University of Hradec Králové

Faculty of Informatics and Management

Department of Information Technologies

Human Activities Simulation Based on Fuzzy Cognitive Maps

Doctoral Thesis

Author: Ing. Tomáš Nacházel

Study program: P1802

Field of study: Applied Informatics

Supervisor: doc. RNDr. Kamila Štekerová, Ph.D.

Department of supervisor: Department of Information Technologies

Hradec Králové October 2019

Declaration

I declare that this thesis and the work presented in it are my own original research and it has been generated by me using only sources cited in the chapter References.

Hradec Králové Ing. Tomáš Nacházel

Acknowledgements

I would like to thank my supervisor doc. RNDr. Kamila Štekerová, Ph.D. for her assistance and constructive criticism throughout my Ph.D. study and preparation of this work. I am also grateful for the help and useful recommendations by RNDr. Petr Tučník, Ph.D. and others who participated in projects and prepared publications with me. Finally, I would like to thank my family for support throughout my whole study.

Annotation

The aim of this work is the design and implementation of a method for human activity simulation. Regarding decision-making, the desired behavior is characterized by both need-oriented goals and routines. To accomplish this objective, this work uses own modification of fuzzy cognitive maps for autonomous agents that has been enhanced by sub-system which enables agents to perform activities planned through scenarios. The developed method was verified by implementation into ambient intelligence model and compared to other projects with similar aim or objective. The comparison shows the complexity of the proposed design and unique features like a combination of need-oriented and routine behavior and adaptation of behavior to specific scenarios by genetic algorithms.

Keywords: human activity simulation; fuzzy cognitive maps; scenarios; routines; multi-agent systems; ambient intelligence; genetic algorithms

Název: Simulace lidského chování založená na fuzzy kognitivních mapách

Anotace

Tato práce se zabývá návrhem a implementací metody pro simulaci lidského chování. Z pohledu rozhodování se takové chování vyznačuje zaměřením na potřeby a zároveň opakujícími se rutinami. K řešení tohoto problému byla využita vlastní úprava fuzzy kognitivních map pro autonomní agenty, která byla obohacena o sub-systém, který umožňuje vykonávat plánované aktivity pomocí scénářů. Vytvořená metoda byla ověřena implementací do modelu ambientní inteligence a porovnána s projekty podobného zaměření nebo cíle. Z porovnání vyplývá komplexnost navrženého řešení a unikátní vlastnosti jakými jsou například kombinace potřebami i rutinami řízeného chování a adaptace chování pro specifické scénáře pomocí genetických algoritmů.

Klíčová slova: simulace lidského chování; fuzzy kognitivní mapy; scénáře; rutinní chování; multiagentové systémy; ambientní inteligence; genetické algoritmy

Content

List of used abbreviations

#	Abbreviation	Description
1	FCM-NAS	Fuzzy cognitive map enhanced by three classes (Needs, Activities, States) for autonomous agents
2	AHP	Analytic hierarchy process
3	ALModel	Artificial life model
4	DFCM	Dynamic fuzzy cognitive map
5	FCM	Fuzzy cognitive map
6	HSS	HomeSensorSimulator
7	KCD	Kingdom Come: Deliverance
8	MAS	Multi-agent system
9	MASSHA	Multi-agent system for human activities simulation
10	NPC	Non playable character
11	OpenSHS	Open Smart Home Simulator
12	PerSim	Ambient intelligence simulator
13	PRNG	Pseudorandom number generator
14	PS	Primary state (in FCM-NAS)
15	RFID	Radio-frequency identification

List of used symbols

#	Symbol	Description
	a_c	The critical level for processing *Activity* nodes
	c_i	The *i*-th concept node in an FCM
	c_p	The *Primary State* node
	C	The set of all concept nodes in an FCM
	$\mathcal{C}$	Partition of the original set of concept nodes C
	C_A	The set of all *Activity* nodes in an FCM-NAS
	C_N	The set of all *Need* nodes in an FCM-NAS
	C_S	The set of all *State* nodes in an FCM-NAS
	d_i	A value representing necessity of a *State* node s_i
	e'_i	A temporary variable for the calculation of necessity coefficient e_i of *Need* node c_i
	e_i	The necessity coefficient of *Need* node c_i
	f	Transformation function
	f_a	The transformation function for processing *Activity* nodes
	f_D	Transformation function for the *Delay* value
	f_P	Transformation function for the *Planned* value
	F	The final fitness function
	F_F	Fitness function based on the average success rate of satisfying needs (value of *Primary State* node)
	F_P	Fitness function based on completion of planned activities
	F_T	Fitness function based on portion of free time during a simulation (extra time which was not used for a planned activity or satisfying needs)
	g	Granularity parameter
	h_a	The number of alternatives in an AHP
	h_c	The number of criterion in an AHP
	i	List, array, or matrix iterator; positive integer
	j	List, array, or matrix iterator; positive integer
	k_n	An increment to shift necessity coefficients of *Need* nodes
	l	Constant affecting the curve of the priority functions
	m	Total number of activities planned during the simulation
	n	The number of concept nodes in an FCM
	n'	The number of concept nodes in a new smaller FCM enhanced by AHP
	n_A	The number of *Activity* nodes in an FCM-NAS
	n_N	The number of *Need* nodes in an FCM-NAS
	n_S	The number of *State* nodes in an FCM-NAS
	p	The position of *Primary State* node

R	The adjacency matrix containing all relations w_{ij}
t	Time step; integer number representing number of past iterations of an FCM
t_d	Time difference between the current time t and planned start u_i^S of activity u_i
t_{free}	Number of time steps when agent was neither doing any planned nor need-oriented activity
t_{max}	Number of past iterations of FCM after a simulation finished (the number of the last time step)
T_D	Time period after planned start u_i^S of activity u_i when the activity u_i is removed from the schedule
T_P	Time period before planned start u_i^S of activity u_i when the activity u_i is added to the schedule
u_i	The i-th planned activity in a list
u_i^E	The evaluation of progress of planned activity P_i
u_i^I	The importance of planned activity u_i
u_i^S	The planned start time of planned activity u_i
u_i^T	The tolerance of planned activity u_i
$\hat{u}_i^t$	The priority of planned activity u_i at time step t
v_i	The truth value of node c_i
v_i^t	The truth value of node c_i at time step t
v_p^t	The value of the *Primary State* node at time step t
V^t	A one-dimensional vector containing values of all concept nodes at time step t
w_{ij}	The value representing the influence of node c_i on node c_j
W_F	Weight of fitness function F_F
W_P	Weight of fitness function F_P
W_T	Weight of fitness function F_T

1 Introduction

Ambient intelligence and smart technologies are continuously spreading into more aspects of our lives. Their objective is to help, provide information, monitor, or secure. When researching and developing an intelligent system for a smart environment, few issues arise. As in any other system, testing needs to be done on the finished system but also during development and design. Usually, a system receives some testing data to test its behavior. However, ambient intelligence needs human interaction with the system which requires real-world implementation of the system and volunteers for long-term testing. To properly test the system, months are often required for the system to collect enough data, so it recognizes routine behavior fully and adjust to the person. Apparently, this solution is demanding on both time and resources. Software models of ambient intelligence were created to overcome this problem. A model of the intelligent system alone is quite straightforward since the only significant difference lies in a software implementation of sensors instead of physical ones. The challenge stands in the creation of a simulation of artificial people who are supposed to interact with the system and generate needed data.

The motivation behind this work originated from GAČR project DEPIES which focused on ambient intelligence. During this project, a model of a smart environment was developed. After the implementation of core components of the intelligent system and environment, the model was missing a mechanism that would provide human-like behavior. In attempts to create natural behavior, many approaches were examined, but their possibilities proved to be limited. This problem initiated the motivation to design a decision-making mechanism that could provide a simulation of natural behavior.

When simulating natural behavior of any living being, routine behavior covers a significant portion of its life. Routine is a purposeful behavior made up of simple goal-oriented actions, which acquire, learn and develop through repeated practice [1]. As such, proper routines enable predictable and efficient completion of frequent and repetitive tasks and activities without going through unnecessary

resource or time-consuming decision-making and planning every time it occurs. Therefore, along with general decision-making based on needs and the current situation, this work will include routines for the simulated persons in an agent-based model. An agent is an autonomous intelligent computer system capable of evaluation of the situation, decision-making and performing actions [2].

In the field of artificial intelligence, there are many various approaches to simulations of intelligent behavior or even the creation of an intelligent entity. It ranges from the most straightforward rule sets based on stimuli and corresponding reactions to soft computing with learning capabilities and the complex neural networks imitating some mechanisms of a biological brain. Different methods are suitable for various tasks as a neural network is inconvenient for simple repetitive tasks, and image recognition is nearly impossible to resolve with simple rule sets. Apparently, there is vast space between both approaches with a different balance of complexity and difficulty or performance.

The dissertation focuses on the method that utilizes matrices to process decision-making. It uses fuzzy cognitive maps enhanced to support autonomous agents and systems through classification of nodes into three classes: *Needs*, *Activities*, and *States* (FCM-NAS). This approach stands right in the middle between non-transparent complex structures with the ability to learn and easy-to-use simple rules. It takes advantages from both sides of the field. It allows creating complex structures and learning similar to neural networks while it keeps the human-readability and comfortable initial design. The other reason speaking in favor of this method is its scalability and adaptability which will be useful for implementing scenarios. Despite all these advantages, neither this approach nor any similar one was developed and used for human activity simulation before.

To verify the proposed design and prove its abilities, the dissertation provides experiments of this method in an ambient intelligence model. In this case, the purpose of the method is not controlling the system, but simulating a person (agent) moving and interacting within the environment covered by the system.

This approach might replace the need for real-world volunteer triggering hardware sensors with an agent triggering software sensors in the model. Obviously, there are many other application areas of human activity simulation like social and environmental simulations, the entertainment industry, or robotics; however, its implementation in ambient intelligence model could also help with detecting and analyzing agent's behavior through its interactions with sensors.

The structure of this work is as follows: The next section, Objectives, presents the aim and main goals of the dissertation. Then the section State of the Art introduces the basics of decision-making, particularly fuzzy cognitive maps, and genetic algorithms. It also summarizes projects with similar goals or means to one presented in this work. The fourth section, Solution, describes the design of the FCM-NAS and routine behavior as well as the implementation of the approach into an ambient intelligence model. The section Results presents the results of experiments conducted on the model regarding behavior, machine learning, and computational performance. Finally, the section Discussion compares the created model with the projects described in the State of the Art, and also returns to objectives, evaluates accomplished goals and achievements, and outlines the future direction of research and possible enhancements of this work.

2 Objectives

The objective of the dissertation is to design a decision-making method for agents which would combine following the schedule with the ability to react to unexpected situations at the same time. This approach requires the combination of two cooperating decision-making systems per single agent. The first system will manage a schedule created according to a scenario describing the agent's usual routine activities during a day. The second system will handle irregularities or fill meantime between scheduled tasks. There needs to be cooperation between both systems to accomplish all planned tasks while covering all agent's needs, random events or emergencies. The FCM-NAS method will be introduced and enhanced by schedule management.

The main objective of this work is to design a method for the simulation of human behavior which would be able to process decision-making based on both internal needs of agents and planned scenario. To accomplish and prove this objective, the following tasks will be done:

1. Review of application areas of human activity simulation.
2. Review of studies describing FCM and its modification with a focus on agent-based approaches.
3. Specification of the modified FCM method for human activity simulation.
4. Proposal of general methodological procedure of how to design and implement human activity simulation for selected application domains.
5. Implementation of the methodology into an ambient intelligence model, conducting experiments, and their evaluation.
6. Synthesis of results.

The proposed solution would be useful for simulations of any living creature following routines during its life. Potentially, it may be used in research to simulate complex behavior in various multi-agent models, or even add an immersive behavior of characters in the entertainment industry.

3 State of the Art

This section introduces all areas of research related to this dissertation and its objectives. It starts with a general insight at decision-making before it dives into details of the core method of this work - fuzzy cognitive maps, and used learning method - genetic algorithms. Then, this section presents current applications of the method along with models based on different approaches with similar goals.

3.1 Decision-making

The first and most important area of research required for human activity simulation is decision-making, especially related to autonomous agents. For agent-based models, artificial intelligence offers many methods to emulate intelligent behavior. Regarding structure and adaptability, these methods fall onto a scale between two opposite approaches. On one end, there is a robust narrowly focused approach with straightforward rule sets and hardcoded procedures. On the other end, there is soft computing; more general complex structures that are usually able to react even to unexpected situations (improvisation), but very hard to configure and often with not transparent decision-making. At the very end of this scale, there is the general artificial intelligence [3]. Neural networks are currently the closest method to this approach.

Methods closer to the first approach (simple, narrow) are the most commonly used for agent-based models for their simplicity of implementation and usually very specific objectives of individual agents. In the large scale models, the complex behavior tends to emerge from simple interactions of many agents. It is effortless to find examples of this approach: from example models included in different multi-agent platforms to show their possibilities to most of the models focused on emergent behavior. The well-known stigmergic mechanism is using many agents with simple rules observed in nature to accomplish complex path-finding problems [4].

However, even without emergence, a large enough set of simple rules can generate behavior which may prove to be very complex or very close to human decision-

making in a specific task. Recently updated artificial intelligence in classic strategic game Age of Empires 2 offers a great challenge for human players despite a very complex environment, a variety of possibilities, and limited structure of scripting rules. On almost 30 000 rows of code, scripters of artificial intelligence managed to implement most strategies of the best human players [5]. In [6], they are dealing with a similar problem. With limited tools, such procedures are very demanding to implement, while there still might be some logic holes left for human players to exploit. In some cases, human players could act intentionally a certain way to confuse algorithm and make it react advantageously for its opponents. To counter this, there are attempts to enrich such procedures with more complex structure with the ability to learn [7], which leans toward the soft computation approach to decision-making.

On the other side, in opposition to the first approach, there are neural networks. There is a lot of different ways to implement these, but the core idea lies in imitation of processes of the biological brain. Its general structure contains inputs and outputs with manually assigned meaning to connect it with surroundings. Between input and output nodes, layers of neurons create the logic of a neural network. Only machine learning [8] can assign a purpose to each neuron. The main advantages are flexibility and the potential resulting from its complexity and ability to learn. However, its complexity is concurrently the cause of its disadvantages. While its implementation is not difficult, it requires a lot of learning (proportional to its number and size of layers) before a network is usable for an intended objective [9]. The second problem of neural networks also arises from its complexity. The final product, neural network after learning, is not human readable – a black box. Even if a network is small and we can track which neurons were activated, single neurons often do not carry one simple distinguishable meaning.

A wide range of methods fits somewhere between these opposites with a more flexible structure, abilities of machine learning, or a wider range of perceived input than the first group, but still somehow limited in scope, focus, or size, keeping it usually human-readable. Methods like Markov decision process [10], Analytic

hierarchy (or network) process [11], Petri nets [12], or Fuzzy cognitive maps [13] are examples of the approaches between scripted procedures and black-box soft computing. These often use graph theory to represent concepts or states and their relations – causal dependencies, transition function or weights [14, 15].

As a special category of decision-making worth a mention, there are methods used specifically in multi-agent models. These approaches require more participants to utilize game theory and interaction models with auctions, negotiation, cooperation and so on [16]. However, this dissertation aims for individual-based decision-making. This aim does not limit the number of agents in a model – only the complexity of possible interactions among them.

3.1.1 Fuzzy Cognitive Maps

Fuzzy cognitive maps (FCMs) [13] are dynamic systems of concept nodes with a network of causal relations. They originated from cognitive/causal maps, which are able to model causality and change of the concepts, by enhancing them with some principles of fuzzy logic [17]. An FCM is described by a collection of n concept nodes C, relations w_{ij} between these nodes and a function used to adjust values of nodes. Figure 1 shows graph representation of an example FCM with five concept nodes. A designed causal map with evaluated oriented edges (relations) is then transformed into a square matrix. The adjacency matrix R in (1) is a general matrix representation of an FCM as the matrix of relations between n concept nodes [13].

$$R = \begin{pmatrix} w_{11} & \cdots & w_{1n} \\ \vdots & \ddots & \vdots \\ w_{n1} & \cdots & w_{nn} \end{pmatrix} \tag{1}$$

where n is the number of concept nodes. Each value w_{ij} represents the influence of node c_i on node c_j. Throughout this work iterators $i, j \in \mathbb{N}$. Values w_{ij} may be any real number between −1 (strong negative causality) and 1 (strong positive causality):

$$\forall w_{ij}: w_{ij} \in [-1, 1] \tag{2}$$

If $w_{ij} = 0$, then node c_i has no direct influence on node c_j. If $w_{ij} > 0$, then a high value of node c_i raises the level of node c_j; if $w_{ij} < 0$, then a high level of node c_i lowers the value of node c_j.

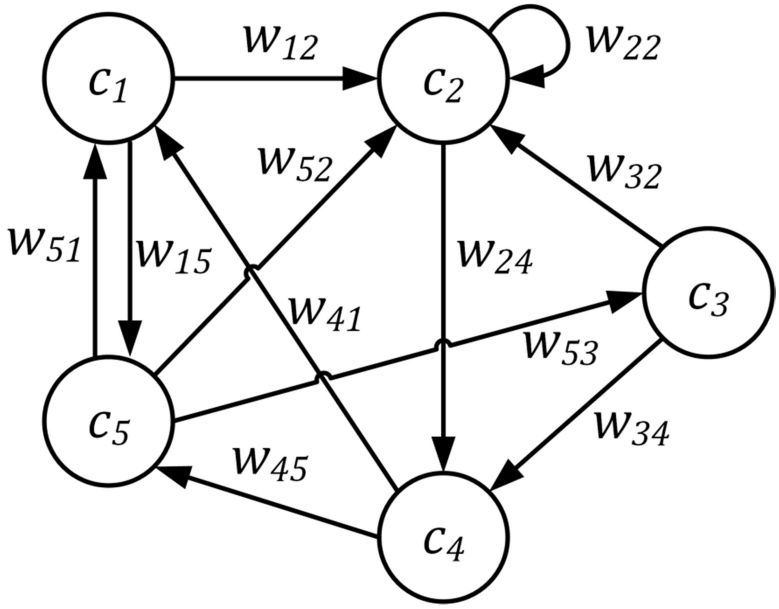

Figure 1: Example of FCM with five concept nodes and relations among them. (source: [18])

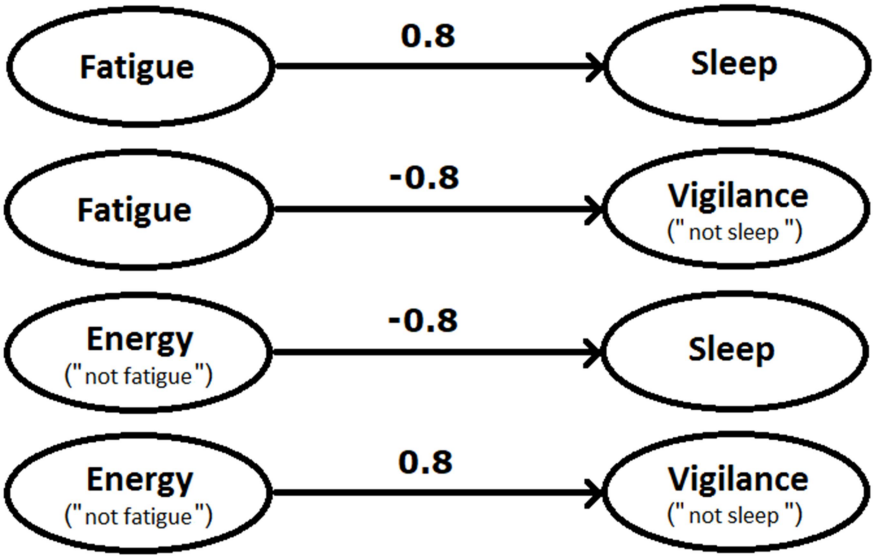

Figure 2: Equivalent relations. (source: author)

In addition to the matrix R, an FCM needs the truth values of nodes. Let v_i denote the truth value of node c_i. This value represents how much is its corresponding concept node active or strong. For example, an agent with the high level of the value corresponding to concept node *Fatigue* means that the agent is tired. Names of nodes determine understanding and design of concept nodes and their relations. When designing causal relations, each concept is interchangeable with its opposite counterpart. If all relations of the inverted concept are inverted as well, then this

change does not influence the logic of the FCM. Figure 2 shows four different relations which have the same meaning. In this example, the truth value of nodes *Fatigue* and its counterpart *Energy* would be inverted to each other: the low level of *Energy* would be equal to the high level of *Fatigue*.

Since this work uses an FCM in a dynamic environment, it will be changing through iterations (often called time steps in simulations). Therefore, the variable of time is necessary for calculations. Let denote time steps with *t*, which is a positive whole number starting at zero as FCMs calculates values in discrete time scale and is only able to advance in time. Changing values of nodes are then specified as v_i^t, which is the value of concept node c_i at time *t*. As a truth value, v_i^t is always a real number ranging between 0 (definitely not true) and 1 (definitely true). Even if the value exceeds these limits after computation, it needs to be immediately reduced to the valid range:

$$\forall v_i^t : v_i^t \in [0, 1] \tag{3}$$

These values form a vector V^t; a one-dimensional array also valid only for the time step *t*:

$$V^t = \begin{pmatrix} v_1^t \\ v_2^t \\ \vdots \\ v_n^t \end{pmatrix} \tag{4}$$

The sizes of all components are constant. The vector V^t is updated at every time step, and the content of *R* is static. (5) shows the computation of iteration at time step t based on the original definition model, while (6) represents a version in which the product of the multiplication is added to the previous value, known as the incremental model [19].

$$V^t = f\left(R \cdot V^{(t-1)}\right) \tag{5}$$

$$V^t = f\left(V^{(t-1)} + R \cdot V^{(t-1)}\right) \tag{6}$$

In both equations, function f represents a transformation of values. It is a real function of a real variable. Many diverse types of nonlinear functions can be used (e.g. sigmoid, hyperbolic, step and others). The primary task for this function is to keep the values within the interval from zero to one. The model described in this

dissertation mostly uses a simple linear transformation unless a value exceeds this interval:

$$f(x) = \begin{cases} 0 & for\ x < 0 \\ x & for\ x \in [0,1] \\ 1 & for\ x > 1 \end{cases} \qquad where\ x \in \mathbb{R} \tag{7}$$

The original FCMs handle all concept nodes in the same way, which causes a few issues in systems that contain elements of a different type. For example, if an FCM considers a few inputs and then decides whether to trigger an action, then a trigger node has to be processed differently than the other nodes. Besides choosing a sufficient level of the node to trigger an action, it usually has to recognize only two states of node: an action is performed or not. Such differences in the processing and interpretation of nodes make FCMs confusing and less modular. This is the reason to use a modification which is built to handle such situations.

FCMs are a powerful tool in modeling various dynamic phenomena and systems which consist of many dependent variables in a complex structure. Usually, an FCM is designed for a single narrow task as a support tool for analysis [20, 21], decision-making [22, 23], predictions [24], and various tasks in social sciences [25, 26]. These FCMs aim for specific values after a few iterations or search convergence and stable state of the system. However, this work regards FCMs as an artificial intelligence method for systems in dynamic environments. Autonomous systems have to handle a much wider range of situations and require a different approach to the FCMs. FCMs with a few enhancements prove to be strong artificial intelligence method even for autonomous agents [27].

In the field of intelligent systems, a few projects aiming for interaction with human use an FCM to simulate emotions. For educational purposes, a virtual pet was designed with an FCM that ensures believable reactions to user's actions [28]. Also, an ambient intelligence system enriched its user interface with FCM-based emotions to provide additional comfort and naturally respond to the presence of users [29].

In many projects, FCMs are a core component of a system or agent in a model. For instance, it is used by monitoring systems, where an FCM assesses risk in critical situations [30], a situation awareness model for infantry platoon leaders [31], or even landing site selection for planetary exploration [32].

Several articles cover the possibilities of learning FCMs. Systems in a dynamic environment have to deal with continuous changes; therefore, unsupervised training of the FCM or even adjusting it during runtime are desirable features in such projects. Genetic algorithms and Hebbian algorithms are commonly used as a base for various extensions dealing with this issue [19, 33].

Dynamic fuzzy cognitive maps (DFCMs) are one of the extensions focusing on the learning of FCMs. It is based on the random neural model, which was designed to react to random events by modifying causal relations. The main feature of this approach is the ability to change an FCM during runtime, which enables adaptation and switching different behaviors at runtime. This extension was used as a supervision system [34] and a navigation system of a robot [27, 35].

FCMs also provide useful services in multi-agent systems. There are two areas suitable for this method: as a supervisor monitoring and controlling an environment [34, 36] or an artificial intelligence controlling individual agents. The second area is not commonly used because FCMs in their original form are not convenient for this purpose nor modifications that could support this role are not well known yet. However, many of the previously mentioned works had to touch the problem of FCMs for autonomous systems at least slightly. Moreover, models like the EcoSim model, which will be described in the following paragraphs, already process agents with an FCM.

EcoSim [37-39], an artificial life model, uses an FCM to process the agents' behavior, and its usage is close to the method in this work. The simulation includes the evolution of values in the FCM, which enables adaptation of behavior. Agents have available just a set of basic actions, from which they choose the optimal one for a current situation. The development and behavior of the population emerge

from interactions between agents. EcoSim contains two types of species (predator and prey), which both evolve to raise their chances against the rival species.

The decision-making of prey and predator species differs in its actions and observed properties. As seen in Figure 3, it combines high-level and low-level control at the single map which generates a very complex FCM (26 concepts). The width of the arrow represents the strength of a relation. Blue color means positive relation; red negative. The structure of the FCM forms layers determining the direction of relations, which resembles the structure of a neural network or the framework of FCMs for autonomous agents presented later in this work. The first (input) layer (left side in Figure 3) contains pairs of inputs. Each pair has a separate node for low and high presence of the corresponding quality. The middle layer connects inputs with the last layer (right side in Figure 3), serving as outputs. In this case, levels of output nodes decide the action that agent will choose in reaction to given inputs.

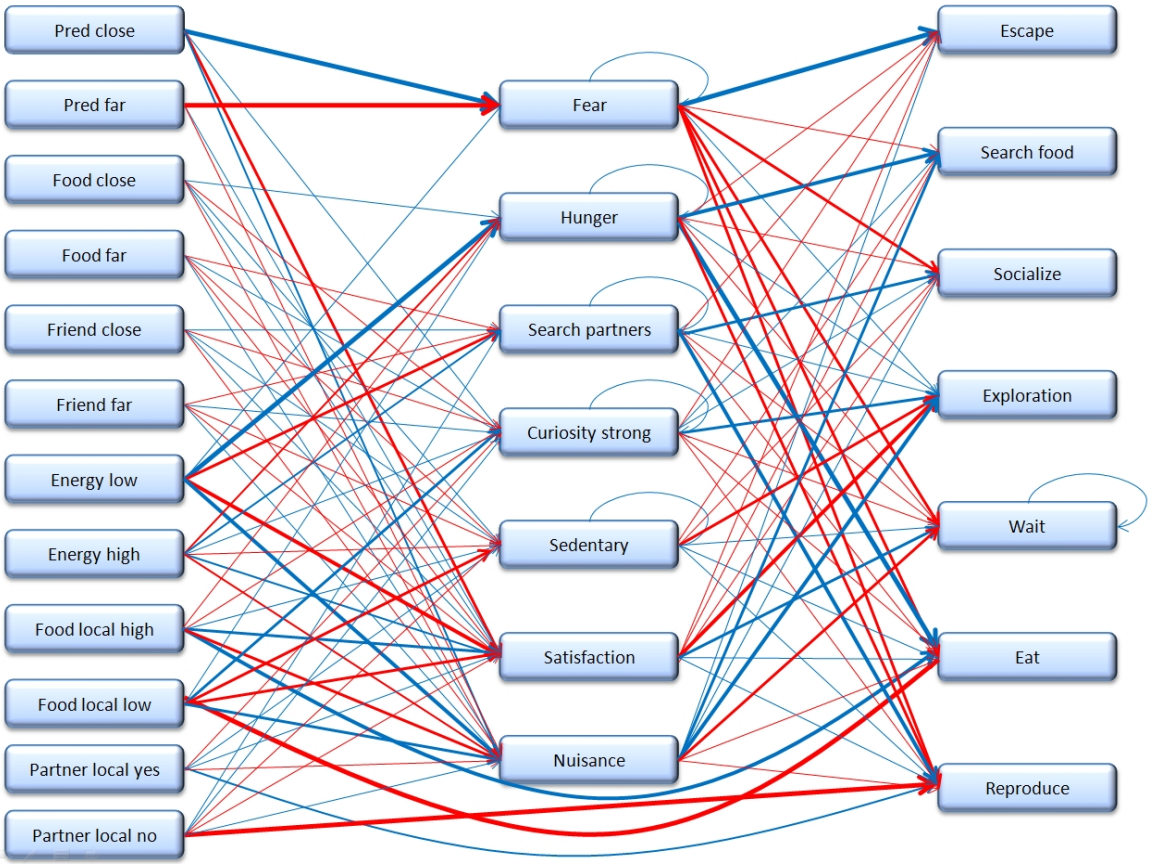

Figure 3: Concepts and their relations of FCM in the EcoSim model. (source: [38])

Despite the size of the FCM, the behavior is focused only on the reproduction and management of food and energy. Generated patterns are visually close to the cell-based models. According to [40], EcoSim aims to observe the emergence and evolution of the population rather than realistically simulate the behavior of individual living creatures.

3.1.2 Genetic Algorithms

Using decision-making requires more than the choice of the right method suited for a task and its implementation. Its design and configuration are essential to ensure consistent and reasonable results. For the hardcoded procedures, a designer has to set everything by hand and be aware of every possible input. However, soft computing methods usually offer possibilities for machine learning. These possibilities depend on the selected method. Some of the methods are available for initial manual configuration, and then an automatic learning process optimizes the result. However, neural networks, for instance, are not suited for by-hand design at all, and therefore they require machine learning before they are ready to operate.

