# A Hierarchical Methodology to Specify and Simulate Complex Computational Systems[*]

César Andrés, Carlos Molinero, and Manuel Núñez

Dept. Sistemas Informáticos y Computación.
Universidad Complutense de Madrid, Spain.
e-mail:{c.andres, molinero}@fdi.ucm.es, mn@sip.ucm.es

**Abstract.** We introduce a novel methodology to formally specify complex multi-agent systems. Our approach allows us to redefine computational problems in terms of agents that perform certain tasks. In our view, a system is formed by the combination of atomic and complex agents. Atomic agents are in charge of executing atomic tasks while complex agents reunite and summarize the properties of their underlying atomic agents. Basically, our approach consists in specifying the smaller parts of the problem as atomic agents. Each atomic agent is in charge of executing a small transformation of resources. Afterwards, the system will recombine them to form complex agents that will embrace the knowledge of several atomic agents. All agents are located on a superstructure of communication cellules created to record the hierarchy of the tasks. In order to provide a useful framework, we have developed a tool that fully implements all the stages of the methodology.

## 1 Introduction

Computational science embraces the concept of aiding the development of other studies in different fields through the use of new computational means. Therefore it has to create open systems that can be applied to a great extent of problems. In addition, it is relevant to take into account that the people to which computational science is directed are not, in general, computer scientists. Therefore, its easiness of use is a must. In this paper we report on a formalism that allows to solve complex problems through the use of agents. We propose a method to factorize the problem, being the first step to break down the problem into the smaller parts possible and assign an agent to each of those tasks. Then, the produced system allows to make petitions that will create other agents that, through recombination, are able to condense the information of several agents, so that they can solve a complex situation.

This paper extends and enhances our previous work presented in [1]. We have simplified some of the notations, so that the resulting formalism is much easier to use. Although we have simplified our approach, the expressive power of the framework remains the same, being able to solve the same problems that we confronted in [1].

Even though there are general purpose formalisms to formally describe complex concurrent systems (such as Process Algebras and Petri Nets) they are not suitable to

[*] Research supported by the Spanish MEC project WEST/FAST (TIN2006-15578-C02-01).

describe agents since these languages and notations do not provide specific operators to deal with the inherent characteristics of agents. However, there has been already several studies to formally describe the use of intelligent electronic agents that are nested into one another (see, for example, [2,3] for two approaches based on Petri Nets and automata, [4,5] for approaches based on process algebras, and [6,7] for approaches based on finite state machines). Most of these approaches have been created in favor of comprehensibility. Therefore they facilitate to derive and apprehend new properties. However, due to its complexity, these formalisms are not supported by suitable user-friendly tools. Thus, the specification of a system is a task that cannot be carried out by somebody that is not a real specialist in formal methods.

Our approach is able to assimilate the systems that we are interested in to a *common places* structure in which one is able to locate the rest of the structure from higher order points. If we use the subway lines as a metaphor, we only need to know the location of the different stations, but the exact location of that small fruit shop that we are trying to reach is bounded to the location of the closest metro station. Once we arrive to that particular metro station, we will check the neighborhood map so that we can find the shop; we do not need to know in advance all the local maps associated with all the stations of the network. This is how our systems will work: Once we have all the atomic agents, each time that a new complex agent, embracing the knowledge of several atomic agents, is created we will refer to this new agent when making subsequent calls to the system. In this line, we are able to *forget* how atomic actions are performed because we have a higher order element to which we can call upon. In any case, even with a complex structure, atomic agents are still the ones that execute *real* tasks.

Using another metaphor we could say that our approach produces systems that are similar to economic structures in which there exist intermediate agents that gives us the result of the transformation of resources as a final product. These agents, in a hidden way, contract the prime manufacturers that create these resource transformations. Another point in favor of our approach is that it allows us to have an unbounded growth (equivalently, subdivisions as small as needed) either by adding agents in between existing ones or by assigning new atomic agents to the system that we had before. It is important to note that the way our systems are subdivided, in so called *communication cellules*, facilitates their deployment in a distributed system in which one can obtain a perspective of variable magnitude of the global tasks. This holds as long as we keep the hierarchical structure of the ensemble.

