Planning of Work Schedules through the Use of a Hierarchical Multi-Agent System

Carlos Molinero and Manuel Núñez
Dept. Sistemas Informáticos y Computación
Facultad de Informática
Universidad Complutense de Madrid, 28040 Madrid, Spain
e-mail: molinero@fdi.ucm.es, mn@sip.ucm.es

Planning of Work Schedules through the Use of a Hierarchical Multi-Agent System

Carlos Molinero and Manuel Núñez
Dept. Sistemas Informáticos y Computación
Facultad de Informática
Universidad Complutense de Madrid, 28040 Madrid, Spain
e-mail: molinero@fdi.ucm.es, mn@sip.ucm.es

Abstract

We propose a methodology to simulate every small task of a site-work with a multi-agent system. These agents handle resources as a way to perform transformations on their world. The system will simulate the construction of a building through the definition of the atomic elements of the system and the automatic recombination of these elements. This allows us to foresee parallel and sequential tasks and handle the creation of a graph, in the form of a Petri net, that facilitates the task of accurately planning the schedule of the site-work.

Keywords: MAS, agents, scheduling, Critical Path Graph

1. Introduction

Construction success highly depends on the capacity of the project manager to handle multiple teams with a wide variety of tasks and with different needs. These teams must be correctly orchestrated during the realization of their labor since the total cost of the construction depends on the total amount of time the building takes to be constructed. Therefore, parallelization of tasks is highly desirable, although this is not always achievable. Different tasks cannot be executed before a certain amount of resources have already been created or before the team in charge of executing the task is free to start its implementation.

It is therefore mandatory to create a timed organizational structure of the planned work. The instruments more commonly used to handle this need are *Gantt diagrams* and *Critical Path graphs*. Although good planning is usually achieved through the experience gathered by the project manager, a number of elements are always left out. Imperfections are therefore left in the schedule of the work site and these sometimes lead to undesired time delays. We present in this paper a computational system that, through the use of agents, simulates the resources created or exchanged by every agent in the construction site and that self-organizes these agents to define a possible sequential frame in which every task is to be executed. More specifically, our goal is to provide a system that automatically constructs a graph, in the form of a Petri net [1], representing the sequence of jobs to be undertaken during a construction process. Actually, one of the main advantages of Petri nets is that there exists several formal methods for analyzing them.

Our methodology uses a system of agents that recreate the construction process in a simulated manner. Because of the modularity of this approach, once an agent is designed, it can take part in any new project that handles similar needs. Therefore, the system is scalable and further increments in complexity of the project do not need to increase computation times. The methodology also allows parallelization of the tasks, which helps to model up to a high level of detail any construction process. Furthermore, since every agent represents a real world agent of the construction process, it is easy to translate the characteristics of the system to the construction site.

We use two types of agents: *atomic* and *complex*. *Atomic* agents are a metaphor of work force. These agents are in charge of the actual transformation of resources. A complex agent is similar to the leader of a team. These agents are used to gather atomic agents (and/or other complex agents) to reunite and conglomerate their individual properties, and arrange the order in which they will start working.

Agents simulate the procedure of the construction of a building by exchanging resources. This is done through destroying and creating new types of resources. In that sense, when using our methodology one has to think a little bit different from the real world. For example, an agent painting a wall, *destroys* a non-painted wall and *creates* a painted one.

Agents can be reincorporated from other projects that we have dealt with before, that is, we do not need to reconstruct every agent each time that we want to plan a new construction site. What we need to do is either to incorporate new complex agents through the use of petitions or to define the new needs of our project in terms of resources. The system will reconfigure itself to be able to handle the demands.

Agents are introduced into a *cell* structure. This structure maintains close those agents that are related in terms of the kind of transformations that they perform. This is useful in computational means since messages between agents flow more directly. This structure has the form of a tree, each cell acting as a parent of its children's cells. Agents are naturally arranged in terms of complexity in the sense that simpler agents are inserted in the lower level cells and when complex agents are formed they are inserted in a higher level. A complex agent will be inserted in a cell that is father of each of the cells that hold the agents that it is going to use. For example, if we have a complex agent that needs to call *agentA* and *agentB* to fulfill its task, and *agentA* belongs to *cell*1, *agentB* belongs to *cell*2, then the complex agents has to be inserted in a cell that is a common father of *cell*1 and *cell*2. If such father does not exist, then a new cell is created in the tree and the complex agent is inserted in it. This assures that messages from the complex agent to other ones can just be sent downwards into the tree structure. This process is automated, needing no interaction from part of the user. The interface of the user with the system is based on the use of petitions that define the needs and the restrictions that the user desires.

The simulation of the user preferences is achieved through the implementation of a *utility function*. These functions assign a value to each basket of resources. This allows us to measure the quality of a certain combination of resources. For more information on utility functions, their properties, and their relation with other mechanisms used in microeconomics, the interested reader is referred to the paper [2].

The rest of the paper is structured as follows. In Section 2 we will briefly summarize some of the recent developments made in the multi-agent research field and its application to project management. In Section 3 we will focus on some efforts to apply computer-aided project management. In Section 4 we will describe our methodology. In Section 5 we will propose a general construction site as an example of our methodology. In Section 6 we will summarize and present our conclusions. In the appendix of the paper we will present the mathematical methodology behind our system, as well as discuss other details of the approach. This appendix is meant for completeness since users of our methodology do not need to know the formal framework underlying it. Finally, let us remark that we have developed a tool that implements our approach and that can be found at $http : \backslash\backslash www.carlosmolinero.com\backslash hierarchicalAgents.zip$.

2. Multi-agents systems

Multi-agent systems (in short, MAS) is a field of growing interest in research [3, 4]. The number of applications to which agents can be applied is endless due to its distributed and flexible computational power. These characteristics derive from the nature of a MAS: it is composed of several agents that communicate and work together in an environment (application field or problem). Each of these agents is designed to give an answer to a specific problem and, depending on the kind of society formed by them, they will come to act in a specific moment of time. The power of these systems comes, therefore, from the combination of separate and multiple viewpoints to confront a problem.

Another of its important features is that a MAS can be a simple translation of real world organizations and, therefore, conform a model (an abstraction) of that system. Similarly, it is always beneficial to have a computational tool that resembles the real organization. Some of the work in this line consists in the assignation of roles to agents by using organizational rules [5].

Other approaches use agents in a completely different manner: agents are separate entities reacting to real world responses and without a fixed role [6]. The role implicitly appears by the kind of conditions that activate the agent to act upon its behavior. This kind of systems belongs to a category called *subsumption architectures*, they work in a layered fashion, constructing in a bottom up manner a complex behavior made from the interaction of several agents that react to different stimuli coming from the outside world. The subsumption model has been used to define a way to create the corresponding chain of events that trigger a certain agent [7].

In the work [8] we can find an overview of the current approaches taken in the field of MAS in respect to construction automation. This paper provides, a general explanation of what MAS are, the problems that are yet to be solved, and the insights of the most relevant work in this field are explained.

3. Approaches to Computer Assisted Project Management

Next we review some of the work on automation of project management tasks.

The task based modeling method (TBM) for modeling processes in project management [9]. The underlying idea is to build a process model, by using the task components as the basic building blocks, in order to improve project efficiency and productivity. Since it is difficult to adapt and customize the standard process, the authors propose a new way to define common management tasks as the basic reusable software components. For its achievement, each task must have its function and corresponding data structure correctly defined. Moreover, instead of trying to standardize a business process, TBM intends to focus on standardizing lower level management tasks, having each the characteristics of independency, encapsulation, completeness and consistency. The basic structure for a task in this methodology has three features: an action (a verb that characterizes the task), a method (a manner to perform the task) and an object (needed in the case of transitive verbs as a direct object).

The ABSM (agent-based modeling and simulation) technique has been applied to construction in isolation or in combination with traditional simulation methodologies [10]. Until now, most approaches for the study of complex systems (like CPM, PERT, productivity delay models, etc) use a discrete event approach, while ABSM can add to those techniques continuous simulation to further understand the underlying complex system. Moreover, the use of agents can explain the construction discipline better than a *central* control approach [11]. The agent-based modeling and simulation testbed can be used to mimic the construction environment. The authors propose two main domains in which this technique will be of a straightforward use: the application to the study of the safety environment of the workers and the reduction of the time applied to the cycle of eradication of the waste. The paper concludes by stating that ABMS combined with the traditional discrete event approach provides added flexibility in modeling of complex construction systems.

MAS can be used to improve the traffic flows in a construction site [12]. Since one of the main characteristics of traffic flow in a construction site is its dynamic behavior, the use of a system of agents is the most suitable. The authors use the MAS as a simulation system to predict possible problems and develop a cost-effective and quick construction process.

Petri nets have been previously used to specify the processes of construction [13]. The approach provides an accurate, precise and flexible mechanism to specify the work-flow processes in construction and, by doing so, to determine the weak points of the procedure and correct them. The authors use a colored, timed and hierarchical Petri net. The proposed methodology will take place in several stages: draw the Petri net for the actual state of the process, simulate the process to study where the problems reside, and redesign the Petri net, checking its correctness.

The case handling paradigm can be used to support the construction process [14]. This paradigm establishes several concepts. For each case there exist activities. An activity is the logical unit of work and corresponds to atomic pieces of work. Each activity is executed by one worker, while a worker can have many roles, an activity has only one; the worker that is capable of executing that activity is a worker with that role. Case handling is driven both by control flow and data flow and each case has a collection of data objects which can be classified into mandatory or restricted. For each process and each activity, three roles need to be specified: the execute role, the redo role and the skip role (all of them can be set to no-one or everybody).

The paper [15] proposes a conceptual model that is composed of 4 sub-models: the task model, the context model, an organization model and a case model plus its interrelationships. The task model includes 4 major concepts: task, group task, simple task and compound task. The context model is a ternary relation of decision, condition and task. The organization model contains role, resource, group, individual, project team, organization and project. Finally, the case model handles concepts like case, supporting document, classification and existing project information.

