A Formal Environment Model for Multi-Agent Systems

Paulo Salem da Silva and Ana C. V. de Melo

University of São Paulo
Department of Computer Science
São Paulo – Brazil
salem@ime.usp.br acvm@ime.usp.br

Abstract. Multi-agent systems are employed to model complex systems which can be decomposed into several interacting pieces called agents. In such systems, agents exist, evolve and interact within an environment. In this paper we present a model for the specification of such environments. This *Environment Model for Multi-Agent Systems* (EMMAS), as we call it, defines both structural and dynamic aspects of environments. Structurally, EMMAS connects agents by a social network, in which the link between agents is specified as the capability that one agent has to act upon another. Dynamically, EMMAS provides operations that can be composed together in order to create a number of different environmental situations and to respond appropriately to agents' actions. These features are founded on a mathematical model that we provide and that defines rigorously what constitutes an environment. Formality is achieved by employing the π-calculus process algebra in order to give the semantics of this model. This allows, in particular, a simple characterization of the evolution of the environment structure. Moreover, owing to this formal semantics, it is possible to perform formal analyses on environments thus described. For the sake of illustration, a concrete example of environment specification using EMMAS is also given.

1 Introduction

Multi-agent systems (MAS) [13] can be used to model complex systems in which the entities to be studied can be decomposed into several interacting pieces called *agents*. Human societies, computer networks, neural tissue and cell biology are examples of systems that can be seen from this perspective. Given a MAS, one technique often employed to study it is simulation [2]. That is, one may implement the several agents of interest, compose them into a MAS, and then run simulations in order to analyse their dynamic behavior. In such works, the analysis method of choice is usually the collection or optimization of statistics over several runs (e.g., the mean value of a numeric variable over time). Examples of this approach include platforms such as Swarm [5], MASON [3] and Repast [6]. There are, however, other possibilities for analysing such simulations. The crucial insight here is that simulations can be seen as incomplete explorations of state-spaces, and thus can be subject to some kinds of formal analyses.

A MAS can be decomposed into two aspects. The first relates to the agents. The second deals with how such agents come together and interact among themselves. The elements that form this second aspect constitute the *environment*[1] of a MAS.

That said, our overall work is concerned with how one can build a MAS to model a complex situation suitable for both exploratory simulation and approximate formal verification. To achieve this, we aim at providing three basic elements: (i) an agent model, which we have already described in [10]; (ii) a formal specification of the environment of these agents, so that they can be composed into a MAS; and (iii) techniques to formally analyse the resulting MAS.

In this paper we focus on the problem of defining environments. Our environments have a social network structure in which nodes are agents, and the links between them are defined by the capabilities that agents have to act upon each other. Furthermore, environments are more than a network structure, as they may change dynamically, either spontaneously or as a reaction to an agent's actions. These design choices arise from the agent model that we consider [10]. In it, agents are described from the point of view of behavioral psychology [11], which suggests a number of desirable features from an environment that brings them together. For instance, great importance is placed on the possibility of performing experiments of different kinds, and of responding to agent's actions in appropriate ways. As we shall see, our approach achieves this by the *environment behaviors* it defines. Furthermore, interaction is mostly interestingly treated by abstracting physical properties away and dealing only with relationships, which we do by adopting a social network structure and operations to modify it. We believe that these characteristics already differentiate our work substantially from other existing environment description methods (see Weyns *et al.* [14] for a survey).

Here we develop a simple formal framework in which to define such environments so that they can be subject to automated analyses procedures. A mathematical model is provided, which we call the *Environment Model for Multi-Agent Systems* (EMMAS), and its semantics is given in terms of the π-calculus process algebra [4,7].

Process algebras are typically employed to describe concurrent systems. They are good at succinctly describing behaviors relevant for inter-process communication. Our particular choice of π-calculus as a theoretical foundation is motivated by a number of its distinguishing features among existing such algebras. First, it takes communication through channels as a primitive notion, which makes it a natural choice for representing networks. Second, it allows for dynamic modification, which makes the creation and destruction of connections between agents possible. Third, it provides a convenient representation for broadcast behavior

[1] Notice that the term "environment" is not used consistently in the MAS literature [14]. Sometimes, it is used to mean the conceptual entity in which the agents and other objects exist and that allows them to interact; sometimes, it is used to mean the computational infrastructure that supports the MAS (e.g., a simulator). We use the term in the former sense.

through its replication operator. Finally, it has few operators and a simple operational semantics, which is attractive for implementation.

It is worth to note that despite all of these qualities of process algebras in general, and of π-calculus in particular, they are not usually employed in the context of multi-agent systems simulation. One exception is the work of Wang and Wysk [12], which uses a modified π-calculus to express a certain class of agents and their environments. But their approach is not sufficient to deal with our problems, and thus we develop our own method.

