

# Compatibility of IR Colonies and Protocols for Communication between IoT devices

Šárka Vavrečková<sup>1</sup>

<sup>1</sup>*Silesian University in Opava, Faculty of Philosophy and Science, Bezručovo nám. 13, Opava, Czech Republic*

## Abstract

IR Colonies represent a theoretical concept derived from membrane and reaction systems, and their purpose is to model communication between Internet of Things (IoT) devices. In this paper, we analyze the possibilities of turning this concept into a real-world application, and determine the possibility of implementing the system as a real communication simulator in an IoT network. IoT devices use a variety of protocols to communicate with each other, which can be classified into one of two communication models. In this paper, we analyze the specifics of these two communication models and consequently, of the individual protocols, and consider appropriate approaches and possible modifications of the concept into a form compatible with the respective protocols.

## Keywords

IR Colony, program, rule, Internet of Things, IoT protocol, Request-Response, Publish-Subscribe, HTTP REST, MQTT

## 1. Introduction

Membrane computing (introduced by Gheorghe Păun in 1998) is a framework for modeling parallel distributed processing. Mathematical models of membrane systems have been called P Systems. Information about this paradigm is available in [1, 2, 3], or the bibliography at <http://ppage.psystems.eu/> [2025-07-03].

The development continued and a number of systems based on P Systems were created. The P Colony (introduced in [4]) is a simple computational model with an environment containing objects and agents. The agents have their own state consisting of objects and they are equipped with programs. The programs are used to influence both the agent's own environment and the shared environment of the parent P Colony.

P Colonies with transferable programs were introduced in [5]. This type of P Colonies enables the transfer of programs between agents and the environment, and vice versa. Thus, an agent can receive a program with which it was not equipped during the previous computation steps. A transferable program is a special type of program located in the environment or in an agent and equipped with a condition that must be met for the program to be transferred between the agent and the environment.


Reaction systems were introduced in [6], and in [7, 8] we can find information about networks of reaction systems. The inspiration was mainly in the field of biochemistry. Reaction systems (R Systems) allow us to use reactants and inhibitors to influence the processes in the system. Reactants can initiate a process, while inhibitors can block it. Networks of reaction systems are graphs with reaction systems as their nodes, they allow reaction systems to communicate with each other and influence neighbours' actions.

IR Colonies were introduced in [9] as a tool designed especially for modeling communication in IoT networks. It is a system based on both the aforementioned P Colonies with transferable programs and reaction systems. IR Colonies take from P Colonies the idea of shared environment and agents equipped with programs, reaction systems were the inspiration for the definition of infrastructure as a graph and the use of the principle of reactants and inhibitors in program rules. In [9] we can also find a sample model of IoT network created using an IR Colony.

---

*ITAT'25: Information technologies – Applications and Theory, September 26–30, 2025, Telgárt, Slovakia*

✉ [sarka.vavreckova@fpf.slu.cz](mailto:sarka.vavreckova@fpf.slu.cz) (Š Vavrečková)

ORCID  0009-0003-7121-4743 (Š Vavrečková)



© 2025 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

The purpose of this paper is to examine the compatibility and identify any differences between IR Colonies and common IoT application protocols. The goal is to prepare for the use of IR Colonies for the purpose of simulating communication in IoT device networks and to transform the IR Colony structure into real-world IoT (Internet of Things) device networks.

This section is followed by the preliminaries section with a brief introduction to IR Colonies, IoT protocols and basic IoT communication models.

Section 3 consists of three subsections: in the first one, we discuss different aspects of the functionality of IR Colonies and identify the possibilities of their implementation. In the second and third subsections, we focus on the possibilities of implementing communication between agents with each other and between agents and the environment, analyzing the correlation of this communication to the two basic communication models for IoT networks (Request-Response and Publish-Subscribe).

Section 4 is devoted to individual application protocols used in the world of IoT devices, focusing in particular on the two most common protocols. We consider their communication model (typically one of the two previously discussed), but also other features that need to be considered when implementing the IR Colony concept.

## 2. Preliminaries

We assume the reader to be familiar with the basics of the formal language theory [10] and membrane computing [3].

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

For details and definitions of 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. IR Colonies

An IR Colony is a system of agents in a shared environment, connections between agents are defined using a graph. Each agent has its own state in the form of a set of objects, and objects can also occur in the shared environment.

In order to make our IR Colony concept applicable to IoT network simulation, we need objects with *properties* (values). The properties of objects are numbers (e.g. for an object of type temperature, the property is the temperature value, an object specifying the version of a device will have the version number directly as a property). In the basic concept of IR Colony, each object has at most one property and the value of this property is a number, but in principle it is not a problem to modify this concept and use objects with more than one property, moreover of different types (numbers, strings).

The agent state is determined by the set of objects. Because our objects can have properties, set operations on these sets must be defined a little differently than usual.

**Definition 1 (According to [9]).** The intersection operation on sets of objects with properties corresponds to the operation without using properties, except when both sets contain the same object with different property values. Suppose an object  $a$  with a property  $x \in \mathbb{R}$  (written as  $a(x)$ ) and a set  $M$ . There are the following cases:

- $a \in M: \{a\} \cap M = \{a\},$
- $a(x) \in M: \{a(x)\} \cap M = \{a(x)\},$
- $a(y) \in M, y \in \mathbb{R}, y \neq x: \{a(x)\} \cap M = \emptyset,$
- $a \in M, \text{ with no property: } \{a(x)\} \cap M = \{a(x)\}.$

The intersection operation is commutative.

