Automated Product Pricing Using Argumentation

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

[1]Department of Sciences - Technical University of Crete,
nikos@science.tuc.gr

[2]Department of Mathematics and Computer Science – Paris Descartes University,
{pavlos, nikolaos.spanoudakis}@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. We also show how we conceived and developed our agent using the Agent Systems Engineering Methodology (ASEME).

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.

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 [8] and for selecting the funds that should be included in an investment portfolio [11]. It is the first time that it is used for decision making in the retail business sector. This paper aims to show that argumentation can be applied successfully in an area that sparse works provide solutions, the retail business sector. Argumentation responded well to our requirements, which demanded a system that would have the possibility to apply a pricing policy adjusted to the market context, in the meanwhile 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. After evaluation, its results have been considered to be successful and are expected to have an important impact in the firm's 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 2 and then, in section 3, we discuss how we modeled the knowledge of the particular application domain. Subsequently, we present the product pricing agent, including information on how we conceived and modeled the system using the Agent Systems Engineering Methodology (ASEME), in section 4, followed by the presentation of the evaluation results in section 5. Finally, in section 6, we discuss related work and conclude.

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.

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. 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 $L, L_1, \ldots, L_n$ 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 $L = h\_p(rule1, rule2)$. 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). An important notion in argumentation is that of **attack**. In the current framework an argument attacks (or is a counter argument of) another when they derive a contrary conclusion. Another notion is that of **admissibility**. An argument (from $\mathcal{T}$) is admissible if it counter-attacks all the attacks it receives. For this it needs to take along priority arguments (from $\mathcal{P}$) and makes itself at least as strong as its counter-arguments

Definition 2. 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. $R_1, R_2 \in \mathcal{P}_R \cup \mathcal{P}_C$.

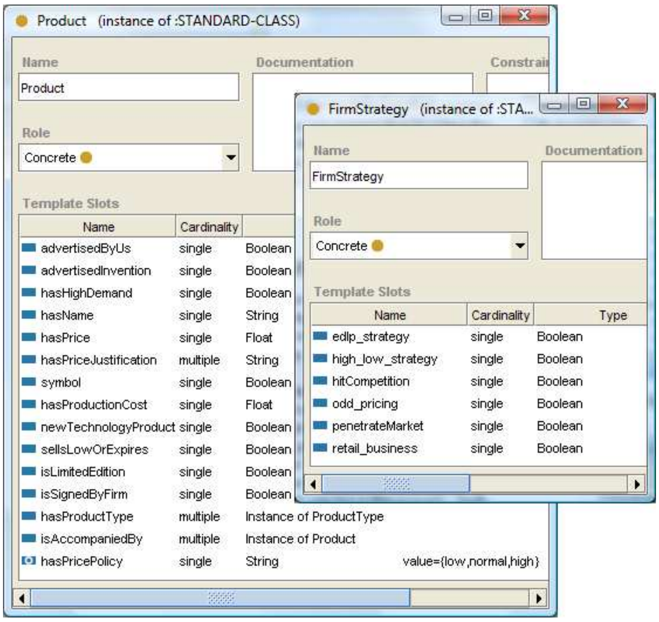
Thus, in defining the decision maker's theory three levels are used. The first level ($\mathcal{T}$) that defines the rules that refer directly to the subject domain, the second level that define priorities over the first level rules and the third level rules that define priorities over the rules of the previous level.

Gorgias (http://www.cs.ucy.ac.cy/~nkd/gorgias/), 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) where 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)., which 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)., which 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.

We used the Protégé (http://protege.stanford.edu/) open source ontology editor for defining the domain concepts and their properties and relations. In Figure 1, the *Product* concept and its properties are presented. The reader can see the properties identified previously *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. In the figure, we also present the firm strategy concept and its properties that are all *Boolean* and represent the different strategies that the firm 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. The property *retail_business* characterizes the firm as one in the retail business sector.



