Specification and Testing of Autonomous Agents in e-commerce Systems*

Manuel Núñez, Ismael Rodríguez, and Fernando Rubio

Dept. Sistemas Informáticos y Programación
Universidad Complutense de Madrid, E-28040 Madrid. Spain.
e-mail: {mn,isrodrig,fernando}@sip.ucm.es

Abstract. This paper presents a generic formal framework to specify and test *autonomous e-commerce agents*. First, the formalism to represent the behavior of agents is introduced. The corresponding machinery to define how implementations can be tested follows. Two testing approaches are considered. The first of them, that can be called *active*, is based on stimulating the implementation under test (IUT) with a test. The peculiarity is that tests will be defined as a special case of autonomous e-commerce agent. The second approach, that can be called *passive*, consists in observing the behavior of the tested agent in an environment containing other agents. As a case study the framework is applied to the e-commerce system Kasbah.
KEYWORDS: *Formal specification and testing*; *e-commerce*; *Multi-agent systems*.

1 Introduction

One of the cutting-edge technologies leading a major evolution and revolution in e-commerce platforms are *autonomous commerce agents* (see e.g. [1,2,3,4]). Autonomous agent technologies allow to release 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. Either by making recommendations, negotiations, or transactions, the behavior of these agents has a critical repercussion in the possessions of their corresponding users. Thus, the success of autonomous agents as the leading technology in e-commerce dramatically depends on the confidence the users have in their reliability. Formal methods provide a powerful tool to validate or verify the behavior of any software system in general and e-commerce environments in particular. Following this line, some formal approaches have been proposed to check the validity of

* Parts of this paper appeared in *Specification of Autonomous Agents in e-commerce Systems* by Ismael Rodríguez et al and in *Testing of Autonomous Agents described as Utility State Machines* by Manuel Núñez et al, Lecture Notes in Computer Science vol. 3236 (pp. 30-43 and 322-336, respectively) ©2004 Springer-Verlag. This revised and expanded version appears with the permission of Springer-Verlag. Research supported by the MCyT project *MASTER* (TIC2003-07848-C02-01), the JCCL project PAC-03-001 and the Marie Curie RTN *TAROT* (MRTN-CT-2003-505121).

communications in e-commerce systems (see e.g. [5,6]). Since communications are the basis of transactions, it is critical to check whether they keep the original meaning the emitter initially gave to them. The proposed techniques 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. However, only by developing domain-specific techniques the *high-level* behavior of autonomous agents can be efficiently validated, by focusing on *whether* it fulfills its objectives rather than on checking *how* it achieves them.

The aim of this paper is to present a specific framework to specify and check the correctness of the high-level part of e-commerce applications. Intuitively, the high-level specification of an autonomous commerce agent may be summarized as *"get what the user said he wants and when he wants it."* Therefore, a proper formalism to represent this kind of specifications has to be introduced. In order to validate the behavior of these agents, novel testing techniques are needed. In this paper, two testing methodologies to check the correctness of implementations with respect to specifications are introduced. In the first one, the implementation under test (that is, the IUT is the implementation of an agent) is stimulated according to a given test case/suite. In contrast with usual testing approaches, the part of the behavior to be tested will not be *which* actions the agent performs but the *result* of these actions, that is, whether the results of the actions conform to the specific user expectations. Since low-level issues have to be abstracted, the suitable way to perform the stimulus consists in giving the tests the form of another agent, so that they interact with the IUT in the same way as a real system would do. In the second approach the correct behavior of an IUT will be tested by including the agent into an e-commerce application (typically, a multi-agent system) and then by tracing its behavior. As in the previous approach, the part of the behavior to be traced will not be which actions the agent performs, but the result of these actions in terms of gained profit.

This paper represents an extended and revised version of previous work by the authors. In [7] a preliminary specification framework was introduced. This formalism is modified in [8] to more appropriately deal with some features. This is essentially the specification language used in this paper. In [9] the *active* testing methodology is presented. In terms of related work, there are innumerable papers on e-commerce systems/architectures and agent-mediated e-commerce. This number strongly decreases when considering formal approaches. For example, [10,11] present process algebras to specify e-barter systems. Some languages have been proposed to formally describe autonomous agents [12,13]. Unfortunately, they are not oriented to the description of agents as economic entities. Regarding testing applied to e-commerce, [14,15] may be mentioned where specific case studies are considered.

The rest of the paper is structured as follows. Section 2 introduces the formalism to specify autonomous commerce agents. In Section 3 the active testing methodology is presented. Its main highlight is that, in order to test an agent, suitable agents are used as test. Section 4 briefly sketches an alternative testing approach based on observing the behavior of an agent embedded in a multi-agent

system. In Section 5 the framework is applied to the Kasbah system. Finally, Section 6 presents the conclusions and some lines for future work.

2 Specification Model

This section introduces the formalism to specify autonomous commerce agents. *Utility state machines* are an extension of classical (extended) finite state machines where information to deal with the special capabilities of agents is included. Since users have different preferences, in order to properly design agents the first step to construct the specification of the corresponding agent consists in expressing these preferences. In this paper, user's preferences in a given moment will be given by a *utility function*. These functions associate a value (a utility measure) with each possible combination of resources a user could own.

Definition 1. Let $\mathbb{R}_+ = \{x \in \mathbb{R} \,|\, x \geq 0\}$. *Vectors* in $\mathbb{R}_+^n$ (for $n \geq 2$) are usually denoted by $\bar{x}, \bar{y}, \ldots$ The vector $\bar{0}$ denotes the tuple having all the components equal to zero. Given $\bar{x} \in \mathbb{R}^n$, x_i denotes its *i-th* component. Usual arithmetic operations are extended to vectors in the natural way. Let $\bar{x}, \bar{y} \in \mathbb{R}_+^n$: $\bar{x} + \bar{y} = (x_1 + y_1, \ldots, x_n + y_n)$, $\bar{x} \cdot \bar{y} = (x_1 \cdot y_1, \ldots, x_n \cdot y_n)$, and $\bar{x} \leq \bar{y}$ if for any $1 \leq i \leq n$, $x_i \leq y_i$ holds.

If there exist n different kinds of resources then a *utility function* is any function $f : \mathbb{R}_+^n \longrightarrow \mathbb{R}_+$. □

Intuitively, $f(\overline{x}) > f(\overline{y})$ means that $\overline{x}$ is preferred to $\overline{y}$. 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 owning 6 apples or 9 oranges. 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. Alternatively, if x_2 represents the amount of money in any currency (for example in dollars) then the first user would be a customer while the second one might be a vendor. A usual assumption is that no resource is a *bad*, that is, if the amount of a resource is increased, so does the value returned by the utility function. Using a derivative expression, this property can be formally expressed as $\frac{\Delta f(x_1,\ldots,x_n)}{\Delta x_i} \geq 0$ 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 by considering a resource representing the *absence* of it.

2.1 Defining single Agents

In order to formally specify autonomous commerce agents, the different objectives of the corresponding user along time have to be considered. Thus, the formalism will provide the capability of expressing different utility functions depending on the situation. Besides, *objectives*, represented by utility functions, will change depending on the availability of resources. These events will govern the transitions between states. The new formalism, called *utility state machines*,

is inspired by the classical notion of *finite state machines*, or *Mealy machines* (see [16] for a good survey on finite state machines and their testing). In fact, utility state machines are closer to *extended finite state machines* since both mechanisms take data into account. However, there are several notorious differences. First, utility state machines have a utility function associated with each of its states. There is nothing similar in (extended) finite state machines. Second, (extended) finite state machines base their transitions on actions while utility state machines transitions do not carry/produce any action: They represent the *change* from a state s, having attached a given utility function u_1, to a state s', having a (probably) different utility function u_2.

Another important extension is that utility state machines include a notion of *time*. Actually, time influences users' preferences 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 compensate it. In fact, gaining profit in the long run may sometimes require to perform transactions that, in the short term, are detrimental. Thus, time will be used to check whether the transaction was beneficial in the long term. Time is introduced by using a mechanism inspired in [17]. However, neither that timed model or other approaches (e.g. timed automata [18]) could be used to appropriately specify autonomous commerce agents. The reason is that *usual* languages (and timed extensions in particular) do not have the possibility of dealing with utility functions.

Definition 2. The tuple $M = (S, s_{in}, V, U, at, mi, T)$ is a *utility state machine*, in short `USM`, where S is the *set of states*, $s_{in} \in S$ is the *initial state*, V is a $(n+1)$-tuple of *variables*, $U : S \longrightarrow (\mathbb{R}_+^{n+1} \longrightarrow \mathbb{R}_+)$ is a function, $at \in \mathbb{R}_+$ is the *amortization time*, $mi \in \mathbb{R}_+$ is the *maximal investment*, and 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, $Q : \mathbb{R}_+^{n+1} \longrightarrow \texttt{Bool}$ is a predicate on the set of variables, and $Z : \mathbb{R}_+^n \longrightarrow \mathbb{R}_+^n$ is a transformation over the current variables.

A *configuration* of M is a tuple $(s, \overline{r}, l)$ where $s \in S$ is the *current state* in M, $\overline{r} = (u, r_1, \ldots, r_n) \in \mathbb{R}_+^{n+1}$ is the current value of V, and we have that $l = [(p_1, e_1), \ldots, (p_m, e_m)]$ is the list of *pending accounts*. □

In the previous definition, given the tuple of variables $V = (t, x_1, \ldots, x_n)$, 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. The function U associates a utility function with each state in M. Note that utility functions depend on the availability of resources as well as on the time consumed in the current state. The *amortization time* denotes the maximal time M may stay without retrieving the profit of the negative transactions that were performed in the past. Finally, the *maximal investment* denotes the maximal amount of negative profit the agent should afford. Regarding transitions, the predicate Q must hold to launch the corresponding transition;

the function Z allows to express that some resources are created/destroyed by performing the transition.

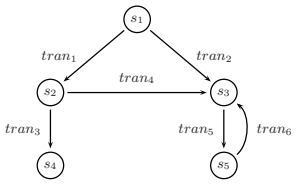
A *configuration* records the current situation of a USM. It consists of the current state, the current value of the variables, and the *pending accounting* of the machine. Each pair (p, e) belonging to the list of pending accounts represents a previous transaction that generated a positive/negative *profit* equal to p, that is, an increase/decrease of utility. The value e represents the *expiration date* of p. If $p < 0$ then a transaction producing a positive outcome to compensate this temporary loss should be performed before date e. If $p > 0$ and date e is reached without accumulating losses bigger than p then the remaining amount is considered a *clear* profit. The idea is that agents will be allowed to perform *bad* operations as long as they are compensated in the (near) past/future with good ones. There is a strong relation between pending accounts and the amortization time of M: Each new operation added to the list has its expiration date set to *amortization time units* in the future, that is, "current time + at." Besides, the amount of cumulated losses (that is, $\sum p_i$ if this value is less than zero) should not be greater (in absolute value) than the maximal investment of M.

The set of transitions T of a utility state machine does not include the complete set of *movements* the specification will allow real agents to perform. Two additional possibilities must be considered: *Transactions* and *passing of time*. The former will be used to denote the transactions agents will be allowed to perform. Passing of time denotes the free decision of agents to idle waiting for a good deal. Essentially, transactions and time consumptions have to be considered as *usual* transitions with the only difference that while transitions have to be explicitly defined (by giving the set T), transactions and time consumptions are *implicitly* inferred from the definition of M (see forthcoming Definition 4).

In order to explain the main characteristics of our framework, a simple (but illustrative) running example will be presented.

