An Autonomous Agent Application for Product Pricing

Nikolaos Spanoudakis[1,2], Pavlos Moraitis[2]

[1]Technical University of Crete, Department of Sciences,
University Campus, 73100 Chania, Greece
nikos@science.tuc.gr
[2]Paris Descartes University, Department of Mathematics and Computer Science,
45 rue des Saints-Pères, 75270 Paris Cedex 06, France
pavlos@mi.parisdescartes.fr

Abstract. This paper describes an argumentation-based approach for automating the decision making process of an autonomous agent for pricing products. Product pricing usually involves different decision makers with different - possibly conflicting - points of view. Moreover, when considering firms in the retail business sector, they have hundreds or thousands of products to apply a pricing policy. Our approach allows for applying a price policy to each one of them by taking into account different points of view expressed through different arguments and the dynamic environment of the application. This is done because argumentation is a reasoning mechanism based on the construction and the evaluation of interacting conflicting arguments.

Keywords: Argumentation, decision making, product pricing agent

1. Introduction

Automating the product pricing procedure in many different types of enterprises like retail businesses, factories, even firms offering services is an important issue. Product pricing is concerned with deciding on which price each of a firm's products will have in the market.

The product pricing agent that we present in this paper allows for the integration of the views of different types of decision makers (like financial, production, marketing officers) and can reach a decision even when these views are conflicting. This is achieved with the use of argumentation.

The information that is available to our agent is the firm strategy, the available products and their initial prices in the form of production costs, procurement costs or costs in the market – obtained by one or more competing firm(s).

Argumentation has been used successfully in the last years as a reasoning mechanism for autonomous agents in different situations, as for example for deliberating over the needs of a user with a combination of impairments [9] and for selecting the funds that should be included in an investment portfolio [10]. It is the

first time that it is used for decision making in the retail business sector and our work presented here-in aims to show that it can be applied successfully in an area that sparse works provide solutions. One such [1] only proposes an architecture and not the pricing mechanism itself. Important works in seller and buyer agents (see [7], [3] and [4]) provided us with important information regarding the challenges that we faced, but with little practical advice since they focus in modeling markets where sellers compete with others in order to provide the smallest price to the consumer. However, the retail business sector and our requirements demanded a system that would have the possibility to just apply a pricing policy adjusted to the market context and reflecting the points of views of diverse decision makers.

This product pricing agent was developed in the context of MARKET-MINER project that was co-funded by the Greek government. SingularLogic SA, a leading Greek software vendor in the area of business software participated in the project. Their consultants and system administrators evaluated our results and considered our agent a success that could have an important impact in the business intelligence software suite in the next four to five years.

In what follows we firstly present the basics of the used argumentation framework in section two and then we discuss how we modeled the knowledge of the particular application domain in section three. We continue with presenting the final product pricing agent in section four, followed by the presentation of the evaluation results in section five. Subsequently, we discuss related work in section six and we conclude in section seven.

2. The Theoretical Framework

Decision makers, be they artificial or human, need to make decisions under complex preference policies that take into account different factors. In general, these policies have a dynamic nature and are influenced by the particular state of the environment in which the agent finds himself. The agent's decision process needs to be able to synthesize together different aspects of his preference policy and to adapt to new input from the current environment. We model the product pricing decision maker as such an agent.

In order to address requirements like the above, Kakas and Moraitis [5] proposed an argumentation based framework to support an agent's self deliberation process for drawing conclusions under a given policy. This is the framework that we adopted.

The following definitions present the basic elements of this framework:

Definition 1. A **theory** is a pair $(\mathcal{T}, \mathcal{P})$ whose sentences are formulae in the **background monotonic logic** $(\mathcal{L}, \vdash)$ of the form $L \leftarrow L_1,\ldots,L_n$, where L, L_1, …, L_n are positive or negative ground literals. For rules in P the head L refers to an (irreflexive) higher priority relation, i.e. L has the general form $L = h\_p(\text{rule1}, \text{rule2})$. The derivability relation, $\vdash$, of the background logic is given by the simple inference rule of modus ponens.

