Modeling Communication in Internet of Things Network using Membranes

Šárka Vavrečková

Silesian University in Opava, Bezručovo nám. 13, Opava, Czech Republic,
sarka.vavreckova@fpf.slu.cz

Abstract: In the networks used to connect Internet of Things devices, we may encounter several communication models, the most common of which is Publisher-Subscriber. These models describe parallel communication, synchronous or asynchronous, and can usually be described or simulated using membrane systems. In this paper, we focus on the simulation of communication in the Internet of Things (IoT) using a membrane system. The structure of membranes describing the components of IoT devices is proposed, as well as evolution rules, and the relation to generating and receiving data by devices.

In addition to the membrane structure and the rules, the auxiliary program code is provided for manipulation and transformation between the IoT data and the objects in the membranes.

1 Introduction

Membrane computing is a framework of parallel distributed processing introduced in 1998 by Gheorghe Păun. Research in this area is very dynamic and the new possibilities of application of this paradigm are constantly emerging. Information is available in [6, 7, 8], or the bibliography at [10].

The intent of Membrane computing is to model distributed computation using a hierarchy of membranes in cells. Mathematical models of membrane systems have been called P Systems, which refers to Gheorghe Păun.

Nowadays, the Internet of Things (IoT) is an important part of our lives. It includes various types of systems, from simple sensors (detecting for instance the temperature, motion, humidity, water, light, distance, RFID) and actuators (for example, alarms or mechanisms for handling windows, lighting) or passive recipients of data (e.g. displays) to more complex devices combining multiple sensors and actuators, communication points, gateways to other network types, and end devices (computers, mobile phones, etc.) which can control other devices. We encounter the Internet of Things at home, in shops, industry, agriculture, in the environment of large cities,...

Since the origin of the idea of IoT, various concepts have gradually emerged to describe communication between interconnected devices. Because the IoT devices are often battery-powered and therefore require energy-efficient operation, their communication is very specific.

Recently, the number of cyber attacks on IoT devices has been increasing, and the sensitivity of data generated by these devices has been growing as well, especially with regard to medical devices. The repositories of data, the place of their processing, and security of transfer are a frequent subject of discussion.[1]

Because there is usually a network of devices running in parallel, IoT is a typical example of parallel data processing. Parallel data processing can also be described using membrane systems, where the data flow among devices is simulated using evolution rules.

The use of membranes or similar principles in the world of the Internet of Things has been considered for years. Villari et al. in [11] introduce the concept of "osmotic computing" as a paradigm, the main purpose of which is to increase the accessibility of resources and services in the computer network (e.g. IoT network), including cloud services. Some of the micro-services traditionally provided mainly from the cloud (physically in large data centers) gradually migrate to the edge of the network (edge computing), i.e. they are performed on devices in the internal network. The paradigm is motivated by procedures from biology or chemistry, where solvent molecules pass through a semi-permeable membrane into other regions in the environment with higher solute concentration (osmosis).

The issue is further developed by the paper [9], which considers the way in which micro-services in particular can migrate between the cloud and edge resources and focuses more on the Internet of Things.

Villari et al. in [12] call the paradigm described in the previous mentioned papers by OSMOSIS and build on it the dynamic management of resources and services (MELs = MicroELements) on the Internet of Things, focusing mainly on the security functions of the network. Datta and Bonnet in [2] show the use of MELs in securing connected "smart" cars and other similar devices.

In the previous mentioned papers, MELs can pass between devices through *software defined membranes* (SDMs). The principle of SDM is very clearly explained in [12]. However, SDMs are a special part of the whole system, they themselves represent an interface, rather than a specific device. In this paper, we can represent the IoT devices themselves by membranes and their communica-

[1] For example, some risks of using smart light bulbs are described in https://www.trendmicro.com/vinfo/us/security/news/cybercrime-and-digital-threats/researchers-use-smart-light-bulbs-to-infiltrate-networks.

tion by the corresponding evolution rules, with the addition of code to transform data into objects and vice versa.

2 Preliminaries

2.1 Membrane Systems

We assume the reader to be familiar with the basics of the formal language theory and membrane computing. For further details we refer to [4] and [8].

The basis of membrane systems is a membrane structure inspired by the structure of a biological cell. A membrane can contain objects and/or other (nested) membranes. Objects can be handled using rules, in some membrane systems there are also rules for manipulating membranes.

One membrane is the main one (it contains all the others), we call it "the skin membrane". The remaining membranes always have one parental membrane in which they are contained.

