Engineering an Agent-based System for Product Pricing Automation

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

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

[2]Laboratory of Informatics Paris Descartes (LIPADE) – Paris Descartes University, France
{nikolaos.spanoudakis, pavlos}@mi.parisdescartes.fr

Abstract This paper describes an autonomous agent conceived for automating the decision making process for pricing products. Product pricing involves the interaction of decision makers with different - possibly conflicting - points of view. Our approach allows for applying individual pricing policies to each product by taking into account different points of view expressed through different arguments and the dynamic environment of the application. This is done through the use of argumentation technology. The agent development process using the Agent Systems Engineering Methodology (ASEME) is also presented.

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 about the price of each of a firm's products. 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 [15] and for selecting the funds that should be included in an investment portfolio [19]. 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 this area that sparse works provide solutions. A relevant patent [3] only proposes an architecture and not the pricing mechanism itself. Existing works in seller and buyer agents development and benchmarking (see [5], [6] and [13]) 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 buyer. The retail business sector demanded a system that would have the possibility to apply a pricing policy adjusted to the market context, while reflecting the points of view of diverse decision makers.

The 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.

This paper shows the knowledge engineering approach for the problem domain (retail business sector). Then, it presents the product pricing agent software development process, for which we used the Agent Systems Engineering Methodology (ASEME) [18, 17]. The reader should note that this is the first demonstration of the usage of ASEME for engineering a real world system.

The basics of the used argumentation framework are firstly presented in section 2. Then, the reader can focus on our knowledge engineering approach for the application domain, which is described in section 3. Subsequently, in section 4, the paper presents the product pricing agent development process, including information on how we conceived and modeled the system using ASEME. Section 5 presents the results evaluation process and assessment followed by a discussion of related work in section 6. Finally, section 7 concludes.

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 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.

Deciding about the price of different products is a dynamic process that depends on several different parameters such as the price of similar products sold by concurrent sellers, the season of selling, the nature of the product, its characteristics, its usefulness/impact in the everyday life of people, etc. These parameters may represent a context (e.g. normal or sales season), general (e.g. the color of a product) or particular characteristics of the product, linked (or not) to the context (e.g. this product-an umbrella- is more useful in the winter than in the summer), and are determined on the basis of different and, very often, contradictory pricing policies for the product. For example, we could consider that the price of a new model of a product (e.g. an umbrella) should be low in order to

increase the market share (e.g. expressing a marketing policy point of view) of the firm and at the same time consider that the price of this model should be high in order to increase the turnover of the firm (e.g. expressing a financial policy point of view). Moreover, other, additional reasons, could influence the price of this model of umbrella like the season of the introduction to the market, considering that if it is during the summer this could be in favor of a low price, while in winter this could be in favor of a high price. Thus, two different, conflicting pricing policies (i.e. high price vs. low price) could be supported by different, acceptable but conflicting reasons (i.e. marketing policy vs. financial policy) or by different contexts (i.e. winter vs. summer). In such scenarios, either only one situation holds, in which case the conflict is easily resolved (we cannot have summer and winter at the same time), or several situations may simultaneously hold (marketing policy and financial policy) and in this case additional preferences are necessary (e.g. usually the increase of the market share is more important than short term profit, but the firm has a big deficit and, therefore, a short term profit increase is mandatory) in order to decide.

For dealing with the above situations we need a decision making framework which would allow to make decisions based on conflicting preferences, expressing different points of view, but also taking into account the particular context in which these decisions are made.

