Specification of Autonomous Agents in e-commerce Systems*

Ismael Rodríguez, Manuel Núñez, and Fernando Rubio

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

Abstract. We present a generic formal framework to specify e-commerce systems. Specifically, we introduce a formalism to represent the behavior of the agents involved in the system. We will concentrate only on the *high level* behavior, that is, the one related to the economic performance of the agents. Finally, we apply our framework to the specification of the e-commerce system Kasbah.

1 Introduction

Among those areas where computers are yielding a higher impact during the last years we may remark electronic commerce. In fact, commerce activities under electronic, computer, and telecommunication support have several advantages with respect to traditional commerce. Nevertheless, e-commerce applications are usually very complex. Thus, it is hard to ensure reliability and correctness. Actually, e-commerce applications use to be distributed environments consisting of a big amount of heterogeneous components, being each of these components eventually depending on a different party. Besides, e-commerce applications usually have repercussions on the private patrimony of the users, either by conducting the user decisions through recommendations or by performing transactions in his name. Therefore, the success of an e-commerce platform dramatically depends on the confidence the users have on it. So, a key issue in the current and future development of e-commerce systems is the introduction of techniques to validate their behavior so that all users can trust the platform.

An aspect that has attracted a lot of attention is the *communication validation* of e-commerce protocols. Transactions in e-commerce environments involve a lot of communications, being critical in the sense that any transmitted message must conform to the meaning the emitter initially gave to it. Only in this case, the actions of a user will depend on his own responsibility, which is a requirement in any economic environment. Thus, typical problems to be issued are authentication, privacy, integrity of data, and (no) repudiation (see e.g. [1,2]). In this line, some formal approaches have been proposed to check the behavior of some

* This research has been supported by the Spanish 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).

e-commerce environments. These solutions focus on formalizing and studying the communication protocols used in the corresponding e-commerce application. In fact, they are not specific of e-commerce domains, as they are used in the same way when any other communication protocol is validated, that is, they deal with a *part* of the *low-level* behavior of an e-commerce application. Thus, they cannot be considered as *specific* e-commerce validation formal techniques. We think that the e-commerce field contains enough specific aspects to deserve its own validation techniques and tools to efficiently deal with its intrinsic features.

The aim of this paper is to develop a specific framework to specify and check the correctness of the *high-level* part of e-commerce applications. To become more definite, we introduce a formalism to specify the behavior of *autonomous commerce agents*. In fact, autonomous commerce agents are one of the most interesting and challenging technologies in e-commerce (see e.g. [6,14,11,8]). They are autonomous entities that perform transactions in the name of their respective users. Their high-level specification 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*.[1] Therefore, we will need a proper formalism to represent this kind of specifications, that we will call *utility state machines* and were introduced in a slightly different form in [13].

In terms of related work there are innumerable papers on e-commerce in general or on topics as e-commerce systems/arquitectures and agent-mediated e-commerce. This number strongly decreases when considering formal approaches to e-commerce. In this case we may mention [9,10] where, taking as basis the language PAMR [12], process algebras to specify e-barter systems (that is, a restricted notion of e-commerce) were introduced.

The rest of the paper is structured as follows. In Section 2 we present the formalism that allows us to specify autonomous commerce agents. In Section 3, we define how agents can be combined to build systems of agents. In Section 4 we apply our formal framework to the Kasbah system. Finally, in Section 5 we present our conclusions and some lines for future work.

2 Utility State Machines

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

[1] In this paper we do not deal with how agents *infer* the exact desires of the users since this goes beyond the scope of the paper (see e.g. [4,5,7]).

by low-level details, but this influence will be considered only on the basis of its consequences, that is, on the basis of the high-level behavior.

