Testing of Autonomous Agents described as Utility State Machines[⋆]

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. We introduce a methodology to test autonomous agents described by means of utility state machines. In contrast with the classical approach for testing state machines, we will use other utility state machines as test. In fact, the machine playing the role of the test will be in charge of guiding the IUT so that it performs the operations (i.e. exchanges of resources) indicated by the specification.

1 Introduction

E-commerce technologies introduce several advantages in usual human economic interaction. For instance, they allow vendors to sell products without temporal or storage restrictions and they provide customers with powerful mechanisms to compare prices. In particular, *autonomous commerce agents* are one of the most interesting and challenging technologies in e-commerce (see e.g. [3,4,14,8,5]). They are autonomous entities that perform, on behalf of their respective users, some of the activities required in economic processes. For instance, autonomous commerce agents may search for interesting products, advise their users about interesting offers, negotiate with other agents, or even perform transactions autonomously. Besides, e-commerce platforms and multi-agent e-commerce systems use to be heterogeneous distributed systems where different components perform complex interactions. Actually, guaranteeing a correct behavior is specially hard in this kind of systems. However, reliability is a must for the success of e-commerce systems: Since they directly affect the patrimony of the users, their social implantation dramatically depends on whether users can trust them.

Taking as objective the validation of e-commerce systems, several works have been proposed to study the correctness of *commerce communication protocols* [10,1]. Communication protocols are a key aspect in the validation of any e-commerce system, since any economic environment takes as first premise that any message sent by an economic party arrives correctly at the receiver. Only in this case commercial negotiation can be successfully performed. However, let us note that techniques for validating commerce communication protocols are

[⋆] Research supported by the MCyT project *MASTER* (TIC2003-07848-C02-01), the Junta de Castilla-La Mancha project *DISMEF* (PAC-03-001) and the Marie Curie RTN *TAROT* (MCRTN 505121).

not different to those used to check any kind of communication protocol, as they mainly focus on analyzing low-level details that are common to any distributed system. Nevertheless, e-commerce systems have enough intrinsic peculiarities to suggest the development of new specific validation techniques for this domain. Let us note that e-commerce systems are economic environments where entities compete for resources. Hence, a suitable conceptualization of the high-level details of such systems consists in representing and studying the *economic* behavior of the entities appearing in them.

In this paper we will provide a testing methodology to validate the high-level behavior of autonomous commerce agents in e-commerce systems. The aim of the testing mechanism will be to check whether the economic behavior of an implemented agent conforms to its corresponding specification. The high-level specification of an agent, concerning its economic behavior, can be defined as *"get what the user said he wants and when he wants it"*. We will consider that the specific preferences of the users will conform a *specification*. Therefore, we will need a proper formalism to represent this kind of specifications, that we will call *utility state machines* (see [12,13]).

In our testing methodology we will stimulate the implementation under test (that is, the IUT is the implementation of an agent) according to a given test case/suite. Since we are interested in high-level behavior, it would not be coherent to stimulate the IUT by sending a sequence of actions that conforms to a given communication protocol. Thus, 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. In other words, we will test whether the *tested agent* is able to gain some profit according to the user preferences. So, our primary goal is not to test whether an agent behaves according to a given communication protocol since, as we said before, there already exist other formalisms to perform this task. On the contrary, as we are interested in abstracting low-level issues, the only suitable way to perform the stimulus will be by giving the tests the form of another autonomous commerce agent, so that they interact with the IUT in the same way as a real system would do.

In terms of related work there are innumerable papers on e-commerce in general or on topics as e-commerce systems/architectures and agent-mediated e-commerce. This number strongly decreases when considering formal approaches. In this case we may mention [6,7] where, taking as basis the language PAMR [9], process algebras to specify e-barter systems were introduced. Regarding testing and validation techniques for e-commerce we may mention [11,2] where specific case studies are considered. In fact, as far as we know, our paper represents the first generic framework to formally specify and test e-commerce systems.

The rest of the paper is structured as follows. In Section 2 we briefly review the formalism that allows us to specify autonomous commerce agents. In Section 3 we present our testing methodology. The main highlight of this approach is that in order to test the behavior of an agent we use a suitable agent as test. Finally, in Section 4 we present our conclusions and some lines for future work.