We purposefully treat agents as black-boxes here. This does not mean that they have no known internal structure; it merely means that such structure is mostly irrelevant as far as their environment is concerned. We assume, thus, that those two aspects of a MAS are complementary, but separate, issues. However, there must be a way to interface the agents with their environment. This is achieved through the assumption that agents receive *stimuli* as input and that they output *actions*.

The text is organized as follows. Section 2 introduces the basic features of the model, and also provides their semantics. Section 3, in turn, defines a number of convenience elements, which are not fundamental, but form a valuable specification repertoire. The reader is supposed to be familiar with the π-calculus process algebra, though the presented specifications are straightforward and should perhaps be accessible to anyone with some knowledge of process algebras. Section 4 presents a concrete example of an EMMAS specification. At last, Sect. 5 summarizes the main points presented and considers the new perspectives that EMMAS brings. The present text is based on and an evolution of a longer technical report [9], which the reader might wish to consult as well.

For the sake of readability, we have omitted π-calculus input and output parameters when such parameters are not relevant (e.g., we write $\overline{a}$ instead of $\overline{a}(x)$ if x is not used later).

2 Environment Model

Our *Environment Model for Multi-Agent Systems* (EMMAS) is a mathematical framework that can be used to specify environments for multi-agent systems. Its translation to the π-calculus process algebra is achieved using a translation function to map constructs of EMMAS into π-calculus expressions (i.e., a construct C is translated to $[C]_\pi$). The full definition of such a function will be given as new constructs are introduced, and for the moment the following suffices.

Definition 1 (Translation function). *The* translation function $[\ ]_\pi$ *maps constructs of EMMAS into π-calculus expressions.*

2.1 Underlying Elementary π-Calculus Events

A π-calculus specification can be divided into two parts. First, and most fundamentally, it is necessary to specify the set of events that are particular to that

specification. Second, it is necessary to specify processes built using those events. In this section we account for this first part.

Input and output events are all made from basic names. Hence, we first formally define a set of names in order to have the corresponding events. The definition below define such names, and Table 1 explains the events that arise.

Definition 2 (Environment Names). *The* environment names *are defined by the following set:*

$$\begin{aligned} ENames = \{&emit_a^n, stop_a^n, beginning_s^n, stable_s^n, absent_s^n, \\ &destroy_{a,n}^{s,m}, ccn, done \mid \\ &a \in Actions, s \in Stimuli, m, n \in AgentIDs\} \end{aligned}$$

Moreover, the set of environment events *that immediately follow from ENames is called EEvents.*

Notice that names are primitive entities, even though they are denoted here with subscripts and superscripts, which could suggest some sort of parametrization. This writing style is merely for readability's sake.

Table 1. Informal description of events, divided in three categories according to their origin and destination. The corresponding output or input events not shown merely allow the ones described to work properly.

Event	**Informal description**
Agent to environment	
$\overline{emit_a^n}$	Agent identified by n performs action a.
$\overline{stop_a^n}$	Agent identified by n stops performing action a.
Environment to agent	
$beginning_s^n$	Delivery of stimulus s to the agent identified by n is beginning.
$stable_s^n$	Delivery of stimulus s to the agent identified by n is stable.
$ending_s^n$	Delivery of stimulus s to the agent identified by n is ending.
$absent_s^n$	Delivery of stimulus s to the agent identified by n becomes absent.
Environment to environment	
$\overline{destroy_{a,n}^{s,m}}$	Requests the destruction of an action transformer that converts action a from agent identified by n into stimulus s accepted by the agent identified by m.
$\overline{ccn}$	Requests the creation of a new action transformer.
$\overline{done}$	Signals that an operation has terminated.

2.2 Operations

In order to exhibit dynamic behavior, the environment depends on *operations* to modify its structures.

Definition 3 (Operation). *An* operation *is any π-calculus expression such that:*

- *its names belong to the set $ENames$;*
- *it signals its termination with the $\overline{done}$ event.*

The second condition is particularly important because it will allow the sequential composition of operations, as we shall see in Sect. 3.1.

Of course such an abstract definition of operations cannot be used directly. Nevertheless, it suffices to define the basic model for environments. Concrete operations shall be given in Sect. 3.2.

2.3 Environment Structures

The *environment* is the central structure of EMMAS specifications. It defines which agents are present, how they are initially connected, and what dynamic behaviors exist in the environment itself. The presentation below follows a top-down approach. We begin by defining the overall environment, and then proceed to examine the nature of its constituent parts.