The rest of the paper is organized as follows. In Section 2 we introduce some auxiliary notation. Section 3 represents the bulk of the paper. There we define the syntax of the proposed formalism, giving a running example of a system implemented with our tool. In Section 4 we briefly describe the technical details of the architecture of the tool developed to specify the systems. Finally in Section 5 we present our conclusions.

## 2 Preliminaries

In this section we introduce some notation that will be used throughout the rest of the paper. First, since users have different preferences, in order to properly design agents the first step consists in expressing these preferences. In order to extract preferences from users several mechanisms have been presented in the literature (see [8,9,10]). In this paper, preferences in a given moment will be given by a *utility function*. These functions associate a value (a utility measure) with each possible combination of resources a user could own. Alternatively, other mechanisms such as *preference relations* could be used (see e.g. [11] for conditions to transform one of the possibilities into the other).

In order to manage resources we will denote them as elements of a vector $\bar{x}$. We consider a *special* resource to record the performance of the system. The time that it takes to complete the tasks of the system will also be considered as another resource. A vector of resources is a vector of real numbers in which each number denotes the total amount of a specific resource. Along this paper we consider that $n$ is the number of resources of the system.

**Definition 1.** Let $\bar{x} \in \mathbf{R}^n$ be a *vector*. We have that $x_i$ represents the *$i$-th* component of $\bar{x}$. Let $\bar{x}, \bar{y} \in \mathbf{R}^n$ be two vectors. We write $\bar{x} + \bar{y}$ to denote the *addition* of $\bar{x}$ and $\bar{y}$. We say that $\bar{q}$ is the addition of $\bar{x}$ and $\bar{y}$ if $1 \leq i \leq n$ we have $q_i = x_i + y_i$.

We denote by $\bar{0} \in \mathbf{R}^n$ the vector having all the value components equal to zero. We write $\bar{x} \leq \bar{y}$ if for all $1 \leq i \leq n$ we have $x_i \leq y_i$.

A *utility function* is defined as any function $f^u : \mathbf{R}^n \to \mathbf{R}$. We denote the set of all utility functions by $\mathcal{F}$.

$\square$

Intuitively, given a utility function $f^u$, We say that $f^u(\bar{x}) > f^u(\bar{y})$ means that $\bar{x}$ is preferred to $\bar{y}$. For instance, if we have $\bar{x} = (x_1, x_2)$ representing the first element of the resource vector the number of apples and the second element the number of oranges, $f^u(\bar{x}) = 3 \cdot x_1 + 2 \cdot x_2$, means that, for example, the agent is equally happy owning 6 apples or 9 oranges. Let us consider another agent whose utility function is $f^u(\bar{x}) = 1 \cdot x_1 + 2 \cdot x_2$. Then, both agents can make a deal if the first one gives 3 oranges in exchange of 4 apples: After the exchange both are happier. Alternatively, if $x_2$ represents the amount of money instead of oranges then the first agent would be a customer while the second one might be a vendor. Utility functions allow a great expressivity in preferences. For instance, $f^u(\bar{x}) = x_1 \cdot x_2$ denotes that variety is preferred. A usual assumption is that no resource is a *bad*, that is, if the amount of a resource is increased, so does the value returned by the utility function. Using a derivative expression, this property can be formally expressed as $\frac{\Delta f^u(x_1,\ldots,x_n)}{\Delta x_i} \geq 0$ for all $x_1, \ldots, x_n \in \mathbf{R}$ and $1 \leq i \leq n$. This requirement does not constrain the expressive power of utility functions, as the existence of any undesirable resource can be always expressed by considering a resource representing the *absence* of it.

Next we introduce a collection of *identifiers* to be able to univocally identify cellules, agents and paths in the system. In the next section, we will formally define these concepts.

**Definition 2.** Let $w$ be a system (see Definition 8). The set of all possible systems is represented by $\mathcal{W}$. We denote by $\mathrm{ID}^{\mathrm{C}}$ the set of cellule identifiers that are assigned uniquely to each of the cellules. The function $\mathbf{newIdCellule} : \mathcal{W} \to \mathrm{ID}^{\mathrm{C}}$ returns an unused identifier for the world $w$. We use a special identifier $\texttt{nill} \in \mathrm{ID}^{\mathrm{C}}$ to denote an empty cellule. We denote by $\mathrm{ID}^{\mathrm{A}}$ the set of agent identifiers that are assigned uniquely to each of the agents belonging to the system. The function $\mathbf{newIdAgent} : \mathcal{W} \to \mathrm{ID}^{\mathrm{A}}$

returns an unused identifier for an agent. We denote by $\mathrm{ID}^{\mathrm{P}}$ the set of path identifiers, that are assigned uniquely to each of the paths. The function $\mathbf{newIdPath} : \mathcal{W} \rightarrow \mathrm{ID}^{\mathrm{P}}$ returns a fresh identifier for a path.