2 Utility State Machines

In this section we briefly review the notion of *utility state machine*. A more detailed presentation of this formalism can be found in the companion paper [13]. We use these machines to describe agents representing users. 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 1. We consider $\mathbb{R}_+ = \{x \in \mathbb{R} | x \geq 0\}$. We will usually denote *vectors* in $\mathbb{R}^n$ (for $n \geq 2$) by $\bar{x}, \bar{y}, \ldots$ We consider that $\bar{0}$ denotes the tuple having all the components equal to zero. Given $\bar{x} \in \mathbb{R}^n$, x_i denotes its *i-th* component. We extend some usual arithmetic operations to vectors. Let $\bar{x}, \bar{y} \in \mathbb{R}^m$. We define $\bar{x} + \bar{y} = (x_1 + y_1, \ldots, x_n + y_n)$ and $\bar{x} \cdot \bar{y} = (x_1 \cdot y_1, \ldots, x_n \cdot y_n)$. We write $\bar{x} \leq \bar{y}$ if for any $1 \leq i \leq n$ we have $x_i \leq y_i$.

Let us suppose that there exist n different kinds of resources. A *utility function* is any function $f : \mathbb{R}_+^n \longrightarrow \mathbb{R}_+$. □

In order to specify agents we use *utility state machines* [12,13]. This formalism is indeed an adaption and extension to our purposes of the classical notion of *extended finite state machines* (`EFSM`). In particular, we have to take into account *time* as a factor that influences the preferences of users. 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 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. We say that the tuple $M = (S, s_{in}, V, U, at, mi, T)$ is a *utility state machine*, in short `USM`, where we consider that

- S is a *set of states*.
- $s_{in} \in S$ 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.
- $U : S \longrightarrow (\mathbb{R}_+^{n+1} \longrightarrow \mathbb{R}_+)$ 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.
- mi 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, $Q : \mathbb{R}_+^{n+1} \longrightarrow$ `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. We require that for any state s there do not exist two different transitions $t_1, t_2 \in T$, with $t_1 = (s, Q_1, Z_1, s'_1)$ and $t_2 = (s, Q_2, Z_2, s'_2)$, and a tuple $\overline{r}$ such that both $Q_1(\overline{r})$ and $Q_2(\overline{r})$ hold. □

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 *passing of time*. The former will be used to denote the transactions agents will be allowed to perform. Given a `USM` M, both transaction and time consumption transitions may be *implicitly* inferred from the definition of M. We will give the formal representation in the forthcoming Definition 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, the current values of the variables, and the *pending accounting* of the machine.

Definition 3. Let $M = (S, s_{in}, V, U, at, mi, T)$ be a `USM`. 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
- $l = [(p_1, e_1), \ldots, (p_m, e_m)]$ is a list of pairs (profit,time) representing 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, if 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 some auxiliary definitions. Intuitive explanations of these notions can be found in [13].

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 $l = [(p_1, e_1), \ldots, (p_m, e_m)]$. The *maximal waiting time* for M in the configuration c, denoted by $\texttt{MaxWait}(M, c)$, is defined as $\min\{e_1, \min\{u' \mid \exists (s, Q, Z, s'') \in T : u' \geq u \wedge Q(u', r_1, \ldots, r_n)\}\}$

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 true the auxiliary condition $\texttt{ClearProfit}(M, c)$. On the contrary, if e_1 is the minimal value and $p_1 < 0$ then we say that 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:

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

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

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

Let us note that profits in any (non-empty) list of pending accounts are either all positive or all negative. We have used a functional programming notation to define lists: $[\ ]$ denotes an empty list, $l \mathbin{++} l'$ 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. We will consider four types of evolutions: *changing the state*, *passing of time*, *passing of time with failure*, and *performing a transaction*.

Definition 5. 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 $\overline{r} = (u, r_1, \dots, r_n)$ and $l = [(p_1, e_1), \dots, (p_m, e_m)]$. 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 $K = \alpha$, $tc = 0$, $s' = s''$, and $\overline{r'} = (0, r'_1, \dots, r'_n)$, where $(r'_1, \dots, r'_n) = Z(r_1, \dots, r_n)$ and $l' = [(p_1, e_1 - u), \dots, (p_m, e_m - u)]$.