An **argument** for a literal L in a theory $(\mathcal{T}, \mathcal{P})$ is any subset, T, of this theory that derives L, $T \vdash L$, under the background logic. A part of the theory $\mathcal{T}_0 \subset \mathcal{T}$, is the **background theory** that is considered as a non defeasible part (the indisputable facts).

Definition 2. Let $(\mathcal{T}, \mathcal{P})$ be a theory, T, $T' \subseteq \mathcal{T}$ and P, $P' \subseteq \mathcal{P}$. Then (T', P') attacks (T, P) iff there exists a literal L, $T_1 \subseteq T'$, $T_2 \subseteq T$, $P_1 \subseteq P'$ and $P_2 \subseteq P$ s.t.:

$T_1 \cup P_1 \vdash_{\min} L$ and $T_2 \cup P_2 \vdash_{\min} \neg L$

$(\exists r' \in T_1 \cup P_1, r \in T_2 \cup P_2$ s.t. $T \cup P \vdash h\_p(r, r')) \Rightarrow (\exists r' \in T_1 \cup P_1, r \in T_2 \cup P_2$ s.t. $T' \cup P' \vdash h\_p(r', r))$.

Simply said, this definition states that an argument attacks (or is a counter argument of) another when they derive a contrary conclusion.

Definition 3. Let $(\mathcal{T}, \mathcal{P})$ be a theory, $T \subseteq \mathcal{T}$ and $P \subseteq \mathcal{P}$. Then (T, P) is **admissible** iff $(T \cup P)$ is consistent and for any $(T' \cup P')$ if $(T' \cup P')$ attacks $(T \cup P)$ then $(T \cup P)$ attacks $(T' \cup P')$. Given a ground literal L then L is a **credulous (resp. skeptical) consequence** of the theory iff L holds in a (resp. every) maximal (wrt set inclusion) admissible subset of $\mathcal{T}$.

Therefore, an argument (from $\mathcal{T}$) is admissible if it takes along priority arguments (from $\mathcal{P}$) and makes itself at least as strong as its counter-arguments.

Definition 4. An agent's **argumentative policy theory or theory**, T, is a tuple $T = (\mathcal{T}, \mathcal{P}_R, \mathcal{P}_C)$ where the rules in $\mathcal{T}$ do not refer to h_p, all the rules in $\mathcal{P}_R$ are priority rules with head $h\_p(r_1, r_2)$ s.t. $r_1, r_2 \in \mathcal{T}$ and all rules in $\mathcal{P}_C$ are priority rules with head $h\_p(R_1, R_2)$ s.t. $R_1, R_2 \in \mathcal{P}_R \cup \mathcal{P}_C$.

Thus, in defining the decision maker's theory three levels are used. The first level ($\mathcal{T}$) defines the (background theory) rules that refer directly to the subject domain, called the *Object-level Decision Rules*. In the second level we have the rules that define priorities over the first level rules for each *role* that the agent can assume or *context* that he can be in (including a *default context*). Finally, the third level rules define priorities over the rules of the previous level (which context is more important) but also over the rules of this level in order to define *specific contexts*, where priorities change again.

Gorgias[1], a prolog implementation of the framework presented above, defines a specific language for the object level rules and the priorities rules of the second and third levels. A negative literal is a term of the form *neg(L)*. The language for representing the theories is given by rules with the syntax:

```
rule(Signature, Head, Body).
```

[1] Gorgias is an open source general argumentation framework that combines the ideas of preference reasoning and abduction, http://www.cs.ucy.ac.cy/~nkd/gorgias/

In the above rule, `Head` is a literal, `Body` is a list of literals and `Signature` is a compound term composed of the rule name with selected variables from the `Head` and `Body` of the rule. The predicate *prefer/2* is used to capture the higher priority relation (*h_p*) defined in the theoretical framework. It should only be used as the head of a rule. Using the previously defined syntax we can write the rule

```
rule(Signature, prefer(Sig1, Sig2), Body).
```

The above rule means that the rule with signature `Sig1` has higher priority than the rule with signature `Sig2`, provided that the preconditions in the `Body` hold. If the modeler needs to express that two predicates are conflicting he can express that by using the rule:

```
conflict(Sig1,Sig2).
```

The above rule indicates that the rules with signatures `Sig1` and `Sig2` are conflicting. A literal's negation is considered by default as conflicting with the literal itself.