Argumentation techniques may be used to perform non-monotonic reasoning [2]. Argumentation can be abstractly defined as the formal interaction of different conflicting arguments for and against some conclusion due to different reasons and provides the appropriate semantics for resolving such conflicts. Thus, it is very well suited for implementing decision making mechanisms dealing with the above requirements. Moreover, the dynamic nature of those conflicting decisions due to different situations or contexts needs a specific type of argumentation frameworks ([16], [11]). These frameworks are based on object level arguments representing the decision policies and then they are using priority arguments expressing preferences on the object level arguments in order to resolve possible conflicts. Subsequently, additional priority arguments can be used in order to resolve potential conflicts between priority arguments of the previous level. Therefore, we are concerned with argumentation frameworks that allow for the representation of dynamic preferences under the form of dynamic priorities over arguments. In this work we are using the framework proposed by Kakas and Moraitis ([11], [10]). This framework has been applied in a successful way in different applications (see e.g. [15], [19]) involving similar scenarios of decision making and it is supported by an open source software called Gorgias. The following definitions formally 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 $L = h\_p$(rule1,

rule2) (h_p stands for higher priority). 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.:

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

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

$T \vdash_{\min} L$ means that $T \vdash L$ under the background logic and that L cannot be derived from any proper subset of T. Here $T \cup P \vdash_{\min} L$ means $T \vdash_{\min} L$. This definition means that a composite argument *(T',P')* is a counter-argument to another such argument when they derive a contrary conclusion L and *(T' ∪ P')* makes the rules of its counter proof at least "as strong" as the rules of the proof of the argument that is under attack. The second condition says that an attack can occur on a contrary conclusion L that refers to the priority between rules.

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 $(T' \cup P')$, $T' \subseteq \mathcal{T}$, $P' \subseteq \mathcal{P}$, 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, for an argument (from $\mathcal{T}$) to be admissible it needs to take along with it priority arguments (from $\mathcal{P}$) to make itself at least "as strong" as the opposing counter-arguments. This need for priority rules can repeat itself when the initially chosen ones can themselves be attacked by opposing priority rules and again we would need to make the priority rules themselves at least as strong as their opposing ones.

An agent's argumentation theory for describing his policy can now be defined as follows

Definition 4. An agent's **argumentative 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. These priorities can be based on the specific *roles* that agents can assume or to generic and specific conditions (or situations), while the third level rules define priorities over these rules based on generic or specific contexts. Finally, we can also have priorities over this third level rules based on preferences between different contexts.

To explain how this argumentation framework works, we present an example where the theory $\mathcal{T}$ represents part of the object-level decision rules of a company employee (nonground rules represent their instances in a given Herbrand universe). Here and later, we use logic-programming notation, in which any term starting with a capital letter represents a variable. Abusing this notation, we'll denote the constant names of the priority rules R and C with capital letters.

r_1: give(A, Obj, A_1) ← requests(A_1, Obj, A)
r_2: ¬give(A, Obj, A_1) ← needs(A, Obj)

In addition, a theory $\mathcal{P}_R$ represents the general default behavior of the company's code of contact in relation to its employees' roles. That is, a request from a superior is generally stronger than an employee's own need, and a request from another employee from a competing department is generally weaker than an employee's own need.

R_1: h-p (r_1, r_2) ← higher_rank(A_1, A)
R_2: h-p (r_2, r_1) ← competitor(A, A_1)

Between the two alternatives to satisfy a request from a superior from a competing department or not, the first is stronger when these two departments are in the specific context of working together on a common project. On the other hand, if the employee has an object and needs it urgently, then he would prefer to keep it. Such policy is represented at the third level in $\mathcal{P}_C$:

C_1: h-p (R_1, R_2) ← common_project(A, Obj, A_1)
C_2: h-p (R_2, R_1) ← urgent(A, Obj)

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. The language for representing the theories is given by rules with the syntax in formula (1).

rule(Signature, Head, Body). (1)

[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 rule presented in formula (1), 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 syntax defined in (1) we can write the rule presented in formula (2).

rule(Signature, prefer(Sig1, Sig2), Body). (2)

Formula (2) 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 presented in formula (3).

conflict(Sig1,Sig2). (3)

The rule in formula (3) 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. A negative literal is a term of the form *neg(L)*. There is also the possibility to define conflicting predicates that are used as heads of rules using the rule presented in formula (4).

