Transferable Programs and Reactions for Modeling IoT Network

Šárka Vavrečková[1]

[1]Silesian University in Opava, Bezručovo nám. 13, Opava, Czech Republic

Abstract

Membrane systems and their different variants (e.g. P Colonies with transferable programs) can be used to model various processes if we are satisfied with simple rules manipulating the elements of a set of objects. However, these systems were designed more for computational purposes, so we may encounter problems when solving non-numerical problems. Reaction systems, on the other hand, come with the concept of reactants and inhibitors, which we can also use in simulating (typically dynamic) processes. In this paper, we compare these two concepts and propose a new type of system: IR Colonies, which are inspired by both of these concepts. The IR Colonies are designed to be mainly applicable for modeling communication in the Internet of Things networks. In the paper, the reader will find both a proposed definition of IR Colonies and an example of a network model with several IoT devices.

Keywords

IR Colony, P System, P Colony, Transferable program, Reaction system, Network, IoT

1. Introduction

Membrane computing (introduced by Gheorghe Păun in 1998) is a framework for modeling parallel distributed processing. Information about this paradigm is available in [1, 2, 3], or the bibliography at http://ppage.psystems.eu/ [2024-07-04]. Membrane systems are based on the hierarchical structure of cell membranes and can be used to model distributed computing. Mathematical models of membrane systems have been called P Systems.

P Colony (introduced in [4] in 2004) is a simple computational model based on membrane systems. On the basic variant, the environment containing objects of a given type is shared by agents that also contain objects inside their internal environment and are equipped with programs consisting of rules. The programs allow the agents to influence both their own environment and also the shared environment.

P Colonies with transferable programs were introduced in [5] and additional examples and discussions can be found in [6]. In the given concept, programs can be transferred between an agent and the environment and vice versa, not only objects.

Reaction systems were introduced in [7], and in [8, 9] we can find information about networks of reaction systems. Reaction systems (R Systems) are a formal framework intended for modeling interactions between biochemical entities. The intention was to simulate the coexistence of two reverse mechanisms – using reactants and inhibitors.

A network of reaction systems is a graph with reaction systems as its nodes. Each reaction system can be affected by the reactions of its neighbours.

There are several papers that combine the concept of P Systems (or P Colonies) and R Systems. In [10] the authors compare the two mentioned mathematical models and construct a P Colony simulating processes taking place in a reaction system. In [11] we can find the concept of PR Systems, where reactions from reaction systems are applied in membranes of a P System.

In [11] P Systems are referred to as a quantitative model because they focus primarily on computation, whereas R Systems are referred to as a qualitative model because they focus more on evolution. This does not mean that P Systems are of poor quality, just that their focus is different (on calculations, the result is a number).

In [12] we introduce a membrane system working as a communication interface between IoT devices, but we encounter problems with implementing some properties of the resulting system based on P Systems. P Systems have been found to be useful for this purpose, however, a structure of membranes with unit rules (i.e. rules with a single symbol or object on both sides) is not flexible enough and some operations are not easy to implement. On the other hand, membrane systems naturally represent the tree structure of a network interconnecting IoT devices.

In [6] we discuss properties of systems derived from P Systems – P Colonies, adding the concept of transferable programs introduced in [5]. In [6] only the basic idea is outlined, in [6] we develop the idea and give further examples. The capabilities of P Colonies with transferable programs are compared with the properties of the concept of osmotic computing and the functionality of computer viruses, showing that similarities can be found

ITAT'24: Information technologies – Applications and Theory, September 20–24, 2024, Drienica, Čergovské vrchy

sarka.vavreckova@fpf.slu.cz (Š. Vavrečková)

between the three concepts. However, for the purposes of this paper, we are primarily interested in P Colonies with transferable programs.

The question is to what extent it would be possible to replace the P Systems in the concept proposed in [12] by P Colonies with transferable programs. The concept of transferable programs naturally lends itself to e.g. the distribution of updates, but on the other hand, the possibility of representing the tree structure of the network between IoT devices by membranes gets lost here – agents in P Colony cannot be nested. And the environment is only a repository of objects and programs, it is not able to execute its own rules. Here, networks of reaction systems, specifically a suitable combination with P Colonies with transferable programs, could be helpful.

This section is followed by the preliminaries section, in which we briefly introduce P Colonies, P Colonies with transferable programs, reaction systems and networks of reaction systems. We also briefly introduce the world of IoT devices.

The subsequent section 3 is a brief comparison and evaluation of the properties of P Colonies (with transferable programs) and (networks of) reaction systems. In particular, we observe states, rules, processes, possibilities of cooperation of involved entities and sharing, also possibilities for conditioning of events taking place in the system.

Section 4 proposes a definition of a new type of system: the IR colony. The section describes and explains various aspects of this system. IR Colony is used in Section 5 to create an outline of a model of communication in a network with several devices.

2. Preliminaries

We assume the reader to be familiar with the basics of the formal language theory[13] and membrane computing [3].

We denote the length of a word w by $|w|$, and also the number of elements in a (multi)set S by $|S|$. The empty word is represented by the symbol ε, so $|\varepsilon| = 0$.

For details and definitions of the graph theory, we can refer e.g. to https://www.britannica.com/topic/graph-theory [2024-07-08]. We denote the set of all nodes from which an edge leads to a node v by $\text{in}(v)$.