Generally, machine learning [41] includes methods for supervised, unsupervised, and reinforcement learning. Supervised learning aims at classification and regression; its objective is to pick the correct output for input sets. Backpropagation is one of its tools to correct wrong answers by adjusting internal weights. After enough learning data, decision-making can recognize and classify even inputs that it did not encounter yet. The objective of the unsupervised alternative is similar to the previous one with the exception of creating its own output classes (clusters). Mentioned learning algorithms focus on handling big data structures, and they can be useful for agent-based models as well; however, it is not suited for training decision-making part of agents. For instance, the previously mentioned EcoSim model uses cluster analysis to create species and sort agents into groups corresponding to their traits [42]. Nevertheless, previous adjustments of these traits were executed by reinforcement learning – specifically by genetic algorithms [43].

The reinforcement learning algorithms are the most relevant for a configuration of agents' decision-making. The base concept of this type of learning lies in rewarding the subject after good (intended) behavior or penalizing it after bad behavior. It also differs by not tying specific inputs to a specific output and permits sub-optimal solutions. As the agent goes through reinforcement learning, inconvenient connections drop and useful ones are reinforced [44, 45].

The Q-learning is an example of reinforced learning, where the subject chooses an action from known selection based on expected reward. After each selection, it evaluates its decision and corrects expected rewards for the future. Its goal is to explore useful states to go through and learn a policy that brings the highest cumulative reward [46]. In cases where every action can be evaluated every step, this method converges very fast to the optimal solution. For complex environments where actions may have delayed or unclear consequences, machine learning offers more general and universal method – genetic algorithms.

After Charles Darwin published principles of evolution in 1859, his ideas were extracted from biology scope and generalized into the algorithm that is able to solve a wide range of problems. The popularity of this method is recently rising thanks to significant increases in available computational power - along with more frequent use of soft computing to solve computationally expensive problems [47].

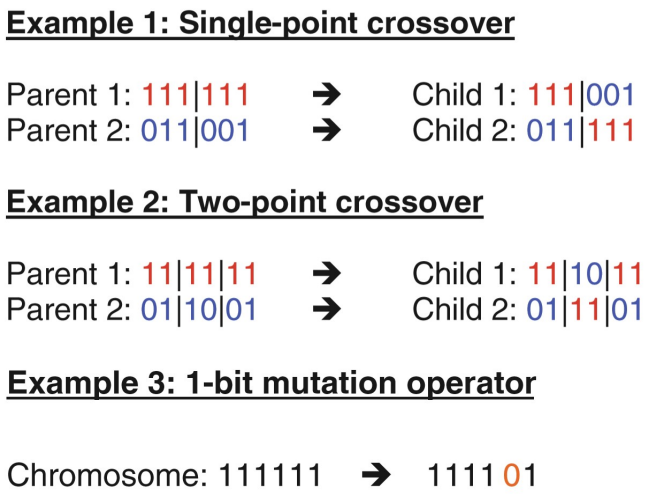

Figure 4: Example operations with binary genotypes. (source: [48])

Genetic algorithms imitate natural phenomena of natural selection, breeding, mutation, and apply it to the gene pool of a population. Each individual from the population is characterized by own genotype (also sometimes called a chromosome) – a string of values that represents a solution to an examined problem. Type of these values can be binary, truth values, integers or any symbols which are then translated by the program into the part of a solution. An important requirement of genetic algorithms is the ability to evaluate each solution on a continuous scale allowing ordering and comparing genotypes. After the evaluation, it proceeds with selection and genotype operations to create a new population (see Figure 4). Following steps explain standard procedures of genetic algorithms [49]:

1. Create a starting population
2. Evaluate each genotype using a fitness function
3. Select pairs of genotypes using a selection mechanism based on fitness values
4. Apply crossover for each selected pair of genotypes
5. Replace old population with newly created genotypes
6. If a target condition is met, then end this procedure; otherwise return to step 2

In [50], the authors evaluated different crossover approaches; particularly the number and position of crossover sites. Crossover sites are points in a child genotype, where the inherited values switch their source (from one parent genotype to another and back). Besides the basic ones seen in Figure 4, they tested multi-point, uniform, and variable-to-variable crossovers. Their results showed that approaches with fewer crossover sites converge to better results faster. The authors recommended single-point crossover as the best choice for general use. The worst approach in this regard was uniform crossover, which deals with every gene on its own leading to random number of crossover sites. However, they

acknowledged that the uniform crossover shows the best exploration ability, which is the reason for its slow convergence.

Many modifications appeared since the first attempts to apply evolutionary principles to a computational problem. Some approaches try to approximate the source of this idea (nature) better. For instance, diploid genetic algorithms appropriate more detailed evolutionary principles of mammals. Instead of one chromosome, each individual has a pair of them – one from each parent. The dominance scheme then decides which specific genes will be expressed into phenotype (actual traits of an individual) [48]. Another interesting idea is to enhance genetic algorithms with the concept of frozen evolution – a natural phenomenon, which possibly has a negative effect in real evolution by limiting further development [51], but its principles could help the learning process with its convergence towards the optimal solution. The application of genetic algorithms into a virtual environment allows for a great variety of possibilities to adapt the evolutionary process to the specific needs of an examined problem and offers tools that are unreachable in the area of biological evolution.

3.2 Human Activity Simulation

Human-like behavior differs from the behavior of a general perception of artificial intelligence. The main difference points to decision-making and its cost. Generally, human decision-making has unlimited alternatives in continuous space and time of the real world while it is limited by the capacity of brain and time. This environment forced decision-making mechanisms to develop a strategy that is not going through all possible alternatives and selects an alternative, which is the best of the explored ones, even if it might not be the optimal alternative. This theory, bounded rationality, started to develop along with rational and game theory to better explain and describe human-like decision-making [16]. The natural interest to cost-efficient decision-making also leads to developing patterns in behavior - routines.

Routines are defined by frequent actions a subject performs in different situations [52]. They are purposeful behavior acquired through repeated practice of goal-

oriented activities. Routines enable predictable and efficient completion of frequent and repetitive tasks and activities. It explains behavioral patterns like the sleeping schedule of a person or even low-level tasks – learned procedures to handle specific situations. Routines, like most other kinds of human behaviors, are not fixed but may vary and adapt based on feedback and the current situation [1].

Description of routines, scenario, has many various approaches. Usually, it is represented by a sequence of activities optionally tied to specific times, when activities are supposed to start. With a higher level of detail, those activities might consist of a series of simple short tasks. For example, when an agent is performing activity *cook dinner*, it starts a sequence of elemental operations, which might trigger various sensors in the environment, like open/close a drawer, move to stove, turn on/off the stove, and so on [53]. There are several possible representations of scenarios: trees, Gantt charts, tables, flow diagrams, or statecharts [54-56]. Obviously, the graphical representations are aimed to be more human-readable. On the other hand, there are machine-readable formats that are more suitable for computer processing - tables or XML.

According to [57], human behavior can be represented in many ways regarding the level of abstraction (see Figure 5). At the highest level of abstraction, a model eliminates human behavior entirely by simplification. In the middle, models using system dynamics represent humans as flows to simulate pedestrians and drivers or total quality management. This work will be focused on the individual approach, which has the lowest level of abstraction by the mentioned study. However, there are a lot of possibilities and levels of abstraction even within this individualistic approach concerning the design of a simulation.

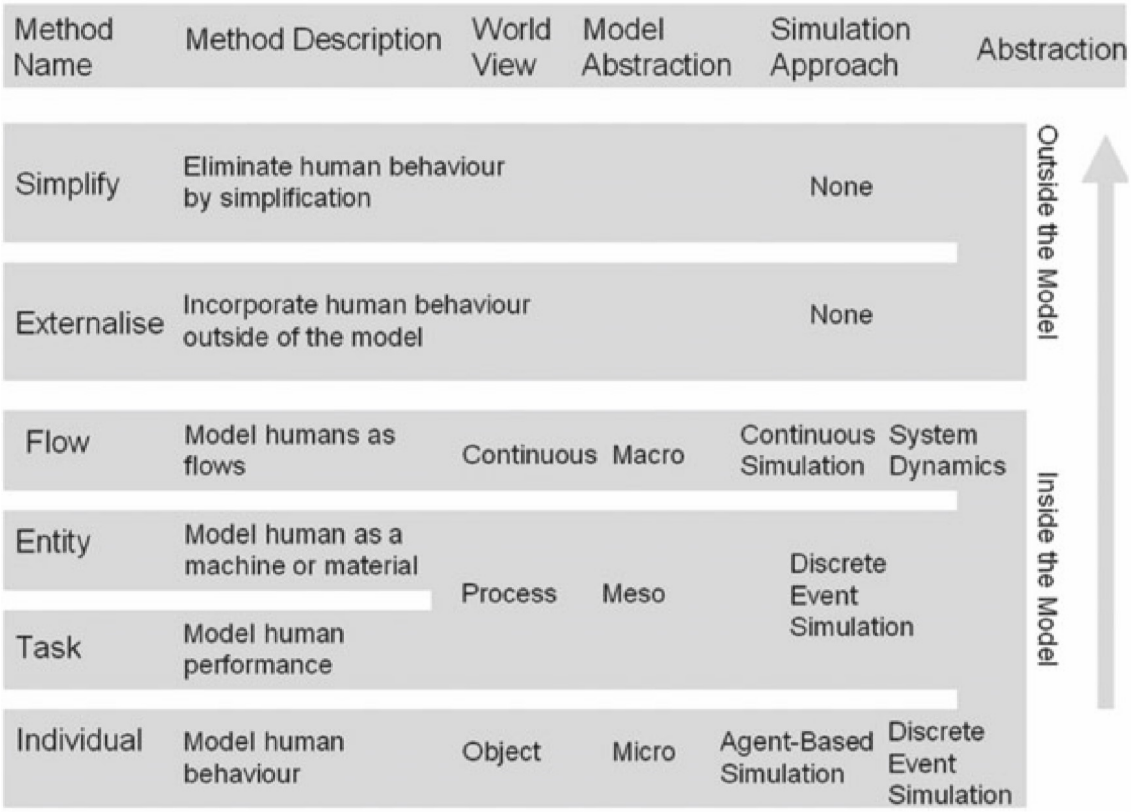

Figure 5: Methods of modeling human behavior in a simulation study. (source: [57])

There are many application areas where simulated human-like agents are useful. Most of them have a narrow focus on a single area, a simple task, and a short span of time. For example, PlayGA [58] system models human movement in an environment with various structures to simulate geometric affordance. This system might be helpful for the architectural design process to evaluate design alternatives. Also, simplified human cyber activity simulation uses a Markov Decision Process to model cybersecurity threats emerging from human errors [59]. Many studies cover the topic of human simulation under extreme conditions of various emergencies. These models explore building fires, crashes, natural disasters, toxic gas releases, radiological releases, and others. During such events, agents' decision-making is limited and has straightforward goals – escape, help other agents or just survive. Some models simulate psychological stress or panic, which causes agents to act ineffectively. With a higher number of agents in close vicinity, crowd behavior also impacts the final results of simulations [60].

The less frequent but still important part of this area explores human behavior in a more general scope. Instead of simulating a human in a specific situation or just narrow indirect effects of human activity, this work focuses on the simulation of human behavior during its natural existence within a more extended period of time. Only then higher-level routine behavior takes place in the virtual life of an agent. Besides ambient intelligence models, which will be described later in detail, reactive decision-making mechanism with routines could improve artificial life models, society models, or ecologic simulations as routine behavior is definitely not exclusive to the humankind only.

Of course, the demand for human simulation is not limited solely to research purposes. In the growing entertainment industry, computer games are getting much more complex, and the audience's requirements rise. There is a high demand for believable human simulation for non-playable characters (NPCs). The living world and its inhabitants immerse the player into the game and improve the overall quality of the game.

Figure 6: A day plan of an NPC in the role of a common peasant in KCD. (source: [61])

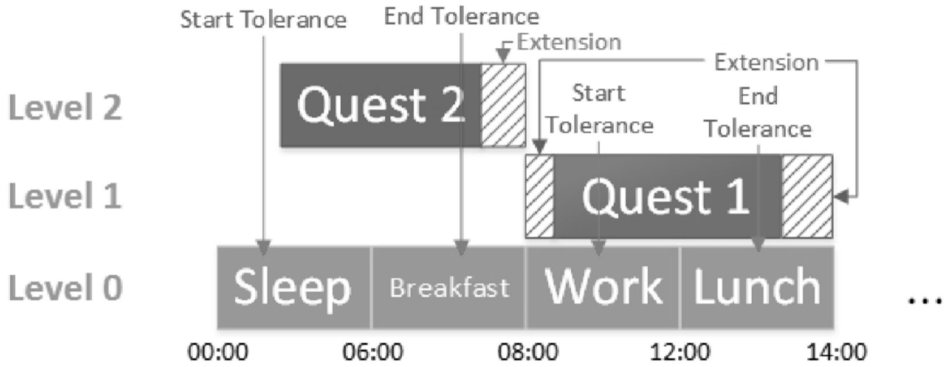

Figure 7: Example of priority lanes in the decision-making of simulated persons in the game Kingdom Come: Deliverance. (source: [61])

For example, one of the biggest Czech computer games, Kingdom Come: Deliverance (KCD), which was very successful on the global market, also uses similar mechanics. Reviewers around the globe often praised its living world as one of the positives which made this game stand out from others [62]. The doctoral thesis at [61] describes in detail believable decision-making implemented in the game. In KCD, NPCs' behavior contains a few sub-systems for different conditions. The higher-level decision-making mechanism selects the general context which NPC should focus on currently. Then, it decomposes a selected goal into tasks and to further lower-level actions that lead to the satisfaction of that goal. The standard behavior of any NPC is straight-forward schedule for the whole day as shown in Figure 6.

In the context of computer games, human simulation has specific requirements regarding interactions with a player (or players). Meaning game designers need to consider all possible interactions either direct ones with the player or indirect ones through changes in the environment initiated by the player. NPCs in KCD operate in modes depending on the situation (for instance combat, regular or quest mode). Figure 7 shows a collision of quest-related activities with regular ones. Schedules with higher priorities overwrite the daily routine simulation. For more natural behavior, NPC can skip activities when there is not enough time to finish it (due to a quest for example). Hence, its internal needs are not strictly managed and skipped activities are not substituted; the visual quality is the main concern in this application area.

3.3 Ambient Intelligence Simulators

Ambient intelligence and smart environment systems aim to observe user interactions with the environment, gather data, learn from it and use it to help its users in daily life without unnecessary intrusion or direct user-system interaction. To research these systems, many experiments need to be prepared and performed. However, conducting them in a real environment is expensive due to space and hardware needed and testing them is hugely ineffective, since it takes real time to collect data from actual usage of the system. This problem motivated the development of various simulation approaches for modeling an intelligent environment to test it virtually [63-66]. A simulation brings advantages over real-world implementation: it is cheaper and capable to generate a record log worth of months of system use in a few seconds.

These simulations need to implement agents with natural behavior to acquire valid results of experiments [53, 67]. They mostly use scenarios with none or random variations omitting less likely or extreme situations. This type of behavior is a sequence of activities that might consist of more elemental actions [67, 68]. Models often apply small random variations to imitate real behavior with small irregularities and break the exact match with a planned schedule from the scenario. The other approach to human activity simulations is based on the agent's parameters which do not respect any daily cycles or regularly repeated behavior. These approaches cannot deliver believable human behavior simulation within more prolonged periods of time on their own. None of the current works offer the combination of both principles to create need-oriented behavior along with routine and planned activities.

Table 1: Analysis of smart home simulation tools. (source: [69])

Tool/Author(s)	Date	Open-Source	3D	Cross-Platform	Approach	Focus	Multi-Inhabitants	Fast-Forwarding
OpenSHS	2017	Yes	Yes	Yes	Hybrid	Dataset generation	Partially	Yes
Park et al.	2015	No	Yes	Yes	Interactive	Visualisation	No	Yes
PerSim 3D	2015	No	Yes	Yes	Model-based	Dataset generation	No	Not applicable
IE Sim extended	2015	No	No	No	Hybrid	Dataset generation	No	Yes
IE Sim	2014	No	No	No	Interactive	Dataset generation	No	No
Kormányos et al.	2013	No	No	No	Model-based	Visualisation	No	Not applicable
Ariani et al.	2013	No	No	No	Interactive	Dataset generation	Yes	No
Fu et al.	2011	No	No	Yes	Interactive	Visualisation	Yes	No
Jahromi et al.	2011	No	No	No	Model-based	Visualisation	No	Not applicable
Buchmayr et al.	2011	No	No	No	Interactive	Dataset generation	No	No
SimCon	2010	No	Yes	Yes	Interactive	Dataset generation	No	No
YAMAMOTO	2010	No	Yes	Not reported	Interactive	Visualisation	No	No
SIMACT	2010	Yes	Yes	Yes	Model-based	Visualisation	No	Not applicable
Poland et al.	2009	No	Yes	Yes	Interactive	Dataset generation	No	No
ISS	2009	No	No	No	Interactive	Visualisation	Yes	No
DiaSim	2009	No	No	Yes	Model-based	Visualisation	No	Not applicable
V-PlaceSims	2008	No	Yes	No	Interactive	Visualisation	Yes	No
Armac et al.	2007	Not reported	No	Not reported	Interactive	Visualisation	Yes	No
CASS	2007	No	No	No	Model-based	Visualisation	Yes	Not applicable
Krzyska et al.	2006	No	No	Yes	Interactive	Dataset generation	Yes	No
CAST	2006	No	No	No	Model-based	Visualisation	No	Not applicable
UbiREAL	2006	No	No	Yes	Interactive	Visualisation	Yes	No
TATUS	2005	No	Yes	Not reported	Interactive	Visualisation	Yes	No
UbiWise	2002	Yes	Yes	Yes	Interactive	Visualisation	Yes	No

In [69], the authors of the OpenSHS model presented an overview of available tools for smart home simulation (see Table 1). Besides previously mentioned approaches to human simulation, there is a large group of tools which completely avoid autonomous human simulation. Instead of an artificial decision-making method, interactive approaches use manual control of a virtual person by a user. Generally, controlling an avatar by a human participant may capture some specific interesting scenarios that most of the autonomous agents could not offer; however, it is not able to generate large datasets. In comparison with the expensive real-world testing of ambient intelligence, this approach will reduce only hardware cost while time costs usually stay the same. Also, generated dataset arguably cannot reach the precision of real-world testing. Since the main objective of this work is an autonomous human simulation, only models which at least partially offer autonomous behavior will be examined.

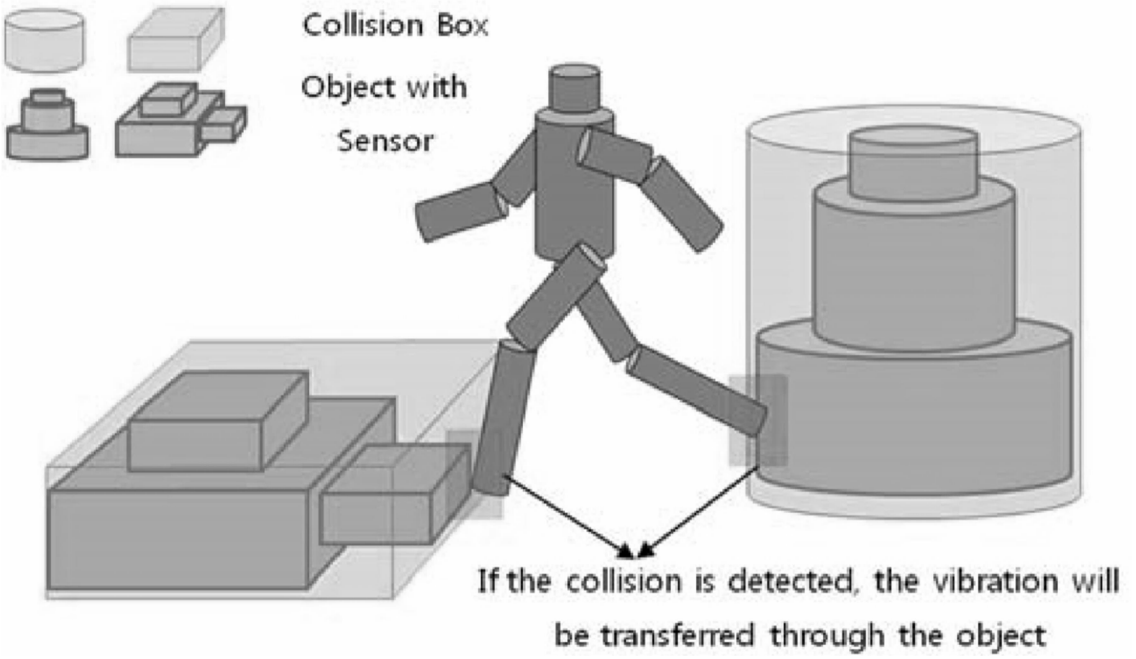

Figure 8: Vibration collision detection model in PerSim3D. (source: [68])

3.3.1 PerSim

PerSim [70] is an event-driven simulator for human activities in smart environments. It focuses on the interaction of a person with sensors and objects in an intelligent environment. Its objective is activity recognition of agents based on sensory activation. Key output data measured in the model are the order in which different sensors were activated and its duration. The agent's behavior is a straightforward fixed sequence designed by hand. The agent loads its behavior from an activity buffer. When it is empty, the simulation ends.

In the most advanced version of this simulator, PerSim 3D [71], users can place sensors in 3D space and watch an agent's behavior within a three-dimensional environment. It has a higher level of detail regarding the physical interactions of an agent with sensors. As shown in Figure 8, its transfer into the 3D environment and the intended detail of interactions require realistic animations, which are then used for precise collision detection of the agent with collision boxes of sensors in the environment. This mechanism simulates the propagation of vibrations caused by the agent's movement or interactions with devices within the environment.

To evaluate the realism of animations, online users rated several animations from the simulation, which they considered sufficiently realistic in the end. Considering decision-making and behavior, this version also replaces the event-driven model with context-driven one. This change adds another layer of behavior above activities, where each context contains one or more activities. Context loops manage the selection, order, and performance of its activities. However, the activities/contexts are defined by a user only - similarly to the original PerSim. Both versions of this simulator aim specifically for realistic sensory detection; therefore, it is designed for related but still different goals of ambient intelligence than this work.

3.3.2 Home Sensor Simulator

The Home Sensor Simulator (HSS) proposed in [53] aims to generate sensor data for ambient intelligence models by simulation of the daily activities of a single person. Regarding sensors, this application is less detailed than the previously presented PerSim3D. However, it uses a very similar hierarchical model of activities (see Figure 9) to break top-level activities (called contexts in the previous work) into ordered lower-level activities, which are then performed by completing corresponding elemental actions. HSS does not use schedules; instead, it makes decisions based on priority. As shown in Figure 10, priority functions consider the environment, possibly ongoing activities, and the agent's internal variables representing its attributes and preferences.

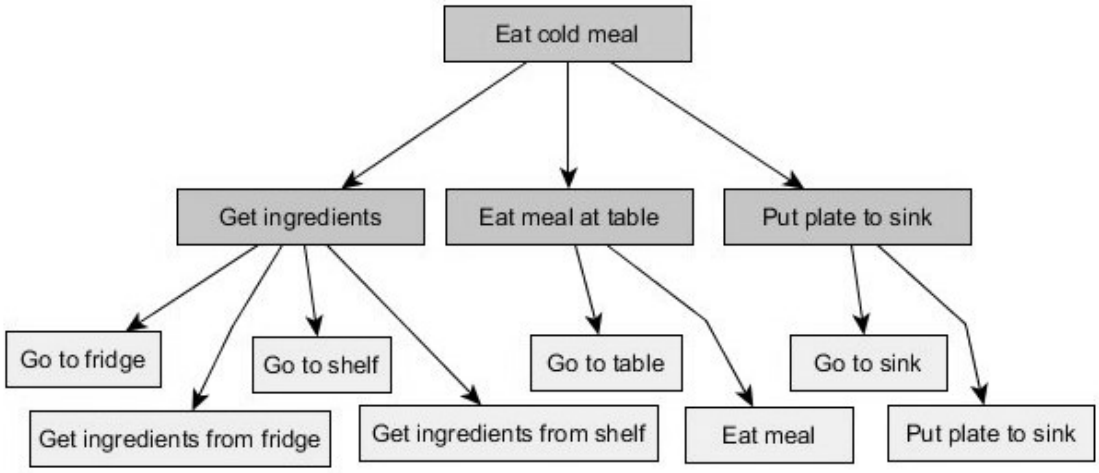

Figure 9: Three levels of actions. (source: [53])

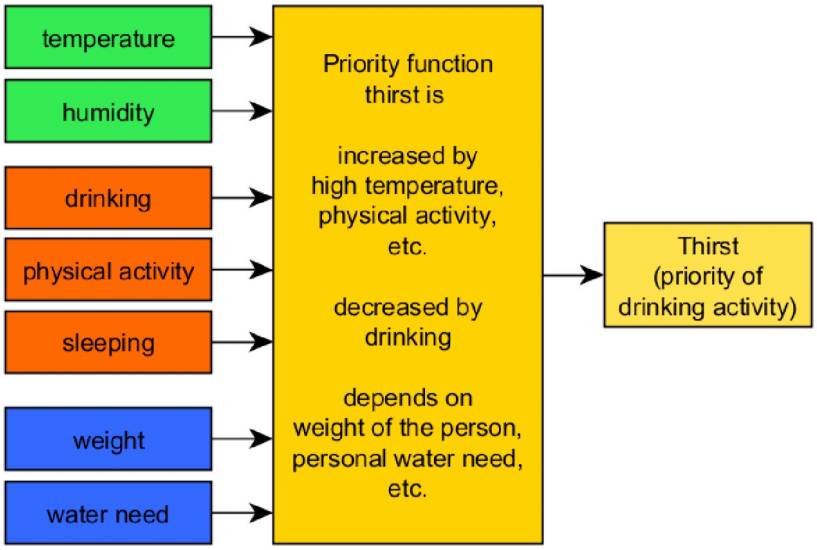

Figure 10: Priority function depending on the environment (green), the activity that is currently performed (orange), and the character of the human (blue). (source: [53])

When the priority of activity reaches a certain threshold, it triggers its start. The agent is capable of interrupting ongoing activity if a more pressing need appears (see Figure 11). For that matter, the new activity has to reach the priority level of the current activity with the additional increase (represented by the dashed line) corresponding to the agent's will to proceed and finish the ongoing activity before moving to anything else.

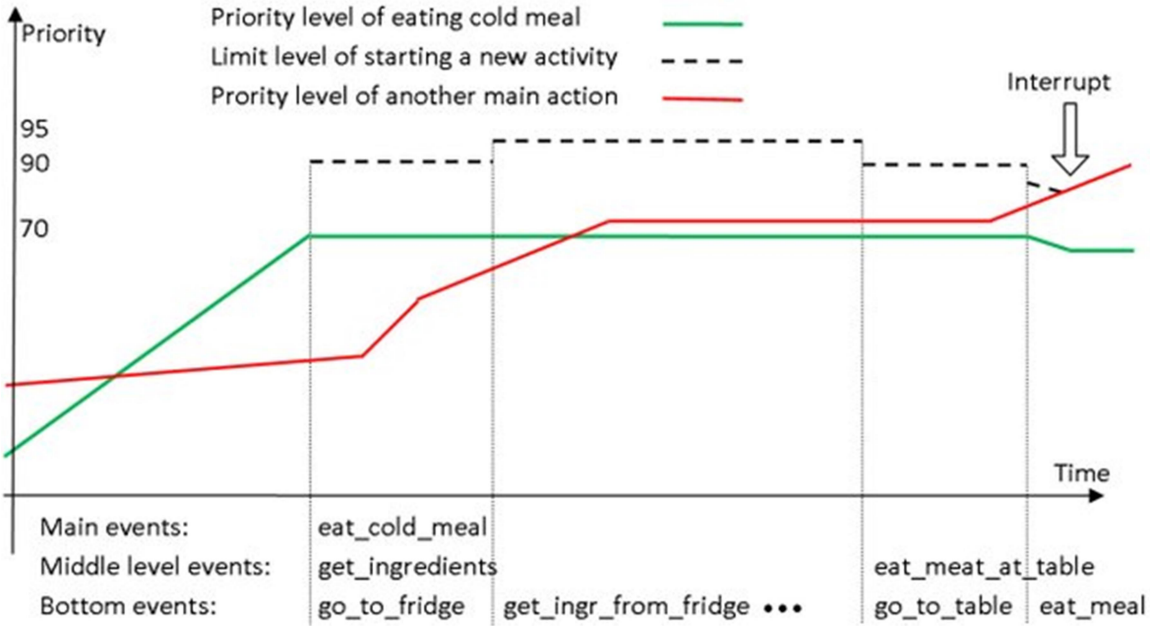

Figure 11: Example of priority levels of activities developing over model time. (source: [53])

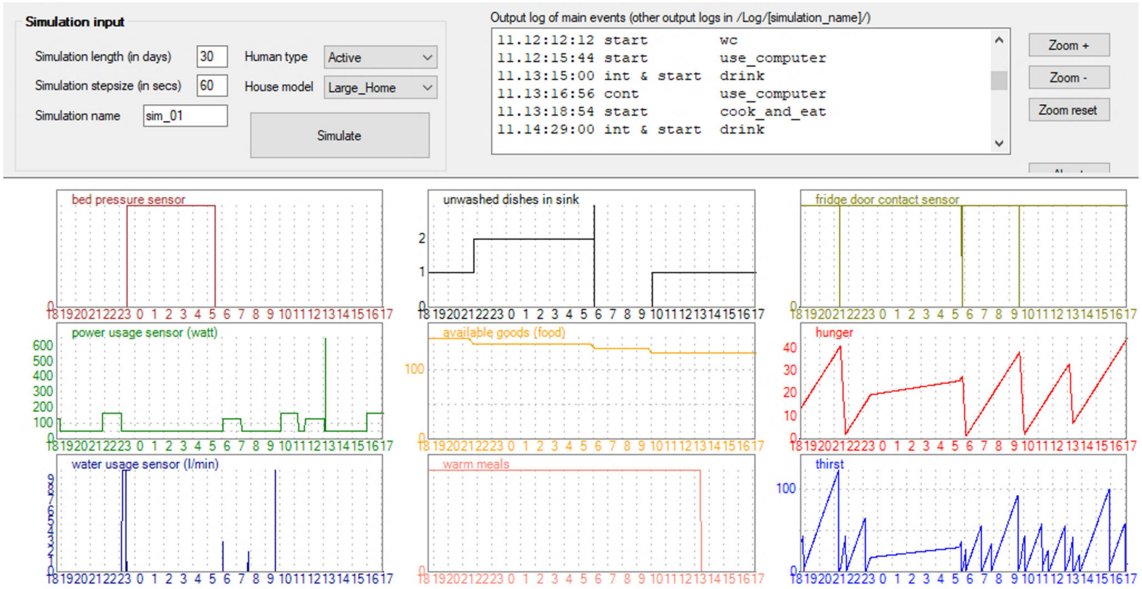

Figure 12: Screenshot of the interface of the Home Sensor Simulator. (source: [72])

HSS offers a simple interface and tools to design the environment and the agent's growth of needs. It is available for download at [72]. Provided output plots show activation of sensors, agent's needs or utility usage during a simulated time (see Figure 12). The record outputted by SSH shows a time when each activity started or possibly was interrupted. Although the presented behavior is flexible, can adjust to the agent's needs, and even handle situations like a lack of food in a fridge, HSS clearly lacks any patterns reminiscent of routines. The regular sleeping cycle is the only time-triggered routine in the model.