Definition 4 (Environment). *An* environment *is a tuple $\langle AG, AT, EB\rangle$ such that:*

- *$AG = \{ag_1 \dots ag_l\}$ is a set of* agent profiles*;*
- *$AT = \{t_1 \dots t_m\}$ is a set of* action transformers*;*
- *$EB = \{eb_1 \dots eb_n\}$ is a set of operations (Def. 3), which are called here* environment behaviors.

Moreover, let $ENames = \{en_1, \dots, en_o\}$. Then the corresponding π-calculus expression for the environment is defined as:

$$\begin{aligned}[\langle AG, AT, EB\rangle]_\pi = (&\nu\ en_1, \dots, en_o)\\ &([ag_1]_\pi|[ag_2]_\pi|\dots|[ag_l]_\pi|\\ &[t_1]_\pi|[t_2]_\pi|\dots|[t_m]_\pi|\\ &[eb_1]_\pi|[eb_2]_\pi|\dots|[eb_n]_\pi|\\ &!NewAT)\end{aligned}$$

where

$$\begin{aligned}NewAT = ccn\langle &emit, stop, absent, beginning, stable, ending, destroy\rangle.\\ &T(emit, stop, absent, beginning, stable, ending, destroy)\end{aligned}$$

and T is given in Def. 6.

This definition merits a few comments. First, all elements are put in parallel composition, which allows them to interact. Notice that all names from $ENames$ are restricted to the environment, which ensures that events are always used in such an interaction (i.e., events cannot be sent to outside the environment

process, and therefore can only be used internally). Second, the set of action transformers provide the network structure that connects the agents, as we shall shortly see. Third, the environment behaviors, as the name implies, specifies behaviors that belong to the environment itself. This is useful to model reactions to agents' actions, as well as to capture ways in which the environment may evolve. This is achieved through operations provided by the specifier. Finally, the component $NewAT$ allows the creation of new action transformers. In order to do so, it receives a message $\overline{ccn}$ ("create connection"), whose parameters initialize the rest of the expression. We shall see an operation that does this in Sect. 3.2.

Environments exist in order to allow agents to interact. As we remarked earlier, the internal structure of these agents, as complex as it may be, is mostly irrelevant for their interaction model. Thus, we have abstracted it away as much as possible. What is left are the interfaces that allow agents to interact with each other and with the environment itself, which we call *agent profiles*. Hence, we have the following definition.

Definition 5 (Agent Profile). *An* agent profile *is a triple* $\langle n, S, A\rangle$ *such that:*

- $n \in AgentIDs$ *is a unique identifier for the agent;*
- $A = \{a_1 \dots a_i\} \subseteq Actions$ *is a set of actions;*
- $S = \{s_1 \dots s_j\} \subseteq Stimuli$ *is a set of stimuli.*

Moreover,

$$[\langle n, S, A\rangle]_\pi = ([Act(a_1, n)]_\pi | [Act(a_2, n)]_\pi | \dots | [Act(a_i, n)]_\pi) | \\ ([Stim(s_1, n)]_\pi | [Stim(s_2, n)]_\pi | \dots | [Stim(s_j, n)]_\pi)$$

such that, for all $a \in A$ *and* $s \in S$*, we have:*

$$[Act(a, n)]_\pi = !(\overline{emit_a^n}.\overline{stop_a^n})$$

$$[Stim(s, n)]_\pi = piStim(beginning_s^n, stable_s^n, ending_s^n, absent_s^n)$$

where

$$piStim(beginning, stable, ending, absent) = \\ beginning.stable.ending.absent.piStim(beginning, stable, ending, absent)$$

In this definition, it is clear that agents have several components, each responsible for controlling one particular action or stimulus. $Act(a, n)$ defines that the agent identified by n can start emitting an action a and can then stop such emission. The replication operator ensures that this sequence can be carried out an unbounded number of times. $Stim(s, n)$, in turn, defines that the agent identified by n can be stimulated by s, and that this stimulation follows four steps (i.e., *absent*, *beginning*, *stable* and finally *ending*). The recursive call ensures that this stimulation sequence can start again as soon as it finishes the last step.

These definitions reflect the assumptions about the agent model we consider [10], which, in particular, defines precise – internal – consequences for each of these stimulation steps.

Agents interact by stimulating each other. But to have this capability, it is first necessary to define that an agent's action causes a stimulation in another agent. This is done through *action transformers*, which specifies that if agent ag_1 performs the action a, then agent ag_2 should be stimulated with s.