Example 1. Little Jimmy wants to sell lemonade in the market of his town. The USM $\mathcal{J} = (S, s_{in}, V, U, at, mi, T)$ will represent Jimmy's economic behavior. The tuple $V = (t, l, d, s, m)$ contains a variable to store time (t) and the amount of each resource: lemons (l), lemonades (d), selling licenses (s), and money (m).

The USM $\mathcal{J}$ is depicted in Figure 1. The initial state s_1 represents a situation where Jimmy has no license to sell lemonade in the market. Unfortunately, all stands in the market are occupied. So, Jimmy has to buy it from another vendor. The utility function in s_1 is given by $U(s_1)(t, \overline{r}) = 10 \cdot s + m$, that is, Jimmy would pay up to \$10 for a selling licence. Besides, lemons and lemonades are irrelevant at this preliminary stage. There are two possibilities for leaving the state s_1. On the one hand, if a week passes and Jimmy has not bought the selling license yet, then he will raise his effort to get the license. The transition $tran_1 = (s_1, Q_2, Z_2, s_2)$, where $Q_2(t, \overline{r}) = (t = 7)$ and $Z_2(\overline{r}) = \overline{r}$, leads Jimmy to the new state. In state s_2 we have $U(s_2)(t, \overline{r}) = 20 \cdot s + m$, which denotes that Jimmy would pay now up to \$20. On the other hand, if Jimmy gets the license then the transition $tran_2 = (s_1, Q_3, Z_3, s_3)$, where $Q_3(t, \overline{r}) = (s = 1)$ and $Z_3(\overline{r}) = (l, d, s - 1, m)$, leads him to s_3. The function Z_3 represents that Jimmy



s_1 = Initial state
s_2 = No license after one week
s_3 = Jimmy has a license
s_4 = Jimmy gives up
s_5 = Jimmy is in business

Fig. 1. Example of `USM`: Jimmy's business.

will take away the license from his trading basket because he wants to keep it for himself. The utility function in s_3 is represented by $U(s_3)(t, \overline{r}) = 0.2 \cdot l + m$. This means that Jimmy wants to stock up with lemons to make lemonades. He will pay up to 20 cents for each lemon.

If Jimmy is in state s_2 then there are two possibilities as well. If another week elapses and Jimmy remains with no license then he gives up his project. This is denoted by the transition $tran_3 = (s_2, Q_4, Z_4, s_4)$, where $Q_4(t, \overline{r}) = (t = 7)$ and $Z_4(\overline{r}) = (0, 0, 0, 0)$. The function Z_4 denotes that all resources are taken away from the trading environment. Hence, $U(s_4)(t, \overline{r})$ is irrelevant. The second possibility happens if Jimmy finally gets his license. In this case, the transition $tran_4 = (s_2, Q'_3, Z'_3, s_3)$, where $Q'_3 = Q_3$ and $Z'_3 = Z_3$, leads him to s_3.

Jimmy uses 3 lemons for each lemonade. Thus, when Jimmy is in s_3 and stocks up with 12 lemons then he makes 4 lemonades. This is represented by the transition $tran_5 = (s_3, Q_5, Z_5, s_5)$, where $Q_5(t, \overline{r}) = (l = 12)$ and $Z_5(\overline{r}) = (l - 12, d + 4, s, m)$. In the state s_5, Jimmy wants to sell his new handmade lemonades. The utility function $U(s_5)(t, \overline{r}) = 2 \cdot d + m$ means that he will sell lemonades for, at least, \$2. Finally, when lemonades run out, Jimmy thinks again about stocking up with lemons. So, the transition $tran_6 = (s_5, Q_6, Z_6, s_3)$, where $Q_6(t, \overline{r}) = (d = 0)$ and $Z_6(\overline{r}) = \overline{r}$, moves Jimmy back to s_3.

A possible initial configuration of $\mathcal{J}$ is $(s_1, (0, 0, 0, 0, 50), [\,])$, which means that Jimmy begins his adventure with \$50 and without pending accounts. □

Next some auxiliary definitions are presented. The maximal time a `USM` is allowed to idle is given by the minimum between the minimal time at which the machine has to change its state and the time at which the oldest pending account expires. If the minimum is given by the second value then two additional predicates are used. The first predicate holds if an old positive profit expires at its amortization time. In this case, the profit will be considered *clear* since it has not been used to compensate any subsequent negative profit. The second one holds in the opposite case, that is, an old negative profit expires without being compensated before. This case denotes an *undesirable* situation in which the corresponding agent does not not fulfill its economic objectives. Even though this situation will be produced according to the behavior specified by a `USM`, it will be convenient that the implementation does not show such a behavior since it is contrary to the economic objectives of the agent. In order to *update* the list of pending accounts when a new transaction is performed, if the sign of the new

transaction coincides with those of the listed transactions then this transaction is added to the list. Otherwise, the value of the new transaction will *compensate* the listed transactions as much as its value can, starting with the oldest one.

Definition 3. Let $M = (S, s_{in}, V, U, at, mi, T)$ be a `USM` and $c = (s, \overline{r}, l)$ be a configuration of M, with $\overline{r} = (u, r_1, \ldots, r_n)$ and $l = [(p_1, e_1), \ldots, (p_m, e_m)]$. The *maximal waiting time* for M in the configuration c, denoted by $\texttt{MaxWait}(M, c)$, is defined as

$$\min\{e_1, \min\{u' \mid \exists (s, Q, Z, s'') \in T : u' \geq u \ \wedge \ Q(u', r_1, \ldots, r_n) = \texttt{True}\}\}$$

If e_1 is actually the minimal value and $p_1 > 0$ then it is said that M performs a *clear profit*, which is indicated by setting true the auxiliary condition $\texttt{ClearProfit}(M, c)$. On the contrary, if e_1 is the minimal value and $p_1 < 0$ then M *fails its economic objective*, which is indicated by setting *true* the auxiliary condition $\texttt{TargetFailed}(M, c)$.

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

$$\texttt{Update}(l,a) = \begin{cases} l & \text{if } a = 0 \\ [(a, at+u)] & \text{if } a \neq 0 \ \wedge \ l = [\,] \\ l \mathbin{+\!\!+} [(a, at+u)] & \text{if } a \neq 0 \ \wedge \ l = (p_1, e_1) : l' \ \wedge \ \frac{p_1}{a} \geq 0 \\ l' & \text{if } a \neq 0 \ \wedge \ l = (p_1, e_1) : l' \ \wedge \ \frac{p_1}{a} < 0 \ \wedge \ \frac{p_1 + a}{p_1} = 0 \\ (p_1 + a, e_1) : l' & \text{if } a \neq 0 \ \wedge \ l = (p_1, e_1) : l' \ \wedge \ \frac{p_1}{a} < 0 \ \wedge \ \frac{p_1 + a}{p_1} \geq 0 \\ \texttt{Update}(l', p_1 + a) & \text{if } a \neq 0 \ \wedge \ l = (p_1, e_1) : l' \ \wedge \ \frac{p_1}{a} < 0 \ \wedge \ \frac{p_1 + a}{p_1} < 0 \end{cases}$$

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

$$\texttt{Accumulated}(l) = \begin{cases} 0 & \text{if } l = [\,] \\ p + \texttt{Accumulated}(l') & \text{if } l = (p, e) : l' \end{cases}$$

□

In order to not overload the notation, the values at and u (amortization and current time, respectively) do not explicitly appear as parameters of `Update`. These values are taken from M and the current configuration c. In the definition of `Update`, conditions such as $\frac{n}{m} \geq 0$ denote that n and m have the same sign. As said before, if a (i.e. the new profit) has the same sign as (all) the profits in the list (e.g. the sign of p_1) then the new transaction is added to the list. On the contrary, if the signs are opposite then a compensates older profits, starting with p_1, as much as its value can. A functional programming notation has been used: $[\,]$ denotes an empty list, $l \mathbin{+\!\!+} l'$ denotes the concatenation of the lists l and l', and $x : l$ denotes the addition, as first element, of x into l.

Example 2. After a month, Jimmy is in configuration $(s_5, (31, 0, 4, 0, 8.50), [\,])$. Today, a lemonade falls from his stand to the floor and spreads around. Suppose

that Jimmy's amortization time is seven days. Then, the new configuration is $(s_5, (31, 0, 3, 0, 8.50), [(-2, 38)])$, where there is a loss of utility equal to 2 that has to be compensated in the next seven days ($38 = 31 + 7$). Bad news go on because the next day \$1 disappears from Jimmy's shoes box. So, the current configuration is $(s_5, (32, 0, 3, 0, 7.50), [(-2, 38), (-1, 39)])$. Two days later, Jimmy has a lucky strike: He finds a \$5 bill that compensates old losses. His new configuration is $(s_5, (34, 0, 3, 0, 12.50), [(2, 41)])$. All these events do not change the state where $\mathcal{J}$ remains (that is, s_5). Thus, the time counter is not reset to 0. □

Given a `USM` M, an *evolution* represents a configuration that the `USM` can reach. Formally, evolutions are tuples $(c, c', tc)_K$ where c and c' are the previous and new configuration, respectively, tc is the time consumed by the evolution, and K is the type of evolution.

Definition 4. Let $M = (S, s_{in}, V, U, at, mi, T)$ be a `USM` and $c = (s, \overline{r}, l)$ be a configuration of M, with $\overline{r} = (u, r_1, \ldots, r_n)$ and $l = [(p_1, e_1), \ldots, (p_m, e_m)]$.

An *evolution* of M from c is a tuple $(c, c', tc)_K$ where $c' = (s', \overline{r'}, l')$, $K \in \{\alpha, \beta, \beta', \gamma\}$ and $tc \in \mathbb{R}_+$ are defined according to the following options:

(1) (*Changing the state*) If there exists $(s, Q, Z, s'') \in T$ such that $Q(\overline{r})$ holds then let $K = \alpha$, $tc = 0$, $s' = s''$, $\overline{r'} = (0, r'_1, \ldots, r'_n)$, where $(r'_1, \ldots, r'_n) = Z(r_1, \ldots, r_n)$, and $l' = [(p_1, e_1 - u), \ldots, (p_m, e_m - u)]$.

(2) (*Passing of time*) If the condition of (1) does not hold then for any value $0 < tr \leq \texttt{MaxWait}(M, c) - u$, let $tc = tr$, $s' = s$, $\overline{r'} = (u + tr, r_1, \ldots, r_n)$, and

$$l' = \begin{cases} [(p_2, e_2), \ldots, (p_m, e_m)] & \text{if } \texttt{ClearProfit}(M, c) \vee \texttt{TargetFailed}(M, c) \\ l & \text{otherwise} \end{cases}$$

In addition, $K = \beta'$ if $\texttt{TargetFailed}(M, c)$ and $K = \beta$ otherwise.

(3) (*Transaction*) If the condition of (1) does not hold then for any $\overline{r''} = (u, r''_1, \ldots, r''_n) \geq \overline{0}$ such that $U(s)(\overline{r''}) - U(s)(\overline{r}) + \texttt{Accumulated}(l) > -mi$, let $K = \gamma$, $tc = 0$, $s' = s$, $\overline{r'} = \overline{r''}$, and $l' = \texttt{Update}(l, U(s)(\overline{r'}) - U(s)(\overline{r}))$.

$\texttt{Evolutions}(M, c)$ denotes the set of evolutions of M from c.