Petri nets can be used to produce construction schedules [16]. A work task is given by the combination of an input place, a transition and an output place. In this work, the authors also consider important to provide the construction schedule with the ability to represent a hierarchical breakdown of the tasks. This is achieved using hierarchical Petri nets, the incorporation of uncertainty in the activities, represented through probabilistic transitions, and the modeling of dynamic resource allocation, which is realized by using colored Petri nets where each color represents a different kind of resource, and a extension of Petri nets, called fusion places, that allows to consider several places appearing in the network as a single place with the same type and number of tokens. These concepts are applied to the modeling of a concrete production plant [17].

Genetic algorithms (GA) [18] have been also used for the assignment of tasks to the available resources [19]. The technique of genetic algorithms is based on the laws proposed by Darwin for the evolution of the species. They use mutation, crossover and selection of the fittest, as the means to achieve an optimal solution to a specific problem. In this approach, the DNA of each inhabitant uses a 2D array, with one dimension for tasks and another one for resources, to codify this specific problem. The project management strategy is applied to the production of software, using an estimation of the cost of producing the software based on the COCOMO model [20]. The user of the system needs to specify a task precedence graph, a database of employees, including skills and salary, and an objective function.

Genetic algorithms can be applied to project management using a two phase model in which resources are limited [21]. Therefore, they impose a boundary on the order of execution of tasks. This model considers time-cost trade-off and dependence relationships between the different tasks. Tasks are given an execution mode which determines the set of resources, duration of the task and its possibility of being parallelized. The approach considers four subsystems: Input subsystem, time-cost trade-off subsystem, resource scheduling subsystem and the output subsystem. This division facilitates that the precedence relationships, resource constraints and interruption/overlap conditions are met. A similar technique [22] focused on projects that have repetitive aspects and, therefore, use the same resources in different periods of time. The approach considers project duration and project costs, plus constrains of precedence relationships between activities and resource work continuity.

Ant colony optimization (ACO) [23] can be also used to find a solution to the optimization of the schedule [24]. ACO is an evolutionary technique in which a set of artificial ants (the colony) are located into a graph and they move in the beginning randomly leaving a trace of pheromones proportional to the length of the path that have taken them from the nest to the goal. In the next turns, ants are drawn by these existing pheromone traces and eventually the colony finds the minimum length of the path from the nest to the goal. The work [25] shows through experimental research that the technique based on representing the CPM as an *activity on node* graph with the inclusion of dummy activities (activities with zero time/resources consumption, used to mark precedence relationships) attains a solution in a 100% of the cases and in less time than other methods based in *activity on arc* graphs [26]. Ants are also used in other problems relative to construction management such as the use of a max-min ant system to solve the construction site layout planning problem [27].

Also the swarm particle system optimization [28] (PSO) method has been used to create critical path graphs for the execution of a site-work [29]. PSO is a technique that uses the movement of a set of particles imitating a flock of birds with local and global velocities. Each particle adapts itself to the positions and velocities of their local neighbors and to the global velocity of the whole flock. Every particle tries to attain an improvement of their position in the fitness landscape. Thus, by adapting to its neighbors every particle eventually arrives to a maximum. A comparison between the PSO approach and

SOMO (self organizing map based optimization) points out that even though both techniques provide promising results, SOMO outperformed PSO in the cases studied [30].

Other approaches focus on the virtual construction of the site work. The work [31] proposes a system based on MD CAD (multi-dimensional CAD) to be implemented in the AutoCAD program, followed by a optimization of the work schedule based on a GA, while the work [32] uses a method based on virtual prototyping of the site-work. This process simulates resources, such as space, equipment and crew to foresee the implications of the construction environment.

MAS has been also used to formalize a negotiation methodology for the problem of distributed project schedule optimization [33]. The motivation of this work is based on the necessity to coordinate all the desires of the different organizations involved in the construction site. Coordination of projects has increased in complexity due to the appearance of an ever increasing number of sub-constructors. This technique represents each of the agents involved in the construction site, that is, each sub-constructor, with its own utility functions. Each agent in the MAS is given negotiation capabilities, in a process of synchronous negotiation, coupled with another negotiation mechanism called *recursive negotiation* used in the negotiation between two specific agents.

A MAS has been also used to mimic all the participants in a management system and the distributed localization of the resources [34]. The approach uses several kinds of agents. Among these agents we can find *macro-agents*, that are intended to be a functional grouping of one or more computers, interconnected through a local area network and devoted to local resources managing. Agents that are englobed into these macro-agents category can be of several types: facilitators, chiefs, supervisors, personnel, machinery, stock manager, supporting materials manager, and acquisition manager. The approach also have a planning agent and negotiation protocols that consider depreciation costs, operations costs, operator costs, and profits. Finally, it is possible to form coalitions and to consider modifications for the negotiation process.

There already exist some systems that handle MAS applied to creating a schedule for a construction process. One of these systems is the Stroboscope Simulation Environment, which can be found at *http://www.cem.umich.edu/Ioannou/StrobWeb/*. It is very complete, allowing the user to propose stochastic actions. This environment uses a system of pre and post conditions to concatenate the agents. The only drawback is that the user needs to define every small aspect of the process to be handled by the agents and its main connections, which is a problem that we intend to tackle in our approach.

4. Definition of our methodology

In this section we present our approach to specify complete systems as well as all the agents taking part in them. We claim that our framework is useful even if the user does not know the underlying formalism. Therefore, we will omit the explanation of the formal elements that compose the language which will be presented in the appendix of this paper.

We start by showing how to represent our *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. Complex agents represent a planning of the works accomplished by the atomic agents.

Agents have unique identifiers assigned. These identifiers can be seen as a word that denotes the concept that the agents represent. Agents also contain a buffer for the reception of messages from other agents and a Petri net that marks its execution procedure.

Informally, a *Petri net* is a state-diagram consisting of places, transitions, tokens and directed arcs. Places are connected by arcs to transitions, and transitions are connected to places by arcs. Each Petri net contains a marking, which is the position of the tokens in the places at a certain moment. The Petri nets used along this paper are by construction *1-safe*, which means that at most one token will be in a place at a certain moment. Depending on the topology of the state-diagram, several transitions may be fired simultaneously. This happens, for example, when a place is connected to multiple transitions. The formalism also allows to represent process synchronization by connecting several places to one transition. This transition will only fire when a token from every place is received. The firing of a transition removes a token from its input places and puts one in the output places. Our framework deals with time in the Petri net, although its conception is slightly different to what is usually considered in timed Petri nets. Usually, a timed Petri net has intervals of time in the transitions, to represent when this transition can be fired and tokens have an age, that is augmented by a clock, and reset each time the token goes through a transition. Our approach does not impose an interval of time for the transition to fire, but it adds age to the tokens, this age is not set to 0 at the firing of a transition, and time evolves not by a global clock but by the traversal of the arcs. It is important to remark that the user of our methodology does not need to know how to formally define a Petri net since it is a graphical formalism and our tool allows the user to represent them (the interested reader is referred to [1] for further details).

The configuration of the Petri net varies depending on whether we are dealing with an atomic agent or a complex one. In the first case, the Petri net will just consist of two places and a transition. The arc from the first place to the transition will contain an identifier of the agent in charge of executing the transformation of resources. In the case of an atomic agent, this identifier will coincide with the identifier of the agent forcing the agent to act upon the environment creating its transformation of resources. In the case of a complex agent, the Petri net can have as many places, transitions and arcs as necessary to define all the calls that will be done asking other agents to start working. This Petri net is able to define parallel or sequential calling of agents, depending on the kind of resources involved in the petition.

Example 1. *As a means to understand the difference between atomic and complex agents we propose a simple example depicted in Figure 1. There exists an atomic agent, called agent0,*

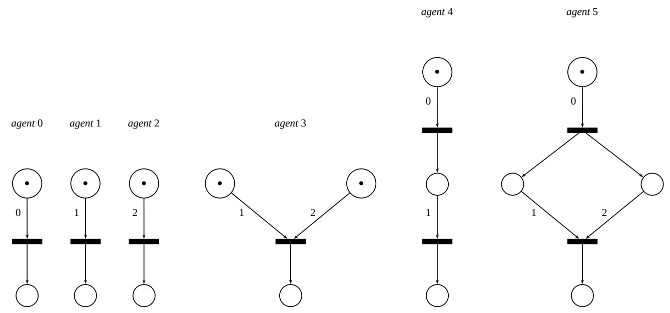


Figure 1: Atomic agents 0, 1 and 2 (left), a complex agent that executes agents 1 and 2 in parallel (center left), a complex agent that executes 0 and 1 sequentially (center right) and a complex agent that executes agent 0 followed by executing in parallel agents 1 and 2 (right).

that creates 2 units of the resource of type A and agents agent1, agent2 consume 1 unit of the resource of type A and create 1 unit of the resource of type B. Then, if we consider that the system has 2 units of the resource of type A, and we ask for 2 units of the resource of type B, the system will create complex agent3 placing in parallel calls to agents 1 and 2. Another possibility is that the system has no resources and we ask for 1 unit of type B, then the system will use agent0 and agent1 (or agent2) placing them sequentially, creating complex agent4. The last possibility is that the system has no resources and asks for 2 units of the resource of type B. Then, it will create complex agent5 and use agent0 followed by parallel calls to agent1 and agent2.

Cells serve as baskets of agents to reunite, organize, conglomerate and handle petitions as well as calls to the agents. Abstractly, a cell is the macro-concept that holds the set of related instances (agents). Cells are organized following a tree structure.

A cell holds a set of agents that are inserted inside it. They also have an identifier and pointers to the cells that are under it as sons, and to its father in the tree structure. Cells also have a buffer to hold the messages that they receive.

Example 2. *Continuing with the previous example, and to better illustrate the cell concept, we will insert two cells in our system. In the first cell we insert agent0 and in the second cell we insert agents 1 and 2. This is shown in the left part of Figure 2.*

Next, we introduce a petition in which we demand the system to provide us with 2 units of resource of type B. The system constructs a complex agent that first uses agent0 to get one resource of A, and then uses agent1 and agent2, in parallel, that consume that resource and provide one resource of type B. Since the atomic agents that the complex agent will use are located in different cells, in order to maintain a coherent cell structure for the sending of the messages, the system introduces a new cell that will be a common parent to both cells and locates the complex agent in that cell. This is shown in the right part of Figure 2.