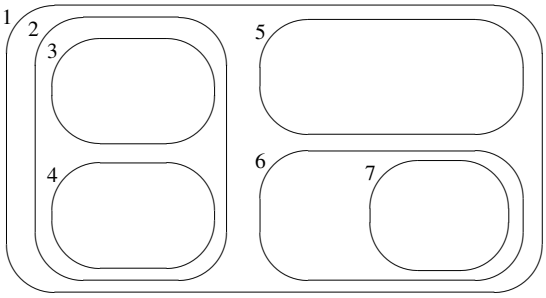

Figure 1: Example of Membrane Structure

The membrane structure can be represented graphically using Venn diagrams, brackets, or a tree of nodes representing membranes. The membrane structure presented in Figure 1 by Venn diagram is represented by the string $[\,[\,]_3\,[\,]_4\,]_2\,[\,]_5\,[\,[\,]_7\,]_6\,]_1$, and by the tree of nodes shown in Figure 2. Analogously to the diagram representation, the tree has one main (root) node, all other nodes have exactly one parent node.

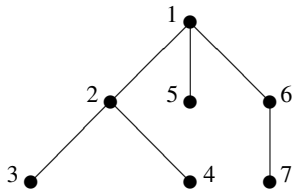

Figure 2: Tree of nodes

The membranes are identified by their labels. The membrane structure presented in Figures 1 and 2 uses simple numbers as labels.

Each membrane has its *region*: the space delimited by the given membrane, all contained objects and subordinate membranes (with their contained objects) are situated in this region. The region of a membrane corresponds to the subtree in the tree representation of the structure.

Definition 1 ([7], [1]). *Let H be a set of labels. A P System of a degree m, $m \geq 1$, is a construct*

$$\Pi = (V, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m)$$

where:

(i) *V is a nonempty alphabet, its elements are called objects,*

(ii) *μ is a membrane structure consisting of m membranes, the membranes are labeled by the elements of H,*

(iii) *w_i, $1 \leq i \leq m$, are strings representing multisets over V associated with the region of the i-th membrane in μ,*

(iv) *R_i, $1 \leq i \leq m$, are finite sets of evolution rules associated with the region of the i-th membrane in μ; an evolution rule is a pair (u, v), also written $u \to v$, where*

- *u is a string over V,*
- *$v = v'$ or $v = v'\delta$, where v' is a string over $\{a_{here}, a_{out}, a_{in_j} \mid a \in V,\ 1 \leq j \leq m\}$, and δ is a special symbol $\notin V$ representing dissolution of membrane.*

The objects can be transported by the evolution rules through membranes due to the targets out *(to the parental membrane) or* in *(to the child membrane specified by the index), or they remain in the original membrane (the* here *target).*

Details and examples can be found in [7] and [1].

2.2 Internet of Things

There are many definitions of the Internet of Things, but none of them is fully descriptive. In [5] we can find five definitions taken from different sources, and in addition several partial definitions. We can compose from these definitions the following:

Definition 2 ([5]). *The Internet of Things (IoT) is a network of various types of smart objects (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 in interoperable communication protocols and specific criteria. These smart things should have unique identities and personalities.*

Communication in the Internet of Things usually takes place in the form of a client-server, i.e. the server device provides information and the client device requests this information. There are several common communication models (or communication patterns) used in IoT networks, but these two models are very common: Request-Response and Publisher-Subscriber.

The Request-Response model comes from traditional computer networks. Clients send a request for data to

servers; the addressed server responds and delivers the requested data. In this model, the client must know not only the identity but also the address of the server in order to be able to contact it.

The Publisher-Subscriber model is closer to the needs of the Internet of Things. There are three types of components: publishers produce data, subscribers consume and process data, and the controller (often called broker) as the central point of the network mediates data (not communication). Publishers send data to the controller, not to subscribers, they don't even know about the existence of subscribers. Each node with the role of a subscriber subscribes particular types of data (often called topics) to the controller, and the controller sends the requested data to all the subscribers interested in them. Sensors are example of publishers (producers), actuators and displays are examples of subscribers (consumers).