Definition 6 (Action Transformer). *An* action transformer *is a tuple* $\langle ag_1, a, s, ag_2 \rangle$ *such that:*

- ag_1 *is an agent profile* $\langle n, S_1, A_1 \rangle$*;*
- ag_2 *is an agent profile* $\langle m, S_2, A_2 \rangle$*;*
- a *is an action such that* $a \in A_1$*;*
- s *is a stimulus such that* $s \in S_2$*;*

Moreover, the corresponding π*-calculus expression for the action transformer is defined as:*

$$[\langle ag_1, a, s, ag_2 \rangle]_\pi = T(emit_a^n, stop_a^n, absent_s^m, beginning_s^m, stable_s^m, ending_s^m, destroy_{a,n}^{s,m})$$

where

$$T(emit, stop, absent, beginning, stable, ending, destroy) = \overbrace{(emit.\overline{beginning}.\overline{stable}.stop.\overline{ending}.\overline{absent}\,.\, T(emit, stop, absent, beginning, stable, ending, destroy))}^{Normal\ behavior} + \underbrace{destroy}_{To\ disable\ the\ action\ transformer}$$

The above definition can be divided in two parts. First, there is its normal behavior, which merely defines the correct sequence through which an action is transformed in a stimulus. Once such a sequence is completed, a recursive call to the process definition restarts the action transformer. Second, there is the part that allows the transformer to be destroyed. By performing *destroy*, the action transformer disappears, since this event is not followed by anything. Figure 1 shows an example of environment in which the role of action transformers can be appreciated.

We choose to have an intermediate structure such as the action transformer between the agents instead of allowing a direct communication because an agent's actions may have other effects besides stimulation. In particular, the environment can also respond to such actions in custom ways through the specified environment behaviors.

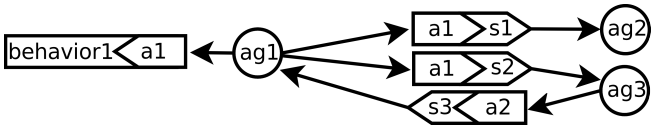

Fig. 1. An example of environment. Circles denote agent profiles ag_1, ag_2 and ag_3. There are three action transformers: $\langle ag_1, a_1, s_1, ag_2\rangle$, $\langle ag_1, a_1, s_2, ag_3\rangle$ and $\langle ag_3, a_2, s_3, ag_1\rangle$. Moreover, there is also an environment behavior, $behavior_1$, that is executed whenever agent ag_1 performs action a_1. Notice that the same action a_1, performed by agent ag_1, has three simultaneous consequences. Notice further that while two of these consequences are stimulations, another merely triggers some operation. This shows that it is technically interesting to have actions and stimuli as different entities, since they are not always related.

2.4 Semantics

The semantics of EMMAS is given by considering: (i) a syntactical translation of EMMAS into π-calculus expressions; and (ii) a mathematical foundation which relates π-calculus events to the stimuli and actions of agents. The π-calculus translation of (i), through its operational semantics, provides an over-approximation of the desired behavior, which is then made precise using the restrictions provided by (ii). By this method, we shall be able to build an LTS that defines the possible states and transitions for any particular environment specification.

For the sake of clarity, we divide this section in two parts. First we define some preliminary structures required for building the transition system, which is then presented.

Preliminary Definitions Our model must have a way to effectively interact with the agents of a MAS. Agents may trigger events that have a meaning in the environment specification (e.g., the performance of an action). Conversely, the environment specification may request the performance of an operation (e.g., to stimulate an agent). It is necessary, therefore, to have a mathematical foundation that formally defines how to accomplish this. We fulfill this requirement by providing both a *vocabulary* in which a few primitives are defined and a definition for what constitutes an *environment status* with respect to these primitives.

Definition 7 (Vocabulary). *A* vocabulary *is a tuple*

$$\langle Stimuli, Actions, AgentIDs\rangle$$

such that:

- *Stimuli is a finite set of* stimuli*;*
- *Actions is a finite set of* actions*;*
- *AgentIDs is a finite set of* agent identifiers*;*

The sets *Stimuli*, *Actions* and *AgentIDs* define, respectively, all available stimuli, actions and agent identifiers. These are sets containing primitive, unstructured, elements.

Definition 8 (Environment Status).
An environment status *is a pair*

$$\langle Stimulation, Response \rangle$$

such that:

- $Stimulation : AgentIDs \times Stimuli \to \{Beginning, Stable, Ending, Absent\}$;
- $Response : AgentIDs \times Actions \to \{Emitting, NotEmitting\}$.

Building the Transition System Given an environment E, we shall build an *annotated environment LTS* by considering the LTS induced by $[E]_\pi$, whose states shall be annotated with our environment status (Def. 8), and whose structure shall be subject to some restrictions based on the possible values for an environment status. Then we shall then have an LTS whose states have the following form.