3.3.3 MASSHA

MASSHA simulator [67] is capable of processing more agents at once. Agents start a day with TODO and DONE lists: lists with tasks they need to accomplish or have already finished that day. TODO list is partially defined by a user but dynamically expanded by events in the environment. For example, the *preparing_dinner* activity generates a new *washing_dishes* activity and adds it to the TODO list. An agent is trying to complete all or at least the most important activities from the list. At the end of the day, lists reset. Agents' decision-making is responsible for the selection of an activity from the TODO list. This process is based on the priority of activities, agent's skill parameters, or roulette wheel mechanism. To compare priorities, each activity holds several parameters defining its duration and interactions between

an agent and environment. Table 2 lists all these parameters used in the model. Because the model expects some uncertainty and variability, two parameters are defining the start time and two the end time. The first set, desirable time, indicate the optimal interval to perform the activity. The decision-making process attempts to plan the activity within this interval. When this is not possible, it will try to perform the activity later. Then there is the second set of parameters representing the mandatory start and end time. If the agent exceeds these timestamps, the activity is considered failed.

Similarly to some previously mentioned models, MASSHA also decomposes activities into a structure, where the high-level objective is a sequence of low-level actions. It aims at very precise detection of lower-level human interactions with the environment as it collects data from every drawer, light switch, or cabinet in the environment. Duration and frequency of sensor usage were compared to the data collected from real smart environments which observed three different persons showing a very high percentage of similarity. At [67], authors recognize that the behavior in MASSHA heavily relies on the TODO list and suggest the future direction of the model could test some soft-computing methods like Markov models or neural networks.

Table 2: Definition of activity parameters in MASSHA model. (source: [67])

Name	Description
Type	Alphanumerical code that identifies the activity type.
Attendance	Attribute for describing whether a person has to be physically present during the execution of the activity. An attended activity will monopolise the attention of the person until it finishes (i.e. *prepare dinner*), whereas an unattended activity will allow the person to just initiate it and carry on with other activities (i.e. *watch TV*). This attribute is key in enabling a concurrent behaviour among people and a more realistic representation of an intelligent environment where the simultaneous activation of several sensors may take place.
Estimated duration	Amount of time the activity will take, according to what the person estimates, even though there may be some deviations in reality
Priority	Number that determines the importance a person gives to a certain activity. This priority is modified according to the preferences of the person and the compliance with the daily schedule.
Desirable start time	Indicative start time at which the person would like to start performing the activity. If possible, the person will try to schedule the start of the activity at this time.
Desirable end time	Indicative end time at which the person would like to have the activity already finished. If possible, the person will try to schedule the end of the activity before this time.
Mandatory start time	Compulsory time at which the activity must start. Necessary to model time critical activities such as going to work or watching a certain TV show.
Mandatory end time	Compulsory time at which the activity must finish.
Preconditions	Activity codes for modelling prerequisites, i.e. other related activities that must be already finished by the time this activity is chosen. These codes will be checked against the person's *DONE* list to ensure their fulfilment. If there are pending preconditions, these missing activities will be added to the *TODO* list and the person will choose once again which activity they will perform.
Outcomes	Performing some activities may lead to the appearance of other new activities, like needing to *take a shower* after *doing exercise*, or *toileting* after *having lunch*. For this purpose, there is a list of activity codes so that when an activity finishes, these new outcomes are appended to the person's *TODO* list.

3.3.4 OpenSHS

OpenSHS [69] uses scenarios for the selection of activities of a virtual person. This model recognizes only the highest level of activities and does not focus on lower-level actions or animations. It offers import of a 3D environment from a free tool, and the model itself is also available for download at [73]. To construct scenarios, it records participants controlling an avatar in the model in specific phases of a day (for instance early morning, afternoon, and so on). Participants control the avatar

from the first-person perspective and manually aim at switches, door handles, or other interactive devices. Participants can start an activity from a limited list, set its duration, and the simulation skips time to the end of the activity, so they continue immediately.

When this interactive approach collects enough data, a researcher aggregates samples generated by participants to create the final dataset of scenarios that modularly covers the whole day (see Figure 13). At [74], the authors described this process in detail. They have gathered data by seven participants to generate complete datasets and detect anomalies (for example, when a person forgets to turn off a light or leave a door open). Datasets were used as classification sets for automated anomaly detection that is applicable in eldercare, healthcare, or security.

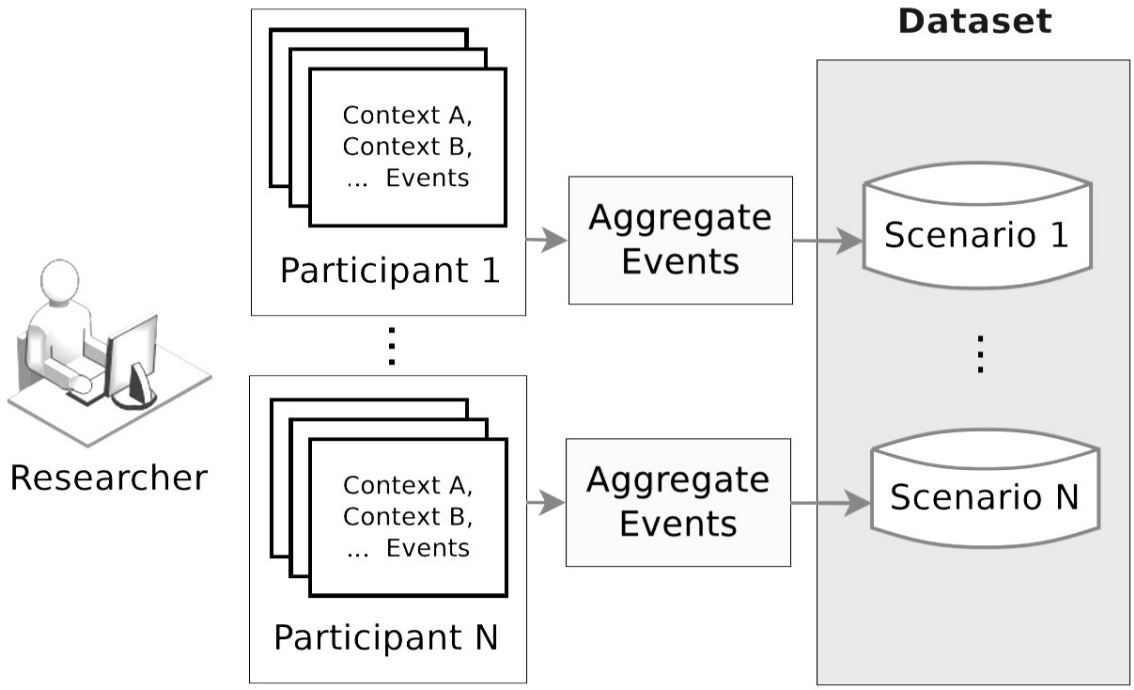

Figure 13: Creation of dataset (collection of scenarios) in OpenSHS. (source: [69])

3.3.5 Other Ambient Intelligence Simulators

Another open-source smart home simulator, SIMACT [75], aims for support of activity recognition in a 3D environment. As such, it is focused primarily on lower-level actions and simulations. The higher-level activity selection is simplified to a pre-recorded scenario. Along with the simulator, authors provide files with gathered data from experiments. This dataset contains performing time of actions, their usual order, mistakes or hesitations during activities, and other data. It also recognizes age, sex, and other attributes of participants to create a realistic performance of activities which could be later used for experiments of activity recognition.

DiaSim [76] is a simulator developed primarily for the detailed simulation of heterogeneous devices in pervasive environments. It supports large scale models with hundreds of agents with simple behavior created in a scenario editor. Three initial scenarios allow testing behavior of various smart devices under different workloads. Figure 14 shows the basic structure of its simulation model. For every type of sensor, there is a type of stimulus producer, which is often a mathematical function. This function is provided by experts and stimulates corresponding sensors to trigger events in the environment. For example, luminosity, as stimuli producer, is detected by light detectors. Depending on the settings of the system, information from light detectors may turn on/off lights (actuators). This change then affects the original stimulus producer.

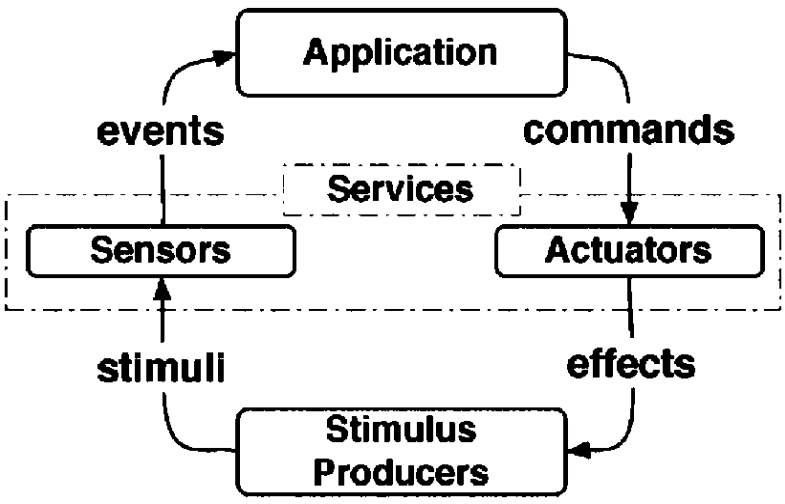

Figure 14: Simulation model in DiaSim. (source: [76])

4 Solution

This section presents the solution to the main goals of this work: FCMs modified for autonomous agents, scenarios management and the cooperation of both decision-making sub-systems. Then, this section also describes the configuration and implementation of genetic algorithms, the environment, and other issues linked with the implementation of the proposed solution into an ambient intelligence model. The content of this section is the original work of the author of this thesis.

4.1 Introduction to FCM-NAS

FCM-NAS modification enables agents to make decisions and simulate internal needs at the same time. The first advantage this matrix-based method offers is keeping human-readability despite its complexity and possibilities. That allows manual initial setup, which makes the agent functional from the beginning of development even without implementing a machine learning mechanism. Moreover, the relations are readable from matrix even after learning, which means reasoning is never black-box like in case of neural networks.

The second advantage of the method is its excellent adjustability which allows easy modifications or additions. The whole part of the method can be easily replaced with a different method if needed. It has been already successfully modified for large-scale models where performance is a crucial factor. In that case, Analytic hierarchy process (AHP) replaced the decision-making part of FCM-NAS [77]. The combination with AHP improved performance but also proved the adaptability and scalability of this method by various extensions. In this work, the schedule management sub-system will enhance the method to accomplish the goal of the human activity simulation.

The original design of FCM needs to be adjusted to fit in the role of agents' decision-making. This work will describe all the major changes in detail. The main difference lies in the classification of concepts. In the original design, all concept nodes have the same range of values, the same behavior, and the same

interpretation. This approach is sufficient in a dynamic system with equal elements of the same type. However, if an FCM needs to react or make decisions, some changes are unavoidable. Managing different kinds of nodes after computation often becomes very confusing and context-dependent; this hinders modularity and scalability of the FCM. The FCM-NAS offers a different design, which expects various types of nodes, and processes them according to the classification during computation. Therefore, any special treatment after computation is not necessary.

This method brings several features, which are impossible to reach by a standard approach without heavy editing of the algorithm. It enables the combination of monitoring and decision-making, easy processing of inputs from sensors, faster reactions of the system, more realistic behavior in simulations (disabling parallel activities if needed), a simulation of the inner needs, adjustments to the granularity of the simulation, and self-evaluation of the agent (fitness), which supports learning (more on these in section 4.2 Additional Features of FCM-NAS).

4.1.1 Structure

The fundamental step to introduce FCM-NAS into a project is the partition of concept nodes into three classes. $\mathcal{C}$ is a partition of the original set of concept nodes C (see (8)). Then C_N denotes the set of nodes which were identified as *Need* nodes; similarly, C_A represents the set of *Activity* nodes, and C_S is the set of *State* nodes:

$$\mathcal{C} = \{C_N \cup C_A \cup C_S\} \tag{8}$$

In the design phase, this approach requires the classification of all concepts into these classes. Also, since classes are pairwise disjoint sets, each concept node is in exactly one class. Original notation of the number of general concept nodes n is extended to distinguish the number of concepts in new classes: n_N for the number of *Need* nodes, n_A for the number of *Activity* nodes, and n_S for the number of *State* nodes. Then the following statements result from the previously mentioned features of the new structure:

$$\forall c_i : c_i \in (C_N \cup C_A \cup C_S) \tag{9}$$

$$n = n_N + n_A + n_S \tag{10}$$

The notation of individual nodes (c_i) and their values (v_i^t) stays the same as in the original FCM design. To distinguish the assignment of nodes to the introduced classes, it is necessary to define that the collection of nodes has the static order of nodes starting with *Need* nodes, then *Activity* nodes, and ending with *State* nodes. Therefore nodes are assigned base on their index:

$$c_i \in \begin{cases} C_N & for\ i = \{1, \dots, n_N\} \\ C_A & for\ i = \{n_N + 1, \dots, n_N + n_A\} \\ C_S & for\ i = \{n_N + n_A + 1, \dots, n\} \end{cases} \tag{11}$$

4.1.2 Concept Class Needs

Needs are the first class of concept nodes. Usually, a designed system has at least one purpose, which it is trying to fulfill. Then, a measure of success in the effort might be useful. Alternatively, it has to observe a variable and perform an action repeatedly to keep the variable under control. If the action costs limited resources or time, the system should consider those costs as it may not be convenient to take action too often. Then, the system could use a time delay or sensitive balancing. In such cases, *Need* nodes are the optimal choice for representation of the concepts. For example, this class is used for the needs of agents in an artificial life model or a simulated level of satisfaction with memory management in a system.

The main difference from the other nodes is their behavior during computation. Their computation is based on the incremental model introduced in (6), meaning it keeps its previous value and adds (or subtracts) an increment based on influence of other nodes. Therefore, if there are not active influences from the other nodes, then it holds its value. (12) shows the computation of value v_i^t of node c_i. This equation uses the basic transformation function f (see (7)). For a reminder, value w_{ij} represents the influence of node c_i on node c_j.

$$v_j^t = f\left[v_j^{t-1} + \left(\sum_{i=1}^{n} w_{ij} \cdot v_i^{t-1}\right)\right] \qquad for\ j = \{1, \dots, n_N\} \tag{12}$$

The value of a *Need* node represents the level of the necessity to do something to satisfy the corresponding need. Then, 0 means the need has been satisfied, and

an agent does not have to do anything. If its level approaches 1, then the agent fails in satisfying the need, and it should take appropriate action as soon as possible. Obviously, the exact critical level of a *Need* node, which initiates a corresponding activity, depends on the configuration of relations in matrix *R*. The design of *Need* nodes and its integration into FCM-NAS later enable features like the granularity of the simulation, addition of true positive causality to itself, self-evaluation of an FCM-NAS, and the varying necessity of actions (all described in section 4.2 Additional Features of FCM-NAS).

4.1.3 Concept Class Activities

The next class of concept nodes, *Activities*, represents all possible actions that a system or agent can perform. If an agent with an FCM-NAS is not just a passive observer and has to react, manage, or anyhow affect its environment or itself, then there are two possible solutions. The first approach involves another mechanism outside of an FCM that reads values from the FCM and makes a decision [78]. The other one puts the actions directly inside the FCM-NAS, which then holds the decision-making responsibility.

As actions, these concepts are or are not being active; therefore, after computation, the *Activity* nodes have only two possible values: zero (*false*; the action is inactive) or one (*true*; an agent performs the activity). However, during their computation, these values keep the full interval from zero to one. They are calculated with a similar algorithm as general nodes in the original FCMs (based on the definition model introduced in (5)). (13) shows the full computation of *Activity* nodes.

$$v_j^t = f_a\left(\sum_{i=1}^{n} w_{ij} \cdot v_i^{t-1}\right) \qquad for\ j = \{n_N + 1, \dots, n_N + n_A\} \tag{13}$$

The values are rounded using a simple algorithm (the transformation f_a), which decides the activation of *Activity* nodes based on their truth values acquired from the computation. Depending on whether parallel activities are available, it selects only the activity with the highest value (see section 4.2.3 Disabling Parallel

Activities) or performs all activities reaching a certain critical level a_c (the default is 0.5) as seen in (14):

$$f_a(x) = \begin{cases} 1 & if\ x \geq a_c \\ 0 & otherwise \end{cases} \qquad where\ x \in \mathbb{R} \tag{14}$$

In cases when *Activity* nodes are supposed to provide a truth value to express an intensity of actions, basic *f* transformation (7) could be used. Transformation functions are adjustable to the demands of the system without any issues with the rest of the design.

Generally, FCMs are not suitable for a combination of high-level decision-making ("what should be done") and low-level operations ("how it should be done"). That combination often requires too many variables in one structure which causes performance issues. Low-level problems (for example, pathfinding) would be extremely ineffective when managed by FCM-NAS along with the other concept nodes. It could be solved by another FCM dedicated to pathfinding. However, in any case, low-level operations should be separated from high-level decision-making to ensure clarity of the model and the most importantly reasonable performance.

4.1.4 Concept Class States

The third class of concept nodes *States* is very similar to general concepts in the original FCMs. The way how it is calculated is almost the same (see (15)), and its purpose has not narrow focus as the previously presented classes.

$$v_j^t = f\left(\sum_{i=1}^{n} w_{ij} \cdot v_i^{t-1}\right) \qquad for\ j = \{n_N + n_A + 1, \dots, n\} \tag{15}$$

Besides general concepts, the *State* nodes are the advantageous choice for external inputs. If an agent needs to be able to perceive an attribute of the environment and take it into account during the decision-making process, then it requires a dedicated *State* node in its FCM-NAS. As a property of the environment, the value of this node is not directly affected by any node in the FCM-NAS. Therefore, all relations to this node in matrix R equal zero, which allows omitting its whole

calculation. Instead, it is updated by sensors. Obviously, the inserted values have to be transformed into truth values (ranging from zero to one).

4.2 Additional Features of FCM-NAS

While the previous section described the core of the proposed method, this section provides optional attachments that are very useful for certain systems but unnecessary for others. Although a few of these are dependent on each other (the dependencies are noted), the core design presented above can be implemented independently of the following features.

4.2.1 Granularity

Since agents process an FCM-NAS through iteration in discrete time steps, the selection of the period between computations of FCMs is a crucial issue regardless of whether the system deals with continuous real-time or discrete time steps. The values of an FCM in a dynamic environment are always valid only within the particular interval for which the FCM was designed. Generally, a shorter interval generates better reactions of agents, but it has higher performance demands per system time unit as it is processed more often. This is very similar to sampling in signal processing; when samples are too distant, much of the information between them can be lost.

Figure 15 shows how an FCM-NAS perceives a continuous variable with different settings of the granularity in continuous time. The FCM-NAS with a granularity of 0.5 is four times more demanding in terms of performance but is also more precise than one with a granularity of 2.0. The parameter g is a multiplier representing the length of an interval between computations, which is relative to the default length of the interval for which a model was initially designed.

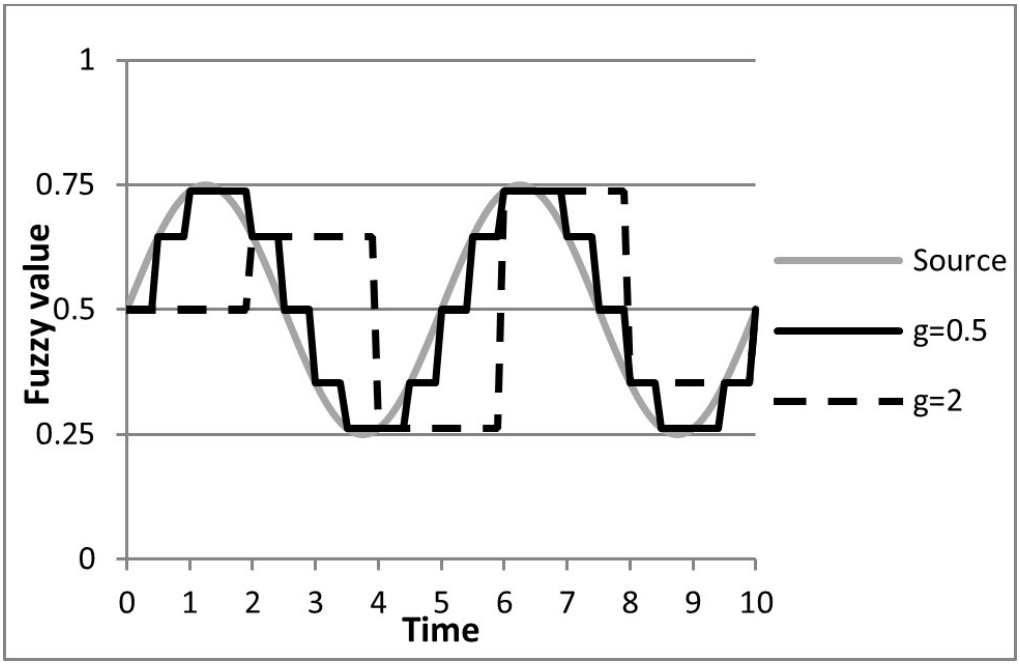

Figure 15: Comparison of different settings of granularity. (source: author)

Note that in a single system, the granularity parameter of agents may vary, meaning that they can have different reaction times. In a model with continuous time, the designer chooses how often an FCM updates its values. In a model with discrete time, the problem instead lies in deciding how much virtual time (or how many time steps) elapses between the computations or how much the environment changes in a single time step. After an FCM is designed, any change to the length of this interval invalidates certain values related to dynamic phenomena in the environment. However, with the proposed solution, the influence of granularity on concept classes is evident; *Need* nodes are obviously affected; *Activity* nodes are not, since a decision could be made at any time; and *State* nodes typically are not, but since they have a broader use, some may be affected in certain cases.

The effect of granularity on *Need* nodes causes differences in their growth (or decrease). This can be easily compensated for with a simple enhancement of the calculation, and the nodes always adjust to the current simulation speed. (16) shows the addition of the granularity parameter g, which adjusts the size of the increment every time step.

$$v_j^t = f\left[v_j^{t-1} + \boldsymbol{g} \cdot \left(\sum_{i=1}^{n} w_{ij} \cdot v_i^{t-1}\right)\right] \qquad for\ j = \{1, \ldots, n_N\} \tag{16}$$

The values of the *Need* nodes reach $(1/g)$ times more computations per virtual time unit than they would have with g = 1. For example, in an artificial life model, an agent escaping from a predator has depleted stamina after three time steps with the default granularity; with the granularity at 0.5, the agent manages to run over six computations. Of course, in both cases, it runs the same length and for the same amount of virtual time, but due to the granularity, it has twice as many opportunities to reconsider the activity or the direction of its escape.

The main advantage of this feature is the adjustability of a model during its development. It allows a designer to set values according to a specific time frame during a design phase and later adjust the time frame without the requirement of redesigning the whole matrix of relations. For example, if the value of node c_1 (for simplicity, assume it is always at 1.0) is supposed to increase the value of *Need* node c_2 at rate 0.6 per hour, and the matrix is designed in this time frame in mind, then w_{12} would be set at 0.6. However, later in the development, a designer would realize that one computation per hour is not enough for the system. For the period of 1 minute between computations, granularity parameter g would be set at 1/60, and the system would process FCM once per minute. Thanks to this feature the designed increase rate of *Need* node c_2 stays automatically at 0.6 per hour (or 0.6/60 per minute). Otherwise, the designer would need to edit every value in matrix R by hand with every change to the granularity of the model. This process would be especially laborious when testing different computation intervals.

4.2.2 Faster Reactions of Agents

As seen in (5) and (6), the original FCMs compute the current values by using the values of the previous iteration. This procedure inevitably causes a delay between a stimulus and the corresponding reaction (i.e., one time step or the interval between computations of the FCM). Depending on how often the FCMs are recomputed, this delay may cause problems if the short reaction time is essential

for the proper operation of a system. For example, a monitoring system should react to fire immediately, as soon as sensors detect it, rather than waiting for the next iteration to take action.

Some researches solve this with incremental or cumulative models [19]. However, their goal is to achieve FCM with stable converged values in the shortest time possible. FCM-NAS does not seek one set of values to able to decide; instead, it is using values of every step to make decisions. With considerations of the objectives and abilities of this method, there is a far more efficient way to ensure not only faster but even immediate reactions of agents.

Thanks to the partition of concept nodes to different classes, the computation of an FCM-NAS can be divided into three parts, which can then be rearranged in any order. Moreover, some parts can even consider the values of the current iteration from parts that have already been computed. The best order has proved to be as follows: first the *Need* nodes, then the *State* nodes and finally the *Activity* nodes. It is because *Need* nodes do not have to correspond with the most current values; since they use an incremental model as shown in (12), their values are primarily based on their own previous values and actions performed in the previous time step. *State* nodes may be based on the current values of needs, but also contain external inputs that have to be considered in the decision-making as soon as possible. Finally, *Activity* nodes, as the decision-making part of the model, should access the latest values in order to give the best possible reaction to the current situation.

The implementation of this feature requires only the replacement of values from the previous time step v_i^{t-1} with the current ones v_i^t for classes that have been already computed. (17) and (18) show adjusted expressions for the order recommended above. The equation for *Need* nodes is not affected since no other current values are yet available for time step *t*.

$$v_j^t = f\left[\left(\sum_{i=1}^{n_N} w_{ij} \cdot v_i^t\right) + \left(\sum_{i=(n_N+1)}^{n} w_{ij} \cdot v_i^{t-1}\right)\right] \quad for\ j = \{n_N + n_A + 1, \dots, n\} \tag{17}$$

$$v_j^t = f_a\left[\left(\sum_{i=1}^{n_N} w_{ij} \cdot v_i^t\right) + \left(\sum_{i=(n_N+1)}^{(n_N+n_A)} w_{ij} \cdot v_i^{t-1}\right) + \left(\sum_{i=(n_N+n_A+1)}^{n} w_{ij} \cdot v_i^t\right)\right]$$
$$for\ j = \{n_N + 1, \dots, n_N + n_A\} \tag{18}$$

For example, in an artificial life model, agents have two concept nodes: the *Danger* state and the *Escape* activity. When an agent recognizes a dangerous situation, it should immediately escape rather than wait until the next time step to take action. Tables 3 and 4 show both approaches; obviously, an agent with the FCM-NAS with this feature has a much better chance of escaping and surviving.

4.2.3 Disabling Parallel Activities

In many systems, the individual actions are independent of each other; however, there are cases where an agent is limited to one action per time step since different activities require different positions of the agent or several of them utilize a single actuator. The original FCM method cannot restrict this without another algorithm that processes and adjusts the values of specific nodes. If any process or value is bound to specific nodes by its position in an FCM, then the algorithm has to be adjusted after any change. This is confusing and a less modular approach.

Table 3: The delay in reaction time in the original FCM. (source: author)

Iteration	1	2	3	4	5	6	7	8
Danger	0	1	0	0	1	1	1	0
Escape	0	0	1	0	0	1	1	1

Table 4: Immediate reactions in the FCM-NAS. (source: author)

Iteration	1	2	3	4	5	6	7	8
Danger	0	1	0	0	1	1	1	0
Escape	0	1	0	0	1	1	1	0

The FCM-NAS uses a simple adjusted transformation algorithm for its *Activity* nodes, meaning that it is not bound to specific nodes, which allows simple changes in concepts without disrupting the algorithm. This feature is useful in human activity simulation, artificial life models, or more generally for any non-trivial decision-making. For example, as a simulated person, the agent should not be able to eat, drink, and sleep at the same time step. The following code describes the adjusted transformation algorithm for non-parallel activities:

max = 0.0
max_position = -1
for (i = n_N + 1, ..., n_N +n_A){
    if (v_i^t > a_c AND v_i^t > max){
        max = v_i^t
        if (max_position ≠-1)
            $v_{max\_position}^t$ = 0.0
        max_position = i
        v_i^t = 1.0
    }else
        v_i^t = 0.0
}

When disabling of parallel activities is required, the algorithm finds the *Activity* node with the highest value after all nodes are calculated. If the found value exceeds a critical level a_c, then the activity is performed; otherwise, the agent does nothing (alternatively returns to its default state or start a free time activity). Within this single cycle, the algorithm finds the most necessary activity and rounds the values to zero or one.

4.2.4 Primary State, Fitness, and Constant Increments

In any field, an evaluation of a system is a critical topic, regardless of whether it is done for designers of the system or purposes of autonomous learning; the development and progress of any system always depend on some kind of feedback. Since the *Need* nodes in the FCM-NAS serve as indicators of success in particular tasks, their values can easily be converted into a general evaluation measure of the success (fitness) of an FCM.

In order to integrate the fitness into an FCM-NAS, a *Primary State* (*PS*) node is added to the *State* nodes. Only *Need* nodes that are included in the evaluation of the system can affect the *PS* node. If the recommended convention for the setting of the *Need* nodes is met (a value of zero indicates maximal satisfaction; no action is required), then all these relations are negative. Therefore, the higher the value of a *Need* node, the lower the fitness of an FCM-NAS. Since fitness ranges from zero to one and is likely to change during the previous computation, its base value has to be reset to one before computations. The relation of the *PS* node to itself w_{pp} is also equal to one, where p is the position of this node in an FCM-NAS.

For example, Figure 16 presents the part of the FCM-NAS of the person in the ambient intelligence model. This part contains three *Need* nodes and the *Primary State* node. Table 5 shows the corresponding part of the matrix *R*. The last row in the table contains the relations of all nodes to the *PS* node. Since the value of the *Need* nodes decreases with increasing satisfaction, these relations are negative, and *PS* node is set to 1.0 before computation begins. In this example, *Hunger* and *Thirst* are more important to success than *Boredom*; they, therefore, have a much higher negative impact on the fitness of an agent.