□

## 3 Definition of the Formalism

In this section we present our formal language to specify complete systems as well as all the agents taking part in them. The basic notion to define the behaviour of agents is a *transition*, that is, a transformation of resources carried out by a specific agent. *Atomic* and *complex* agents will both hold transitions as a way to accomplish tasks, but *only atomic agents* will actually perform the transformation of resources. A transformation of resources is represented by a tuple $\bar{z} \in \mathbf{R}^n$. Intuitively, a positive component of the tuple $x_i$ unit of the *i-th* resource while a negative component $x_j$ denotes that the transition consumes $x_j$ units of the *j-th* resource.

**Definition 3.** A *transition* of the system is represented by the tuple $(\bar{z}, id_p)$ where $\bar{z} \in \mathbf{R}^n$ is the transformation of resources and $id_p \in \mathrm{ID}^{\mathrm{P}}$ identifies the path that is in charge of executing the transition. The set of all transitions is denoted by $\mathtt{TR}$. □

A *path* is a sequence of transitions. It is conformed of transitions, in a specific order, through concatenation. Paths allow to specify the situation where a *complex* agent has to execute several consecutive tasks.

**Definition 4.** A *path* is a sequence of transitions. If $tr_1, \ldots, tr_m \in \mathtt{TR}$ then $p =< tr_1, \ldots, tr_m >$ represents the path conformed by them. We have that $p_i$ denotes the *i-th* element of the path, that is the transition $tr_i$. The set of all paths is denoted by $\mathcal{P}$. We can inductively define paths as follows:

– $<>\in \mathcal{P}$.
– If $tr \in \mathtt{TR}$ and $p \in \mathcal{P}$ then $tr \cdot p \in \mathcal{P}$.

Thus, paths are either empty or are conformed by adding an element to a path. □

Next we show how to represent *agents*. We can distinguish between *complex* and *atomic* agents. *Atomic* agents assume the responsibility of actually implementing tasks, and *complex* agents cluster and delegate in the ulterior ones to accomplish complex tasks and summarize the properties of the agents that are implicity inside of them.

**Definition 5.** An *agent* is a tuple $a = (id, ib, P)$ where $id \in \mathrm{ID}^{\mathrm{A}}$ is a unique identifier for this agent, $ib \subseteq \mathcal{M}$ is the input buffer where messages will be stored, and $P \subseteq \mathcal{P} \times \mathrm{ID}^{\mathrm{P}}$ is the set of paths defining the possible behaviours of this agent, being each path labeled with an identifier. Intuitively, the meaning of this set of paths is that this specific agent will achieve through any of this paths a similar global transformation of resources. In other words, every path takes him from the same initial state towards a similar final state, differing one from each other in the kind of transformations that they perform.

We denote by $\mathcal{A}$ the set of all agents. We define the function $\mathtt{VTr} : \mathrm{ID}^{\mathrm{P}} \rightarrow \mathcal{P}$ as follows. Let $P = \{(<tr_1^\alpha, \ldots, tr_m^\alpha>, \alpha), (<tr_1^\beta, \ldots, tr_{m'}^\beta>, \beta), \ldots\}$ be a set of paths. We define $\mathtt{VTr}(\alpha) = <tr_1^\alpha, \ldots, tr_m^\alpha>$. We also define the function $\mathtt{VA} : \mathrm{ID}^{\mathrm{P}} \rightarrow \mathrm{ID}^{\mathrm{A}}$ that returns the agent that performs this path. □