2.1. P Colonies

P Colonies enrich the concept of P Systems by agents evolving activities according to defined programs. On the contrary, the complex structure of membranes was abandoned. An agent is actually analogous to a membrane within an environment of the main membrane so that the P Colony can be viewed as a two-level membrane structure.

Definition 1 ([5]). *A P Colony of capacity k, $k \geq 1$, is a construct $\Pi = (A, e, f, v_E, B_1, \ldots, B_n)$ where*

- *A is an alphabet, its elements are called objects,*
- *$e \in A$ is the environmental object,*
- *$f \in A$ is the final object,*
- *v_E is a finite multiset over $A - \{e\}$ called the initial state of the environment,*
- *B_i, $1 \leq i \leq n$, are agents where each agent $B_i = (o_i, P_i)$ is defined as follows:*
  - *o_i is the initial state of the agent, a multiset over A consisting of k objects,*
  - *$P_i = \{p_{i,1}, \ldots, p_{i,k_i}\}$ is a finite set of programs where each program consists of k rules, the rules can be in one of the following forms:*
    - *$a \to b$, $a, b \in A$ called an evolution rule,*
    - *$c \leftrightarrow d$, $c, d \in A$ called a communication rule,*
    - *r_1/r_2 called a checking rule, r_1, r_2 are both evolution or communication rules.*

The evolution rules are of the form $a \to b$. This type of rule allows the agent to influence its internal environment: an object a inside the agent's environment is rewritten to the specified object b.

The communication rules ($c \leftrightarrow d$) are intended for communication between the given agent and the environment. An object c inside the agent is swapped with the given object d in the environment.

The checking rules (r_1/r_2) are composed of two rules r_1 and r_2 of any of the previous two types. The first rule has a higher priority to apply, and if the first rule cannot be executed, the second rule in order may be executed.

The configuration of a P Colony Π with capacity k is an $(n+1)$-tuple of multisets o_i for $1 \leq i \leq n$ for the agents B_i, and v_E for the environment

$$(w_1, \ldots, w_n, w_E)$$

where $w_i \in A^*$, $|w_i| = k$, $w_i \in A^*$, $w_E \in (A - \{e\})^*$.

Several derivation modes have been defined, in [5] and [6] the maximally parallel derivation mode is primarily taken into account where all agents can work parallelly in each derivation step (each agent non-deterministically chooses one of its programs with applicable rules). The calculation halts if no agent finds an applicable program.

2.2. P Colonies with Transferable Programs

As mentioned above, the concept of transferable programs for P Colonies has been introduced in [5]. A transferable program is an ordered pair

$$(\langle \text{simple\_program} \rangle \ ; \ \{\text{conditions}\})$$

located inside an agent or the environment. The program can be transferred from the agent to the environment and vice versa (from the source to the destination, depending on the direction), and the stated condition determines under what circumstances this transfer can occur. Two types of conditions are admissible: an object condition and a program condition.

The "object" condition specifies which objects must (or must not, with the negation symbol) be present in the destination for the program to be transferable. This condition is formed by multisets of objects, the size of the multisets is equal to the capacity of the P Colony.

The "program" condition specifies programs that must (or must not) be present in the destination for the given program to be transferable.

Example 1. Let Π be a P Colony of capacity 2. The capacity is reflected in both the number of objects in agents' environments and the number of rules in each program.

An example program with an object condition inside an agent can be as follows:

$$(\langle a \to b;\ c \leftrightarrow d \rangle \ ; \ \{dm, \neg ag\})$$

This means that the given program is transferable only if at least one occurrence of d and one occurrence of m is present and there is no ag pair, both in the environment (the shared environment is the destination).

An example program with a program condition inside the environment can be as follows:

$$(\langle a \to b;\ c \leftrightarrow d \rangle \ ; \ \{\langle a \leftrightarrow b;\ b \to c \rangle\})$$

The given program can be transferred into an agent only if the agent contains the program $\langle a \leftrightarrow b;\ b \to c \rangle$.

A combination of both types of conditions in a single transferable program is allowed.

According to [5], a program can be classified as *permament*. When a permanent program is being transferred, the program remains in the original location and the copy is included in the destination location, but the program loses this property at the new location. When a non-permanent program is being transferred, it is removed from its original location.

In each computation step, an agent can either apply one of its applicable programs or transfer one of the programs in or out[5]. This implies that the transfer of programs is actually an analogy of programs.

2.3. R Systems and their Networks

A reaction takes place over a set of symbols, reactants. A reactant enters a (chemical or other) reaction and changes into a product of the given reaction. The reaction may not take place under certain circumstances if an inhibitor is present in the environment (set), i.e. a substance that slows down or prevents the reaction.

Definition 2 (according to [9]). *A reaction over a finite nonempty set S is a triple $a = (R, I, P)$ where*

- *$R \subseteq S$ is a set of reactants, $R \neq \emptyset$,*
- *$I \subseteq S$ is a set of inhibitors, $R \cap I = \emptyset$,*
- *$P \subseteq S$ is a set of products, $P \neq \emptyset$.*

The set S is called the background set.

