ABSTRACT
Reliability is a key challenge in e-commerce platforms in general and in autonomous commerce multi-agent systems in particular. Most formal methods proposed to check reliability of these systems focus on checking low-level details such as the communication protocols among agents. In this paper we present a formalism to specify autonomous commerce agents as well as systems made of them. This formalism focuses on describing the high-level behavior of agents. Thus, agents are considered economic entities that have different preferences along time and must perform transactions according to them. Besides, we specify systems as environments where all the agents fulfill their own specification. These formalisms can be applied to check the reliability of an agent or system by comparing its behavior with that of its specification.

Categories and Subject Descriptors
J.8.e [Electronic Commerce]; G.4.b [Certification and Testing]

Keywords
e-commerce, formal methods.

1. INTRODUCTION
Nowadays, one of the cutting-edge technologies leading a major evolution and revolution in e-commerce platforms are autonomous commerce agents (see e.g. [5, 10, 9, 7]). As e-commerce technologies have opened a new path in human Economics in terms of efficiency and comfort for both vendors and customers by eliminating physical distances, autonomous agent technologies go one step beyond by releasing users from performing some of the processes needed in economic interaction. By interacting on behalf of their users, autonomous agents take advantage of their domain-specific knowledge when searching for interesting items in the market, advising users about offers fitting into their preferences, negotiating to reach profitable deals, or even performing transactions autonomously. Thus, specific knowledge regarding available vendors, prices, market rules, or strategies can remain transparent for users. Actually, autonomous agents put an abstraction layer between objectives and mechanisms in the same way Computer Science splits the software development process into specification and implementation.

Either by making recommendations and negotiating or transacting, the behavior of an autonomous agent has a critical repercussion in the possessions of its corresponding user. Thus, it is clear that the success of autonomous agents as the leading technology in e-commerce depends dramatically on the confidence the users have in their reliability. Actually, as it happened with the first e-commerce environments, agent-based commerce applications need the customers to come through the doubts and fears inherent to any new technology. Formal methods provide a powerful tool to validate or verify the behavior of any software system in general or e-commerce environment in particular. Following this line, several formal techniques have been proposed to check the validity of communications in e-commerce systems. Since communications are the basis of transactions, it is critical to check whether they keep the original meaning the emitter initially gave to it. Key requirements to validate regarding communications in e-commerce systems are privacy, authentication, non-repudiation, and integrity of data [1]. The techniques proposed to address these issues focus on validating the low-level behavior of e-commerce systems. Therefore, they are quite similar to those proposed to deal with any other communication protocol, and they are not specific to the e-commerce domain. However, the field of e-commerce, and more specifically the field of autonomous exchange agents in e-commerce, deserves its own formal techniques to deal with its own specific features. Only by developing domain-specific techniques we will be able to validate efficiently the high-level behavior of an autonomous agent, focusing on whether it fulfills its objectives rather than checking how it achieves them.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SAC '04, March 14-17, 2004, Nicosia, Cyprus
©2004 ACM 1-58113-812-1/03/04 ...$5.00.

2004 ACM Symposium on Applied Computing

Formal Specification of Autonomous Commerce Agents *

Ismael Rodríguez
Dept. Sistemas Informáticos y Programación
Universidad Complutense de Madrid
28040 Madrid. Spain
isrodrig@sip.ucm.es

*Work supported by the Spanish MCyT project MASTER (TIC2003-07848-C02-01) and the Junta de Castilla-La Mancha project DISMEF (PA-C-03-001).
can be stated as "get what the user wants when the user wants". Thus, the specification of an agent requires that the transactions performed by the agent fit into the preferences of its user in each moment. In this paper we will not deal with how the agents infer the preferences of the users, because it goes beyond our scope (for discussions of this topic, see e.g. [3, 4, 6]). So, we will suppose that these preferences are explicitly stated in the specifications. Our specifications will focus on identifying the high-level economic behaviors that are profitable according to the user preferences. Thus, they will not include any requirement regarding the low-level behavior of agents. For example, specifications will describe neither the specific communication protocols among agents nor the specific strategies used by agents to reach good deals with other agents. So, specifications will define accurately the goals, not the way to achieve them. Similarly to the specifications of agents, specifications of systems will define behaviors of systems. These specifications will denote systems where the specification of each single agent within the system is fulfilled. Both kinds of specifications will be equipped with a set of operational semantics to denote the kind of evolutions a correct agent system could perform. These semantics will allow us to check whether the behavior of a real agent actually conforms to its specification.