The cell structure is principal to the passing of the messages. By construction, it allows a cell to send *STARTJOBs* messages

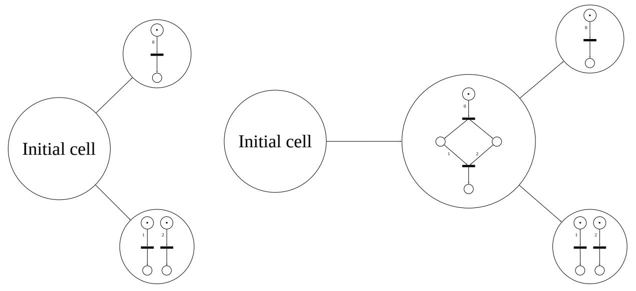


Figure 2: Cell structure in which two cells where atomic agents are inserted (left) and a petition is made that creates a complex agent composed of the three atomic agents (right).

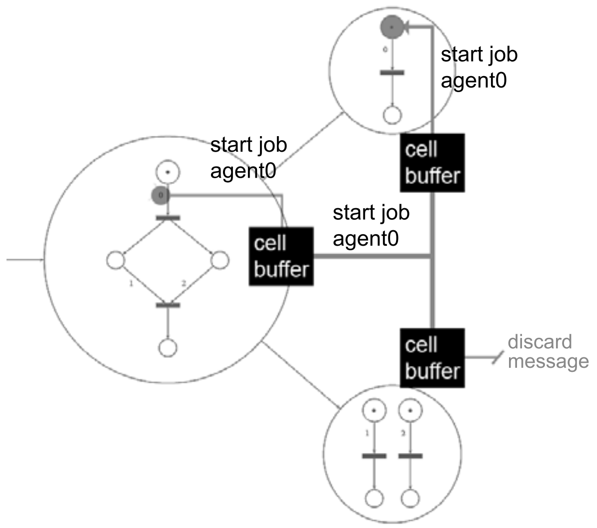


Figure 3: Passing one *STARTJOB* message.

downwards to its sons until it finds the specified agent. Since the cell structure was automatically constructed by the system having in consideration that a complex agent is inserted in a cell that is a common parent to all the cells that hold agents inserted into the complex agent, these messages will follow the shortest path possible. An example of the passing of the messages is shown in Figure 3. The system can send two families of messages. The first one is used during the creation of a new complex agent through a petition. These messages are either a *BROADCAST*, used to ask for the creation of resources to every agent in the system or a *REPLIES*, sent back by the agents that are capable of partially fulfilling the petition. The second family is used during the execution of a complex agent and they can be of type *STARTJOB*, used to ask an agent to start its procedure, and *FINISHEDJOB*, sent back by the agent when its tasks have been accomplished.

The system contains in a tree-like structure, implicity defined by the father-son relationship, the cells that conform the whole system. This structure allows the user to have a hierarchical organization of concepts. In order to completely define a *system* we need to define the resources that it holds, the origin cell and a threshold value that will be used to discriminate between

CELL																					
c_1	$\bar{s}_1$	= [	10,	−100,	0,	0,	0,	0,	1,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0	]
c_2	$\bar{s}_2$	= [	8,	−50,	−100,	0,	0,	0,	−1,	1,	1,	0,	0,	0,	0,	0,	0,	0,	0,	0	]
c_3	$\bar{s}_3$	= [	2,	−10,	0,	0,	−100,	0,	0,	0,	−0.5,	1,	0,	0,	0,	0,	0,	0,	0,	0	]
c_3	$\bar{s}_4$	= [	2,	−10,	0,	0,	−100,	0,	0,	0,	−0.5,	1,	0,	0,	0,	0,	0,	0,	0,	0	]
c_4	$\bar{s}_5$	= [	30,	−120,	−100,	0,	0,	0,	0,	0,	0,	−2,	1,	1,	0,	0,	0,	0,	0,	0	]
c_5	$\bar{s}_6$	= [	10,	−60,	0,	0,	0,	−100,	0,	0,	0,	0,	0,	−0.333,	1,	0,	0,	0,	0,	0	]
c_5	$\bar{s}_7$	= [	10,	−60,	0,	0,	0,	−100,	0,	0,	0,	0,	0,	−0.333,	1,	0,	0,	0,	0,	0	]
c_5	$\bar{s}_8$	= [	10,	−60,	0,	0,	0,	−100,	0,	0,	0,	0,	0,	−0.333,	1,	0,	0,	0,	0,	0	]
c_5	$\bar{s}_9$	= [	7,	−60,	0,	0,	0,	−100,	0,	0,	0,	0,	0,	0,	0,	0,	−0.333,	0.334,	0,	0	]
c_5	$\bar{s}_{10}$	= [	7,	−60,	0,	0,	0,	−100,	0,	0,	0,	0,	0,	0,	0,	0,	−0.333,	0.334,	0,	0	]
c_5	$\bar{s}_{11}$	= [	7,	−60,	0,	0,	0,	−100,	0,	0,	0,	0,	0,	0,	0,	0,	−0.333,	0.334,	0,	0	]
c_6	$\bar{s}_{12}$	= [	4,	−70,	0,	−100,	0,	0,	0,	0,	0,	0,	0,	0,	−0.333,	0.5,	0.5,	0,	0,	0	]
c_6	$\bar{s}_{13}$	= [	4,	−70,	0,	−100,	0,	0,	0,	0,	0,	0,	0,	0,	−0.333,	0.5,	0.5,	0,	0,	0	]
c_7	$\bar{s}_{14}$	= [	2,	−30,	0,	0,	−20,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	−0.333,	0.334,	0.334	]
c_7	$\bar{s}_{15}$	= [	2,	−30,	0,	0,	−20,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	−0.333,	0.334,	0.334	]
c_7	$\bar{s}_{16}$	= [	2,	−30,	0,	0,	−20,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	−0.333,	0.334,	0.334	]

Table 1: Values of the agents.

good and bad values of the utility functions defined through the user petition. Resources are represented by a tuple. Any negative number in that tuple indicates that the system needs to be provided with that specific resource.

There are two ways to create agents. The first one is to insert an atomic agent during the creation of the system. The second one 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. In order to define a petition we need to specify what resources we need to obtain and what is the utility function to be considered. We can understand petitions as the means that the user has to interact with the agent.

4.1. *Steps of a petition*

Next we present how petitions are handled in our approach.

1. A user inserts a petition in the system asking for the combination of resources objective of the petition. In addition the user defines a utility function that encodes his preferences.
2. The system creates a new agent with a temporal configuration, including a new Petri net.
3. In the next step the system discriminates depending on the number of negative resources (resources to be fulfilled). In the case that more than one resource is negative, it subdivides the petition into different sub-petitions. This refinement is performed to allow parallelization of tasks.
4. The system sends messages between the agents through the structure of cells, asking for agents that are able to cover the negative resources of the petition, that is, they are able to create the resources that are currently negative.
5. The agent receives all the responses from the other agents and configures its Petri net for the calling of other agents upon its execution. If for the same call, there exists more than one agent capable of fulfilling the call, parallelization of these agents is performed as long as the utility function does not eliminate one of them using the threshold defined in the world.
6. The transformation functions of all the agents involved are added and if any resource is negative another search in the system is performed with the new needs. This execution stops when every negative resource is covered.
7. If after this process the number of agents responding to the petition is just one then the agent is not created because there already exists one agent capable of responding to the petition without the need of adding a new agent in the system. Otherwise, the agent is inserted into a cell that is a common father to every agent that will be executed upon execution of this agent. In the case that no cell exists with such properties (being the least upper bound of all the agents in the Petri net) a new cell is created and the structure of the cell tree is modified so that the father of all the cells that hold agents will be addressed to the new cell.

4.2. *Execution of an agent*

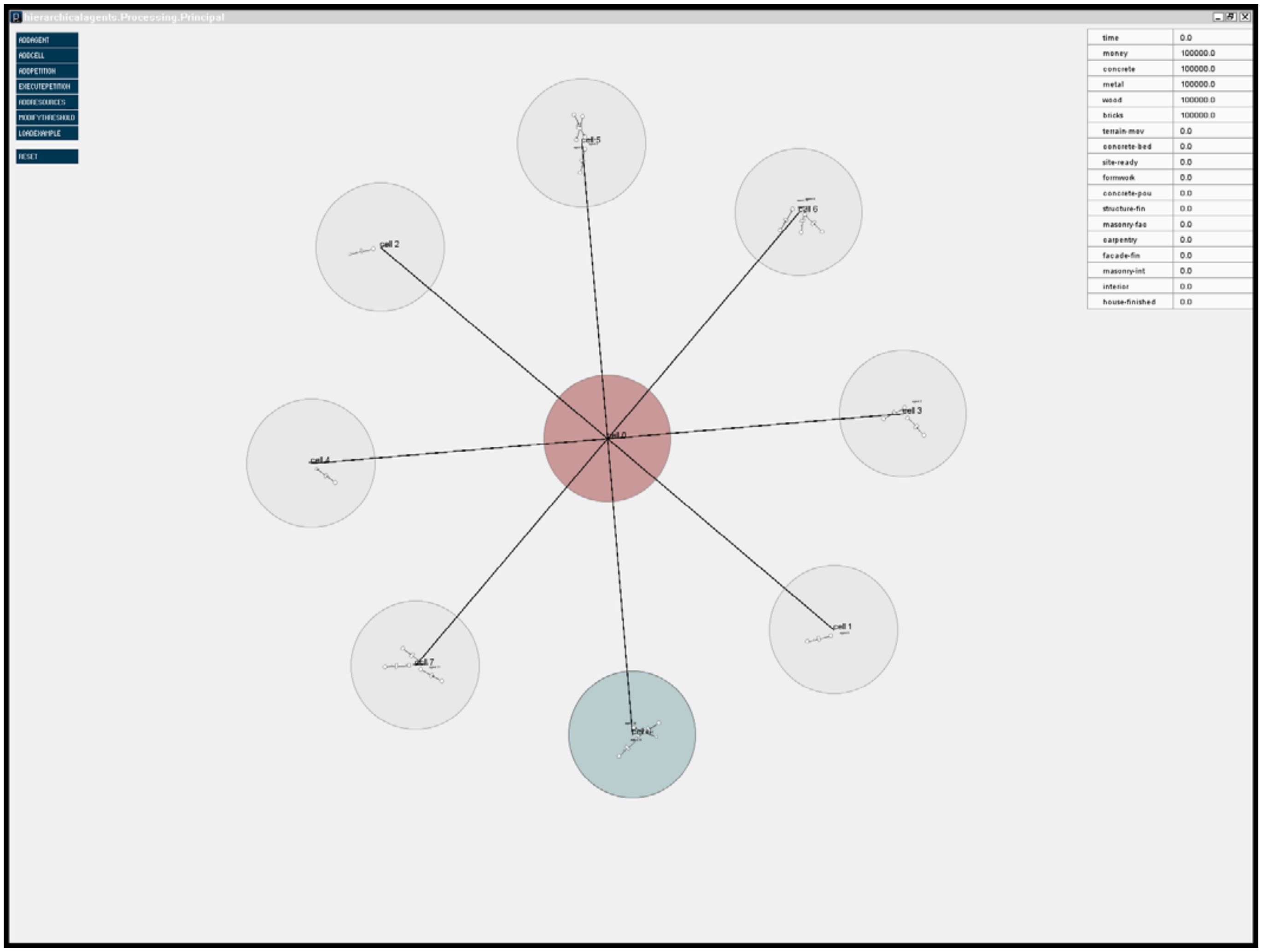
The execution of an agent essentially consists in the execution of its Petri net. Every transition will make a call to the agent that it withholds asking it to start working, so that they implement the transformation of resources.

