Communication Papers of the of the 17th Conference on Computer Science and Intelligence Systems pp. 173–180 DOI: 10.15439/2022F234 ISSN 2300-5963 ACSIS, Vol. 32

A comprehensive framework for designing behavior of UAV swarms

Piotr Cybulski
Military University of Technology in Warsaw
ul. Gen. Sylwestra Kaliskiego 2,
00-908 Warszawa, Poland
Email: piotr.cybulski@wat.edu.pl

Zbigniew Zieliński
Military University of Technology in Warsaw
ul. Gen. Sylwestra Kaliskiego 2,
00-908 Warszawa, Poland
Email: zbigniew.zielinski@wat.edu.pl

Abstract—**This paper aims to present a method of designing the behavior of robotic swarms, emphasizing swarms of unmanned aerial vehicles using bigraphs. The method's primary goal is to define a set of actions to be performed in subsequent moments by the members of a swarm that lead to the completion of the given task. In addition to formal definitions, an example use case is also included to demonstrate how utilizing our method allows overcoming typical difficulties related to swarm robotics engineering. The example covers verifying non-functional requirements and scaling a task both horizontally and vertically.**

I. Introduction

ROBOTIC swarm (SR) is a special case of multi-agent system [8] (MAS), and as such have been the subject of study for many years[10][9]. Beyond attributes typically associated with MAS, such as agent's independence, decentralized control, and the lack of access to information about the global state of a system, they can be attributed to features of real-world swarms[11][13]. These can be features of both individual members of the swarm and the swarm as a whole. The former group consists of the ability to interact with the environment or other members of the swarm, local perception capabilities, and relatively low cognitive capacity. On the other hand, the features attributed to the swarm are robustness, scalability, and flexibility. Finally, one aspect of robotic swarms that is often emphasized is that their emergent behavior is more complex than its members' behavior alone. From now on, by swarm behavior, we shall understand the behavior of its members.

Current research results on robotic swarms can be categorized by how they fit on multiple different axes[16][12]. Most common are the size of a swarm, ability to communicate (one-to-one, one-to-all, etc.), communication bandwidth, computational capabilities, or whether the swarm is homogeneous or heterogeneous. There are also classifications not by attributes of the swarm itself but whether it was designed with a top-down or bottom-up approach[17].

In the literature on robot swarms, there is a clear tendency to consider the goal posed to the swarm as a variant of tasks from a predefined set [11][14][15]. Tasks that exhibit the highest potential for practical application include moving in formation, area coverage, rendezvous, and foraging.

Main difficulties in designing swarms of robots arise from expectations that we do not have of ordinary multi-agent systems. In particular, the low capabilities of a single element of the swarm and, consequently, the problems with communication and control of the whole swarm[18]. Another issue will be to define a level of automation[4] of a swarm element. The more autonomous the swarm is expected to be, the more difficult it becomes to self-control itself in real-time, or the given task has to be less unpredictable. For highly autonomous swarms, there is also the problem of collective decision-making [19]. On the other hand, swarms that are highly dependent on an external operator may become useless in practice due to difficulties described in [18].

Existing methods for designing the behavior of a robot swarm can be divided into those that try to find similarities in the behavior of the designed swarm to the operating mechanisms of systems well understood within other areas of science and those that try to be universal. The former group consists of bio-inspired methods[25][26], physics-inspired ones [27][28][29] (with the strong emphasis on particles repulsion/attraction interactions), and those that considers members of the swarm as a dynamical system with their knowledge modeled with graph theory[30]. The latter group of methods for designing tries not to use analogies to well-studied phenomena and instead proposes its schemes, often with multiple stages[20][21][22]. We would also include in this group methods that use specially designed abstract mathematical structures. The most important of these would be amorphous[23] and aggregated[24] computing due to their resemblance to the concept of swarms.

Bigraphs[3] are formal structures that allow to model systems, in which spatial arrangement and interconnection of elements play an important role. It is also feasible to define the dynamics of a system within bigraphs framework. So far they have been found useful in designing Internet of Things [31][32] and computer networks [33]. The applicability of bigraphs in the process of designing behavior of a multi-agent system's elements requires further research. Most of existing works [34][35] focus merely on utilizing bigraphs to represent a system, without taking under consideration how its elements should operate to accomplish a given goal. These works [36] that include also this aspect of the design process do so with an additional level of abstraction.

This work aims to present a method for designing swarm behavior using bigraphs. It is a summary of work previously presented in [2] and [1] extended with a formal definition of the schedule of actions. Section II defines all the formal structures together with how they can be used to develop a swarm behavior performing a designer-defined task. Section III presents a minimal example of a problem that can be solved using the proposed method. It covers alternative scenarios for executing the given task, verifying non-function requirements, and scaling the task both horizontally and vertically. Section IV is devoted to further direction of work related to the development of the presented method.

The design method described further on does not impose restrictions on the designed swarm members. Later in this paper, we will consider swarms whose members are small unmanned aerial vehicles (UAVs), called drones.

II. Designing UAV swarm behavior

A. Assumptions and limitations of designed swarms

The following will define, in a condensed form, a method for designing the behavior of elements of a multi-agent system [2]. It allows the development of a manner in which a designer-defined task can be completed successfully.