3. Domain Knowledge Modeling

Firstly, we gathered the domain knowledge in free text format by questioning the decision makers that participate in the product pricing procedure. They were officers in Financial, Marketing and Production departments of firms in the retail business but also in the manufacture domain.

Then, we processed their statements aiming on one hand to discover the domain ontology and on the other hand the decision making rules. For example, let's consider the expression "If the firm has a high-low strategy then if it advertises a product and its price is low the products that accompany it in the consumers' basket are priced high". This expression identifies the concepts "firm strategy" and "product". The concept firm strategy can have the property "high-low" and the product can have the property "price" and can be related to other products as "accompanied in the consumer's basket by" them.

The next step was to ask a team of decision makers to decide on priorities between the different conflicting extracted rules. These priorities could be default or dependent on context.

The knowledge representation process is detailed in the following paragraphs, firstly, the ontology definition part and, secondly, the knowledge base development. However, the process was not sequential; it was rather iterative as the concepts proved to need to evolve as the knowledge base was developed.

3.1. Ontology definition

We used the Protégé[2] open source ontology editor for defining the domain concepts and their properties and relations. Even though we aimed to do reasoning using a logic programming language we decided to use a standardized way to represent our ontology. The main reasons were that the human-machine interface of our agent and its interface to other components (like a database) would be done using the Java object-oriented language. The Protégé tool allows – through the use of the beangenerator[3] add-on – to export the ontology in Java class format.

In Figure 1, the *Product* concept and its properties are presented. The reader can see the properties identified in the previous paragraph *hasPrice* and *isAccompaniedBy*. Price is defined as a real number (*Float*) and *isAccompaniedBy* relates the product to multiple other instances of products that accompany it in the consumer's cart.

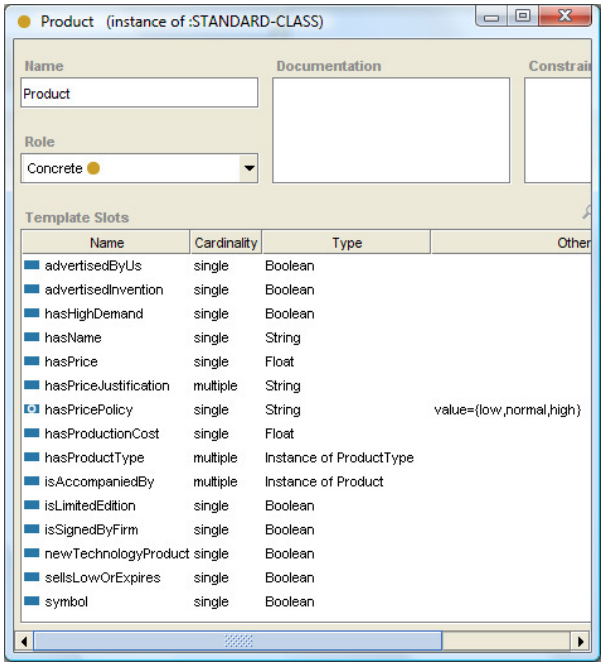

Fig. 1. The Product ontology concept.

In Figure 2 we present the firm strategy concept and its properties. As the reader can observe they are all *Boolean* and represent the different strategies that the firm

---

2 Protégé is a free, open source ontology editor and knowledge-base framework, http://protege.stanford.edu/

3 The ontology bean generator plug-in for Protégé generates java files representing an ontology, http://protege.stanford.edu/

can have activated at a given time. For example, the *hitCompetition* property is set to *true* if the firm's strategy is to reduce the sales of its competitors. Only the last property (i.e. *retail_business*) does not refer to strategy, it just characterizes the firm as one in the retail business sector.