A *trace* of M from c is a list of evolutions l with either $l = [\,]$ or $l = e : l'$, where $e = (c, c', v)_K \in \texttt{Evolutions}(M, c)$ and l' is a trace of M from c'. $\texttt{Traces}(M, c)$ denotes the set of traces of M from c.

The trace $\sigma = [(c, c_1, t_1)_{K_1}, \ldots, (c_{n-1}, c_n, t_n)_{K_n}] \in \texttt{Traces}(M, c)$ is a *valid trace* of M from c if there does not exist $1 \leq i \leq n$ such that $K_i = \beta'$. The set of valid traces of M from c is denoted by $\texttt{ValidTraces}(M, c)$. □

In the previous definition, if one of the guards associated with a transition between states holds then the state changes. In addition, the time counter of the state is reset to 0 and the expiration dates of the pending accounts are shifted to fit the new counter. The second clause reflects the situation where the machine let the time pass. In this case, the amount of time has to be less than or equal to the maximal waiting time. Besides, if the elapsed time is the one in which a positive or negative previous profit expires, then either this profit is considered

clear or a *failure* is produced, respectively, being such profit eliminated from the list. In the second case, the transition is labelled by β' to denote an undesirable behavior of the agent associated to the corresponding `USM`. If a machine performs a transaction then two conditions are required: It does not give resources that it does not own and the accumulated losses stay below the maximal threshold. When a transaction is executed then the pending accounts are updated to register the new transaction. It is worth to point out that the second and third types of transition can be performed only if the corresponding `USM` cannot change its state. Finally, a trace is simply a list of catenated evolutions while a valid trace is a trace where no evolution is labelled by β'.

2.2 Systems: Composing Agents

In this section the previous framework is extended so that *systems*, made of `USMs` interacting among them, can be defined. This notion allows to represent e-commerce multi-agent environments. Intuitively, systems will be defined just as tuples of `USMs`, while the configuration of a system will be given by the tuple of configurations of each `USM`. Even though the main aim is to specify and test single agents, understanding how a system of agents evolves is the key to developing the forthcoming testing methodology. In particular, the concurrent evolution of agents is required to design tests since tests will be also represented by `USMs`.

Definition 5. Let $M_1, \ldots, M_m$ be `USMs`. Then, the tuple $S = (M_1, \ldots, M_m)$ is a *system* of `USMs`. For any $1 \leq i \leq m$, let c_i be the configuration of M_i. Then, the tuple $c = (c_1, \ldots, c_m)$ is the *configuration* of S. □

The evolutions of a system will not be the simple addition of the ones of each `USM` within the system. This is so because some of the actions a `USM` can perform will require *synchronization* with those performed by other `USMs`. This will be the case of passing of time and transactions. In the former case the system will be able to idle an amount of time t provided that all of the `USMs` can idle t units of time. This does not constrain the capacity of agents to idle long periods since a long period of time can be denoted by several *time steps*. Regarding synchronization of transactions, they are *conservative*, that is, the total amount of resources existing in a system remains invariant after a transaction is performed. The only evolutions that do not require a synchronization are the ones associated with changing the state of a `USM`. In contrast with transactions and passing of time, changing the state is not a *voluntary* action. In other words, if the condition for a `USM` to change its state holds then that `USM` must change its state; transactions and passing of time will be forbidden.

Once again, an evolution may be a changing of state, a passing of time, or a transaction. In order to make explicit the *failure* of any of the `USMs` in the system, the set of `USMs` producing the failure will be explicitly indicated. Hence, β_A will denote a passing of time transition in which the `USMs` belonging to A produced a failure. Thus, $\beta_\emptyset$ denotes a failure-free passing of time transition.

Definition 6. Let $S = (M_1, \ldots, M_m)$ be a system and $c = (c_1, \ldots, c_m)$ be a configuration such that for any $1 \leq i \leq m$, $M_i = (S_i, s_{in\ i}, V, U_i, at_i, mi_i, T_i)$ and $c_i = (s_i, \overline{r_i}, l_i)$.

An *evolution* of S from c is a tuple $(c, c', tc)_K$ where $c' = (c'_1, \ldots, c'_m)$, with $c'_i = (s'_i, \overline{r'_i}, l'_i)$, $K \in \{\alpha, \gamma\} \cup \{\beta_A \mid A \subseteq \{1, \ldots, m\}\}$, and $tc \in \mathbb{R}_+$ are defined according to the following options:

(1) (*Changing the state*) If there exists $(c_i, c''_i, 0)_\alpha \in \texttt{Evolutions}(M_i, c_i)$ then let $K = \alpha$, $tc = 0$, and for any $1 \leq j \leq m$, if $i \neq j$ then $c'_j = c_j$ else $c'_j = c''_i$.
(2) (*Passing of time*) If the condition of (1) does not hold and there exits tr such that for any $1 \leq i \leq m$ there exists $(c_i, c''_i, tr)_\delta \in \texttt{Evolutions}(M_i, c_i)$ with $\delta \in \{\beta, \beta'\}$, then let $tc = tr$, $c'_i = c''_i$ for any $1 \leq i \leq m$, and $K = \beta_A$ where $A = \{i \mid (c_i, c''_i, tr)_{\beta'} \in \texttt{Evolutions}(M_i, c_i)\}$.
(3) (*Transaction*) If the condition of (1) does not hold and there exist two indexes j and k, with $j \neq k$, such that for $l \in \{j, k\}$, $(c_l, c''_l, 0)_\gamma \in \texttt{Evolutions}(M_l, c_l)$ and $c''_l = (s''_i, \overline{r''_i}, l''_i)$, with $\overline{r_j} + \overline{r_k} = \overline{r''_j} + \overline{r''_k}$, then let $K = \gamma$, $tc = 0$, and for any $1 \leq i \leq m$, if $i = j$ then $c'_i = c''_j$ else if $i = k$ then $c'_i = c''_k$ else $c'_i = c_i$.

$\texttt{Evolutions}(S, c)$ denotes the set of evolutions of S from c.

A *trace* of S from c is a list of evolutions l where either $l = [\,]$ or $l = e : l'$, being $e = (c, c', v)_K \in \texttt{Evolutions}(S, c)$ and l' a trace of S from c'. $\texttt{Traces}(S, c)$ denotes the set of traces of S from c .

The trace $\sigma = [(c, c_1, t_1)_{K_1}, \ldots, (c_{n-2}, c_{n-1}, t_{n-1})_{K_{n-1}}] \in \texttt{Traces}(S, c)$ is a *valid trace* of S from c for the USM M_i if there does not exist $1 \leq j \leq n-1$ such that $K_j = \beta_A$, with $i \in A$. The set of valid traces of S from c for M_i is denoted by $\texttt{ValidTraces}(S, c, M_i)$. □

If a USM can change its state then the corresponding transition is performed while any other USM remains unmodified. If no USM can modify its state then passing of time and transactions are enabled. In the first case the system can let the time pass provided that all the USMs belonging to the system can. In the second case it is required that trading USMs can (individually) perform the exchange and that goods are not created/destroyed as a result of the exchange. Note that only bilateral transactions are considered since any multilateral exchange can be expressed by means of the concatenation of some bilateral transactions. Similarly to the notion presented for single agents, the valid traces of systems can be identified. In this case, it is more convenient to get the traces such that a specific USM belonging to the system produces no failure.

Example 3. Little Billy plays all the day. This activity turns him thirsty. Billy's behavior is given by $\mathcal{B} = (S', s'_1, V, U', at, mi, T')$. Even though the variables considered by Billy coincide with Jimmy's (i.e. $V = (t, l, d, s, m)$), he is interested neither in lemons nor in selling licenses.

First, $S' = \{s'_1, s'_2\}$. If Billy is in s'_1 then he just plays. Since he does not need to spend his money, $U'(s'_1)(t, \overline{r}) = m$. After a quarter of day of hard playing he turns thirsty. The transition $tran_1 = (s'_1, Q'_1, Z'_1, s'_2)$ moves him to the state s'_2, with $Q'_1(t, \overline{r}) = (t \geq 0.25)$ and $Z'_1(\overline{r}) = \overline{r}$. In s'_2 Billy is thirsty and he wants to

buy a lemonade. Actually, he would pay more as he turns thirstier. Thus, he will have a utility function such as $U'(s'_2)(t, \overline{r}) = 25 \cdot t \cdot d + m$. This means, for example, that Billy would pay up to \$2.50 for a lemonade after waiting a tenth part of a day. After Billy gets a lemonade he drinks it and goes back to play. Transition $tran_2 = (s'_2, Q'_2, Z'_2, s'_1)$ denotes such behavior, where $Q'_2(t, \overline{r}) = (d = 1)$ and $Z'_2(\overline{r}) = (l, d-1, s, m)$.

Jimmy can be *composed* with his friend Billy. More characters, such as the seller that provides Jimmy with the selling license or the lemons retailer, could be included. However, for the sake of clarity, the system is defined simply as $\mathcal{S} = (\mathcal{J}, \mathcal{B})$. Jimmy's initial configuration is $c_{\mathcal{J}} = (s_5, (0, 0, 4, 0, 10), [\,])$, being ready to sell lemonades. Billy's initial configuration is $c_{\mathcal{B}} = (s'_1, (0, 0, 0, 0, 20), [\,])$. Hence, the initial configuration of the system $\mathcal{S}$ is $c_{\mathcal{S}} = (c_{\mathcal{J}}, c_{\mathcal{B}})$. An example of trace $\sigma \in \texttt{Traces}(\mathcal{S}, c_{\mathcal{S}})$ denoting a possible behavior of that system is $\sigma = [(c_{\mathcal{S}}, c_1, 0.25)_{\beta_\emptyset}, (c_1, c_2, 0)_\alpha, (c_2, c_3, 0.1)_{\beta_\emptyset}, (c_3, c_4, 0)_\gamma, (c_4, c_5, 0)_\alpha]$, for some system configurations $c_1, \ldots, c_5$. The first evolution $(c_{\mathcal{S}}, c_1, 0.25)_{\beta_\emptyset}$ denotes that 0.25 days pass. Then, $(c_1, c_2, 0)_\alpha$ means that $\mathcal{B}$ changes its state from s'_1 to s'_2 because after 0.25 days Billy turns thirsty and the guard to change the state from s'_1 to s'_2 holds. The evolution $(c_2, c_3, 0.1)_{\beta_\emptyset}$ denotes that 0.1 more days pass. After that time Jimmy sells a lemonade to Billy for \$2.10. The evolution $(c_3, c_4, 0)_\gamma$ denotes such a transaction. Immediately after Billy buys the lemonade, he drinks it and goes back to play. Thus, $(c_4, c_5, 0)_\alpha$ denotes the modification of the state of $\mathcal{B}$ from s'_2 to s'_1. In this simple example Jimmy's state does not change along σ because Jimmy sells a single lemonade but all lemonades must run out for changing the state from s_5 to s_3. The final configuration of $\mathcal{S}$ after σ is $c_5 = (c'_{\mathcal{J}}, c'_{\mathcal{B}})$, where $c'_{\mathcal{J}} = (s_5, (0.35, 0, 3, 0, 12.10), [(0.10, 7.35)])$ and $c'_{\mathcal{B}} = (s'_1, (0, 0, 0, 0, 17.90), [(0.40, 7)])$. The remaining configurations are:

$$
\begin{aligned}
c_1 = (c^1_{\mathcal{J}}, c^1_{\mathcal{B}}) \text{ where } & c^1_{\mathcal{J}} = (s_5, (0.25, 0, 4, 0, 10), [\,]) \\
& c^1_{\mathcal{B}} = (s'_1, (0.25, 0, 0, 0, 20), [\,]) \\
c_2 = (c^2_{\mathcal{J}}, c^2_{\mathcal{B}}) \text{ where } & c^2_{\mathcal{J}} = (s_5, (0.25, 0, 4, 0, 10), [\,]) \\
& c^2_{\mathcal{B}} = (s'_2, (0, 0, 0, 0, 20), [\,]) \\
c_3 = (c^3_{\mathcal{J}}, c^3_{\mathcal{B}}) \text{ where } & c^3_{\mathcal{J}} = (s_5, (0.35, 0, 4, 0, 10), [\,]) \\
& c^3_{\mathcal{B}} = (s'_2, (0.10, 0, 0, 0, 20), [\,]) \\
c_4 = (c^4_{\mathcal{J}}, c^4_{\mathcal{B}}) \text{ where } & c^4_{\mathcal{J}} = (s_5, (0.35, 0, 3, 0, 12.10), [(0.10, 7.35)]) \\
& c^4_{\mathcal{B}} = (s'_2, (0.10, 0, 1, 0, 17.90), [(0.40, 7.10)])
\end{aligned}
$$

□

The following result is useful to understand the behavior of transactions. It shows that the utility of the agents that do not fail 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 on the utility in that state.