Product (instance of :STANDARD-CLASS)

Name: Product

Role: Concrete ●

Template Slots

Name	Cardinality		
advertisedByUs	single	Boolean	
advertisedInvention	single	Boolean	
hasHighDemand	single	Boolean	
hasName	single	String	
hasPrice	single	Float	
hasPriceJustification	multiple	String	
symbol	single	Boolean	
hasProductionCost	single	Float	
newTechnologyProduct	single	Boolean	
sellsLowOrExpires	single	Boolean	
isLimitedEdition	single	Boolean	
isSignedByFirm	single	Boolean	
hasProductType	multiple	Instance of ProductType	
isAccompaniedBy	multiple	Instance of Product	
hasPricePolicy	single	String	value={low,normal,high}

FirmStrategy (instance of :STA...

Name: FirmStrategy

Role: Concrete ●

Template Slots

Name	Cardinality	Type
edlp_strategy	single	Boolean
high_low_strategy	single	Boolean
hitCompetition	single	Boolean
odd_pricing	single	Boolean
penetrateMarket	single	Boolean
retail_business	single	Boolean

Fig. 1. The Product and FirmStrategy ontology concepts.

For our knowledge base definition we used Prolog. To use the concepts and their properties as they were defined in Protégé we defined that a Boolean property is encoded as a unary predicate, for example the *advertisedByUs* property of the *Product* concept 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 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 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. To resolve conflicts we consulted with the firm (executive) officers and defined priorities over the conflicting object rules. Consider, for example, the following rules (variables start with a capital letter as it is in Prolog):

```
rule(r1_2_2(Product), hasPricePolicy(Product, low), [hitProductTypeCompetition(
            ProductType), hasProductType(Product, ProductType)]).
rule(r2_3(Product), hasPricePolicy(Product, high), [newTechnologyProduct(Product),
            advertisedInvention(Product)]).
rule(pr1_2_6(Product), prefer(r1_2_2(Product), r2_3(Product)), []).
```

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 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 preferred to *r2_3*.

4 The Product Pricing Agent

In this section we firstly describe the Market-mIner product Pricing Agent (also referred to as MIPA) development process and then we focus in two important aspects of it, the decision making module and human-computer interaction.

We designed our agent using the Agent Systems Engineering Methodology (ASEME) [10]. During the analysis phase we identified the actors and the use cases related to our agent system (see Figure 2). Note that the Agent Modeling Language [9] (AMOLA), which is used by ASEME for modeling the agent-based system, allows for actors to be included in the system box, thus indicating an agent-based system. The system actor is MIPA, while the external actors that participate in the system’s environment are the user, external systems of competitors, weather report systems (as the weather forecast influences product demand as in the case of umbrellas) and municipality systems (as local events like concerts, sports, etc, also influence consumer demand). We started by identifying general use cases (like *interact with user*) and then we elaborated them in more specific ones (like *present information to the user* and *update firm policy*) using the <<include>> relation.

Then, we completed the roles model as it is presented in Figure 3(a). This model defines the dynamic aspect of the system, general use cases are transformed to capabilities, while the generic ones are transformed to activities. We used the Gaia operators ([14]) for creating *liveness formulas* that define the dynamic aspect of the agent system, what happens and when it happens. A. B means that activity B is executed after activity A, A^{ω} means that activity A is executed forever (when it finishes it restarts), A | B means that either activity A or activity B is executed and A || B means activity A is executed in parallel with activity B.