All membership operators (e.g.  $\subseteq$ ) work with properties in the same way.

The union 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. There are the following cases:

- $a \in M$  or  $a(x) \in M$ :  $a(x) \in \{a(x)\} \cup M$  and  $a(x) \in M \cup \{a(x)\}$ ,
- $a(y) \in M$ ,  $y \neq x$ :  $a(x) \in M \cup \{a(x)\}$ , but  $a(x) \notin \{a(x)\} \cup M$  and  $a(y) \in \{a(x)\} \cup M$ .

For the union operation, the second case listed: This corresponds to updating properties in the right-to-left direction, e.g.  $\{a(x), b(y)\} \cup \{b(z), c(r)\} = \{a(x), b(z), c(r)\}$ .

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.  $\{a(x)\} \cup \{a, b\} = \{a(x), b\}$ .

If objects could have more than one property, the definition would need to be slightly modified. We would assume that two objects of the same type are different if they differ in at least one property.

**Definition 2 (IR Colony, [9]).** An IR Colony (IoT Reaction Colony) is a construct

$\Pi_{IR} = (\Sigma, \mathcal{A}, G, \mu, W_0, P_0)$  where

- $\Sigma$  is a finite nonempty alphabet, a set of base objects, the objects can have default properties,
- $\mathcal{A} = \{A_1, \dots, A_m\}$  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,
- $\mu: V \rightarrow \mathcal{A}$  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

- $w_j \subseteq \Sigma$ ,  $w_j \neq \emptyset$  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.

The symbols (objects) in rules 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  $\mathbb{R}$ . The rules can be in one of the following forms:

- evolution rule:  $a \rightarrow b$  or  $a(x) \rightarrow b(y)$ ,  $a, b \in \Sigma$ , an agent evolves its state,  $x, y \in \mathbb{R}$  are properties of the given objects,
- deletion rule:  $a \rightarrow \varepsilon$ ,  $a \in \Sigma$ ,
- 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:  $a \rightarrow$ ,  $a \in \Sigma$ , 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 format is explained below at the end of the definition.

The backup and restoration rule types apply the union operation, including the processing of parameters. Denote  $\mathcal{U}$  the set of all possible rules for  $\Pi_{IR}$ . A program in  $\Pi_{IR}$  is a construct  $p_{\text{lab}} = (\text{lab}, U, R, I)$  where

- each program has the own unique label  $\text{lab}$ ,
- $U \subseteq \mathcal{U}$  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 the 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 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. The programming rule creates a new program with the given parameters and stores it in the rule repository in the environment.

**Definition 3 (Application of rules and programs, [9]).** Assume an IR Colony

$\Pi_{IR} = (\Sigma, \mathcal{A}, G, \mu, W_0, P_0)$  and  $G = (V, E)$  with the set of agents  $\mathcal{A} = \{A_1, \dots, A_m\}$ .

A state of a given agent  $A_j$ ,  $1 \leq j \leq m$  is a set of objects, the objects have or have not properties.

A rule  $r \in U$  with an object  $a \in \Sigma$  on the left side is applicable to a state  $W_j$  of a given agent  $A_j$ , iff the object  $a$  is present in  $W_j$  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.

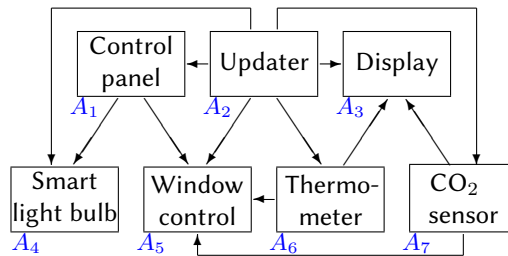
A program  $p_{\text{lab}}$  in an agent  $A_j = (W_j, P_j)$  is applicable iff

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

During a single step, the following happens (for a  $j^{\text{th}}$  agent):

1. the agent checks the program repository and updates its own programs: if there is a program with a label belonging to one of the agent's programs, the agent's original program is overwritten by a new program with the same label located in the repository,
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.

**Example 1.** The paper [9] also defines the terms process, context sequence, result sequence, and others, but we won't need them in this paper. There is also an example of an IoT device network modelled with IR Colony, including a set of agents with programs: Figure 1 shows a diagram of the presented system.



**Figure 1:** Sample IoT network[9]

For example, the agent  $A_5$  (Window control) is defined as follows:

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

Thus, its initial state contains three symbols, the first of which is the version of its software (to see if it an update is necessary), the next one indicates the current state of the window opening (value

0 means “closed”), the third one is to indicate the manual mode of window control (value 0 means “manual mode is off”, i.e. the window is controlled by this component. There are three programs in the set of programs: the first one for opening the window if the temperature in the room is too high (info from the thermometer), the second one after the information about high CO<sub>2</sub> level, the third one for closing the window. The first program:

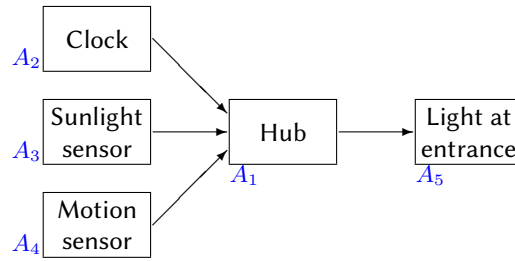
(wopent,  $\{ws(0) \rightarrow ws(1)\}$ ,  $\{tn(> 22)\}$ ,  $\{wm(1)\}$ )

The set of rules contains a single rule that changes the state of the window to “open”, the set of reactants contains a single reactant specifying that the rule is to be executed if an object  $tn$  with a property value greater than 22 is received from the thermometer (the object then remains in the agent state and is overwritten by any other incoming objects of the same type). The inhibitor of the rule is a manual window control object in the state “manual mode is on”.