Lemma 1. Let $S = (M_1, \ldots, M_m)$ be a system where for any $1 \leq j \leq m$, $M_j = (S_j, s_{j\ in}, V_j, U_j, at_j, mi_j, T_j)$. Let $\sigma = [(c_1, c_2, t_1)_{K_1}, \ldots, (c_{n-1}, c_n, t_{n-1})_{K_{n-1}}]$ be

a trace of S where for any $1 \leq j \leq n$, $c_j = ((s_1^j, \overline{r_1^j}, l_1^j), \ldots, (s_m^j, \overline{r_m^j}, l_m^j))$. Suppose that for some $1 \leq k \leq m$, $l_k^n = [\,]$ and let $1 \leq i \leq n$ be such that $l_k^i = [\,]$ and for any $i \leq j \leq n$, $s_k^j = s_k^i$ and $K_j \notin \{\beta_A \mid k \in A\}$. Finally, suppose that $\frac{\Delta U_k(s_k)(\overline{x})}{\Delta t} = 0$. Under these conditions, $U(s_k)(\overline{r_k^n}) \geq U(s_k)(\overline{r_k^i})$.

Proof. The agent M_k stays in the same state s_k^i in the evolution from c_i to c_n. Thus, its utility function does not change during this period. In addition, since there does not exist K_j, with $i \leq j \leq n$, such that $K_j = \beta_A$, with $k \in A$, the agent M_k does not fail in the evolution from c_i to c_n. Suppose $U(s_k)(\overline{r_k^n}) < U(s_k)(\overline{r_k^i})$. Since time does not affect the utility function of k, because $\frac{\Delta U_k(s_k)(\overline{x})}{\Delta t} = 0$, the reduction of utility must be produced by a transaction. So, the set $Q \subseteq \{i, \ldots, n\}$ such that for any $j \in Q$, $K_j = \gamma$ and $U(s_k)(\overline{r_k^j}) < U(s_k)(\overline{r_k^{j-1}})$ cannot be empty. For each index $j \in Q$ the effect of such negative transaction will be immediately included in its list of pending accounts l_k^j. However, since $l_k^n = [\,]$, such a profit disappears from the list before the *n-th* transition is performed. So, there are two possibilities: Either the negative profit expired or it was compensated before achieving c_n. The first situation is impossible because M_k did not fail. Besides, if the negative profit was compensated then it cannot negatively influence the final utility of M_k. Thus, $U(s_k)(\overline{r_k^n}) < U(s_k)(\overline{r_k^i})$ is not possible. □

3 Testing Autonomous Commerce Agents

This section describes how to *interact* with an IUT so that the observed behavior is as helpful as possible to infer conclusions about its validity with respect to a specification. The idea is to create *artificial* environments to stimulate the IUT according to a foreseen plan. These environments will be the *tests* in the testing process. Since tests will be defined in the same way as specifications, its definition will focus only on the high-level economic behavior.

As it is usually the case in formal approaches to testing, the design of tests will be guided by the specification. The main aim of a test will be to check whether the IUT exchanges items as the specification says. It is worth to point out that the behaviors of the specification producing a *failure* (that is, a β' transition) will not be considered as valid in the implementation since they should be avoided. Each test will focus on testing a given state (or set of states). So, the first objective of a test will be to *conduct* the IUT to the desired state of the specification. Then, the test will behave according to a given economic behavior. Thus, it has to be checked whether the joint evolution of the test and the IUT conforms to the (failure free) behavior of the specification of the system containing both agents.

Conducting the IUT to a desired state consists in taking it through states *equivalent* to those of the specification until that state is reached. In order to perform this task the utility functions of the test must be designed so that they favor the correct flow of resources, that is, the test must *get happier* by giving

resources to the IUT in such a way that the IUT fulfills its guards. Nevertheless, to endow the test with the suitable utility functions is not enough to guarantee that the test and the IUT will *always* evolve together until the corresponding guards are fulfilled. This is so because the specification gives freedom to the IUT so that it can non-deterministically perform transactions. Thus, the joint evolution of the IUT and the test could lead the IUT to own different baskets of resources to those expected. Since deviating from the foreseen path of the test is not incorrect in general, tests will be designed so that they provide diagnosis results even when the desired path is not followed by the IUT. In order to define a specific test, the first decision consists in fixing the set of states it will be intended to guide the IUT through.

Definition 7. Let $M = (S, s_{in}, V, U, at, mi, T)$ be a `USM`. A *path of states* in M is a list $[s_1, \ldots, s_m]$ where $s_1 = s_{in}$ and for each $1 \leq i \leq m-1$ there exists $(s_i, Q, Z, s_{i+1}) \in T$. □

The following definition introduces a notion to check whether a `USM` can evolve across a given path. Before, some auxiliary notation is given, in particular, how to access some elements within a list of tuples. Given a list l, $l.i$ denotes a list formed by the *i-th* element of each tuple appearing in l, while l_i denotes the *i-th* element of the list. For example, if $l = [(1,2,3),(3,4,1),(5,2,7),(8,1,6),(6,5,4)]$ then $l.2 = [2,4,2,1,5]$ and $l_3 = (5,2,7)$. Moreover, both functions can be combined. For instance, $(l.2)_4 = 1$.

Definition 8. Let M be a `USM`, c be a configuration of M, and $\sigma \in \texttt{Traces}(M,c)$ be a trace of M from c. The list of *state transitions* through σ, denoted by $\texttt{StateTran}(\sigma)$, is defined as

$$\texttt{StateTran}(\sigma) = \begin{cases} [\,] & \text{if } \sigma = [\,] \\ (s, \overline{r}, \overline{r'}) : \texttt{StateTran}(\sigma') & \text{if } \sigma = ((s, \overline{v}, l), (s', \overline{v'}, l'), 0)_\alpha : \sigma' \\ \texttt{StateTran}(\sigma') & \text{if } \sigma = e_K : \sigma' \ \wedge \ K \neq \alpha \end{cases}$$

where $\overline{r} = (r_1, \ldots, r_n)$ and $\overline{r'} = (r'_1, \ldots, r'_n)$, with $\overline{v} = (t, r_1, \ldots, r_n)$ and $\overline{v'} = (t', r'_1, \ldots, r'_n)$.

Let l be a list of n-tuples. For any $i \in \mathbb{N}$, with $i \geq 1$, such that l has at least i elements, l_i denotes the i-th element of l. For any $1 \leq i \leq n$, the list $l.i$ is defined as $[\,]$ if $l = [\,]$ and as $x_i : (l'.i)$ if $l = (x_1, \ldots, x_i, \ldots, x_n) : l'$.

The path $\eta = [s_1, \ldots, s_m]$ in M is *possible* from c if there exists a trace $\sigma = [(c_1, c_2, t_1)_{K_1}, \ldots, (c_{n-1}, c_n, t_{n-1})_{K_{n-1}}] \in \texttt{ValidTraces}(M,c)$ such that $\eta = \texttt{StateTran}(\sigma).1 +\!+ [st_n]$, where $c_n = (st_n, \overline{r_n}, l_n)$. In this case, $\texttt{StateTran}(\sigma).2$ is a *possible use of resources to perform* η and σ is a *possible trace to perform* η. □

After identifying a possible use of resources to perform a path, a test giving to the IUT the resources needed to perform the next transition of the path at every state can be defined. Basically, the test will be initially provided with the resources needed to lead the IUT to the final state. These can be computed by

using the notion of *possible use of resources*. Besides, the test will be built in such a way that each state in the path will have a respective state in the test. In each of these states, the utility function of the test will induce that the test does not care about giving to the IUT as much resources as it needs so that it may pass to the next state. In order to do so, that utility function will return the same utility regardless of the amount of items it owns, provided that it keeps the needed *reserve* of resources. This reserve of resources consists of those resources the test will give to the IUT in future states to continue guiding it towards the final state of the path. The only exception will be the case when some of the resources of the IUT must be *removed* in order to fulfill the transition guard. In order to *take* the appropriate resources from the IUT, the test will reward these items in its utility function. The following definition describes how to create the utility function of the test so that it promotes an exchange between the resources the test wants to remove from the IUT and those it wants to give to it. In this case, $\overline{v}$ plays the role of the former and $\overline{w}$ plays the role of the latter. It is worth to point out that each kind of resource will have to be either given or removed, but not both.

Definition 9. Let $\overline{v}, \overline{w} \in \mathbb{R}_+^n$ such that $\overline{v} \cdot \overline{w} = \bar{0}$. Let $\rho, \varrho \in \mathbb{R}_+$ be such that $\rho > \varrho$. If $\overline{v} \neq \overline{0}$ then the *function denoting preference of* $\overline{v}$ *over* $\overline{w}$, denoted by $\Psi_{\overline{v},\overline{w}}$, is defined, for any $\overline{x} \in \mathbb{R}_+^n$, as

$$\begin{aligned}\Psi_{\overline{v},\overline{w}}(\overline{x}) = \; & \rho \cdot \Phi\left(\textstyle\sum_{1\leq i\leq n} \Phi(x_i, v_i),\ |\,\{i \mid v_i > 0\}\,|\right) + \\ & \varrho \cdot \Phi\left(\textstyle\sum_{1\leq i\leq n} \Phi(x_i, w_i),\ |\,\{i \mid w_i > 0\}\,|\right)\end{aligned}$$

where if $b = 0$ then $\Phi(a, b) = 0$ else $\Phi(a, b) = \frac{a}{b}$. If $\overline{v} = \overline{0}$ then $\Psi_{\overline{v},\overline{w}}(\overline{x}) = 0$. □

Lemma 2. Let $\overline{v}, \overline{w} \in \mathbb{R}_+^n$ such that $\overline{v} \neq \bar{0}$ and $\overline{v} \cdot \overline{w} = \bar{0}$. Then $\Psi_{\overline{v},\overline{w}}(\overline{v}) > \Psi_{\overline{v},\overline{w}}(\overline{w})$.