Developing a swarm behavior is a process of constructing a *schedule of actions* (defined formally later in this paper) for all members of the swarm. A schedule can be interpreted as a set of operations or activities to be carried out at specific moments. A schedule also indicates a role played by each member of the swarm participating in any action. The designing process begins with defining a task and an initial state of a system. By system, we will understand all elements of the real world that are relevant to the specified task. It is also necessary to define all kinds of activities that may happen within the system. An example of an activity can be a movement of drones between areas or picking up an object. Activities may involve multiple members of the swarm or its single member. Within each activity, each participant may play a different role. Both the initial state and the activities are described in with *bigraph* notation. The next step is to define all states of the system that are reachable from the initial state by performing any combination of activities from the set defined earlier. *Tracking transition systems* are used to describe both states and activities, along with roles played in each of them. The next objective is to find a sequence of activities that, when performed, allow to transform the system from being in its initial state into the desired state. To do this, a structure called *state space* has been proposed, where each state is reduced to a vertex (forgetting its bigraphical representation) of a directed multigraph, and arcs between vertices denote an activity enabling a change from one state to the other. Additionally, each arc has assigned a *mission progress function* which describes relations between swarm elements participating in the activity in a computationally inexpensive manner. Within this structure, searching for a sequence of actions that allows accomplishing the given task can be considered a problem of finding a walk (in graph theory meaning) between two vertices. A method for finding all walks of specified length has been developed to solve this problem. The last step is to transform one of the founded walks into a schedule of actions. Having such a schedule, we can verify whether executing the given task meets the non-functional requirements.

If we consider designing UAV behavior as a decision process, then designing with our method can be regarded as a level 7 automatic process according to the scale proposed in [4]. This level of automation means that the designer may leave the designing process to a specialized tool after providing initial parameters.

Components of systems for which behavior can be designed using this method can be divided into three categories: *elements of the environment*, *passive objects* and *active objects*. Environmental objects can not initiate any action, and time-lapse is irrelevant to them. An example of such elements may be areas between which drones can travel or an object that can be lifted by at most one drone. Passive objects can not initiate any action, but the passage of time has to be considered for these kinds of objects. An example of such an object can be an item multiple drones can possess during task execution. The last kind of system components we recognize are active objects which can both initialize actions and time relevant to them during a mission. Drones are an example of such objects. It is assumed that every active object in a system can be controlled. It means there are no active objects in the system that may act unexpectedly. The final limitation is that the number of objects (both active and passive) is constant during task execution.

Knowing the above, we can classify swarms for which the behavior we will try to design as “set & forget“. If we consider the execution of a task as a decision process, where the decisions regarding whether and, if so, how to perform actions not included in a schedule, then this process is level 8 or 9 on the autonomy scale mentioned earlier. Level 8 means that a swarm executes a given task and gives feedback when the operator requests it. Level 9 means that feedback is at the discretion of the swarm performing a task.

B. Bigraphs

The primary formal tool utilised in our method are bigraphs [3]. They allow for modeling ubiquitous computing with just graphical notation.

Below, we define the key concepts necessary to understand later parts of the paper. For more in dept introduction we refer to [5].

A (concrete) bigraph over the signature $(\mathcal{K}, ar : \mathcal{K} \to \mathbb{N})$ is defined as:

$$B = (\mathbb{V}_B, \mathbb{E}_B, ctrl_B, G_B^P, G_B^L) : I \to O$$

where:

- $\mathbb{V}_B$ - is a set of vertex identifiers;
- $\mathbb{E}_B$ - is a set of hyperedges identifiers;
- $ctrl_B : \mathbb{V}_B \to \mathcal{K}$ - is a function assigning controls to vertices;

- G^P_B, G^L_B - is a place graph G^P_B and a link graph G^L_B comprising the bigraph B:
  - $G^P_B = \langle \mathbb{V}_B, ctrl_B, prnt_B \rangle : m \to n$ - is the place graph of bigraph B with m sites and n roots. Both m and n are finite ordinals of the form $x\{0, 1, \ldots, x - 1\}$. A function $prnt_B : m \uplus \mathbb{V}_B \to \mathbb{V}_B \uplus n$ defines a hierarchical relationship between vertices, sites and roots. It is important to emphasize that the map $prnt$ defines a set of rooted trees.
  - $G^L_B = \langle \mathbb{V}_B, \mathbb{E}_B, ctrl_B, link_B \rangle : X \to Y$ - is the link graph of bigraph B, with inner names X and outer names Y. A link map $link_B : X \uplus P_B \to \mathbb{E}_B \uplus Y$ defines connections from vertices and inner names to hyperedges and outer names of the link graph. A set $P_B = \{(v, i) \mid i \in ar(ctrl_B(v))\}$ is the set of ports of G^L_B. Its elements (v, i) shall be interpreted as ith port of vertex v.
- $I = \langle m, X \rangle$ and $O = \langle n, Y \rangle$ are the inner and outer interface of the bigraph B.

We use $\uplus$ to denote a union of sets assumed to be disjoint.

Dynamics of systems modeled with bigraphs can be expressed with reaction rules. In this paper we will use *linear parametric reaction rules* proposed in [6], extended with *tracking*. Formally this kind of a reaction rule is defined as:

$$r = (R : \langle m, \emptyset \rangle \to \langle n, X \rangle, R' : \langle m, \emptyset \rangle \to \langle n, X \rangle, \tau)$$

where:

- R - is a bigraph called *redex* of reaction rule *r*;
- R' - is a bigraph called *reactum* of reaction rule *r*;
- $\tau : \mathbb{V}_{R'} \to \mathbb{V}_R$ - is a *tracking map* indicating which vertices in redex correspond to which vertices in reactum. If this function is partial, or if it is not a surjection, then it means that as a result of action represented by this reaction rule some of the elements either have been created or disappeared.

For the purpose of this introduction we will just indicate that applying a reaction rule means finding occurrences of the redex in a target bigraph and replacing them with reactum. Details of these operations are skipped in this paper.

C. Tracking transition system

Apart from defining actions (reaction rules) that may happen in the system and specifying when preconditions for each of them are satisfied (finding redexes), we need to answer the following question regarding the dynamics of the system:

- Who participates in an action that transforms a system state in a particular way?
- Which elements of a state resulting from the transformation are "residue" of the state before the action?