Denote by $\text{res}_a(T)$ the result of applying the reaction a to the set T, we call the function $\text{res}_a()$ the result function. A reaction $a = (R, I, P)$ with $P = \text{res}_a(T)$ is enabled *in a configuration $T \subseteq S$ iff*

- *$R \subseteq T$, and*
- *$I \cap T = \emptyset$.*

If $\text{res}_a(T) = \emptyset$, the reaction a is not enabled in the configuration with the set T.

We write R_a, I_a, P_a, if we can stress the relationship to the reaction a.

If A is the set of all reactions a over the background set S, we denote res_A set of results of all reactions belonging to A, so $\text{res}_A = \cup_{a \in A} P_a$.

A reaction system *is an ordered pair $\mathcal{A} = (S, A)$ where the finite nonempty set of reactions A is built over the background set S.*

In [9] we can also find the condition $I \neq \emptyset$ for the set of inhibitors, and the author states that by omitting this condition we get an equivalent system, so here we stick to the shorter definition without this condition.

As we can see, the core of each reaction a is its function res_a. The input of this function is a set $T \subseteq S$, all members of R_a are in T, no member of a set I_a should be a part of T, and the output of the function is the set P_a. The function should be computable, we can represent it by rules, program code, etc. according to the particular use of the system.

Definition 3 (according to [9]). *An* interactive process *π in a reaction system $\mathcal{A}$ is a pair (γ, δ) such that*

- *$\gamma = C_0, C_1, \ldots, C_n$ (the context sequence),*
  $\delta = D_0, D_1, \ldots, D_n$ (the result sequence),
- *$C_i \subseteq S,\ D_i \subseteq S,\ 0 \leq i \leq n$,*
- *$D_i = \text{res}_A(C_{i-1} \cup D_{i-1})$ for each $1 < i \leq n$.*

The state sequence of the interactive process π is $W_0, \ldots, W_n$ where $W_0 = C_0$ is the initial state of π and $W_i = C_i \cup D_i$ for all $1 \leq i \leq n$.

The definitions show that the output of the system depends on the initial state W_0 (resp. the first context C_0). For two different initial states, we get two different outputs.

For the purposes of this paper, it is very practical that symbols (potential reactants) can be continuously added to the system (by the members of the context sequence), corresponding to processes occurring in dynamic systems.

Definition 4 (according to [9]). *A network of reaction systems is a tuple $\mathcal{N} = (G, \mathcal{F}, \mu)$ where*

- *$G = (V, E)$ is a finite graph where V is a finite set of nodes (vertices) and E is a finite set of edges,*
- *$\mathcal{F}$ is a nonempty finite set of reaction systems,*
- *$\mu\colon V \to \mathcal{F}$ is a location function, assigning reaction systems to nodes.*

Moreover, if G is undirected, then it is connected. If G is directed, then it is weakly connected.

Each reaction system can have a different background set. Denote S^j the background set of the reaction system $v^j,\ 1 \leq j \leq m$.

The superscript ("x^j") in the following paragraphs does not denote multiplicity, it denotes membership to a j^{th} R System in the network.

Definition 5 (according to [9]). *Let $\mathcal{N} = (G, \mathcal{F}, \mu)$ be a network of reaction systems with $|V| = m,\ m \geq 1$. For $n \in \mathbb{N}^+$, an interactive (n-step) network process is a tuple $\Pi = (\pi^1, \ldots, \pi^m)$ where for all vertices $v^j,\ 1 \leq j \leq m$:*

- *$\pi^j = (\gamma^j, \delta^j), \gamma^j = C_0^j, C_1^j, \ldots, C_n^j$ (the context sequence of the j^{th} vertex), $\delta^j = D_0^j, D_1^j, \ldots, D_n^j$ (the result sequence of the j^{th} vertex),*
- *$C_i^j \subseteq S^j,\ D_j^j \subseteq S^j,\ 0 \leq j \leq n$,*
- *$C_i^j = S^j \cap \left(\bigcup_{1 \leq k \leq m} D_{i-1}^k \mid v^k \in \mathrm{in}(v^j)\right)$,*
- *$D_i^j = \mathrm{res}_{A^j}\left(C_{i-1}^j \cup D_{i-1}^j\right)$ for each $1 \leq i \leq n$.*

Moreover, if $\mathrm{in}(v^j) = \emptyset$, then $C_i^j = \emptyset,\ 1 \leq i \leq n$.

The key for us is the ability of the individual components of the network (here reaction systems) to communicate with each other. It may look a bit complicated in the definition, but the point is that at each step of the process, each system can send the result of its own reaction (product) to another system using the edge that connects them.

2.4. IoT Concepts

IoT (Internet of Things) devices are small, simple devices that emphasize the ability to connect to other devices, interact with each other, and automate their operation. Their programs are not usually complex (except perhaps for security devices), and their operation consists mostly of transmitting simple data (either one-time or at regular intervals) or, conversely, receiving simple data and then reacting. For example, a thermometer sends the current temperature to network at regular intervals, to which a window controller can respond by opening or closing a window, or a heating or air conditioning controller can start or stop a related device. In [14] several definitions of IoT network can be found, a more compact definition is provided in [12].