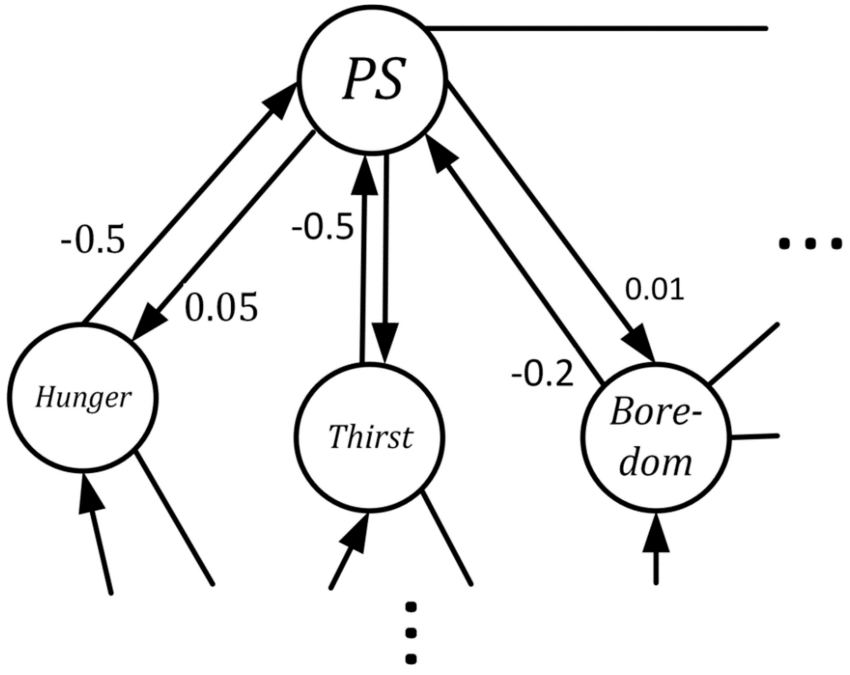

Figure 16: An example of Primary State node in a part of an FCM. (source: author)

Table 5: An example of matrix R of an FCM with Primary State node. (source: author)

	Hun.	Thir.	Boredom	Prim. S.	…
Hunger	0	0	0	0.05	
Thirst	0	0	0	0.1	
Boredom	0	0	0	0.01	
Primary State	-0.5	-0.5	-0.2	1	
…					…

If the *PS* node starts at 1.0, then it can be used as a constant node, i.e. serving as a constant increment to any node. For the *PS* node c_p and a node c_i, the relation w_{pi} guarantees that a steady increase (or decrease) is added to node c_i at every time step. This relation is especially useful for the stable growth of *Need* nodes. The last column of Table 5 shows the positive relations of the *PS* node to the *Need* nodes. These relations simulate a constant increase of the needs over time. For instance, *Thirst* would be increased by 0.1 per every time step (with default granularity parameter g = 1.0).

4.2.5 Necessity

The necessity of actions provides the FCM-NAS with another useful measure for decision-making. If several *Need* nodes have high values, then the decision-making prefers actions that relate to the most vital need. This feature is useful for the decision-making process in an FCM-NAS with disabled parallel activities. For example, in the artificial life model, an agent with values of both *Hunger* and *Boredom* of 1.0 will prefer activities that lead to the meeting of a more critical need. In the case shown in Table 5, the agent would select eating rather than entertaining activity, since the *Hunger* need affects the *PS* more than the *Boredom* does.

This feature uses states to evaluate the necessity of the *Need* nodes. The *PS* node is recommended since at least one fitness value is required. More states can represent different fitness functions of the system, and the necessity feature covers even this possibility. At the first step, a designer identifies those *State* nodes that are used as fitness values and compares their necessity to the system. The constant d_i represents these evaluations for all *State* nodes in the form of values ranging

from zero to one, where d_i = 1.0 means that *State* node c_i has the highest priority and vice versa. Therefore, the necessity value for a *PS* node would be at 1.0, other *State* nodes would range from 0 to 1 depending on their relative importance to the *PS* node, and all *State* nodes that do not serve as a fitness value (for instance external inputs) would be at 0.0. Note that index *i* does not start at one for d_i values since *i* represents the position of the node in an FCM-NAS that also contains other types of nodes which come before *State* nodes.

During initialization of the system, the necessity of each *Need* node is calculated according to their influences on the *State* nodes and the corresponding values d_i:

$$e'_i = \sum_{\mathrm{j}=(n_N+n_A+1)}^{n} \left(-\mathrm{w_{ij}} \cdot \mathrm{d}_j\right) \qquad for\ i = \{0, \dots, n_N\} \tag{19}$$

Then the coefficients are adjusted by an increment k_n that shifts their values, so their mean is 1.0. This way, setting of absolute values of necessity does not affect the activation of *Activity* nodes:

$$k_n = 1 - \frac{\sum_{\mathrm{i}=1}^{n_N}(e'_i)}{n_N} \tag{20}$$

$$e_i = k_n + e'_i \qquad for\ i = \{0, \dots, n_N\} \tag{21}$$

Then, e_i is the final necessity coefficient of *Need* node c_i and e'_i is the necessity value e_i without the compensation of the offset k_n. In the computation of an FCM-NAS, the necessity is used to calculate the *Activity* nodes from the *Need* nodes. (22) shows the placement of the necessity coefficient. Calculations of the effects of *Need* nodes to *Activity* nodes have to be separated; otherwise the rest stays the same as in (13).

$$v_j^t = f_a\left[\left(\sum_{\mathrm{i}=1}^{n_N} \boldsymbol{e_i} \cdot w_{ij} \cdot v_i^{t-1}\right) + \left(\sum_{i=(n_N+1)}^{n} w_{ij} \cdot v_i^{t-1}\right)\right]$$
$$for\ j = \{n_N + 1, \dots, n_N + n_A\} \tag{22}$$

4.2.6 Processing FCM-NAS

The process of computing the FCM-NAS occurs periodically in intervals with constant length. This procedure contains calculations of each class, reading values for input nodes, and the selection of activity. The fact that all classes are processed individually means it can be rearranged and also use new values of previously processed nodes. Figure 17 shows the best possible order of computations which allows the decision-making part of the FCM-NAS to react to the current situation instead of the situation at the previous time step (more in section 4.2.2 Faster Reactions of Agents). At the first phase, *Need* nodes are updated; then, computation of *State* nodes can optionally use these new values. After this part, external inputs are imported to dedicated *State* nodes. Alternatively, this import could appear in the very beginning since the calculation of nodes dedicated to external inputs would be ideally omitted. However, if these calculations are processed, they would overwrite the value after its import.

At this stage, the new updated value of *Primary State* is copied to an external variable, if this node is used for constant increments. In that case, *PS* node would be reset to 1.0 before the computation of *Activity* nodes. During this stage, *Activity* nodes are calculated; optionally using both updated sets of nodes to make reactions of the decision-making immediate. Finally, values of *Activity* nodes go through the transformation function to select the activity (or activities) to perform. Then lower-level decision-making or actuators take control until the next computation.

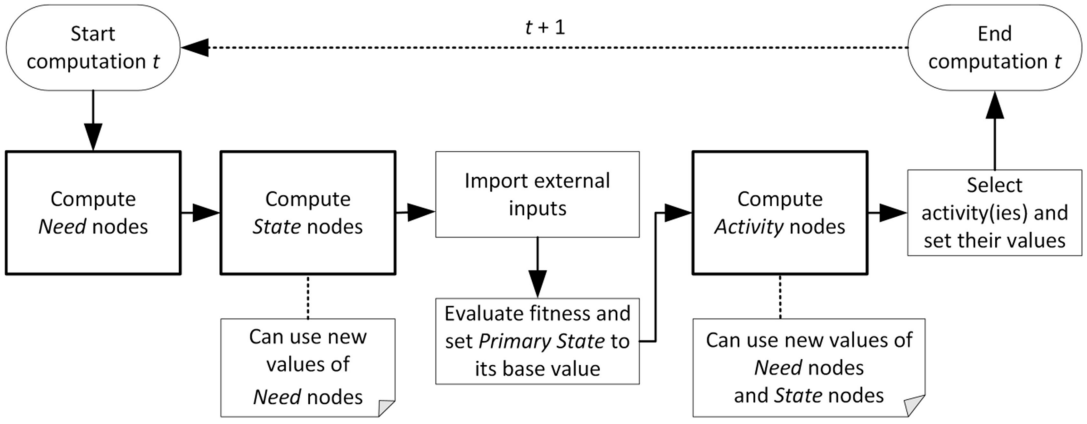

Figure 17: Summary of the computation of the FCM-NAS optimized for autonomous agents with described features. (source: author)

4.3 Applications of FCM-NAS

The FCM-NAS method (or its previous iterations) has been already successfully used in two other models on different platforms. It has been developed and enhanced since the author's bachelor thesis [79]. The initial motivation to create such a method was the implementation of an intelligent system in a virtual environment of Second Life. It is a unique platform from a programming standpoint. It is limited by memory and computational performance as everything runs on distant servers optimized for multiple user experiences in a 3D environment. It offers basic procedural programming language Linden Script Language which is meant for interactions of users with scripted objects. The implementation of any complex system within this setting presents a challenge.

This system manages memory usage and sensory data transfer in the computationally very limited environment of the virtual world Second Life. It decides when to transfer data collected by sensors and clear memory depending on inner states and the presence of agents (in this case, avatars of users). Since it manages only its internal sub-systems, its behavior does not directly affect the environment. The only way to extract data is through its communication sub-system, which translates user messages in a native language into database queries and returns corresponding information. Figure 18 shows the architecture of the system regarding information flow.

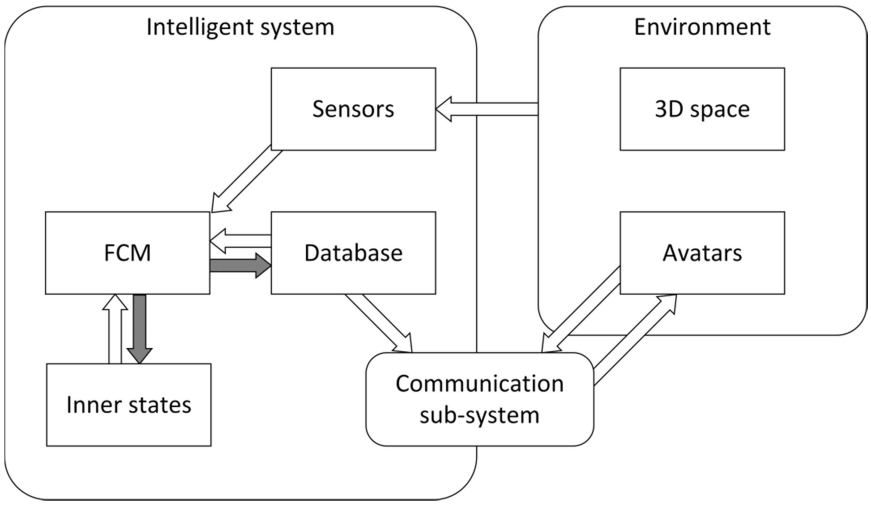

Figure 18: Diagram of the intelligent system managing its database within a limited environment; white arrows mean direction of information flow, dark arrows represent the direction of control or influence. (source: author)

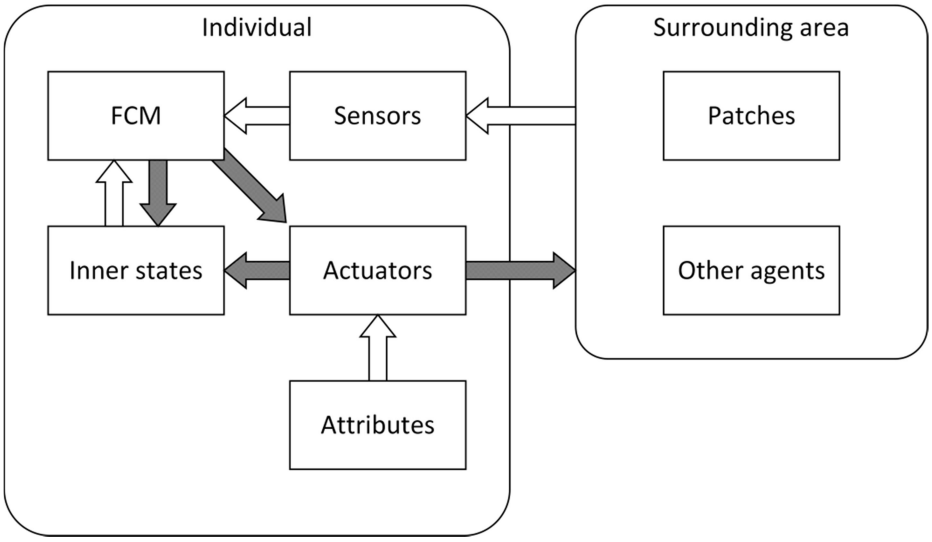

Figure 19: Diagram of an agent in the ALModel; white arrows mean direction of information flow, dark arrows represent the direction of control or influence. (source: author)

After the intelligent system, this method was adjusted for agent-based approach, and it was implemented into an artificial life model ALModel that features a randomly generated two-dimensional environment with resources and thousands of agents [78, 80]. It was built on the platform NetLogo 5, and it is available for download at [81]. The simulation runs in discrete time steps at which each agent makes a decision about its activity. Parallel activities are disabled; therefore, agents cannot perform more than one activity per time step. As shown in Figure 19, the FCM-NAS handles almost whole agents' logic as it simulates inner states (especially needs), processes information from sensors, and performs decision-making.

Decision-making in ALModel covers a few context areas which agents need to consider: food, energy consumption, reproduction, and keeping out from any danger. As seen in Table 6, each aspect of artificial life requires more than one node. For instance, food management of agents uses one *Need* node for a level of *Hunger*, two *Activity* nodes for *Feeding* and *Searching for food*, and a *State* node used as an external input, which allows taking into consideration currently available food supplies on the current positions. Generally, more concept nodes in any FCM mean more possibilities, but also, the FCM becomes more performance demanding.

Figure 20 shows 100 time steps of a single agent in the model. The top plot in the figure depicts the development of the *Need* nodes and *Primary State* node (fitness). The bottom plot explains the changes in the values in the plot above with a log of the activities that this agent performed during the observed 100 time steps. During the first 25 time steps, the agent struggled to find resources: *Need* nodes were increasing, and its fitness was decreasing. Then the agent finally found resources, rested and reproduced. Reproduction raised values of *Need* nodes due to its cost set as negative relations to *Need* nodes. Since agents remember the last spot where they found resources, it was able to finish the next search for food and water much faster. Except for *Sleep*, *Escape*, and *Searching*, all activities floor the value of the corresponding *Need* node in a single time step.

Table 6: Concept nodes and their classification in ALModel. (source: author)

Need nodes	Activity nodes	State nodes
Hunger (c_1)	Feed (c_5)	Primary State (c_{13})
Thirst (c_2)	Drink (c_6)	Lack of Food (c_{14})
Fatigue (c_3)	Sleep (c_7)	Lack of Water (c_{15})
Reproductive Need (c_4)	Reproduce (c_8)	Lack of Partners (c_{16})
	Search for Food (c_9)	Danger (c_{17})
	Search for Water (c_{10})	
	Search for Partner (c_{11})	
	Escape (c_{12})	

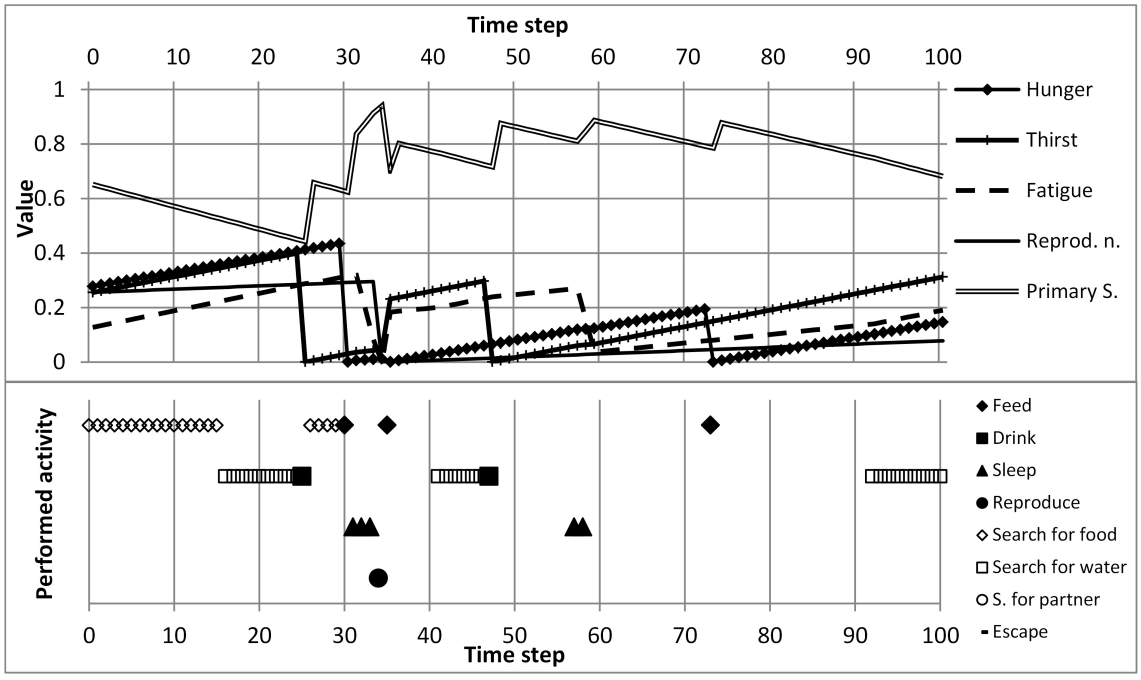

Figure 20: Need and Activity nodes of an agent in the ALModel over 100 time steps (source: author)

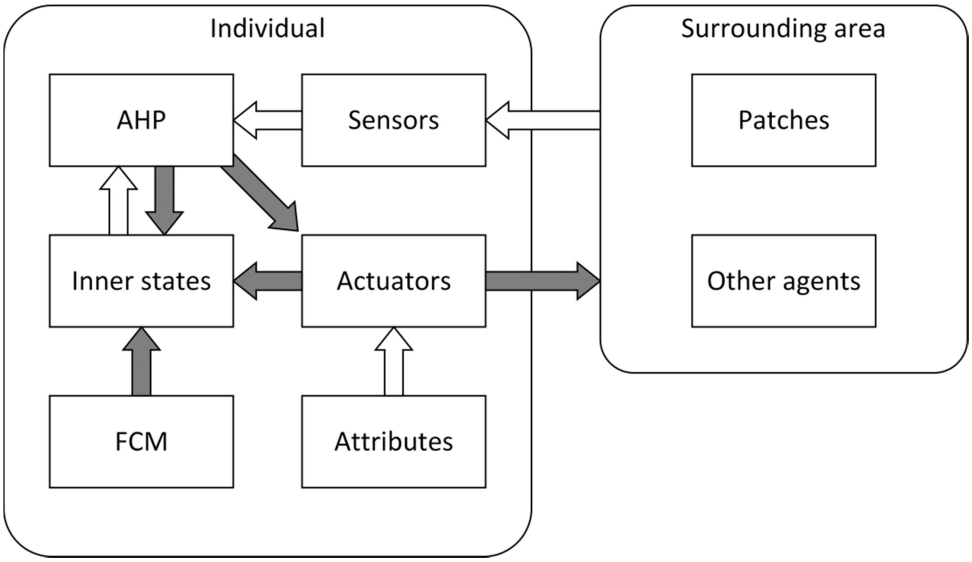

Figure 21: Diagram of an agent with the combination of FCM-NAS and AHP; white arrows mean direction of information flow, dark arrows represent the direction of control or influence. (source: author)

Although this platform is not that limited as the previous one, the computational complexity is a still serious concern for the model with thousands of agents at the same time. Consequently, the method has been modified for large-scale models. As shown in Figure 21, the AHP replaced the decision-making part of FCM-NAS. The modified FCM-NAS was used only for simulation of inner states which made its matrix of relations significantly smaller. It improved performance but slightly decreased the decision-making capabilities of the method. This modification was published at [77].

The integral part of any agent-based artificial life model is the evolution emerging from the competition for resources, inheritance, and breeding. In the case of ALModel, along with agents' physical traits in the model, evolutionary principles were used to adjust the decision-making part of the FCM-NAS which enabled learning and behavior adaptation of species. This decision-making section of FCM-NAS consists of all values of relations in the matrix *R* that affect an *Activity* node – meaning all rows corresponding to Activities are part of the decision-making process, as shown in Table 7. During processing, these values are multiplied with the current value of the corresponding node (column) to add to the value of the activity node (row). These values define the whole behavior of the agent – in ALModel, this section contained 136 values (with 8 activities and 17 concept nodes total).

Table 7: Overview of the most important sections of matrix R of an FCM-NAS. (source: author)

	Needs	Activities	PS	States
Needs		Satisfying needs	Growth of needs	
Activities	Decision-making section			
Primary state	Evaluation			
States	All values here are zero for external input states			

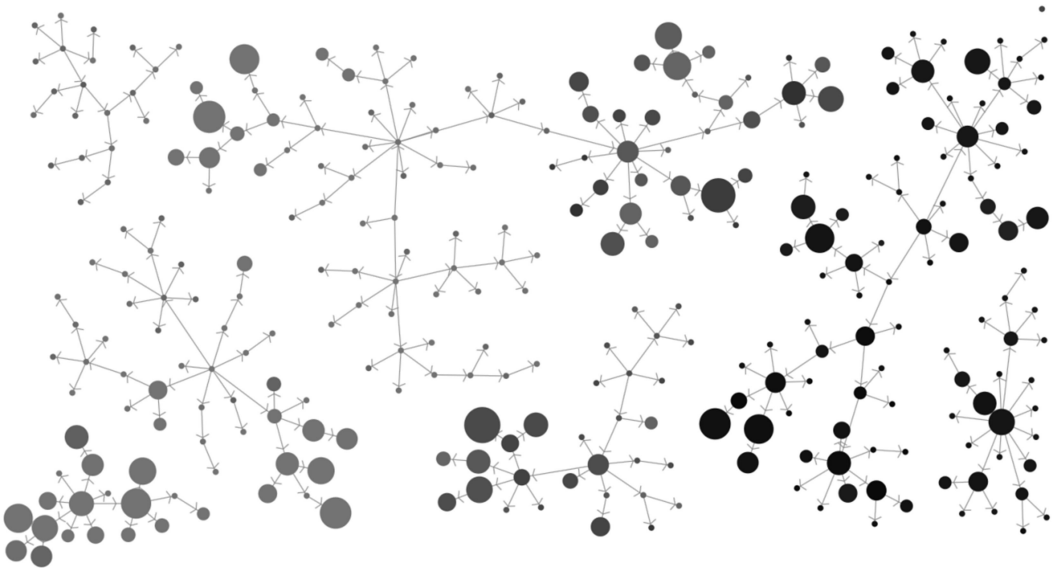

Figure 22: Example of phylogenetic trees created in ALModel. (source: author)

ALModel also uses cluster analysis to differentiate species and build phylogenetic trees – a very similar way to EcoSim [42]. Figure 22 shows phylogenetic trees of species produced in a single model run, in which each bubble represents a different species. The arrows point to the species that evolved from the species on the other end of the arrow, and sizes of the bubbles represent the numbers of agents that were classified as members of that species at the moment of creating the graph (small dots correspond to extinct species). The brightness of the bubbles depicts a certain attribute of a species; in this case, it is the type of diet (a black bubble means a carnivore species; a brighter one a herbivore species).

Obviously, evolutionary principles are not limited to artificial life area only. In models without simulated natural selection, genetic algorithms use fitness evaluation and artificial selection based on the fitness instead. This way the similar learning can be applied to FCM-NAS in every situation where the conditions allow its evaluation and its simulation environment supports hundreds or thousands of repeats of a model run to process generations of genotypes.

4.4 Routine Behavior

The previous implementations proved that FCM-NAS allows for rational behavior with multiple objectives, learning, and interactions. However, this is only a part of a problem in the human activity simulation. As presented in the section Human Activity Simulation, routine behavior is an essential component of this area as well. This section explains its requirements and obstacles that may appear during the design and implementation of routine behavior. This section is also the author's contribution to papers [82] and [83], which are not published yet.

4.4.1 Scenarios

In the domain of ambient intelligence, scenarios are used for a long time as a mean to capture standard behavior of agents. A scenario is a set of activities with given order and often attached timestamps. For the simulations, scenarios might include additional convenient information about activities: the location where the activity should be performed, which objects does the activity use in the environment, and other useful information.

Scenarios may have various formats and representations of time or continuity. However, any scenario for the model must be transferred into a machine-readable format like CSV files, XML structures, or tables. This chapter describes an implementation based on tables because the model use database to store the data, and tables are still human-readable as well.

The used table representation contains two tables to comprise all the needed information about a scenario. The first one (see Table 8) stores a schedule with data exclusive to each occurrence of an activity in the scenario. There are two necessary items: the name of the activity and its anticipated start time. Two other complementary values were added for the better inclusion of routines into the model. The *Importance* value serves during decision-making as a measure of how important is the activity. When the agent is under time pressure and has less time than scheduled activities take, then it skips or postpones the less important activity. The *Probability* value enables simple irregularities – when this value is lower than 1.0 (meaning it will be always in the final schedule), the activity has

a chance to not appear in some daily schedules of the agent. For instance, the activity exercise with the value of *Probability* 0.7 will appear in 7 out of 10 days on average. Note that the *Probability* of 1.0 does not mean the agent always performs the action; it is added to its schedule every day, but it still might be skipped (more in 4.4.2 Schedule Management).

Table 8: An example of the table with a scenario. (source: author)

Time	Activity	Importance	Probability
6:00	toileting	0.9	1.0
6:15	medicine	1.0	1.0
6:30	exercise	0.5	0.7

Table 9: Description of parameters of activities. (source: author)

Parameter	**Type of input**	**Description**
Duration	Number of minutes	How long it will take until the activity is finished (minimum time).
Max multiplier	Coefficient	How many times longer than its base duration it may take.
Place	Location	Where activity takes place.
Needs to be completed	True/false	If true, the effect of the activity is applied after its end. Otherwise, activity affects agent continuously.
Required activity	Name of activity	When the activity cannot be performed, the agent does require activity instead (to collect a consumable item required by this activity).
Uses	Name of device	Furniture or device used during the activity.
Consumes	Name of item	The item in the agent's inventory which will be consumed by the activity.
Creates	Name of item	The item which will be created and added into the agent's inventory.
Conditions	Atribute, operator and target value	What conditions need to be met to proceed with the activity.
Backup activity	Name of activity	Activity triggered if conditions of current activities are not met.
Effects	Attribute and value	How the activity affects needs of the agent.

The second table contains general information about all possible activities regardless of when or how often they are occurring in the scenarios. The second table holds the complete list of all possible activities with their configuration applying to every occurrence of the corresponding activity. All required parameters of activities used in this model are described in Table 9. Both tables are not limited in either of two dimensions: adding rows for new activities or even adding columns for additional parameters about activities.

The parameter "*Needs to be completed*" splits activities into two main groups regarding whether they are effective only once at the end or their effects are active through the whole period of execution. In the first group, activities have an objective, expected duration, and their effect occurs only when they are fully finished. When interrupted, it may continue later or start from the beginning, but there is no effect until its completion. Examples of the first type are the preparation of dinner, taking medicine, or getting a laptop. The second type of activities affects an agent even during its performance. There might be expected duration, but when interrupted, the agent still partially benefits from its effects. For example, sleeping, watching television, playing games, or working belong to this category.

Parameters like *Conditions*, *Effects,* or *Backup activity* were added to connect with the FCM-NAS component of an agent. Before an activity starts, *Conditions* are checked. Each condition consists of three values: name of the attribute, comparing sign (<, > or =) and the target value. Each effect has two values: the name of an attribute and a value, which may be both positive and negative. In this model, conditions and effects are connected to concept nodes in FCM-NAS. However, there may appear other attributes of an agent like its age, gender, health, and others.

4.4.2 Schedule Management

To be able to work with a scenario in the model, each agent loads it from the database and stores it as a schedule of activities. Each activity keeps information about where and when it is supposed to happen, which object it consumes, which equipment it uses, and so on. Because the agents deal with their needs or

unexpected events, they have to handle the possibility of delaying or even skipping a planned activity. To manage these situations, the schedule has a buffer where it stores activities that are about to begin or are already delayed.

Since each activity has a truth value determining its importance (given by a scenario configuration), when an agent does not manage to complete all scheduled activities, then *Importance* values along with supposed starting times determine which activities should be prioritized. However, as a static value, *Importance* is only a part of the mechanism calculating a dynamic value representing the priority of the activity at every time step. To work properly in a dynamic environment, it is accompanied by other values defining its lifespan during the simulations: *Near_planned*, *Delay*, and *Planned*. All three values have an effect on the decision-making process, although only two of them are used for the priority function and final evaluation of behavior. The *Near_planned* value only alerts decision-making about planned activities in the near future (60-30 minutes in this case), so the agent has time to prepare and satisfy needs that would otherwise threaten to interrupt the ongoing planned activity.

To determine *Planned* and *Delay* values in time, a few auxiliary parameters of activities were added into their configuration (see Figure 23). The parameter *Tolerance* defines the interval around the planned start time. If the activity starts within this interval, it is considered that the activity started in time. The parameter S*tart tolerance* represents a number of minutes before the planned start. After this point in time, the corresponding activity is added to the activity buffer, and its *Planned* value starts to grow – alerting the decision-making process that there is an activity in the buffer soon to be performed. If the activity is started before reaching the tolerance interval, then it is considered as prematurely triggered, which penalizes its evaluation. The similar rules apply for the parameter *Delay tolerance* except it is tied to the *Delay* value. When this point in time is reached without the activity started yet, this activity is considered skipped, and it is removed from the buffer with the lowest evaluation for this activity which decreases overall agent's rating (depending on the *Importance* value of the activity).