To answer these questions, a structure called *tracking transition system* has been proposed in [2] and later refined in [1]. It is an extension of the basic transition systems defined in [7].

Formally, a tracking transition system is a seven tuple $TTS = (Agt, Red, Lab, Apl, Par, Res, Tra)$ where:

- Agt - is a set of bigraphs;
- Red - is a set of redexes of reaction rules;
- Lab - is a set of labels of the reaction rules. They are meant to distinguish different types of actions in the system.
- $Apl \subseteq Agt \times Lab$ - is an applicability relation. It indicates which reaction rules can be applied from which bigraph;
- $Par = \{\mathbb{V}_b \to \mathbb{V}_r \mid b \in Agt,\ r \in Red\}$ - is a set of *participation* functions. Each of them maps vertices in a redex from Red to vertices in a bigraph from Agt. Every function $par \in Par$ is injective.
- $Res = \{\mathbb{V}_{b_1} \to \mathbb{V}_{b_2} \mid b_1, b_2 \in Agt\}$ - is a set of residue functions. Each of them maps vertices in a bigraph representing a state after an action to vertices in a bigraph representing a state where the action has happened;
- $Tra \subseteq Apl \times Agt \times Par \times Res$ - a transition relation. Elements of this set will hereafter be referred to as "transitions".

A tracking transition system can be interpreted as follows. The set Agt represents the reachable states of the system after performing any sequence of actions. At this stage, the time-lapse for objects is not considered, so the actual set of possible states might be smaller. Elements of Red can be viewed as preconditions necessary to occur in the states for any action to happen, modeled as a reaction rule. The binary relation Apl allows marking which action (Lab) can occur in which state of the system (Agt). The set of functions Par makes it possible to indicate which elements (vertices) of a state play what role in an action (reaction rule) with particular redex. Let us adopt the following convention, a state from which an action may be performed will be called an *input state*, while a state being the result of acting will be called an *output state*. The set of functions Res allows mapping elements of an output state corresponding to elements of an input state. If such a function is partial, then not all output state elements have the corresponding element in the input state. Thus, these "untracked" elements are new to the system. If, on the other hand, such a function is not surjective, then some of the elements in the input state are not existing in the output state. Thus these elements have vanished from the system. The set of transitions Tra utilizes the above elements to describe feasible alterations in the system.

D. State space

Having a tracking transition system (TTS), we can indicate roles in an action played by each element of an input state. Still, we can not determine how participating in a single activity affects the possibility of interacting with other system elements in different actions. To overcome this, a structure called *state space* has been proposed. It can be constructed as a directed multigraph from a TTS, where each arc has a *mission progress function* assigned to it. By mission, we understand any course of actions made by a swarm, leading to successful task completion.

A state space of a system consisting of n_o objects and n_s states is a tuple:

$$SS = (S, E, L, I, C, T, M_f)$$

where subsequent tuple's elements are as follows:

- $S \subset \mathbb{N}$ - is a set of states of the modelled system. Each state in the state space corresponds to exactly one element of Agt in the TTS;
- $E \subseteq S \times S$ - a multiset of arcs between states;
- L - a set of labels describing possible changes in the system. Each label unambiguously indicate a single transition in the TTS. To determine the order in which actions are evaluated another set is introduced $\mathbb{H} = \{l_t \mid l \in L, t \in \mathbb{N}\}$;
- $I \subset \mathbb{N}_1^2 \times \cdots \times \mathbb{N}_{n_o}^2$ - is a set of state-at-time (SaT) configurations. With this set it is possible to define order of objects (both active and passive) as well as the moment of time in which each of them is;
- $C \subset (I \times 2^{\mathbb{H}}) \cup \{0\}$ - a set of mission courses. Consists of the current SaT configuration and subsequent actions that have led to it. Element 0 indicates an unfeasible course of a mission and will be used as to mark states of the system that are unreachable by expanding an existing course of a mission;
- $T = \{f_i : C \times \mathbb{N} \to C \mid i \in \mathbb{N}\} \cup \{f_{null}\}$ - is a set of *mission progress functions*. Each of these functions takes as an argument the current course of a mission and the number of actions currently evaluated. A set $T_{i,j} \subset T$ consist of all mission progress functions from *i*th state to the *j*th state. Function f_{null} returns 0 (unfeasible course of a mission) regardless of input;
- $M_f : E \to T$ - a bijection assigning mission progress functions to arcs.

A walk in a state space from the state recognized as initial to one of the satisfying final states is *behavior policy*. An algorithm for finding all walks of specified length, in a state space has been presented in [2]. Below are defined its main components:

- $K_s^t = \sum_{i=1} c_i \quad c_i \in C$ - a finite sequence of mission courses of a length t to a state s. A function $n_K : K_s^t \to \mathbb{N}$ returns the number of elements in the provided sequence;
- $F_{i,j}^t = \sum_{f \in T_{i,j}} f(x,t) \quad i,j \in \{0, \cdots, n_s - 1\}, x \in C$ - a finite sequence whose elements are mission progress functions from *i*th state to the *j*th state. Each of these functions takes two arguments x and t. Only a value of t is know at the moment of constructing a sequence.
- $\mathbb{M}_K^t = \begin{bmatrix} K_0^t & \cdots & K_{n_s-1}^t \end{bmatrix}$
- $\mathbb{M}_F^t = \begin{bmatrix} F_{0,0}^t & \cdots & F_{0,(n_s-1)}^t \\ \cdots & \cdots & \cdots \\ F_{(n_s-1),0}^t & \cdots & F_{(n_s-1),(n_s-1)}^t \end{bmatrix}$