Let $a = (id, ib, P)$ be an agent and $P = \{(<tr_1^\alpha, \ldots, tr_m^\alpha>, \alpha), (<tr_1^\beta, \ldots, tr_{m'}^\beta>, \beta), \ldots\}$ be the set of paths of agent $a$. We define the function $\mathtt{VP} : \mathrm{ID}^{\mathrm{P}} \rightarrow \mathbf{R}^n$ using the auxiliary function $\mathtt{VPAux} : \mathcal{P} \times \mathrm{ID}^{\mathrm{A}} \rightarrow \mathbf{R}^n$ as $\mathtt{VP}(\alpha) = \mathtt{VPAux}(\mathtt{VTr}(\alpha), \mathtt{VA}(\alpha))$ being defined as:

$$\mathtt{VPAux}(<>, id) = \bar{0}$$

$$\mathtt{VPAux}(<tr_1, tr_2, \ldots, tr_n>, id) = \bar{z} + \begin{cases} \mathtt{VPAux}(<tr_2, \ldots, tr_n>, id) & \text{if } \; tr_1 = (\bar{z}, id_p) \wedge \\ & \quad\;\; id = \mathtt{VA}(id_p) \\ \mathtt{VPAux}(\mathtt{VTr}(id_p), \mathtt{VA}(id_p)) + & \text{if } \; tr_1 = (\bar{z}, id_p) \wedge \\ \mathtt{VPAux}(<tr_2, \ldots, tr_n>, id) & \quad\;\; id \neq \mathtt{VA}(id_p) \end{cases}$$

An agent is *atomic* if it has only one path, that path is conformed by a single transition, and the agent itself is in charge of executing the transition. Formally, $a = (id, ib, P)$ is an *atomic* agent if $\mid P \mid \; = 1$ and there exists $p = <tr_1, \ldots, tr_m> \in P$ such that for all $1 \leq i \leq m$ if $tr_i = (\bar{z}_i, id_p)$ then $\mathtt{VA}(id_p) = id$.

During the rest of the paper we consider that agents use *messages* to communicate among them. The next definition introduces the different kinds of messages that can be sent.

**Definition 6.** A *message* is given by a tuple $(t, s, ob, \bar{r})$ such that $t \in \{\mathtt{BROADCAST}, \mathtt{REPLIES}, \mathtt{START\ JOB}, \mathtt{FINISHED\ JOB}\}$, denotes the nature of the message, $s \in \mathrm{ID}^{\mathrm{P}} \cup \{\mathbf{null}\}$ the path origin of the message. In some cases this path can have the value **null**. The next item $ob \in \mathrm{ID}^{\mathrm{P}} \cup \{\star\}$ is the objective of the message, it can be a specific path of an agent, or a broadcast message. The last component, $\bar{r} \in \mathbf{R}^n$ represents a transformation of resources. In the rest of this paper, we denote by $\mathcal{M}$ the set of all messages. □

*Example 1.* Let $id \in \mathrm{ID}^{\mathrm{A}}$ be an agent identifier, $p_1, p_2 \in \mathrm{ID}^{\mathrm{P}}$ be paths identifier, and $\bar{r}$ be a vector of resources. A message $m = (\mathtt{BROADCAST}, \mathbf{null}, \star, \bar{r})$ represents a broadcast message ($\star$) sent by a petition wanting to find an agent that accomplish the transformation induced by $\bar{r}$. If we have a message $m = (\mathtt{REPLIES}, p_1, p_2, \bar{r})$; $m$ denotes the message from agent $\mathtt{VA}(p_1)$ that offers the path $p_1$, that replies to agent $\mathtt{VA}(p_2)$ to the petition of performing a certain task of the path $p_2$, and specifies the transformation of resources $\bar{r}$.If we have a message $m = (\mathtt{START\ JOB}, p_1, p_2, -)$, $m$ now represents the message from agent $\mathtt{VA}(p_1)$ which is performing the path $p_1$ for asking to start the job to the path $p_2$ of the agent $\mathtt{VA}(p_2)$. Finally, if $m = (\mathtt{FINISHED\ JOB}, p_1, p_2, -)$, then $m$ is the message from agent $\mathtt{VA}(p_1)$ to agent $\mathtt{VA}(p_2)$ to indicate that the path $p_1$, which is a sub-path of $p_2$, has just finished. □

*Cellules* are elements that serve as baskets of agents to reunite, organize, conglomerate and handle petitions as well as calls to the agents.