Definition 9 (State). *Let E be an environment and P be a π-calculus process obtained by applying π-calculus operational semantics rules to $[E]_\pi$. Moreover, let $\langle Stimulation, Response \rangle$ be an environment status. Then a* state *is defined as the following pair:*

$$(P, \langle Stimulation, Response \rangle)$$

By this construction, at any point of the LTS we shall be able to know both what is the current situation of the agents (because of the added environment status) and what are the possible changes from that point (because of the π-calculus operational semantics).

To proceed with this construction, we need a number of definitions. Let us begin by providing a way to observe the internal transitions of an environment, which is a fundamental capability that we need before proceeding. Recall from Def. 4 that an environment's π-calculus process has a number of restrictions that would prevent such observations (i.e., the transitions would be internal to the process and not discernible in the LTS). It is, however, possible to characterize these restrictions syntactically, and thus we may provide a simple method to remove them when needed. This is accomplished by the following *environment unrestriction function unr*.

Definition 10 (Environment Unrestriction Function).
Let P and Q be π-calculus processes such that

$$P = (\nu\ en_1, \ldots, en_o)Q$$

where $\{en_1, \ldots, en_o\} = ENames$. Then the environment unrestriction function *is defined as $unr(P) = Q$.*

We may now define the *Stimulation* function present in each state as follows.

Definition 11 (Stimulation).
Let $(P, \langle Stimulation, Response\rangle)$ be a state. Moreover, let $\rightarrow$ be the transition relation induced by the π-calculus operational semantics. Then, for all $s \in Stimuli$ and $n \in AgentIDs$, we have:

$$Stimulation(n,s) = \begin{cases} Absent & if \quad \exists P' such\ that\ unr(P) \overset{beginning_s^n}{\rightarrow} P' \\ Beginning & if \quad \exists P' such\ that\ unr(P) \overset{stable_s^n}{\rightarrow} P' \\ Stable & if \quad \exists P' such\ that\ unr(P) \overset{ending_s^n}{\rightarrow} P' \\ Ending & if \quad \exists P' such\ that\ unr(P) \overset{absent_s^n}{\rightarrow} P' \end{cases}$$

The *Stimulation* definition establishes the status of a particular stimulation based on the order that stimulations must change (see Def. 5). For instance, if a process is capable of receiving a $beginning_s^n$ event, it must be the case that stimulus s is currently absent in agent identified by n. The *Stimulation* function, therefore, merely gives a way of reading the π-calculus LTS in order to have this information explicitly for every agent and stimulus in any given process.

The *Response* function, on the other hand, is assumed as given (e.g., by a simulator that implements the black-box behavior of the agents). Thus, we do not define it. However, it imposes some constraints on the LTS, which we must specify and take in account. As we shall see shortly, these constraints turn the π-calculus over-approximation into an exact description of the transition system's structure that we wish to assign to EMMAS.

Definition 12 (Transition constraints).
Let $s_1 = (P_1, \langle Stimulation_1, Response_1\rangle)$ and $s_2 = (P_2, \langle Stimulation_2, Response_2\rangle)$ be states in an annotated environment LTS $\langle S, L, \rightsquigarrow\rangle$. Moreover, let $\rightarrow$ be the transition relation induced by the π-calculus operational semantics. Then the transition $s_1 \overset{l}{\rightsquigarrow} s_2$ is forbidden if one of the cases hold:

- *there exists $a \in Actions$ and $n \in AgentIDs$ such that:*
  - $Response_1(n,a) = Emitting$;
  - *P_2 was obtained by internally producing the event $\overline{stop_a^n}$ in P_1.*
- *there exists $a \in Actions$ and $n \in AgentIDs$ such that:*
  - $Response_1(n,a) = NotEmitting$;
  - *P_2 was obtained by internally producing the event $\overline{emit_a^n}$ in P_1.*
- *there exists $a \in Actions$ and $n \in AgentIDs$ such that:*
  - $Response_1(n,a) = Emitting$;
  - $Response_2(n,a) = NotEmitting$;
  - *there exists a P' such that $unr(P_1) \overset{emit_a^n}{\rightarrow} P'$.*
- *there exists $a \in Actions$ and $n \in AgentIDs$ such that:*
  - $Response_1(n,a) = NotEmitting$;

- $Response_2(n, a) = Emitting$;
- *there exists a* P' *such that* $unr(P_1) \stackrel{stop_a^n}{\rightarrow} P'$.

At last, we may define the annotated environment LTS as follows.

