)

**Figure 5: Dynamic Pricing Interfaces**

The key interface in this figure is “Dynamic Pricing”. The retailer offers information on the three dimensions and will be given a discount as output. Configurable aspects will be the amount of detail to be taken into consideration, and the minimum and maximum discount to be given. The exact algorithm used will be under constant evaluation by the service provider. Based on the historic data that they are fed they can constantly analyse and improve the algorithm.

### 5.4 Theoretical Evaluation

Earlier we saw the eight SOA design principles according to Thomas Erl (Figure 2). In this Section we will evaluate our design using these principles.

Standardized service contracts: this principle requires that the design is compliant with contract design standards for services in the same inventory. It is hard to evaluate this principle, because of the scale of our design. We can say that the challenge for this principle lies in how well we succeed in defining standards or using existing standards for product and customer properties and profiles.

Service loose coupling: according to this principle, the service should minimize consumer coupling requirements, and also keep themselves as decoupled as possible from their environment. In our design, the interfaces are independent of the implementation and the assumptions that consumer and provider make about each other are kept to a minimum. This allows for loose coupling.

Service abstraction: this principle is about hiding the underlying details of a service. This is what is done with the actual algorithm that is to be used: in our design it is considered to be unimportant what exactly it does. Only the required input and output is considered.

Service reusability: this principle requires that the service is reusable in different contexts. This is done by designing our service in such a way that any dynamic pricing strategy can use the same service because the input parameters are always the same. Only the amount of detail provided is different.

Service autonomy: this principle requires that the service can run independently from its environment. In our design, the quality of the service provided is dependent on how it is configured by the retailer, and on the accuracy of the historic data provided. Therefore this principle might be violated.

Service statelessness: this can be understood as the independence of separate invocations of the service. Even though the service improves over time based on historic data, this does not necessarily violate this principle. There is no reason to think that separate calls to the service would conflict with each other.

Service discoverability: this principle is about the discoverability of services that are positioned as IT assets with repeatable ROI [5]. Because the service described in this design is not such a service, this principle is not applicable.

Service composability: this is about the possibility to compose new services out of combinations of existing services. In our case, the pricing service may be embedded in a catalogue service or perhaps an even bigger composition of services.

We can conclude that we score well on most of the principles that are applicable. The only real concern is the autonomy. The quality of the service will be quite dependent on the quality of the configuration by the retailer and its historic feedback. This might be a real liability, because in a production situation, the service provider will be held accountable for the quality of the service.

### 5.5 Practical Evaluation

To demonstrate how this design could work in practice, we arbitrarily selected a Dutch web shop as a case study to apply the design to. This case study is not meant as a critical case, it does not attempt to prove the correctness of the design, but merely to demonstrate how it could be applied.

The anonymous case study web shop is called “The Energy Saving Shop” and features about 4000 products for which the prices are adjusted on a monthly to yearly basis. It divides its customers into different “Business Partner Groups”. These price lists may be used, which results in different prices as well as different range of products offered.

So their strategy can be called close to moderate. Prices change less frequently than daily, but customers are divided into business partner groups. However, the differentiation between business partner groups is not governed by algorithms, but merely a static reduction of the base price.

This web shop is implemented as an extension to the ERP system Compiere. This ERP system provides a data model for products, orders, prices and much more. It is possible to make modifications to this data model, but this is not recommended and should only be done when absolutely necessary, since this makes the system more costly to maintain.
While analysing the impact of dynamic pricing for this system, we came across two significant problems. In any ERP system, it is necessary to store historic prices for accounting reasons. For every order placed, it must be traceable what the current price for that product was for that customer.

We think that these are actually such big problems that it is probably not feasible to implement dynamic pricing as a service when such a data model is used in an underlying ERP system. On the other hand, it is possible to make customizations to the ERP system. The system used, Compiere, is even open source and the case study retailer actually has done quite some modifications to the system to fit their specific needs.

However, this defers from the initial goal of this research, namely to find a way to offer dynamic pricing as a reusable, readily implementable and cost effective service. It is also worth noting that the price model is an especially tedious part of an ERP system to customize, because it is central to the system and therefore there are a lot of dependencies. This makes it hard to predict all the problems that could come in to play when customizing this.

Let’s shortly discuss two ways to implement this requirement. One way is to have a model like Figure 6. In this model, every price is stored once, so it is effective in storage. However, it is very bad performance wise, because querying the current price is very costly when the table grows large: this query has to evaluate all prices and decide which “valid from date” is applicable for the current date.

For this reason a different design decision was made in Compiere. A simplified version of this model can be seen in Figure 7.

Let’s talk about some specific problems with this design.

One problem is that prices are not “live”. To effectuate a price change, the “create price list” process has to be run, either manually or through a scheduled task. To use our dynamic pricing service, we would have to find a way to import the discounts that the service outputs as a discount schema into Compiere. At the very least, this will be very tedious and detract from service oriented design.

Another problem is that prices are created for all products on a price list, even if only one product price changes. This in itself has dramatic consequences for the implementation of a dynamic pricing strategy. Even with daily price changes database capacity and performance will be thoroughly tested.

We think that these are actually such big problems that it is probably not feasible to implement dynamic pricing as a service when such a data model is used in an underlying ERP system. On the other hand, it is possible to make customizations to the ERP system. The system used, Compiere, is even open source and the case study retailer actually has done quite some modifications to the system to fit their specific needs.

However, this defers from the initial goal of this research, namely to find a way to offer dynamic pricing as a reusable, readily implementable and cost effective service. It is also worth noting that the price model is an especially tedious part of an ERP system to customize, because it is central to the system and therefore there are a lot of dependencies. This makes it hard to predict all the problems that could come in to play when customizing this.