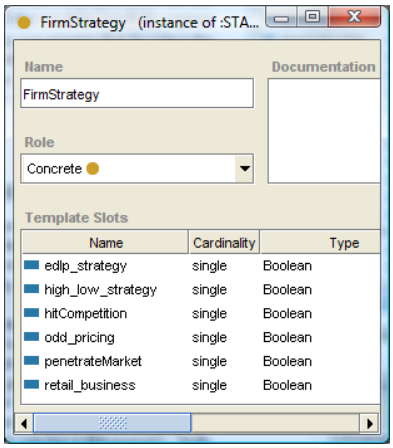

Fig. 2. The FirmStrategy ontology concept

3.2. Knowledge Base Definition

For our knowledge base definition we used Prolog. In order to use the concepts and their properties as they were defined in Protégé we did the following encodings:

- A Boolean property is encoded as a unary function, for example the *advertisedByUs* property of the *Product* concept (see Figure 1) is encoded as *advertisedByUs(ProductInstance)*
- A property with a string, numerical, or any concept instance value is encoded as a binary predicate, for example the *hasPrice* property of the *Product* concept (see Figure 1) is encoded as *hasPrice(ProductInstance, FloatValue)*
- A property with a string, numerical, or any concept instance value with multiple cardinality is encoded as a binary predicate. However the encoding of the property to predicate can be done in two ways. The first possibility is for the second term of the predicate to be a list. Thus, the *isAccompaniedBy* property of the *Product* concept (see Figure 1) is encoded as *isAccompaniedBy(ProductInstance, [ProductInstance1, ProductInstance2, …])*, where product instances must not refer to the same product. A second possibility is to create multiple predicates for the property. For example the *hasProductType* property of the Product concept is encoded as *hasProductType(ProductInstance, ProductTypeInstance)*. In the case that a product has more than one product types, one such predicate is created for each product type.

Then, we used the Gorgias framework for writing the rules. The goal of the knowledge base would be to decide on whether a product should be priced high, low or normally. Thus it emerged, the *hasPricePolicy* property of the *Product* concept. After this decision we could write the object-level rules each having as head the predicate *hasPricePolicy(Product, Value)* where *Value* can be *low*, *high* or *normal* – the relevant limitation for this predicate is also defined in the ontology (see the *hasPricePolicy* property of the *Product* concept in Figure 1). Then, we defined the different policies as conflicting, thus only one policy was acceptable per product. In order to resolve conflicts we consulted with the firm (executive) officers and defined priorities over the conflicting object rules. For our knowledge base we assume that the closed world assumption holds.

Figure 3 shows two object level rules. We use the special rule format of Gorgias and variables start with a capital letter as it is in Prolog. Rules *r1_2_2* and *r2_3* are conflicting if they are both activated for the same product. The first states that a product should be priced low if the firm wants to hit the competition for its product type, while the second states that a new technology product that is also an advertised invention should be priced high. To resolve the conflict we add the *pr1_2_6* priority rule which states that *r1_2_2* is preferable to *r2_3*.

```
#object level rules
…
rule(r1_2_2(Product), hasPricePolicy(Product, low),
          [hitProductTypeCompetition(ProductType),
          hasProductType(Product, ProductType)]).
…
rule(r2_3(Product), hasPricePolicy(Product, high),
          [newTechnologyProduct(Product),
          advertisedInvention(Product)]).
…
#priority rules
…
rule(pr1_2_6(Product), prefer(r1_2_2(Product),
          r2_3(Product)), []).
…
```

Fig. 3. An extract from the rule base.

4. The Product Pricing Agent

The most challenging part of the agent, on which we focus in this section, is the decision making capability, which involves Prolog, Java and argumentation technologies. We have already presented the use of argumentation and prolog. The rule base includes 274 rules 31 of which are the object rules and 243 are priority rules.

The rule base models the points of view of all related decision makers and strategies for handling conflicting rules. The outcome of the argumentation based decision making process is the pricing policy for any given product. This is the main outcome of the process. Then, a pricing algorithm grounds the policy to a fixed price for each product. Having described the argumentation part of the decision making process, in this section we focus in presenting the pricing algorithm and the human-machine interface.