complement(Head1, Head2). (4)

3. Domain Knowledge Modeling

Our first step in domain knowledge modeling was to gather 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" concept 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 be 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 developed ontology is presented as a UML class diagram. Focusing in the *Product* concept, the reader can see the properties identified in the previous paragraph *hasPrice* and *isAccompaniedBy*. Price is defined as a real number (*Single*) and *isAccompaniedBy* relates the product to multiple other instances of products that accompany it in the consumer's cart. The *Product* fields correspond to

- corporate database values (*hasName, advertisedInvention, hasHighDemand, hasProductionCost, isLimitedEdition, isSignedByFirm, newTechnologyProduct, sellsLowOrExpires, symbol*)
- external data (*hasPrice, hasHighDemand*)
- company policy (*advertizedByUs*)
- mined data (*isAccompaniedBy*)
- decision making results (hasPricePolicy, hasPriceJustification, hasPrice)

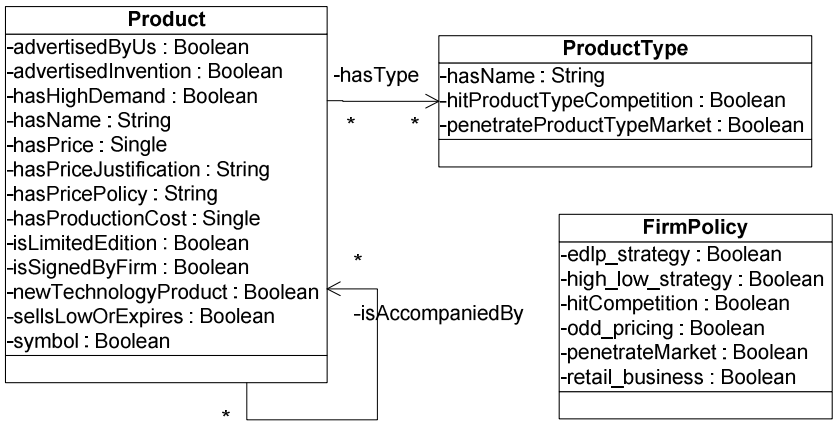


Figure 1. The MarketMiner Ontology depicted as a UML class diagram.

---

[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://protegewiki.stanford.edu/index.php/OntologyBeanGenerator)

In Figure 1 we also present the *FirmStrategy* concept and its properties. As the reader can observe they 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. 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. Finally the *ProductType* concept has the firm policy properties of *hitProductTypeCompetition* and *penetrateProductTypeMarket*.

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* – a constraint of enumeration type. Then, we defined the different policies as conflicting, thus only one policy was acceptable (or admissible) per product. This is expressed by the Prolog statements shown in Figure 2.

In order to resolve conflicts we consulted with the firm (executive) officers and defined priorities over the conflicting object rules. However, the management of

those rules became difficult as their number increased. In order to overcome this issue we created a matrix with all defined rules both in the row and column headings. Then we used the "greater than" symbol (>) to show which rule had priority. The decision makers (either by voting or by the decision of the Chief Executing Officer) defined the priorities over conflicts.

```
…
level(high).
level(normal).
level(low).
complement(hasPricePolicy(Product, X), hasPricePolicy(Product, Y)) :-
        level(X), level(Y), X\=Y.
…
```

Figure 2. An extract from the rule base showing how we define conflicting *hasPricePolicy* predicates.