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

Definition 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 $f : \mathbb{R}_+^n \longrightarrow \mathbb{R}_+$. □

Intuitively, if f is a utility function then $f(\overline{x}) > f(\overline{y})$ 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 owing 2 apples or 3 oranges. Let us consider another user whose utility function is $f(x_1, x_2) = x_1 + 2 \cdot x_2$. Then, both users can make a deal if the first user gives 3 oranges in exchange of 4 apples: After the exchange both users are happier. Let us suppose now that x_2 represents the amount of money (in euros). In this situation, the first user would be the customer while the second one might represent the vendor. Let us remark that utility functions allow a great expressivity in preferences. For instance, $f(x_1, x_2) = x_1 \cdot x_2$ represents a utility function denoting that variety is preferred. A usual assumption is that no resource is a *bad*, that is, $\frac{\Delta f(x_1,\ldots,x_n)}{\Delta x_i} \geq 0$ 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 just by considering another resource representing the *absence* of it.

In order to formally specify autonomous commerce agents we have to represent the different objectives of the corresponding user along time. Thus, our formalism will provide the capability of expressing different utility functions depending on the situation, represented by the current *state*. Besides, *objectives*, represented by utility functions, will change depending on the availability of resources. These events will govern the transitions between states. Our formalism, that we call *utility state machines*, is indeed an adaption and extension to our purposes of the classical notion of *extended finite state machines* (`EFSM`). We will be able to deal with *time* as a factor that influences the preferences of users, either by affecting the value returned by a given utility function (e.g. the interest in a given item could decay as time passes) or by defining when the utility function must change (e.g. an old technology is considered obsolete and it is no

longer interesting). Besides, time will affect agents in the sense that any negative transaction will imply a deadline for the agent to retrieve the benefits of it. In fact, gaining profit in the long run may sometimes require to perform actions that, considered in the short term, are detrimental. Thus, time will be used to check whether the transaction was beneficial in the long term. In addition, time will appear as part of the freedom that specifications give to implementations: It does not matter whether an agent immediately performs a transaction as long as its decision is useful to improve the utility in the long term.

Definition 2. 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, $Q : \mathbb{R}_+^{n+1} \longrightarrow \texttt{Bool}$ is a predicate on the set of variables, and $Z : \mathbb{R}_+^n \longrightarrow \mathbb{R}_+^n$ is a transformation over the current variables. 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.

□

We can consider environments where the resources increase/decrease as a result of the agent activities. These modifications represent the decision of the user to introduce new resources in the market or to take away some of them. Both activities are accounted for by the Z functions appearing in transitions. Besides, let us remark that available resources and time influence the conditions required to change the state of the machine. This is taken into account by the Q functions appearing in transitions. Let us also note that in these constraints we may also consider the time previously consumed. In fact, we could use an additional *abstract* resource to accumulate the time consumed in previous states.

It is worth to point out that the set of transitions T does not include the complete set of transitions the specification will allow real agents to perform. Some additional transitions must be considered: *transactions* and *passing of time*. The former will be used to denote the transactions agents will be allowed to perform. These transactions will include some simple constraints that allow us

to avoid that an agent gives more resources than those actually owned, or to avoid giving resources if the amount of value lost in previous negative transactions, and not retrieved yet, is too high. Passing of time denotes the free decision of agents to idle. We consider two constraints on passing of time. First, an agent should not let the time pass so that previous negative transactions are not retrieved before their deadlines. Second, time conditions may trigger the modification of the state of an agent. Thus, in the exact time it happens, and before the passing of time *action* continues, the transition between states will be performed. Given a `USM` M, both transaction and time consumption transitions may be *implicitly* inferred from the definition of M. 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, in the case that p is negative, or the time in which p will be considered a *clear* profit if no negative profit is registered before. In order to explain the main characteristics of our framework, we will consider a simple (but illustrative) running example where some kids engage in the complicated world of becoming *entrepreneurs*.

Example 1. Let us suppose that little Jimmy wants to sell lemonade in the market of his town. We will construct a `USM` $J = (S, s_{in}, V, U, at, mi, T)$ to represent the economic behavior of Jimmy. The tuple of variables $V = (t, l, d, s, m)$ contains a value denoting time (t) and the amount of each resource: lemons (l), lemonades (d), selling licenses (s), and money (m).