In IoT device networks, we encounter one of two types of communication: Request-Response or Publisher-Subscriber. The Request-Response model comes from the decentralized world of WWW networks. The Publisher-Subscriber model is better adapted to requests of IoT networks or other automated systems, however, usually a central control device is being used. More details about IoT network communication models, including protocols, can be found in [15].

3. Comparison of Base Systems

In this section, we compare the capabilities of P Colonies with transferable programs and R Systems and/or networks of R Systems.

As mentioned above, various mathematical models are either quantitative (as P Systems) or qualitative (as R Systems). P Colonies can be considered as something between quantitative and qualitative concepts, but close to the first one. We need a system somewhere between as well, but close to the second one, to be more suitable for modeling and simulations.

We can find some parallels between the concepts of P Colonies with transferable programs and reaction systems.

States. In P Colonies, each agent has its own state, and the shared environment has the state as well (all states are the parts of the configuration). The agents can evolve their own state using evolution rules, and influence the state of the shared environment using communication rules. Agents cannot directly affect the state of other agents, only indirectly through the environment.

Each R System has its own state too. Networked R Systems can both evolve their own state and influence the states of neighbouring nodes through reactions, and they can influence the states of other R Systems in the network only indirectly through their neighbours.

Rules, Processes, Functions. With using transferable programs, the sets of programs inside agents and shared environment are changeable. But programs can only be transferred, not new ones created. When defining a new P Colony, the format of the rules and programs is predetermined. Even the number of objects in the agents' environments and the number of rules in programs is given by the capacity of the P Colony.

Every R System has its own set of reactions, and it is not possible to change or upgrade it subsequently. There is no strict form for the function res_a describing the reaction a inside a reaction system, this function should only be computable. However, the purpose of this function is to process a set of reactants and transform them into a set of products, so this function can also be represented by a set of rules (not necessarily simple or regular).

When defining a new R System, we have a relatively free hand and can better customize the system to what we need to model (which is very practical for a qualitatively oriented system intended for simulations of real systems); some specific simulated systems cannot be represented by regular or context-free rules.

Conditionality of Transfer or Reaction. The original programs with rules in P Colony agents are static, but transferable programs add dynamism. The object conditions for transferable programs are analogous to reactants (positive conditions) and inhibitors (conditions with negation) used in R Systems.

We can also consider as conditionality in R Systems the fact that the reaction is only enabled in certain configurations.

P Colonies go a bit further, allowing the transfer to be conditioned not only by objects but also by the (non)presence of rules in the agent's environment or in the shared environment (depending on the transfer direction).

Cooperation and Sharing. In P Colonies with transferable programs, agents cooperate through a shared environment. It is a two-level hierarchy, the shared environment serves as a repository for objects and rules. Individual agents do not communicate directly with each other. This communication model corresponds to an infrastructure with a central control component represented by the shared environment.

R Systems themselves do not have a defined neighbourhood. However, the network of R Systems precisely defines the connections of R Systems as nodes of a graph. Each R System is adjacent to at least one different R System, all nodes communicate right with their neighbours, with respect to edge directions. There is no shared environment. This communication model allows using various structures: centralized, decentralized, and distributed. The potential central component can be represented by one of the network nodes, with an adjustment of the network structure (e.g. a tree structure).

4. IR Colonies with Transferable Programs and Reactions

In [11], the authors have designed PR Systems in such a way that the rules of the P System have been replaced by reactions, i.e. each membrane has an associated set of reactions. This concept is interesting, but not very suitable for our purposes (optimization for IoT network simulation).

From the definition of the network of R Systems, we take:

- definition of infrastructure using a graph,
- system of reactions with reactants and inhibitors, the rules will follow the computational function with variable input arguments (not only static objects),
- partly the principle of processes, context sequence and result sequence,
- flexibility in the number of symbols/objects inside agents' states and rules in programs.

From the definition of the P Colonies with transferable programs, we take:

- the system of agents and shared environment as storage for objects and programs,
- a set of objects as agent state, supplemented by the ability to store objects in the shared environment,
- some rule types in programs, transferable programs.

Since we want to design a quality-oriented system, we will abandon the capacity parameter. Each agent has a specific role for which it needs a specific number of objects in the environment and a differently complex program. While abandoning capacity means that the ability to compare with other systems and to represent various characteristics of the system numerically is degraded, but these characteristics are not important for our purposes.

Because, unlike other similar systems, we add variable properties to objects, it makes no sense to work with multisets. In the definition, we can only find sets, which will ensure the determinism of each operation and make programming easier.

Object Properties. Our system has a shared alphabet of objects, but each object can have variable properties, numbers from $\mathbb{Z}$. For example, an agent representing a device has an object in its state for the version of the

system installed in the agent, the specific version number is a property of this object. A rule in some program will work with an object specifying the version, the (variable) argument will be a specific version number. Or an agent simulating a weather station products an object for temperature with the current temperature as its property.

The intersection operation on sets of objects with properties corresponds to the operation without using properties (properties are ignored when comparing). E.g. for an object a with a property x and a set of objects Σ: $\{a(x)\} \cap \Sigma \cong \{a\} \cap \Sigma$. The exception is the case when the same object is on both sides of the operator, but with different properties: $\{a(x)\} \cap \{a(y)\} = \emptyset$ if $x \neq y$. All membership operators (e.g. $\subseteq$) work with properties in the same way.