**Definition 7.** A *cellule* is a tuple $(\mathcal{A}, id, \mathrm{Sons}, \mathrm{Father}, ib)$ where

- $\mathcal{A} \subseteq \mathrm{ID}^{\mathrm{A}}$ is the set of agents that belong to the cellule.
- $id \in \mathrm{ID}^{\mathrm{C}}$ is a unique identifier for this cellule.
- $\mathrm{Sons} \subseteq \mathrm{ID}^{\mathrm{C}}$ is the set of identifiers of the sons of this cellule. If $\mathrm{Sons} = \varnothing$ then we are in a node cellule.
- $\mathrm{Father} \in \mathrm{ID}^{\mathrm{C}}$ is the identifier of the cellule that is father of this cellule. If Father=`nill` then we are in the initial cellule, from which all other cellules are defined.
- $ib \subseteq \mathcal{M}$ is the input buffer where messages will be stored.

We denote by $\mathcal{C}$ the set of all cellules. □

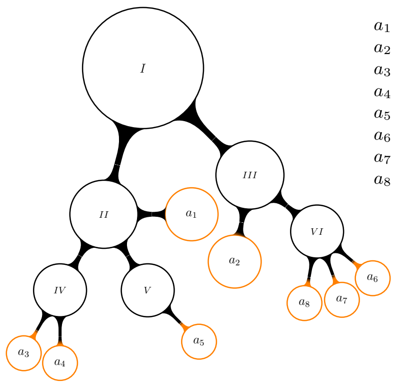
Next, we define the whole system that contains in a tree like structure implicity defined by the father-son relationship, the cellules that conform the whole system.

**Definition 8.** We say that our *system* (sometimes called *world*) is defined with a so called *origin cellule* from where the tree of cellules hang and by the vector of resources available in the system. Therefore, a *system* is a pair $w = (c, \bar{x})$ where $c \in \mathrm{ID}^{\mathrm{C}}$ is the origin cellule, and $\bar{x}$ is the set of resources with which we deal in this world $\bar{x} \in \mathbf{R}^n$.

We will use a running example to illustrate previously introduced concepts. In order to ease the presentation, we have simplified the real system that we have represented in our formalism. □

*Example 2.* Let us consider that we have the world represented in Figure 1. As we observe in the figure, we have six cellules, labeled from $I$ to $VI$ and eight agents distributed in them. For example, let us consider agent $a_3 = (id_3, ib_3, P_3)$. $P_3$ is the set of paths that this agent can perform, $ib_3$ represents the input buffer of this agent and $id_3$ is the identifier of this agent. The set of paths $P_3$ contains a unique pair (pair, path identifier) $P_3 = \{(<(\bar{z}_a, \kappa)>, \kappa)\}$. The path identifier is $\kappa$ the first element of the pair represents the chain of transitions that compose this path. In this case the path is formed by a unique transition. This transition, $<(\bar{z}_a, \kappa)>$ represents that it is performed by the path $\kappa$ of the agent $id_3 = \mathtt{VA}(\kappa)$ and the exchange of resources after performing this transition is noted by $\bar{z}$. This means that the resources of the world will change by applying $\bar{x} \leftarrow \bar{x} + \bar{z}_a$, in other words, it will generate a **formwork** unit, by wasting 50 units of **money**, 40 units of **wood**, and 20 **time** units.

For example let us suppose that agent $a_1 = (id_1, ib_1, P_1)$ has two different paths. $(<(\bar{z}_g, \kappa), (\bar{z}_g, \mu)>, \alpha)$ and $(<(\bar{z}_h, \kappa), (\bar{0}, \nu)>, \beta)$. Next we explain one of these paths. The path identified by $\alpha$, has two transitions in it. The first transition, denoted by $(\bar{z}_g, \kappa)$, represents that this agent has to call to the $\alpha$-*path* of agent $\mathtt{VA}(\kappa)$ to perform it, and the transformation of resources by applying this transition is $\bar{x} \leftarrow \bar{x} + \bar{z}_g$. Then, after performing this transition the resources of the world would change to $\bar{x} \leftarrow \bar{x} + \bar{z}_a + \bar{z}_g$. Let us remember that the agent $id_3 = \mathtt{VA}(\kappa)$ transformation function for the path $\kappa$ is $\bar{z}_a$. Let us note that the agent performing this transition earns money by calling another agent. □