This `USM` 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 $U(s_1)(\overline{r}) = 10 \cdot s + m$, 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

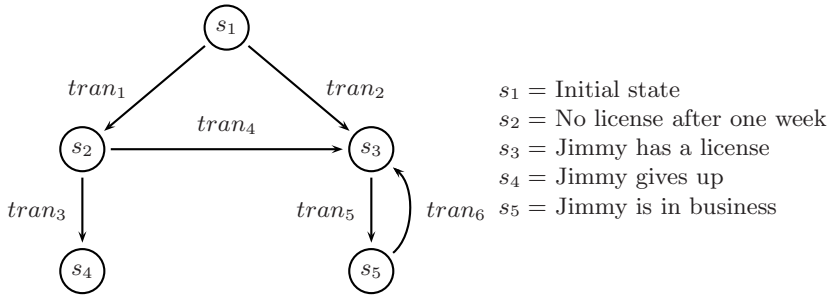

Fig. 1. Example of `USM`: Jimmy's business.

to the new state. In state s_2 we have $U(s_2)(\overline{r}) = 20 \cdot s + m$, 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 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 $U(s_3)(\overline{r}) = 0.2 \cdot l + m$. 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, $U(s_4)(\overline{r})$ 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.

We suppose that Jimmy uses 3 lemons for each lemonade. Thus, when Jimmy is in the state 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 $Q_4(t, \overline{r}) = (l = 12)$ and $Z_4(\overline{r}) = (l - 12, d + 4, s, m)$. In the state s_5, Jimmy wants to sell his new handmade lemonades. The utility function $U(s_5)(\overline{r}) = 2 \cdot d + m$ 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 $Z_4(\overline{r}) = (\overline{r})$, moves Jimmy back to s_3.

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

Next we present some auxiliary definitions. First, we define the maximal time a `USM` is allowed to idle. Intuitively, this limit will be given by the minimum between the minimal time in which the machine has to change its state and the

time in which the oldest pending account expires. If the minimum is given by the second value then we will use two auxiliary predicates to denote two special situations. 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. Let us note that 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. Therefore, that information will have a special relevance to assess whether an implementation fulfills the requirements imposed by a specification. After introducing these predicates, we will present how the list of pending accounts must be updated when a new transaction is performed. If the sign of the new transaction coincides with those of the listed transactions (that is, either all of them are positive or all of them are negative) then this transaction will be added to the list. Otherwise, the value of the new transaction will *compensate* the listed transactions as much as its value can, from the oldest transaction to the newest. Finally, we will define how to add the profit accumulated in the list of pending accounts.

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

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

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:

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

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. In the definition of `Update`, conditions such as $\frac{n}{m} \geq 0$ denote that n and m have the same sign. As we 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 both signs are opposite then a compensates the oldest profit p_1 as much as its value can. In this case, if $\frac{p_1+a}{p_1} \geq 0$ then the absolute value of a is lower than the absolute value of p_1. So, a does not completely compensates p_1. If $\frac{p_1+a}{p_1} < 0$ then we have the opposite situation, that is, p_1 is completely eliminated and the remainder of a is recursively applied to the rest of the list. We have used a functional programming notation to define lists: $[\,]$ denotes an empty list, $l +\!\!+ l'$ denotes the concatenation of the lists l and l', and $x : l$ denotes the inclusion, as first element, of x into the list l.

Example 2. Let us revisit Example 1. After a month, let us suppose that Jimmy is in configuration $(s_3, (31, 0, 4, 0, 8.50), [\,])$. Besides, let us suppose that Jimmy accepts payments for the lemonade in instalments. Today, Timmy consumes a lemonade at \$2 but pays \$1 as down payment. Besides, the next day Johnny consumes a lemonade at \$2 and pays only \$0.50 as deposit. Let us suppose that the amortization time of Jimmy is a week. Then, a possible configuration for Jimmy is $(s_3, (32, 0, 2, 0, 10), [(-1, 38), (-1.50, 39)])$. At this time, Mr. Brown buys a lemonade, paying cash, for \$3.50. Then, the new configuration for Jimmy is $(s_3, (32, 0, 1, 0, 13.50), [(-1, 39)])$. □

Given a `USM` M, an *evolution* of M represents a configuration that the `USM` can take from a previous configuration. Formally, evolutions are tuples $(c, c', tc)_K$ where c and c' are the previous and new configurations, respectively, tc is the time consumed by the evolution, and $K \in \{\alpha, \beta, \beta', \gamma\}$ is the type of evolution (*changing the state*, *passing of time*, *passing of time with failure*, and *performing a transaction*). The difference between β and β' transitions is that in the second case the agent fails its economic objectives since a past negative profit expires at its amortization time.