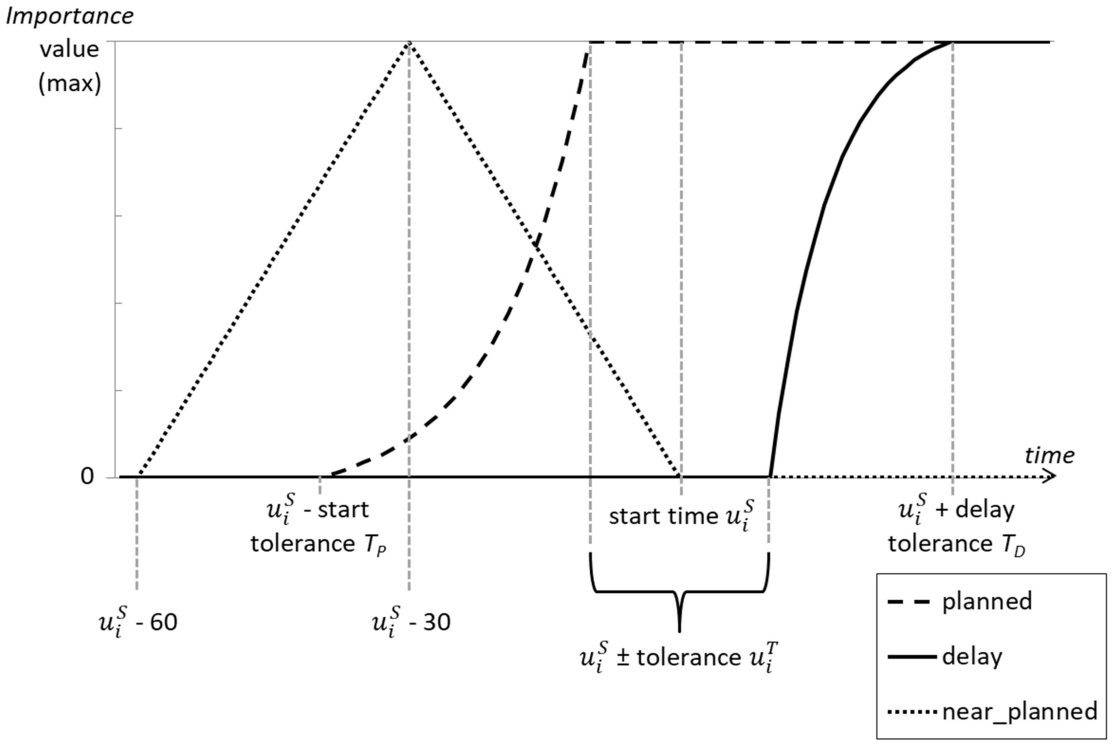

Figure 23: Dynamic values defining an activity during their lifespan (source: author)

The previously mentioned values are used to express the priority of each activity at each time step *t*. The higher-level decision-making requires priorities to solve possible conflicts and sort activities by their current priority. The algorithm calculates the priority value of each activity from its importance and the interval between the current time and supposed start time of the activity. The priority is calculated regarding time difference t_d of the current time t and the planned start of the activity u_i^S (see (23)). After adding an activity to the schedule at time step $u_i^S - T_P$, its priority raises until the time reaches tolerance interval around the planned start of the activity. During this tolerance period, the priority stays constant at the level of the importance of the activity u_i^I. When the activity is delayed beyond the tolerance interval, the delay part of the priority starts to rise until it is removed at time step $u_i^S + T_D$:

$$t_d = t - u_i^S \tag{23}$$

$$\hat{u}_i^t = \begin{cases} f_P\left\{\frac{T_P - |t_d|}{T_P - u_i^T}\right\} u_i^I & , t_d < -u_i^T \\ u_i^I & , t_d = [-u_i^T, u_i^T] \\ u_i^I + f_D\left\{\frac{|t_d| - u_i^T}{T_D - u_i^T}\right\} u_i^I & , t_d > u_i^T \end{cases} \tag{24}$$

where $\hat{u}_i^t$ is the priority of activity u_i at time t. Constants T_P and T_D determine chosen intervals for adding/discarding activities into/from Activity buffer. Both functions f_P and f_D adjust the growth of the value over time and caps it at 1.0:

$$f_P(x) = \max\left(\frac{l^x - 1}{l - 1}; 1\right) \qquad where\ x \in \mathbb{R}\ and\ l \in (1, \infty) \tag{25}$$

$$f_D(x) = \max\left(\frac{l^{-x} - 1}{l^{-1} - 1}; 1\right) \qquad where\ x \in \mathbb{R}\ and\ l \in (1, \infty) \tag{26}$$

where l is a constant affecting the curve of the priority function. For experiments in this thesis, constant values were set in this way: l = 20 and $T_P = T_D = 60$ (minutes) by default. Figure 23 shows the development of both parts (*Planned* and *Delay* values) of the priority function of an activity.

The schedule manages interruptions among planned activities when ongoing less important activity collides with the planned start of an important one. Such interruptions are less common than ones caused by FCM-NAS simply because activities in scenarios are usually planned without overlapping. However, any delays within a dense schedule may lead to a shift of activities. That is handled by priority functions of activities, which are derived from the *Planned* and *Delay* values. For example, two planned activities are waiting in the buffer: long-lasting activity *working* and very short and important activity *take_medicine*. Table 10 shows the configuration of both activities, which defines their priority function over time.

Table 10: An example of the configuration of activities (source: author)

Activity	Duration*	Tolerance*	Start tol.*	Delay tol.*	Importance
1 working	120	30	60	60	0.6
2 take_medicine	4	5	10	10	1.0

*in minutes

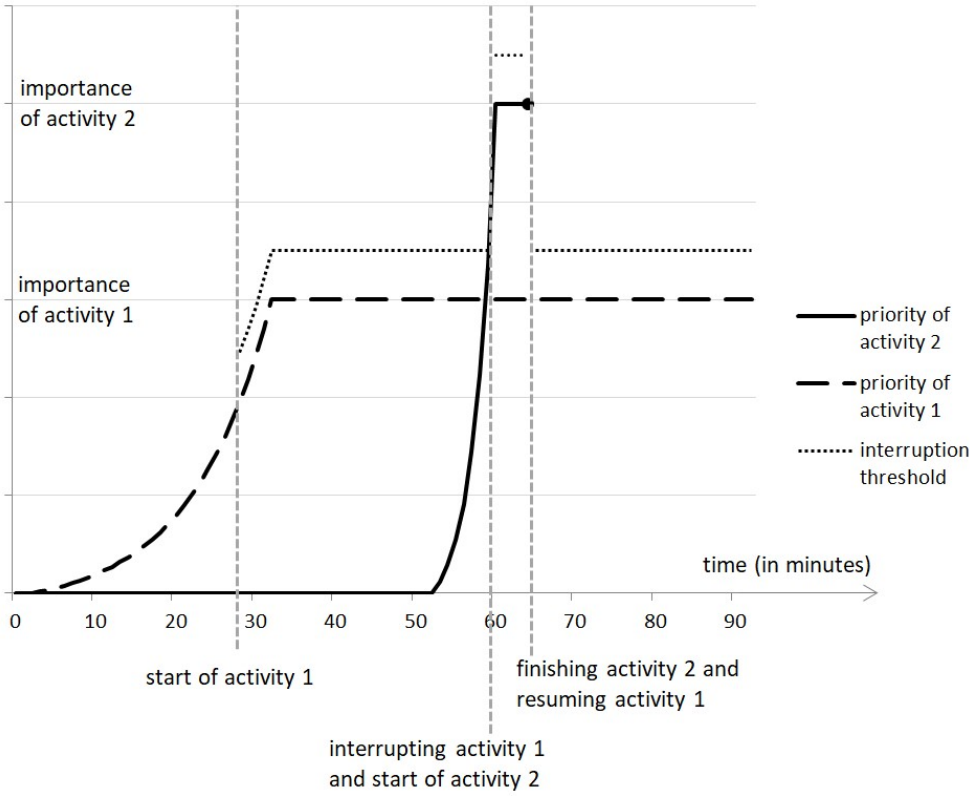

Figure 24: An example of interruption of an activity (source: author)

Figure 24 shows a time plot of priority functions over time. In this example, the first activity starts four minutes earlier. Ongoing activity then may have a priority boost by adding an interruption threshold, which other activities need to reach to be able to replace the current one. That encourages an agent to proceed and finish the activity without unnecessary switching back and forth between many activities. A similar interruption mechanism can be found in the HSS model [53]; although there it is used for need-oriented activities. An interruption occurs when the priority of the second activity crosses the interruption threshold. After finishing the second activity, it is removed from the buffer, and the interrupted activity is resumed. Note that the real duration of activities in the model also includes walking time needed to get to the appropriate location.

Interruptions initiated by FCM-NAS look similar way, except its more complex structure involves values of all concept nodes which have non-zero relation to activity nodes. Through the learning process, some activities may receive a similar boost (or decrease) as planned activities in the schedule management subsystem.

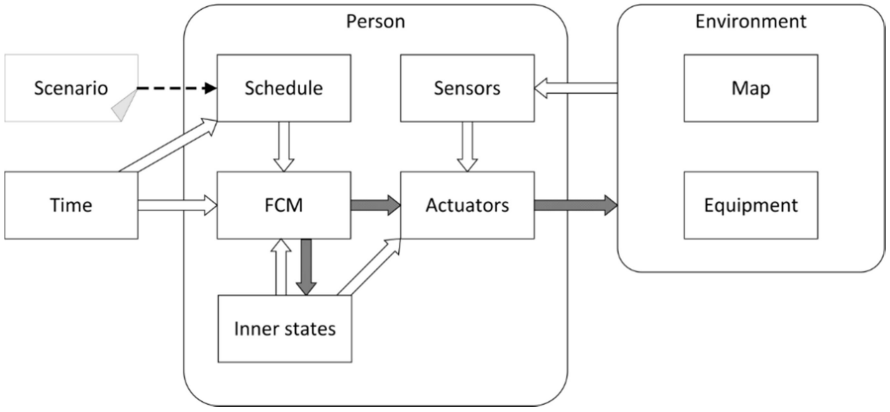

Figure 25: Diagram of the agent Person; white arrows mean direction of information flow, dark arrows represent the direction of control or influence. (source: author)

4.5 Behavior of Agents

The proposed design introduces the combination of FCM-NAS and routine behavior which creates a new problem: there are two independent decision-making processes that might want to control the behavior of a single agent in an incompatible manner. The task of the connection between these two processes is complicated. It requires one of the processes always has priority, or there has to be the third method to decide which one should be preferred at the moment. This work utilizes the first option. The preferred process has to be able to decide when it moves the responsibility to select an activity to the other process.

The model described in this work utilizes the FCM-NAS as the primary decision-making method for agents. This method simulates the needs of the person and selects appropriate activities, but it also receives information from the schedule whether there is any planned activity currently or shortly. If there is no immediate need to cover vital functions, then FCM passes selection of activity to the schedule. This model covers needs including hunger, thirst, fatigue, hygiene, bladder, boredom, and socializing need. This set is not limited in any way. Besides basic physical and psychological needs, there could be ambitions regarding work, hobbies, or family, for instance.

Figure 25 shows the structure of the agent Person. The white arrows represent a flow of information; the dark arrows express a direction of influence or control.

The dashed arrow depicts the import of the scenario into the schedule. To decide, the FCM gathers data from the schedule and inner states. The schedule returns values representing if there is an activity that should be in progress right now or is about to start soon. Inner states provide values of all needs and the activity performed last time. FCM is able to make higher-level decisions, what should the agent do. It selects a general group of activities (a context) and forwards it to actuators, which pick specific action regarding the current location or conditions. These lower-level decisions are made by simple rules which select a specific action and ensure the right position of the agent. For example, when the agent is supposed to eat, the algorithm checks the agent's level of the corresponding need (*Hunger*). If it is high, then the agent starts to cook a meal. Otherwise, it takes a snack. If the agent is not in a kitchen, it starts moving to the corresponding spot.

Figure 26 visualizes all levels of activities considered for the solution with examples. The activity context level is exclusive for the higher-level decision-making provided by the FCM-NAS. Some activities may have several required steps before the activity becomes available. These chains are logically arranged into higher-level activities. Scenarios may contain all required steps, or just the final activity and actuators make sure all conditions are met. For instance, if a scenario contains activity *eat_dinner* and a meal is not prepared, then actuators start the preparation of the meal first. Action level in Figure 26 represents atomic actions of activities; meaning each action is connected to a single device or position. Movement in the environment to a target location is considered as an action as well. The lowest level of the hierarchy represents animations; these are not implemented in the proposed model as it does not aim to precise physical interactions since they are not in the scope of this work. Note that even without animations; interactions with devices are implemented, just not visually. However, the addition of the animation layer would be possible as it is compatible with the design, although a more suitable platform would need to be selected accordingly.

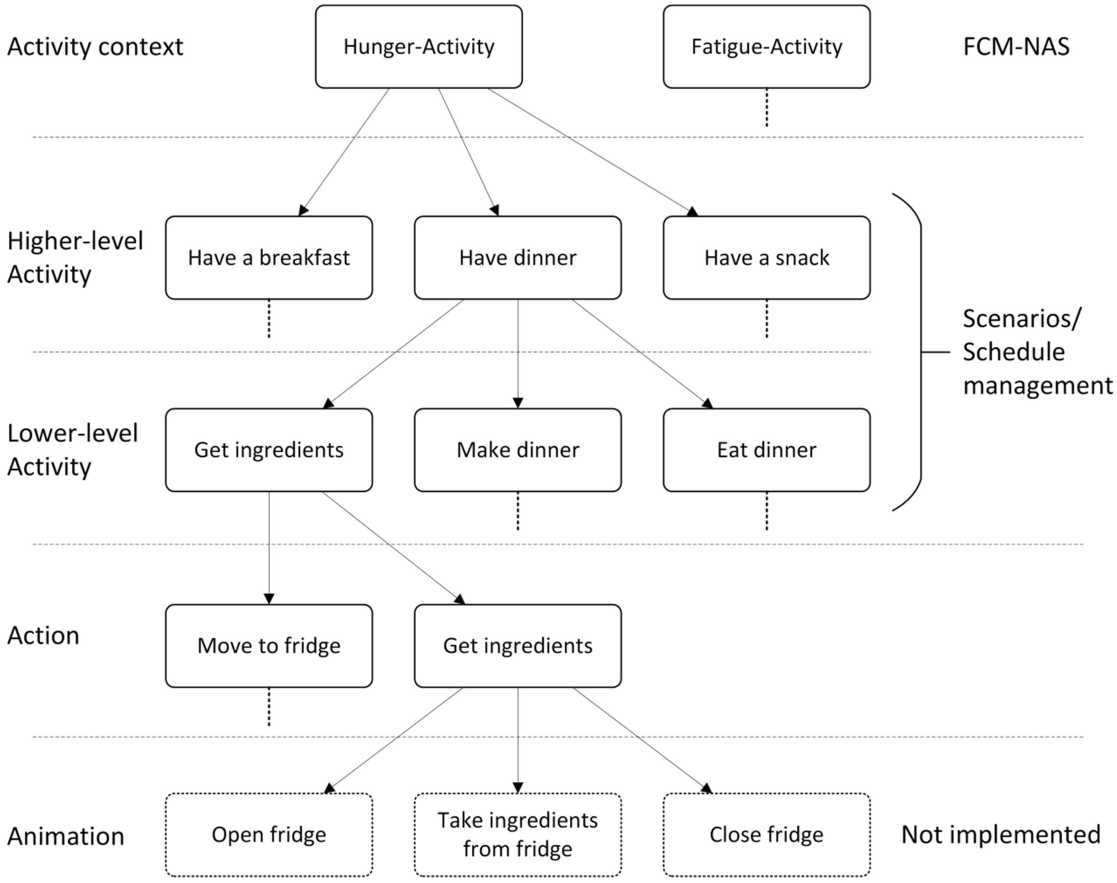

Figure 26: Hierarchy of activities with examples showing activities/actions related to having dinner. (source: author)

4.6 Genetic Algorithms

Genetic algorithms will be applied to the decision-making part of the FCM-NAS to adjust agents' behavior. Although it could use other less computationally expensive learning algorithms, there is a problem of immediate evaluation after the decision. When rating a complex behavior, a single decision cannot be evaluated properly as it may have consequences which might change events in the future. For instance, taking a meal seems like a bad decision for an agent that is not very hungry right now, but if he plans to go out for several hours later, then it is a good decision in a long-term perspective as it would get very hungry later on. Therefore, the agent's behavior is evaluated after a certain period of time. When the behavior is evaluated, its complexity still prevents effective interpretation of its problems. Fortunately, genetic algorithms can overcome this issue. The only major drawback

is the computational time needed to process sufficiently large populations through a sufficient number of generations.

Implementing genetic algorithms requires a lot of repetitions of the same simulation with different configurations of decision-making. One of the ways to implement this is running parallel simulations corresponding with the size of the population. This model launches the whole population in the environment for a set number of days. To avoid any interactions between agents and their perceived environment, agents need to record any possible changes to the environment to local variables, and the ambient intelligence system must be turned off during this learning process. These measures ensure the same conditions for each agent and guarantees that only the behavior alone affects its final score.

After a given simulation period, fitness function evaluates each genotype; meaning the configuration of decision-making in this case. The fitness function in the model does not only rate behavior, but in the case of genetic algorithms, the fitness defines the direction of evolution that forms the intended goal of the learning process. As the decision-making method covers two different concepts, its fitness function requires at least two components. The first fitness component evaluates the need-oriented part of behavior based on the average success rate of satisfying needs during the simulation. This is possible through the *Primary State* node c_p, which monitors all needs at each computation of FCM-NAS (see 4.2.4 Primary State, Fitness, and Constant Increments for detailed explanation). The first part of fitness F_F is the mean of values of the Primary State v_p^t node through a simulation:

$$\mathrm{F}_F = \frac{\sum_{t=0}^{t_{max}} v_p^t}{t_{max}} \tag{27}$$

The second component focuses on routine behavior and planned activities. It reflects how many of the planned activities were finished during the simulation. Because these activities have different importance, the evaluation is weighted accordingly. If the agent misses important activity, then fitness decrease more than in case of missed less critical activity. (28) shows the expression used to calculate this second component of the fitness function F_P.

$$F_P = \frac{\sum_{i=0}^{m} u_i^E}{\sum_{i=0}^{m} u_i^I} \quad (28)$$

where m is the total number of activities planned during the simulation including missed ones, the value u_i^I represents the *Importance* of activity u_i and u_i^E is the evaluation of its progress when it is removed from the schedule. The importance value is a constant of each activity given by the configuration of the scenario. The evaluation of activities u_i^E is a value ranging from zero (when an activity is skipped) to the importance u_i^I. This value is also adjusted according to the delay or early start of the activity. Then the ratio (u_i^E/u_i^I) represents the fulfillment of each activity (from zero to one). In other words, the second fitness component represents the average success rate of planned activities during the simulation weighted by the *Importance* of activities.

The early experiments revealed the shortcomings of the described evaluation. Both these fitness components lead to optimization of behavior corresponding to both decision-making parts; however, the learning process still did not produce effective and reasonable behavior. Time distribution on Figure 27b shows that agent tends to spend any free time between planned activities to keep needs satisfied. This behavior seems to be always alerted and repeatedly triggers activities even before there is the actual need. That is the reason why the third component of the fitness function was introduced. It motivates decision-making to be more effective with the limited time agents have each day. This fitness component F_T is represented by the percentage of the time when the agent was neither doing any planned nor need-oriented activity (denoted as t_{free}) out of total simulation time t_{max}:

$$F_T = \frac{t_{free}}{t_{max}} \quad (29)$$

Figure 27c shows that the learned behavior with this component is significantly more effective as it spends much less time to manage both needs and planned activities. Instead of using 58% of the time to reach to need-oriented fitness of 0.974 (Figure 27b), it spends only 8% of the time to reach the value of 0.963 (Figure 27c).

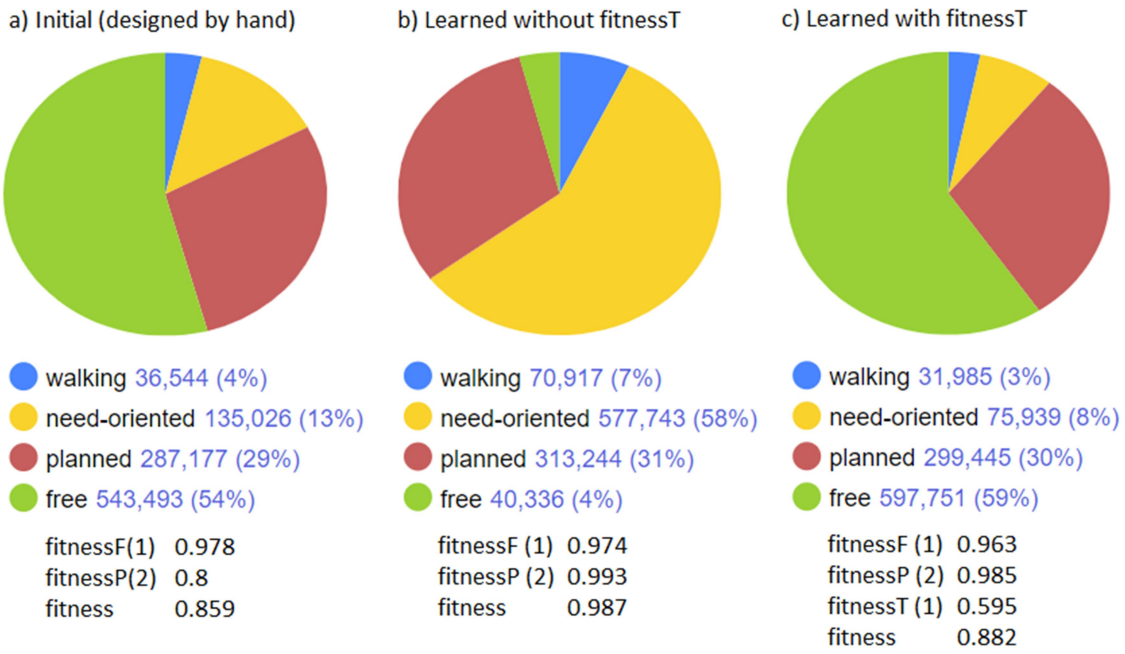

Figure 27: Comparison of fitness values and time distributions of initial behavior with two learned behaviors - with and without the third fitness component (source: author)

There are three components of the fitness function, but because genetic algorithms expect one value, they are all merged into the final fitness value as a weighted average of all three components (see (30)). All three values are inside the interval from zero to one and share the same orientation (0.0 is the worst, 1.0 is the best evaluation) which makes their combination into the final fitness value very easy.

$$F = \frac{\mathrm{F}_F \cdot \mathrm{W}_F + \mathrm{F}_P \cdot \mathrm{W}_P + \mathrm{F}_T \cdot \mathrm{W}_T}{\mathrm{W}_F + \mathrm{W}_P + \mathrm{W}_T} \tag{30}$$

Finally, after the fitness function is complete, the other parts of genetic algorithms require configuration. The selection in this model is based on the provided fitness of genotypes. A few of the best ones are cloned into a new generation; some of the worst are discarded and replaced by children of surviving individuals. The rest of the genotypes enter the lottery of crossover and mutation. Each of these is randomly paired with another one which ensures they are all used at least once. Chance of additional selection is dependent on the fitness value of a genotype. This model uses the uniform crossover that randomly collects individual genes from one or the other parent. According to [50], this type of crossover has the best abilities to explore possible configurations at the cost of slower convergence.

Mutation in this model randomly shifts some genes according to three variables: the rate, power, and step of mutation. The rate defines how many genes are affected by the mutation. Since genotype in the model is a series of truth values instead of binary ones, the mutation does not simply switch the value. Instead, the value is increased or decreased by a product of mutation step parameter and rounded random number from a normal distribution with a mean of 0.0 and a standard deviation equal to the mutation power parameter.

This model also experiments with the principles based on generalized frozen evolution [51], which examines the variation of a gene pool. Although it is a negative phenomenon in biological evolution, it may be useful for the convergence of the virtual population to the optimal solution. It is implemented into the model by adjusting mutation parameters during the learning process. In the beginning, the mutation is powerful and changes genes by significant steps. That means fast but risky learning with great variability and low chance of getting stuck in a local maximum of the fitness function. Later on, mutation parameters are softened to allow population easier convergence towards the best solution. Finally, parameters are focused on fine-tuning of the genes. At this point, evolution is unable to make any severe changes or switch to a different local maximum. However, it is assumed that previous learning steps have already chosen the best direction and now the learning process searches for the very best solution in that direction by tiny adjustments.

4.7 Multi-agent Support

Ambient intelligence models become a lot more complicated when there is more than one agent. Especially in concepts like smart home, users can have different preferences of environment that can be conflicting. Therefore, multi-agent support is an important feature in human simulation. This work focuses on individual-based decision-making, which means agents are autonomous units and as such the model is able to simulate a lot of them in parallel. Since the model includes genetic algorithms, two different multi-agent modes are required. The first one needs to ignore any interactions to support parallel isolated simulations for individual

evaluations during the learning process. The second mode supports two or more agents per single simulation.

Multi-agent support has several levels and perspectives. Agents' higher-level decision-making could be altered to acknowledge the presence of other agents and has possible interactions in the set of activities (as it is in ALModel [77]). Lower-level decision-making requires management of capacity of places or devices to avoid the parallel use of devices by more agents than it could support. For instance, at most one agent should be allowed to use a computer at any moment. In case other agents request to use the same device, they either wait or select a backup activity depending on the activity context. Figure 28 shows agents indirectly interacting within a single environment. The list on the right side of the screenshot shows activities that are currently performed by all seven agents. Note that the second person (by computer) is waiting for another agent to leave fridge area so he can also prepare his breakfast. Meanwhile, there are more agents watching TV because this device has a higher limit of concurrent users. Since the model does not use animations, all agents watching TV are clustered on a single node, this is why it seems like only one agent is watching TV.

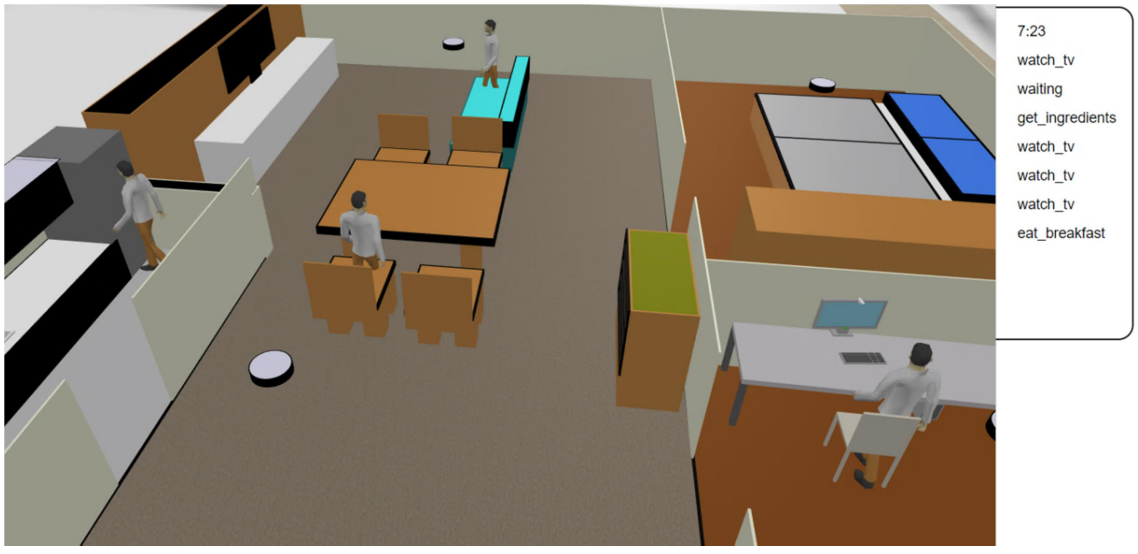

Figure 28: Screenshot from the model during the experiment with multiple agents in single simulation; each agent has an activity listed on the right side (source: author)

To use the multiple agents for testing their interactions with an intelligent environment, the intelligent system along with sensory sub-systems needs to be able to read correct data and recognize more than one behavior within the model. Unfortunately, the model in which the proposed human simulation was implemented does not fully support more than one agent. This causes that system treats some sensory data as they originated from a single source. However, to fully support multiple agents, the system would need to be redesigned. That part of ambience intelligence is out of the scope of this work. Nonetheless, the environment and agents are still able to interact in basic ways successfully. As proved even in ALModel [77], the FCM-NAS is generally able to support multiple agents in a simulation.

4.8 Implementation

This section will describe the implementation of some significant or platform-specific parts of the proposed solution. The designed human activity simulation was implemented in platform AnyLogic 8, which is optimized for agent-based systems. For the processes, besides object-oriented programming language Java, it offers state charts and action charts. Regarding simulation, this platform allows real-time simulation speed along with fast-forwarding speed limited only by computational capabilities of the hardware. The final model can be viewed in both 2D and 3D representation.