The unification operation applied on a set of objects with properties is not commutative. If the same object is in both sets, but with different properties, the object from the set on the right side of the union operator will be the member of the resulting set. E.g. $\{a(x), b(y)\} \cup \{b(z), c(r)\} = \{a(x), b(z), c(r)\}$. This corresponds to updating properties in the right-to-left direction.

If the same object has a property in only one of the sets, then the result of the union will contain the object with the property. E.g. $\{a(x)\} \cup \{a, b\} = \{a(x), b\}$.

IR Colony and Agents. An IR Colony (IoT Reaction Colony) is a construct $\Pi_{IR} = (\Sigma, \mathcal{A}, G, \mu, W_0, P_0)$ where

- Σ is a finite nonempty alphabet, a set of base objects, the objects can have default properties,
- $\mathcal{A} = \{A_1, \ldots, A_m\}$ is a finite nonempty set of agents,
- $G = (V, E)$ is a graph, V is a finite set of nodes, $|V| = m$, E is a finite set of edges,
- $\mu\colon V \to \mathcal{A}$ is a location function, locating agents into the graph nodes,
- $W_0 \subseteq \Sigma$, $W_0 \neq \emptyset$ is the initial state of the shared environment,
- P_0 is the initial set of programs located in the shared environment.

If the graph G is undirected, then it is connected. If G is directed, then it is weakly connected.

An agent A_j, $1 \leq j \leq m$, is a pair $A_j = (w_j, P_j)$ where

- $w_j \subset \Sigma$, $w_j \neq \emptyset$ is the initial state of the agent,
- P_j is the initial set of programs of the given agent, P_j is finite, and can be empty.

In the following paragraphs, we specify the individual parts of this basic definition, we take into account an IR Colony $\Pi_{IR} = (\Sigma, \mathcal{A}, G, \mu, W_0, P_0)$ and $G = (V, E)$.

Rules and Programs. The rules in Π_{IR} consist of symbols, these symbols may or may not have properties specified. Listing a property on the left side of a rule indicates a conditional application of that rule. The properties are numbers from $\mathbb{Z}$. The rules can be in one of the following forms:

- evolution rule: $a \to b$ or $a(x) \to b(y)$, $a, b \in \Sigma$, an agent evolves its state, $x, y \in \mathbb{Z}$ are properties of the given objects,
- deletion rule: $a \to \varepsilon$, $a \in \Sigma$,
- multicast rule: $a \xrightarrow{\text{out}} b$, $a, b \in \Sigma$ to send the object b to all outgoing edges, a remains inside the sending agent; if $a = b$, it is not necessary to specify the properties of the objects, the current property of a is used,
- backup rule: $a \to$, $a \in \Sigma$, to put the object down into the environment (including its current property), the object a remains in the agent's state,
- restoration rule: $\leftarrow b$, $b \in \Sigma$, to pick an object up from the environment (including its current property), the object b remains in the environment; if b has been present in the state of the given agent, the parameter will be rewritten (updated),
- programming rule: $p \triangleright$

The backup and restoration rule types apply the unification operation, including the processing of parameters. The programming rule will be explained later.

Denote $\mathcal{U}$ the set of all possible rules for Π_{IR}. A program in Π_{IR} is a construct $p_{\text{lab}} = (\text{lab}, U, R, I)$ where

- each program has the own unique label lab,
- $U \subseteq \mathcal{U}$ is a finite nonempty set of rules,
- $R \subseteq \Sigma \cup \mathcal{U}$ is a finite set of reactants, reactants can be:
  - an object with or without a property (a relational expression can be added to the object for its property),
  - a relation between properties of different objects,
  - a program,
- $I \subseteq \Sigma \cup \mathcal{U}$ is a finite set of inhibitors, $R \cap I = \emptyset$, the syntax of elements of this set is the same as for R.

The set U must be deterministic in the sense that the same object must not appear on the left-hand side of any two rules.

A rule $r \in U$ with an object $a \in \Sigma$ on the left side *is applicable* to a state W of a given agent iff the object a is present in W including potential parameters. The restoration rule $\leftarrow b$ is applicable iff the object on the

right side is present in the environment. The programming rule is always applicable.

A program p_{lab} in an agent $A_j = (W_j, P_j)$ *is applicable* iff

- all rules in U are applicable to W_j,
- $(R \cap \Sigma) \subseteq W_j$, $(R \cap \mathcal{U}) \subseteq P_j$,
- $I \cap W_j = \emptyset$, $I \cap P_j = \emptyset$.

It should be noted that only agents can run programs, it is not possible to run any program directly in the environment.

Process and States. The agents in $\mathcal{A}$, $|\mathcal{A}| = m$, work synchronously in the weakly parallel mode, in subsequent steps. In each step, every agent A_j, $1 \leq j \leq m$, non-deterministically chooses one of its applicable programs and executes this program on its state.

An n-step process in Π_{IR} is a tuple $(\pi_0, \pi_1, \ldots, \pi_m)$ where for all agents A_j, $1 \leq j \leq m$, is $\pi_j = (\gamma_j, \delta_j)$:

- $\gamma_j = C_{j,0}, C_{j,1}, \ldots C_{j,n}$ (the context sequence),
- $\delta_j = D_{j,0}, D_{j,1}, \ldots D_{j,n}$ (the result sequence).