Proof. In order to compute $\Psi_{\overline{v},\overline{w}}(\overline{v})$, note that $\sum_{1\leq i\leq n} \Phi(v_i, v_i) = |\,\{i \mid v_i > 0\}\,|$. This is so because the addition in the left hand side adds 1 for each $v_i > 0$. Hence, $\Phi(\sum_{1\leq i\leq n} \Phi(v_i, v_i),\ |\,\{i \mid v_i > 0\}\,|) = 1$. Thus, the first additive factor in the expression $\Psi_{\overline{v},\overline{w}}(\overline{v})$ is ρ. Besides, since for any $1 \leq i \leq n$ either $v_i = 0$ or $w_i = 0$, for any $1 \leq i \leq n$, $\Phi(v_i, w_i) = 0$. So, $\sum_{1\leq i\leq n} \Phi(v_i, w_i) = 0$ holds. Then, the second additive factor in the expression $\Psi_{\overline{v},\overline{w}}(\overline{v})$ is 0, so $\Psi_{\overline{v},\overline{w}}(\overline{v}) = \rho$. By using similar arguments, $\Psi_{\overline{v},\overline{w}}(\overline{w}) = \varrho$ and, since $\rho > \varrho$, $\Psi_{\overline{v},\overline{w}}(\overline{v}) > \Psi_{\overline{v},\overline{w}}(\overline{w})$. □

The previous result implies, in particular, that the use of the function $\Psi_{\overline{v},\overline{w}}$ as a component of the utility function of the test favors the appropriate exchange of resources between the test and the IUT. In particular, if $\Psi_{\overline{v},\overline{w}}$ is *adequately* inserted in the utility function of the test, so that $\overline{v}$ represents the resources the test must take from the IUT and $\overline{w}$ represents those it will give to the IUT, then the foreseen exchange improves the utility of the test. This is so because its utility function returns a higher value with $\overline{v}$ than with $\overline{w}$. Besides, if the exchange of $\overline{v}$ by $\overline{w}$ is represented in a *possible use of resources* of the specification then the specification will fulfill its guard to change its state by performing that exchange.

As previously said, each state of the test will represent the corresponding state of the specification in the path. Besides, in order to properly track the IUT, the test must change its state exactly when the specification is supposed to do it. Therefore, the guards of the transitions of the tests will hold exactly when the corresponding guard of the specification does. Since the total amount of resources in the system is known, the test can trivially compute the resources the IUT has by subtracting its own resources from the total amount and by accounting the resources introduced or removed from the market in previous transitions. Thus, the guards of the test can be easily defined in terms of its own resources. In the next definition, $\widehat{U}$ and $\widehat{r}$ denote, respectively, the utility function and the resources the test will apply in its last state.

Definition 10. Let $M = (S, s_{in}, V, U, at, mi, T)$ be a $\texttt{USM}$, $c = (s_{in}, \overline{r_{in}}, [\,])$ be a configuration of M, $\eta = [s_1, \ldots, s_m]$ be a possible path of M from c, and $\sigma \in \texttt{ValidTraces}(M, c)$ be such that σ is a possible trace to perform η. A *test leading M through η and applying $\widehat{U}$ and $\widehat{r}$* is a $\texttt{USM}$ $\mathcal{T} = (S', s'_{in}, V, U', at, mi, T')$ where $S' = \{s'_1, \ldots, s'_m\}$, being $s'_1, \ldots, s'_m$ fresh states, $s'_{in} = s'_1$, $U'(s'_m) = \widehat{U}$, and for any $1 \leq i \leq m-1$, if $\overline{x} < \texttt{Reserve}_i$ then $U'(s'_i)(t, \overline{x}) = 0$ else $U'(s'_i)(t, \overline{x}) = \Psi_{\texttt{Take}_i, \texttt{Give}_i}(\bar{x} - \texttt{Reserve}_i)$, with

$$\begin{aligned}
\texttt{Reserve}_i &= \widehat{r} + \textstyle\sum_{i<j\leq m} \Omega((\texttt{StateTran}(\sigma).2)_j - (\texttt{StateTran}(\sigma).3)_{j-1}) \\
\texttt{Take}_i &= \Omega((\texttt{StateTran}(\sigma).3)_{i-1} - (\texttt{StateTran}(\sigma).2)_i) \\
\texttt{Give}_i &= \Omega((\texttt{StateTran}(\sigma).2)_i - (\texttt{StateTran}(\sigma).3)_{i-1})
\end{aligned}$$

where Ω returns the original tuple but substituting negative numbers by 0, and we assume $(\texttt{StateTran}(\sigma).3)_0 = \overline{r_{in}}$.

$T' = \{(s'_1, Q'_1, id, s'_2), \ldots, (s'_{m-1}, Q'_{m-1}, id, s'_m)\}$ fulfills that id is the identity function and for any $1 \leq i \leq m$, $Q'_i(t, \overline{r}) = Q_i(t, (\overline{r_{in}} + \texttt{Reserve}_0 + \texttt{Modif}_i - \overline{r}))$, where Q_i is such that $(s_i, Q_i, Z_i, s_{i+1}) \in T$ and $Q_i(t', (\texttt{StateTran}(\sigma).2)_i)$ holds for some t' and Z_i with $Z_i((\texttt{StateTran}(\sigma).2)_i) = (\texttt{StateTran}(\sigma).3)_i$, and

$$\texttt{Modif}_i = \textstyle\sum_{1\leq j<i} (\texttt{StateTran}(\sigma).3)_j - (\texttt{StateTran}(\sigma).2)_j$$

The initial configuration of $\mathcal{T}$ is given by $c' = (s'_{in}, (0, \texttt{Reserve}_0), [\,])$. □

Every state in the path η has a corresponding state in the test. However, if η contains cycles then each occurrence of the same state in η will be represented by a different state in the test. Thus, tests will not contain loops. The utility function of each state in the test will promote that the resources the test wants to give to the IUT (that is, $\texttt{Give}_i$) are less valued than those it wants to take from the IUT (that is, $\texttt{Take}_i$). Since, by construction, the preferences reflected in the specification are opposite to those of the test, the exchange will be possible if the IUT conforms to the specification. Besides, the utility function of the test will take care of keeping the needed resources for future states in the path (given by $\texttt{Reserve}_i$). This is done by setting the utility to null if these resources are not owned. The values of $\texttt{Give}_i$, $\texttt{Take}_i$, and $\texttt{Reserve}_i$ are calculated

by analyzing σ, that is, the trace to execute the path η. By comparing the available resources when arriving at the current state and when leaving it (denoted by $(\texttt{StateTran}(\sigma).3)_{i-1}$ and $(\texttt{StateTran}(\sigma).2)_i$, respectively), the resources the IUT must get/lose in each state can be computed. The transitions of the tests will be activated exactly when the specification would do it. Thus, the guards of the test (Q'_i) are defined in terms of the guards of the specification (Q_i). The expression $\overline{r_{in}}+\texttt{Reserve}_0+\texttt{Modif}_i-\overline{r}$ allows the test to determine the resources the specification owns, assuming that the test owns the basket $\overline{r}$. The reason is that $\overline{r_{in}}+\texttt{Reserve}_0$ is equal to the total amount of resources existing initially in the system, while $\texttt{Modif}_i$ denotes the subsequent variation of this amount due to the resources introduced/removed by the specification in the next transitions by its functions Z_i. The value $\texttt{Modif}_i$ is computed by comparing the tuples of resources before and after each modification of the state (denoted by $(\texttt{StateTran}(\sigma).2)_j$ and $(\texttt{StateTran}(\sigma).3)_j$, respectively). Note that the test does not modify the total amount of resources since its functions of transformation of resources are always given by the identity function.

Example 4. A test $\mathcal{T}$ to investigate whether an implementation behaves as the *ideal* Jimmy can be constructed. In particular, $\mathcal{T}$ will study the behavior of the IUT in the state s_5 of the specification. So, the first task of $\mathcal{T}$ will be to conduct the IUT to that state. The considered path is $\eta = [s_1, s_3, s_5]$. Next, a possible trace to follow that path must be identified. Suppose that the initial configuration of $\mathcal{J}$ is $c = (s_1, (0, 0, 0, 0, 50), [\,])$. In that configuration, no time has elapsed yet, Jimmy has \$50, and no pending account is registered. Consider the trace $\sigma = [(c, c_1, 3)_\beta, (c_1, c_2, 0)_\gamma, (c_2, c_3, 0)_\alpha, (c_3, c_4, 2)_\beta, (c_4, c_5, 0)_\gamma, (c_5, c_6, 0)_\alpha]$, where

$$c_1 = (s_1, (3, 0, 0, 0, 50), [\,]) \qquad c_2 = (s_1, (3, 0, 0, 1, 45), [(5, 10)])$$
$$c_3 = (s_2, (0, 0, 0, 0, 45), [(5, 7)]) \qquad c_4 = (s_2, (2, 0, 0, 0, 45), [(5, 7)])$$
$$c_5 = (s_2, (2, 12, 0, 0, 42), [(3.8, 7)]) \qquad c_6 = (s_3, (0, 0, 4, 0, 42), [(3.8, 5)])$$

The evolution $(c, c_1, 3)_\beta$ means that 3 days pass. Then, in $(c_1, c_2, 0)_\gamma$, Jimmy buys a selling license for \$5. Since he would pay up to \$10, a positive profit equal to 5 is accounted in the list of pending accounts, where the expiration date (10) is equal to the current time (3) plus the amortization time (7). When Jimmy has the license its state changes to s_3. This is denoted by the evolution $(c_2, c_3, 0)_\alpha$. Besides, the unique expiration date in the list is shifted to fit into the new time counter. The evolution $(c_3, c_4, 2)_\beta$ denotes that 2 more days pass. Then, in $(c_4, c_5, 0)_\gamma$ Jimmy buys 12 lemons for \$3.60. As its utility function says that it would be good to pay up to \$2.40, that transaction produces a loss of utility equal to 1.2, compensating part of the previous positive profit included in the list of accounts. When Jimmy has 12 lemons he uses them to create 4 lemonades and changes its state to s_5. This is denoted by the evolution $(c_5, c_6, 0)_\alpha$.

The trace σ is thus a possible trace to perform the path of states η. Hence, it can be used to create the test $\mathcal{T}$ to conduct the IUT through σ according to Definition 10. Let $\mathcal{T} = (S'', s''_1, V, U'', at, mi, T'')$ where $S'' = \{s''_1, s''_2, s''_3\}$ and the utility functions, defined by U'' for each state, are $U''(s''_1)(t, \overline{r}) = 4.95 \cdot s + m$, $U''(s''_2)(t, \overline{r}) = 0.29 \cdot l + m$, while $U''(s''_3)(t, \overline{r})$ is set to an arbitrary expression. Besides, the set of transitions is defined as $T'' = \{(s''_1, Q''_1, id, s''_2), (s''_2, Q''_2, id, s''_3)\}$,

where $Q''_1(t, \overline{r})$ holds iff $s = 0$, and $Q''_2(t, \overline{r})$ holds iff $l = 0$. Finally, the initial configuration of $\mathcal{T}$ is $c'' = (s''_1, (0, 12, 0, 1, 0), [\,])$.

The utility functions in $\mathcal{T}$ are set so that Jimmy can buy from $\mathcal{T}$ some items at the same prices that it was planned in the trace σ. Besides, note that the test plays the roles of both the selling license seller and the lemons retailer. □

The following result states that tests given by Definition 10 fulfill indeed the expected requirements. Let M be a `USM`, $\mathcal{T}$ be a test for M, and S be the system created by the composition of both. Then, there is a trace in S such that the sequence of configurations of M through that trace is the one the test $\mathcal{T}$ is intended to conduct M through.