In Table 1, the reader can see the priorities set for conflicting rules. Rules with the format 1.X are rules for pricing a product low, those with the format 2.X are for pricing a product high and rule 3 is for normal pricing. Rule number 4 defines the *Leader pricing* strategy, where a high demand product is priced low so that customers are motivated to visit the store. Rule number 5 defines the *high-low* strategy, according to which when an advertised product is priced low the products that accompany it in the consumers basket are priced high. Rule number 6 defines the *EDLP* (Every Day Low Price) strategy, according to which prices are always low. Rule number 7 is about pricing high products based on dynamic information (e.g. weather–based sales such as umbrellas). Finally, rule number 8 is about pricing low products that are close to their expiration date or that simply need to be sold quickly for any firm-related reason. For example, rule 1.2 dictates that the product should be priced low if the firm wants to hit the competition. However, even these rules may be further refined to more rules. Rule 1.2.1 states that all products should be priced low for a general hit on the competition, while rule 1.2.2 states that only products of a specific product type are priced low as the firm just wants to hit the competition on that product type (e.g. electric appliances). Rule 2.3 proposes that a product that is a new technology advertized invention should be priced high. Rule 1.2 has precedence over rule 2.3 and this is marked with the ">" symbol in the second row of the table.

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 a specific 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*. The reader should note that for our knowledge base we assume that the closed world assumption holds.

Table 1. The matrix for defining the rules' priorities. Reading each row the reader can understand which rule has priority over another.

	1.1	1.2	1.3	2.1	2.2	2.3	2.4	3	4	5	6	7	8
1.1				>	>	>	>	>		>		>	
1.2				>	>	>	>	>		>		>	
1.3													
2.1			>					>	>				
2.2			>					>	>				
2.3			>					>					
2.4			>					>					
3													
4						>		>		>		>	
5			>					>					
6						>		>				>	
7			>					>					
8						>		>		>		>	

```
#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)), []).
…
```

Figure 3. An extract from the rule base showing two conflicting rules and the rule defining which has priority over the other.

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 on two important aspects of it, the decision making module and human-computer interaction.

According to the definition of [23], "an agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives". MIPA fully qualifies as an agent as it:

a) is situated in the firm environment,
b) proactively changes the prices regularly, and,

c) reacts to changes in his environment.

Thus, we used the Agent Systems Engineering Methodology (ASEME) [18] for designing the MIPA.

4.1 The Agent Development Process

ASEME was selected among other Agent Oriented Software Engineering methodologies [1, 9] because its models can lead to agent development without imposing constraints on how the mental model of the agent will be represented (in this case using argumentation for decision making). Furthermore, it is an agile process allowing for rapid prototyping needing in the fastest case (the one presented herein) the editing of just three models. Finally, it allows for modular development allowing specialized teams to develop different modules (usually related to diverse technologies) using the intra-agent control model (which is a statechart [8]) to glue them together.

During the analysis phase we identified the actors and the use cases related to our agent system. We documented these findings using the ASEME System Use Cases (SUC) model (see Figure 4). It is defined by the Agent Modeling Language [17] (AMOLA), which is used by ASEME for modeling the agent-based system, and its innovation, with relation to classical UML Use Case diagrams, is that it allows for actors to be included in the system box, thus indicating an agent-based system.

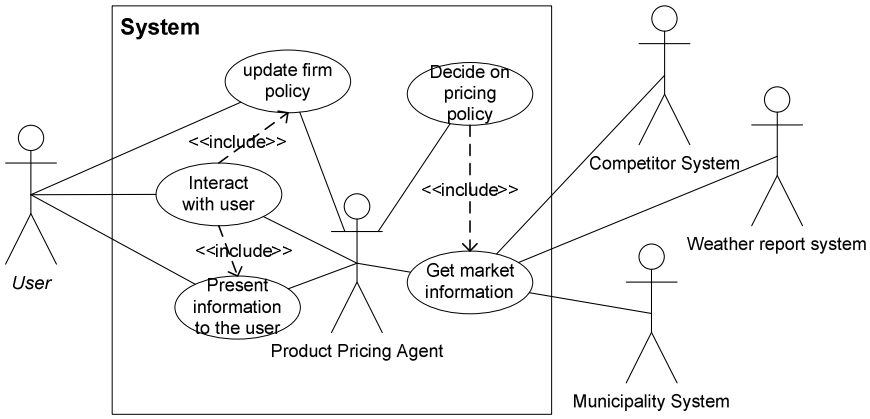


Figure 4. The MARKET-MINER System Use Cases (SUC) model.