In the paper [9] the given example is complete.

Other possible reactants and inhibitors could be related to outdoor temperature, and weather in general, but for this we would need additional agents (sensors).

**Example 2.** The following diagram shows a simple centralized network of several IoT devices.



**Figure 2:** Sample centralized IoT network

The goal is to control the light before entering the house so that the key and keyhole can be easily found. The light will turn on if it is night, or if it is day but dark for some reason (e.g. it is cloudy), and we only want to turn on the light if someone appears at the door.

The IR Colony:

$\Pi_{IR} = (\Sigma, \mathcal{A}, G, \mu, W_0, P_0)$  where  $\mathcal{A} = \{A_1, \dots, A_5\}$ ,  $G = (V, E)$  is a graph corresponding to the set  $\mathcal{A}$  and the structure shown in Figure 2 using the location function  $\mu$ .

There are currently no objects in the shared environment (i.e.,  $W_0 = \emptyset$ ); it is a simple closed system. The set  $P_0$  is also empty; the program repository would be used e.g. if we added an update mechanism or if the hub were user-programmable and allowed changes to the functionality of the connected devices.

The agent  $A_1$  is defined as follows:

$A_1 = (\{\text{day}(1), \text{sun}(1), \text{move}(0), \text{lightState}(0)\}, \{\text{lightOn1}, \text{lightOn2}, \text{lightOff}\})$

with three programs:

(lightOn1,  $\{\text{lightState}(0) \rightarrow \text{lightState}(1), \text{lightState} \xrightarrow{\text{out}} \text{lightState}\}$ ,  
 $\{\text{day}(0), \text{move}(1)\}, \{\text{lightState}(1)\})$

(lightOn2,  $\{\text{lightState}(0) \rightarrow \text{lightState}(1), \text{lightState} \xrightarrow{\text{out}} \text{lightState}\}$ ,  
 $\{\text{day}(1), \text{sun}(0), \text{move}(1)\}, \{\text{lightState}(1)\})$

The set of inhibitors could be empty if we placed  $\text{lightState}(0)$  in the set of reactants.

(lightOff,  $\{\text{lightState}(1) \rightarrow \text{lightState}(0), \text{lightState} \xrightarrow{\text{out}} \text{lightState}\}$ ,  
 $\{\text{day}(1), \text{sun}(1), \text{move}(0)\}, \{\text{lightState}(0)\})$

The multicast rules contain objects without property, they only transmit the current values.

This agent acts as a hub and is the central element of the entire network. It can be equipped with a display showing the current status of connected devices, i.e., the properties of objects in the agent’s state.

The next three agents are simple:

$A_2 = (\{\text{day}(1)\}, \{\text{transportDay}\})$  with one program:  $(\text{transportDay}, \{\text{day} \xrightarrow{\text{out}} \text{day}\}, \{\text{day}\}, \emptyset)$

$A_3 = (\{\text{sun}(1)\}, \{\text{transportSun}\})$  with one program:  $(\text{transportSun}, \{\text{sun} \xrightarrow{\text{out}} \text{sun}\}, \{\text{sun}\}, \emptyset)$

$A_4 = (\{\text{move}(0)\}, \{\text{transportMove}\})$  with one program:

$(\text{transportMove}, \text{move} \xrightarrow{\text{out}} \text{move}, \{\text{move}\}, \emptyset)$

The agents  $A_2, A_3, A_4$  operate asynchronously at regular intervals because the status of the sensors does not change very often and it is necessary to preserve the battery life of these devices. The light should also remain switched on for a certain period of time even when no movement is detected.

The properties of the enclosed objects day, sun and move changes independently of the IR colony according to the current status of the (physical) sensors. The role of the agents is to be transmitting the current values to the agent  $A_1$ .

The last agent is very simple, it has no programs:

$A_5 = (\{\text{lightState}(0)\}, \emptyset)$

The enclosed object is being updated by the hub in regular intervals, the current value is passed to the device.

## 2.2. IoT Protocols

IoT (Internet of Things) devices are usually small devices with simple functionality, where the main importance is their interconnection in the network and automation of their communication with each other. Some IoT devices are equipped with sensors and send data from these sensors to the network (e.g. a thermometer or more complex weather station, smoke detector, light detector, etc.). Other devices can be used to interact with a user (screen, controller), or act as a central controller (broker) or router.

Networks of IoT devices are a typical example of synergy: very simple devices can, when properly connected, form a relatively smart system. For example, a change in the state of a thermometer can have the effect of switching the heating on or off and opening the windows to ensure the room is at the right temperature. Smart bulbs can also have the intensity and colour of light adapted to the phase of day and the amount of sunlight according to the current weather.