Additionally, two operations are defined:

- $K_s^t \circ F_{i,j}^t = \sum_{f \in F_{i,j}^t} \sum_{i=1}^{n_K(K_s^t)} f(c_i, t)$ - a convolution of sequences;
- $\mathbb{M}_K^{t+1} = \mathbb{M}_K^t \cdot \mathbb{M}_F^t$ - a multiplication of a matrix $\mathbb{M}_K^t$ by a matrix $\mathbb{M}_F^t$. Elements of the result matrix are calculated according to the formula:

$$K_j^{t+1} = \sum_{i=0}^{n_s-1} K_i^t \circ F_{i,j}^t$$

In order to find all behavior policies of a specified length t it necessary to perform just as many multiplications $\mathbb{M}_K^t \cdot \mathbb{M}_F^t$, with each multiplication the t parameter will be incremented and the matrix $\mathbb{M}_K^t$ will be the result of the previous product.

The number of steps required to find all walks of a specified length is dependent on four parameters:

- the length of sought walks - l;
- a sum of all mission progress functions - n_f;
- the number of states in a system - n_s;
- a sum of all mission courses in a matrix $\mathbb{M}_K^0$ - n_k;

and it will be denoted as $t^{(l)}(n_s, n_f, n_k)$.

A set of functions asymptotically bounded above by a function $g(n)$ is defined as:

$$O(g(n)) = \{f(n) \mid \exists c, n_0 \forall n > n_0 \quad 0 \leq f(n) \leq c \cdot g(n)\}$$

Using the above notations, the number of steps required to find all walks of a specified length is bounded above asymptotically by $l \cdot n_s^2 + n_k \cdot \frac{(n_f)^{l+1}-1}{n_f-1}$. This in turn can be written as:

$$t^{(l)}(n_s, n_f, n_c) \in O(l \cdot n_s^2 + n_k \cdot \frac{(n_f)^{l+1} - 1}{n_f - 1})$$

E. Schedule of actions

The last component required to define behavior for swarm elements is a *schedule of actions* for every active object. An algorithm verifying the correctness of a behavior policy has been described in [1]. It returns a set of ordered elements, each comprising three items. The first is a system state at a specified moment. The state is in the form of a bigraph. The second component is a function assigning *unique identifiers* to every vertex of the bigraph. The last component is a SaT configuration of the state. Knowing that every task element (whether this being an environmental element or an object) is distinguishable with a unique identifier, we want to define how each of these elements will behave. Unique identifiers for task elements are drawn from an infinite set $\mathcal{U}$.

Formally, a schedule of actions is a triple:

- A binary relation $Assignment : \mathcal{U} \times (T \times Lab \times \mathbb{N})$ which assigns mission progress functions, a label of an activity represented by that function and a moment of time to mission elements. The time here represents a moment the activity shall began to be executed.
- A function $role : \mathcal{U} \times (T \times Lab \times \mathbb{N}) \to \mathbb{V}_r$ assigning roles to task elements participating in an activity. Roles are indicated by vertices of the redex in a reaction rule

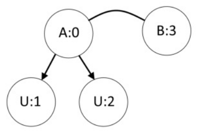


Fig. 1. Initial state of the system.

corresponding to a mission progress function provided as an argument.

- A function $duration : T \to \mathbb{N}$ which assigns how long does each of the activities take time.

Although, the main goal of the schedule of actions is to define how each element of a swarm shall behave during a mission, it is also possible to apply this structure to verify non-functional requirements.

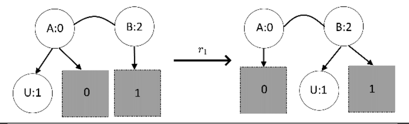
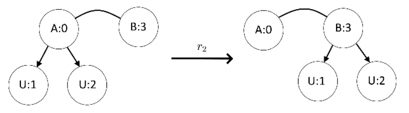
III. Results

The structures defined in Section II can be used to design the behavior of members of a UAV swarm. A schedule of actions allows to complete a task (hence fulfill functional requirements) and makes it possible to verify if non-functional requirements are also satisfied.

This chapter aims to demonstrate a simple example of practical usage of the structures described in this paper. To do this, the following scenario will be used. Let us consider a UAV swarm consisting of two drones whose goal is to move between two areas, namely area A and B. Assume they can do this in two ways. The first one allows each of the drones to move independently; this method of movement is power-efficient (it is assumed that moving this way for an hour consumes 20Wh) and takes two units of time for a drone to complete a transition from an area A to an area B. Internally, each drone is responsible for both obstacle detection and avoiding and determining the trajectory of a movement. The second way of moving between the areas requires cooperation. It is more energy-consuming (here, assume that each drone moving this method is a 50W device), but it is twice as fast, which means it requires only one unit of time for both drones to move from an area A to an area B. Internally, we can think of this kind of movement as an activity where one of the drones follows the other, but each is responsible for different functions during the movement. By functions, we understand things like obstacle detection and avoidance and determine the trajectory of the flight. The first kind of transition will be denoted as *r1*, while the second as *r2*. The unit of time in this example is to be understood as 6 minutes. Reaction rules representing both kinds of transitions are listed in Table I. The initial state of the system is depicted in Figure 1. Vertices with control U represent drones, while those with control A and B represent areas.

Figure 2 presents a graphical form of a tracking transition system for the system described above. Circles denote states of the system. Every arc represents a single transition (a possible action) between states. Types of actions (either *r1* or *r2*) along with functions par and res are placed inside each arc. There

TABLE I
Reaction rules for two kinds of drones movement. Both of the τ functions are identities.