Java was used for creating the human-machine interface and the pricing algorithm. The aim of this algorithm is to produce a final price for each product. The algorithm's inputs are:

1. the procurement/manufacture cost for a product, or its price in the market,
2. the outcome of the reasoning process (the price policy for each product)
3. the default profit ratio for the firm
4. a step for rising the default profit ratio
5. a step for lowering this ratio
6. the lowest profit ratio that the firm would accept for any product

The pricing algorithm also takes into account the number of arguments that are admissible for choosing a specific price policy, strengthening the application of the policy. This does not hold for normal pricing, the product price in this case is:

$$P = C * (1 + R) . \quad (1)$$

In the above formula, *P* is the product price, *C* is the procurement or manufacture cost and *R* is the default profit ratio for the firm.

If the policy is determined as high then:

$$P = C * (1 + R + m * H) . \quad (2)$$

In the above formula, *m* is the number of admissible arguments for applying a high price policy, i.e. those with head *hasPricePolicy(Product, high)*, and *H* is the defined step by which the firm expands its profit ratio.

If the pricing policy is determined as low then the profit is lowered by a step per supporting argument. However, in this case the firm can set a threshold defining the lowest profit margin that it would accept:

$$\begin{array}{c} \text{If } n* L > D \\ \text{Then } P = C * (1 + R - D) \\ \text{Else } P = C * (1 + R – n * L) . \end{array} \quad (3)$$

In the above formula, *n* is the number of admissible arguments for applying a low price policy, *L* is the defined step by which the firm limits its profit ratio and *D* is the lowest profit ratio that the firm would accept.

If the product's price is not based on procurement or production costs but to its actual price in the market (e.g. the price that it has in a competitive firm), then in

formulas 1, 2 and 3 the *R* term is eliminated and the term *C* refers to the product's price in the competition. For example, a product with a normal price policy will have the same price with the one it has in a competitor's store.

Finally, the algorithm allows for the application of odd-pricing, a strategy for pricing products where the price's last digit is always 5, 8, or 9.

A screenshot from the human-machine interface is presented in Figure 4. In the figure we present the pricing results to the application user for some sample products. The facts inserted to our rule base for this instance are presented in Figure 5. The reader should notice the application of the rules presented in Figure 3 for the *lcd_tv_32_inches* product that is a new technology product and an advertised invention but is priced with a low policy because its product type (*electrical_domestic_appliances*) has been marked by the firm as a market where it should hit competition. Moreover, the firm also has decided that it wants to penetrate the *electrical_domestic_appliances* market, therefore there are two arguments for pricing the *lcd_tv_32_inches* product low. In Figure 4, these reasons are explained to the user in human-readable format and also the final price is computed. The human-readable format is generated automatically by having default associations of the predicates to free text.

The *t_shirt_XXL* and *jacket_XXL* products are clothes that are having a normal pricing policy. However, the *jacket_XXL* product accompanies in the consumer's basket the *lcd_tv_32_inches* product and the latter is both priced low and advertised by the firm. Therefore, the *jacket_XXL* product is priced high according to the *high_low_strategy* of the firm.