In [11] several definitions of IoT network can be found, a more compact definition is provided in [12]. We can compose the following definition from them:

**Definition 4 ([11]).** *The Internet of Things (IoT) is a network of various types of smart objects (so called things) and devices. The things are connected to the Internet and communicate with each other with minimum human interface. They are embedded with abilities as sensing, analyzing, processing and self-management based on interoperable communication protocols and specific criteria. These smart things should have unique identities and personalities.*

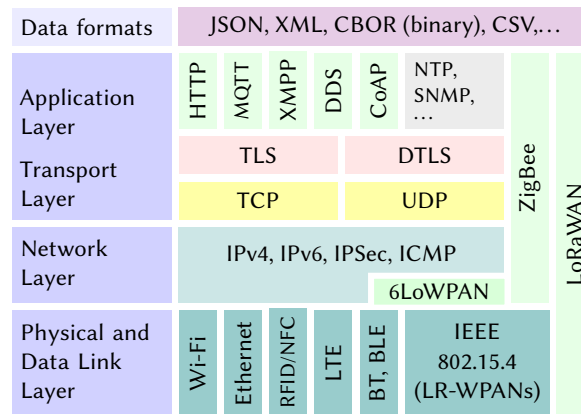
Most IoT devices are battery powered, so their operation must not be energy intensive, including communication. Some devices use Wi-Fi, but there are other options. Figure 3 outlines the communication architecture typical of IoT devices.

The bottom layer contains transmission technologies that determine how data is transformed into a signal, how the signal is handled (frequency, bandwidth, etc.), how the connection is established, etc. As we can see, some IoT devices handle Wi-Fi, Ethernet or LTE just like laptops or cell phones, but there are other possibilities.

RFID/NFC is designed for communication over very short distances (about 10 cm) between a pair of devices. Bluetooth and its optimized form BLE (Bluetooth Low Energy) are used to connect two devices over longer distances (tens of meters), especially the BLE variant is much more energy-efficient than Wi-Fi.

IEEE 802.15.4 (Low-Rate Wireless Personal Area Networks) is a communication standard optimized for wireless sensor networks, and is the basis for various higher-layer technologies designed especially for M2M (Machine-to-Machine) communication. This can be seen in the figure: the 6LoWPAN and





**Figure 3:** The IoT protocol stack, selected protocols (based on [13, 14, 15, 16, 17])

ZigBee technologies are typical examples. Besides other features, 6LoWPAN (IPv6 over Low-power Wireless Personal Area Networks) adds a very compact header (metadata) format to avoid having to transfer too much extra data.[11, 18]

IPv4, IPv6, ICMP, IPSec, TCP, UDP, NTP, and SNMP are common network protocols that provide addressing, connectivity, higher-layer protocol identification, time synchronization, network management, and other functions; their description is beyond the scope of this paper. The TLS and DTLS protocols are used to encrypt connections. Details can be found e.g. on [http://tcpipguide.com/free/t\\_NetworkingFundamentals.htm](http://tcpipguide.com/free/t_NetworkingFundamentals.htm) [2025-07-09].

ZigBee [16] is a very popular complex protocol for small IoT networks (e.g. for home automation) used mainly by vendors offering multiple types of IoT devices working together. LoRaWAN [15], on the other hand, is a protocol connecting even very remote IoT devices (e.g. sensors in fields, vineyards, halls). Both ZigBee and LoRaWAN have components for the application layer, but because they are very specific and there are usually options to connect to a gateway handling another protocol (e.g. MQTT), we will not deal with them as application layer solutions here.

Nowadays, Thread and Matter protocols are very popular in the IoT world. The Thread protocol is based on (heaps) some of the protocols mentioned above: it works on top of IEEE 802.15.4 and includes IPv6, 6LoWPAN, UDP, DTL and CoAP, among others.<sup>1</sup> Therefore, we do not need to deal with the Thread protocol separately.

Matter is one of the application layer protocols. Matter is often combined with the Thread protocol in devices, although it can also work with other protocols such as Wi-Fi or BLE. This protocol assumes a centralized network and has a relatively complex communication architecture. The interaction between devices is called a transaction and can be of type Read, Write, Subscribe, Report or Invoke.<sup>2</sup>

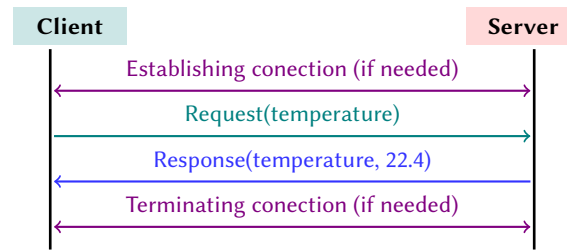
### 2.3. IoT Communication Models

There are several basic *communication models* [11] (communication architectures) used in the IoT world. All of them are based on the essential client-server concept (the server provides data or services, the client uses them, the communication typically starts with the client's request).

**Request-Response communication model.** This communication model is based on how common computer networks work: the client sends a request for data, the server processes it and sends a response with the data.

<sup>1</sup>Details can be found at <https://www.silabs.com/documents/public/user-guides/ug103-11-fundamentals-thread.pdf> [2025-08-18].

<sup>2</sup>Details can be found at [https://csa-iot.org/wp-content/uploads/2024/11/24-27349-006\\_Matter-1.4-Core-Specification.pdf](https://csa-iot.org/wp-content/uploads/2024/11/24-27349-006_Matter-1.4-Core-Specification.pdf) [2025-08-18], especially Chapter 8.

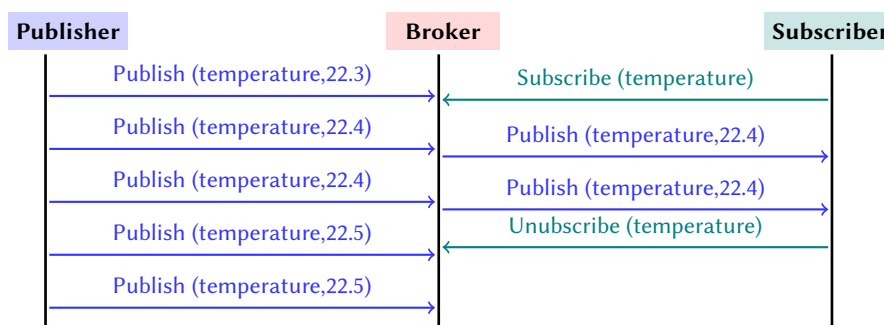


**Figure 4:** Example of communication in the Request-Response communication model

This model does not require the existence of a special device as an intermediary. A device on the network can play both client and server roles as needed in relation to other devices inside the network, but the roles within a single communication are strictly defined.