$D_{j,i}$ is a *result of an agent* A_j (towards its neighbourhood, applying the multicast rules) for the i^{th} step, $i \geq 0$, and a program $p \in P_j$ is used by A_j in the given step:
$D_{j,i} = \left\{ b(y) \;\middle|\; \left(a(x) \xrightarrow{\text{out}} b(y) \right) \in p \right\}$.

If the property of object b is not specified in the rule, the property x assigned when applying the rule (the current property of object a) is used.

$C_{j,i}$ is a context set of an agent $A_j = (W_j, P_j)$ for the i^{th} step:
$C_{j,i} = \left(\bigcup_{1 \leq k \leq n} D_{k,i-1} \;\middle|\; v_k \in \text{in}(v_j) \right)$, $\mu(v_s) = A_s$ for all $1 \leq s \leq m$.

Denote $\text{map}_{A_j}(W, p)$ the *mapping function of an agent* A_j (towards its state) for a program p and a set of objects W: the function captures the use of all rules affecting the agent's state contained in the program p, just except the multicast-type ones on the set W which do not affect the state of the agent A_j.

Denote $\text{map}_0(W, P)$ the *mapping function of an environment* for some subset of agents' programs P affecting the environment (the backup and restoration rules).

The sequence of states of the shared environment appropriate to the given process is $W_{0,0}, W_{0,1}, \ldots, W_{0,n}$:

- $W_{0,0} = W_0 \cup C_{0,0}$,
- $W_{0,i} = \text{map}_0(W_{i-1}, P)$, P is a set of all programs being used by agents in the given step affecting the environment.

The sequence of states of an agent A_j appropriate to the given process is $W_{j,0}, W_{j,1}, \ldots, W_{j,n}$:

- $W_{j,0} = W_j \cup C_{j,0}$,
- $W_{j,i} = \text{map}_{A_j}(W_{j,i-1} \cup C_{j,i}, p)$, $1 \leq i \leq n$, p is a program applied in the i^{th} step.

The given process can be shortly represented by the sequence $(W_{0,0}, \ldots, W_{m,0}) \Rightarrow^* (W_{0,n}, \ldots, W_{m,n})$.

Transferring Programs. Unlike the original P Colonies with transferable programs, here we set the automatic transfer of programs when the given conditions are met.

As stated above, a program is defined by a label, a set of rules, a set of reactants, and a set of inhibitors. All agents have their own initial set of programs, and the environment carries the base repository of programs with the initial state P_0.

A programming rule is a construct $(\text{label}, U, R, I) \triangleright$ with the parts of this sequence of the same meaning as in the definition of a program. This rule creates a new program with the given parameters and stores it in the rule repository in the environment.

Before starting each step in an IR colony, agents check the program repository for a new program with a label belonging to one of their programs. If so, the agent's original program is overwritten (updated) by a new program with the same label located in the repository.

The flow of a process step. During a single step of a process, the following happens (for a j^{th} agent):

1. the agent checks the program repository and updates its own programs,
2. the agent nondeterministically chooses one of its applicable programs,
3. all rules in the program are processed with influencing the own state and computation of the result and context sets for the given agent and step,
4. the result sets are used to calculate the new state of agents.

5. Sample Model of IoT Network

The system proposed in the previous section is used here to outline a model of communication in a network of IoT devices. The sample network consists of 7 devices:

- A_1: control panel with buttons and other controls for manual handling of several following devices,
- A_2: display (an LCD panel) to show some sensor values (thermometer, CO_2 sensor),
- A_3: updater that provides updates for all devices on the network, keeps an inventory of software versions on different devices, and forwards updates of programs to the environment,
- A_4: smart light bulb,

A_5: window control device that can open or close a window based on data from other IoT devices (here thermometer and CO_2 sensor),

A_6: thermometer, its values can affect the window control,

A_7: CO_2 sensor, its values can also affect the window control.

Figure 1 shows the demonstrated network with the listed devices and connections (directed edges).

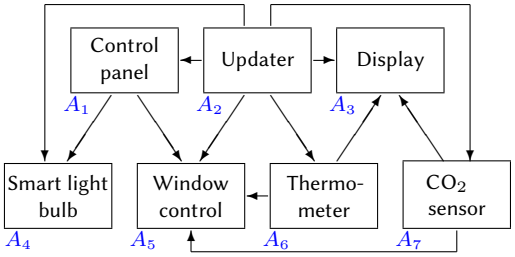

Figure 1: Sample IoT network

Each agent can have a different number of objects in its state, and this number can also be changed.

The agent A_6 is a thermometer. It is a very simple device: it needs only two objects inside the state. The object tv means the version, the object tn is intended to store the current temperature. There is only one program in the set of programs, with one (multicast) rule.

$A_6 = (\{tv(1), tn(24)\}, \{\text{tout}\})$ with the program:

$(\text{tout}, \{tn \xrightarrow{\text{out}} tn\}, \emptyset, \emptyset)$

The agent exports the object tn with its current property to all edges directed from the agent (so the object appears in the states of the agents A_5 and A_3 for the next step, their contexts).