When a transition is fired, the system checks whether the identifier of the transition coincides with the identifier of the agent to which the Petri net belongs to. Two things may happen:

- If these two identifiers are the same, then the system executes the transformation of resources, modifying the resources in the world.
- If the agent in charge of the transition is different from the one that calls it, then a message is sent to the aforementioned agent so that its execution starts.

5. Case Study: Definition of a site work

In this section we will make use of all the capabilities of our framework to model a simplified construction site. Obviously, in a real planning, every phase should be further subdivided, by the use of more agents. However, a reduced version will allow the reader to better understand how our approach works to model complex systems. We will also illustrate the use of our tool to support our methodology.



time	0.0
money	100000.0
concrete	100000.0
metal	100000.0
wood	100000.0
bricks	100000.0
terrain-mov	0.0
concrete-bed	0.0
site-ready	0.0
formwork	0.0
concrete-pou	0.0
structure-fin	0.0
masonry-fac	0.0
carpentry	0.0
facade-fin	0.0
masonry-int	0.0
interior	0.0
house-finished	0.0

Figure 4: Insertion of the agents in the cell tree.

In this section some terminology from the formal framework will be used (for further reference refer to the appendix of this paper).

5.1. *Definition of the resources of the system*

In order to complete the foreseen tasks, the system needs to have a certain amount of resources. We will therefore begin by adding a collection of resources. In this case, they have been assimilated as raw materials. The resources that we are adding (and its places in the tuple of resources) are: time(1), money(2), concrete(3), metal(4), wood(5), and bricks(6).

The rest of the tuple of resources will be applied to the following concepts: Terrain-movement(7), concrete-bed(8), site-ready(9), formwork(10), concrete-pouring(11), structure-finished(12), masonry-facade(13), carpentry-windows(14), facade-finished(15), interiors(16), finishes(17), and built-house(18).

Time is the only *special* resource in the sense that the amount that we add at the start represents the time the agent takes to accomplish the petition. In fact, although time is not an actual resource, it is included as the first element of the resource tuple to simplify notation, and so that it resembles the interface the user receives in the program. In this sense, if the resources are represented by $\overline{x}$, the tuple represented throughout this example will be $(time, x_1, x_2, \ldots, x_n)$.

5.2. *Definition of the atomic agents*

The first step will be to create the tree that contains the cells that will locate the agents. The system has already created the cell c_0 and we will add 7 sons to it, to be able to classify the different types of atomic agents correctly.

Cell c_1 will be used to hold agents related to *terrain movement*. We add a first atomic agent that when executed will consume 10 time units and 100 money units and create a unit of terrain-movement. The rest of the agents definition are similar and will be shown in the table portrayed in Table 1 (all of the agents are atomic), including information about the cell where they are inserted and what resources will they consume and produce.

The insertion of these agents is represented in Figure 4.

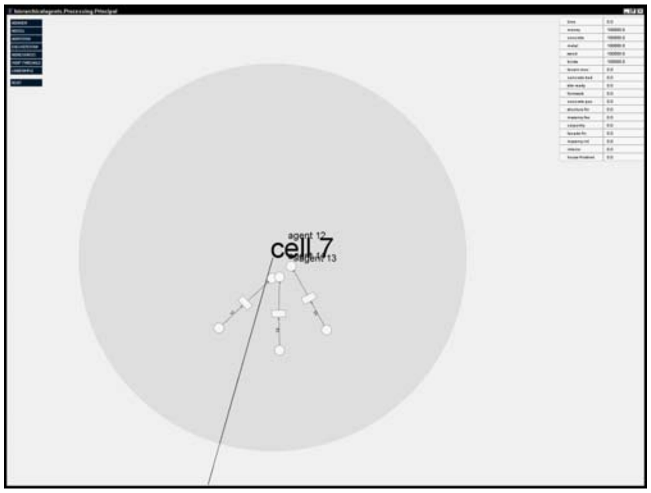


Figure 5: Close up of one of the cells, showing some atomic agents.

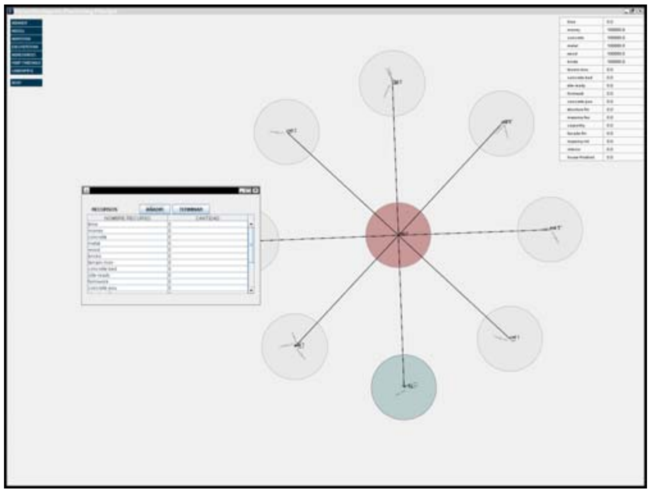


Figure 6: Preparing to create the first complex agent: Site is ready to build.

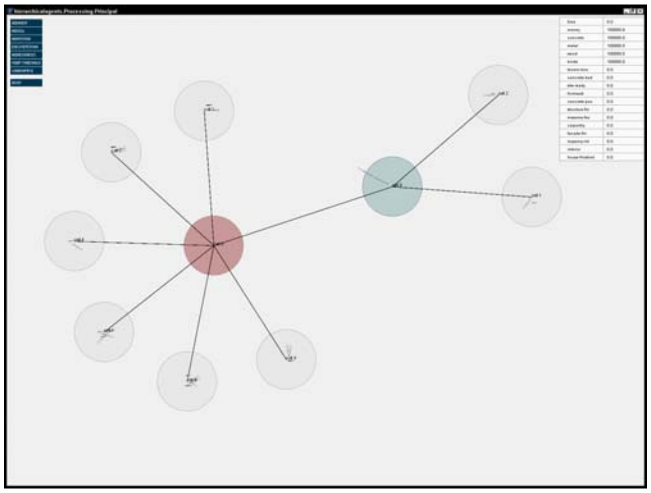


Figure 7: Creation of the first complex agent: Site is ready to build.

5.3. *First petition: Prepare the site to be built*

We will introduce a petition in the system that prepares the site to be built. We consider a petition $pet = (f^u, \overline{o}, \mathcal{A}_{pet})$ where:

- $f^u = 2 * x_1 + 1 * x_2 + .3 * x_3 + 1 * x_7 + 1 * x_8$ is the utility function (all values $0 * x_n$ are not represented). This utility function indicates the proportional importance that we give in our petition to each of the resources represented in it. That is, we value twice the amount of time that it takes to be fulfilled than the money, and three times more the amount of concrete used to implement this transformation of resources, we also give importance to the creation of the terrain movement and to the creation of the concrete-bed.

- $\overline{o} = (0, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)$. This means that we are asking for the creation of a concrete-bed, the system will have to look also for another agent that is capable of creating the terrain-movement, as it is a prerequisite for the application of the concrete-bed.

- $\mathcal{A}_{pet} = \{\}$ is the set of agents that are used by the petition (empty in the beginning).

The system sends the message with the petition and finds a_2 that supplies one unit of the resource where the negative value of the petition is. Thus, it adds agent a_2 to the petition so that $\mathcal{A}_{pet} = \{a_2\}$. Next, it creates agent a_{17}, adds a place, a transition connected to that place, and another place that connects to the transition in which agent a_2 is referenced to be executed in its turn.

Now, agent a_2 has added a new negative value in another position of the resources tuple, because in order to create the concrete-bed, it needs to be supplied a unit of the resource terrain-movement (in a physical meaning, it is the representation that before proceeding with the application of the concrete bed, the site needs to finish the terrain-movement). The petition sends a new message and agent a_1 replies. The petition adds agent a_1 so that $\mathcal{A}_{pet} = \{a_2, a_1\}$, adds another transition and another place with a connection that references a_1. Since $resources \geq \overline{0}$, the petition finishes and adds a token to the last place created. We compute the *least upper bound* of the cells that hold the agents that we are using, that is, $\bigsqcup\{c_i \mid c_i \in C \;\wedge\; a \in \mathcal{A}_{pet} \;\wedge\; a \in \mathcal{A}_{c_i}\}$. Since it is undefined, a new cell is created, and agent a_{17} is inserted in it. The transformation of resources from agent a_{17} is $\overline{s_{17}} = \sum_{a \in \mathcal{A}_{pet}} \overline{s_a}$.

The result can be observed in Figure 7.

5.4. *Second petition: Create structure*

Now we insert a new petition into the system, to construct the structure of the building. We consider a petition $pet = (f^u, \overline{o}, \mathcal{A}_{pet})$ where:

- $f^u = 2 * x_1 + 1 * x_2 + .3 * x_3 + 1 * x_{10} + 1 * x_{11}$ is the utility function (all values $0 * x_n$ are not represented).