Some protocols using this model require establishing a connection at the beginning (negotiating connection parameters, etc.) and terminating the connection at the end of the communication, while others simply allow a request to be sent and a response to be returned without establishing a connection.

**Publish-Subscribe communication model.** When using this communication model, there is usually a central device (broker, dispatcher, controller) in the network that mediates the communication and manages the queues with the forwarded data.



**Figure 5:** Example of communication in the Publish-Subscribe communication model

Publishers are nodes in the network serving as data sources, subscribers are nodes serving as data destinations.

The data always belongs to a category (called a *topic*), which is usually related to the type and origin of the data. E.g. there may be a topic for the temperature from a thermometer outside on the north side of the house.

Publishers send the pairs (topic,data) to the broker (whether once or on a regular basis). Any device can subscribe to a topic from the broker, and the broker then forwards everything it receives on that topic to that device.

The broker therefore performs several roles: it maintains queues for individual topics containing published data, receives publish messages, processes subscription orders for topics (subscribe messages) and sends published data according to these subscriptions.

A simplified version of this model (the *Publish* model, [17]) does not require the existence of a broker. The role of the broker is then partially taken over by publishers themselves, subscribers order subscriptions directly from publishers.



### 3. Requirements of IR Colonies for Implementation and Cooperation with Protocols

Our goal is to determine the relationship between the concept of IR Colonies and the protocols used in the IoT world and to analyze the possibilities of implementing IR Colonies in IoT device networks. In this section, we identify areas that need to be considered for this purpose.

#### 3.1. Implementation of Various Aspects of IR Colonies with Respect to IoT Protocols

There are two levels in the use of IR Colonies: the first level is the use of IR Colonies in simulating a network of IoT devices, for example, to plan such a network and to verify the connectivity and communication possibilities between devices. This plan is almost implementation independent of the actual hardware, we just need to take into account certain characteristics and possible limitations of the actual devices we want to simulate. We can develop this system either in a common programming language (C++, Java, Python, C#, ...), or in a language designed for membrane systems (but IR Colonies are probably too specific for such languages). Or we can use IR Colonies only for the theoretical design of the IoT system; programming is not necessary in this case.

The second level is the transfer of the design created in this way to the real world. Here we have to take into account the specifics of both the IoT devices themselves and the protocols that will be used to ensure communication between them.

The problem of implementation can be divided into several parts:

1. Linking to sensors inside the device, i.e. getting an object (or a property of an existing object) into the agent state (e.g. for a thermometer, pressure sensor, light detector, etc.) or, conversely, transferring an object (or its property) from the agent state to an actuator, a visualizer or other component (e.g. a motor for opening/closing a window, a smart bulb, a display).
2. Working with states containing objects inside the agents, implementing programs/rules including working with reactants and inhibitors.
3. Identification of individual agents, communication between agents within the IR Colony.
4. Communication with the outside world (cloud, Internet services).

We will discuss in detail each of the items listed above.

**Item 1:** Each type of device has a specific way of accessing sensors and various other components; there is no space for the implementation details in this paper.

**Item 2:** The agents themselves can be developed quite easily in any programming language supported by the given IoT devices. And if the chosen programming language allows it, we can use e.g. the object paradigm (we use objects in IR Colonies) and event-driven code. Working with reactants and inhibitors in programs is very easy to program, they are simple conditions. We can use state-driven programming (i.e. the software design pattern “state”) as in the implementation of finite automata, which can be easily combined with the event-driven programming mentioned above.

Also, programming both the evolution and deletion rules that are related only to the agent’s state, and operations on that state in general, will not be a problem. Other rules already require communication with other agents, so their application depends on the communication protocol used (see item 3 below).

**Item 3:** This implementation step depends on the chosen IoT application protocol (or protocols—nothing hinders us from making IR Colony compatible with multiple protocols at the same time). We are not forced to code the protocols themselves (there are ready-made libraries for that), but we need the communication model in IR Colony to match the communication model of the given protocol:

- possible roles of devices, identifying the devices in communication,
- identifying the device state, data format for data exchange,

- establishing and terminating connection and acknowledging (if it is applied at all),
- requesting and sending data, sending unsolicited data,
- subscribing data to be sent periodically, etc.

The implementation of most rules in programs depends on this item because this is the communication between agents or between an agent and the environment.

**Item 4:** For this item, predefined APIs (Application Programming Interfaces) of individual services are usually used, including the IFTTT service<sup>3</sup> communicating with devices from different vendors. We will not address this issue here.

It follows from this list that we will deal in particular with item 3 subsequently. Let us focus on the two communication models mentioned above. For each model we will be interested in the following:

- requirements for the structure of both the IR Colony and the IoT network,
- identification of communicating devices if needed in messages, addressing, other IDs,
- implementation of rules in agents' programs,
- the possibility of implementing an IR Colony environment.

Currently, almost any IoT network can only be accessed by a device that is equipped with an identifier and login credentials, which is handled by the IoT protocol itself (especially when connecting the device to the network) and we don't need to deal with that here. Except when the device ID, address, etc. are part of the messages being sent—in that case, we can use objects with the appropriate data.