The agent A_7 is similar: it has only two objects inside the state and one program with the multicast rule:

$A_7 = (\{sv(1), sn(1000)\}, \{\text{sout}\})$ with the program:

$(\text{sout}, \{sn \xrightarrow{\text{out}} sn\}, \emptyset, \emptyset)$

The agent A_5 is a bit more complicated. Its role is a window control, and it is necessary to synchronize inputs from three different sources: the thermometer, the CO_2 sensor, and the manual operation on the control panel. The agent has three programs: opening a window in response to high room temperature or high CO_2 levels, and closing the window. These programs are not triggered when the manual mode (operation from the control panel) is enabled.

Besides the object for the version, we have an object for the state (0=closed, 1=open) and an object for the manual mode indication.

$A_5 = (\{wv(1), ws(0), wm(0)\},$
$\{\text{wopent}, \text{wopens}, \text{wclose}\})$

$(\text{wopent}, \{ws(0) \to ws(1)\},$
$\{tn(> 22)\}, \{wm(1)\})$

$(\text{wopens}, \{ws(0) \to ws(1)\}, \{ts(> 1200)\},$
$\{wm(1)\})$

$(\text{wclose}, \{ws(1) \to ws(0)\}, \{tn(\leq 22), ts(\leq 1200)\},$
$\{wm(1)\})$

We can notice that there is a conjunction relation between the elements in the set of reactants, so the opening operation is divided into two programs.

The agent A_3 only displays information obtained from its own state. The initial state consists of one object $dv(1)$, it is the version. The remaining objects will be delivered during the system operation. No programs are needed, the change of the display view state is done automatically if there is a change in the objects and their parameters from other agents.

$A_3 = (\{cv(1)\}, \emptyset)$

The agent A_4 is a light bulb. There are three objects in its state: the version, the device state (0 for lights off, 1 for lights on), and the light intensity (0–12).

$A_4 = (\{bv(1), bs(0), bi(2)\}, \emptyset)$

The agent A_1 plays the role of the control panel for the manual handling of devices. It needs the objects with the current state of the controlled devices (the light bulb, the window), and one additional object indicating manual handling for the window.

$A_1 = (\{cv(1), cb(0), cwm(0), cws(0)\},$
$\{\text{clight}, \text{cwindowman}, \text{cwindowstate}\})$

$(\text{clight}, \quad \{cb \xrightarrow{\text{out}} cb\}, \quad \emptyset, \quad \emptyset)$

$(\text{cwindowman}, \{cwm \xrightarrow{\text{out}} wm\}, \emptyset, \quad \emptyset)$

$(\text{cwindowstate}, \{cws \xrightarrow{\text{out}} ws\}, \quad \{cwm(1)\}, \emptyset)$

The last agent, A_2, is the updater. It holds the database of all programs and their versions (as object properties). Because of lack of space, we will show here only the creation of an update program (more precisely: two programs) for the agent A_5, when it is necessary to change the temperature at which the window will automatically open. We use one special helper object wvn indicating a new value intended for the object wv.

$A_2 = (\{cv(1), uv(1), wv(1), \ldots, wvn(2)\},$
$\{\text{uw2}, \text{uw2d}, \ldots\})$

The program uw2 creates and exports the new version of the program wopent, multicasts the object wv with the property 2, and evolves its own copy of the object wv by changing the property. The set of rules must be deterministic, therefore the multicast rule has a different symbol on each side.

$(\text{uw2}, \{ (\text{wopent}, \{ws(0) \to ws(1)\},$
$\{tn(> 24)\}, \{wm(1)\}) \triangleright,$
$wvn(2) \xrightarrow{\text{out}} wv(2),$
$wv(1) \to wv(2)\},$
$\{wvn(2)\}, \emptyset)$

The set of reactants has the member $wvn(2)$, so the given rule is applied immediately after the new helper

object wsn is created. After the update is provided, the helper object is removed using a deletion rule:
(uw2d, $\{wvn(2) \to \varepsilon\}$, $\{wvn(2), wv(2)\}, \emptyset$)

While most devices (and agents) will operate more or less automatically based on hardware signals affecting the properties of objects in the agent state and according to contained rules, for the updater we assume external intervention resulting in the dynamic creation of new rules for program updates.

6. Discussion and Conclusion

The proposed system is optimized for a very specific application – IoT network modeling.

Unlike other systems, here we count on the existence of object properties directly in the definition because the purpose of most IoT devices is just to send or receive (mostly numeric) data, or to react to them in a short, simple code (these devices are not computationally demanding, they are often powered by a battery, which limits their performance considerably).

Compared to P Colonies, we also abandoned the static number of objects in the agent environment (state) and the number of rules in programs. While this feature makes it easier to detect and compare the computational power of systems, it complicates the modeling of a group of heterogeneous entities. In our system, it is even possible to continuously add objects to (or remove from) the agent's state that were not originally there.

The agent A_5 does not have any tn and sn objects in its state at the beginning of the system operation, it gets them only after the agents running the thermometer (A_6) and CO_2 sensor (A_7) become functional and use their multicast rules.

The format of rules and programs is a hybrid between P Colonies with transferable programs and networks of R Systems. All programs are in principle transferable, this property (unlike P Colonies) is not determined directly by a condition, but by a match in the label.