(2) (*Passing of time*) If the condition of (1) does not hold then for any $tr \in \mathbb{R}_+$ such that $0 < tr \leq \texttt{MaxWait}(M, c) - u$, we have $tc = tr$, $s' = s$, $\overline{r'} = (u + tr, r_1, \dots, r_n)$, and l' is defined as follows:

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

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 $\overline{r''} = (u, r''_1, \dots, r''_n) \geq \overline{0}$ such that $U(s)(\overline{r''}) - U(s)(\overline{r}) + \texttt{Accumulated}(l) > -mi$, we have $K = \gamma$, $tc = 0$, $s' = s$, $\overline{r'} = \overline{r''}$, and $l' = \texttt{Update}(l, U(s)(\overline{r'}) - U(s)(\overline{r}))$.

We denote by $\texttt{Evolutions}(M, c)$ 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'. We denote by $\texttt{Traces}(M, c)$ the set of traces of M from c. $\square$

First, if one of the guards associated with a transition between states holds then the state changes, 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 an amount of time less than or equal to the maximal waiting time. Besides, if the elapsed time corresponds to the one when 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. Finally, if a machine performs a transaction then we require that it does not give resources that it does not own and that the accumulated losses stay below the maximal threshold. Let us note that the second and third types of transition can be performed only if the corresponding `USM` cannot change its state.

In the next definition we identify the traces that are free of failures.

Definition 6. Let M be a USM and c be a configuration of M. Let us suppose $c_1 = c$ and let $\sigma = [(c_1, c_2, t_1)_{K_1}, \ldots, (c_{n-1}, c_n, t_{n-1})_{K_{n-1}}] \in \texttt{Traces}(M, c)$ be a trace of M from c. We say that σ is a *valid trace* of M from c if there does not exist $1 \leq i \leq n-1$ such that $K_i = \beta'$. We define the set of valid traces of M from c by $\texttt{ValidTraces}(M, c)$. □

Next we extend the previous framework so that *systems*, made of USMs interacting among them, can be defined. 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.

Definition 7. Let $M_1, \ldots, M_m$ be USMs such that for any $1 \leq i \leq m$ we have $M_i = (S_i, s_{in\ i}, V, U_i, at_i, mi_i, T_i)$. We say that 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. 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 performed by other USMs. This will be the case of passing of time and transactions. In fact, the only actions 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. In the meanwhile, transactions and passing of time transitions will be forbidden.

In the following definition we identify the set of evolutions a system can perform from a given configuration. In order to make explicit the *failure* of any of the USMs in the system, we distinguish again passing of time transitions producing a failure. In this case, we will explicitly indicate the set of USMs producing a failure. Hence, the label β_A denotes a passing of time transition in which the USMs included in A produced a failure. So, a failures-free passing of time transition is denoted by $\beta_\emptyset$.

Definition 8. Let $S = (M_1, \ldots, M_m)$ be a system and $c = (c_1, \ldots, c_m)$ be a configuration such that we have $M_i = (S_i, s_{in\ i}, V, U_i, at_i, mi_i, T_i)$ and $c_i = (s_i, \overline{r_i}, l_i)$ for any $1 \leq i \leq m$. 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 $K = \alpha$, $tc = 0$, and for any $1 \leq j \leq m$ we have $c'_j = c_j$ if $i \neq j$, while $c'_i = c''_i$.

(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 $tc = tr$, $c'_i = c''_i$ for any $1 \leq i \leq m$, and $K = \beta_A$ where the set A is defined as $A = \{i \mid (c_i, c''_i, tr)_{\beta'} \in \texttt{Evolutions}(M_i, c_i)\}$.

(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 $l \in \{j,k\}$ we have $(c_l, c''_l, 0)_\gamma \in$ $\texttt{Evolutions}(M_l, c_l)$ and $c''_l = (s''_i, \overline{r''_i}, l''_i)$, with $\overline{r_j} + \overline{r_k} = \overline{r''_j} + \overline{r''_k}$, then $K = \gamma$, $tc = 0$, $c'_j = c''_j$, $c'_k = c''_k$, and for any $1 \leq i \leq m$, with $i \neq j,k$, we have $c'_i = c_i$.