### 3.2. Request-Response Model

There is no central device in the Request-Response model and anyone can request data or service from anyone else, which corresponds to the unrestricted graph structure used in IR Colonies. Thus, the structure of an IR Colony can be easily converted to the structure of devices using this communication model and vice versa; we only need to have a device in the network acting as an IR Colony environment, as will be explained later.

The use of the Request-Response communication model assumes that the client is capable of identifying the server before the communication begins. Conversely, the server can determine the client's identification after receiving the request and then use it to send an addressed response. Communication in an IR Colony is between agents that are connected by an edge. Identification needs to be addressed first of all when creating an edge, which in a real network will be handled by the IoT protocols themselves: creating an edge corresponds to establishing a connection between devices, resolving identification, permissions, etc. Something else is addressing in messages in a previously established connection, which is implementation dependent and will be handled according to a specific protocol. The client can be informed of the servers' addresses via objects and their properties.

But the problem is something else: IR Colonies are built mainly on unaddressed multicast communication (the multicast rule for sending data to connected agents) and other unaddressed rules (the backup and restoration rules for transferring data between agent and environment, the programming rule for sending programs to agents via repository in the environment). Most Request-Response protocols, such as HTTP REST, do not support multicast (one-to-many) communication; they use only unicast (one-to-one) type. Thus, we will need to simulate multicast communication using a sequence of unicast connections.

The Request-Response model strictly adheres to the roles of the client and server, with the server being the source of the data or service and the client starting the communication, we relate

- the start of the (multicast) communication to the process of creating a sequence of oriented edges leading from the agent containing the multicast rule to the agents that are supposed to obtain the data in this way,

---

<sup>3</sup><https://ifttt.com/> [2025-07-17]

- the creation of such edge will correspond to the pair of operations establishing connection plus Request (or Bulk Request), with respect to real functions of the specific protocol,
- the Response operation, and
- the removal of the edge to the operation terminating the connection.

The role of the environment will probably be fulfilled by some equipment, because the environment objects and the program repository for the products of program rules must also be stored somewhere. Therefore, the above will also apply to rules for communicating with the environment (backup, restoration and programming rules).

In IR colonies, each agent should periodically check the environment's rule repository for a rule intended for them ( e.g. an update to one of their existing rules). This step can be implemented by requesting the device acting as the environment. The new rule will then be sent in response to this request.

### 3.3. Publish-Subscribe Model

The use of the Publish-Subscribe model assumes a centralized network: there is a central equipment (broker) to which all publishers send data. This data is sorted and stored in queues for individual topics, then subscribers subscribe to a specific topic and then data from the corresponding queue is sent to the given subscriber.

Thus, the graph of the IR Colony must be in the form of a star: the main node of the graph will be the device acting as a broker, while the other devices are connected to it and have the role of publishers and subscribers as needed. If we have an IR colony in the form of a generic graph, transforming it into star form will not be a problem, we just need to add more objects with information about the original connection.

A single device can act in the both roles, but for different topics. If the device is only acting as a publisher, it will be connected to the broker by a directed edge pointing towards the broker. Subscribers will be connected by a directed edge pointing away from the broker. If a device performs both roles, the edge will be traversable in both directions. Alternatively, we can use an undirected graph and in the IR Colony we would use the mechanism of reactants and inhibitors to sort messages. We have to use this mechanism anyway, because every device/agent will certainly not want to get all messages, but only those belonging to specific topics.

Protocols implementing this communication model also use IDs, addresses and login credentials, but we can leave that up to the protocol. There are usually special message types for establishing and terminating connections, and we can also use these to dynamically create and destroy edges in the graph.

Compared to the previous model, there is a mandatory identification of the message type, called topics. Topics are part of different message types (especially publish and subscribe). In IR Colonies we use objects (topics) and their properties (values sent within the topic) for this purpose.

The basic rule for communication between agents is the multicast rule. Since we will require a star graph structure for the IR Colony, the publisher will only send a message to the broker when using this rule. The broker will then send a multicast to all subscribers registered to the topic, according to its subscriber database (which can be object-based, which is more equivalent to an IR Colony). When switching from an IR colony to the real world, some of the functionality of the programs will be transferred from the agent/publisher to the agent/broker.

The role of the IR Colony environment will be played by the broker. This will make it easy to implement rules for data transfer between an agent and the environment (backup and restoration rule). The broker will also maintain the rule repository, so the implementation of the programming rule will consist of passing a new rule to the broker, which will store the rule in the repository. All agents will contact the broker periodically to ask for new applicable rules (updates).

## 4. Analysis of the Relationship between IR Colonies and IoT Application Protocols

From Figure 3 it is clear that there are many IoT protocols in the application layer (although we have selected only the most common protocols). For each of the listed protocols we will specify the possibilities of its cooperation with the IR Colony concept.

### 4.1. HTTP REST

Some IoT devices communicate with their environment using the HTTP protocol, which is commonly known from WWW communication, or rather its modified variant HTTP REST (Representational State Transfer Protocol) standardized by IETF. The advantage is especially the possibility of direct interaction with common web services and servers.

HTTP REST [11, 19] uses the Request-Response communication model with connection establishment. The communication can be secured by TLS. HTTP REST uses the Post, Put, Delete and Get methods, which fully cover the needs of rules in IR Colony, considering the information given about the Request-Response model in the previous section.