Table 11: Nodes of FCM-NAS used in the proposed model. (source: author)

Need nodes	Activity nodes	State nodes
Hunger (c_1)	HungerActivity (c_8)	PrimaryState (c_{16})
Thirst (c_2)	ThirstActivity (c_9)	Planned (c_{17})
Fatigue (c_3)	FatigueActivity (c_{10})	Delay (c_{18})
Boredom (c_4)	BoredomActivity (c_{11})	NearPlanned (c_{19})
HygieneNeed (c_5)	HygieneActivity (c_{12})	Nighttime (c_{20})
ToiletNeed (c_6)	ToiletActivity (c_{13})	Daytime (c_{21})
SocialNeed (c_7)	SocialActivity (c_{14})	
	PlannedActivity (c_{15})	

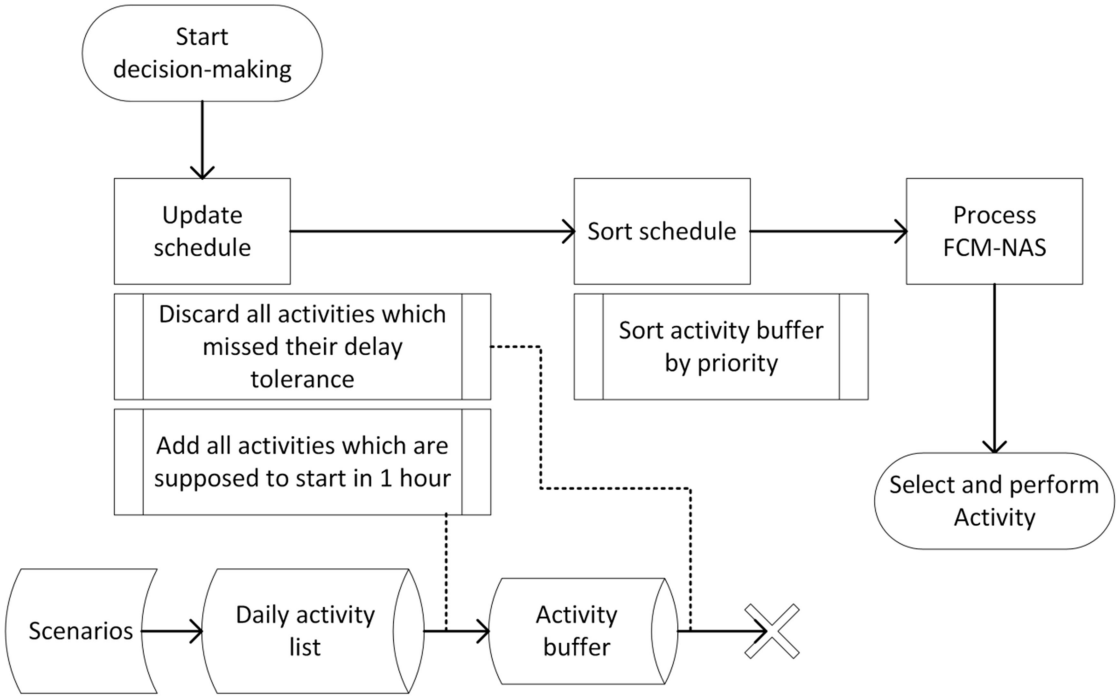

Figure 29: Diagram of the higher-level decision-making process. (source: author)

4.8.1 Implementation of Decision-making

The model triggers the decision-making of each agent every single minute. This interval can be arbitrarily adjusted thanks to the granularity parameter in FCM-NAS. This process starts with higher-level decision-making using FCM-NAS and schedule management. As seen in Figure 29, regardless of the FCM-NAS decision, the schedule always updates its activity buffer to check for upcoming or missed activities. Firstly, it checks all activities in the buffer for missed activities. There are two possible ways for activities to be missed. The first possibility is that the activity has not yet started and missed delay tolerance. The second one is the situation when the activity started but has been interrupted and missed its latest finish time. After discards, the schedule adds the upcoming activities that either start in one hour or have a longer start tolerance interval. Then the schedule management sorts activities by their current value of priority function, so if the FCM-NAS decides to process a planned activity, then the first one from the buffer is performed.

After the schedule management, FCM-NAS is computed. This process is not dependent on neither platform nor model; it has been described in detail in section 4.2.6 Processing FCM-NAS. The FCM-NAS in this model contains 21 nodes: seven *Need* nodes with seven corresponding *Activity* nodes, one additional *Activity* node for planned activities, *Primary State* node, and five *State* nodes for external inputs. Table 11 lists all nodes in the designed FCM-NAS. Agent's behavior covers the following needs: hunger, thirst, fatigue, boredom, hygiene need, toilet need, and social need. Each of these has a corresponding *Activity* node, which triggers one of the activities that are supposed to satisfy the need. The choice of the specific activity to handle the need is the responsibility of lower-level decision-making.

All *State* nodes except Primary State are external inputs. In this context, it means outside of FCM-NAS, not necessarily external for the agent as a whole. Values of nodes *Planned*, *Delay*, and *NearPlanned* are taken from the schedule as a sum of corresponding values of activities in the buffer (these values were described in section 4.4.2 Schedule Management). Values of states *Nighttime* and *Daytime* are global; meaning their values are the same for all agents in a model run. These values allow agents to include time in the decision-making of FCM-NAS. Other more precise variants would also be available. For instance, nodes *Morning*, *Afternoon*, *Evening*, and *Night* would potentially allow machine learning to better adjust behavior to day time. With more complex scenarios, nodes for distinguishing weekdays from the weekend would also be very helpful. Even if the designer does not see an obvious way how an extra node could change the behavior, machine learning uses every piece of information available, and it often finds the optimal way to include it in the decision-making. For this model, the FCM-NAS was designed with the least nodes possible to allow fast testing and learning. Otherwise, additional State nodes for external inputs have relatively small because its computation is omitted.

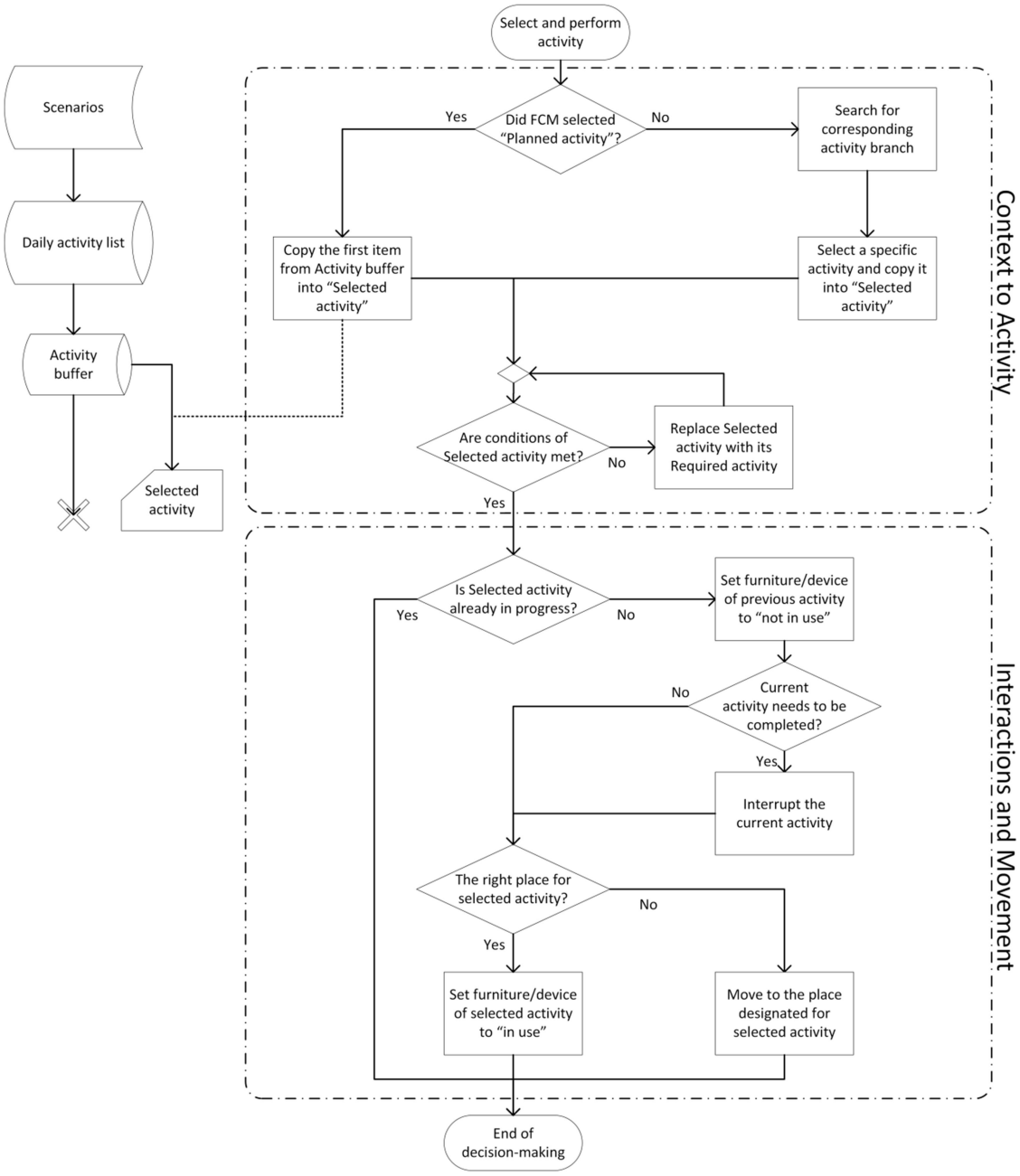

Figure 30: Lower-level decision-making process managing conditions and interactions. (source: author)

The outputs of FCM-NAS are updated values of all nodes and particularly one *Activity* node with value 1.0 signalizing the selected type of activity (context). This selected context is then processed by lower-level decision-making mechanism which is straight-forward algorithm checking conditions, managing effects, and interactions with the environment or eventually other agents. The whole process is shown in Figure 30. At the first stage, an activity is selected based on the current activity context. For example, if FCM-NAS chose *PlannedActivity*, then the first

planned activity in the buffer (the one with the highest priority) is performed. If the context is *HungerActivity*, then the algorithm picks the default activity linked to this context. That may be *Having dinner*, but it is then switched to *Having a snack* because the agent's *Hunger* need does not meet the conditions of the first activity. In case no context is picked, the agent starts to perform free time activities (watching TV, playing games, or hobbies).

At the second stage of the lower-level decision-making mechanism, the activity is translated to a specific action. It stops the interaction with the device that may have been in use by the previous activity. If this previous activity waits for its effect, then its timer is reset, and the activity interrupted. Next, the agent's position is checked whether it equals to the required position of the new activity, and the agent is eventually sent to that position. If the selected activity was already in progress, this section is skipped.

4.8.2 Environment

The original ambient intelligence model created during GAČR project DEPIAS is set in the 3D model of a standard apartment placed on a map with several buildings around. It was built on the template of map plan of the city Trencianske Teplice, specifically local spa resort. The reason for this placement is the initial motivation of this ambient intelligence model: possibilities of ambient intelligence system for guests of the spa resort. The apartment offers all elemental functionalities required for comfortable living (see Figure 31). It was fully described at [83] (not published yet) where the author of this work contributed with the design of the connection between agent's behavior and environment; meaning lower-level decision making and interactions with devices or furniture.

Since the experiments were conducted in this environment with just a few minor alterations, the dependency of the agent's behavior on the environment needs to be explained. The higher layer of decision-making does not rely on a configuration of the environment and disposition of rooms or furniture at all. The only variable of environment that enters this level of decision-making is time. FCM-NAS reads a truth value of *Daytime* and *Nighttime* be able to adjust the sleep cycle or other

time-dependent activities. Schedule management, obviously, needs the current time and day for its correct function. Since all ambient intelligence models are supposed to be close to reality, it can be assumed that all such models would follow real time and calendar. Therefore, this approach would work in any of these models the same way.

Agents' actuators, the lower layer of decision-making, handle the logic for the interaction of the agent with the environment, which consequently leads to some dependency on the environment. To filter out unnecessary details of the environment, decision-making perceives only a weighted graph (see Figure 32) where edges are possible paths with different lengths, and nodes are either intersections or locations assigned for activities.

Figure 31: Screenshot of the apartment in the AnyLogic. (source: author)

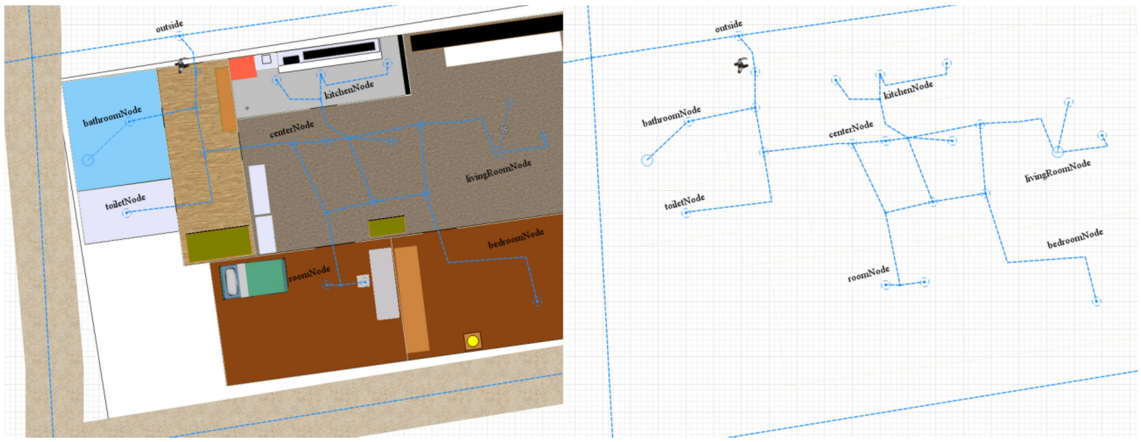

Figure 32: Weighted graph of the environment. (source: author)

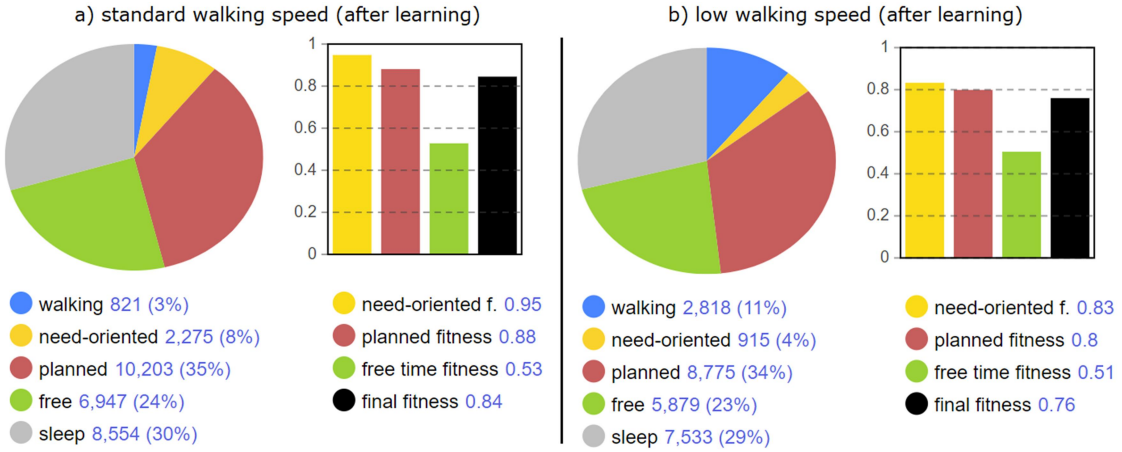

Figure 33: Comparison of fitness and composition of a day of two agents with different speeds after the learning process. (source: author)

Activities are bound to the corresponding locations by value in the configuration. Some activities are missing this value, meaning it can be performed everywhere; for instance, the agent can drink anywhere as long as it has a drink in its inventory. By default, the decision-making process assumes that the target location of every possible activity exists somewhere in the environment. At least one of all types of locations should be present in the environment (including outside of the apartment). If there are more instances of such locations (for instance chairs by table), it searches for the closest one usable by the agent.

The only influence the environment has on the behavior is by the distance between nodes that an agent needs to walk; therefore, the starting time of the activities may vary. In the current apartment in the model, the maximal distance between two nodes is around 23 meters. Even if the agent is set very slow with speed of one meter per second, it can easily move to any location within one minute, which is the period between decision-making computations. In extreme cases, an environment with significant walking distances could have an impact on the agents' evaluation since the walking would cost; however, the decision-making still works and adjusts for this cost. Figure 33 shows an experiment comparing the same agent within the same environment, scenario, and configuration with the exception of walking speed. After learning with genetic algorithms, the ten times slower agent (Figure 33b) managed to achieve a reasonable schedule and need-oriented fitness values. Of course, due to very long walking times between nodes

(up to 5 minutes of simulated time between rooms inside the apartment) evaluation was reduced by delays or starting too soon because the agent had to start earlier to compensate for extreme walking times between planned activities.

4.8.3 Randomness and Repeatability

Both decision-making mechanisms are deterministic, meaning randomness is not involved in the process of making decisions. However, the model uses some randomness during the creation of a daily schedule from a scenario. The first set of random values, which is generated from a positive uniform distribution, is used for all activities with the probability of their appearance in a daily schedule lower than 1.0. Also, the duration of activities is randomized at the moment when an activity is added to the buffer. The parameter *Max multiplier* in the configuration of activity defines the possible variation of its duration. This way, the same activities may take different durations emulating variability in human behavior. Finally, the second area affected by randomness is the learning process: genetic algorithms during the selection, crossover, and mutation. All parts use a uniform distribution, and mutation uses a normal one for the random adjustment of genes.

Although randomness offers desired variability and tests the flexibility of decision-making, the design needs to consider the repeatability of experiments and fairness during learning processes. The first issue, repeatability of experiments, is covered by a built-in pseudorandom number generator (PRNG). The AnyLogic platform allows users to edit the seed value of PRNG, which ensures the same numbers are generated every model run with the same seed. The second problem requires a specific solution for genetic algorithms though. If a scenario contains activities with the *Probability* parameter lower than 1.0, then different random values will cause individuals of the same population having different schedules. Therefore, the evaluation would be distorted by the difficulty of a generated schedule. Meaning better behavior could have lower fitness than worse ones that received a schedule that is easier to complete (with less or shorter activities). To prevent this, a list of random numbers is generated in advance (using built-in PRNG). Individuals then obtain the same values in the same order, hence only their decision-making directs their final behavior and consequently their fitness.

5 Results

This section offers possible outputs of the behavior along with separate tests of both behavioral sub-systems. Then, the results of computational performance tests and findings from evolutionary learning are presented. Its achieved computational performance allows simulating virtual years in minutes. Experiments with genetic algorithms demonstrated its ability to change behavior in the desired direction.

5.1 Need-oriented Behavior Only

The HSS model uses only need-like mechanisms to trigger activities, which corresponds to the proposed model without a scenario. Values representing needs grow and periodically trigger activities satisfying corresponding needs throughout a day. Figure 34 depicts patterns of needs during a single day in both simulations. Note the similarity of the slower growth during sleep.

Besides usual needs, it has similar mechanisms for some other activities like using a computer, shopping, exercise, and others. Such activities would be otherwise in a scenario. Through HumanDesigner application (available for download at [72]), the HSS model allows manually adjust some parameters of behavior like the growth of needs and length of some activities. FCM-NAS offers the same adjustment through relations of a constant node (in this case the *Primary State*) to the corresponding need. Besides that, the proposed approach allows user to set different priorities or a critical level of needs to trigger the connected activity. Some of these parameters are also adjustable by genetic algorithms.

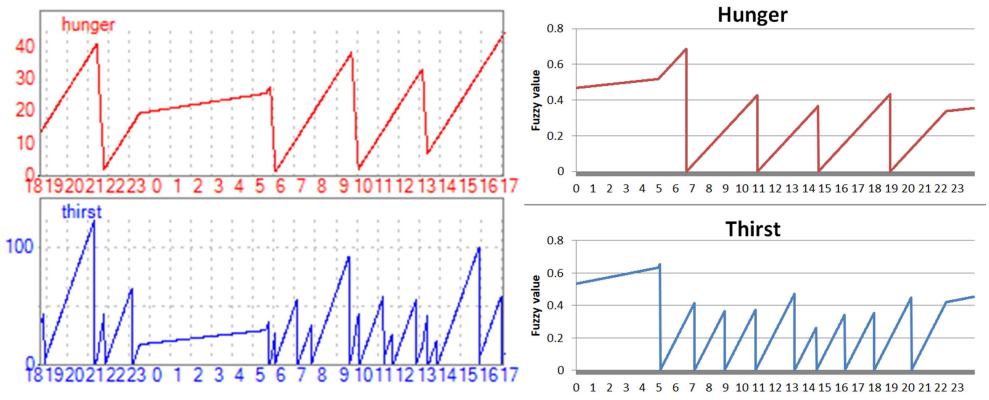

Figure 34: Comparison of needs in 1 day in HSS and the proposed model. (source: [72] and author)

5.2 Routine Behavior Only

To test the routine behavior only, FCM-NAS was set to ignore all need inputs and to trigger only the planned activity context. Activities were set with less flexible time frames when they are supposed to start and end to imitate a straight-forward set of activities triggered by time. This way, the proposed model is able to execute any hand-crafted schedule of activities. For comparison, the scenario was configured to imitate the short activity recording by PerSim3D presented in [71] (see Table 13). Table 14 shows almost the same recording as the original with tiny deviations caused by different environment and primarily by the flexible connection between FCM-NAS and the schedule management. Since PerSim3D aims for precise detection, unlike the proposed model, it supports animations and more specific sensors like pressure, vibration or radio-frequency identification (RFID) sensors.

Table 12: Sample output of PerSim3D from execution of scenario in beginning of the day. (source: [68])

Activity	*Start time* *End time*	*Location*	*Sensors*
Sleeping	7:30:30 7:35:22	Bedroom	Pressure sensor Vibration sensor
Using a toilet	7:35:25 7:36:10	Bathroom	Contact sensor
Making hot tea	7:36:30 7:40:35	Kitchen	Pressure sensor, RFID Temperature sensor
Having Breakfast	7:41:40 7:47:15	Dining Room	Pressure sensor RFID

Table 13: Output of the model during the experiment with scenario only. (source: author)

Activity	**Start/End time**	**Location**	**Sensors**
sleep	7:29:18	Bed	Movement (Bedroom)
	7:34:13		Movement (Center)
toilet	7:34:28	WC	Movement (Toilet), Use sensor (toilet)
	7:36:08		Movement (Hall)
make_breakfast	7:37:03	Kitchen	Movement (Kitchen), Use sensor (fridge)
	7:41:03		Movement (Center)
eat_breakfast	7:41:13	Table	Movement (Center)
	7:48:03		Movement (Living room)

5.3 Combination of Need-oriented and Routine Behavior

Experiments were conducted to test the behavior of the agent Person with scenario shown in Table 14. All the agent's activities with corresponding starting times were recorded during the day. The person was performing planned activities and satisfying needs for a few simulated weeks in the model of ambient intelligence. Since the person starts days with different inner values, the final order and starting times of activities may differ each day although the scenario was always the same. Table 15 shows the course of one day in comparison with the scenario introduced in Table 14. Most of the activities were completed, some on time, few of them later. For example, afternoon the agent had no time to continue read news because he had to make dinner. Later evening, the agent was tired, so he rested ignoring less critical activity in the schedule (playing games). However, he was able to return to it later.

Table 14: Scenario tested in the model. (source: author)

Start time	Activity	Importance
6:00	toileting	0.95
6:15	medicine	1
6:30	exercise	0.7
6:45	medicine	1
6:50	shower	0.9
7:00	eat_breakfast	0.95
7:10	read_news	0.8
7:20	medicine	1
9:30	massage	0.9
10:30	swimming	0.9
12:30	eat_dinner	0.9
16:00	watch_tv	0.3
17:00	read_news	0.3
17:30	health_practice	0.8
18:00	eat_dinner	1
18:50	medicine	1
19:00	go_out_exercise	0.8
20:00	work	0.5
21:00	play_games	0.3
22:30	drink	0.8
22:45	toileting	0.95
23:00	sleep	1

Table 15: Recording of the actual behavior of the agent in one day and its comparison with the scenario. (source: author)

Time	Activity	Schedule comparison	Time	Activity	Schedule comparison
5:47	toileting	13 min early	14:48	get_drink	not planned
5:58	medicine	17 min early	14:51	drink	not planned
6:03	take_snack	not planned	14:53	take_snack	not planned
6:05	eat_snack	not planned	14:55	eat_snack	not planned
6:11	watch_tv	not planned	15:01	watch_tv	59 min early
6:23	exercise	7 min early	16:27	get_drink	not planned
6:39	medicine	3 min early	16:30	drink	not planned
6:44	get_drink	not planned	16:32	watch_tv	resumed
6:47	drink	not planned	17:16	read_news	16 min late
6:49	watch_tv	not planned	17:17	health_practice	13 min early
7:00	eat_breakfast	not hungry (skipped)	17:41	read_news	resumed
7:00	read_news	10 min early	17:49	get_ingredients	required act
7:05	medicine	15 min early	17:55	make_dinner	required act
7:10	use_toilet	not planned	18:14	missed activity:	
7:11	watch_tv	not planned		read_news (3 minutes left)	
8:23	get_drink	not planned	18:16	eat_dinner	16 min late
8:26	drink	not planned	18:32	get_drink	not planned
8:28	watch_tv	not planned	18:35	drink	not planned
9:28	massage	2 min early	18:37	use_toilet	not planned
10:29	swimming	1 min early	18:38	medicine	12 min early
11:29	get_drink	not planned	18:43	go_out_exercise	17 min early
11:33	drink	not planned	20:25	work	25 min late
11:35	get_ingredients	required act	21:01	get_drink	not planned
11:41	make_dinner	required act	21:04	drink	not planned
12:02	eat_dinner	30 min early	21:06	rest	not planned
12:18	use_toilet	not planned	22:14	play_games	74 min late
12:19	watch_tv	not planned	22:29	get_drink	required act
13:09	get_drink	not planned	22:32	drink	2 min late
13:12	drink	not planned	22:34	play_games	resumed
13:14	wash_hands	not planned	22:42	toileting	3 min early
13:40	use_toilet	not planned	22:53	play_games	resumed
13:41	watch_tv	not planned	22:58	sleep	2 min early

Table 15 shows output data originating directly from the agent Person. Obviously, the system controlling the smart environment does not have those outputs. It depends only on its sensors, and there is no direct communication between agents and the system even in the model. The log of sensors is much more detailed than the record of the person's activities. Hence Table 16 contains information available to the system during only the first hour of the same day.

Table 16: Output data of the intelligent system from the first hour of the example scenario. (source: [83])

Time	Type	Location	Action
5:47:08	Move	Center	Person is caught by Center
5:47:13	Move	Hall	Person is caught by Hall
5:47:18	Move	Bathroom	Person is caught by Bathroom
5:48:03	On/off	Bathroom	Bathroom sink was turned on.
5:58:03	On/off	Bathroom	Bathroom sink was turned off.
6:03:03	Move	Hall	Person is caught by Hall
6:03:08	Move	Center	Person is caught by Center
6:03:13	Move	Kitchen	Person is caught by Kitchen
6:04:03	On/off	Kitchen	Fridge was turned on.
6:05:03	On/off	Kitchen	Fridge was turned off.
6:11:03	Move	Center	Person is caught by Center
6:11:13	Move	Living room	Person is caught by Living room
6:12:03	On/off	Living room	TV was turned on.
6:39:08	Move	Center	Person is caught by Center
6:39:08	On/off	Living room	Person is in a different room than TV which was turned off by House System.
6:39:13	Move	Hall	Person is caught by Hall
6:39:18	Move	Bathroom	Person is caught by Bathroom
6:44:03	Move	Hall	Person is caught by Hall
6:44:08	Move	Center	Person is caught by Center
6:44:13	Move	Kitchen	Person is caught by Kitchen
6:49:03	Move	Center	Person is caught by Center
6:49:13	Move	Living room	Person is caught by Living room

As both tables describe the same day, it is easy to compare what the intelligent system knows about the agent's activities and what the agent actually did. For example, at 6:11 agent Person decided to watch TV, at 6:12 TV was turned on. At 6:23 the person stopped watching TV and start exercise. Since both activities are in the same room and person forgot to turn off the TV, the system did not notice any change until 6:39 when the person left room (with the TV still on). Then the system turned off TV because no one was watching it. That is just an example of how the output of the ambient intelligence model could look. The model uses basic sensors partially to avoid privacy issues which often smart environments need to deal with. However, it is not the objective of this work to further explore the possibilities and limitations of such a system.

5.4 Computational Performance

In the field of multi-agent models, computational complexity is a serious concern if a model seeks to process a large number of agents fast. Although the final output of this model depends only on one agent or small group of them, lower computational complexity is still a beneficial achievement because of possible future enhancement of the model and computationally demanding learning algorithm. Growing the population of agents is often the highest performance hit because it multiplies all processes of the agent. Genetic algorithms do exactly that: generally speaking, the larger population, better the results of evolution.

Beside standard optimization of algorithms, the effort to increase simulation speed in multi-agent systems also focuses on the reducing of processing load on individual agents and pre-computing the data which are supposed to be used often by agents. In [84], thousands of agents drive through the map. The greatest challenge for this project was path-finding. Certain approaches are sufficient as long as the population stays relatively small, but their simulation time grows fast along with the number of agent or complexity of their environment (see Figure 35). The best solution for this particular model started with simplification of the environment from a detailed Geographic information system (GIS) map to a weighted graph. Then the best-proven strategies kept already computed paths and then just distributed results, so agents never process the same path again.

For the purpose of the human simulation, the environment in the proposed model was also simplified to a weighted graph. Although pre-computation of paths is possible, it is not implemented in this model because the number of agents does not reach thousands like in [84], and primarily the overall complexity of behavior makes path-finding costs insignificant. Instead, agents use built-in AnyLogic path-finding algorithm (A* by default), which is sufficient for this purpose.