### Figure 7: Product Price Compiere Data Model

The key entity in this price model is the “Price List Version”. The attribute “Valid From” is moved from “Product Price” to “Price List Version”. A price list version is created by running a process, which normally takes as input a historical price list and a new or updated discount schema. By creating a price list version per business partner group, you can differentiate between customers. This model scores much better performance wise, because all the price calculations are handled in the batch process of creating the price list version. However, there are some problems with this design.

One problem is that prices are not “live”. To effectuate a price change, the “create price list” process has to be run, either manually or through a scheduled task. To use our dynamic pricing service, we would have to find a way to import the discounts that the service outputs as a discount schema into Compiere. At the very least, this will be very tedious and detract from service oriented design.

Another problem is that prices are created for all products on a price list, even if only one product price changes. This in itself has dramatic consequences for the implementation of a dynamic pricing strategy. Even with daily price changes database capacity and performance will be thoroughly tested.

We think that these are actually such big problems that it is probably not feasible to implement dynamic pricing as a service when such a data model is used in an underlying ERP system. On the other hand, it is possible to make customizations to the ERP system. The system used, Compiere, is even open source and the case study retailer actually has done quite some modifications to the system to fit their specific needs.

However, this defers from the initial goal of this research, namely to find a way to offer dynamic pricing as a reusable, readily implementable and cost effective service. It is also worth noting that the price model is an especially tedious part of an ERP system to customize, because it is central to the system and therefore there are a lot of dependencies. This makes it hard to predict all the problems that could come in to play when customizing this.

### Figure 6: Product Price Simple Data Model

Let’s shortly discuss two ways to implement this requirement. One way is to have a model like Figure 6. In this model, every price is stored once, so it is effective in storage. However, it is very bad performance wise, because querying the current price is very costly when the table grows large: this query has to evaluate all prices and decide which “valid from date” is applicable for the current date.

For this reason a different design decision was made in Compiere. A simplified version of this model can be seen in Figure 7.

Let’s talk about some specific problems with this design.

One problem is that prices are not “live”. To effectuate a price change, the “create price list” process has to be run, either manually or through a scheduled task. To use our dynamic pricing service, we would have to find a way to import the discounts that the service outputs as a discount schema into Compiere. At the very least, this will be very tedious and detract from service oriented design.

Another problem is that prices are created for all products on a price list, even if only one product price changes. This in itself has dramatic consequences for the implementation of a dynamic pricing strategy. Even with daily price changes database capacity and performance will be thoroughly tested.

We think that these are actually such big problems that it is probably not feasible to implement dynamic pricing as a service when such a data model is used in an underlying ERP system. On the other hand, it is possible to make customizations to the ERP system. The system used, Compiere, is even open source and the case study retailer actually has done quite some modifications to the system to fit their specific needs.

However, this defers from the initial goal of this research, namely to find a way to offer dynamic pricing as a reusable, readily implementable and cost effective service. It is also worth noting that the price model is an especially tedious part of an ERP system to customize, because it is central to the system and therefore there are a lot of dependencies. This makes it hard to predict all the problems that could come in to play when customizing this.

### 6. CONCLUSION

The second sub question was: Which of these strategies are useful for online retailers that sell non-perishable goods in plenty supply? We saw that most of these retailers will already be close to an implementation of the moderate strategy. They are known to already distinguish between customer groups and will change their prices on a regular basis. Another viable strategy is the precise strategy, but only if the danger of negatively affecting customer loyalty can be properly mitigated.

The third sub question was: Can one or more appropriate strategies be implemented using the SOA paradigm? Using the modelling standard SoaML, we showed that each of the strategies discussed can be designed in a service oriented architecture. The architecture at the highest level does not change when the strategy changes, only the level of detail that is to be provided and taken into consideration by the algorithms. This will also have an impact on the IT infrastructure needed for both client and server, which has to be able to support constant traffic and computations when the extreme strategy would be employed.

The fourth sub question was: What, if any, problems would be caused by implementing this design into an existing legacy web shop and can these be overcome? We have seen that our case study legacy web shop has a major issue to overcome with the price model that its ERP system is using. We believe that an alternative way of keeping track of historic prices will have to be designed. Even though it is possible to make customizations to the ERP system, the price list model is such a vital and integral part of it, that it is highly questionable if an implementation of dynamic pricing is even feasible at all at this point.

That brings us to our main research question: Can dynamic pricing functionality be offered to small to medium sized web shops, using the SOA paradigm? The short answer is yes. It is feasible to design a service that takes contextual variables as
input and outputs a price or a discount. However, we found that for at least one case study web shop that was built as an extension to an ERP system, this service would not only lay a heavy burden on the underlying database, but would also require costly and high-risk customizations. We suspect that similar problems will also occur with other ERP systems. Because we believe that most of the small to medium sized web retailers have a web shop that is linked to an ERP system, we cannot confidently state that this is possible, unless the specific ERP system has found a way to make their price data model flexible.

7. FUTURE WORK
More research should be done into the effects on customer loyalty of dynamic pricing for e-commerce retailers of non-perishable goods in plenty supply. It should be further researched if price framing does indeed help maintaining customer loyalty. In this research we mentioned e-coupons, but there are probably more ways in which price framing can be employed. Another thing that should be further researched are the ethical aspects of dynamic pricing. If dynamic pricing in this market is deemed unethical, it is questionable if it should even be pursued at all.

Another important field for future research is the data model for product prices. It should be researched if such a design lends itself better to highly dynamic pricing and whether such a design is already in production. Perhaps the traceability of historic prices could even be implemented as a service as well. This might very well be the most ideal solution.

Finally, it would be useful to build an actual prototype for the service described in this paper. This could confirm the concerns about the data model and perhaps shed light on some other interesting challenges that arise by implementing such a service.

8. REFERENCES