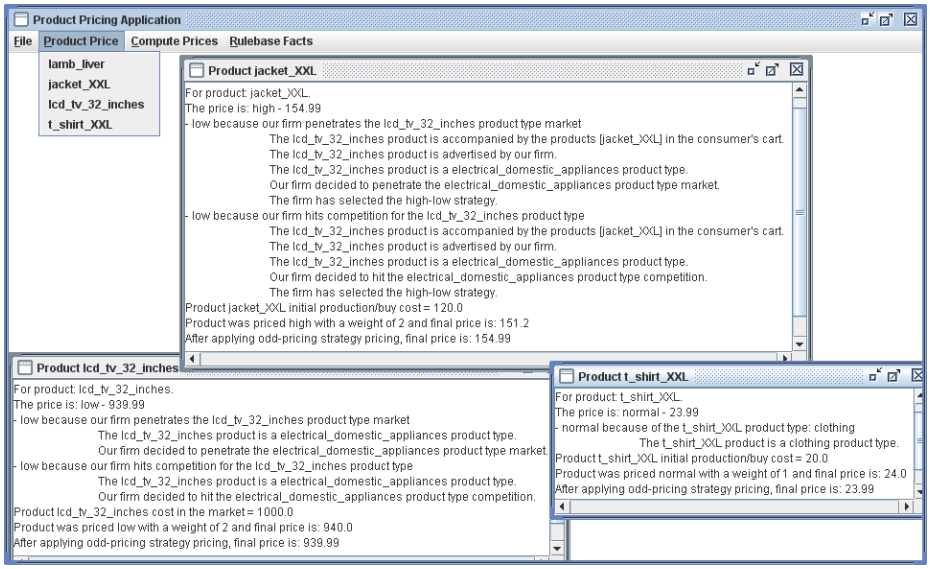

Fig. 4. The Product Pricing Agent Application

```
rule(f1, high_low_strategy, []).
rule(f2, retail_business, []).
rule(f3, hasSuppliers(food, 1), []).
rule(f4, isSensitiveProductType(food), []).
rule(f5, hitProductTypeCompetition(
         electrical_domestic_appliances), []).
rule(f6, penetrateProductTypeMarket(
         electrical_domestic_appliances), []).
rule(f7, hasProductType(lamb_liver, food), []).
rule(f8, hasProductType(jacket_XXL, clothing), []).
rule(f9, advertisedByUs(lcd_tv_32_inches), []).
rule(f10, advertisedInvention(lcd_tv_32_inches), []).
rule(f11, newTechnologyProduct(lcd_tv_32_inches), []).
rule(f12, isAccompaniedBy(lcd_tv_32_inches,
          [jacket_XXL]), []).
rule(f13, hasProductType(lcd_tv_32_inches,
          electrical_domestic_appliances), []).
rule(f14, hasProductType(t_shirt_XXL, clothing), []).
```

Fig. 5. A set of sample facts

5. Evaluation

The product pricing agent application was evaluated by SingularLogic SA, the largest Greek software vendor for SMEs. The Software business unit is involved in the development and provision of business software products for the SME market, the provision of services (implementation and adaptation of applications, training and maintenance services), as well as the promotion and support of products by third parties, both in the entirety of the Greek market and the Balkan markets. The unit's software applications are trusted by 40,000 businesses both in Greece and abroad.

The MARKET-MINER project included the application analysis, design, implementation and evaluation phases. It also produced an exploitation plan [11]. The application evaluation goals were to measure the overall satisfaction of its users. In the evaluation report [12] three user categories were identified, System Administrators, Consultants and Data Analysts.

At this point the reader should note that the MARKET-MINER project had a wider scope than that of our application, therefore we will focus in the part of the study relevant to it - the pricing application. Thus, only the Consultants and System Administrators user categories are relevant (data analysts were engaged in the data mining module of MARKET-MINER that is beyond the scope of this paper).

The following criteria were used for measuring user satisfaction:

1. *Performance* (C1): This criterion measures the capability of the system to produce valid and accurate results.
2. *Usability* (C2): This criterion measures the satisfaction of the user with regard to his experience in using the system, including the training phase and the ease of achieving his tasks.
3. *Interoperability* (C3): MARKET-MINER depends heavily on its seamless integration with legacy systems databases. Thus we needed to measure the openness of the system or the efficiency of connecting it to the existing databases.
4. *Security and Trust* (C4): Market Miner accesses enterprise databases and handles sensitive information relevant to the firm's market strategy. Thus, it is important that the user feels that the data are securely handled and remain confidential.

The users expressed their views in a relevant questionnaire where each criterion was presented with several sub-criteria and they marked their experience on a scale of one (dissatisfied) to five (completely satisfied) and their evaluation of the importance of the criterion on a scale of one (irrelevant) to five (very important). The evaluation was based on 25 questionnaires, 15 of which were completed by decision makers (with financial background), seven by data analysts (computer science background) and three by system administrators.