The rest of the paper is structured as follows. In section 2 we describe some basic concepts and afterwards we introduce our formalism to specify agents as well as its operational semantics. In section 3 we extend the notions of the previous section to describe the specifications of systems. Finally, in section 4 we present our conclusions and some lines of future work.

2. SPECIFICATION OF AGENTS

In this section we present the formalism we will use to specify our autonomous exchange agents. As we said in the introduction, our validation methodology will focus on determining whether an implementation fulfills the desires of the users in terms of utility gained, but we will not be interested on how the utility is gained. So, the formalism will not include details about how private data are encrypted, how to find potential customers, how to estimate the confidence level on a given vendor, or what is the specific strategy to compete with other agents. Instead, we will check whether the transactions are performed according to the preferences of the users. Obviously, the performance of agents will be influenced by low-level details, but this influence will only be considered on the basis of its consequences, that is, on the basis of the high-level behavior.

The objectives of the users could be different according to the situation. In an e-commerce environment, the objectives are measured in terms of the obtained resources. Users will have different preferences and the first step to construct the specification of the corresponding agent consists in expressing these preferences. The preferences of the user in a given moment will be given by its utility function. Utility functions associate a value (a utility measure) with each possible combination of resources a user could own.

**Definition 2.1.** We consider \( n \) different resources. A utility function is any function \( f : \mathbb{R}^n \to \mathbb{R}_+ \).

From now on tuples such as \((x_1, \ldots, x_n)\) will sometimes be denoted just by \( \mathbf{x} \). Intuitively, if \( f \) is a utility function then \( f(\mathbf{x}) > f(\mathbf{y}) \) means that \( \mathbf{x} \) is preferred to \( \mathbf{y} \). For instance, if the resource \( x_1 \) denotes the amount of apples and \( x_2 \) denotes the amount of oranges, then \( f(x_1, x_2) = 3 \cdot x_1 + 2 \cdot x_2 \) means that, for example, the user is equally happy owing 2 apples or 3 oranges. Let us consider another user whose utility function is \( f(x_1, x_2) = x_1 + 2 \cdot x_2 \). Then, both users can make a deal if the first user gives 3 oranges in exchange of 4 apples: After the exchange both users are happier. Let us suppose now that \( x_2 \) represents the amount of money (in euros). In this situation, the first user would be the customer while the second one might represent the vendor. Let us remark that utility functions allow a great expressivity in preferences. For instance, \( f(x_1, x_2) = x_1 \cdot x_2 \) represents a utility function denoting that variety is preferred. A usual assumption is that no resource is a bad, that is, \( \Delta f(x_1, \ldots, x_n) \geq 0 \) for any \( x_1, \ldots, x_n \in \mathbb{R} \) and \( 1 \leq i \leq n \). This requirement does not constrain the expressive power of utility functions, as the existence of any undesirable resource can be always expressed just by considering another resource representing the absence of it.

In order to formally specify autonomous commerce agents we have to represent the different objectives of the corresponding user along time. Thus, our formalism will provide the capability of expressing different utility functions depending on the situation, represented by the current state. In addition, objectives, represented by utility functions, will change depending on the availability of resources. These events will govern the transitions between states. Our formalism, that we call utility state machine, is indeed an adaption to our purposes of the classical notion of EFSA. We will be able to deal with time as a factor that influences the preferences of users, either by affecting the value returned by a given utility function (e.g. the interest in a given item could decay as time passes) or by defining when the utility function must change (e.g. an old technology is considered obsolete and it is no longer interesting). Besides, time will affect agents in the sense that any negative transaction will imply a deadline for the agent to retrieve the benefits of it. In fact, gaining profit in the long run may sometimes require to perform actions that, considered in the short term, are detrimental. Thus, time will be used to check whether the transaction was beneficial in the long term. In addition, time will appear as part of the freedom that specifications give to implementations: It does not matter whether an agent immediately performs a transaction as long as its decision is useful to improve the utility in the long term.