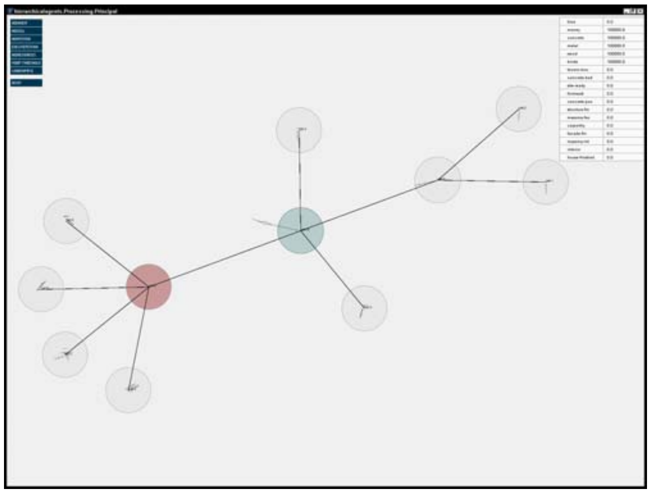


Figure 8: Creation of the second complex agent, creation of the structure.

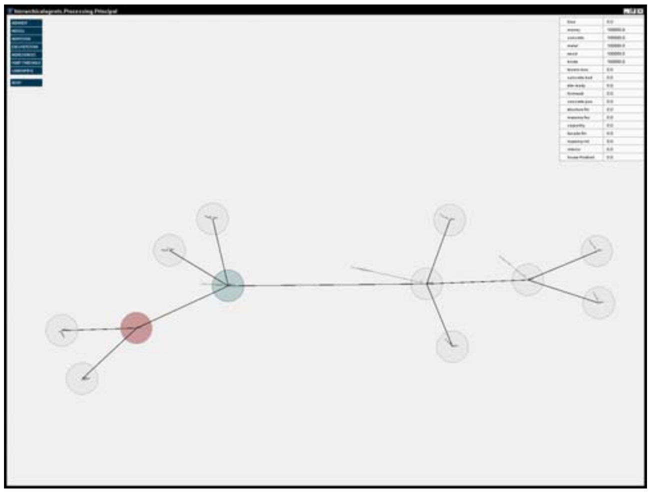


Figure 9: Creation of the third complex agent, the facade is finished.

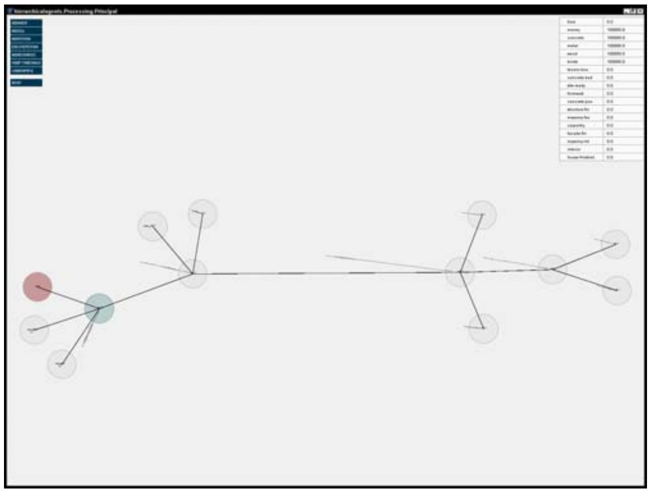


Figure 10: Final phase, the house is created.

- $\overline{o} = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0, 0)$.
- $\mathcal{A}_{pet} = \{\}$ is the set of agents that are used by the petition (empty in the beginning).

First, we find agent a_5 and re-send the petition after adding its resources. Next, we find two similar agents that are able to provide the new resource needed. Since both agents have identical utility functions, it does not choose between them but uses both of them in parallel to handle the petition quicker. If one of the agents would have a *bad* utility function, that is, if $f^u_{a_i} < \phi * max(f^u)_{a_3,a_4,a_5}$, then the system would have to decide whether it was under a certain threshold and, in this case whether it was better not to use it. Since in this case all utility functions fulfill $f^u_{a_i} \geq \phi * max(f^u)_{a_3,a_4,a_5}$, then they are parallelized. Next, we create two places with two connections that start in parallel and hold agents a_4 and a_3. Then, we re-send the petition and use the just created *complex* agent a_{17} as a starting point. The petition now looks for the lub (least upper bound) of the cells. Since it does not find it, it creates a new cell and inserts agent a_{18}. Again, the transformation of resources is given by $\overline{s_{18}} = \sum_{a \in \mathcal{A}_{pet}} \overline{s_a}$, such that $a = (id, ib, PN, s_a)$. Let us remind that now we have $\mathcal{A}_{pet} = \{a_5, a_4, a_3, a_{17}\}$. Figure 8 represents graphically this petition.

5.5. *Third petition: Create the facade*

Next, we want to create the facade of the building. We insert a new petition into the system to construct the facade. Let us consider a petition $pet = (f^u, \overline{o}, \mathcal{A}_{pet})$ where:

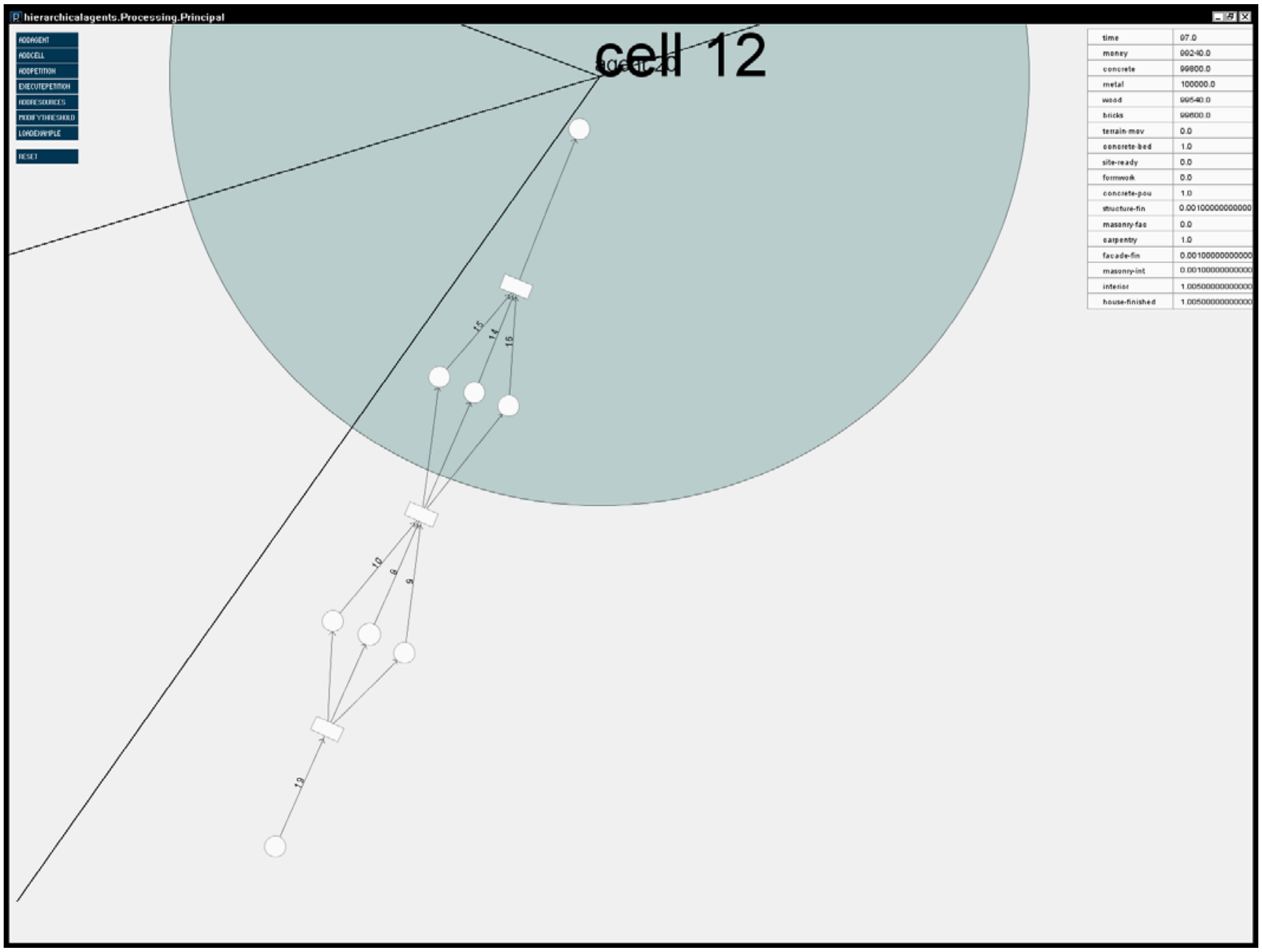
- $f^u = 2 * x_1 + 1 * x_2 + .3 * x_3 + 1 * x_4 + 1 * x_6 + 1 * x_{13} + 1 * x_{14}$ is the utility function (all values $0 * x_n$ are not represented).
- $\overline{o} = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0)$.
- $\mathcal{A}_{pet} = \{\}$ is the set of agents that are used by the petition (empty in the beginning).

The petition sends its messages, receives agents $\mathcal{A}_{pet} = \{a_{12}, a_{13}, a_6, a_7, a_8, a_{18}\}$, and creates a cell and agent a_{19} and inserts it. Figure 9 represents the result of this process.