$$\begin{aligned}
a_1 &= (\ id_1, ib_1, P_1\ )\\
a_2 &= (\ id_2, ib_2, P_2\ )\\
a_3 &= (\ id_3, ib_3, P_3\ )\\
a_4 &= (\ id_4, ib_4, P_4\ )\\
a_5 &= (\ id_5, ib_5, P_5\ )\\
a_6 &= (\ id_6, ib_6, P_6\ )\\
a_7 &= (\ id_7, ib_7, P_7\ )\\
a_8 &= (\ id_8, ib_8, P_8\ )
\end{aligned}$$

$$W = \{I, \bar{x}\}$$

$$P_1 = \left\{ \begin{array}{l} (<(\bar{z}_g, \kappa)\ , (\bar{z}_g, \mu)>\ , \alpha), \\ (<(\bar{z}_h, \kappa)\ , (\bar{0}, \nu)>\ \ , \beta) \end{array} \right\}$$

$$P_2 = \left\{ \begin{array}{l} (<(\bar{z}_g, \varpi)\ , (\bar{0}, \varsigma)>\ , \gamma), \\ (<(\bar{0}, \varpi))\ , (\bar{0}, \tau)>\ , \delta), \\ (<(\bar{0}, \upsilon)\ \ \ , (\bar{0}, \varsigma)>\ , \epsilon), \\ (<(\bar{z}_h, \upsilon)\ \ , (\bar{0}, \tau)>\ , \iota) \end{array} \right\}$$

$$P_3 = \{(<(\bar{z}_a, \kappa)>, \kappa)\}$$
$$P_4 = \{(<(\bar{z}_b, \mu)>, \mu)\}$$
$$P_5 = \{(<(\bar{z}_c, \nu)>, \nu)\}$$
$$P_6 = \{(<(\bar{z}_d, \varpi)>, \varpi)\}$$
$$P_7 = \left\{ \begin{array}{l} (<(\bar{z}_{e1}, \varsigma)>,\ \varsigma), \\ (<(\bar{z}_{e2}, \tau)>,\ \tau) \end{array} \right\}$$
$$P_8 = \{(<(\bar{z}_f, (id_8, \upsilon))>, \upsilon)\}$$

$$\begin{array}{ll}
\bar{z}_a & = [\ -50, \quad 0 \quad 0 \quad -40, 0, \quad 0, \quad 0, \quad 1, \quad -20\ ] \\
\bar{z}_b & = [\ -100, -300, \quad 0, \quad 0, \quad 1, \quad 0, \quad 0, \ -1, \ , -30\ ] \\
\bar{z}_c & = [\ -80, \ -450, \quad 0, \quad 0, \quad 1, \quad 0, \quad 0, \ -1, \ -40\ ] \\
\bar{z}_d & = [\ -200, \quad 0, \ -300, \quad 0, \quad 1, \quad 0, \quad 0, \quad 0, \ -20\ ] \\
\bar{z}_{e1} & = [\ -50, \ -10, \quad 0, \quad 0, \quad 0, -20, \ , 1, \quad 0, \ -30\ ] \\
\bar{z}_{e2} & = [\ -55, \ -9, \quad 0, \quad 0, \quad 0, -23, \ 1, \quad 0, \ -15\ ] \\
\bar{z}_f & = [\ -250, \quad 0, \ -250, \quad 0, \quad 1, \quad 0, \quad 0, \quad 0, \ -30, \ ] \\
\bar{z}_g & = [\ -25, \quad 0, \quad 0, \quad 0, \quad 0, \quad 0, \quad 0, \quad 0, \ -20\ ] \\
\bar{z}_h & = [\ -30, \quad 0, \quad 0, \quad 0, \quad 0, \quad 0, \quad 0, \quad 0, \ -10\ ]
\end{array}$$

| Position in the resource tuple of the world | Semantic |
|---|---|
| 1 | **money** |
| 2 | **concrete** |
| 3 | **steel** |
| 4 | **wood** |
| 5 | **structure** |
| 6 | **bricks** |
| 7 | **wall** |
| 8 | **formwork** |
| 9 | **time** |