**Definition 2.2.** We say that \( M = (S, s_0, V, U, at, mi, T) \) is a utility state machine, in short USM. We consider that

- \( S \) is a set of states.
- \( s_0 \in S \) is the initial state of \( M \).
- \( V = (t, x_1, \ldots, x_n) \) is a tuple of variables, where \( t \) represents the time elapsed since the machine entered the current state and \( x_1, \ldots, x_n \) represent the resources that are available to be traded in the e-commerce environment.
- \( U : S \to (\mathbb{R}_+)^{n+1} \to \mathbb{R}_+ \) is a function associating a utility function with each state in \( M \).
- \( at \) is the amortization time. It denotes the maximal time \( M \) may stay without retrieving the profit of the negative transactions that were performed in the past.
• $m_i$ is the maximal investment. It denotes the maximal amount of negative profit the machine should afford.

• $T$ is the set of transitions. Each transition is a tuple $(s, Q, Z, s')$, where $s, s' \in S$ are the initial and final state of the transition respectively, $Q : \mathbb{R}_+^{n+1} \to \text{Bool}$ is a predicate on the set of variables, and $Z : \mathbb{R}_+^n \to \mathbb{R}_+^{n+1}$ is a transformation over the current variables. We require that for any state $s$ there do not exist two transitions $t_1, t_2 \in T$, with $t_1 = (s, Q, Z, s')$ and $t_2 = (s, Q, Z, s'')$, and a tuple $\mathbf{r}$ such that both $Q_1(\mathbf{r})$ and $Q_2(\mathbf{r})$ hold.

Let us remark that both available resources and time influence the conditions required to change the state of the machine. This is taken into account by the $Q$ functions appearing in transitions. Let us also note that in these constraints we may also consider the time previously consumed. In fact, we could use an additional abstract resource to accumulate the time consumed in previous states.

It is worth to point out that the set of transitions $T$ does not include the complete set of transitions the specification will allow real agents to perform. Some additional transitions must be considered: transactions and time consumption. The former will be used to denote the transactions agents will be allowed to perform. These transactions will include some simple constraints, like avoiding to give more resources than those actually owned, or avoiding giving resources if the amount of value lost in previous negative transactions, and not still retrieved, is too high. Regarding time consumption transitions, they will denote the free decision of agents to idle. Their waiting times will be constrained by two conditions. First, an agent cannot let the time pass so that previous negative transactions are not retrieved before their deadlines. Second, in some situations time triggers the modification of the state of an agent. Thus, in the exact time it happens, and before the waiting action continues, the transition between states will be performed. Given a USM $M$, both transaction and time consumption transitions may be implicitly inferred from the definition of $M$ and we will give the formal representation in the forthcoming Definition 2.5. Next we introduce the notion of configuration, that is, the data denoting the current situation of a USM. A configuration consists of the current state of the machine, the current values of the variables, and the pending accounting of the machine.

Definition 2.3. Let $M = (S, s_0, V, U, at, m, T)$ be a USM. A configuration of $M$ is a tuple $(s, \mathbf{r}, l)$ where

• $s \in S$ is the current state in $M$.

• $\mathbf{r} = (r_1, \ldots, r_n) \in \mathbb{R}_+^{n+1}$ is the current value of $V$.

• $l = [(l_1, e_1), \ldots, (l_m, e_m)]$ is the list of pending accounts. For each pair $(p, e) \in l$ we have that $p$ represents a (positive or negative) profit and $e$ represents the expiration date of $p$, that is, the time in which a profit greater than or equal to $-p$ must be retrieved, in the case that $p$ is negative, or the time in which $p$ will be considered a clear profit if no negative profit is registered before.

Next we present a few more auxiliary definitions. First, we define the maximal time a USM is allowed to wait. Intuitively, the limit will be given by the minimum between the minimal time in which the machine will have to change its state and the time in which the oldest pending account will expire. Afterwards, we will present how the list of pending accounts must be updated when a new transaction is performed. In the case that the signs of the new transaction is the same as those of the listed transactions (that is, either all of them are positive or all of them are negative) then this transaction will be added to the list. Otherwise, the value of the new transaction will compensate the listed transactions as much as its value can, from the oldest transaction to the newest. Finally, we will define how to add the profit accumulated in the pending accounts list.