In particular, the Put method is very practical for our purposes (for all rules intended for communication between agents, or an agent and the environment), because it is idempotent<sup>4</sup>, i.e. if the agent/client is sending a message using this method repeatedly with the same object and identical property, there is no change (it corresponds to one message), whereas if the Put method is sent sequentially with different objects and/or properties, the object and the property in the destination are updated. Or the object is created if it does not exist in the destination yet.

HTTP REST can, among other things, send data in JSON text format, which is suitable for representation of objects used in IR Colonies.

Suppose an agent containing a temperature sensor executes a program containing the following rule:  
 $temp \xrightarrow{\text{out}} temp$

Among other objects, the agent also has a *temp* object with the property 22.4 in its state. The purpose of the rule is to send the current state of the temperature sensor to all the nodes in the IR Colony graph that are destinations of edges leading from this agent.

Because HTTP REST must always establish a unicast connection, the original multicast message has to be decomposed into a series of unicast messages (each with a different receiver) and each message specifies the destination device. For the first listed example, one of the destinations could be a display device.

The IoT device, which is a real-world representation of this agent, sends the PUT Requests to all devices with an edge leading from it, including the display device (assuming that the connection between the devices has already been established):

```
PUT /agent/display HTTP 2.0
Accept: application/json
Content-type: application/json
...
```

and in the body of the message is the following message in the JSON format:

```
{
  "name": "temp";
  "property": "22.4";
  ...
}
```

This Request is followed by a Reply acknowledging acceptance of the message.

---

<sup>4</sup><https://restfulapi.net/rest-put-vs-post/> [2025-07-17]

## 4.2. MQTT

MQTT (Message Queuing Telemetry Transport, [11, 13]) is another widely used IoT application protocol, for a change standardized by OASIS. It is built on a Publish-Subscribe communication architecture with a central device (broker), with the methods Connect, Disconnect, Publish, Subscribe, Unsubscribe, Close. A (usually long-term) connection is established between the broker and clients, and Publish, Subscribe and Unsubscribe messages are sent within the connection. The communication can be secured by the TLS protocol.

MQTT has several other specific features that are very practical especially for IoT devices. One of them is “Last Will and Testament Message”: a special type of message stored at the broker that is sent to all subscribers of a particular publisher when the network loses connection to that publisher.

The objects that we send between agents within IR Colony will be represented in MQTT by topics. For example, the object *temp* used in the first listed example will be represented by a topic with the same name, and the agent publishing this topic will send a Publish message to the broker with the current temperature value belonging to the topic named *temp*. The broker will forward this message to all nodes that have subscribed the given topic.

Message data are in plain text format, structured formats such as JSON or XML are not supported. However, in our case it doesn’t matter, because in MQTT messages there are fields for topic and value, we don’t actually need fields for the message data itself.

The format of the MQTT message header<sup>5</sup> is quite complicated. Besides determining the message type (such as “PUT” in HTTP REST), there are, among others, several control bits that specify how some of the more advanced MQTT properties are set. In the Publish message, the topic name, property value, and accompanying plain text message data are present.

## 4.3. Other Protocols

CoAP (Constrained Application Protocol, [20]), standardized by the IETF, does not require a connection to be established; messages are simply sent to the destination. As a result, there is no problem with sending multicast messages. CoAP uses a Request-Response architecture, and there are even implementations with a REST architecture (CoAP REST). Most of what was said above about HTTP REST applies to this protocol, except that multicast messages do not need to be serialized and creating an edge does not need to be bound to connection establishment.

AMQP (Advanced Message Queuing Protocol, [17, 20]) is quite similar to MQTT. It is also standardized by OASIS and uses a Publish-Subscribe communication architecture, but the use of a central component (broker) is optional. If a broker exists in the network, it maintains a message queue separately for each subscriber, and subscribers take messages from their queues themselves (unlike MQTT, which forwards messages from publishers directly to subscribers). Thus, AMQP allows several types of connections to be established: client-broker (the same type as MQTT), client-client and broker-broker.

XMPP (eXtensible Message and Presence Protocol, [16, 20]) by IETF was originally designed for instant messaging and its message format is adapted to that (text, XML, but also calls and video calls). It supports several different communication models, including Request-Response and Publish-Subscribe. While the other IoT protocols are primarily designed for M2M (Machine-to-Machine) communication, XMPP is more towards H2H (Human-to-Human) communication.

DDS (Data Distribution Service, [20]) defined by OMG uses the Publish-Subscribe communication model and is decentralized. This means that the individual devices in the network are much more autonomous than in other Publish-Subscribe protocols, there is no broker in DDS. Instead, a GDS (Global Data Space) is defined: GDS is a kind of virtual space to which all publishers and subscribers connect and where messages are passed. In relation to IR Colonies, the GDS principle resembles a shared system environment in which all agents are located.

---

<sup>5</sup>[https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html#\\_Toc3901100](https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html#_Toc3901100) [2025-07-17]

#### 4.4. Summary

There are many application protocols designed for use in IoT networks (more than listed here, we've only covered the most common ones), but there are some similarities between them, which makes it easier to access them a bit more uniformly.

The individual protocols use either the Publish-Subscribe (MQTT, AMQP, XMPP, DDS) and/or Request-Response (HTTP REST, CoAP, XMPP) communication model with some specifics. There are differences, for example, in whether a connection needs to be established to send a message or the message can be sent directly without an "introduction", there are also differences in the types of supported messages.

We have only discussed the HTTP REST and MQTT protocols in detail, but the other protocols mentioned in the previous subsection are quite similar in handling.