We denote by $\texttt{Evolutions}(S,c)$ 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'. We denote by $\texttt{Traces}(S,c)$ the set of traces of S from c . □

If a USM can change its state then the corresponding transition is performed while any other USM remains unmodified (first clause). If no USM can modify its state then passing of time and transaction transitions 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 we require that trading USMs can (individually) perform the exchange and that goods are not created/destroyed as a result of the exchange. Let us 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 reasoning that we did for single agents, we can identify the valid traces of systems. In this case, we are interested in identifying the traces such that a specific USM belonging to the system produces no failure.

Definition 9. Let $S = (M_1, \ldots, M_m)$ be a system and c be a configuration of S. Let us suppose $c_1 = c$ and let $\sigma = [(c_1, c_2, t_1)_{K_1}, \ldots, (c_{n-1}, c_n, t_{n-1})_{K_{n-1}}] \in$ $\texttt{Traces}(S,c)$ be a trace of S from c. σ 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 $i \in A$. We denote the set of valid traces of S from c for M_i by $\texttt{ValidTraces}(S, c, M_i)$. □

3 Testing Autonomous Commerce Agents

In this section we describe 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. We will create *artificial* environments to stimulate the IUT according to a foreseen plan. These environments will be the *tests* in our testing process. As we said in the introduction, tests will be defined in the same way we define specifications. Thus, the definition of tests will focus only on the high-level economic behavior. Besides, the actions available for tests will be the same as those for specifications. Let us remark that if tests included low-level details that made *explicit* their way to interact with the implementation then the test would check details that the specification actually does not define.

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, we will check whether the joint evolution of the test and the IUT conforms to the (failure free) behavior we would have in the specification of a 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 other words, the test has to *induce* the IUT to fulfill the guards governing the transitions between states. The test will be able to do so by *exchanging* resources with the IUT so that the guards are fulfilled. 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, endowing 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 leaves a free room for the IUT to *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 test, the first decision consists in fixing the set of states it will be intended to guide the IUT through.

Definition 10. 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 $1 \leq i \leq m-1$ we have that there exists $(s_i, Q, Z, s_{i+1}) \in T$. $\square$

The availability of a part of a path to be performed will be strongly influenced by the exchanges performed in previous states of the path. The key to understand and predict the evolution of resources at a given state is that a `USM` is forced to perform exchanges so that, in the long term, utility never worsens. Therefore, if a guard between states is only fulfilled in configurations where the utility is less than the current utility then the `USM` will never be able to perform the corresponding transition. The following definition checks whether it will be *possible* for a `USM` to evolve across a given path. Prior to formally define this concept we introduce some auxiliary notation. We give a simple notion to identify the list of states ranged in a trace as well as both the baskets of resources which made possible each transition and the baskets obtained after each transition. These baskets give us a possible way to deal with resources so that the test can guide the IUT through the path. In the following definition we also introduce notation to access some elements within a list of tuples. Given a list l we have that $l.i$ denotes a list formed by the i^{th} element of each tuple appearing in l. Besides, l_i denotes the i^{th} element of the list. For example, if we consider $l = [(1,2,3),(3,4,1),(5,2,7),(8,1,6),(6,5,4)]$ then $l.2 = [2,4,2,1,5]$ and $l_3 = (5,2,7)$. Moreover, we can combine both functions. For instance, $(l.2)_4 = 1$.

Definition 11. 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 we denote by l_i the i-th element of l. For any $1 \leq i \leq n$ we define the list $l.i$ as

$$l.i = \begin{cases} [\,] & \text{if } l = [\,] \\ x_i : (l'.i) & \text{if } l = (x_1,\ldots,x_i,\ldots,x_n) : l' \end{cases}$$

We say that 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 we say that $\texttt{StateTran}(\sigma).2$ is a *possible use of resources to perform* η and that σ is a *possible trace to perform* η. □

Once we have identified a possible use of resources to perform a given path, we can define a test that gives to the IUT the resources it needs to perform the next transition of the path at every state. 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* introduced in the previous definition. 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, the utility function of the test 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 the 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 *steal* the appropriate resources from the IUT, so that it fulfills the guard, the test will reward these items with its utility function. In the following definition we describe 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. Let us note that each kind of resources will have to be either given or removed, but not both.