**Fig. 1.** Representation of a world.

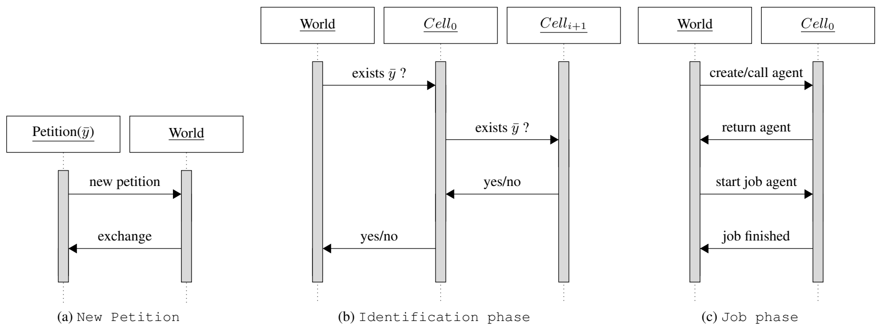


**Fig. 2.** Schematic diagrams of world behaviour.

All agents that are not *atomic* are *complex*, there are two ways to create agents one is to insert an atomic agent during the creation of the system and the other is through *petitions* to the system, being the system in charge of recombining atomic and/or complex agents already embedded in the system to create a new complex agent.

**Definition 9.** We say that a *petition* is a tuple $pet = (f^u, \bar{y}, \bar{o})$ where $f^u \in \mathcal{F}$ is a utility function, $\bar{y} \in \mathbf{R}^n$ is the vector of resources that is added to the resources already existing in the world, and $\bar{o} \in \mathbf{R}^n$ is the objective of the transitions, that is, the vector of resources that we expect to have after performing the petition. □

Intuitively, if we have a petition $pet = (f^u, \bar{y}, \bar{o})$ is a petition, and $a = (id, ib, P)$ is the agent that has created the petition, if $\exists\ p \in \texttt{TR}$ such that exists $(p, id_p) \in P$ : $\texttt{VP}(id_p) + \bar{x} + \bar{y} \geq \bar{o}$.

*Example 3.* We will explain the main messages by applying a petition (see a graphical representation in Figure 2). Let us consider a petition $pet = (f^u, \bar{y}, \bar{o})$. The first component of $pet$ contains the initial resources, $\bar{y} = [1000, 500, 500, 100, 0, 100, 0, 0, 500]$, the second one is a utility function (in this case $f^u = 10 \cdot x_1 + 5 \cdot x_9$), and the third element is the objective tuple of resources $\bar{o} = [0, 0, 0, 0, 1, 0, 1, 0, 0]$.

The first diagram of Figure 2 denotes that $pet = (f^u, \bar{y}, \bar{o})$ is inserted in the world $w = (I, \bar{x})$. When a new petition is inserted in the world, the resources of the petition are added to the existing vector of resources. After this initial stage, the world "asks" to its structure of cellules if there are any agent(s) which can achieve the objective function $\bar{o}$. □

## 4 Implementation

In this section we present our tool that facilitates the task of representing the different components of our framework. First, we are going to enumerate some of the technical requirements of the tool. Next, we will comment on some relevant parts of the implementation, and we will show how the example can be represented.

The tool has been developed using J2EE Technology (Java, JDK 1.5, EJB) and the Netbeans software. It makes usage of MVC architecture, to enable ease of maintenance, and uses session facade and proxy design patterns. It also uses Java Swing components in order to develop Graphical User Interfaces(GUI).

The tool offers four different ways to create systems. The first one is by using an input XML-formatted file which contains all the description data of the system. Another way to introduce a model is by using Java Database Connectivity (JDBC). JDBC is an API for the Java programming language that defines how a client may access a database. It provides methods for querying and updating data in a database. The third way to input a system is by using the editor included in the GUI. The user may customize the editor and make sure the Editor Presentations command group is checked under the Commands tab. The last way to create systems is by loading serializable models saved previously. Serializability of a class is enabled by the class implementing the java.io.Serializable interface. By using an MVC design, the system can easily make it. When a model is in the tool, it can be also saved in the same ways, by using an XML formatted file, in a database, and by serialization of the model.