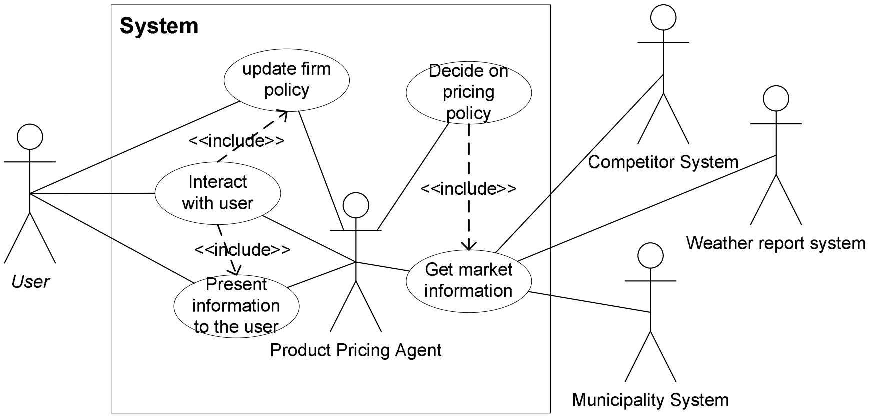


Fig. 2. MIPA Use Case Diagram

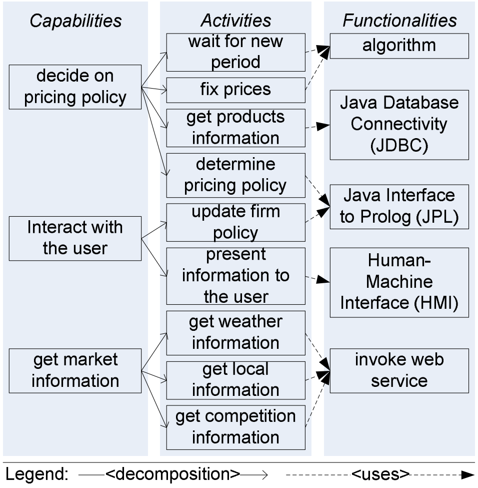
The next step was to associate each activity to a functionality, i.e. the technology that will be used for its implementation. In Figure 3(b) the reader can observe the capabilities, the activities that they decompose to and the functionality associated with each activity. The choice of these technologies is greatly influenced by non-functional requirements. For example the system will need to connect on diverse firm databases. Thus, we selected the JDBC technology (http://java.sun.com/javase/technologies/database/) that is database provider independent.

The last step, before implementation, is to extract from the roles model the statechart that resembles the agent. This is achieved by transforming the liveness formula to a statechart in a straightforward process that uses templates to transform activities and Gaia operators to states and transitions (see [9] for more details). The resulting statechart, i.e. the intra-agent control (as it is called in ASEME) is depicted in Figure 7. The statechart can then be easily transformed to a computer program.

The decision making capability includes four activities:

1. *wait for new period* activity: It waits for the next pricing period
2. *get products information* activity: It accesses a corporate database to collect the data needed for inference,
3. *determine pricing policy* activity: It reasons on the price category of each product, and,
4. *fix prices* activity: Based on the previous activity’s results, it defines the final product price.

Role: Product Pricing Agent
Liveness:
product pricing agent = (decide on pricing policy)$^{\omega}$ || (interact with user)$^{\omega}$ || [(get market information)$^{\omega}$]
decide on pricing policy = wait for new period. get products information. determine pricing policy. fix prices.
interact with user = (present information to the user | update firm policy)+
get market information = get weather information. get local information. get competition information.



(a) (b)

Fig. 3. MIPA Role Model (a) and the relation between Capabilities, Activities and Functionalities (b).