For our system, the system actor is MIPA (or Product Pricing Agent as shown in Figure 4), 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, like in the case of umbrellas) and

municipality systems (as local events like concerts, sports, etc, also influence consumer demand).

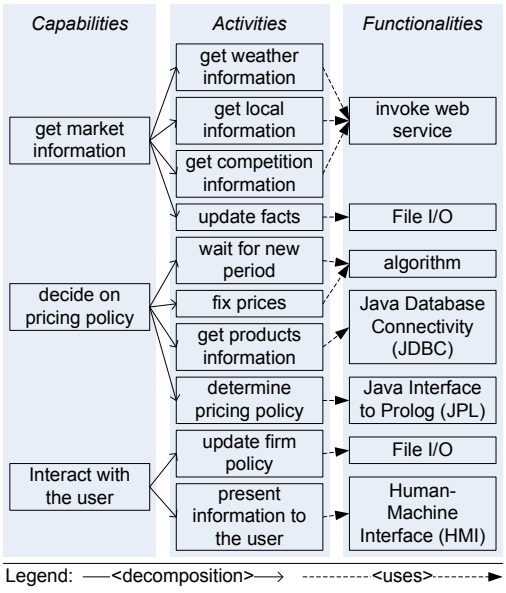
Having defined the involved actors we started identifying general use cases (like *interact with user*) and then we elaborated them in more specific ones (like *present information to the user* and *update firm policy*) using the <<include>> relation (see Figure 4).

After refining the use cases, the SUC model was transformed to the AMOLA System Roles Model (SRM). This model defines the dynamic aspect of the system. An SRM model is created for each actor in the use case diagram. General use cases connected to the role are transformed to capabilities, while the generic ones are transformed to activities.

We used the Gaia operators ([22, 17]) for creating *liveness formulas* that define the dynamic aspect of the agent system, what happens and when it happens. Briefly, 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. The SRM model for the Product Pricing Agent is presented in Figure 5(a).

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. update facts

(a)



(b)

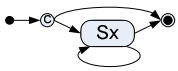
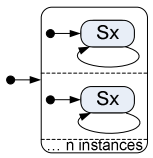
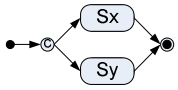
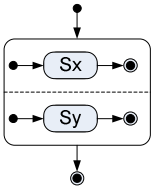
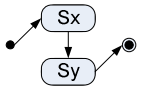
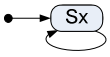
Figure 5. The MARKET-MINER Product Pricing Agent System Roles Model (SRM) (a) and the relation between its Capabilities, Activities and Functionalities (b).

The next step was to associate each activity to a functionality, i.e. the technology that will be used for its implementation. In Figure 5(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[4] technology that is database provider independent. Moreover, the different information channels that are currently used depend on the same functionality, i.e. a web service invocation. Thus, in the future, new information channels such as a financial channel where from to get relevant news, such as a financial crisis, can be integrated in the system using the same functionality. This step is important as it defines what competencies are needed (or the required know-how) for the system implementation team.

The last step, before implementation, is to extract from the roles model the Intra-Agent Control (IAC) model 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 [17] for more details). Table 2 shows the Gaia operators transformation templates. They are applied recursively reading the SRM model liveness formula top to bottom and left to right.

Table 2. Templates of extended Gaia operators (Op.) for Statechart generation.

Op.	*Template*	*Op.*	*Template*
x^*		$\|x^{\omega}\|^n$	
[x]			
x|y		x||y	
x.y			
x+		x^{ω}	

[4] The Java Database Connectivity (JDBC) is a standard for database-independent connectivity between the Java programming language and a wide range of databases providing a call-level API for SQL-based database access (http://java.sun.com/javase/technologies/database).