Conditionality refers more to the execution of rules (similar to R Systems) and is even represented in two places in the system design:

- sets of reactants and inhibitors in programs,
- properties of objects on the left-hand sides of rules.

Cooperation is very simple between directly connected agents, agents can send objects with properties to each other at each step. A multicast rule is used for this purpose, which corresponds to one-to-many communication in computer networks. We also considered a rule for one-to-one communication, but this would mean adding a destination agent label to the rule definition, which can be a problem in a dynamically changing graph node structure.

Another option for communication between agents is the use of a shared environment. If one agent uses a backup rule and another agent uses a restoration rule, both with the same object, we get the equivalent of broadcast communication (one-to-all). This method of communication was not shown in the example in the previous section, but the principle is not complicated.

IR Colonies assume not only the processes mentioned in the definition but also potential external interventions (e.g. a temperature sensor as hardware directly influences the property containing the instantaneous temperature, this value is not influenced by any rule). In the previous section with the IR Colony example, two agents are influenced in this way: A_6 (Thermometer) and A_7 (CO_2 sensor).

The activities taking place in the IR Colony can therefore be divided into two groups:

- explicit (provided by rules in programs),
- implicit (provided by other means).

Implicit operations are included because the system is intended to model real systems dealing with heterogeneous data, including dynamic systems with variable structure and purpose.

Some aspects of the system could, of course, be designed differently. For example, it is not possible to create and distribute completely new rules with a new label. In some circumstances this functionality would be useful, if needed it is not a problem to add it to the system.

Further research can be focused in several directions:

- improving the definition of IR Colonies to better match expected uses,
- detailed comparison of the properties, capabilities, and relationships with P Colonies (with transferable programs), R Systems, and possibly other similar systems,
- evaluation of possibilities and limits of the use of IR Colonies,
- creation of supporting tools that will allow the system to be used for modeling in digital form.

References

[1] G. Păun, Membrane Computing: An Introduction, Springer, Heidelberg, 2002.

[2] G. Păun, G. Rozenberg, A Guide to Membrane Computing, Theoretical Computer Science 287 (2002) 73–100. doi:https://doi.org/10.1016/S0304-3975(02)00136-6.

[3] G. Păun, G. Rozenberg, A. Salomaa, The Oxford Handbook of Membrane Computing, Oxford University Press, New York, 2010.

[4] J. Kelemen, A. Kelemenová, G. Păun, P Colonies: A Biochemically Inspired Computing Model, in: Workshop and Tutorial Proceedings. Ninth International Conference on the Simulation and Synthesis of Living Systems (Alife IX), Boston, Massachusetts, 2004, pp. 82–86.

[5] L. Ciencialová, L. Cienciala, Transferable Knowledge in P Colonies, in: Information Technologies – Applications and Theory 2022 (ITAT 2022), volume 3226, Zuberec, Slovakia, 2022, pp. 167–174.

[6] Š. Vavrečková, Notes on Relationship of P Colonies to Osmotic Computing and Computer Viruses, in: CEUR Proceedings of the 23rd Conference Information Technologies – Applications and Theory (ITAT 2023), 2023, pp. 147–153.

[7] A. Ehrenfeucht, G. Rozenberg, Reaction Systems, Fundam. Informaticae 75 (2007) 263–280. URL: http://content.iospress.com/articles/fundamenta-informaticae/fi75-1-4-15.

[8] P. Bottoni, A. Labella, G. Rozenberg, Networks of Reaction Systems, International Journal of Foundations of Computer Science 31 (2020) 53–71. doi:10.1142/S0129054120400043.

[9] B. Aman, From Networks of Reaction Systems to Communicating Reaction Systems and Back, in: J. Durand-Lose, G. Vaszil (Eds.), Machines, Computations, and Universality, Springer International Publishing, Cham, 2022, pp. 42–57. doi:https://doi.org/10.1007/978-3-031-13502-6_3.

[10] L. Ciencialová, L. Cienciala, E. Csuhaj-Varjú, P Colonies and Reaction Systems, volume 2, Springer International Publishing, 2020, pp. 269–280. doi:10.1007/s41965-020-00051-1.

[11] P. Sethy, Notes on P Systems versus R Systems, in: Developments in Computer Science(DCS), 2021, pp. 263–266.

[12] Š. Vavrečková, Membrane System as a Communication Interface between IoT devices, in: CEUR Proceedings of the 22nd Conference Information Technologies – Applications and Theory (ITAT 2022), 2022, pp. 184–190.

[13] J. E. Hopcroft, J. D. Ullman, Introduction to Automata Theory, Languages and Computation, Addison-Wesley Series in Computer Science and Information Processing, Addison-Wesley, 1979.

[14] W. Kassab, K. A. Darabkh, A–Z Survey of Internet of Things: Architectures, Protocols, Applications, Recent Advances, Future Directions and Recommendations, Journal of Network and Computer Applications 163 (2020). URL: https://doi.org/10.1016/j.jnca.2020.102663.

[15] J. Dizdarević, F. Carpio, A. Jukan, X. Masip-Bruin, A Survey of Communication Protocols for Internet of Things and Related Challenges of Fog and Cloud Computing Integration, Association for Computing Machinery 51 (2019). URL: https://doi.org/10.1145/3292674. doi:10.1145/3292674.