For the representation of the world, cellules, and agents we have used *threads*. A java.lang. Thread object maintains bookkeeping and control for this activity. In fact, by representing each of the components by using threads we let the system represent a more realistic world. For example agent $a_1$ can be waiting until agent $a_3$ has finished its task, while the world continues receiving petitions, and the cellules continue sending messages between the agents. The tool also introduces priority among the threads. Each Thread has a priority, ranging between 1 and 10. Priorities have no other impact on semantics or correctness. In particular, priority manipulations cannot be used as a substitute for locking. Priorities can be used only to express the relative importance or urgency of different threads, being these priority indications useful to take into account when there is heavy competition among threads waiting to be executed. For example the initial cellule normally has bigger priority than other cellules; the reason is that management of petitions and the data traffic flow is mainly done through this cellule.

Another important task in a concurrency scheme is the management of shared memory, being the buffers implemented as circular buffers using a single, fix size. Circular buffers are also used for data transfer between processes. The tool uses monitors in theses buffers to synchronize accessing threads. Conceptually, a monitor is a class whose data members are private and whose member functions are implicitly executed with mutual exclusion. In addition, monitors may define waiting conditions that can be used inside the monitor to synchronize the members functions.

## 5 Conclusions and Future Work

The work presented in this paper provides a useful framework for the developing of complex computational systems. With the presentation of the tool we offer the possibility of use to specialists from different fields, not being constrained its applicability to the field of computing technology and therefore being useful as a computational science tool.

The work will be extended to allow testing and checking of conformance of the system implemented with it. Other future line of work will be to develop the structure of the cellules in an AVL tree-like structure, so that the message flow will be re-equilibrated. Another possible continuation of the work will be to specify and implement the subdivision of cellules when the number of agents that they withholds surpasses a limit, this allows not to surcharge a specific computational resource in which the cellule may be implemented, like a specific processor, and derive its work flow to another resource. Finally, we are currently working on a complex application of our methodology to the construction world.

## References

1. Andrés, C., Molinero, C., Núñez, M.: A formal methodology to specify hierarchical agent-based systems. In: 4th Int. Conf. on Signal-Image Technology & Internet-based Systems, SITIS'08, IEEE Computer Society Press (2008) 169–176
2. Lomazova, I.: Communities of interacting automata for modelling distributed systems with dynamic structure. Fundamenta Informaticae **60**(1-4) (2004) 225–235
3. Lomazova, I.: Nested Petri Nets for adaptive process modeling. In: Pillars of Computer Science, Essays Dedicated to Boris Trakhtenbrot on the Occasion of His 85th Birthday, LNCS 4800, Springer (2008) 460–474
4. Núñez, M., Rodríguez, I.: PAMR: A process algebra for the management of resources in concurrent systems. In: 21st IFIP WG 6.1 Int. Conf. on Formal Techniques for Networked and Distributed Systems, FORTE'01, Kluwer Academic Publishers (2001) 169–185
5. Núñez, M., Rodríguez, I., Rubio, F.: Formal specification of multi-agent e-barter systems. Science of Computer Programming **57**(2) (2005) 187–216
6. Núñez, M., Rodríguez, I., Rubio, F.: Specification and testing of autonomous agents in e-commerce systems. Software Testing, Verification and Reliability **15**(4) (2005) 211–233
7. Merayo, M., Núñez, M., Rodríguez, I.: Formal specification of multi-agent systems by using EUSMs. In: 2nd IPM Int. Symposium on Fundamentals of Software Engineering, FSEN'07, LNCS 4767, Springer (2007) 318–333
8. Dastani, M., Jacobs, N., Jonker, C., Treur, J.: Modelling user preferences and mediating agents in electronic commerce. In: Agent Mediated Electronic Commerce, The European AgentLink Perspective, LNCS 1991, Springer (2001) 163–193
9. Geisler, B., Ha, V., Haddawy, P.: Modeling user preferences via theory refinement. In: 5th Int. Conf. on Intelligent User Interfaces, IUI'01, ACM Press (2001) 87–90
10. Ha, V., Haddawy, P.: Similarity of personal preferences: Theoretical foundations and empirical analysis. Artificial Intelligence **146**(2) (2003) 149–173
11. Mas-Colell, A., Whinston, M., Green, J.: Microeconomic Theory. Oxford University Press (1995)