5.6. *Fourth petition: Interiors and finishes*

Finally, we will base ourselves on the agents we were creating above and finish the building by adding the interior walls and the finishes. To do so, let us consider a petition $pet = (f^u, \overline{o}, \mathcal{A}_{pet})$, where:

- $f^u = 2 * x_1 + 1 * x_2 + .3 * x_3 + 1 * x_4 + 1 * x_6 + 1 * x_{15} + 1 * x_{16} + 2 * x_{17} + 1 * x_{18}$ is the utility function (all values $0 * x_n$ are not represented).
- $\overline{o} = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 0)$.
- $\mathcal{A}_{pet} = \{\}$ is the set of agents that are used by the petition (as usual, the set is empty in the beginning).



time	97.0
money	99240.0
concrete	99800.0
metal	100000.0
wood	99540.0
bricks	99600.0
terrain-mov	0.0
concrete-bed	1.0
site-ready	0.0
formwork	0.0
concrete-pou	1.0
structure-fin	0.00100000000000
masonry-fac	0.0
carpentry	1.0
facade-fin	0.00100000000000
masonry-int	0.00100000000000
interior	1.00500000000000
house-finished	1.00500000000000

Figure 11: Close up of the final agent a_{20}.

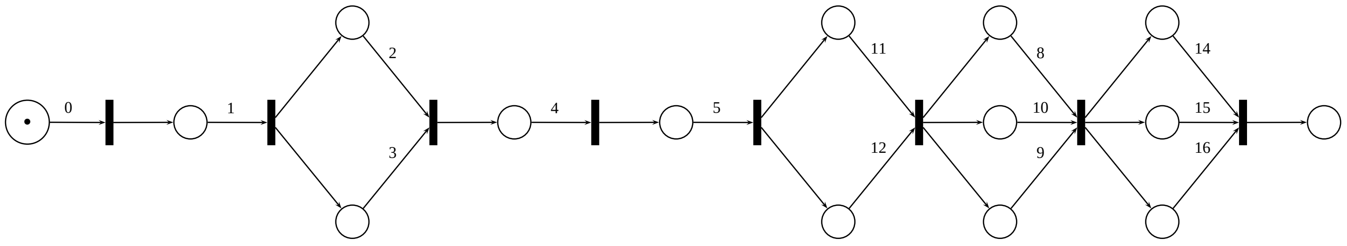


Figure 12: Resulting Petri net.

The petition sends its messages, receives the responses from agents $\mathcal{A}_{pet} = \{a_{15}, a_{14}, a_{11}, a_{10}, a_9, a_{19}\}$, creates a new cell and inserts the newly created agent a_{20} in it. Agent a_{20} already provides the built house. So, if at any other moment in time we need to construct a house with the same characteristics, we will just need to call upon agent a_{20} and it will make all subsequent calls to every other agent. Figure 10 represents the insertion of this last agent.

5.7. Resulting Petri net from the whole process

If we were to expand subsequently all the complex agents into a complete Petri net of the whole process we would obtain the Petri net graphically depicted in Figure 12. This expanded version is equivalent to the one that agent a_{20} reflects and corresponds to insert all the atomic agents and then perform a petition in which we ask the house to be directly created. This is similar to consider an agent of the construction process in charge of personally calling every other person involved in constructing the house and tell them what to do, without intermediaries. This complex agent will be inserted in a new cell, that has every other cell as its children, and the passing of messages would behave as any other complex agent, through the cell structure.

6. Conclusions

We have defined a multi-agent system that allows the user to define a hierarchical structuring of the tasks that these agents perform. Our framework provides a natural way to plan a schedule of tasks and permits parallel and sequential calling of the agents, forming a structure similar to a Critical Path Graph, that would allow a project manager to ease the definition of this part of the project. Our goal is to provide a system that automatically constructs a graph, in the form of a Petri net [1], representing the sequence of jobs to be undertaken during a construction process, forming a critical path graph.

We are aware that our formalism is difficult to understand since there is a lot of mathematical machinery underlying the definition of our systems. Thus, we have decided to build a tool that fully implements our methodology. In this way, a user of our methodology does not need to pay attention to the formal details and can concentrate on defining the appropriate hierarchical structure.

Our approach allows the user to model systems that will expand with every use. Declaring all the possible atomic tasks that a system can perform as outputs allows the system to complete any petition that the user can foresee. This is done through recombining atomic agents. We continue to add complex agents in every interaction with the tool. Thus, the system is able to perform more complex tasks with each use, that will not need to be re-computed. Let us note that the distinction between *atomic* and *complex* agents is fundamental since without this separation, every behavior of the system would need to be pre-implemented before needing it.

We had presented a previous version of the formalism [35, 36]. It has been through working on these papers that we realized some details that, when resolved, would create a more complete and flexible approach. This led to the substitution of paths by Petri nets and finding a way to automatically create the cell tree. The use of Petri nets, in the enhanced model, has been a great advance in relation with the published papers since it has added the possibility of parallelizing tasks (agents). This does not only allow us to shorten execution time, but in some cases even creates new emerging behaviors. Also, the automatic creation of the cell tree, due to the computation of the least upper bound of a cell set, is a big advantage. It allows us to keep an order, a conceptual structure, of how agents are inserted into the tree. Thus, with this advance, the system is able to save time in its searches. This feature depends on the way the system sends the messages, through its cells, and from them down to each of their sons. Moreover, keeping close in the hierarchy agents that perform similar tasks is an automated feature. Therefore, this represents an improvement with respect to a manual procedure for the insertion of agents, in which agents could have been inserted anywhere.

Acknowledgements

We would like to thank César Andrés with whom we developed the initial formal language that led, after several transformations and improvements, to the actual framework and language.

We would like to thank the anonymous reviewers for the careful reading and their useful comments.

Research partially supported by the Spanish MEC project TESIS (TIN2009-14312-C02-01).

Appendix

In this appendix we will present the formal framework of our approach. We will begin by outlining some preliminary notation regarding Petri nets and message communication in Appendix A and then present the formal definition of the framework itself in Appendix B.

Appendix A. Preliminaries

In this Section we briefly comment on the variant of the Petri nets formalism that we use in our approach and define the messages that we allow to be exchanged between agents. Along the appendix we will consider a set ID^a that contains all the possible agent identifiers.

Definition 1. In this work, a *Petri net* is a tuple $PN = (P, T, F, M_0, FP)$ where:

- $P = \{p_1, p_2, \dots, p_j\}$ is a set of places.
- $T = \{t_1, t_2, \dots, t_i\}$ is a set of transitions.
- $F : (P \times T \times ID^a) \cup (T \times P) \rightarrow \{0, 1\}$ assigns a weight of one (or zero if the connection does not exist) to every connection between places and transitions and viceversa. It also adds an agent identifier to the connection between a place and a transition.

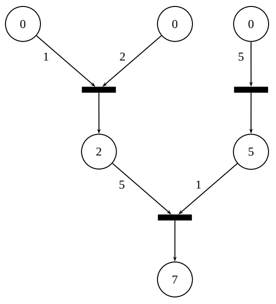


Figure 13: A complex agent used to show how time evolves in the Petri net.

- $M_0 : P \to \mathbb{N} \cup \{-1\}$ is the original marking. In this work, the Petri net is, by construction *1-safe*, that is, there is at most one token in each place. Let us note that the marking is returning the token age, not the number of tokens in the place. If there is not a token in the place, the marking returns -1. Initially all tokens are set to 0.

- $FP \subseteq P$ is the set of *final* places that will be considered as final places, that is, when one of these places is reached the execution of the Petri net finishes.

- Let $t \in T$ be a transition. We define the posset of t as $t\bullet = \{p_i \mid \exists f_i = (t, p_i) : F(f_i) = 1\}$. We define the preset of $t \in T$ as $\bullet t = \{p_i \mid \exists f_i = (p_i, t, -) : F(f_i) = 1\}$.

Next, we define the evolution of the Petri net, as follows. We consider that there exists a function $\mu : ID^a \to \mathbb{N}$ that returns the time that an agent with identifier *id* takes to fulfill its task. This function is formally introduced in Definition 3.

- The firing of a transition t, denoted by $M \to_{PN,t,id} M'$, may happen if $\forall p \in \bullet t, M(p) \neq -1$ and M' fulfills the following conditions
  - $\forall p \in \bullet t, M'(p) = -1$
  - $\forall p' \in t\bullet, M'(p') = max\{M(p) + \mu(id) | F(p, t, id) = 1\}$

- We write $M \to_{PN} M'$ if there exists t and id such that $M \to_{PN,t,id} M'$.

□

Example 3. *We will introduce a simple example to show the time evolution of the Petri net. The number in the places represent the age the token would have when the evolution takes it to that place, and the numbers in the arcs represent the time that the corresponding atomic agents take to fulfill its tasks. The example is shown in Figure 13.*

The following definition introduces the different kinds of messages that can be sent in our framework.

There exists two different kinds of messages, depending if they are used during the phase of the creation of the petition, or during the execution of the Petri net.

- The first family of messages corresponds to the *BROADCAST* and *REPLIES* identifiers.

- The second family of messages corresponds to the *STARTJOB* and *FINISHEDJOB* identifiers.

Definition 2. We denote by ID^m the set of identifiers for messages and by $\mathcal{M}$ the set of all messages.
Messages from the first family are tuples in $ID^m \times ID^t \times ID^a \times \mathbb{R}^n$. Therefore, if we have a message $m = (id_m, t_{act}, a_o, \overline{r})$ then:

- $id_m \in \{BROADCAST, REPLIES\}$ denotes the nature of the message.

- $t_{act} \in ID^t$ is the identifier of the actual transition of the Petri net that originated the *BROADCAST* MESSAGE (see Appendix 2.1 for an extended explanation).

- $a_o \in ID^a$ is the agent that originates the message.

- $\overline{r}$ is the tuple of resources needed in the *BROADCAST* message and the one that can supply the agent origin of the *REPLIES* message.

Messages from the second family are in $m \in ID^m \times ID^a \times ID^a$. Therefore, if we have a message $m = (id_m, a_0, a_f)$ then:

- $id_m \in \{STARTJOB, FINISHEDJOB\}$ denotes the nature of the message.

- $a_0 \in ID^a$ is the agent origin of the message.

- $a_f \in ID^a$ is the agent objective of the message.

□

Appendix B. Definition of the formalism

A transformation of resources is represented by a tuple $\overline{s} \in \mathbb{R}^n$. Intuitively, a positive component of the tuple denotes that the agent produces s_i units of the *i-th* resource while a negative component denotes that the transition consumes s_j units of the *j-th* resource.

Next we show how to represent the *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. Agents have unique identifiers assigned. These identifiers can be seen as a word that denotes the concept that the agents represent.