Lemma 3. Let M be a `USM`, c_1 be a configuration of M, η be a possible path of M from c_1, and $\sigma = [(c_1, c_2, t_1)_{K_1}, \ldots, (c_{n-1}, c_n, t_{n-1})_{K_{n-1}}]$ be a possible trace to perform η. Let $\mathcal{T}$ be a test leading M through η and applying some utility function and resources $\widehat{U}$ and $\widehat{r}$, respectively, and let c' be the initial configuration of $\mathcal{T}$. Let $S = (M, \mathcal{T})$ and $c'' = (c_1, c')$ be the initial configuration of S. There exists a trace $\sigma'' = [((d_1, d'_1), (d_2, d'_2), t_1)_{K'_1}, \ldots, ((d_{m-1}, d'_{m-1}), (d_m, d'_m), t_{m-1})_{K'_{m-1}}] \in \texttt{Traces}(S, c'')$, for some configurations $d_1, \ldots, d_{m-1}$ and $d'_1, \ldots, d'_{m-1}$ of M and $\mathcal{T}$, respectively, such that after removing those d_i such that $d_i = d_{i-1}$ from the sequence $[d_1, \ldots, d_{m-1}]$, the sequence $[c_1, \ldots, c_{n-1}]$ is obtained.

Proof. Since η is a possible path of M from c_1 and σ is a possible trace to perform η, then $\sigma \in \texttt{Traces}(M, c_1)$. It has to be checked that the behavior of $\mathcal{T}$ allows the system S to perform a trace σ'' such that the behavior of M in that trace fits into σ. In particular, it must be checked whether for each evolution in σ there exists a suitable evolution of $\mathcal{T}$ such that both evolutions allow to infer the existence of the corresponding evolution in σ''. The result will be proved by induction over the length of σ.

$\sigma = [\,]$ represents the anchor case. The property trivially holds because $[\,] \in \texttt{Traces}(S, c'')$. In order to consider the inductive case, the induction hypothesis is: "for any trace $\sigma' \in \texttt{Traces}(M, c_1)$ its corresponding test $\mathcal{T}'$ allows the system $S' = (M, \mathcal{T}')$ to perform a trace σ''' such that the property holds." It has to be proved that the property also holds for the trace $\sigma = \sigma' +\!\!+ [e] \in \texttt{Traces}(M, c_1)$ and its test $\mathcal{T}$. Let c' be the initial configuration of $\mathcal{T}$, $S = (M, \mathcal{T})$, and $c'' = (c_1, c')$. First, note that $\sigma''' \in \texttt{Traces}(S, c'')$. The reason is that, according to the construction described in Definition 10, the addition of a new evolution in the *possible trace* of the test yields the creation of at most a new state and a new transition after its *former* last state. Nevertheless, the previous states (those influencing the interaction of $\mathcal{T}$ and M through σ''') remain unmodified. So, $\mathcal{T}$ allows S to perform σ''' because $\mathcal{T}'$ allows S' to do so, being M the same in S' and S. There are three possible cases for e:

(a) If $e = (f, f', 0)_\alpha$ then S can perform a trace $\sigma'' = \sigma''' +\!\!+ [e']$, where $e' = ((f, q), (f', q), 0)_\alpha$ for some configuration q of $\mathcal{T}$. This is so because a system modifies its state if any of the agents in the system can do it.

(b) If $e = (f, f', t)_\beta$ then S can perform a trace $\sigma'' = \sigma''' +\!\!+ [e']$, where $e' = ((f, q), (f', q'), t)_{\beta_A}$ for some configurations q, q' of $\mathcal{T}$ and some set A with $1 \notin A$. This is so because only the changing of state of an agent can disable the passing of time. However, M can wait for that time since e is in σ and $\mathcal{T}$ changes its state exactly when M does. Note that the guards of $\mathcal{T}$ require that its resources are the complementary of those M needs to trigger its own corresponding guards, as it is shown in Definition 10. Besides, M does not *fail* during the passing of time because $\sigma \in \texttt{ValidTraces}(M, c_1)$.

(c) If $e = (f, f', 0)_\gamma$ then S can perform a trace $\sigma'' = \sigma''' +\!\!+ [e']$, where $e' = ((f, q), (f', q'), 0)_\gamma$ for some configurations q, q' of $\mathcal{T}$. This is so because, after performing σ''', the resources $\mathcal{T}$ owns are enough to perform that exchange and the utility function of $\mathcal{T}$ allows, by Lemma 2, the exchange. Note that the original amount of resources of $\mathcal{T}$ was calculated by taking into account the resources needed to perform all transactions through the possible trace σ, as it is shown in Definition 10.

Hence, in all cases there exists a trace of the system $\sigma'' = \sigma''' +\!\!+ [e']$ that reflects the behavior of M in e. Since, by induction hypothesis, the previous evolutions of σ are reflected in σ''' then the result holds. □

Once a test is defined it can be applied to the IUT. Faults are detected by comparing the *ideal* observable behavior of a system consisting of the specification and the test with the *real* observable behavior of the real system under the influence of the test. First, it is necessary to fix the kind of observable actions. It can be assumed that the initial amount of resources of the tested agent is known. It can be also assumed that each agent provides a mechanism to observe its basket of resources. This requirement will be essential to trace the economic behavior of the system. Note that if the baskets of resources can be accessed then the performed transactions can be detected just by checking whether the baskets change. Alternatively, it could be supposed that the performed transactions are the only visible information. In this case it is also trivial to infer the baskets of resources of the agents at any time. Another event that can be detected is the passing of time. On the contrary, as tested agents are *black boxes*, it cannot be checked whether an agent changes its state. Summarizing, transactions and passing of time will be the visible events.

Definition 11. An *observable trace of a system* is a list $[e_1, \ldots, e_m]$ where for any $1 \leq i \leq m$ either $e_i = (t, \beta)$, with $t \in \mathbb{R}_+$, or $e_i = (\overline{x}, (a_1, a_2), \gamma)$, with $\overline{x} \in \mathbb{R}_+^n$ and $a_1, a_2 \in \mathbb{N}$. □

In the previous definition (t, β) represents the passing of t units of time while $(\overline{x}, (a_1, a_2), \gamma)$ represents a transaction between the agents M_{a_1} and M_{a_2}. In this last case the difference between the new and the old baskets of resources is equal to $\overline{x}$ in the case of the agent M_{a_1} and equal to $-\overline{x}$ in the case of the agent M_{a_2}. In order to avoid unneeded redundancies, $a_1 < a_2$ is assumed. A special label to denote a *failed* passing of time has not been included because the observation of a system does not explicitly provide such information. In order to detect a

fault it is necessary to check whether the observed behavior matches a possible behavior of the specification of that system in which such an agent produces no failure. This can be done by checking whether the observed trace belongs to the set of valid traces of the specification *for that agent*, that is, those for which that agent does not fail. Obviously, it is required only that the behavior of the IUT is free of faults, as a fault of the test does not mean the incorrectness of the IUT.

Definition 12. Let $S = (M_1, \ldots, M_m)$ be a system, $1 \leq i \leq m$, and $\sigma \in \texttt{ValidTraces}(S, c, M_i)$. The *observable trace* of the trace σ for M_i, denoted by $\texttt{OTr}(\sigma, M_i)$, is defined as $\texttt{OTr'}(\sigma, M_i, 0)$, where

$$\texttt{OTr'}(\sigma, M_i, t) = \begin{cases} [(t, \beta)] & \text{if } \sigma = [\,] \\ \texttt{OTr'}(l', M_i, t) & \text{if } \sigma = (c', c'', 0)_\alpha : l' \\ \texttt{OTr'}(l', M_i, t + t') & \text{if } \sigma = (c', c'', t')_{\beta_A} : l' \wedge \\ & \quad i \notin A \\ (t, \beta) : (\overline{x}, (a_1, a_2), \gamma) : \texttt{OTr'}(l', M_i, 0) & \text{if } \sigma = (d', d'', 0)_\gamma : l' \end{cases}$$

considering $d' = ((s'_1, \overline{r'_1}, l'_1), \ldots, (s'_{a_1}, \overline{r'_{a_1}}, l'_{a_1}), \ldots, (s'_{a_2}, \overline{r'_{a_2}}, l'_{a_2}), \ldots, (s'_m, \overline{r'_m}, l'_m))$, $d'' = ((s'_1, \overline{r'_1}, l'_1), \ldots, (s'_{a_1}, \overline{r''_{a_1}}, l''_{a_1}), \ldots, (s'_{a_2}, \overline{r''_{a_2}}, l''_{a_2}), \ldots, (s'_m, \overline{r'_m}, l'_m))$, and $\overline{x} = \overline{r''_{a_1}} - \overline{r'_{a_1}}$.

Let c be a configuration of S. The *set of observable traces* of S from c for M_i, denoted by $\texttt{ObsTraces}(S, c, M_i)$, is defined as $\{\sigma' \mid \exists\, \sigma \in \texttt{ValidTraces}(S, c, M_i) : \sigma' = \texttt{OTr}(\sigma, M_i)\}$. □

A trace observed in the composition of the IUT and the test represents a *fault* if it does not belong to the set of observable traces of the system consisting of the (non-failing) specification and the test.

Definition 13. Let $S = (Spec, \mathcal{T})$ be a system, c be the initial configuration of S, and σ be an observable trace of the system consisting of the IUT and the test $\mathcal{T}$. The trace σ is a *φ-fault* of the IUT with respect to *Spec* if $\sigma \notin \texttt{ObsTraces}(S, c, Spec)$. □

The previous definition introduces a criterion to conclude that a given IUT does not conform to a specification. However, a more abstract concept to describe what a *correct* IUT is can be given. While the previous definitions assumed the existence of some observable behavior of the IUT that can be detected and traced, the next definition goes one step further by assuming that the IUT can be modelled by the same formalism used to describe the specification (in this case, `USMs`). This hypothesis, usually called *test hypothesis*, allows to define when a IUT *conforms to* a specification. The conformance relation is based on comparing the behavior of the specification with that of the IUT when each of both `USMs` is in interaction with any other set of `USMs`. This relation follows the pattern of classical conformance relations [19,20]: Since the implementation should not show *undesired* behavior, its traces (in a context) should be a subset of those for the specification (in the same context).

Definition 14. Let $S, I, M_1, \ldots, M_n$ be USMs, $c, c', c_1, \ldots, c_n$ be their initial configurations, and $\mathcal{S} = (S, M_1, \ldots, M_n)$, $\mathcal{S}' = (I, M_1, \ldots, M_n)$ be systems, having as initial configurations $\mathcal{C} = (c, c_1, \ldots, c_n)$ and $\mathcal{C}' = (c', c_1, \ldots, c_n)$. I *conforms to* S *in the context of* $M_1, \ldots, M_n$, denoted by $I\ \mathtt{confc}_{\{M_1,\ldots,M_n\}}\ S$, if $\mathtt{ObsTraces}(\mathcal{S}', \mathcal{C}', I) \subseteq \mathtt{ObsTraces}(\mathcal{S}, \mathcal{C}, S)$.

I *conforms to* S *in any context*, denoted by $I\ \mathtt{confc}_*\ S$, if for any $n \geq 1$ and USMs $M_1, \ldots, M_n$, $I\ \mathtt{confc}_{\{M_1,\ldots,M_n\}}\ S$. □

The following result, which relates faults and the previous conformance relation, is easily obtained by considering Definitions 13 and 14.

Lemma 4. Let I and S be two USMs. Let σ be a φ-fault of I with respect to S. Then, $I\ \mathtt{confc}_*\ S$ does not hold.