Reaction rule	Graphical notation
r_1	
r_2	

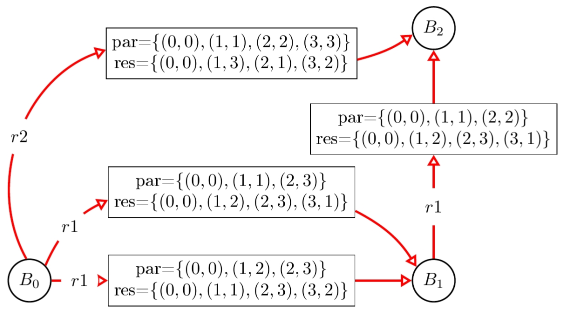

par={(0,0),(1,1),(2,2),(3,3)}
res={(0,0),(1,3),(2,1),(3,2)}
B_2
par={(0,0),(1,1),(2,2)}
res={(0,0),(1,2),(2,3),(3,1)}
r2
par={(0,0),(1,1),(2,3)}
res={(0,0),(1,2),(2,3),(3,1)}
r1
r1
B_0
r1
par={(0,0),(1,2),(2,3)}
res={(0,0),(1,1),(2,3),(3,2)}
B_1

Fig. 2. A graphical form of the TTS for the example system.

are three possible states of the system. First one (B_0) is equal to the bigraph from Figure 1. B_1 represents a situation where one of the drones has already moved from A to B. There are two transitions between B_0 and B_1 because any of the drones can move between the areas. Which one actually participated in the activity is indicated by the functions par and res. The B_2 state is equal to the situation where both drones are in the B area. Knowing that, there is only one transition between the state B_1 to the state B_2 because regardless of which one of the drones has previously moved, only the second one can participate in *r1* activity. If we differentiate functions par and res between the same states, only by their domains and codomains, rather than actual mappings, then there can only be one transition from the state B_0 to B_2 representing an execution of *r2* activity.

Figure 3 shows a state space transformed from TTS in Figure 2. Only drones are classified as objects in the task, so mission mission courses (elements of the set C) comprise of tuples of pairs. Mission progress functions definitions are listed below. A convention was adopted that each function takes the form of $f_x(c, t)$ where c is a current mission course, and t is the length of the course (number of included activities). A mission course can either be of the form $[\langle(a, x), (b, y)\rangle, \mathbb{H}']$ or 0. In the first case a and b are objects identifiers in a SaT, while x and y are moments of time these objects are at. Knowing this, each subsequent mission progress function will transform a given course of the mission as follows:

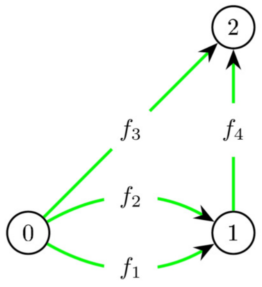


Fig. 3. A state space transformed from the TTS in Figure 2.

$$1)\ \begin{cases}\left[\left\langle (b,y),(a,x+2)\right\rangle, \mathbb{H}' \cup \{r1^1_{t+1}\}\right] & c \neq 0\\ 0 & c = 0\end{cases}$$

$$2)\ \begin{cases}\left[\left\langle (a,x),(b,y+2)\right\rangle, \mathbb{H}' \cup \{r1^2_{t+1}\}\right] & c \neq 0\\ 0 & c = 0\end{cases}$$

$$3)\ \begin{cases}\left[\left\langle (a,x+1),(b,y+1)\right\rangle, \mathbb{H}' \cup \{r2^3_{t+1}\}\right] & c \neq 0 \wedge x = y\\ 0 & c = 0 \vee x \neq y\end{cases}$$

$$4)\ \begin{cases}\left[\left\langle (b,y),(a,x+2)\right\rangle, \mathbb{H}' \cup \{r1^4_{t+1}\}\right] & c \neq 0\\ 0 & c = 0\end{cases}$$

A permutation of SaT components depends on the order of a bigraph's vertices that correspond to task objects, and it derives directly from function res and par. Incrementing x and y values results from the assumptions taken about the time needed for *r1* and *r2* activities to be completed. The set $\mathbb{H}'$ represents a currently constructed walk in the state space.

Having all mission progress functions defined we can proceed to constructing a behavior policy. Assuming that the task can only be started from the state denoted as *0* in Figure 2 and both drones commence the task execution at the same time (let us denote this moment as 0) the matrix $\mathbb{M}^0_K$ is of the form:

$$\mathbb{M}^0_K = \begin{bmatrix}[\langle(1,0),(2,0)\rangle,\emptyset] & 0 & 0\end{bmatrix}$$

Based on the state space in Figure 3 the matrix $\mathbb{M}^t_F$ takes the form:

$$\mathbb{M}^t_F = \begin{bmatrix} f_{null} & f_1 + f_2 & f_3 \\ f_{null} & f_{null} & f_4 \\ f_{null} & f_{null} & f_{null} \end{bmatrix}$$

Multiplying the above matrices we get all possible behavior policies for drones. Successive $\mathbb{M}^t_K$ matrices are as follows:

$$\mathbb{M}^1_K = \mathbb{M}^0_K \cdot \mathbb{M}^0_F$$

$$\mathbb{M}^1_K = \begin{bmatrix} 0 \\ [\langle(2,0),(1,2)\rangle,\{r1^1_1\}] + [\langle(1,0),(2,2)\rangle,\{r1^2_1\}] \\ [\langle(1,1),(2,1)\rangle,\{r2^3_1\}] \end{bmatrix}^\intercal$$

$[\quad]^\intercal$ denotes the transpose of a matrix $[\quad]$.