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 $\overline{r} = (u, r_1, \ldots, r_n)$ 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 $K = \alpha$, $tc = 0$, $s' = s''$, and $\overline{r'} = (0, r'_1, \ldots, r'_n)$, 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 $tr \in \mathbb{R}_+$ such that $0 < tr \leq \texttt{MaxWait}(M, c) - u$, we have $tc = tr$, $s' = s$, $\overline{r'} = (u + tr, r_1, \ldots, r_n)$, and l' is defined as follows:

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

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 $U(s)(\overline{r''}) - U(s)(\overline{r}) + \texttt{Accumulated}(l) > -mi$, 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. □

Let us comment the previous definition. First, 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 by. 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, we label the transition by β' to denote an undesirable behavior of the agent associated to the corresponding `USM`. 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. When a transaction is executed then the pending accounts are updated to register the new transaction. Let us remark 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 $\texttt{ValidTraces}(M, c)$. □

3 Composing Single Agents to Create Systems

In the previous section we have defined the full set of valid transitions a `USM` can perform. Thus, the definition of the formal specification framework for a single agent is complete. At this point we have to extend the previous framework so that *systems*, made of `USMs` interacting among them, can be defined. This notion will allow us 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`.

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 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 for longer periods since a long period of time could be denoted by several *time steps*. Let us note that passing of time will affect equally to all the `USMs` in the system. In other words, if a certain amount of time passes for one machine then it must also pass for any other machine of the system. Regarding synchronization of transactions, we will suppose that they are *conservative* in the sense that the total amount of resources existing in a system remains invariant after a transaction is performed. 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. Once again, a transition 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, we distinguish 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. □

Let us comment the previous definition. 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 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.

Next we present a useful result in order to understand the behavior of transactions. We show 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 $1 \leq j \leq m$ we have $M_j = (S_j, s_{j\,in}, V_j, U_j, at_j, mi_j, T_j)$. Besides, let σ be a trace of S such that $\sigma = [(c_1, c_2, t_1)_{K_1}, \ldots, (c_{n-1}, c_n, t_{n-1})_{K_{n-1}}]$, where for any $1 \leq j \leq n$ we have $c_j = ((s_1^j, \overline{r_1^j}, l_1^j), \ldots, (s_m^j, \overline{r_m^j}, l_m^j))$. Let us suppose that for some $1 \leq k \leq m$ we have $l_k^n = [\,]$ and let $1 \leq i \leq n$ be such that $l_k^i = [\,]$ and for any $i \leq j \leq n$ it holds $s_k^j = s_k^i$ and $K_j \notin \{\beta_A \mid k \in A\}$. Let us also suppose that $\frac{\Delta U_k(s_k)(\overline{x})}{\Delta t} = 0$ holds. Under these conditions we have $U(s_k)(\overline{r_k^n}) \geq U(s_k)(\overline{r_k^i})$.

Proof. The agent M_k stays in the same state s_k^i from configuration c_i to configuration 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 configuration c_i to configuration c_n. Let us suppose $U(s_k)(\overline{r_k^n}) < U(s_k)(\overline{r_k^i})$. Then, since the 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 $j \in Q$ it holds $K_j = \gamma$ and $U(s_k)(\overline{r_k^j}) < U(s_k)(\overline{r_k^{j-1}})$ must be non-empty. Let us note that for each $j \in Q$ 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 (which is impossible, since M_k did not fail) or it was compensated before achieving c_n. Nevertheless, if it was compensated then it cannot influence negatively the final utility of M_k. Thus, we conclude that $U(s_k)(\overline{r_k^n}) < U(s_k)(\overline{r_k^i})$ is not possible.