Definition 2.4. Let $M = (S, s_0, V, U, at, m, T)$ be a USM and let $c = (s, \mathbf{r}, l)$ be a configuration of $M$, where we have $\mathbf{r} = (r_1, \ldots, r_n)$ and $l = [(p_1, e_1), \ldots, (p_m, e_m)]$. The maximal waiting time for $M$ in the configuration $c$, denoted by $\text{MaxWait}(M, c)$, is defined as

$$\min \left\{ \left\{ u' \mid \exists (s, Q, Z, s') \in T : u' \geq u \land Q(s, u', r_1, \ldots, r_n) = \text{True} \right\} \cup \{ e_1 \} \right\}$$

If $e_1$ is actually the minimal value and $p_1 > 0$ then we say that $M$ performs a clear profit, which is indicated by setting the true condition $\text{ClearProfit}(M, c)$.

The update of the pending accounts list $l$ with the new profit $a$, denoted by $\text{Update}(l, a)$, is defined as:

$$\text{Update}(l, a) = \left\{ \begin{array}{ll}
(a, at + u) & \text{if } l = [a] \\
(l + +a, at) & \text{if } l \neq [a] \land \frac{a}{u} \geq 0 \\
(p_1 + a, e_1) & \text{if } \frac{a}{u} < 0 \land l = (p_1, e_1) \land \frac{a}{u} \geq 0 \\
\text{Update}(l', p_1 + a) & \text{if } \frac{a}{u} < 0 \land l = (p_1, e_1) \land \frac{a}{u} < 0
\end{array} \right.$$

Finally, the accumulated profit of a given pending accounts list $l$, denoted by $\text{Accumulated}(l)$, is defined as

$$\text{Accumulated}(l) = \left\{ \begin{array}{ll}
0 & \text{if } l = [a] \\
p + \text{Accumulated}(l') & \text{if } l = (p, e) \land l'
\end{array} \right.$$

In the definition of $\text{Update}$, conditions as $\frac{a}{u} \geq 0$ denote that $n$ and $m$ have the same sign. We have used a functional programming notation to define lists: $[a]$ denotes an empty list, $l + +a$ denotes the concatenation of the lists $l$ and $l'$, and $x : l$ denotes the inclusion, as first element, of $x$ into the list $l$. Given a USM $M$, an evolution of $M$ represents a configuration that the USM can take from a previous configuration. Formally, evolutions are tuples $(c, c', tc) \in \text{Ev}$ where $c$ and $c'$ are the previous and new configurations, respectively, $tc$ is the time consumed by the evolution, and $K \in \{a, \beta, \gamma\}$ is the type of evolution (changing the state, waiting time, performing a transaction).

Definition 2.5. Let $M = (S, s_0, V, U, at, m, T)$ be a USM and $c = (s, \mathbf{r}, l)$ be a configuration of $M$, where we have $\mathbf{r} = (r_1, \ldots, r_n)$ and $l = [(p_1, e_1), \ldots, (p_m, e_m)]$. An evolution of $M$ from $c$ is a tuple $(c, c', tc) \in \text{Ev}$ where $c' = (s', \mathbf{r}', l')$, $K \in \{a, \beta, \gamma\}$ and $tc \in \mathbb{R}_+$ are defined according to the following options:

776
3. SPECIFICATION OF SYSTEMS

In the previous section we have defined the full set of valid transitions a USM can perform. Thus, the definition of the formal specification framework for a single agent is complete. At this point it may be interesting to extend the previous framework so that we can define systems made of USMs interacting among them. This notion will allow us to represent e-commerce multi-agent environments. In short, a specification of a system of USMs is a system where all the specifications of each single agent are fulfilled. Therefore, it defines the behavior of an ideal environment where all the agents fulfill their objectives. A system specification will be useful to compare its behavior with that of a real system made of real agents whose reliability is to be checked. So, we will be able to check the reliability of agents on the basis of their interactions with other previously known agents.

**Definition 3.1.** Let \( M_1, \ldots, M_m \) be USMs such that for any \( 1 \leq i \leq m \) we have \( M_i = (S_i, s_{0i}, E, V_i, U_i, a_{t_i}, m_{i}, T_i) \). We say that the tuple \( S = (M_1, \ldots, M_m) \) is a system of USMs. Let \( c_i \) be the configuration of \( M_i \) for any \( 1 \leq i \leq m \). We say that \( c = (c_1, \ldots, c_m) \) is the configuration of \( S \).