The consultants were experienced in applying business intelligence solutions to enterprises mostly in the retail sector. The retail sector was identified as the most important for the project's exploitation by the exploitation strategy report. They evaluated the system with regard to all the criteria. The system administrators were experienced in setting up and maintaining information systems in the business software sector. They evaluated the system only with regard to the criteria C3 and C4. Also, experienced independent scientists in the economic (as consultants) and computer science (as system administrators) fields working at another MARKET-MINER project partner (Informatics and Telematics Institute, Greece) evaluated the application for the same criteria.

The Process of Evaluation of Software Products [2] (MEDE-PROS) was used for evaluating our system. MEDE-PROS is in use for over 15 years, continually evolving and it has been applied to more than 360 software products.

The results of the evaluation of the MARKET-MINER software prototype are presented in Table 1 and they have been characterized as "very satisfactory" by the SingularLogic research and development software assessment unit. MARKET-MINER has been decreed as worthy for recommendation for commercialization and addition to the Firm's software products suite.

6. Related Work

In the agent community literature product pricing agents have been referred to as economic agents, as price bots, or, simply, as seller agents (see e.g. [7], [3] and [4]) and their responsibility is to adjust prices automatically on the seller's behalf in response to changing market conditions [7].

Real world agent-based systems are mostly consumer oriented solutions seeking information on the internet and comparing prices for their owners acting as recommender systems. However, the Book.com online book-selling store has been reported to adjust its price so that it is a little lower than other well-known book-sellers like Amazon.com [7]. In traditional markets (like the one we are targeting with market-miner) it is difficult to continuously re-price goods, as there is a costly (both in time and resources) procedure for updating prices on the self, in contrast to digital markets there is no real cost to changing prices [4].

Table 1. MARKET-MINER evaluation results. The rows with white background are those of the consultants, while those with grey background represent the evaluation of the system administrators.

Criterion	Criterion performance	Criterion Importance
C1	86%	0,78
C2	83%	0,88
C3	91%	0,88
C4	83%	0,64
C3	86%	0,92
C4	61%	0,92

In [7], the authors provide results for a wide range of possibilities for seller agents. They consider three types of seller agents:

1. Game-theoretic computation agents (GT). These agents chooses a random price from a distribution function whose inputs are a) the number of alternatives that each buyer will consider before deciding which product to buy, b) the buyer's maximum price and c) the number of seller agents.
2. Myopically optimal agents (MY). These agents choose product prices using the information needed by the GT agents and the current prices of all sellers aiming to maximize their profit in the short term.
3. Derivative Follower agent (DF). DF does not need any information about the buyers or competitors, ha adjusts his prices according to their success in the market.

All these types of agents have different success rates depending on the competitors' types.

In [4], the authors propose a theoretical framework for selling a specific type of good (i.e. baseball tickets) introducing the notion of using different strategies based on the market condition. Such conditions are the increase or decrease of consumer demand throughout the market season. Their first strategy aims to sell all the tickets at the highest possible rate and have them available until the last day of the market, while their second strategy is similar to the DF strategy in [7].

In another work ([3]), the authors address the problem of product pricing in environments with limited information. They set the agent's goal to reset the product price at regular intervals. In all these works, seller agents that compete with their counterparts engage in cycles of price wars where prices decrease until they reach a

limit where they reach a minimum utility and then decide to raise prices and start over.

All these existing solutions focus on a selected product negotiation rather than bundles of products (as in the retail business sector). The MARKET-MINER product pricing agent borrows interesting features from these works, i.e. resets prices at regular intervals and can employ different strategies for pricing depending on market conditions. The added value of the MARKET-MINER product pricing agent regarding these approaches is the capability to model human knowledge and apply human-generated strategies to automate product pricing with the possibility to provide logical explanations to decision makers, if needed.

Conclusion and future perspectives

This paper presented a novel application of autonomous agents for automating the product pricing process. This issue has never been tackled before in this scale. A patent just provided some guidelines on an architecture for such a system exclusively for super market chains [1]. Earlier works proposed a support of the product pricing process for the retail business sector but did not provide an automated decision mechanism [8]. In this paper we used argumentation that allows for expressing conflicting views on the subject and a mechanism for resolving these conflicts. Moreover, with argumentation it is possible to provide an explanation of the decisions that the agent makes. This is also the main technical difference with existing works in the agent technology literature (see e.g. [7], [3], [4]).