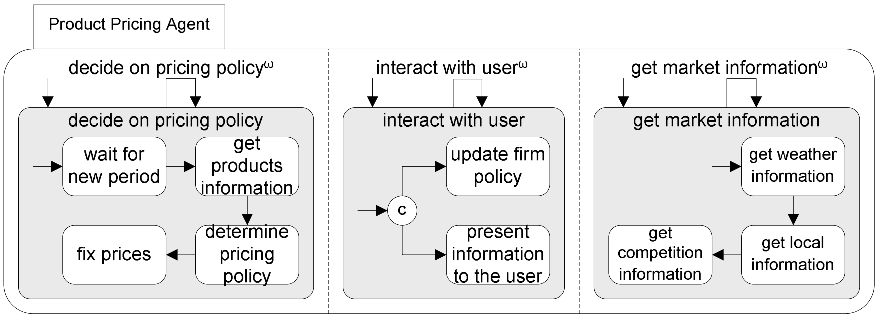


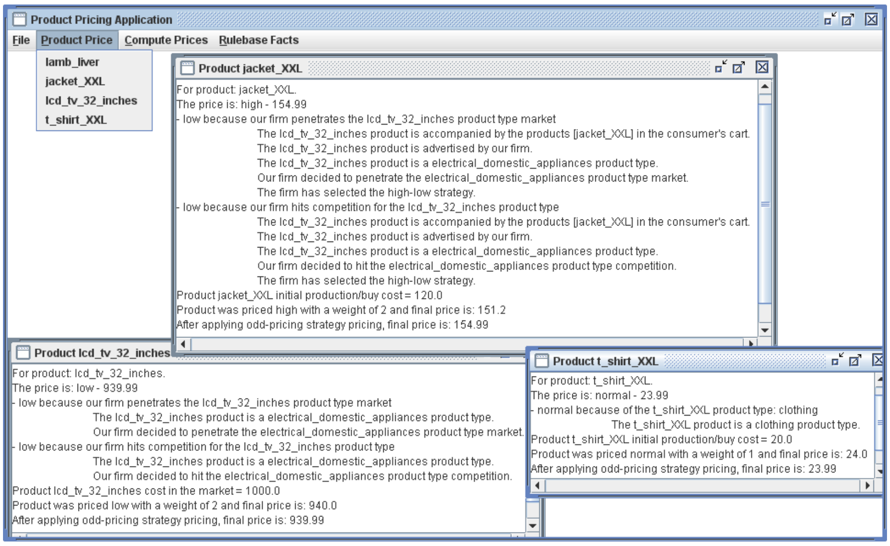
Fig. 4. MIPA Intra-agent Control Model

The *determine pricing policy* activity invokes the prolog rule base presented in §3 that includes 274 rules, 31 of which are the object rules and 243 are the priority rules. The *fix prices* activity's algorithm aims to produce a final price for each product. The algorithm's inputs are a) the procurement/manufacture cost for a product, or its price in the market, b) the outcome of the reasoning process (the price policy for each product), c) the default profit ratio for the firm, d) a step for rising the default profit ratio, e) a step for lowering this ratio, and, f) 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.

A screenshot from the human-machine interface is presented in Figure 5. 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 the:

```
rule(f1, high_low_strategy, []).
rule(f2, hitProductTypeCompetition(electrical_domestic_appliances), []).
rule(f3, penetrateProductTypeMarket(electrical_domestic_appliances), []).
rule(f4, hasProductType(jacket_XXL, clothing), []).
rule(f5, advertisedByUs(lcd_tv_32_inches), []).
rule(f6, advertisedInvention(lcd_tv_32_inches), []).
rule(f7, newTechnologyProduct(lcd_tv_32_inches), []).
rule(f8, isAccompaniedBy(lcd_tv_32_inches, [jacket_XXL]), []).
rule(f9, hasProductType(lcd_tv_32_inches, electrical_domestic_appliances), []).
rule(f10, hasProductType(t_shirt_XXL, clothing), []).
```

The reader should notice the application of the rules presented in §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 has also 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 5, 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, therefore, it is priced high according to the *high_low_strategy* of the firm.



```
For product: jacket_XXL.
The price is: high - 154.99
- low because our firm penetrates the lcd_tv_32_inches product type market
            The lcd_tv_32_inches product is accompanied by the products [jacket_XXL] in the consumer's cart.
            The lcd_tv_32_inches product is advertised by our firm.
            The lcd_tv_32_inches product is a electrical_domestic_appliances product type.
            Our firm decided to penetrate the electrical_domestic_appliances product type market.
            The firm has selected the high-low strategy.
- low because our firm hits competition for the lcd_tv_32_inches product type
            The lcd_tv_32_inches product is accompanied by the products [jacket_XXL] in the consumer's cart.
            The lcd_tv_32_inches product is advertised by our firm.
            The lcd_tv_32_inches product is a electrical_domestic_appliances product type.
            Our firm decided to hit the electrical_domestic_appliances product type competition.
            The firm has selected the high-low strategy.
Product jacket_XXL initial production/buy cost = 120.0
Product was priced high with a weight of 2 and final price is: 151.2
After applying odd-pricing strategy pricing, final price is: 154.99
```

```
For product: lcd_tv_32_inches.
The price is: low - 939.99
- low because our firm penetrates the lcd_tv_32_inches product type market
            The lcd_tv_32_inches product is a electrical_domestic_appliances product type.
            Our firm decided to penetrate the electrical_domestic_appliances product type market.
- low because our firm hits competition for the lcd_tv_32_inches product type
            The lcd_tv_32_inches product is a electrical_domestic_appliances product type.
            Our firm decided to hit the electrical_domestic_appliances product type competition.
Product lcd_tv_32_inches cost in the market = 1000.0
Product was priced low with a weight of 2 and final price is: 940.0
After applying odd-pricing strategy pricing, final price is: 939.99
```

```
For product: t_shirt_XXL.
The price is: normal - 23.99
- normal because of the t_shirt_XXL product type: clothing
            The t_shirt_XXL product is a clothing product type.
Product t_shirt_XXL initial production/buy cost = 20.0
Product was priced normal with a weight of 1 and final price is: 24.0
After applying odd-pricing strategy pricing, final price is: 23.99
```