Definition 3. An agent is a tuple $a = (id, ib, PN, \overline{s}, ti)$ where:

- $id \in ID^a$ is the agent identifier.

- $ib \subseteq \mathcal{M}$ is the input buffer.

- $PN = (P, T, F, M_0, FP)$ is a Petri net.

- $\overline{s}$ is the overall transformation of resources that the agent accomplish. For an atomic agent this vector will be equal to the transformation induced by the connection from the first place to its transition and for a complex agent this vector will be the addition of the vectors of all the agents nested into itself.

- $ti \in \mathbf{N}$ is the time that the agent needs to complete its tasks. The time that an atomic agent takes to complete its task is defined at the time of creation of the agent. In contrast, the time for a complex agent is the age of the token in the final place, once a simulated run from its Petri net has finished. This is the time returned by the μ function

An agent is called atomic if it is in charge of executing a single task. Formally, for all $a = (id_a, ib, PN, \overline{s}, ti)$ is *atomic*, if $PN = (P, T, F, M_0, FP)$, and $f_1 = (p_1, t_1, id_f)$, such that $F(f_1) = 1$, then a is an *atomic* agent if the following restrictions hold: $| P | = 2$, $| T | = 1$, and $id_f = id_a$.

We denote by $\mathcal{A}$ the set of all agents.

□

Let us note that we do not store the order in which messages are received in the buffer. That is why we define a buffer as a set. Let us also note that the notion of *atomic* agent means that the agent is itself in charge of executing the transformation of resources.

Cells serve as baskets of agents to reunite, organize, conglomerate and handle petitions as well as calls to the agents. Abstractly, a cell is the macro-concept that holds the set of instances (agents) related in between them.

Definition 4. A *cell* is a tuple $(\mathcal{A}_{cell}, id, \text{Sons}, \text{Father}, ib)$ where

- $\mathcal{A}_{cell} \subseteq ID^a$ is the set of agents that belong to the cell.
- $id \in ID^c$ is a unique identifier for this cell. This can be seen as the concept that it represents.
- $\text{Sons} \subseteq ID^c$ is the set of identifiers of the sons of this cell. If $\text{Sons} = \varnothing$ then we are in a node cell.
- $\text{Father} \in ID^c$ is the identifier of the cell that is father of this cell. If Father=`nil` then we are in the initial cell, from which all other cells are defined.
- $ib \subseteq \mathcal{M}$ is the input buffer where messages will be stored.

We denote by C the set of all cells. □

Next, we define the whole system that contains in a tree like structure implicity defined by the father-son relationship, the cells that conform the whole system. This allows a hierarchical structuring of concepts.

Definition 5. A *system* is a tuple $w = (c_0, \overline{x}, \phi)$, where

- c_0 is the origin cell.
- $\overline{x} \in \mathbf{R}^n$ is the set of available resources in the system.
- ϕ is a threshold value that is used to discriminate between good and bad values of the utility functions.

□

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 6. Let $w = (c_0, \overline{x}, \phi)$ be a system. A *petition* is a tuple $pet = (f^u, \overline{o}, \mathcal{A}_{pet})$, where

- $f^u \in \mathcal{F}$ is a utility function.
- $\overline{o} \in \mathbf{R}^n_-$ is the objective of the transitions, that is, the vector of resources that we expect to have after performing the petition.
- $\mathcal{A}_{pet} \subseteq ID^a$ is the set of agents capable of answering the petition. Initially this set is empty, and the petition fills it as it searches through the system.

We denote by $\mathcal{PET}$ the set of all petitions. We say that a petition $pet = (f^u, \overline{o}, \mathcal{A}_{pet})$ is fulfilled when

$$\sum_{a_i \in A_{pet}} \overline{s^a_i} + \overline{o} + \overline{x} \geq \overline{0}$$

where for each agent belonging to the set A_{pet} we have $a_i = (id, ib, PN, \overline{s^a_i}, ti)$. □

Next, we define a function GF that will be used to climb through the cell tree, until we reach a cell that has as father c_0.

Definition 7. Let $c = (\mathcal{A}_{cell}, id, \text{Sons}, \text{Father}, ib) \in C$, and $w = (c_0, \overline{x}, \phi)$ be a system. We define $GF : C \to C$ as:

$$GF(c) = \begin{cases} c & \text{If Father} = c_0 \text{ or Father} = nil \\ GF(\text{Father}) & \text{Otherwise} \end{cases}$$

□

A petition deepens into the cell tree structure looking for a combination of agents capable of handling the needed transformation of resources. Subsequently, it creates an agent in the way described below.

Definition 8. The function $constrAg : \mathcal{PET} \to \mathcal{A}$, that creates an agent from a specific petition, is defined as follows. Let $pet = (f^u, \overline{o}, \mathcal{A}_{pet})$ be a petition. Then, $constrAg(pet) = (id, ib, PN, \overline{s}, ti)$, where id is a fresh agent identifier (created by the function *newIdAgent*) and ib is an empty new buffer. The resulting agent will have a connection between a place and a transition in PN for each agent in the set $\mathcal{A}_{pet}$. An invocation of this agent will generate a call to each of those agents to execute their associated tasks. The transformation of resources that this agent will accomplish is $\overline{s} = \sum_{a_i \in \mathcal{A}_{pet}} \overline{s^a_i}$, where for each agent in the set A_{pet} we have $a_i = (id_a, ib_a, PN_a, \overline{s^a_i}, ti_i)$. Finally ti will be calculated by making a simulated run of the Petri net, in which no resources are transformed, and assigning the value of the age of the token in the final place. □

Before we insert the new agent into the tree structure, we must define how to compute the *least upper bound* of a set of agents.

Definition 9. The least upper bound (in short, lub) of a set of agents, given by a function $\bigsqcup : \wp(C) \rightarrow C$, is induced by the following order relation: $a \leq b$ iff there exist a descending path through the cell tree that goes from b to a. We define the lub as the lowest cell (in terms of the level in the tree) that remains a common path to reach all the cells in the set.

Insertion of an agent: Let $w = (c_0, \overline{x}, \phi)$ be a system and $pet = (f^u, \overline{o}, \mathcal{A}_{pet})$ be a petition, let $cells_{pet} = \{c \mid \exists c : c = (\mathcal{A}_{cell}, id, \text{Sons}, \text{Father}, ib) \in C \ \wedge \ \exists a : a \in \mathcal{A}_{pet} \ \wedge \ a \in \mathcal{A}_{cell}\}$ be a set of cells, and $a_{new} = constrAg(pet)$ be the agent to be inserted. The a_{new} agent is inserted into a cell as follows:

- If there exists $\bigsqcup(cells_{pet})$ and $\bigsqcup(cells_{pet}) \neq c_0$, then insert a_{new} in $\bigsqcup(cells)$.
- Otherwise, let us consider the set $Fcells = \{GF(c) \mid c \in cells_{pet}\}$, we insert a_{new} into $c_{new} = (\{a_{new}\}, newIdCell(w), Fcells, c_0, ib)$ where ib is an empty buffer. In addition, for every element belonging to $Fcells$, change the father to be c_{new}.

□

Appendix B.1. Steps of a petition

Next we formally present how petitions are handled in our approach.

1. Let $w = (c_0, \overline{x}, \phi)$ be our system. Let $pet = (f^u, \overline{o}, \mathcal{A}_{pet})$ be our petition:
   - First we use a temporal vector of resources $\overline{z}$ that originally is assigned the value of $\overline{o}$, that is, $\overline{z} \leftarrow \overline{o}$.
   - The petition creates a temporal agent, $constrAg(pet)$, $a_{pet} = (id_{pet}, ib_{pet}, PN_{pet}, \overline{s}_{pet}, ti)$, where we have $PN_{pet} = (P, T, F, M_0, FP)$. The petition creates and inserts a place (p_0) in this newly created Petri net, that is, $P = P \cup \{p_0\}$. This place will also be added to the final place set: $FP = FP \cup \{p_0\}$. It also creates and adds a transition t_0, $T = T \cup \{t_0\}$. Finally the petition creates a connection between the transition and the place, that is, $F = F \cup \{((t_0, p_0), 1)\}$.
   - We will also consider a special transition t_{act} initially assigned as t_0 that represents the actual transition to which the places in the next turn must be linked.
2. Next, the petition must discriminate depending on the number of negative resources in $\overline{z}$:
   - We consider a function $\varphi : \mathbb{R}^n \rightarrow \mathbb{N}$ that returns the number of negative resources.
   - If $\varphi(\overline{z}) = 1$, then the petition sends a message $m = (BROADCAST, t_{act}, \star, \overline{z})$ to the input buffer of every cell in the tree.
   - If $\varphi(\overline{z}) > 1$, then we would use another function $\chi : \mathbb{R}^n \rightarrow \wp(\mathbb{R}^n)$ that subdivides the resources from the petition in a collection of vectors of resources in which only one negative resource is allowed. This operation allows us to create parallel calling of the agents in the resulting Petri net and sends the corresponding messages. As an example let us consider that $\varphi(\overline{z}) = 2$ and $\chi(\overline{z}) = \{\overline{z_1}, \overline{z_2}\}$. Then messages $m_1 = (BROADCAST, t_{act}, \star, \overline{z_1})$ and $m_2 = (BROADCAST, t_{act}, \star, \overline{z_2})$ will be sent simultaneously.
3. The cells retrieve the message from their input buffers (Choose(ib)) and retransmit the message in their input buffer to the agents that they withhold. So, for every agent in the set $\{a_i = (id_a, ib_a, PN, \overline{s}, ti) \mid a_i \in \mathcal{A}_{cell}\}$ retransmit the message to its input buffer, that is, to ib_a.
4. The agents handle the message from the input buffer as follows: Let $a_i = (id_a, ib_a, PN, \overline{s}, ti)$ be the agent that is handling the message and let $m = (BROADCAST, t_{act}, \star, \overline{z})$ be the message being handled, if the agents internal transformation creates a resource that is negative in the resources of the petition, that is there exists i such that $s_i > 0, z_i < 0$ considering s_i as the i-*th* element of $\overline{s}$ and z_i as the i-*th* element of $\overline{z}$, then the agent sends back a message saying that he can fulfill the petition: $m = (REPLIES, t_{act}, a_i, \overline{s})$.
5. If the number of *REPLIES* messages with the same t_{act} is greater than one, then the petition uses the utility function to discriminate between the different possibilities. The way this is handled is by using a threshold value $0 < \phi < 1$, defined in the system ($w = (c_0, \overline{x}, \phi)$), we calculate the utility functions of all the agents involved and using the maximal value $max(f^u) = max(f^u(\overline{s_1^a}), \ldots, f^u(\overline{s_n^a}))$, where n is the number of agents that have answered with a *REPLIES* message: In the case where one agent's utility function is below the multiplication of this threshold by the maximal utility function $f^u(\overline{s_i^a}) < \phi * max(f^u)$ with $0 < i \leq n$ then agent a_i is discarded from the set. All other agents are parallelized, as follows:
   - We create a transition t_j.
   - Add t_j to the transition set: $T = T \cup \{t_j\}$.
   - Afterwards for every agent $a_i = (id_a, ib, PN, \overline{s}, ti)$, we do:
     (a) We update the set $A_{pet} = A_{pet} \cup \{a_i\}$.
     (b) We create a new place p_i with $0 < i \leq n$, where n is the number of agents
     (c) $P = P \cup \{p_i\}$.
     (d) $F = F \cup \{(p_i, t_{act}, id_a, 1), ((t_j, p_i), 1)\}$ where id_a the identifier of the agent.
     (e) We recalculate the resources needed by the petition, $\overline{z} \leftarrow \overline{z} + \overline{s}$.
     (f) We check if the petition is already fulfilled. Let our system be $w = (c_0, \overline{x}, \phi)$, then:
       – If $\overline{z} + \overline{x} \geq \overline{0}$, then we add a token in M_0 to the posset of t_j, that is, $M_0(t_j\bullet) = 1$. And stop adding agents. Delete t_j, that is, $T = T \backslash \{t_j\}$ and for every connection in F such that $f_i = (t_j, -)$ we set $F(f_i) = 0$. Afterwards we insert the agent in its cell, as explained under this lines.
       – Otherwise, we update $t_{act} \leftarrow t_j$. and re-send a message with the following information, $m = (BROADCAST, t_{act}, id, \overline{z})$.