$$\mathbb{M}^2_K = \mathbb{M}^1_K \cdot \mathbb{M}^1_F =$$

$$\begin{bmatrix} 0 \\ 0 \\ [\langle(1,2),(2,2)\rangle,\{r1^1_1,r1^4_2\}] + [\langle(2,2),(1,2)\rangle,\{r1^2_1,r1^4_2\}] \end{bmatrix}^\intercal$$

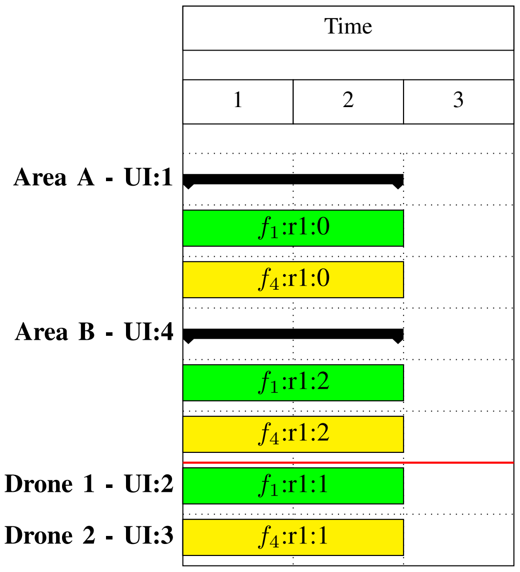


Fig. 4. A schedule of actions for all task elements based on the walk $0 \xrightarrow{f_1} 1 \xrightarrow{f_4} 2$. Areas A and B, being elements of the environment, can participate in multiple activities concurrently. Colors indicate different mission progress functions (activities). The text inside inform about a mission progress function, type of an activity it corresponds to and a role an element plays in the activity.

$\mathbb{M}^1_K$ matrix indicates there are two walks leading to the state 1 of the length 1. There there is also a single walk to the state 2 of the same length. The matrix $\mathbb{M}^2_K$ shows there are two walks to the state 2 of the length 2.

Having behavior policies, it is now possible to construct a schedule of actions for swarm elements. Using the algorithm described in [1] one can check that all found behavior policies produce a valid schedule. Figure 4 shows a schedule of actions from a walk of the form $0 \xrightarrow{f_1} 1 \xrightarrow{f_4} 2$. The formal definition of the same schedule is as follows:

- $Assignment = \begin{Bmatrix} (1,(f_1,\mathrm{r1},0)),(1,(f_4,\mathrm{r1},0)), \\ (2,(f_1,\mathrm{r1},0)),(2,(f_4,\mathrm{r1},0)), \\ (3,(f_1,\mathrm{r1},0)),(4,(f_4,\mathrm{r1},0)) \end{Bmatrix}$
- $role = \begin{Bmatrix} ((1,(f_1,\mathrm{r1},0)),0),((1,(f_4,\mathrm{r1},0)),0), \\ ((4,(f_1,\mathrm{r1},0)),2),((4,(f_4,\mathrm{r1},0)),2), \\ ((2,(f_1,\mathrm{r1},0)),1),((3,(f_4,\mathrm{r1},0)),1) \end{Bmatrix}$
- $duration = \{(f_1,2),(f_4,2)\}$

The developed framework allows scaling a task both horizontally and vertically. Scaling a task horizontally is expanding the current task by including more elements. For example, scaling the example covered in this section horizontally can increase the number of drones. To modify a task this way, the designer has to change the system's initial state. This requires a reconstruction of the tracking transition system and the state space. Hence, any modifications of this kind are very intrusive to the existing model because it requires a change in a structure that every other relies on. Vertical scaling is expanding a task by new functional requirements or additional stages. It is significantly less intrusive to an already constructed model because it does not imply changes to current structures. A new

Algorithm 1 Finding the moment of last finished action

```
Require: $assignment; duration$
1: $current\_max \leftarrow 0$;
2: $unchecked = assignment$;
3: while $unchecked \neq \emptyset$ do
4:     $u, (fun, act, start) \leftarrow unchecked.Pick$
5:     $req\_time \leftarrow duration.Find(fun)$
6:     if $current\_max < start + req\_time$ then
7:         $current\_max = start + req\_time$;
8:     end if
9: end while
10: return $current\_max$;
```

stage may be treated as a new task starting where the previous one has ended, both in terms of bigraph representation of the initial state and the SaT configuration in the matrix $\mathbb{M}^0_K$. An example of vertical scaling of the task from this chapter can consist of requiring drones to return to the area A.

The last aspect of behavior designing that our framework covers is verifying non-functional requirements. While verification of functional requirements comes down to finding a certain bigraph in a TTS, verification of non-functional requirements is more complex. It is mainly because such verification depends on the insight how system changes internally while the task is executed. This implies that the verification can only be performed after a schedule of actions is available. For the task in this chapter, two non-functional requirements will be considered. The first one limits the time it can take to complete the task to 8 minutes. Whether this requirement is satisfied can be verified using the $Assignment$ relation and the $duration$ function, both components of a schedule of actions. Algorithm 1 shows how both structures can be utilized to verify the non-function requirement. It calculates the moment when each activity ends and chooses the latest one. This moment is later converted from abstract units to time in the real world. In this example, one unit of time is equal to 6 minutes. The execution of the task according to the schedule of activities in Figure 4 takes two units of time. It means that the last activity will be finished twelve minutes after the start of the task, which means that the walk $0 \xrightarrow{f_1} 1 \xrightarrow{f_4} 2$ does not produce a schedule satisfying this non-functional requirement. A fix might be to use a schedule based on another behavior policy and verify if it satisfies the requirement.