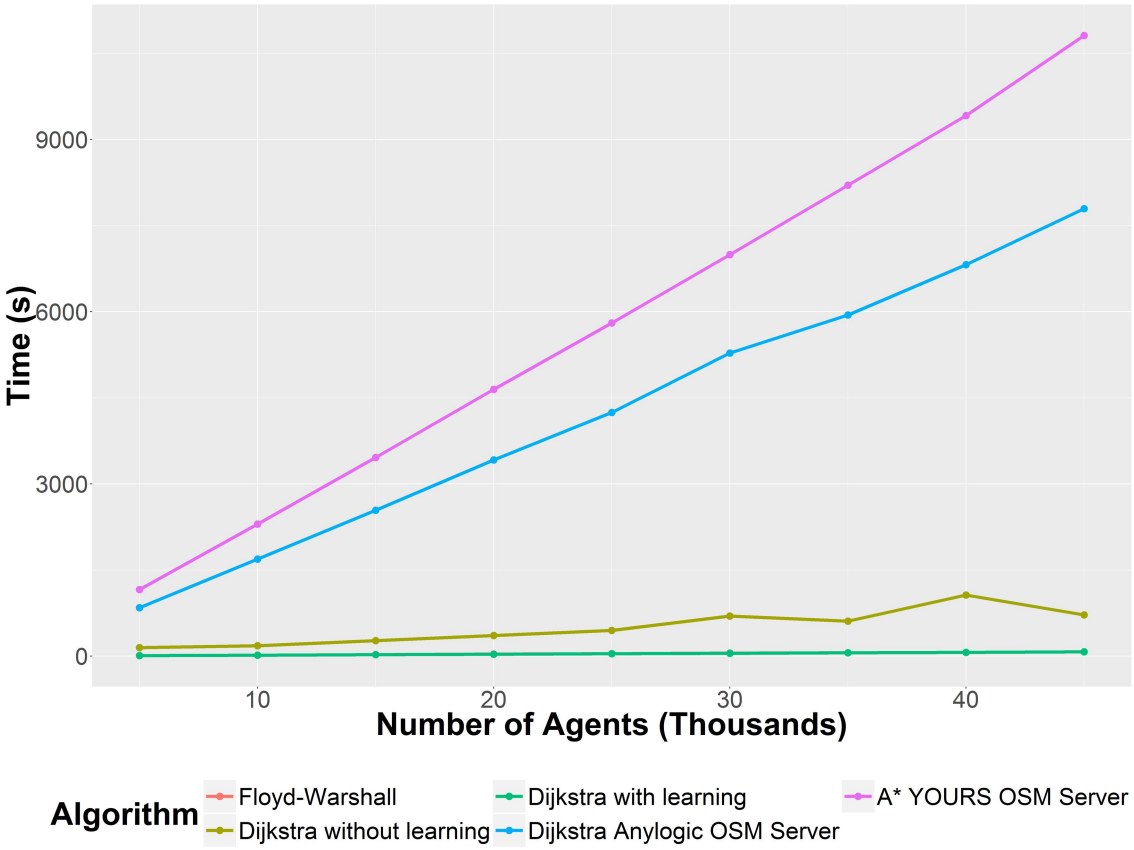

Figure 35: Mean runtime performance of algorithms. (source: [84])

To further decrease the computational time of the simulation, some reductions in the processing load of agents were used during the implementation of the model. Any variable or function which is valid globally through the model should be placed in the main class of the model. This way, agents request the value they need instead of processing it individually each time. For instance, the truth values describing the time of day are at any moment the same across the whole model; therefore, agents take the value from the global variable instead of calculating it themselves. A performance test was conducted to compare the impact of the location of timers on the simulation speed. At the first case, agents had own timers individually; the later version uses global timers which trigger corresponding actions for each agent. Although this change undoubtedly removes some unnecessary operations, its effect on the simulation speed has been statistically insignificant. Therefore, performance optimization efforts were redirected to the decision-making process.

The core of FCM-NAS is the multiplication of a vector with a matrix. Despite this calculation being divided by different concept classes, its time complexity is the same. Increasing the number of concept nodes means more possibilities. However, adding a new node to an FCM with c concept nodes raises its number of operations by $2n$+1 per computation. The time complexity of multiplication of $1\times n$ vector and $n\times n$ matrix is $O(n^2)$. Reduction of its complexity is undoubtedly possible: there are loss and lossless methods to do so. Beneficial lossless optimization is the utilization of the specific structure of FCM-NAS and different purposes of nodes to skip some sections during computations. If an FCM-NAS contains nodes of external inputs, the whole this section is omitted because these values are inserted from the outside; hence no other node can influence it.

Experiments with other modifications were conducted to decrease the time complexity of FCM-NAS even further. Multi-criterion decision support method AHP was used for the decision-making part of FCM-NAS. Although AHP has the same complexity as it processes matrix and vector multiplication as well, it spreads time complexity over two smaller sets: $O(n'^2)$ and $O(h_a h_c)$, where n' is the number of nodes in a new smaller FCM-NAS, h_c represents the number of criteria and h_a the number of alternatives in AHP. In [77], there was proven that the final number of operations is always lower this way because the following statements are always true:

$$n' + h_c < n \tag{31}$$

$$h_a < n \tag{32}$$

$$n'^2 + h_a h_c < n^2 \tag{33}$$

It proved to significantly decrease processing time of FCM-NAS depending on the size of the FCM. For the relatively small matrix, 17 concept nodes, computational time of this method is less than half of the full FCM-NAS approach (54% decrease). Figure 36 shows dependency on the size of FCM-NAS: for example, huge FCM-NAS with 180 nodes would be about five times faster with AHP modification.

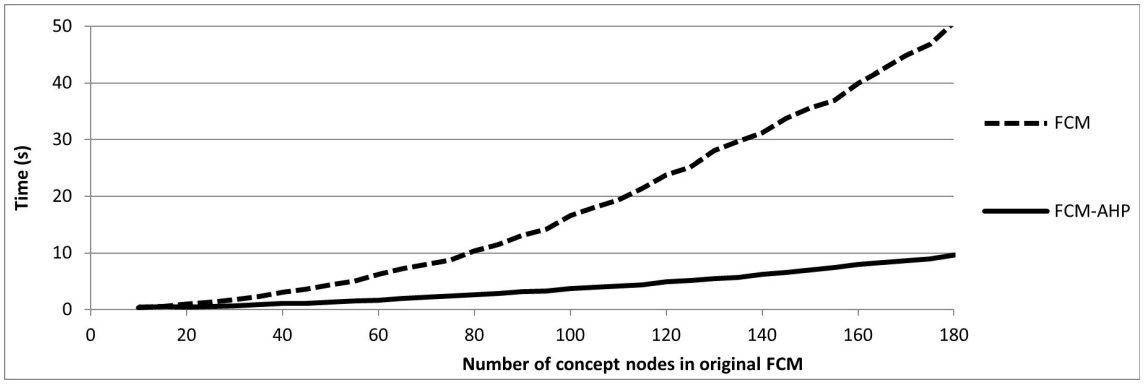

Figure 36: Performance comparison of FCM-NAS and its modification with AHP: the dependency between the number of concepts and the time needed to perform a million computations. (source: [77])

The main drawback of this method is that fewer values enter the decision-making process. The tested artificial life model in [77] with 17 concept nodes had initially 136 values forming agents' decision-making. After the combination with AHP, this number has been reduced to 20 which considerably limits the learning capabilities of agents. This simplification resulted in slightly less effective decision-making: 4.4% average decrease of fitness in the model. Since the presented ambient intelligence model has significantly fewer agents to process and specific demands of human behavior, full FCM-NAS matrix with lossless optimization only is used to process decision-making. However, if the scope of this model grows or its focus does not require such detail, then the replacement of its decision-making part with AHP will be a viable option to improve computational complexity.

The next important area of computational performance is the cost of agents' operations in the model. This includes the lower-level activity selection, movement through the environment, interactions and even processes of the intelligent system with its sensors. The high rate of processing agents' logic (under 1 millisecond per agent) prevents reliable measurement of individual parts of the simulation. Fortunately, the model allows disabling specific parts of behavior to compare it with the standard configuration. This technique will help to estimate a balance of computational demands of the proposed decision-making approach and the other operating costs of the simulation.

Different configurations of agents' behavior were tested while decision-making processes kept the same demands (the same size of the FCM-NAS and the same

scenario). A blank decision-making configuration, which does not trigger any activity, reached the rate of over 100 000 processed agents per second (see Figure 37). It is about three times higher rate compared to a standard one. On the other hand, the behavior intentionally configured to change activities every minute, which keeps agents constantly walking in the environment, has this rate 47% lower than a standard one. Conducted performance experiments show that the simulation speed is highly dependent on the intensity of agents' behavior. In most cases, operational costs limit this model by greatest contribution regarding computational demands. The experiments also shows that processing decision-making including the schedule management is sufficiently fast even in the worst case scenario.

All the previous results regarding computational performance were run on the same system: PC1 described in Table 17. Devices listed in Table 17 were used to test the model on different hardware configurations. Figure 38 shows the average rate of processed agents per second on all these devices.

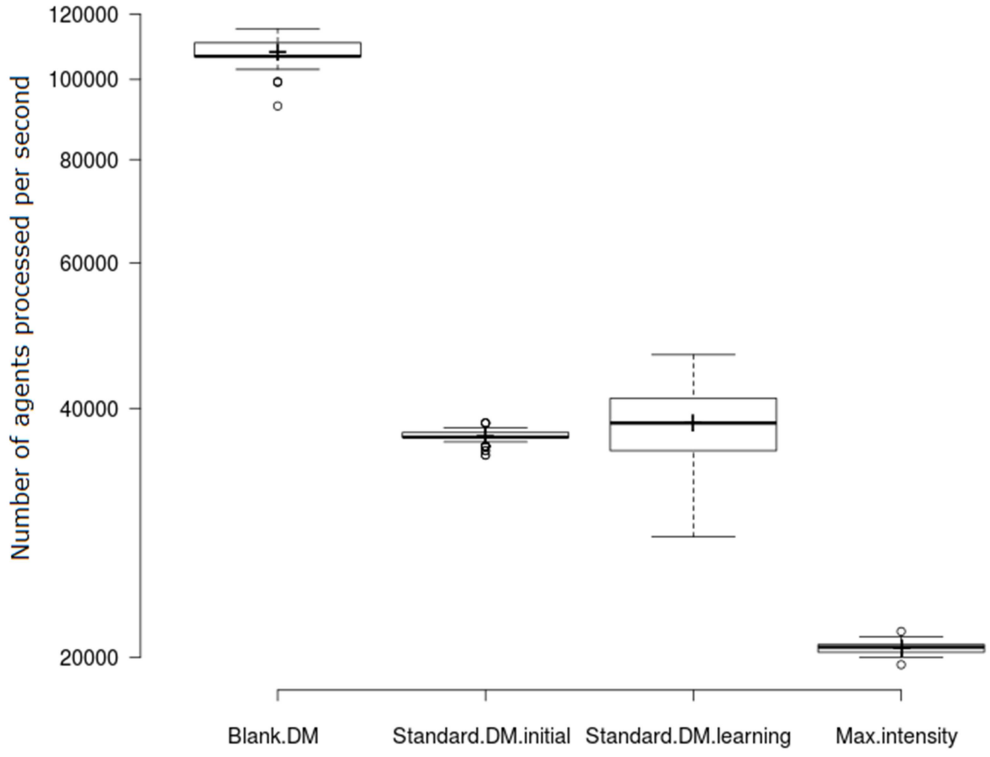

Figure 37: Comparison of processing rates of different behaviors. (source: author)

Table 17: Specification of testing devices. (source: author)

Label	Operation system	Processor (cores, max frequency)	RAM	Model performance*
PC1	Win 10 (64-bit)	i7 8700 (6 cores, 4.6 GHz)	16 GB	37 113
PC2	Win 7 (64-bit)	i5 3210M (2 cores, 3.1 GHz)	8 GB	22 245
PC3	Win 7 (64-bit)	i3 530 (2 cores, 2.93 GHz)	8 GB	11 571

All devices used AnyLogic 8.3.3 with Java 9.0.1

*Average number of processed agents per second – with a standard (default) configuration

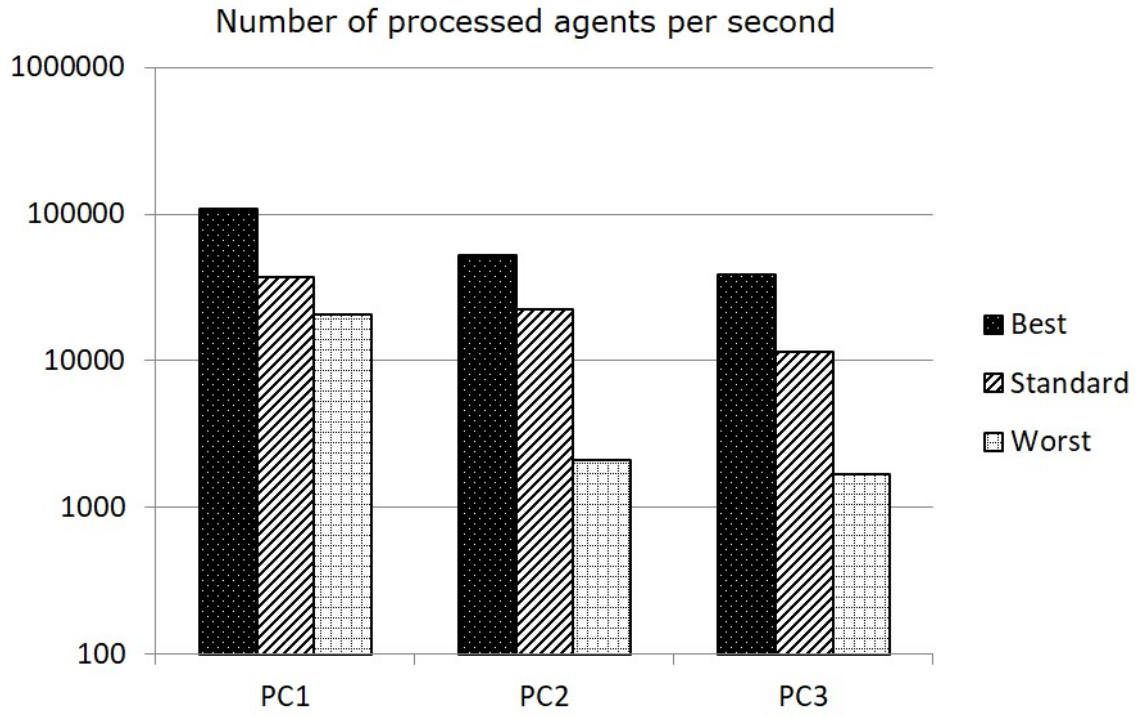

Figure 38: Comparison of average simulation speed on different hardware configurations. (source: author)

The performance of the proposed approach allows a smooth run of hundreds or thousands of agents while keeping complex behavior. Its modularity also enables to modify this approach for either large scale simulations with tens of thousands of agents or very complex behavior with over a hundred concept nodes. In the ambience intelligence model, its simulation speed is more than sufficient by a large margin. Learning through genetic algorithms is the only situation with a large population within the model. The primary use of this model allows simulating very long periods of time in a few seconds. For example, the model on PC1 with one agent go through one year in 35 seconds on average (80 seconds on PC3).

5.5 Machine Learning

During the learning process, the agent's behavior improves generation by generation in pursuit of improvement of the fitness. Many configurations were tested to balance parameters of evolution and weights of three parts of the fitness function. The ability to design initial decision-making by hand (which would not be possible with neural networks) proved to be very useful and time-saving concerning the use of genetic algorithms because it allowed starting with evident and important relations from the beginning. Otherwise, when learning every relation from scratch, it takes significantly more generations to uncover and reinforce essential connections of needs and appropriate activities with this complex behavior.

Generally, configurations with lower variability get easily stuck in a local maximum of fitness function by triggering activities by wrong inputs. In this situation, agents tend to transfer responsibility for decisions to schedule management since triggering it is easy to learn and the reward for its scenario completion is high. Then, planned activities are covered, but the learning process struggles to discover ties between needs and their corresponding activities. Figure 39 shows such a situation. Even after one thousand generations, genetic algorithms did not manage to teach agents to handle needs, and it optimized behavior only according to two other components of fitness function (schedule and free time). A schedule containing some need-oriented activities (for example planned dinner) further confuses learning mechanism. For this and following experiments with machine learning, the sample scenario described above in Table 14 was used. To prevent that, either initial by-hand design is needed to guide the population in the right direction, or genetic algorithms require different configuration. The latter option may be higher selection pressure towards desired behavior or high variability (meaning more pronounced mutation) for significantly more time to properly explore possibilities.

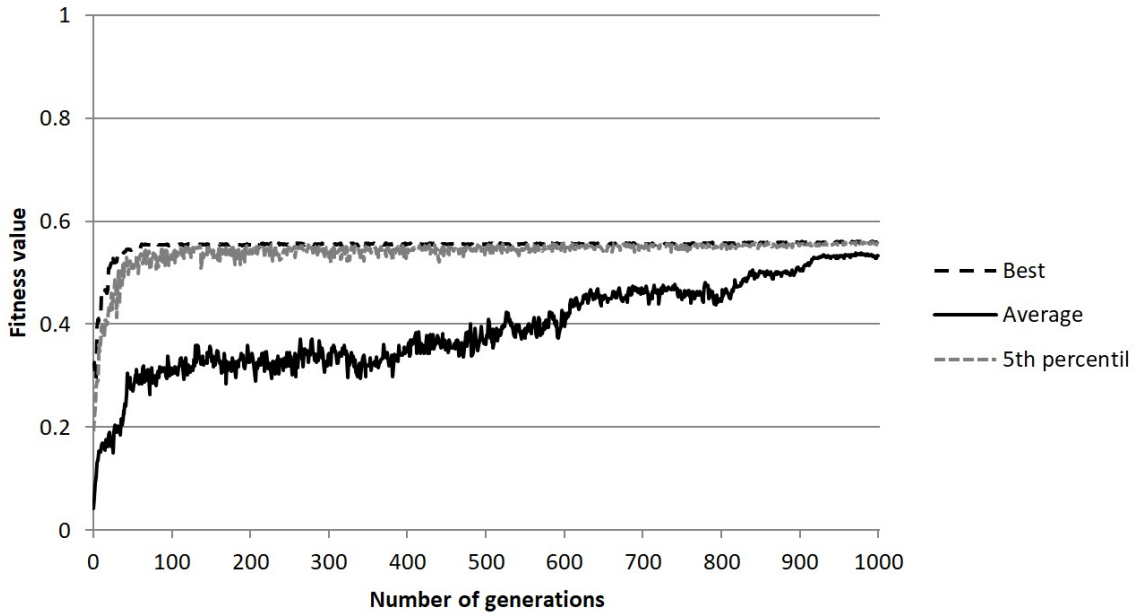

Figure 39: Development of fitness during genetic algorithms getting stuck at local maximum. (source: author)

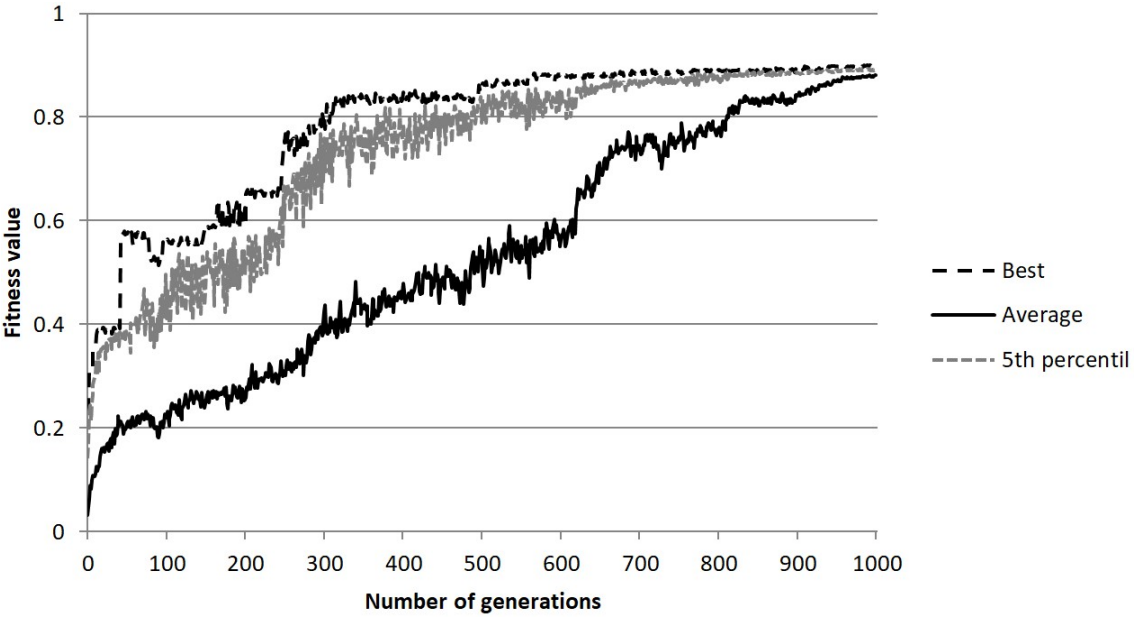

Figure 40: Development of fitness during genetic algorithms with higher selection pressure towards needs. (source: author)

Figure 40 shows another attempt with the same parameters except weights of the fitness component evaluating needs. This time it was set higher than the schedule component leading to higher selection pressure towards needs satisfaction. This configuration managed to overcome difficulties of learning complex behavior from blank decision-making. However, since the pressure for satisfying needs was the strongest, the final behavior tends to ignore the weaker free time fitness

component. Which leads to spending most of the time on needs despite their low level; hence the behavior is less effective.

Figure 41 shows the development of the fitness value during the experiment that started with an initial by-hand designed matrix of relations *R*. This way, FCM-NAS starts the learning process in a basic operational state. This initial configuration is not optimized to recognize priorities of different needs and does not have proper sensitivity to switching behavior between need-related and planned activities. However, all needs are connected to corresponding activities saving genetic algorithms a lot of time, so it starts with the optimization of these values instead of searching for them. Notice that the fitness value in the experiment on Figure 41 converged to a very high value in a much shorter time in comparison with the previous experiments starting from scratch.

The configuration of genetic algorithms is an essential step in creating desired behavior. Some values significantly influence the final behavior or the duration of learning. Firstly, the fitness function with its three components pushes evolution in the desired direction. As the best balance of weights of this three components proved to be as follows: weight of schedule component W_P at 1.5; the weight of need-related fitness W_F at 1.0; and weight of free time component W_T at 0.5. It is recommended to keep planned activities more important than free time; otherwise, it might simply skip some longer activities with lower priority just to save time. For the configuration of parameters of mutation, the model uses dynamic values gradually lowering variability. In the beginning, mutation is pronounced to randomize genes and spread across a wide range of possibilities. After this phase, every 200 (or 400 depending on the speed of convergence) generations, mutation softens and makes smaller adjustments of values to find a more precise solution. The size of steps which are genes modified with starts at 0.1; at the end, this size was set to 0.01, which determines the final precision of the solution.

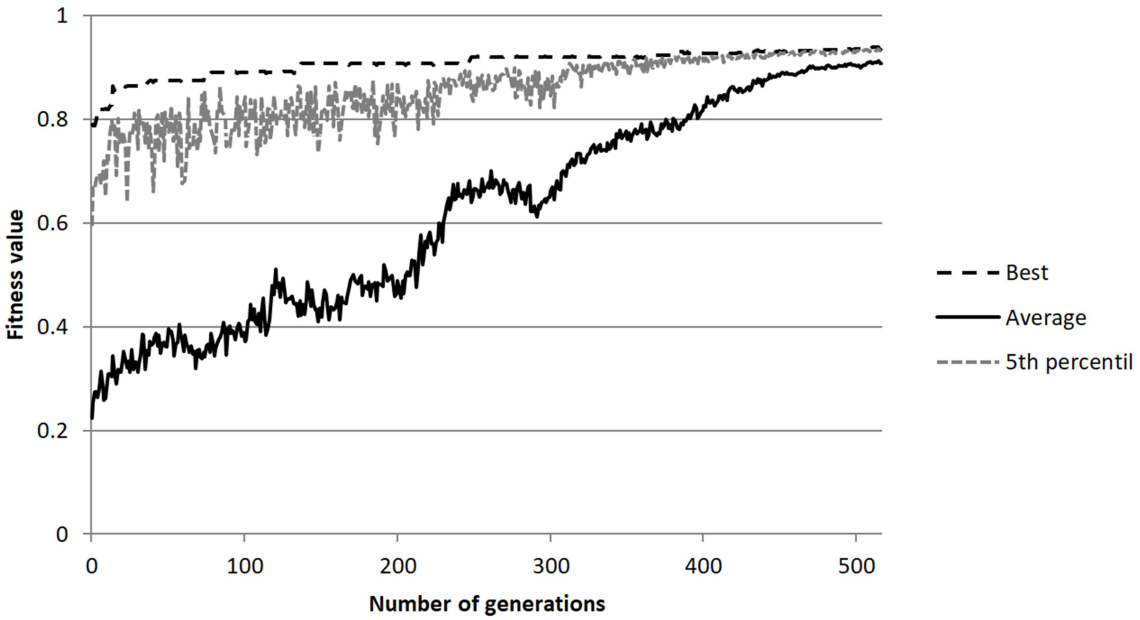

Figure 41: Development of fitness during genetic algorithms starting from by-hand designed FCM-NAS matrix. (source: author)

6 Discussion

This section compares the created model with the projects described in the State of the Art, and also returns to objectives of this work, evaluates accomplished goals and achievements, and outlines the future direction of research and possible enhancements of this work.

6.1 Comparison with Other Projects

Table 18 shows the feature comparison of the method proposed by this work to other projects with similar approaches or goals. It offers an overview of the models previously presented in section State of the Art. There are also two models considered before the design of FCM-NAS connected with schedule management. The first one is the finite-state machine that was implemented in the model in the beginning. During its implementation, many shortcomings appeared. The demand for a better solution was driven by tedious manual creation of a scenario, low flexibility, difficulties to connect activities to proper time, and some other drawbacks that are listed in Table 18. The second method considered before was a simple FCM-NAS, which has been already developed and tested. However, the goal of the project expected an implementation of scenarios, which are not supported by FCM-NAS alone.

All these projects and approaches aim for different objectives and focus their attention to different aspects of the simulation of human activity. Many of them do not explicitly aim for the decision-making which leads to incomplete, temporary, or limited solutions which are just enough for the objective but are not scalable. For instance, the PerSim simulations rely on a simple list of activities feeding agent's behavior. Since the model focuses on very detailed physical interactions of the agent with sensors, it does not run the simulation over an extended period of time when the list of activities would reach the end and agent stopped working. On the other hand, the HSS model, which would be capable of running infinitely, lacks the ability of routine behavior or any scenario implementation which limits it to need-driven activities only.

Another possible feature of human simulation is the ability to handle or at least support unexpected events in the environment. This feature requires either a flexible decision-making method or the mechanism offering to delay or interrupt activities. Obviously, models with strict scenarios do not support that feature as the simulation is directed only by given scenario. HSS with the need-oriented decision-making enables to interrupt activities to satisfy a more pressing need. NPCs (agents) in Kingdom Come are capable of handling unexpected situations by switching decision-making sub-models corresponding to the current context (regular, combat, and so on) [61].

The support of more than one agent in the environment is an important feature for testing ambient intelligence systems in more complex situations. MASSHA model [67] supports and tested two agents in a simulation. Each one has its own dataset directing its behavior through a day. This model focuses on interactions with the environment and has unvarying behavior; hence the agents' behavior does not get more complex by adding more agents. On the other hand, large-scale models with hundreds of agents like DiaSim [76] or ALModel with FCM-NAS only [77] tends to focus on the emergence and simplify agent's behavior which does not support routines or more complex interactions. The proposed method offers potentially full support of interactions based on both situational need and planned activities. However, only basic multi-agent support and interactions were tested in the model (see 4.7 Multi-agent Support).

The support of multiple scenarios per agent and conditions of activities makes behavior varying and more situation-specific. The proposed solution allows setting agents with an unlimited number of scenarios for different days of the week (or even month). Their schedule can be further randomized by the probability of appearance of activities. The method also enables activities to set conditions that need to be met in order to start. Unmet conditions then trigger alternative activities. These features make possible behaviors very complex and varying through a simulation.

Table 18: Comparison of the proposed human activity simulation to other similar projects. (source: author)

Model	PerSim [70]	PerSim3D [68, 71]	MASSHA [67]	HSS [53, 72]	OpenSHS [69, 73]	Kingdom Come [61]	Finite-state machine	FCM-NAS only	FCM-NAS with SM
Type of decision-making	Procedure with list of activities	Procedure with list of contexts	List of activities with priorities	Selection by priority function	Scenarios replicating user control	List of activities and DM sub-models	Finite-state machine	FCM-NAS	FCM-NAS + schedule management
Focus	Lower level - sensory detection	Lower level - sensory detection	Decision-making and interactions	Higher level of decision-making	Higher level of decision-making	Visual quality	Higher level of decision-making	Higher level of decision-making	Higher level of decision-making
Time scope	1 day	1 day	Days	Weeks	Days	Weeks	1 day	Months	Months
Scenarios support	Y	Y	Y	N	Y	Y	Y	N	Y
Simulation of needs	N	N	N	Y	N	N	Y	Y	Y
Unexpected events support	N	N	Y	Y	N	Y	N	Y	Y
Multi-agent support	N	N	Y	N	N	Y	N	Y	P
Multi-scenario support	N	N	N	N	Y	Y	N	N	Y
Activity conditions	N	Y	Y	N	N	Y	N	Y	Y
Activity hierarchy	N	Y	Y	Y	N	Y	P	N	Y
Animations	N	Y	N	N	N	Y	N	N	N
3D environment	N	Y	N	N	Y	Y	Y	Y	Y
Machine learning	N	N	N	N	N	N	N	Y	Y

Y: a model contains/supports a feature and it was tested; P: a model partially supports a feature or it has not been tested; N: a model does not support the feature

Although in some models it may be possible, none of the models actually uses machine learning. All their decision-making processes are set and configured by hand including scenarios. As described in section 5.5 Machine Learning, the proposed model uses genetic algorithms to adjust behavior for specific goals or a scenario. It can ultimately simulate each agent with different behavioral configuration learned specifically for its style of life (scenario). This feature is exclusive to the proposed model.

Regarding decision-making, the method presented in this work is able to emulate approaches of any previously described human activity simulation by omitting some elements of decision-making or learning the decision-making mechanism with adjusted weights of the fitness function. As shown in section 5.1 Need-oriented Behavior Only, the method can generate the same behavior patterns as simpler decision-making without scheduling capabilities. The opposite approach with the scripted scenario was emulated as well (see section 5.2 Routine Behavior Only). In section 5.3 Combination of Need-oriented and Routine Behavior, the combination of both approaches, which is exclusive to the proposed solution, proved it can autonomously manage dynamic agent's needs within planned scenario by delaying less critical activities or omitting redundant ones.

Experiments in section 5.4 Computational Performance verified the proposed solution is not limited by computational performance as it can process one virtual year in dozens of seconds, even on a bellow average hardware configuration.

6.2 Fulfillment of Objectives

The main objective of this work was to design a method for the simulation of human behavior which would be able to process decision-making based on both internal needs of agents and planned scenario. To accomplish and prove this objective, the following tasks were completed:

1. Review of application areas of human activity simulation.
2. Review of studies describing FCM and its modification with a focus on agent-based approaches.

3. Specification of the modified FCM method for human activity simulation.
4. Proposal of general methodological procedure of how to design and implement human activity simulation for selected application domains.
5. Implementation of the methodology into an ambient intelligence model, conducting experiments, and their evaluation.
6. Synthesis of results.

First two tasks were completed in section State of the Art. This section presented reviews of human activity simulation, FCMs in agent-based approaches, as well as all other research areas connected to this work. For each area of interest, it defined basic concepts and presented related works. It also described the mathematical background of FCM as the core method for the proposed approach, which further enhances these formulas in the context of agent-based models.