The transitions of a system will not be the simple addition of the transitions of each USM within the system, as some of the actions a USM can perform will require synchronization with those of other USMs. This will be the case of waiting times and transactions. In the former case, the system will be able to wait any amount of time provided that all of the USMs can wait this time. This does not constrain the capacity of agents to wait for long times, as a long wait could be denoted by various wait steps. When a wait is performed, the time consumption will affect equally to all the USMs in the system. Regarding the synchronization of transactions, we will suppose that they are conservative in the sense that the total amount of resources existing in a system remains invariant before and after a transaction is performed. The only actions that do not need to synchronize are the ones associated with changing the state of a USM. In contrast with transactions and waits, changing the state is not a voluntary action. So, every time a USM would do it the system will perform the corresponding transition.

In the following definition we identify the set of evolutions of a system can perform from a given configuration. However, this set will not be still valid. As we will see below, we will have to remove some of the evolutions. This is the reason why we introduce the notion of auxiliary evolutions.

**Definition 3.2.** Let \( S = (M_1, \ldots, M_m) \) be a system and \( c = (c_1, \ldots, c_m) \) be a configuration, such that for any \( 1 \leq i \leq m \) we have \( M_i = (S_i, s_{0i}, E, V_i, U_i, a_{t_i}, m_i, T_i) \) and \( c_i = (s_{0i}, T_i, k_i) \). An auxiliary evolution of \( S \) from \( c \) is a tuple \((c, c', \tau, k)\) where \( c' = (c_1', \ldots, c_m') \) with \( c_i' = (s_i', T_i', l_i') \), \( k \in \{\alpha, \beta, \gamma\} \), and \( \tau \in \mathbb{R}_+ \) are defined according to the following options:

1. **(Changing the state)** If there is \( 1 \leq i \leq m \) such that there exists \((c_i, c_i', 0)\) in \( \text{Evolve}(M_i, c_i) \) then for any \( 1 \leq j \leq m \) with \( i \neq j \) we have \( k = \alpha, c_i' = c_i, c_j' = c_j', \) and \( \tau c = 0. \)

2. **(Waiting time)** If the condition of (1) does not hold and there exists \((c_i, 0, l)\) in \( \text{Evolve}(M_i, c_i) \), then for any \( 1 \leq i \leq m \) we have \( k = \beta, c_i' = c_i, c_j' = c_j', \) and \( \tau = 0. \)

3. **(Performing a transaction)** If the condition of (1) does not hold and there exists \((c_i, c_i', 0)\) in \( \text{Evolve}(M_i, c_i) \) then for any \( 1 \leq i \leq m \) with \( i \neq j \) we have \( k = \gamma, c_i' = c_i, c_j' = c_j', \) and \( \tau = 0. \)

We denote the set of auxiliary evolutions of \( S \) from \( c \) by \( \text{AuxEvolve}(S, c) \).

An auxiliary trace of \( S \) from \( c \) is a list of auxiliary evolutions \( l \) where either \( l = [ ] \) or \( l = c : l' \), being \( c = (c, c', \tau, k) \in \text{AuxEvolve}(S, c) \) and \( l' \) an auxiliary trace of \( S \) from \( c' \). We denote the set of auxiliary traces of \( S \) from \( c \) by \( \text{AuxTrace}(S, c) \).

As we have pointed out before, the set of auxiliary evolutions does not exactly correspond with the set of valid evolutions a system can perform. Actually, this set includes evolutions which could lead the system to a configuration where the specification of the system is not fulfilled. This is so because, according to these evolutions, the system could evolve into a time-locking configuration. This situation appears if one of the USMs refuses to let the time pass. A USM could eventually do so because in the case that the time
elapses it would not match the requirement of retrieving past losses before the deadline. In this case, its set of evolutions would not include any wait evolution, as it would be contrary to its specification. As a result, time cannot elapse in the whole system and the system deadlines. Nevertheless, the specification of a system should fulfill the specifications of all of its UMMs. Therefore, those evolutions that lead the system towards a time-lock will not be included in the actual evolutions of the true specification. Intuitively, an auxiliary evolution will be invalid if it leads the system either to a timelocking configuration or to a configuration where all the available evolutions are invalid.