Definition 13 (Annotated Environment LTS). *Let* E *be an environment (Def. 4), and let* $\rightarrow$ *be the transition relation induced by the* π*-calculus operational semantics. Then an* annotated environment LTS *is an LTS* $\langle S, L, \rightsquigarrow \rangle$ *such that:*

- $L = EEvents$ *(see Def. 2);*
- S *and* $\rightsquigarrow$ *are constructed inductively as follows:*
  - ***Initial state***. $([E]_\pi, es) \in S$, *where* $es = \langle Stimulation, Response \rangle$ *such that for all* $a \in Actions$, $s \in Stimuli$, *and* $n \in AgentIDs$ *we have* $Stimulation(n, s) = Absent$ *and* $Response(n, a) = NotEmitting$.
  - ***Other states and transitions***.
    If $s_1 = (P_1, \langle Stimulation_1, Response_1 \rangle) \in S$,
    then $s_2 = (P_2, \langle Stimulation_2, Response_2 \rangle) \in S$ *and* $s_1 \stackrel{l}{\rightsquigarrow} s_2$ *if and only if:*
    - $P_1 \stackrel{l}{\rightarrow} P_2$;
    - $Stimulation_2$ *is defined w.r.t.* P_2 *according to Def. 11;*
    - $s_1 \stackrel{l}{\rightsquigarrow} s_2$ *is not forbidden by Def. 12.*

3 Convenience Elements and Operations

So far we have defined the bare minimum for describing environments so that they can be formally analysed. Clearly, though, more constructs are necessary in order to make such specifications. For example, in Def. 3 we established what is an operation in general, but we have not presented any particular one. In the present section, then, we provide a number of convenience elements that can be used to build concrete EMMAS models. These, however, are merely examples of what can be expressed with the basic model given before, designed to show its usefulness, and the reader may well imagine many other convenience elements.

3.1 Composition Operators

In order to build complex operations on top of the basic ones, it is useful to define composition operators. Some of these can be mapped directly to π-calculus operators, but others require more sophistication.

Definition 14 (Sequential Composition). *Let* Op_1 *and* Op_2 *be operations. Then their* sequential composition *is also an operation and is written as:*

$$Op_1; Op_2$$

Moreover,

$$[Op_1; Op_2]_\pi = (\nu\ start)[Op_1]_\pi\{start/done\}|start.[Op_2]_\pi$$

The above translation aims at accounting for the intuition that Op_1 must take place before Op_2. However, we cannot translate $Op_1;Op_2$ immediatly as $[Op_1]_\pi.[Op_2]_\pi$, because in general π-calculus would not allow the resulting syntax (e.g., $(P+Q).R$ would not be a valid expression). Therefore, we adapt the suggestion offered by Milner [4] (in Example 5.27), which requires every operation to signal its own termination with a *done* event.

Definition 15 (Sequence). *Let Op be an operation and n be an integer such that $n \geq 1$. Then a* sequence *of n compositions of Op is defined as:*

$$Seq(Op, n) = \begin{cases} Op; Seq(Op, n-1) & n > 1 \\ Op & n = 1 \end{cases}$$

Definition 16 (Unbounded Sequence). *Let Op an operation. Then an* unbounded sequence *of compositions of Op is defined as:*

$$Forever(Op) = Op; Forever(Op)$$

The translation of these two kinds of sequences to π-calculus follows, of course, from the translation of the sequential composition operator.

Definition 17 (Choice). *Let Op_1 and Op_2 be operations. Then their composition as a* choice *is also an operation and is written as:*

$$Op_1 + Op_2$$

Moreover,

$$[Op_1 + Op_2]_\pi = [Op_1]_\pi + [Op_2]_\pi$$

Definition 18 (Parallel Composition). *Let Op_1, Op_2, ..., Op_n be n operations. Then their* parallel composition *is also an operation and is written as:*

$$Op_1 \parallel Op_2 \parallel \ldots \parallel Op_n$$

Moreover,

$$\begin{aligned}[Op_1 \parallel Op_2 \parallel \ldots \parallel Op_n]_\pi = (\nu\ start)[Op_1]_\pi\{start/done\}|[Op_2]_\pi\{start/done\}|\ldots| \\ [Op_n]_\pi\{start/done\}|\underbrace{start.start.\ldots.start}_{n\ times}.\overline{done}\end{aligned}$$

The translation for the parallel composition is not straightforward because it is necessary to ensure that $\overline{done}$ is sent only once in the composed operation. That is to say, the parallel composition of n operations[2] is an operation itself, and it only terminates when each of its components terminates.

[2] We define the operator for n operations instead of just two because this avoids the problem of establishing its associativity properties.

3.2 Core Operations

We can now provide a core of operations upon which others can be built.

Agent Stimulation Operations The following operations are provided to control the stimulation of agents.