The resulting IAC model for MIPA is depicted in Figure 6. It was defined in the Rhapsody Computer-Aided Software Engineering (CASE) tool[5]. This tool automates the process of transforming a statechart to C++, Java, C and Ada code. The statechart, in this case was transformed to a Java program. After the initial creation of the *ProductPricingAgent.java* file the developer needed to locate the methods concerning the different activities execution and add the functionality – specific code there (see e.g. Figure 7).

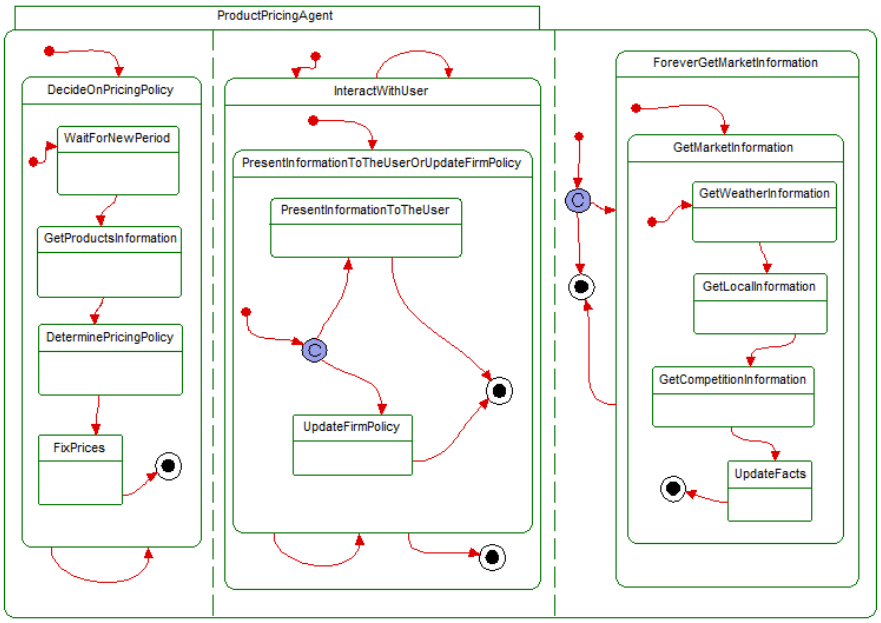


Figure 6. The MARKET-MINER Product Pricing Agent Intra-Agent Control (IAC) model.

```
…
public void GetProductsInformationEnter() {
        //#[ state ROOT.ProductPricingAgent.ForeverDecideOnPricingPolicy.
DecideOnPricingPolicy.GetProductsInformation.(Entry)
        //TO DO to connect to firm data base and update the products' information
        //#]
}
…
```

Figure 7. An extract from the automatically generated *ProductPricingAgent.java* class by the Rhapsody© CASE tool.

[5] IBM Rational Rhapsody is a commercial model-driven development tool (http://www.telelogic.com/products/rhapsody)

4.2 The Decision Making Capability

Figure 5(a) shows that 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 in order 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.

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:

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 computed using the formula (5).

$$P = C * (1 + R) . \quad (5)$$

In formula (5), *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 the product price is computed using the formula (6).

$$P = C * (1 + R + m * H) . \quad (6)$$

In formula (6), *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. In this case the product price is computed using the formula (7).

$$\begin{aligned} &\text{If } n^* L > D \\ &\text{Then } P = C * (1 + R - D) \\ &\text{Else } P = C * (1 + R - n * L)\ . \end{aligned} \tag{7}$$

In formula (7), 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 (5), (6) and (7) 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. Odd pricing can also refer to the practice of ending prices in any odd number (1, 3, 5, 7, 9) or to that of ending prices in a number other than zero; or to that of pricing just below a zero (e.g. $2.99 or $19.95). Odd pricing is prevalent in retailing [7].