Another example of non-functional requirement regards staying within power source capacity during a mission. Let us assume that each drone is powered with a 4.5 Wh battery. We can use $role$ and a higher-order function that transforms this function to a value representing the maximum energy consumption by any task element during a mission. The mention higher-order function can be of the form:

$$f(role) = max \circ power(role)$$

where:

- $power$ is a higher-order function that assigns power consumption to each task element (represented by their unique identifiers). It is important to remember that not all elements participating in an activity use energy. For example areas between which drones are moving do not consumes energy during that process. So it is not only the label of an activity that indicates the power usage but also the role played by an element participating in it. A role is indicated by a vertex in the redex of a reaction rule representing an activity. A function calculating power usage for each task element is defined as a set of pairs (u, p), with u being a unique identifier of a task element and p being a power usage during a whole mission. This function is of the form:

$$\left\{ (u,p) \mid p = \sum_{((u,(f,a,t)),v) \in role} cons(a,v) \right\}$$

The *cons* function calculates energy consumption in watt-hours given a type of an activity and a role in it:

$$cons(a,v) = \begin{cases} 20 \cdot 0.2 & \text{if } a = \text{r1} \wedge v = 2 \\ 50 \cdot 0.1 & \text{if } a = \text{r2} \wedge (v = 1 \vee v = 2) \\ 0 & \text{in other cases} \end{cases}$$

For example, a drone participating in *r1* activity for an hour uses 20Wh. After adjusting for a fact that a single transition between a pair of areas A and B takes 2 units of time (12 minutes = 0.2 hour), energy consumption of that single transition is equal to $20 \cdot 0.2 = 4$ Wh. A role of element participating in an activity is checked against set of vertices indicating drones. The vertex identifier of a drone in *r1* activity denoted with reaction rule from Table I is 2, hence the condition $v = 2$.
- $max : (\mathcal{U} \rightarrow \mathbb{N}) \rightarrow \mathbb{N}$ is a higher-order function that returns maximum value in codomain of the input function.

For the example covered in this section, function $power$ takes the following form:

$$power = \{(1,0),(2,4),(3,4),(4,0)\}$$

It says that task elements with unique identifiers 1 and 4 do not use any energy, while elements with identifiers 2 and 4 use 4Wh during the whole mission.

IV. CONCLUSIONS

Designing a behavior of robotic swarms is an iterative process, where subsequent iterations consider the designed behavior with a higher level of detail. This paper presents a framework that enables design of behavior for a UAV swarm members within a single iteration. It is highly automated and enables leaving the designing process to a specialized tool after providing initial task parameters and a swarm. It takes into account verification of both functional and non-functional requirements, as well as scaling an existing task vertically and horizontally.

Practical limitations of the described method come down to the computational capacity and complexity of the modeled system. The time needed for developing behavior for elements of a system consisting of thousands of states and a few dozens of thousands of transitions is in the order of minutes. It has to be emphasized that even for such systems, dozens of gigabytes of available memory are required. The most significant limitation of the modeled tasks is the lack of active objects that can not be fully controlled, such as humans.

There are three directions for further developing the method described in this article. The first one is by developing a high-quality tool allowing for the convenient development of schedules of actions for system elements. The currently available software shall be regarded as a proof of concept. The second direction is to improve the efficiency of methods in the behavior designing process. As was already mentioned, a relatively small system can exceed the capabilities of a modern desktop computer. There are currently intense studies on algorithms for bigraphs matching problems, which may further accelerate sub-processes in the framework. The last direction to improve is to extend the method to allow active objects not fully controlled in the system or to enable a variable number of objects (both active and passive) during a mission.

References

[1] P. Cybulski, Z. Zieliński, “Design and Verification of Multi-Agent Systems with the Use of Bigraphs“, Applied Sciences, https://www.mdpi.com/2076-3417/11/18/8291, DOI:10.3390/app11188291

[2] P. Cybulski, Z. Zieliński, “UAV Swarms Behavior Modeling Using Tracking Bigraphical Reactive Systems“, Sensors, https://www.mdpi.com/1424-8220/21/2/622, DOI:10.3390/s21020622

[3] R. Milner, 2009, “The Space and Motion of Communicating Agents“, ISBN=978-0-521-73833-0, Cambridge University Press, DOI:10.1017/CBO9780511626661

[4] T. Sheridan, W. Verplank, “Human and Computer Control of Undersea Teleoperators“, 1978.

[5] S. Benford, M. Calder, T. Rodden, M. Sevegnani, “On Lions, Impala, and Bigraphs: Modelling Interactions in Physical/Virtual Spaces“, ACM Trans. Comput.-Hum. Interact., May 2016, DOI:10.1145/2882784

[6] J. Krivine, R. Milner, A. Troina, “Stochastic Bigraphs“, Electronic Notes in Theoretical Computer Science, year = 2008, https://www.sciencedirect.com/science/article/pii/S1571066108004003, DOI:10.1016/j.entcs.2008.10.006

[7] O. H. Jensen, “Mobile Processes in Bigraphs“, 2006, https://www.cl.cam.ac.uk/archive/rm135/Jensen-monograph.pdf.

[8] P. Stone, M. Veloso, Multiagent Systems: A Survey from a Machine Learning Perspective. Autonomous Robots 8, 345–383 (2000). DOI:10.1023/A:1008942012299

[9] H. Hamann. (2018). Swarm Robotics: A Formal Approach. DOI:10.1007/978-3-319-74528-2.

[10] Y. Mohan and S. G. Ponnambalam, "An extensive review of research in swarm robotics," 2009 World Congress on Nature & Biologically Inspired Computing (NaBIC), 2009, pp. 140-145, DOI:10.1109/NABIC.2009.5393617.