This application can be used with different facts in different branches of a firm, thus it can have a different policy in a geographic area where the competition is high (lowering prices) and different in an area where there is no competition (pricing high its products).

The decision maker can simulate the resulting prices if his firm adopts a specific strategy and determine the profit margins that will come after such a move. MARKET-MINER can apply the existing pricing policies in regular intervals adjusting it to the current market condition.

It can be used both for the retail and production business sectors. It allows for applying pricing policies according to the competitors' prices or the profit margin defined by the company decision makers.

This application puts in evidence autonomous agent technology added value, because our agent is situated in the firm environment and proactively monitors the internet for changes that would have an impact in the pricing process (e.g. a competitor changes his prices). Products are priced individually and daily. Products that are near their expiration date can be priced low in a day to day basis. Finally, the resulting prices can be justified to the firm officers in plain text format understandable by humans who can always have a final say in each decision. In this paper, we modeled, inside an agent, different policies for product pricing within a company. However, we can have several such agents interacting in a market upon the price of a

product. This kind of interaction can be modeled using the same argumentation framework that we use in this paper (as in [6]).

The presented application is part of a research project, MARKET-MINER co-funded by the Greek government. Its results (including the pricing application) were evaluated according to a widely used process (MEDE-PROS [2]) and they were proposed by the SingularLogic research and development department for commercialization by the firm.

References

1. C. Charles and Jr. Freeny, "Automated Synchronous Product Pricing and Advertising", United States Patent 6076071, 2000.
2. R. Colombo and A. Guerra, "The Evaluation Method for Software Product", Proc. of the 15th Int. Conf. on Software & Systems Engineering & Applications (ICSSEA 2002), Paris, France, December 3-4, 2002.
3. P. Dasgupta and S. Das. Dynamic pricing with limited competitor information in a multi-agent economy. In O. Eztion and P. Scheuermann, editors, Cooperative Information Systems: 7th International Conference, volume 1906 of Lecture Notes in Computer Science, pages 291-310, Eilat, Israel, September 2000. BoopIs, Springer.
4. J. M. DiMicco, A. Greenwald, and P. Maes, "Dynamic pricing strategies under a finite time horizon", Proc. of ACM Conf on Electronic Commerce (EC'01), October 2001.
5. A. Kakas, and P. Moraitis, "Argumentation based decision making for autonomous agents", Proc. of the second Int. Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS03), Melbourne, Australia, July 14-18, 2003.
6. A. Kakas, and P. Moraitis, "Adaptive Agent Negotiation via Argumentation", Proc. Of the fifth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'06), Hakodate, Japan, 2006, pp. 384-391.
7. J. O. Kephart, J. E. Hanson, and A. R. Greenwald. Dynamic pricing by software agents. Computer Networks, 36(6):731-752, May 2000.
8. N. Matsatsinis, P. Moraitis, V. Psomatakis, N. Spanoudakis, "An Agent-Based System for Products Penetration Strategy Selection", Applied Artificial Intelligence Journal, Taylor & Francis, Vol. 17, No. 10, 2003, pp. 901-925.
9. P. Moraitis and N. Spanoudakis, "Argumentation-based Agent Interaction in an Ambient Intelligence Context", IEEE Intelligent Systems, Special Issue on Argumentation Technology, Vol. 22, No. 6, November-December 2007, pp. 84-93.
10. N. Spanoudakis and K. Pendaraki, "A Tool for Portfolio Generation Using an Argumentation Based Decision Making Framework", Proceedings of the annual IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2007), Patras, Greece, October 29-31, 2007.
11. P. Toulis, D. Tzovaras, N. Spanoudakis, "MARKET-MINER Project Exploitation Plan", MARKET-MINER Project, Deliverable Π6.1 (in Greek language), Singular Logic S.A., Greece, February 2007.
12. P. Toulis, D. Tzovaras, S. Pantelopoulos, "MARKET-MINER System Evaluation Report", MARKET-MINER Project, Deliverable Π5.1 (in Greek language), Singular Logic S.A., Greece, February 2007.