4.3 Human-Computer Interaction

A screenshot from the human-machine interface is presented in Figure 8. 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 9.

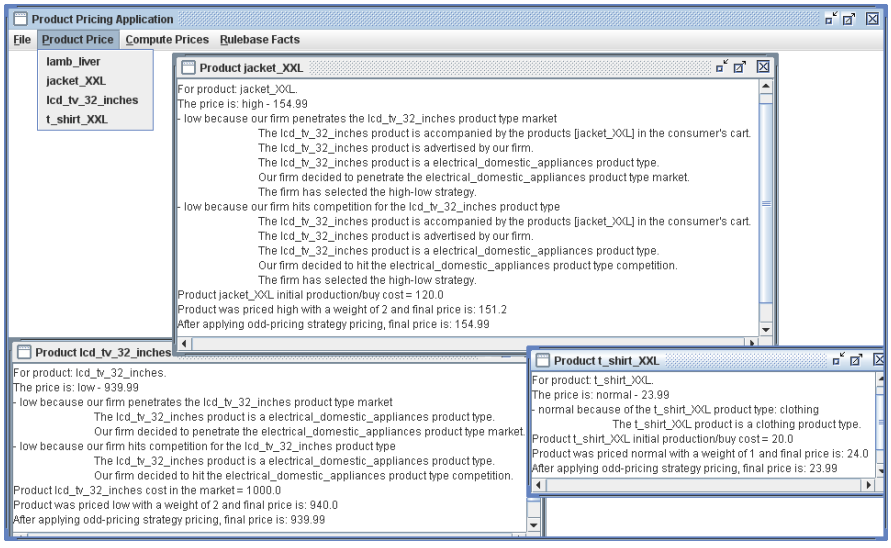


Figure 8. The Product Pricing Agent Application

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.

```
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), []).
```

Figure 9. A set of sample facts for decision making.

In Figure 8, 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.

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 [20]. The application evaluation goals were to measure the overall satisfaction of its

users. In the evaluation report [21] 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 [4] (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 3 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 3. 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 [21] 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

The responses of the decision makers (executive personnel) of the firms had some results that were not expected when defining the requirements for MARKET-MINER. The 80% of the responses of executives in the retail business domain found an added value in using MARKET-MINER for defining different strategies per branch (e.g. in one region they need to hit competition and in another they don't). On the other hand the 75% of the responses of executives in the services business domain thought that the technology is still immature for automatically pricing a service. Thus, MARKET-MINER is expected to be more useful for the retail business sector, while at the beginning it was conceived to address the needs for the retail, production and services sectors.

6. Related Work

In the agent technology literature product pricing agents have been referred to as economic agents, as price bots, or, simply, as seller agents (see e.g. [13], [5] and [6]) and their responsibility is to adjust prices automatically on the seller's behalf in response to changing market conditions [13].

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 [13]. 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 [6].

In [13], 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 choose 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, he 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 [6], 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 [13].

In another work ([5]), 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.

A patent just provided some guidelines on an architecture for such a system exclusively for super market chains [3]. Earlier works proposed a support of the product pricing process for the retail business sector but did not provide an automated decision mechanism [14].

7. 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. In this paper we used argumentation that allows for expressing conflicting views on the subject and a mechanism for resolving these conflicts for the first time. Thus, this approach represents the points of view of different departments of a firm such as the marketing department, the financial department, the production department

but also information coming from external sources and defines the priorities between conflicting rules. It allows the adoption of diverse pricing strategies in the literature and most importantly it applies them individually to each product in regular intervals (e.g. imagine the marketing department reading the weather forecasts just to determine the price of the umbrellas for the next day).

MARKET-MINER 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 conditions. 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 points out the 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 [12]) and is part of our future work. The same solution (the use of different agents) could be an improvement of the current system in the case of scaling because the growing knowledge of an agent could be shared with different agents each of them representing a different department. Such a solution has been proposed for a similar problem by the authors in [15].