Definition 12. 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

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

where $\Phi(a, b)$ is equal to $\frac{a}{b}$ if $b \neq 0$ and equal to 0 if $b = 0$. If $\overline{v} = \overline{0}$ we simply consider that $\Psi_{\overline{v},\overline{w}}(\overline{x}) = 0$. □

Lemma 1. Let $\overline{v}, \overline{w} \in \mathbb{R}_+^n$ such that $\overline{v} \neq \bar{0}$ and $\overline{v} \cdot \overline{w} = \bar{0}$. We have that $\Psi_{\overline{v},\overline{w}}(\overline{v}) > \Psi_{\overline{v},\overline{w}}(\overline{w})$.

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

Let us remark that by using the function $\Psi_{\overline{v},\overline{w}}$ as a component of the test we will favor that the test and the IUT exchange resources so that the appropriate transition guards of the IUT hold. This is so because both agents will get happier by exchanging the exact resources we want them to exchange. In particular, if $\Psi_{\overline{v},\overline{w}}$ is *adequately* inserted in 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 exchange we want to perform would improve the utility of the test, since its utility function will return 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. Since a possible use of resources takes into account only those exchanges where the `USM` does not lose utility, the utility function of the specification will accept such a exchange. Similarly, any IUT conforming to the specification will do so.

We are now ready to define the test that will lead the IUT through a given path. As we said before, each state in the test will represent the corresponding state of the specification in the path. Besides, in order to properly track the IUT, we need the test to 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. As we know the total amount of resources in the system, the test can trivially compute the resources the IUT has by subtracting its own resources from the total amount and 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. The next definition provides a mechanism for creating a test to guide the IUT through a given path. In this definition, $\widehat{U}$ and $\widehat{r}$ denote, respectively, the utility function and the resources the test will apply in its last state.

Definition 13. Let $M = (S, s_{in}, V, U, at, mi, T)$ be a USM and let $c = (s_{in}, \overline{r_{in}}, [\,])$ be a configuration of M. Let $\eta = [s_1, \ldots, s_m]$ be a possible path of M from c and let $\sigma \in \texttt{ValidTraces}(M, c)$ 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 USM $T = (S', s'_{in}, V, U', at, mi, T')$ where

– $S' = \{s'_1, \ldots, s'_m\}$, where $s'_1, \ldots, s'_m$ are fresh states.
– $s'_{in} = s'_1$.
– $U'(s'_m) = \widehat{U}$ and for any $1 \leq i \leq m-1$ we have

$$U'(s'_i)(t, \overline{x}) = \begin{cases} \Psi_{\texttt{Steal}_i, \texttt{Give}_i}(\bar{x} - \texttt{Reserve}_i) & \text{if } \overline{x} \geq \texttt{Reserve}_i \\ 0 & \text{otherwise} \end{cases}$$

where

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

where Ω returns the original tuple but substituting negative numbers by 0 and we assume that $(\texttt{StateTran}(\sigma).3)_0 = \overline{r_{in}}$.
– $T' = \{(s'_1, Q'_1, id, s'_2), \ldots, (s'_{m-1}, Q'_{m-1}, id, s'_m)\}$ is such that for any $1 \leq i \leq m$ we have $Q'_i(t, \overline{r}) = Q_i(t, (\overline{r_{in}} + \texttt{Reserve}_0 - \overline{r}))$, where id is the identity function, Q_i is such that $(s_i, Q_i, Z_i, s_{i+1}) \in T$ and $Q_i(t', (\texttt{StateTran}(\sigma).2)_i) = \texttt{True}$ for some t' and Z_i, with $Z_i((\texttt{StateTran}(\sigma).2)_i) = (\texttt{StateTran}(\sigma).3)_i$, and $\texttt{Modif}_i = \sum_{1\leq j<i} (\texttt{StateTran}(\sigma).3)_j - (\texttt{StateTran}(\sigma).2)_j$