[11] Brambilla, Manuele & Ferrante, Eliseo & Birattari, Mauro & Dorigo, Marco. (2013). Swarm Robotics: A Review from the Swarm Engineering Perspective. Swarm Intelligence. 7. 1-41. DOI:10.1007/s11721-012-0075-2

[12] Navarro, Iñaki & Matía, Fernando. (2013). An Introduction to Swarm Robotics. ISRN Robotics. 2013. DOI:10.5402/2013/608164

[13] Iocchi, Luca & Nardi, Daniele & Salerno, Massimiliano & Hannebauer, Markus & Wendler, Jan & Pagello, Enrico. (2001). Reactivity and Deliberation: A Survey on Multi-Robot Systems. 2103. 9-32. DOI:10.1007/3-540-44568-4_2.

[14] Nedjah, Nadia & Silva Junior, Luneque. (2019). Review of methodologies and tasks in swarm robotics towards standardization. Swarm and Evolutionary Computation. 50. 100565. DOI:10.1016/j.swevo.2019.100565

[15] Bayindir, Levent. (2015). A Review of Swarm Robotics Tasks. Neurocomputing. 172. DOI:10.1016/j.neucom.2015.05.116

[16] Dudek, G., Jenkin, M.R., Parker, L.E., & Lin, L. (2003). A Taxonomy of Multirobot Systems.

[17] Crespi, V., Galstyan, A.G., & Lerman, K. (2008). Top-down vs bottom-up methodologies in multi-agent system design. Autonomous Robots, 24, 303-313.

[18] A. Kolling, P. Walker, N. Chakraborty, K. Sycara and M. Lewis, "Human Interaction With Robot Swarms: A Survey," in IEEE Transactions on Human-Machine Systems, vol. 46, no. 1, pp. 9-26, Feb. 2016, DOI:10.1109/THMS.2015.2480801

[19] Valentini, Gabriele. (2017). Achieving Consensus in Robot Swarms. DOI:10.1007/978-3-319-53609-5

[20] Francesca, Gianpiero & Brambilla, Manuele & Brutschy, Arne & Trianni, Vito & Birattari, Mauro. (2014). AutoMoDe: A novel approach to the automatic design of control software for robot swarms. Swarm Intell. 8. 1-24. DOI:10.1007/s11721-014-0092-4

[21] Brambilla, Manuele & Brutschy, Arne & Dorigo, Marco & Birattari, Mauro. (2014). Property-Driven Design for Robot Swarms. ACM Transactions on Autonomous and Adaptive Systems. 9. 1-28. DOI:10.1145/2700318

[22] E. Pereira, C. Potiron, C. M. Kirsch and R. Sengupta, "Modeling and controlling the structure of heterogeneous mobile robotic systems: A bigactor approach," 2013 IEEE International Systems Conference (SysCon), 2013, pp. 442-447, DOI:10.1109/SysCon.2013.6549920

[23] Bachrach, Jonathan & Mclurkin, James & Grue, Anthony. (2008). Protoswarm: A language for programming multi-robot systems using the amorphous medium abstraction. 2. 1175-1178.

[24] Pianini, D., Viroli, M., & Beal, J. (2015). Engineering multi-agent systems with aggregate computing.

[25] Byrski, Aleksander & Drezewski, Rafal & Siwik, Leszek & Kisiel-Dorohinicki, Marek. (2015). Evolutionary multi-agent systems. The Knowledge Engineering Review. 30. 171-186. DOI:10.1017/S0269888914000289

[26] Floreano, Dario & Mattiussi, Claudio. (2008). Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies. ISBN:978-0262062718

[27] Spears, William & Spears, Diana & Hamann, Jerry & Heil, Rodney. (2004). Distributed, Physics-Based Control of Swarms of Vehicles. Auton. Robots. 2. DOI:10.1023/B:AURO.0000033970.96785.f2

[28] Çelikkanat, Hande & Sahin, Erol. (2010). Steering self-organized robot flocks through externally guided individuals. Neural Computing and Applications. 19. 849-865. DOI:10.1007/s00521-010-0355-y

[29] J. Yu, S. M. LaValle and D. Liberzon, "Rendezvous Without Coordinates," in IEEE Transactions on Automatic Control, vol. 57, no. 2, pp. 421-434, Feb. 2012, DOI:10.1109/TAC.2011.2158172

[30] Bullo, F & Cortés, J & Martínez, S. (2009). Distributed Control of Robotics Networks. ISBN:9780691141954

[31] Souad, Marir and Faiza, Belala and Nabil, Hameurlain. Formal Modeling IoT Systems on the Basis of BiAgents and Maude, DOI:10.1109/ICAASE51408.2020.9380126

[32] Archibald, Blair and Shieh, Min-Zheng and Hu, Yi-Hsuan and Sevegnani, Michele and Lin, Yi-Bing, BigraphTalk: Verified Design of IoT Applications, DOI:10.1109/JIOT.2020.2964026

[33] Muffy Calder and Alexandros Koliousis and Michele Sevegnani and Joseph Sventek, Real-time verification of wireless home networks using bigraphs with sharing, DOI:10.1016/j.scico.2013.08.004

[34] Mansutti, Alessio and Miculan, Marino and Peressotti, Marco", editor="Magoutis, Kostas and Pietzuch, Peter, Multi-agent Systems Design and Prototyping with Bigraphical Reactive Systems, DOI:10.1007/978-3-662-43352-2_16

[35] DIB, Ahmed Taki Eddine and MAAMRI, Ramdane, Bigraphical Modelling and Design of Multi-Agent Systems, DOI:10.1145/3467707.3467762

[36] Pereira, Eloi and Potiron, Camille and Kirsch, Chirstoph M. and Sengupta, Raja, Modeling and controlling the structure of heterogeneous mobile robotic systems: A bigactor approach, DOI:10.1109/SysCon.2013.6549920