It has turned out that IR Colony rules used to communicate with other agents or the environment can be implemented using both HTTP REST and MQTT protocols, although it is advisable to take this relationship into account when designing the IR Colony structure: either by creating specific objects to identify the communicating parties or, in the case of the Publish-Subscribe model, by modifying the structure of the IR Colony graph into a star shape with a central device, if any central device is to be used (the central device, the broker, is required for the MQTT protocol, other Publish-Subscribe protocols do not require or even assume any broker).

In terms of these protocols, the IR Colony environment can be implemented as a separate device or as part of the functionality of a central device.

### 5. Discussion and Conclusion

The purpose of this paper was to analyze the relevance of the IR Colony concept with respect to real-world applications in the IoT device world and to identify potential limits to its applicability. IR Colonies were directly designed to model the events in a network of IoT devices, thus no major problems were expected. However, it should be taken into account that there are a number of IoT protocols with specific communication architectures, supported messages and other features.

At the beginning of Section 3, we outlined the individual steps in implementing the IR Colony concept into real-world IoT device networks. There are a total of four areas to consider: the first area concerns communication with the hardware inside device, the second one concerns operations inside the IoT device (or agent in the IR Colony), the third one involves communication between IoT devices (agents) with each other, and the fourth area relates to communication with the rest of the world (e.g. web services or other networks). Of these items, we have focused on the third one, i.e., communication between IoT devices, because this is where compatibility with IoT-designed communication protocols will need to be considered.

Section 4 explores these protocols specifically, focusing on two of them in particular, and for the others we have only taken their similarities with the first two protocols. The IoT world originally started in chaos, with each vendor going its own way, and it was only later that interoperability between devices from different vendors started to be considered, hence the existence of a number of IoT application protocols.

It turns out that for the purpose of implementation it will not be necessary to make significant changes to the IR Colonies, only future implementation will need to be taken into account when proposing a specific IR Colony (existence of objects for device identification purposes, modification of the IR Colony graph). The role of the environment also needs to be ensured. Each of the protocols requires a slightly different approach, but on the whole these approaches are quite compatible with each other. The concept of a shared virtual space GDS used by the DDS protocol is interesting as well, resembling the IR Colonies environment.

It would be advisable not to restrict ourselves to a single IoT application protocol and instead consider the possibility of cooperating with several different protocols, because, despite their differences, we can find common features.



In subsequent research, we intend to use the results presented above to work on implementing the IR Colonies concept into the real world of IoT devices.

## Declaration on Generative AI

During the preparation of this work, the author used Grammar Checker (<https://www.grammarcheck.net/editor/>) in order to: Grammar and spelling check. After using this service, the author reviewed and edited the content as needed and takes full responsibility for the publication's content.

## 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:[doi.org/10.1016/S0304-3975\(02\)00136-6](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] A. Ehrenfeucht, G. Rozenberg, *Reaction Systems*, *Fundam. Informaticae* 75 (2007) 263–280. URL: <http://content.iospress.com/articles/fundamenta-informaticae/fi75-1-4-15>.
- [7] 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](https://doi.org/10.1142/S0129054120400043).
- [8] 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:[doi.org/10.1007/978-3-031-13502-6\\_3](https://doi.org/10.1007/978-3-031-13502-6_3).
- [9] Š. Vavrečková, Transferable programs and reactions for modeling iot network, in: *CEUR Proceedings of the 24rd Conference Information Technologies – Applications and Theory (ITAT 2024)*, 2024, pp. 111–120.
- [10] 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.
- [11] 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>.
- [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] A. Rullo, D. Saccá, E. Bertino, Past: Protocol-Adaptable Security Tool for Heterogeneous IoT Ecosystems, in: *IEEE Conference on Dependable and Secure Computing (DSC)*, Kaohsiung, Taiwan, 2018, pp. 1–8. doi:[10.1109/DESEC.2018.8625143](https://doi.org/10.1109/DESEC.2018.8625143).
- [14] D. McCain, *Implementing Cellular IoT Solutions for Digital Transformation*, Packt Publishing, Germany, 2023.
- [15] M. Danladi, M. Baykara, Low Power Wide Area Network Technologies: Open Problems, Challenges, and Potential Applications, *Review of Computer Engineering Studies* 9 (2022) 71–78. doi:[10.18280/rces.090205](https://doi.org/10.18280/rces.090205).
- [16] A. Choudhary, Internet of things: a comprehensive overview, architectures, applications, simulation tools, challenges and future directions, *Discover Internet of Things* 4 (2024). URL: <https://doi.org/10.1007/s43926-024-00084-3>.

- [17] T. Domínguez-Bolaño, O. Campos, V. Barral, C. J. Escudero, J. A. García-Naya, An overview of IoT architectures, technologies, and existing open-source projects, *Internet of Things* 20 (2022). URL: <https://www.sciencedirect.com/science/article/pii/S254266052200107X>. doi:<https://doi.org/10.1016/j.iot.2022.100626>.
- [18] IEEE Standard for Low-Rate Wireless Networks, IEEE Std 802.15.4-2024 (Revision of IEEE Std 802.15.4-2020) (2024) 1–967. doi:10.1109/IEEESTD.2024.10794632.
- [19] J. P. Dias, A. Restivo, H. S. Ferreira, Designing and constructing internet-of-things systems: An overview of the ecosystem, *Internet of Things* 19 (2022) 100529. URL: <https://www.sciencedirect.com/science/article/pii/S2542660522000312>. doi:<https://doi.org/10.1016/j.iot.2022.100529>.
- [20] 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.