Fig. 5. The Product Pricing Agent Application

5 Evaluation

The product pricing agent application was evaluated by SingularLogic SA, the largest Greek software vendor for SMEs. The MARKET-MINER project included an exploitation plan [12]. The application evaluation goals were to measure the overall satisfaction of its users. In the evaluation report [13] three user categories were identified, System Administrators, Consultants and Data Analysts. The criteria C1) *Performance*, C2) *Usability*, C3) *Interoperability*, and C4) *Security and Trust* were used for measuring user satisfaction. The users expressed their views in a relevant questionnaire 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 Process of Evaluation of Software Products [2] (MEDE-PROS) was used for our set of criteria. The results of the evaluation 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.

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 (see [13] for more details).

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

6 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 [7]. 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, where product pricing agents have been referred to as economic agents, as price bots, or, simply, as seller agents (see e.g. [6], [3] and [4]) and their responsibility is to adjust prices automatically on the seller's behalf in response to changing market conditions [6].

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.

The presented application's results 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. Charles C, Freeny Jr (2000) Automated Synchronous Product Pricing and Advertising. United States Patent 6076071
2. Colombo R, Guerra A (2002) The Evaluation Method for Software Product. In Proc 15th Int Conf on Softw & Syst Eng & Appl, Paris, France, December 3-4
3. Dasgupta P, Das S (2000) Dynamic pricing with limited competitor information in a multi-agent economy. In LNCS 1906, Springer-Verlag: 291-310
4. DiMicco JM, Greenwald A, Maes P (2001) Dynamic pricing strategies under a finite time horizon. In Proc ACM Conf on Electron Commer, October
5. Kakas A, Moraitis P (2003) Argumentation based decision making for autonomous agents. In Proc 2nd Int Conf on Auton Agents and Multi-Agent Syst, Melbourne, Australia, July 14-18
6. Kephart JO, Hanson JE, Greenwald AR (2000) Dynamic pricing by software agents. Comput Netw 36(6):731-752
7. Matsatsinis N, Moraitis P, Psomatakis V et al (2003) An Agent-Based System for Products Penetration Strategy Selection. Appl Artif Intell J 17(10):901-925
8. Moraitis P, Spanoudakis N (2007) Argumentation-based Agent Interaction in an Ambient Intelligence Context. IEEE Intell Syst 22(6):84-93
9. Spanoudakis N, Moraitis P (2008) The Agent Modeling Language (AMOLA). In LNCS 5253, Springer, Varna, Bulgaria
10. Spanoudakis N, Moraitis P (2007) The Agent Systems Methodology (ASEME): A Preliminary Report. In Proc 5th European Workshop on Multi-Agent Systems, Hammamet, Tunisia, December 13 - 14
11. Spanoudakis N, Pendaraki K (2007) A Tool for Portfolio Generation Using an Argumentation Based Decision Making Framework. In Proc Annual IEEE Int Conf on Tools with Artif Intell, Patras, Greece, October 29-31
12. Toulis P, Tzovaras D, Spanoudakis N (2007) MARKET-MINER Project Exploitation Plan. MARKET-MINER Proj Deliv Π6.1 (in Greek language), Singular Logic S.A.
13. Toulis P, Tzovaras D, Pantelopoulos S (2007) MARKET-MINER System Evaluation Report. MARKET-MINER Proj Deliv Π5.1 (in Greek language), Singular Logic S.A.
14. Wooldridge M, Jennings NR, Kinny D (2000) The Gaia Methodology for Agent-Oriented Analysis and Design. J Auton Agents and Multi-Agent Syst 3(3):285-312