4 Observing an Agent inside a System of Agents

This section is devoted to describe how to *observe* the behavior of an agent being part of a system so that conclusions about its validity with respect to a specification can be inferred. Observations will consist in studying the visible behavior of the IUT within a real system. The only stimuli received by the IUT will be those the system sends to it. Thus, there will be no interaction between the IUT and the tester.

Regarding the capability to observe, two possibilities can be considered. In the first one the only information the tester can collect is the one related to the behavior of the IUT, as the rest of the system will be *opaque* to him. This option is feasible when checking the validity of the IUT on the basis of its behavior in a real system populated by other agents whose owners are not the owner of the IUT. In this case, it is not always possible to detect the actions performed by other agents, as they should be considered *private*.

Definition 15. An *observable trace of an agent* is a list $[e_1, \ldots, e_m]$ where for $1 \leq i \leq m$ either $e_i = (t, \beta)$, where $t \in \mathbb{R}_+$, or $e_i = (\overline{x}, \gamma)$, where $\overline{x} \in \mathbb{R}_+^n$.

Let M be a USM and $\sigma \in \mathtt{ValidTraces}(M, c)$. The *observable trace* of σ, denoted by $\mathtt{OTr}(\sigma)$, is defined as $\mathtt{OTr'}(\sigma, 0)$, where

$$\mathtt{OTr'}(\sigma, t) = \begin{cases} [(t, \beta)] & \text{if } \sigma = [\,] \\ \mathtt{OTr'}(l', t) & \text{if } \sigma = (c', c'', 0)_\alpha : l' \\ \mathtt{OTr'}(l', t + t') & \text{if } \sigma = (c', c'', t')_{\beta_A} : l' \\ (t, \beta) : (\overline{x}, \gamma) : \mathtt{OTr'}(l', 0) & \text{if } \sigma = (c', c'', 0)_\gamma : l' \end{cases}$$

where $c' = (s', \overline{r'}, l')$, $c'' = (s'', \overline{r''}, l'')$, and $\overline{x} = \overline{r''} - \overline{r'}$.

Let c be a configuration of M. The *set of observable traces* of M from c, denoted by $\mathtt{ObsTraces}(M, c)$, is defined as $\{\sigma' \mid \exists\ \sigma \in \mathtt{ValidTraces}(M, c) : \sigma' = \mathtt{OTr}(\sigma)\}$. □

Similarly to the notion of observable trace presented in Section 3, these observable traces do not match the usual concept of trace since they actually

include different information. In particular, observable traces do not include information concerning the modification of the state, as they only represent passing of time and transactions of resources.

The second possibility regarding the capability to observe assumes that the real system containing the IUT is *owned* by the owner of the IUT. It also assumes that the owner has access to every observable action that is performed in the system. So, the observer will be the *supervisor* of the system. In order to check the validity of the IUT the supervisor will have to discriminate the part of the system behavior that corresponds to the IUT. The observable trace of a specific IUT is essentially computed by *projecting* the system trace over the corresponding agent.

Definition 16. Let $S = (M_1, \ldots, M_m)$ be a system and σ be an observable trace of S. The *projection* of σ over M_i, denoted by $\texttt{Proj}(\sigma, M_i)$, is the observable trace of M_i defined as $\texttt{Proj'}(\sigma, M_i, 0)$, where

$$\texttt{Proj'}(\sigma,M_i,t) = \begin{cases} [(t,\beta)] & \text{if } \sigma = [\,] \\ \texttt{Proj'}(l',M_i,t+t') & \text{if } \sigma = (t',\beta):l' \\ \texttt{Proj'}(l',M_i,t) & \text{if } \sigma = (\overline{x},(j,k),\gamma):l' \wedge i \neq j,k \\ (t,\beta):(\overline{x},\gamma):\texttt{Proj'}(l',M_j,0) & \text{if } \sigma = (\overline{x},(j,k),\gamma):l' \wedge i = j \\ (t,\beta):(-\overline{x},\gamma):\texttt{Proj'}(l',M_k,0) & \text{if } \sigma = (\overline{x},(j,k),\gamma):l' \wedge i = k \end{cases}$$

Let c be a configuration of S. The set of *projected observable traces* of S from c over M_i, denoted by $\texttt{ProjObsTraces}(S, c, M_i)$, is defined as the set of traces $\{\sigma' | \exists\, \sigma \in \texttt{ValidTraces}(S, c, M_i) : \sigma' = \texttt{Proj}(\sigma, M_i)\}$. □

In the following definition, the first notion of fault corresponds to the approach introduced in Definition 15 while the second notion corresponds to the approach given in Definition 16.

Definition 17. Let *Spec* be a `USM` denoting the *specification* of an IUT, σ be an observable trace of the IUT, and c be the initial configuration of *Spec*. The trace σ is a χ*-fault* of the IUT with respect to *Spec* if $\sigma \notin \texttt{ObsTraces}(Spec, c)$.

Let $S = (Spec, M_1, \ldots, M_m)$ be a system, σ be an observable trace of the system consisting of the implementations of $M_1, \ldots, M_m$ and the IUT implementing *Spec*, and c be the initial configuration of S. The trace σ is a ψ*-fault* of the IUT with respect to *Spec* if $\texttt{Proj}(\sigma, IUT) \notin \texttt{ProjObsTraces}(S, c, Spec)$. □

Therefore, if a non-expected behavior is detected along an observation of the implementation then it can be concluded that the implementation does not conform to the specification. Similarly to the previous section, a more abstract conformance relation based on the comparison of the traces of the specification and the traces of the IUT can be defined.

Definition 18. Let I, S be `USMs` and c, c' be their initial configurations. I *conforms isolated to* S, denoted by $I\,\texttt{confi}\,S$, if $\texttt{ObsTraces}(I, c') \subseteq \texttt{ObsTraces}(S, c)$. □

The following result states that each kind of fault makes its corresponding conformance relation to fail. The proof is straightforward.

Lemma 5. Let I and S be two $\texttt{USMs}$. If a trace σ is a χ-fault of I with respect to S then $I \,\texttt{confi}\, S$ does not hold. If a trace σ' is a ψ-fault of I with respect to S then $I \,\texttt{confc}_*\, S$ does not hold.

The following result shows that the two main conformance relations actually match. Hence, assessing the conformance on the basis of the traces produced by an isolated agent is equivalent to check it on the basis of the traces that this agent produces when it is embedded in a system in company of *any* set of agents.

Theorem 1. $I \,\texttt{confi}\, S$ iff $I \,\texttt{confc}_*\, S$.

Proof. The implication from right to left is proved by contrapositive. Suppose that $I \,\texttt{confi}\, S$ does not hold. This means that I can perform an observable trace that S cannot. Let σ_1 be such observable trace and σ be a trace of I such that $\sigma_1 = \texttt{OTr}(\sigma)$. Clearly, S cannot perform σ because it would be able to perform the observable trace σ_1. The proof proceeds by constructing a test $\mathcal{T}$ for I and the possible trace σ. By Lemma 3, the system $\mathcal{S}' = (I, \mathcal{T})$ can perform a trace σ' where the sequence of configurations of I is given by σ. So, σ_1 is one of the observable traces $\mathcal{S}'$ can perform for I. Since S cannot perform any trace such that its observable trace is σ_1, the system $\mathcal{S} = (S, \mathcal{T})$ cannot perform a trace σ'' such that the sequence of configurations of S through that trace is given by a trace σ''' with $\sigma_1 = \texttt{OTr}(\sigma''')$. Thus, σ_1 is not an observable trace of $\mathcal{S}$ for S. Hence, $I \,\texttt{confc}_*\, S$ does not hold.

For the reverse implication, suppose that $I \,\texttt{confc}_*\, S$ does not hold. Let $M_1, \ldots, M_n$ be $\texttt{USMs}$ such that $I \,\texttt{confc}_{\{M_1,\ldots,M_n\}}\, S$ does not hold. Let $\mathcal{S} = (S, M_1, \ldots, M_n)$, $\mathcal{S}' = (I, M_1, \ldots, M_n)$, and c, c' be the initial configurations of $\mathcal{S}, \mathcal{S}'$, respectively. Let σ_1 be an observable trace that $\mathcal{S}'$ can perform for I but $\mathcal{S}$ cannot perform for S. Then, there exists a trace σ of I and a trace σ' of $\mathcal{S}'$ such that the sequence of configurations of I in σ' is given by σ and $\sigma_1 = \texttt{OTr}(\sigma)$. However, there do not exist two traces σ'' and σ''' such that this property holds for S and $\mathcal{S}$. In fact, if there existed a trace σ'' of S such that $\sigma_1 = \texttt{OTr}(\sigma'')$ then a trace σ''' of $\mathcal{S}$ making the previous property to hold can be easily obtained. The reason is that σ'' would interact with $M_1, \ldots, M_n$ in the same way as σ interacts with $M_1, \ldots, M_n$, since their observable behavior is the same (that is, σ_1). So, if the previous property does not hold for S and $\mathcal{S}$ then the reason must be that there does not exist a trace σ'' of S such that $\sigma_1 = \texttt{OTr}(\sigma'')$. Hence, S cannot perform the observable trace σ_1, so that $I \,\texttt{confi}\, S$ does not hold. □

This theorem together with Lemmas 4 and 5 imply the following result.

Corollary 1. Let I and S be two $\texttt{USMs}$. Let σ be a φ-fault, a χ-fault, or a ψ-fault of I with respect to S. Then, neither $I \,\texttt{confc}_*\, S$ nor $I \,\texttt{confi}\, S$.

5 Case Study: Kasbah

In this section the formalism is applied to specify and study Kasbah [21]. This is one of the pioneer proposals in the field of e-commerce multi-agent systems. It consists of a market of autonomous commerce agents that behave on behalf of their corresponding users. The agents interact with each other in order to reach good deals. In Kasbah, agents are either *sellers* or *buyers*. Each agent is intended to buy or sell one single specific item at the best possible price (lowest or highest respectively). Every time a user wishes to perform a selling/buying operation, he creates a new Kasbah agent. If he wants to buy an item (the case of selling is symmetric) then he configures the agent according to four requirements: The initial price it should offer (ip), the maximal price it should pay (mp), the deadline when it should refuse to buy the item (d), and a function denoting the way the agent should increase the buying price as time passes. This last function can be linear, square, or cubic. It will work so that the price mp will be offered exactly when d comes. The function will be defined by its exponent ex.

The first step to model a Kasbah buying agent with a `USM` consists in defining a utility function fitting into these requirements. If it denotes the kind of item the buying agent is interested in, mo denotes the money, and t denotes the time elapsed, then the utility function of a buying agent in Kasbah can be defined as $u(t, it, mo) = mo + (ip + \alpha \cdot t)^{ex} \cdot it$, where $\alpha = \frac{mp^{\frac{1}{ex}} - ip}{d}$. The value α can be easily obtained by taking into account that the maximal price mp should be paid just when the deadline d expires. Using a similar reasoning, the utility function for a seller agent is defined as $u(t, it, mo) = mo + (ip - \alpha \cdot t)^{ex} \cdot it$, where $\alpha = \frac{ip - mp^{\frac{1}{ex}}}{d}$. Both buyer and seller agents will be specified by using a `USM` with two states. The first one, s_1, will denote that the agent is active, that is, it can make transactions with other agents. The second one, s_2, represents that the agent is inactive. As long as the agent stays in s_1, it can make a deal with another agent provided that the transaction is *good* according to its utility function. If the time runs out and the deadline comes before any transaction is performed then the state changes to s_2. In s_2 no transaction is accepted by the agent. This will be expressed, in the case of the buyer, by using a utility function that gives a full importance to money, so transactions will be no longer possible. Besides, in the case that a transaction is performed before the deadline, the agent will immediately move to the state s_2. The corresponding transition will indicate that the item is deleted from the basket of resources of the agent, denoting that the agent sends the bought item to its user.