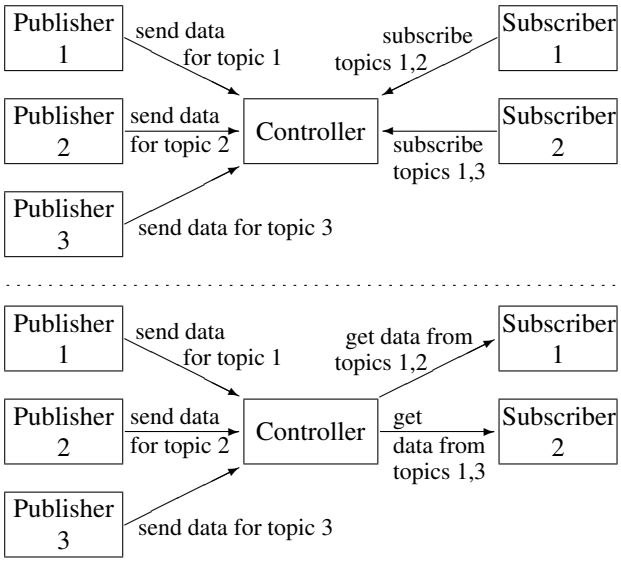

Figure 3: The Publisher-Subscriber communication model (subscriptions and forwarding)

Figure 3 shows an example network with three publishers, two subscribers and one controller, where each subscriber is interested in data from two different sources (publishers).

In practice, several different higher-level protocols are used in IoT networks, such as MQTT, XMPP, CoAP, AMQP, or simply HTTP as for classic computer networks, and most of them can communicate in both request-response and publisher-subscriber modes, however, publisher-subscriber is slightly more suitable for IoT.

More details about IoT network communication models, including protocols, can be found in [3].

3 Membrane Structure and Messages

An IoT system can either be completely separate, independent of another network, or it can be connected to the Internet (usually via a central device). For simplicity, we will assume the first option, i.e. a closed system with no connection to the outside world.

We can represent the whole system with a membrane structure and evolution rules, but in addition we need an interface to real devices. This interface will take data from a sensor and transform it into an object that the respective membrane will take over and processes with an evolution rule, and conversely take an object from a membrane and transform it into data that it passes to the proper actuator or display element.

The objects processed by membranes also contain semantic information (specific data from sensors, sender identification, etc.), but from the point of view of evolution rules, each object is a whole. For example, if a device generates different data item-by-item, two different objects are created and processed by two different evolution rules. For example, two temperature sensors (internal and external) in one device create two objects with different numbers indicating temperature, and with different source designations.

4 Simulation of Publisher-Subscriber Model

4.1 Membrane Structure

There are several possibilities to construct a membrane structure of an IoT system, every possibility implies different evolution rules. Since communication usually takes place via a central device called controller, we can either represent all devices (including the controller) with membranes embedded in the skin membrane, or the controller can play the role of the skin membrane. In both cases, the objects will always be transferred via the controller.

We will focus on the second case (the controller as the skin membrane). The considered structure has three levels: inside the skin membrane (controller) there is the second level for IoT devices, and the third level is presented

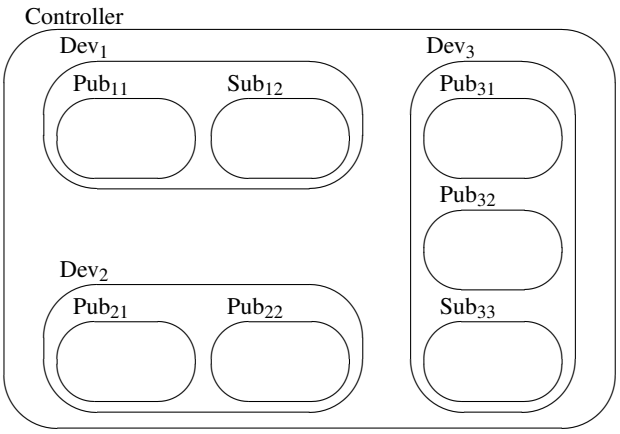

Figure 4: Example of IoT Membrane Structure with Publishers and Subscribers

by components (inside IoT devices) acting as publishers or subscribers. Each IoT device carries one or more components.

Figure 4 shows the membrane structure of an example IoT system with one controller and three IoT devices. The first device contains one publisher (a sensor) and one subscriber (an actuator), the second device holds two publishers (e.g. it can be a weather station with a thermal and moisture sensors). The third device consists of two publishers and one subscriber: e.g. a smartphone with a gyroscope and accelerometer as sensors (of course, a real smartphone has many more sensors, here we have only a simplified case), and its display can act as a subscriber.

4.2 Objects and Evolution Rules

Objects representing messages sent from publishers via the controller to subscribers carry the following information:

- data (e.g. temperature, moisture, motion, touch, image from camera, etc.),
- direction related to the controller (from publisher or to subscriber),
- both publisher IDs (device ID, component ID),
- for the direction from the controller to a subscriber, both subscriber IDs.