□

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. We say that 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 $\texttt{ValidTraces}(S, c, M_i)$. □

4 Case Study: Kasbah

In this section we apply our formalism to specify the system Kasbah [3]. Kasbah 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. Basically, 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. Let us suppose that he wants to buy an item (the case of selling is symmetric). The user chooses the item he wants to buy and configures the agent according to four requirements: The initial price it should offer to an agent selling that item (ip), the maximal price it should pay in exchange of the item (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. So, 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 the buying agent is 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 agent 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

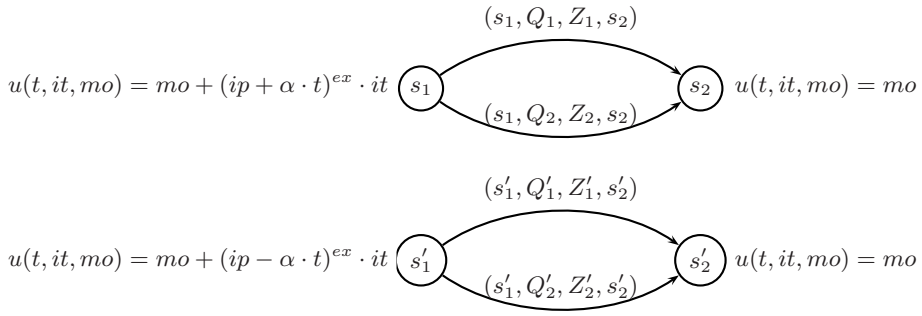

Fig. 2. `USMs` of the buyer and seller agents.

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 10. The `USM` M_b associated to a buyer agent is defined as the tuple $(S, s_{in}, V, U, at, mi, T)$, 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 definition of the `USM` M_s associated with sellers is quite similar. The only differences are the utility function in the first state s'_1, which is defined as we said before, and the functions denoting the modification of resources in transitions. In this case, the roles of Z_1 and Z_2 must be swaped, as the item will be returned to the user in the case that it could not be sold. In Figure 2 we show graphical representations of the `USMs` corresponding to buyers and sellers. 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 defined containing several buyers and sellers that trade with different items.

5 Conclusions and Future Work

In this paper we have presented a framework to formally specify e-commerce systems where agents represent the interest of the users. We concentrated on

the high level part of the behavior of agents. We have used a notion, called *utility state machines*, that allows us to specify the economic behavior of agents. Finally, we have applied our framework to the specification of the main entities appearing in Kasbah.

We contemplate two lines for future work. In the first one, with a major theoretical component, we plan to extend our notion of `USM` so that low level behavior is also taken into account. In the second one, with a more practical component, we would like to apply our framework to other agent-based e-commerce systems (as we have done with Kasbah).

References

1. A. Bhimani. Securing the commercial Internet. *Communications of the ACM*, 39(6):29–35, 1996.
2. S. Braynov. Tutorial on *agent-mediated electronic commerce* at AIMSA'02, 2002.
3. A. Chavez and P. Maes. Kasbah: An agent marketplace for buying and selling goods. In *PAAM'96*, pages 75–90, 1996.
4. M. Dastani, N. Jacobs, C.M. Jonker, and J. Treur. Modelling user preferences and mediating agents in electronic commerce. In *Agent Mediated Electronic Commerce, LNCS 1991*, pages 163–193. Springer, 2001.
5. B. Geisler, V. Ha, and P. Haddawy. Modeling user preferences via theory refinement. In *Int. Conf. on Intelligent User Interfaces*, pages 87–90. ACM Press, 2001.
6. R. Guttman, A. Moukas, and P. Maes. Agent-mediated electronic commerce: A survey. *The Knowledge Engineering Review*, 13(2):147–159, 1998.
7. V. Ha and P. Haddawy. Similarity of personal preferences: Theoretical foundations and empirical analysis. *Artificial Intelligence*, 146(2):149–173, 2003.
8. 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.
9. 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.
10. 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.
11. M. Ma. Agents in e-commerce. *Communications of the ACM*, 42(3):79–80, 1999.
12. 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.
13. I. Rodríguez. Formal specification of autonomous commerce agents. In *19th ACM Symposium on Applied Computing, SAC'04*, pages 774–778. ACM Press, 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.