**Definition 3.3.** Let $\mathcal{S}$ be a system and $\mathcal{c}$ be a configuration of $\mathcal{S}$. The set of valid traces of $\mathcal{S}$ from $\mathcal{c}$, denoted by $\text{ValidTraces}(\mathcal{S}, \mathcal{c})$, is defined as the set $\text{AuxTraces}(\mathcal{S}, \mathcal{c}) \setminus \mathcal{R}$, where $\mathcal{R}$ is the minimal set containing:

- every trace $[(c_1, c_2, t_1), \ldots, (c_{n-1}, c_n, t_{n-1})]$ belonging to $\text{AuxTraces}(\mathcal{S}, \mathcal{c})$ such that there does not exist $e \in \text{AuxEvolutions}(\mathcal{S}, \mathcal{c})$, and
- every trace $[(c_1, c_2, t_1), \ldots, (c_{n-1}, c_n, t_{n-1})]$ belonging to $\text{AuxTraces}(\mathcal{S}, \mathcal{c})$ such that for any $e = (c_n, c', t') \in \text{AuxEvolutions}(\mathcal{S}, \mathcal{c})$ we have that the auxiliary trace $[(c_1, c_2, t_1), \ldots, (c_{n-1}, c_n, t_{n-1}), (c_n, c', t')]$ is in $\mathcal{R}$.

The set of valid evolutions of $\mathcal{S}$ from configuration $\mathcal{c}$, denoted by $\text{ValidEvolutions}(\mathcal{S}, \mathcal{c})$, is defined as the set $\{ (c, c', t') \mid (c, c', t') \in \text{ValidTraces}(\mathcal{S}, \mathcal{c}) \}$.

Let us remark that valid evolutions are the sole evolutions a system can perform to avoid a time-lock. Thus, they endow systems with the suitable set of evolutions to properly denote their specifications, as they describe environments where each agent fulfills its own specification.

Next we present a useful in order result to understand the behavior of transactions. We show that the utility of the agents cannot decrease as long as they remain in the same state (that is, as long as they keep the same preferences), provided that the time has no influence in the utility in that state.

**Lemma 3.4.** Let $\mathcal{S} = (M_1, \ldots, M_n)$ be a system, where we have $M_i = (S_i, s_{i0}, V_i, U_i, a_{i0}, m_i, T_i, T_{c_i})$ for any $1 \leq k \leq n$, and let $e$ be a valid trace of $\mathcal{S}$ such that $e = [(c_1, c_2, t_1), \ldots, (c_{n-1}, c_n, t_{n-1}), (c_{n}, \mathcal{c}_n, t_{n})]$, where we have $c_{i} = ((s_1, t_{i0}, [1]), \ldots, (s_m, t_{im}, [1]))$. $1 \leq i \leq n$ be such that $t_{i} = \max(t_{i}, t_{i+1})$ in trace $e$. Let us suppose that for any $1 \leq k \leq m$ we have $U(s_{k}) \geq U(s_{k+1})$. Then, for any $1 \leq k \leq m$ we have $U(s_{k})(t_{k}) \geq U(s_{k})(t_{k})$.

**Proof Sketch:** As every agent $M_i$ remains in the same state $s_{k}$ from $c_i$ to $c_{n}$, we have that no agent has changed its utility function during this period. As $\sigma \in \text{ValidTraces}(\mathcal{S})$, we have that there does not exist $M_i$ such that it reduces its long term profit, that is, all the negative profits appearing in the list of pending accounts are retrieved before the amortization time. As the final lists of pending accounts in all the agents are equal to $[1]$, then at the end there are no negative transactions to retrieve. The desired result follows from this and by taking into account that if some positive transactions were not compensated then they become clear profits.

## 4. Conclusions and Future Work

In this paper we have presented a formalism to specify the high-level economic behavior of autonomous commerce agents. This formalism considers the agents as economic entities that behave according to some preferences that can change along time depending on the available resources. We provide an operational semantics for both agents and systems of agents.

As future work, we plan to provide a testing methodology to check the validity of an implemented agent with respect to its specification. We will consider two possibilities. In the first one, the agent will be embedded in a real system and an observer will check whether the observable behavior of the agent fits into its specification. In the second one, a tester will be able to stimulate the agent to check whether the responses of the agent conform to its specification. In addition, we also plan to develop an example of our methodology using a well known e-commerce multi-agent system (e.g. Kasbah [2]). Finally, we plan to develop extensions of the formalism to consider low-level details like commerce communication protocols or more elaborated time representations (e.g., stochastic time [8]).

**Acknowledgements**

I want to thank the anonymous referees for their helpful comments in the topic of the paper.

## 5. References