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

Let us comment the previous definition. Every state in the path η has a corresponding state in the test. Let us remark that if η contains cycles then each occurrence of the same state in η will be represented by a different state in the test. 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, $\texttt{Steal}_i$). 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 $\texttt{Give}_i$, $\texttt{Steal}_i$, and $\texttt{Reserve}_i$ are calculated by analyzing σ, that is the possible trace we choose 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), we calculate the resources the IUT must get/lose in each state to advance along the path η. 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). Let us remark that the expression $\overline{r_{in}} + \mathtt{Reserve}_0 + \mathtt{Modif}_i - \overline{r}$ allows the test to determine the resources the specification owns, assuming that the test owns the basket $\overline{r}$. The reason is that $\overline{r_{in}} + \mathtt{Reserve}_0$ is equal to the total amount of resources existing initially in the system, while $\mathtt{Modif}_i$ 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 $\mathtt{Modif}_i$ is computed by comparing the tuples of resources before and after each modification of the state (denoted by $(\mathtt{StateTran}(\sigma).2)_j$ and $(\mathtt{StateTran}(\sigma).3)_j$, respectively). Let us 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.

Once a test is defined, we can apply it to the IUT to detect faults. We will be able to detect faults 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 consisting of both agents. In order to perform these observations we have to fix the kind of actions we will be able to observe. First of all, we will suppose that the initial amount of resources of the tested agent is known. Second, we will suppose that each agent in the system provides us a mechanism to observe its basket of resources at any time. This requirement will be basic to trace the economic behavior of the system. Let us note that if we can access the baskets of resources of the agents then we can detect the performed transactions just by checking whether the baskets change.[1] Besides, another event we can detect is the passing of time. Nevertheless, as we assume that the tested agents are black boxes, we will not be able to check whether an agent changes its state. Summarizing, the visible events will be the transactions of our agents and the elapsed time. The following definition formalizes the kind of traces we will be able to observe from a system of agents.

Definition 14. An *observable trace of a system* is a list $[e_1, \ldots, e_m]$ where for any $1 \leq i \leq m$ we have that either $e_i = (t, \beta)$, with $t \in \mathbb{R}_+$, or $e_i = (\overline{x}, (a_1, a_2), \gamma)$, with $\overline{x} \in \mathbb{R}^n_+$. □

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 we assume $a_1 < a_2$. Let us note that we have not included a special label to denote a *failed* passing of time because the observation of a system does not explicitly provide such information. In order to detect a fault in one of the agents included in a system (in this case, the IUT) we will need to check whether the observed behavior matches a possible behavior of

[1] Alternatively, we could suppose that the performed transactions are the only information we can observe. In this case it is also trivial to infer the baskets of resources of the agents at any time.

the specification of the 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.[2] However, we must take into account that the traces of the specification include information about the *internal* behavior of the system. Thus, we must first identify the set of *visible* traces of the specification. Intuitively, a trace becomes visible after removing all the events related to the modification of the states and by joining all the consecutive passing of time transitions together into a single one. In the following definition, we identify the set of observable traces of a system. We will require that the behavior of *one* of the `USMs` included in the system is free of failures. The reason is that our aim is to identify the observable traces of a system consisting of a specification and a test where the behavior of the specification is correct. By comparing them with the traces of the system consisting of the IUT and the test we will be able to detect failures in the IUT.

Definition 15. Let $S = (M_1, \dots, M_m)$ be a system, let $1 \leq i \leq m$, and let $\sigma \in \texttt{ValidTraces}(S, c, M_i)$. The *observable trace* of the trace valid for M_i σ, denoted by $\texttt{OTr}(\sigma, M_i)$, is defined as $\texttt{OTr'}(\sigma, M_i, 0)$, where

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

where

$$\begin{aligned} d' &= ((s'_1, \overline{r'_1}, l'_1), \dots, (s'_{a_1}, \overline{r'_{a_1}}, l'_{a_1}), \dots, (s'_{a_2}, \overline{r'_{a_2}}, l'_{a_2}), \dots, (s'_m, \overline{r'_m}, l'_m)) \\ d'' &= ((s'_1, \overline{r'_1}, l'_1), \dots, (s'_{a_1}, \overline{r''_{a_1}}, l''_{a_1}), \dots, (s'_{a_2}, \overline{r''_{a_2}}, l''_{a_2}), \dots, (s'_m, \overline{r'_m}, l'_m)) \end{aligned}$$

and $\overline{x} = \overline{r''_{a_1}} - \overline{r'_{a_1}}$.