6. Last, if $|T| = 1$ then we just call execution of the agent in charge of that transition. That is because, it means that there is already an agent capable of handling the petition in the system. In any other case, we calculate the lub of the cells that hold the agents, that is, $\bigsqcup\{c \mid \exists c : c = (\mathcal{A}_{cell}, id, \text{Sons}, \text{Father}, ib) \in C \ \wedge \ \exists a : a \in \mathcal{A}_{pet} \ \wedge \ a \in \mathcal{A}_{cell}\}$ as explained in its definition, and create a new cell if it does not exist. We insert the agent in the cell and execute it.

Appendix B.2. Execution of an agent

Let $w = (c_0, \overline{x}, \phi)$ be a system, $a = (id, ib, PN, \overline{s}, ti)$ be the agent that we are executing, let $PN = (P, T, F, M_0, FP)$ be its Petri net, we define the function $execute(id)$ as:

1. For all p such that $F(p, t, id_2) = 1 \ \wedge \ M_0(p) > -1$ perform the following steps:
   (a) $\begin{cases} \text{If } id_2 = id \text{ then } a \text{ is an } atomic \text{ agent. Therefore, we transform the resources of the system: } \overline{x} \leftarrow \overline{x} + \overline{s}. \\ \text{Otherwise, we call upon } execute(id_2)\text{, by sending } m = (STARTJOB, id, id_2). \end{cases}$
   (b) If either $id = id_2$ or there exists $m = (FINISHEDJOB, id_2, id) \in ib$, then we continue the evolution of the Petri net and delete the message from ib (the buffer of id).
2. Next, we check whether the execution of the Petri net is finished.
   - If for all $p_i \in FP$, we have $M_0(p_i) = 1$, then we finish the execution and if there exists $m = (STARTJOB, id, id_2) \in ib$, then the system will send a message $m = (FINISHEDJOB, id_2, id)$, and delete the former message from the buffer of id_2.
   - Otherwise we return to step 1 and continue.

References

[1] T. Murata, Petri Nets: Properties, Analysis, and Applications, Proceedings of the IEEE 77 (4) (1989) 541–580.

[2] A. Mas-Colell, M. Whinston, J. Green, Microeconomic Theory, Oxford University Press, 1995.

[3] M. Wooldridge, An Introduction to MultiAgent Systems, John Wiley & Sons Ltd, 2002.

[4] Y. Shoham, K. Leyton-Brown, Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations, Cambridge University Press, 2008.

[5] F. Zambonelli, N. Jennings, M. Wooldridge, Organisational Rules as an Abstraction for the Analysis and Design of Multi-Agent Systems, Journal of Software Engineering and Knowledge Engineering 11 (3) (2001) 303–328.

[6] R. Brooks, Elephants don't play chess, Robotics and Autonomous Systems 6 (1990) 3–15.

[7] P. Maes, The Dynamics of Action Selection, in: 11th Int. Joint Conf. on Artificial Intelligence, IJCAI'89, Morgan Kaufmann, 991–997, 1989.

[8] Z. Ren, C. Anumba, Multi-agent systems in construction: State of the art and prospects, Automation in Construction 13 (3) (2004) 421–434.

[9] J. Wang, X. Liu, A Task-Based Modeling Method for Process Modeling and Automation in Project Management, in: 4th Int. Conf. on Wireless Communications, Networking and Mobile Computing, WiCOM '08, IEEE Computer Society Press, 1 –4, 2008.

[10] A. Sawhney, H. Bashford, K. Walsh, A. Mulky, Agent-based Modeling and Simulation in Construction, in: 35th Winter Simulation Conf., IEEE Computer Society Press, 1541 – 1547, 2003.

[11] G. Howel, White Paper for Berkeley/Stanford CE&M Research Workshop, www.ce.berkeley.edu/~tommelein/CEMworkshop/Howell.pdf, 1999.

[12] K. Kim, K. J. Kim, Multi-agent-based simulation system for construction operations with congested flows, Automation in Construction 19 (7) (2010) 867 – 874.

[13] H. Li, Petri net as a formalism to assist process improvement in the construction industry, Automation in Construction 7 (4) (1998) 349 – 356.

[14] W. van der Aalst, M. Stoffele, J. Wamelink, Case handling in construction, Automation in Construction 12 (3) (2003) 303 – 320.

[15] Y. Zhu, G. Augenbroe, A conceptual model for supporting the integration of inter-organizational information processes of AEC projects, Automation in Construction 15 (2) (2006) 200 – 211.

[16] A. Sawhney, Petri net based simulation of construction schedules, in: 29th Winter simulation Conf., IEEE Computer Society, 1111–1118, 1997.

[17] A. Sawhney, O. Abudayyeh, T. Chaitavatputtiporn, Modeling and Analysis of a Concrete Production Plant Using Petri Nets, Journal of Computing in Civil Engineering 13 (3) (1999) 178–186.

[18] D. Goldberg, Genetic Algorithms in Search, Optimisation and Machine Learning, Addison-Wesley, 1989.

[19] C. Chang, M. Christensen, T. Zhang, Genetic algorithms for Project Management, Annals of Software Engineering 11 (1) (2001) 107–139.

[20] B. Boehm, Software Engineering Economics, Prentice Hall, 1981.

[21] P.-H. Chen, H. Weng, A two-phase GA model for resource-constrained project scheduling, Automation in Construction 18 (4) (2009) 485–498.

[22] L. Long, A. Ohsato, A genetic algorithm-based method for scheduling repetitive construction projects, Automation in Construction 18 (4) (2009) 499–511.

[23] M. Dorigo, V. Maniezzo, A. Colorni, The Ant system: Optimization by a colony of cooperating agents, IEEE Transactions on Systems, Man and Cybernetics B 26 (1) (1996) 29–41.

[24] S. Christodoulou, Construction imitating ants: Resource-unconstrained scheduling with artificial ants, Automation in Construction 18 (3) (2009) 285–293.

[25] Q. Duan, T. W. Liao, Improved ant colony optimization algorithms for determining project critical paths, Automation in Construction 19 (6) (2010) 676–693.

[26] H. Abdallah, H. Emara, H. Dorrah, A. Bahgat, Using ant colony optimization algorithm for solving project management problems, Expert Systems with Applications 36 (6) (2009) 10004–10015.

[27] X. Ning, K. Lam, M.-K. Lam, Dynamic construction site layout planning using max-min ant system, Automation in Construction 19 (1) (2010) 55–65.

[28] J. Kennedy, R. Eberhart, Particle swarm optimization, in: IEEE Int. Conf. on Neural Networks, ICNN'95, 1942–1948, 1995.

[29] M. Lu, H.-C. Lam, F. Dai, Resource-constrained critical path analysis based on discrete event simulation and particle swarm optimization, Automation in Construction 17 (6) (2008) 670–681.

[30] J.-H. Chen, L.-R. Yan, M.-C. Su, Comparison of SOM based optimization and particle swarm optimization for minimizing the construction time of a secant pile wall, Automation in Construction 18 (6) (2009) 844–848.

[31] C.-W. Feng, Y.-J. Chen, J.-R. Huang, Using the MD CAD model to develop the time-cost integrated schedule for construction projects, Automation in Construction 19 (3) (2010) 347–356.

[32] H. Li, N. Chan, T. Huang, H. Guo, W. Lu, M. Skitmore, Optimizing construction planning schedules by virtual prototyping enabled resource analysis, Automation in Construction 18 (7) (2009) 912–918.

[33] K. Kim, J. Paulson, J. Petrie, V. Lesser, Compensatory Negotiation for Agent-Based Project Schedule Coordination, 2000.

[34] J. Fonseca, E. D. Oliveira, A. Steiger-Garçao, A DAI Based Resource Management System, Applied Artificial Intelligence 11 (6) (1997) 525–550.

[35] C. Andrés, C. Molinero, M. Núñez, 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, 169–176, 2008.

[36] C. Andrés, C. Molinero, M. Núñez, A Hierarchical Methodology to Specify and simulate complex computational Systems, in: 9th Int. Conf. on Computational Science, ICCS'09, LNCS 5544, Springer, 347–356, 2009.