The following section, Solution, accomplished tasks number three and four. Its first part thoroughly describes the proposed enhancement of FCMs targeted to agent-based uses. Then the solution of routine behavior is presented showing general data structure and helpful information for the implementation of routines into any agent-based model. After both of these components are presented, this section discusses their cooperation and implementation into the model.

The fifth task, implementation of the model, was completed on the platform AnyLogic 8. Some of this process is described in section Solution; while section Results presents experimental results acquired by running the built model.

The response to the sixth task is presented in the section Results, which offers results of several experiments focused on the function of the model as a whole, the rationality of the decision-making and also its learning abilities through genetic algorithms. Then, this section compares the proposed method with models described in State of the Art to point out the positives and negatives of different approaches in the researched area. This comparison highlights features that are unique to the proposed model: the combination of need-oriented and routine

behavior and the ability to adapt behavior to the specific scenarios using machine learning.

Main benefits of the dissertation are as follows:

- It provided a complete overview of the problem of simulating human activities in agent-based applications.
- It provided the design of FCM-NAS method that could serve as a guide for the design of decision-making in a wide range of agent-based applications.
- It introduced the approach that combines the introduced FCM-NAS method with routine behavior for the human activity simulation.
- It further specified steps of implementation of the introduced decision-making mechanism into agents in an ambient intelligence model and pointed out possible obstacles.
- It presented the results of experiments showing the usefulness and abilities of this approach for human activities simulations.

The proposed solution could be used for a simulation of any living creature following routines during its life. In theory, it may improve an artificial life model with the addition of a day/night cycles, used as artificial intelligence for characters in computer games, or simulate agents in multi-agent models for research purposes or testing of ambient intelligence systems.

The last mentioned use case was tested. The behavior of the entity with this approach was implemented and tested within the virtual environment of an ambient intelligence model. The model is using the AnyLogic, which is a Java-based development platform optimized for agent-based models. The solution uses machine learning to optimize decision-making method. Agents' behavior is evaluated through fitness function and then adjusted by reinforcement learning – genetic algorithms. The fitness value depends on the percentage of accomplished scheduled activities and satisfaction of the agent's needs. Like in human behavior, the goal is not to start every scheduled activity exactly at the second it was meant

to begin. The intended behavior always considers not only schedule but even the current situation or expected events in the near future. Then, the agent is able to accomplish all the important scheduled activities. It may skip less important ones; however, there should always be a reason for every skip resulting from the current situation.

6.3 Future Work

Despite these achievements, this model has still the potential to advance. For instance, extending the simulated environment outside of the living area and adding more agents with complex interactions would allow simulating even work environment, potentially whole smart city. It would require new activities, interactions and even cooperation with other agents. Most importantly the method would need to process expectancy and uncertainty regarding other agents' behavior, which means that agents would need to synchronize their schedule and react when others are delayed. Currently, the model assumes that agents are not going through any significant changes (dramatic change in a lifestyle or agents moving in/out) during a simulation, which would also be possible to implement.

On the other hand, there is also a possibility to aim for a more detailed environment instead. That would, of course, require detailed low-level operations inside already implemented activities. For example, when the agent is making dinner, it would split into many simple tasks like open/close drawer or cabinet, take dishes and so on. In theory, the proposed method allows the addition of animations and very detailed physical sensory detection; however, the current platform is not well suited for it.

7 Conclusion

This work presented the new approach of human activity simulation using an agent-oriented modification of fuzzy cognitive maps with the addition of routine behavior sub-system. The proposed decision-making method is designed to follow a daily schedule while still being able to react to unexpected situations and manage an agent's needs autonomously. The main objective of this work was the design of a method for the simulation of human behavior which would be able to process decision-making based on both internal needs of agents and planned scenario. To accomplish the objective and test the proposed design, this method was implemented into an ambient intelligence model and compared with projects with similar objectives. It features abilities like delaying or interrupting activities due to unexpected events, multi-agent and multi-scenario support, conditions and requirements of activities, and hierarchical structure of contexts-activities-actions. On top of these more or less usual features of human simulation, the proposed model has added two unique ones: the cooperation of need-oriented and routine behavior and adaptation of behavior through machine learning.

In the beginning, this work summarized the current state of the art regarding the original fuzzy cognitive maps and their mathematical background, ambient intelligence models and other fields that are related to human activity simulation. This section also presented some projects that either use similar methods or aim for similar goals as the model implementation of this work.

The FCM-NAS, as the core method, was described in detail along with its abilities to learn. Also, the second component of the decision-making mechanism was presented: routine behavior implemented by scenarios and schedule management. Then the cooperation of these methods was introduced since both components are independent decision-making processes which might otherwise want to control the behavior of a single agent in an incompatible manner. After the design, this work presented its implementation into an ambient intelligence model and its adaptation by genetic algorithms. The final section describes conducted experiments and their results proving the accomplishment of the objective.

8 References

[1] N. Banovic, T. Buzali, F. Chevalier, J. Mankoff, and A. K. Dey, "Modeling and understanding human routine behavior," presented at the Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems, 2016.

[2] S. Russell and P. Norvig, *The Artificial Intelligence*: Prentice Hall Press, 2010.

[3] A. R. Jensen, *Bias in Mental Testing*: Free Press, 1980.

[4] C. R. Renato, V. Z. J. Fernando, and F. F. Mauricio, "Stigmergic Autonomous Navigation in Collective Robotics," in *Stigmergic Optimization*, ed Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 25-63.

[5] Promiskuitiv and Archon. (2014). *Age of empires II AI Scripting - GitHub*. Available: https://gist.github.com/Andygmb/1e3a6d9d444b2dfa8c40

[6] M. Certicky, *Implementing a Wall-In Building Placement in StarCraft with Declarative Programming* vol. abs/1306.4460, 2013.

[7] S. Ontañón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, "RTS AI Problems and Techniques," in *Encyclopedia of Computer Graphics and Games*, N. Lee, Ed., ed Cham: Springer International Publishing, 2015, pp. 1-12.

[8] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks,* vol. 61, pp. 85-117, 2015/01/01/ 2015.

[9] R. Lent, "A generalized reinforcement learning scheme for random neural networks," *Neural Computing and Applications,* October 19 2017.

[10] R. A. Howard, "Dynamic programming and markov processes," 1960.

[11] N. Khademi, A. S. Mohaymany, J. Shahi, and S. Zerguini, "An Algorithm for the Analytic Network Process (ANP) Structure Design," *Journal of Multi-Criteria Decision Analysis,* vol. 19, pp. 33-55, 2012.

[12] W. Reisig and G. Rozenberg, *Lectures on petri nets i: basic models: advances in petri nets*: Springer Science & Business Media, 1998.

[13] B. Kosko, "Fuzzy cognitive maps," *International Journal of Man-Machine Studies,* vol. 24, pp. 65-75, 1986/01/01 1986.

[14] R. Yu and G.-H. Tzeng, "A soft computing method for multi-criteria decision making with dependence and feedback," *Applied Mathematics and Computation,* vol. 180, pp. 63-75, 9/1/ 2006.

[15] F. Čapkovič, "Failures in discrete-event systems and dealing with them by means of Petri nets," *Vietnam Journal of Computer Science,* vol. 5, pp. 143-155, May 01 2018.

[16] N. Bulling, *A Survey of Multi-Agent Decision Making* vol. 28, 2014.

[17] J. P. Carvalho, "On the semantics and the use of fuzzy cognitive maps and dynamic cognitive maps in social sciences," *Fuzzy Sets and Systems,* vol. 214, pp. 6-19, 3/1/ 2013.

[18] E. I. Papageorgiou, K. E. Parsopoulos, C. S. Stylios, P. P. Groumpos, and M. N. Vrahatis, "Fuzzy Cognitive Maps Learning Using Particle Swarm Optimization," *Journal of Intelligent Information Systems,* vol. 25, pp. 95-121, July 01 2005.

[19] O. Motlagh, Z. Jamaludin, S. H. Tang, and W. Khaksar, "An agile FCM for real-time modeling of dynamic and real-life systems," *Evolving Systems,* vol. 6, pp. 153-165, 2015.

[20] Y.-H. Hsieh, I.-H. Chen, and S.-T. Yuan, "FCM-based customer expectation-driven service dispatch system," *Soft Computing,* vol. 18, pp. 359-378, 2013.
[21] V. Senniappan, J. Subramanian, E. I. Papageorgiou, and S. Mohan, "Application of fuzzy cognitive maps for crack categorization in columns of reinforced concrete structures," *Neural Computing and Applications,* pp. 1-11, 2016.
[22] S. Ahmadi, C. H. Yeh, E. I. Papageorgiou, and R. Martin, "An FCM-FAHP approach for managing readiness-relevant activities for ERP implementation," *Computers & Industrial Engineering,* vol. 88, pp. 501-517, Oct 2015.
[23] G. Kyriakarakos, K. Patlitzianas, M. Damasiotis, and D. Papastefanakis, "A fuzzy cognitive maps decision support system for renewables local planning," *Renewable and Sustainable Energy Reviews,* vol. 39, pp. 209-222, 2014.
[24] L. S. Jayashree, N. Palakkal, E. I. Papageorgiou, and K. Papageorgiou, "Application of fuzzy cognitive maps in precision agriculture: a case study on coconut yield management of southern India's Malabar region," *Neural Computing and Applications,* vol. 26, pp. 1963-1978, 2015.
[25] P. J. Giabbanelli and R. Crutzen, "Creating groups with similar expected behavioural response in randomized controlled trials: a fuzzy cognitive map approach," *BMC Medical Research Methodology,* vol. 14, pp. 1-19, 2014.
[26] H. Barón, R. Crespo, J. Pascual Espada, and O. Martínez, "Assessment of learning in environments interactive through fuzzy cognitive maps," *Soft Computing,* pp. 1-14, 2014.
[27] M. Mendonça, L. V. R. d. Arruda, and F. Neves-Jr, "Cooperative Autonomous Agents Based on Dynamical Fuzzy Cognitive Maps," in *Fuzzy Cognitive Maps for Applied Sciences and Engineering: From Fundamentals to Extensions and Learning Algorithms*, I. E. Papageorgiou, Ed., ed: Springer Berlin Heidelberg, 2014, pp. 159-175.
[28] A. L. Laureano-Cruces and A. Rodriguez-Garcia, "Design and implementation of an educational virtual pet using the OCC theory," *Journal of Ambient Intelligence and Humanized Computing,* vol. 3, pp. 61-71, 2011.
[29] G. Acampora, V. Loia, and A. Vitiello, "Distributing emotional services in Ambient Intelligence through cognitive agents," *Service Oriented Computing and Applications,* vol. 5, pp. 17-35, 2011.
[30] P. Szwed, P. Skrzynski, and W. Chmiel, "Risk assessment for a video surveillance system based on Fuzzy Cognitive Maps," *Multimedia Tools and Applications,* pp. 1-24, 2014.
[31] R. T. Jones, E. Connors, M. Mossey, J. Hyatt, N. Hansen, and M. Endsley, "Using fuzzy cognitive mapping techniques to model situation awareness for army infantry platoon leaders," *Computational and Mathematical Organization Theory,* vol. 17, pp. 272-295, 2011/09/01 2011.
[32] R. Furfaro, W. Fink, and J. S. Kargel, "Autonomous real-time landing site selection for Venus and Titan using Evolutionary Fuzzy Cognitive Maps," *Applied Soft Computing,* vol. 12, pp. 3825-3839, 2012.

[33] W. Stach, L. Kurgan, and W. Pedrycz, "A divide and conquer method for learning large Fuzzy Cognitive Maps," *Fuzzy Sets and Systems,* vol. 161, pp. 2515-2532, 10/1/ 2010.
[34] A. Jose, "Dynamic Fuzzy Cognitive Maps for the Supervision of Multiagent Systems," in *Fuzzy Cognitive Maps: Advances in Theory, Methodologies, Tools and Applications*, M. Glykas, Ed., ed: Springer Berlin Heidelberg, 2010, pp. 307-324.
[35] M. Mendonça, L. V. R. de Arruda, I. R. Chrun, and E. S. da Silva, "Hybrid Dynamic Fuzzy Cognitive Maps Evolution for autonomous navigation system," in *2015 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 2015, pp. 1-7.
[36] M. Stula, D. Krstinic, and L. Seric, "Intelligent forest fire monitoring system," *Information Systems Frontiers,* vol. 14, pp. 725-739, 2011.
[37] R. Gras, A. Golestani, M. Hosseini, M. Khater, Y. Farahani, M. Mashayekhi*, et al.*, "Ecosim: an individual-based platform for studying evolution," in *European Conference on Artificial Life*, 2011, pp. 284-286.
[38] M. Khater, E. Salehi, and R. Gras, "The Emergence of New Genes in EcoSim and Its Effect on Fitness," in *Simulated Evolution and Learning*. vol. 7673, L. Bui, Y. Ong, N. Hoai, H. Ishibuchi, and P. Suganthan, Eds., ed: Springer Berlin Heidelberg, 2012, pp. 52-61.
[39] R. Gras. (2013). *EcoSim: An ecosystem simulation*. Available: https://sites.google.com/site/ecosimgroup/research/ecosystem-simulation
[40] M. Khater and R. Gras, "Adaptation and Genomic Evolution in EcoSim," in *From Animals to Animats 12: 12th International Conference on Simulation of Adaptive Behavior, SAB 2012, Odense, Denmark, August 27-30, 2012. Proceedings*, T. Ziemke, C. Balkenius, and J. Hallam, Eds., ed: Springer Berlin Heidelberg, 2012, pp. 219-229.
[41] T. M. Mitchell, *Machine Learning*: McGraw-Hill, Inc., 1997.
[42] A. Aspinall and R. Gras, "K-Means Clustering as a Speciation Mechanism within an Individual-Based Evolving Predator-Prey Ecosystem Simulation," in *Active Media Technology*. vol. 6335, A. An, P. Lingras, S. Petty, and R. Huang, Eds., ed: Springer Berlin Heidelberg, 2010, pp. 318-329.
[43] R. Scott and R. Gras, *Comparing Distance-Based Phylogenetic Tree Construction Methods Using An Individual-Based Ecosystem Simulation, EcoSim*, 2012.
[44] A. Šošić, A. M. Zoubir, and H. Koeppl, "Reinforcement learning in a continuum of agents," *Swarm Intelligence,* vol. 12, pp. 23-51, March 01 2018.
[45] A. R. KhudaBukhsh, J. G. Carbonell, and P. J. Jansen, "Robust learning in expert networks: a comparative analysis," *Journal of Intelligent Information Systems,* vol. 51, pp. 207-234, October 01 2018.
[46] M. Abdoos, N. Mozayani, and A. L. C. Bazzan, "Hierarchical control of traffic signals using Q-learning with tile coding," *Applied Intelligence,* vol. 40, pp. 201-213, March 01 2014.
[47] T. Chugh, K. Sindhya, J. Hakanen, and K. Miettinen, "A survey on handling computationally expensive multiobjective optimization problems with evolutionary algorithms," *Soft Computing,* December 11 2017.

[48] H. Bhasin, G. Behal, N. Aggarwal, R. K. Saini, and S. Choudhary, "On the applicability of diploid genetic algorithms in dynamic environments," *Soft Computing,* vol. 20, pp. 3403-3410, September 01 2016.
[49] T. Y. Lim, "Structured population genetic algorithms: a literature survey," *Artificial Intelligence Review,* vol. 41, pp. 385-399, March 01 2014.
[50] O. Hasançebi and F. Erbatur, "Evaluation of crossover techniques in genetic algorithm based optimum structural design," *Computers & Structures,* vol. 78, pp. 435-448, 2000/11/01/ 2000.
[51] J. Flegr, "Elastic, not plastic species: Frozen plasticity theory and the origin of adaptive evolution in sexually reproducing organisms," *Biology Direct,* vol. 5, p. 2, January 13 2010.
[52] G. M. Hodgson, "The ubiquity of habits and rules," *Cambridge journal of economics,* vol. 21, pp. 663-684, 1997.
[53] B. Kormányos and B. Pataki, "Multilevel simulation of daily activities: Why and how?," presented at the 2013 IEEE International Conference on Computational Intelligence and Virtual Environments for Measurement Systems and Applications (CIVEMSA), 2013.
[54] M. Grabisch, "Temporal scenario modelling and recognition based on possibilistic logic," *Artificial Intelligence,* vol. 148, pp. 261-289, Aug 2003.
[55] I. Cavrak, A. Stranjak, and M. Zagar, "SDLMAS: A Scenario Modeling Framework for Multi-Agent Systems," *Journal of Universal Computer Science,* vol. 15, pp. 898-925, 2009.
[56] M. P. Huget, "Agent UML notation for multiagent system design," *IEEE Internet Computing,* vol. 8, pp. 63-71, Jul-Aug 2004.
[57] A. Greasley and C. Owen, "Behavior in Models: A Framework for Representing Human Behavior," in *Behavioral Operational Research: Theory, Methodology and Practice*, M. Kunc, J. Malpass, and L. White, Eds., ed London: Palgrave Macmillan UK, 2016, pp. 47-63.
[58] Y. G. Lee, "Developing a Human Behavior Simulation System Based on Geometry Affordance," Cham, 2018, pp. 475-479.
[59] W. Warwick, N. Buchler, and L. Marusich, "An Integrated Model of Human Cyber Behavior," Cham, 2019, pp. 290-302.
[60] Y. Cheng, D. Liu, J. Chen, S. Namilae, J. Thropp, and Y. Seong, "Human Behavior Under Emergency and Its Simulation Modeling: A Review," Cham, 2019, pp. 313-325.
[61] T. Plch, "Belieavable decision making in large scale open world games for ambient characters," PhD. Doctoral thesis, Department of Software and Computer Science Education, Faculty of Mathematics and Physics, Charles University in Prague, Praha, 2017.
[62] Metacritic. (2019). *Kingdom Come: Deliverance for PC Reviews - Metacritic*. Available: https://www.metacritic.com/game/pc/kingdom-come-deliverance
[63] P. Mikulecky and R. Cimler, "Simulating Decision Processes in Intelligent Environments," presented at the Intelligent Environments 2016, Amsterdam, 2016.

[64] G. Azkune, A. Almeida, D. Lopez-de-Ipina, and L. M. Chen, "Combining Users' Activity Survey and Simulators to Evaluate Human Activity Recognition Systems," *Sensors,* vol. 15, pp. 8192-8213, Apr 2015.
[65] D. Bonino and F. Corno, "Modeling, simulation and emulation of Intelligent Domotic Environments," *Automation in Construction,* vol. 20, pp. 967-981, Nov 2011.
[66] M. Buchmayr, W. Kurschl, and J. Küng, "A Simulator for Generating and Visualizing Sensor Data for Ambient Intelligence Environments," *Procedia Computer Science,* vol. 5, pp. 90-97, 2011/01/01/ 2011.
[67] O. Kamara-Esteban, G. Azkune, A. Pijoan, C. E. Borges, A. Alonso-Vicario, and D. Lopez-de-Ipina, "MASSHA: An agent-based approach for human activity simulation in intelligent environments," *Pervasive and Mobile Computing,* vol. 40, pp. 279-300, Sep 2017.
[68] A. Helal, K. Cho, W. Lee, Y. Sung, J. W. Lee, and E. Kim, "3D Modeling and Simulation of Human Activities in Smart Spaces," presented at the 2012 9th International Conference on Ubiquitous Intelligence & Computing and 9th International Conference on Autonomic & Trusted Computing, Los Alamitos, 2012.
[69] N. Alshammari, T. Alshammari, M. Sedky, J. Champion, and C. Bauer, *OpenSHS: Open smart home simulator* vol. 17, 2017.
[70] S. Helal, J. W. Lee, S. Hossain, E. Kim, H. Hagras, and D. Cook, "Persim - Simulator for Human Activities in Pervasive Spaces," presented at the 2011 Seventh International Conference on Intelligent Environments, 2011.
[71] S. Liu, S. Helal, and J. W. Lee, "High Fidelity Simulation and Visualization of Activities of Daily Living in Persim 3D," Cham, 2017, pp. 136-148.
[72] B. Kormányos and B. Pataki. (2013). *Home Sensor Simulator*. Available: http://home.mit.bme.hu/~kormi/simulator/
[73] N. Alshammari. (2017). *OpenSHS by openshs*. Available: https://openshs.github.io/openshs/
[74] T. Alshammari, N. Alshammari, M. Sedky, and C. Howard, *SIMADL: Simulated Activities of Daily Living Dataset* vol. 3, 2018.
[75] K. Bouchard, A. Ajroud, B. Bouchard, and A. Bouzouane, "SIMACT: A 3D Open Source Smart Home Simulator for Activity Recognition," Berlin, Heidelberg, 2010, pp. 524-533.
[76] J. Bruneau, W. Jouve, and C. Consel, "DiaSim: A parameterized simulator for pervasive computing applications," in *2009 6th Annual International Mobile and Ubiquitous Systems: Networking & Services, MobiQuitous*, 2009, pp. 1-10.
[77] T. Nachazel, "Analytic hierarchy process in artificial life model based on fuzzy cognitive maps," *Journal of Ambient Intelligence and Smart Environments,* vol. 10, pp. 127-141, 2018.
[78] T. Nachazel, "Optimization of Decision-Making in Artificial Life Model Based on Fuzzy Cognitive Maps," in *Intelligent Environments (IE), 2015 International Conference on*, 2015, pp. 136-139.
[79] T. Nacházel, "Inteligentní systémy ve virtuálním prostředí," Bc. Bachelor thesis, Faculty of Informatics and Management, University of Hradec Králové, Hradec Králové, 2012.

[80] T. Nachazel, "Models of artificial life based on fuzzy cognitive maps," in *Advances in Informatics, Information Management and Administration*, first ed Liberec: Technical University of Liberec, 2014, pp. 75-87.
[81] T. Nachazel. (2016). *NetLogo User Community Models: ALModel*. Available: http://ccl.northwestern.edu/netlogo/models/community/ALModel
[82] P. Tucnik, T. Nachazel, and P. Mikulecky, "On the Methods of Scenarios Representation in Ambient Environments," *[Not published yet]*.
[83] P. Tucnik, T. Nachazel, P. Blecha, and P. Mikulecky, "Modelling Routine Interactions with Intelligent Environments," *[Not published yet]*.
[84] P. Tucnik, T. Nachazel, P. Cech, and V. Bures, "Comparative analysis of selected path-planning approaches in large-scale multi-agent-based environments," *Expert Systems with Applications,* vol. 113, pp. 415-427, 2018/12/15/ 2018.

8.1 List of Author's Publications

Publications in journals with impact factor

1. Nachazel, T. (2018) Analytic hierarchy process in artificial life model based on fuzzy cognitive maps. Journal of Ambient Intelligence and Smart Environments, vol. 10, no. 2, pp. 127-141

2. Tucnik, P.; Nachazel, T.; Cech, P.; Bures V. (2018) Comparative Analysis of Modified Pathfinding Algorithms in Large-Scale Multi-Agent Based Environments. Expert Systems with Applications, 113, pp. 415-427.

Publications in proceedings of international conferences

3. Nacházel, T. (2015) Optimization of Decision-Making in Artificial Life Model Based on Fuzzy Cognitive Maps. Intelligent Environments 2015, Praha, pp. 136-139.

4. Tucnik, P.; Nachazel, T. (2016) Agent-based Computational Economics: Modelling of Economic Entities. HED 2016, Hradec Králové, pp. 1133-1140.

5. Nachazel, T.; Tucnik, P. (2017) Decision Making in Agent-based Virtual Economic Model. HED 2017, Hradec Králové, pp. 629-635.

6. Tucnik, P.; Nemcova, Z.; Nachazel, T. (2017) Multiplant Production Design in Agent-Based Artificial Economic System. 9th International Conference on Computational Collective Intelligence, ICCCI 2017, Nicosia, Cyprus, pp. 371-380.

7. Bures, V.; Cech, P.; Husakova, M.; Kriz, P.; Krylova, K.; Nachazel, T.; Ponce, D.; Poulova, P.; Tacheci, I.; Tucnik, P. (2019) Learning by Doing in Medicine: Solution and Configuration of Virtual Medical Cases. International Symposium on Educational Technology 2019, Hradec Králové.

Other publications

8. Nacházel, T. (2014) Modely umělého života na bázi fuzzy kognitivních map. IMEA 2014, Sychrov, pp. 205-210.

9. Nacházel, T. (2014) Models of Artificial Life Based on Fuzzy Cognitive Maps. Advances in Informatics, Information Management and Administration. pp. 75-87.

Publications in review process

10. Tucnik, P.; Nachazel, T.; Blecha, P.; Mikulecky, P. (????) Modelling Routine Interactions with Intelligent Environments. (submitted to Human-centric Computing and Information Sciences)

11. Tucnik, P.; Mikulecky, P.; Nachazel, T. (????) On the Methods of Scenarios Representation in Ambient Environments. (in preparation)

12. Nachazel, T. (????) Fuzzy cognitive maps for autonomous agents in dynamic environments. (submitted to Genetic Programming and Evolvable Machines)

8.2 Overview of Research Activities

- Specific research project "Socio-economic models and autonomous systems 2" (2019, Ing. Tomáš Nacházel)
- Specific research project "Socio-economic models and autonomous systems" (2018, Ing. Tomáš Nacházel)
- GAČR "DEPIES - Decision Processes in Intelligent Environments" - n. 15-11724S (2015-2017, prof. RNDr. Peter Mikulecký, Ph.D.)
- Specific research project " Autonomous Socio-Economic Systems" (2017, RNDr. Petr Tučník, Ph.D.)
- Specific research project "Řešení produkčních, přepravních a alokačních problémů v agentově-orientovaných modelech" (2016, RNDr. Petr Tučník, Ph.D.)
- Project Excelence "Modeling and Simulation of Processes in Socioeconomic Systems and Intelligent Environments" (2017, prof. RNDr. Peter Mikulecký, Ph.D.)
- Project Excelence "MAS Applications in Modeling of Complex Socioeconomic Systems and Intelligent Environments" (2015, RNDr. Petr Tučník, Ph.D.)
- Specific research project "Agentové modely a simulace" (2015, doc. RNDr. Kamila Štekerová, Ph.D.)
- Specific research project "SCM and Control of Markets and Production in Agent-based Computational Economics" (2015, RNDr. Petr Tučník, Ph.D.)
- Specific research project "Informační a znalostní management a kognitivní věda v cestovním ruchu" (2017, prof. RNDr. Josef Zelenka, CSc.)
- Specific research project "Informační a znalostní management a kognitivní věda v cestovním ruchu" (2018, prof. RNDr. Josef Zelenka, CSc.)

Attachments

Content of the attached disc

The disc attached to this work contains the full text of the doctoral thesis, the model of ambient intelligence on AnyLogic platform, and the artificial life model (ALModel), which was mentioned as an example of FCM-NAS on platform NetLogo. User guides, generated output data, source codes, and installation files for platforms in the correct version are also included. The content of the disc has the following structure:

- Thesis
    - Text
    - Figures
- Models
    - Ambient intelligence model
    - ALModel
- Software

Algorithm

The following pseudocode includes the algorithm of the computation of FCM-NAS with all support functions needed to process values every time step.

Input:

```
matrix          //two-dimensional array of values of relations between nodes
values          //one-dimensional array (vector) of values of all nodes

necessityValues  //values for optional necessity calculations
necessityOfStates//evaluation of necessity of State nodes
necessity       //true if necessity calculation is active; false if it is not
primaryState    //position of Primary State node
firstInput      //position of the first input node; expecting all following are
                input nodes as well
granularity     //granularity parameter
criticalValue   //minimal level of Activity nodes required to start activity
```

High-level procedure:

```
processFCM(values, matrix){
    previousValues = copy(values)
    processNeeds()
    processStates()
    updateInputStateNodes()    //if there are any external input nodes
    processActivities()
    transformationOfActivities()        //select activity to perform
}
```

Computations of FCM-NAS sections:

```
processNeeds(){
    for (j = 1, ..., numberOfNeeds){
            increment = 0.0
            for (i = 1, ..., numberOfNodes){
                    increment += matrix[i, j] · previousValues[i]
            }
            values[j] = transformation(previousValues[j] +
            + (granularity · increment))
    }
}
```

```
processStates(){
    for (j = numberOfNeeds + numberOfActivities + 1, ..., firstInput){
            value = 0.0
            for (i = 1, ..., numberOfNodes){
                    value += matrix[i, j] · previousValues[i]
            }
            values[j] = transformation(value)
    }
    fitness = values[primaryState]
    values[primaryState] = 1.0
}
processActivities(){
if (necessity){
    for (j = numberOfNeeds + 1, ..., numberOfNeeds + numberOfActivities){
            value = 0.0
            for (i = 1, ..., numberOfNeeds){
                    value += matrix[i, j] · previousValues[i] · necessityValues[i]
            }
            for (i = numberOfNeeds + 1, ..., numberOfNodes){
                    value += matrix[i, j] · previousValues[i]
            }
            values[j] = value
    }
}else{
    for (j = numberOfNeeds + 1, ..., numberOfNeeds + numberOfActivities){
            value = 0.0
            for (i = 1, ..., numberOfNodes){
                    value += matrix[i, j] · previousValues[i]
            }
            values[j] = value
    }
}}

transformationOfActivities(){   //no pararel activities allowed in this version
    max = 0.0
    maxPosition = -1
    for (i = numberOfNeeds + 1, ..., numberOfNeeds + numberOfActivities){
        if (values[i] > criticalValue  AND values[i] > max){
                max = values[i]
                if (maxPosition != -1)
                        values[maxPosition] = 0.0
                maxPosition = i
                values[i] = 1.0
        }else
                values[i] = 0.0
    }
}
```

Support functions:

```
transformation(value){
    if (value > 1.0)
            value = 1.0
    if (value < 0.0)
            value = 0.0
    return value
}

setUpNecessity(){                    //executed once at the start of a model
    for (i = 1, ..., numberOfNeeds){
            coeficient = 0
            for (j = numberOfNeeds + numberOfActivities + 1, ..., numberOfNodes){
                    coeficient -= matrix[i, j] · necessityOfStates[j]
            }
            necessityValues[i] = coeficient
    }
    sum = 0.0
    for (i = 1, ..., numberOfNeeds){
            sum += necessityValues[i]
    }
    diff = 1 - (sum / numberOfNeeds)
    for (i = 1, ..., numberOfNeeds){
            necessityValues[i] += diff
    }
}
```