The MIPA agent development successfully demonstrates the use of the ASEME development process [17, 18] for real-world agent-based systems development. It also provides an encoding of the popular Protégé tool ontology to Prolog predicates. It is within the intentions of the authors to automate this encoding as future work.

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 [4]) and they were proposed by the SingularLogic research and development department for commercialization by the firm.

Acknowledgements

We would like to thank the reviewers for their valuable and constructive comments. We also thank Singular Logic SA and the General Secretariat for Research and Technology of the Greek Ministry of Development for partially funding and for supporting this work.

References

1. Bergenti F, Gleizes MP, Zambonelli F, editors (2004) Methodologies and Software Engineering for Agent Systems. Kluwer
2. Bench-Capon TJM, Dunne, PE (2007) Argumentation in Artificial Intelligence. Artificial Intelligence 171(10-15):619-641
3. Charles C, Freeny Jr (2000) Automated Synchronous Product Pricing and Advertising. United States Patent 6076071
4. Colombo R, Guerra A (2002) The Evaluation Method for Software Product. In Proc of the 15th Int Conf on Softw & Syst Eng & Appl, Paris, France, December 3-4
5. Dasgupta P, Das S (2000) Dynamic pricing with limited competitor information in a multi-agent economy. In LNCS 1906, Springer-Verlag: 291-310
6. DiMicco JM, Greenwald A, Maes P (2001) Dynamic pricing strategies under a finite time horizon. In Proc of ACM Conf on Electron Commer, October
7. Gendall P, Fox MF, Wilton P (1998) Estimating the effect of odd pricing, J Product & Brand Management 7(5):421-432
8. Harel D, Naamad A (1996) The STATEMATE Semantics of Statecharts. ACM T Softw Eng Meth 5(4):293-333
9. Henderson-Sellers B, Giorgini P, editors (2005) Agent-Oriented Methodologies. Idea Group Publishing
10. Kakas A, Moraitis P, (2002) Argumentative Agent Deliberation, Roles and Context. Electronic Notes in Theoretical Computer Science 70(5):39-53
11. Kakas A, Moraitis P (2003) Argumentation based decision making for autonomous agents. In Proc of the 2nd Int Conf on Auton Agents and Multi-Agent Syst, Melbourne, Australia, July 14-18
12. Kakas A, Moraitis P (2006) Adaptive Agent Negotiation via Argumentation. In Proc of the 5th Int Conf on Auton Agents and Multi-Agent Syst, Hakodate, Japan:384-391
13. Kephart JO, Hanson JE, Greenwald AR (2000) Dynamic pricing by software agents. Comput Netw 36(6):731-752
14. Matsatsinis N, Moraitis P, Psomatakis V, Spanoudakis N (2003) An Agent-Based System for Products Penetration Strategy Selection. Appl Artif Intell J 17(10):901-925

15. Moraitis P, Spanoudakis N (2007) Argumentation-based Agent Interaction in an Ambient Intelligence Context. IEEE Intell Syst 22(6):84-93

16. Prakken H, Sartor G (1996) A dialectical model of assessing conflicting arguments in legal reasoning. Artifcial Intelligence and Law 4(3-4):331-368

17. Spanoudakis N, Moraitis P (2008) The Agent Modeling Language (AMOLA). In LNCS 5253, Springer, Varna, Bulgaria

18. Spanoudakis N, Moraitis P (2007) The Agent Systems Methodology (ASEME): A Preliminary Report. In Proc of the fifth European Workshop on Multi-Agent Systems, Hammamet, Tunisia, December 13 - 14

19. Spanoudakis N, Pendaraki K (2007) A Tool for Portfolio Generation Using an Argumentation Based Decision Making Framework. In Proc of the Annu IEEE Int Conf on Tools with Artif Intell, Patras, Greece, October 29-31

20. 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.

21. 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.

22. 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

23. Wooldridge M (2002) An introduction to multiagent systems. Wiley