Definition 19 (Begin stimulation operation). *Let $ag = \langle n, S, A\rangle$ be an agent profile, and $s \in S$ be a stimulus. Then the* begin stimulation *operation is writen as:*

$$BeginStimulation(s, ag)$$

Moreover,

$$[BeginStimulation(s, ag)]_\pi = \overline{beginning_s^n}.\overline{stable_s^n}.\overline{done}$$

Definition 20 (End stimulation operation). *Let $ag = \langle n, S, A\rangle$ be an agent profile, and $s \in S$ be a stimulus. Then the* end stimulation *operation is writen as:*

$$EndStimulation(s, ag)$$

Moreover,

$$[EndStimulation(s, ag)]_\pi = \overline{ending_s^n}.\overline{absent_s^n}.\overline{done}$$

Definition 21 (Stimulate operation). *Let $ag = \langle n, S, A\rangle$ be an agent profile, and $s \in S$ be a stimulus. Then the* stimulate *operation is defined as:*

$$Stimulate(s, ag) = BeginStimulation(s, ag); EndStimulation(s, ag)$$

Action Transformers Operations The following operations are provided to manipulate action transformers.

Definition 22 (Create action transformer operation). *Let $ag_1 = \langle n, S_1, A_1\rangle$ be an agent profile, $ag_2 = \langle m, S_2, A_2\rangle$ be another agent profile, $a \in A_1$ be an action, and $s \in S_2$ be a stimulus. Then the* create action transformer *operation is written as:*

$$Create(ag_1, a, s, ag_2)$$

Moreover,

$$[Create(ag_1, a, s, ag_2)]_\pi = \overline{ccn}(emit_a^n, stop_a^n, absent_s^m, beginning_s^m, stable_s^m, ending_s^m, destroy_{a,n}^{s,m}).\overline{done}$$

In the above definition, notice that $\overline{ccn}$ is crafted to react with the component $NewAT$ given in Def. 4. Since operations will ultimately be put together with parallel composition in the environment, it follows that the $Create(ag_1, a, s, ag_2)$ operation will be able to react with $NewAT$ and originate a new action transformer.

Definition 23 (Destroy action transformer operation). *Let $ag_1 = \langle n, S_1, A_1 \rangle$ be an agent profile, $ag_2 = \langle m, S_2, A_2 \rangle$ be another agent profile, $a \in A_1$ be an action, and $s \in S_2$ be a stimulus. Then the* destroy action transformer *operation is writen as:*

$$Destroy(n, a, s, m)$$

Moreover,

$$[Destroy(n, a, s, m)]_\pi = \overline{destroy}_{a,n}^{s,m}.\overline{done}$$

4 Example

Let us consider the following simple example. We shall specify an online social network, in which users may register themselves and interact.[3] The objective of the specification is to test advertisement strategies through simulation. To this end, we define an environment $\langle AG, AT, EB \rangle$ with n agents, such that each agent $ag_i \in AG$ is capable of performing the action buy (i.e., to buy the advertised product) and $sendMsg$ (i.e., to send some message to another agent), as well as receiving the stimuli gui_1, gui_2 (i.e., the graphical user interface of the website can be set in two different ways), ad_1, ad_2 (i.e., there are two different possible advertisements) and msg (i.e., a message received). Formally, $ag_i = \langle i, \{gui_1, gui_2, ad_1, ad_2, msg\}, \{buy, sendMsg\} \rangle$.

The action transformers among the agents allow them to send messages to their friends. Of course, the particular topology of this network can vary, but in essence each agent ag_i shall have some action transformers of the form $\langle ag_i, sendMsg, msg, ag_j \rangle$. The effect of such a message could be similar to that of an advertisement (i.e., a product recommendation by a friend).

Finally, and most importantly, for each agent $ag \in AG$, we define the following new environment behavior $eb_i \in EB$:

$(BeginStimulation(gui_1, ag_i) + BeginStimulation(gui_2, ag_i))$;
$(Stimulate(ad_1, ag_i) + Stimulate(ad_2, ag_i))$

These eb_i specify four possible simulation sequences concerning each agent. For instance, in some simulation run, agent ag_1 could be stimulated by $BeginStimulation(gui_1, ag_1)$ and then by $Stimulate(ad_1, ag_1)$. However, it could be that this particular sequence would be ineffective in eliciting the agent's buy action, in which case another sequence could be tried. The important thing,

[3] Actual examples of such networks include popular websites such as `www.facebbok.com`, `www.orkut.com` and `www.myspace.com`.

though, is that these trials can all be performed automatically, since they are explicit in the environment definition. This shows how EMMAS can endow simulators with some formal verification capabilities.

5 Conclusion

In this paper we have presented a formalization for environments of MASs. We provided a high-level description for this formalization, with a semantics given using the π-calculus. We found necessary to perform some adjustments on the standard behavior induced by the π-calculus' operational semantics in order to allow its integration with the remaining parts of the proposed approach. Furthermore, we avoided explicit temporal references in this formalization. However, it should be possible to add an explicit notion of time to EMMAS, though this would introduce new complications as well.