Let $S = (M_1, \dots, M_m)$ be a system and c a configuration of S. We define the *set of observable traces* of S from c for a valid M_i, denoted by $\texttt{ObsTraces}(S, c, M_i)$, as $\{\sigma' \mid \exists\, \sigma \in \texttt{ValidTraces}(S, c, M_i) : \sigma' = \texttt{OTr}(\sigma, M_i)\}$. □

Now that we can compute the observable traces of a system specification, we can detect whether the observable behavior of a IUT represents a faulty behavior. 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 16. Let $S = (Spec, Test)$ be a system where $Test$ is a test to guide the specification $Spec$ through a given path η. Let σ'' be an observable trace of a real system RS consisting of an IUT and the test $Test$. Let c denote the initial configuration of S. We say that σ'' represents a *fault* of the IUT with respect to $Spec$ if $\sigma'' \notin \texttt{ObsTraces}(S, c, Spec)$. □

[2] Let us remark that we only require that the behavior of the IUT is free of faults, as a fault of the test does not mean the incorrectness of the IUT.

4 Conclusions and Future Work

In this paper we have presented a framework to test agents whose underlying model is given by a utility state machine. The main highlight of our *active* testing approach is that tests, in contrast with the usual situation, are also given by agents. This paper together with [13] firmly sets the basis for a formal model to deal with autonomous commerce agents. However, we consider that further work on this topic should follow to complement and extend the existing framework. First, other notions of testing should be explored. In particular, we are working with a more *passive* approach where an agent is simply observed. Second, we think that in order to asses the usefulness of our framework it is necessary to deal with other agent-based e-commerce systems (as we have done with Kasbah).

References

1. K. Adi, M. Debbabi, and M. Mejri. A new logic for electronic commerce protocols. *Theoretical Computer Science*, 291(3):223–283, 2003.
2. A. Cavalli and S. Maag. Automated test scenarios generation for an e-barter system. In *19th ACM Symposium on Applied Computing, SAC'04*, pages 795–799. ACM Press, 2004.
3. A. Chavez and P. Maes. Kasbah: An agent marketplace for buying and selling goods. In *PAAM'96*, pages 75–90, 1996.
4. R. Guttman, A. Moukas, and P. Maes. Agent-mediated electronic commerce: A survey. *The Knowledge Engineering Review*, 13(2):147–159, 1998.
5. M. He, N.R. Jennings, and H. Leung. On agent-mediated electronic commerce. *IEEE Trans. on Knowledge and Data Engineering*, 15(4):985–1003, 2003.
6. N. López, M. Núñez, I. Rodríguez, and F. Rubio. A formal framework for e-barter based on microeconomic theory and process algebras. In *Innovative Internet Computer Systems, LNCS 2346*, pages 217–228. Springer, 2002.
7. N. López, M. Núñez, I. Rodríguez, and F. Rubio. A multi-agent system for e-barter including transaction and shipping costs. In *18th ACM Symposium on Applied Computing, SAC'03*, pages 587–594. ACM Press, 2003.
8. M. Ma. Agents in e-commerce. *Communications of the ACM*, 42(3):79–80, 1999.
9. M. Núñez and I. Rodríguez. PAMR: A process algebra for the management of resources in concurrent systems. In *FORTE 2001*, pages 169–185. Kluwer Academic Publishers, 2001.
10. J.A. Padget and R.J. Bradford. A pi-calculus model of a spanish fish market - preliminary report. In *AMET'98, LNCS 1571*, pages 166–188. Springer, 1998.
11. R.L. Probert, Y. Chen, M. Cappa, P. Sims, and B. Gahaziadeh. Formal verification and validation for e-commerce: Theory and best practices. *Journal of Information and Software Technology*, 45(11):763–777, 2003.
12. I. Rodríguez. Formal specification of autonomous commerce agents. In *19th ACM Symposium on Applied Computing, SAC'04*, pages 774–778. ACM Press, 2004.
13. I. Rodríguez, M. Núñez, and F. Rubio. Specification of autonomous agents in e-commerce systems. In *Workshop on Theory Building and Formal Methods in Electronic/Mobile Commerce (TheFormEMC), LNCS 3236*. Springer, 2004.
14. T. Sandholm. Agents in electronic commerce: Component technologies for automated negotiation and coalition formation. In *CIA'98, LNCS 1435*, pages 113–134. Springer, 1998.