Definition 19. The `USM` associated to a buyer is $M_b = (S, s_{in}, V, U, at, mi, T)$, where $S = \{s_1, s_2\}$, $s_{in} = s_1$, $V = (t, it, mo)$, at and mi are set to any arbitrary values, U is defined as

$$U(s_1)(t, it, mo) = mo + (ip + \alpha \cdot t)^{ex} \cdot it \qquad \text{where } \alpha = \frac{mp^{\frac{1}{ex}} - ip}{d}$$
$$U(s_2)(t, it, mo) = mo$$

and the set of transitions T is defined as $T = \{(s_1, Q_1, Z_1, s_2), (s_1, Q_2, Z_2, s_2)\}$, where $Q_1(t, it, mo)$ holds iff $t \geq d$, $Z_1(it, mo) = (it, mo)$, $Q_2(t, it, mo)$ holds iff $it = 1$, and $Z_2(it, mo) = (it - 1, mo)$.

The USM associated to a seller is $M_s = (S', s'_{in}, V, U', at, mi, T')$, where $S' = \{s'_1, s'_2\}$, $s'_{in} = s'_1$, $V = (t, it, mo)$, at and mi are set to any arbitrary values, U' is defined as

$$U'(s'_1)(t, it, mo) = mo + (ip - \alpha \cdot t)^{ex} \cdot it \qquad \text{where } \alpha = \frac{ip - mp^{\frac{1}{ex}}}{d}$$
$$U'(s'_2)(t, it, mo) = mo$$

and the set of transitions T' is defined as $T' = \{(s'_1, Q'_1, Z'_1, s'_2), (s'_1, Q'_2, Z'_2, s'_2)\}$, where $Q'_1(t, it, mo)$ holds iff $t \geq d$, $Z'_1(it, mo) = (it - 1, mo)$, $Q'_2(t, it, mo)$ holds iff $it = 0$, and $Z'_2(it, mo) = (it, mo)$. □

Once both buyer and seller agents are defined, a minimal system S can be trivially defined by composing both agents so that $S = (M_b, M_s)$. More complex systems can be constructed by combining several buyers and sellers that trade with different items.

Next, some tests to check the validity of a given IUT that is supposed to implement a Kasbah agent are given. Consider the case of a buyer. The test $\mathcal{T}_1$ will check its behavior in its initial state. Then, by observing the behavior of a system consisting of the IUT and $\mathcal{T}_1$, it can be detected whether it corresponds to what would be obtained in the specification of such a system. The test $\mathcal{T}_1$ is very simple because it does not need to conduct the IUT to some target state, since it indeed begins the interaction with the IUT in the desired state.

Definition 20. Let $\mathcal{T}_1 = (\{s^1\}, s^1, V, U^1, at, mi, \emptyset)$ where $U^1(s^1)$ is the utility function to be used in the interaction of $\mathcal{T}_1$ with M_b in their states s^1 and s_1, respectively. The initial configuration of $\mathcal{T}_1$ is $c^1 = (s^1, (0, x^1, y^1), [\,])$, where $\overline{r}^1 = (x^1, y^1)$ is the basket of resources to be used in the interaction of $\mathcal{T}_1$ and M_b in their states s^1 and s_1. □

Note that $U^1(x^1)$ and $\overline{r}^1$ are the parameters $\widehat{U}$ and $\widehat{r}$ in the procedure of construction of tests described in Definition 10. It is obvious that $\mathcal{T}_1$ and M_b can reach the state s_1 of M_b. A more complex test $\mathcal{T}_2$ to check the behavior of the IUT in its final state can be created. So, the first aim of $\mathcal{T}_2$ will be to find a way to conduct the IUT to that state. The path considered in this case will be $\eta = [s_1, s_2]$. A possible trace to perform this path has also to be planned. Suppose that the initial configuration of M_b is $(s_1, (0, 0, y), [\,])$, that is, the current state is s_1, the time is 0, the agent owns 0 items and y dollars, and the list of pending accounts is empty. Suppose also that $y = (ip + \alpha \cdot \frac{d}{2})^{ex}$, that is, y is the amount of money the agent would accept to pay when half of the time to the deadline has elapsed. Then, consider the trace $\sigma = [(c, c_1, \frac{d}{2})_\beta, (c_1, c_2, 0)_\gamma, (c_2, c_3, 0)_\alpha]$ where $c_1 = (s_1, (\frac{d}{2}, 0, y), l')$, $c_2 = (s_1, (\frac{d}{2}, 1, 0), l'')$, and $c_3 = (s_2, (0, 0, 0), l''')$. The first evolution corresponds to let the time pass $\frac{d}{2}$ units of time. Then, in the second evolution, the agent exchanges all of its money by the item. At this moment, the agent fulfills the condition Q_1 to change its state to s_2. So, in the third evolution

the agent moves to s_2. The transformation function Z_1 makes the agent to send the bought item to its user. Thus, in s_2 the item is already missing. Clearly σ is a possible trace to perform η.

Definition 21. Let $\mathcal{T}_2 = (S^2, s_1^2, V, U^2, at, mi, T^2)$ where $S^2 = \{s_1^2, s_2^2\}$, the utility functions are $U^2(s_1^2)(t, it, mo) = \frac{mo}{y} + 0.95 \cdot it$ while $U^2(s_2^2)(t, it, mo)$ is set to an arbitrary value, the set of transitions is $T^2 = \{(s_1^2, Q_1^2, id, s_2^2)\}$, where $Q_1^2(t, it, mo)$ holds iff $it = 0$, and the initial configuration of $\mathcal{T}_2$ is $c' = (s_1^2, (0, x^2 + 1, y^2), [\,])$, where $\overline{r}^2 = (x^2, y^2)$ is set to an arbitrary value. □

$\mathcal{T}_2$ provides a mechanism to check whether the behavior of the IUT in its final state conforms to that of the specification M_b. The following lemma shows that $\mathcal{T}_2$ is capable to conduct (an IUT that conforms to) M_b to the state s_2 through the possible trace σ. The result is a direct consequence of Lemma 3.

Lemma 6. Let $\mathcal{S} = (M_b, \mathcal{T}_2)$ and c be its initial configuration. Then, there exists a trace $\sigma' \in \texttt{Traces}(\mathcal{S}, c)$ such that the sequence of configurations of M_b through σ' is equal to that of σ.

6 Conclusions and Future Work

This paper introduces a framework to formally specify and test e-commerce systems where agents represent the interest of the users. Emphasis is specially placed on the high level part of the behavior of agents. A formalism, called *utility state machines*, allows to appropriately specify the economic behavior of these agents. In addition, different testing methodologies to check whether an implementation of a specified agent behaves indeed as expected (according to the corresponding specification) have been defined. The main highlight of the *active* testing approach is that tests, in contrast with the usual situation, are also given by agents. Finally, the framework has been used to specify the main entities appearing in Kasbah as well as to derive tests for these specifications.

This paper firmly sets the basis for a formal model to deal with autonomous commerce agents. However, further work on this topic should follow to complement and extend the existing framework. In fact, two lines for present/future work are already contemplated. In the first one, with a major theoretical component, the notion of `USM` will be extended to also deal with low-level behavior. This extension will also affect the definition of the testing methodologies. In the second one, with a more practical component, the framework is going to be applied to other agent-based e-commerce systems (as we have done with Kasbah).

Acknowledgements: The authors would like to thank the anonymous referees of this paper for their careful reading and helpful comments.

References

1. Guttman R, Moukas A, Maes P. Agent-mediated electronic commerce: A survey. *The Knowledge Engineering Review*, 1998; **13**: 147–159.

2. Sandholm T. Agents in electronic commerce: Component technologies for automated negotiation and coalition formation. *Proceedings of CIA'98, LNCS 1435*. Springer, 1998; 113–134.
3. Ma M. Agents in e-commerce. *Communications of the ACM*, 1999; **42**: 79–80.
4. He M, Jennings N, Leung H. On agent-mediated electronic commerce. *IEEE Trans. on Knowledge and Data Engineering*, 2003; **15**: 985–1003.
5. Padget J, Bradford R. A pi-calculus model of a spanish fish market - preliminary report. *Proceedings of AMET'98, LNCS 1571*. Springer, 1998; 166–188.
6. Adi K, Debbabi M, Mejri M. A new logic for electronic commerce protocols. *Theoretical Computer Science*, 2003; **291**: 223–283.
7. Rodríguez I. Formal specification of autonomous commerce agents. *Proceedings of 19th ACM Symposium on Applied Computing, SAC'04*. ACM Press, 2004; 774–778.
8. Rodríguez I, Núñez M, Rubio F. Specification of autonomous agents in e-commerce systems. *Proceedings of Workshop on Theory Building and Formal Methods in Electronic/Mobile Commerce (TheFormEMC), LNCS 3236*. Springer, 2004; 30–43.
9. Núñez M, Rodríguez I, Rubio F. Testing of autonomous agents described as utility state machines. *Proceedings of Workshop on Integration of Testing Methodologies, LNCS 3236*. Springer, 2004; 322–336.
10. López N, Núñez M, Rodríguez I, Rubio F. A formal framework for e-barter based on microeconomic theory and process algebras. *Proceedings of Innovative Internet Computer Systems, LNCS 2346*. Springer, 2002; 217–228.
11. López N, Núñez M, Rodríguez I, Rubio F. A multi-agent system for e-barter including transaction and shipping costs. *Proceedings of 18th ACM Symposium on Applied Computing, SAC'03*. ACM Press, 2003; 587–594.
12. Rao A. AgentSpeak(L): BDI agents speak out in a logical computable language. *In Agents Breaking Away, LNAI 1038*. Springer, 1996; 42–55.
13. Hindriks K, de Boer F, van der Hoek W, Meyer JJ. Formal semantics for an abstract agent programming language. *Proceedings of Intelligent Agents IV, LNAI 1365*. Springer, 1998; 215–229.
14. Probert R, Chen Y, Cappa M, Sims P, Gahaziadeh B. Formal verification and validation for e-commerce: Theory and best practices. *Journal of Information and Software Technology*, 2003; **45**: 763–777.
15. Cavalli A, Maag S. Automated test scenarios generation for an e-barter system. *Proceedings of 19th ACM Symposium on Applied Computing, SAC'04*. ACM Press, 2004; 795–799.
16. Lee D, Yannakakis M. Principles and methods of testing finite state machines: A survey. *Proceedings of the IEEE*, 1996; **84**: 1090–1123.
17. Núñez M, Rodríguez I. Encoding `PAMR` into (timed) `EFSMs`. *Proceedings of FORTE 2002, LNCS 2529*. Springer, 2002: 1–16.
18. Alur R, Dill D. A theory of timed automata. *Theoretical Computer Science*, 1994. **126**: 183–235.
19. Brinksma E, Scollo G, Steenbergen C. LOTOS specifications, their implementations and their tests. *Proceedings of Protocol Specification, Testing and Verification VI*. North Holland, 1986; 349–360.
20. Tretmans J. Test generation with inputs, outputs and repetitive quiescence. *Software – Concepts and Tools*, 1996; **17**: 103–120.
21. Chavez A, Maes P. Kasbah: An agent marketplace for buying and selling goods. *Proceedings of PAAM'96*. 1996; 75–90.