The presented environments have both structural and operational aspects. That is to say, they represent certain structures, which can then be changed by certain operations. These operations serve to two purposes. First, they provide a way to specify behaviors of the environments themselves (e.g., environment responses to the actions of agents). Second, they allow the succinct specification of several possible scenarios for an environment (e.g., several possible ways of stimulating agents). This latter possibility is one of the great advantages offered by the use of a process algebra as a semantic basis (e.g., an algebraic expression $a + b$ defines the non-deterministic *possibility* of either a or b), and renders our approach particularly unique insofar as environments for MASs are concerned. We may now formulate questions concerning the analysis of our MASs:

- Since the semantics of EMMAS is given as an LTS, it follows that now we need criteria for selecting paths in it. With such paths, we shall be able to perform concrete simulations.
- Concerning implementation, we believe that the π-calculus base can be particularly useful, since we could implement its few elements in order to have our whole model on top of it. A similar approach is taken by Wang and Wysk [12]. More generally, there are programming languages based on π-calculus, such as the Join-Calculus [1] and Pict [8].

Finally, though EMMAS is designed to work with a particular agent model [10], it actually imposes few restrictions on the agents, and its principles are general. Hence, it could perhaps be adapted to work with other agent models.

Acknowledgements

The authors would like to thank Prof. Dr. Marie-Claude Gaudel (Laboratoire de Recherche en Informatique, Université Paris-Sud 11) for her numerous comments and suggestions during the preparation of this work.

This project benefited from the financial support of *Coordenação de Aperfeiçoamento de Pessoal de Nível Superior* (CAPES) and *Conselho Nacional de Desenvolvimento Científico e Tecnológico* (CNPq).

References

1. Fournet, C., Gonthier, G.: The join calculus: A language for distributed mobile programming. In: In Proceedings of the Applied Semantics Summer School (APPSEM), Caminha. pp. 268–332. Springer-Verlag (2000)
2. Gilbert, N., Bankers, S.: Platforms and methods for agent-based modeling. Proceedings of the National Academy of Sciences of the United States 99(Supplement 3) (2002)
3. Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K.: MASON: A new multi-agent simulation toolkit (2004), http://cs.gmu.edu/ eclab/projects/mason/
4. Milner, R.: Communicating and Mobile Systems: the Pi-Calculus. Cambridge University Press (1999)
5. Minar, N., Burkhart, R., Langton, C., Askenazi., M.: The Swarm simulation system: A toolkit for building multi-agent simulations (1996), working Paper 96-06-042
6. North, M., Collier, N., Vos, J.R.: Experiences creating three implementations of the Repast agent modeling toolkit. ACM Transactions on Modeling and Computer Simulation 16(1), 1–25 (2006), http://repast.sourceforge.net/
7. Parrow, J.: An introduction to the pi-calculus. In: Bergstra, J.A., Ponse, A., Smolka, S.A. (eds.) Handbook of Process Algebra, pp. 479–543. Elsevier (2001)
8. Pierce, B.C., Turner, D.N.: Pict: A programming language based on the pi-calculus. In: Plotkin, G., Stirling, C., Tofte, M. (eds.) Proof, Language and Interaction: Essays in Honour of Robin Milner. MIT Press (1997)
9. da Silva, P.S.: An environment specification language for multi-agent systems (2009), Technical Report 1531 – Université Paris-Sud 11, Laboratoire de Recherche en Informatique.
10. da Silva, P.S., de Melo, A.C.V.: A simulation-oriented formalization for a psychological theory. In: Dwyer, M.B., Lopes, A. (eds.) FASE 2007 Proceedings. Lecture Notes in Computer Science, vol. 4422, pp. 42–56. Springer-Verlag (2007)
11. Skinner, B.F.: Science and Human Behavior. The Free Press (1953)
12. Wang, J., Wysk, R.A.: A pi-calculus formalism for discrete event simulation. In: WSC '08: Proceedings of the 40th Conference on Winter Simulation. pp. 703–711. Winter Simulation Conference (2008)
13. Weiss, G. (ed.): Multiagent systems: a modern approach to distributed artificial intelligence. MIT Press, Cambridge, MA, USA (1999)
14. Weyns, D., Parunak, H.V.D., Michel, F., Holvoet, T., Ferber, J.: Environments for multiagent systems: State-of-the-art and research challenges. In: *et al.*, D.W. (ed.) Proceedings of the 1st International Workshop on Environments for Multi-agent Systems (Lecture Notes in Computer Science, 3374). pp. 1–47. Springer (2005)