For objects, we will use the following syntax:

- $p(\langle data \rangle, publisher\ deviceID, publisher\ componentID)$ for the direction from a publisher to the controller,
- $s(\langle data \rangle, publisher\ deviceID, publisher\ componentID, subscriber\ deviceID, subscriber\ componentID)$ for the direction from the controller to subscribers.

Differentiation by the symbols p or s is needed for the correct operation of the controller.

First, for all publishers Pub_{ij}, where i is the deviceID and j is the componentID, we need the rule:

$$p(\langle data \rangle, i, j) \rightarrow p(\langle data \rangle, i, j)_{out}$$

The generation of this symbol inside a publisher is solved by the publisher's code in the following subsection.

The next part of the path for this object leads through the device carrying the publisher. For each device Dev_i and for all components with the componentIDs denoted by j where the j-th component is a publisher (Pub_{ij}):

$$p(\langle data \rangle, i, j) \rightarrow p(\langle data \rangle, i, j)_{out}$$

The object is now in the skin membrane. First, the controller has to change the type of the object (from p to s), and make as many copies of the object as the subscribers have subscribed to, and add the subscriber information to each object. This action is solved by the controller's code in the following subsection.

The p-type objects were transformed to s-type objects, and these objects are transported by these rules from the skin (the controller) membrane: for all devices Dev_i, for all their components j and for all possible publishers Pub_{kl}:

$$s(\langle data \rangle, k, l, i, j) \rightarrow s(\langle data \rangle, k, l, i, j)_{in_{\mathrm{Dev}_i}}$$

The last part of the path is from the i-th device into the j-th subscriber. For all devices Dev_i, all their components indexed by j which are subscribers and for all possible publishers Pub_{kl}:

$$s(\langle data \rangle, k, l, i, j) \rightarrow s(\langle data \rangle, k, l, i, j)_{in_{\mathrm{Sub}_{ij}}}$$

4.3 Semantics and Auxiliary Code

The proposed evolution rules only address transfers between membranes, but in addition, the following operations need to be added:

- the ability to generate data by publishers,
- the ability to transform incoming published objects p to the corresponding outgoing objects s forwarded to subscribers,
- the ability to process data by subscribers.

The code in Algorithm 1 defines properties of objects, the controller, devices and their components.

Algorithm 1: Controller, Device, Component

```
object:
  objType, // from_publisher (p) | to_subscriber (s)
  data,
  // the publisher's and the subscriber's IDs:
  pub_devID, pub_compID,
  sub_devID, sub_compID;

controller:
  devices [],
  subscriptions [] (pub_devID, pub_compID,
     sub_devID, sub_compID);

device:
  deviceID,
  components [];

component:
  deviceID, compID,
  compType;   // publisher | subscriber

publisher(child of: component);
subscriber(child of: component);
```

This pseudocode assumes that for simplicity we use object arrays or lists, which in addition to accessing members offer other properties and functions (getting number of members, easy addition of a new member, etc.).

Algorithm 2: Controller's Code

```
function controller.Start()
begin
    while true do if ((self.presentObjectInMembrane(&object)) and (object.objType = from_publisher)) then
        // one published object must be copied to objects for all subscribers who have ordered it:
        self.removeObjectFromMembrane(object);
        resObjects = "";    // resulting multiset of objects for all subscribers
        object.objType = to_subscriber;
        for (i = 1 to self.devices.count) do
            for (j = 1 to self.devices[i].components.count) do
                // if there is a subscription for the given publisher and subscriber, create and add the object:
                if (self.subscriptions.find(object.pub_devID, object.pub_compID, i, j)) then
                    object.sub_devID = i;
                    object.sub_compID = j;
                    resObjects.add(object);    // add the object into the multiset of objects
                end
        self.exportToMembrane(resObjects);
    end
end

function controller.OrderSubscription(pub_devID, pub_compID, sub_devID, sub_compID)
begin
    if ((self.devices[pub_devID].components[pub_compID].compType == publisher) and
     (self.devices[sub_devID].components[sub_compID] == subscriber)) then
        self.subscriptions.add(pub_devID, pub_compID, sub_devID, sub_compID);
end
```

Publishers and subscribers differ in their functions rather than their properties. The following algorithms describe functions of the controller, publisher and subscriber.

Algorithm 2 shows two controller's functions. The first function describes a normal behavior of the controller that checks in a loop whether an object of the type "from_publisher" has appeared inside its membrane. If the object is present, it is replaced by the string of the objects of the type "to_subscriber", one for every subscriber registered for subscription from the given publisher.

Alternatively, we could use the evolution rule replacing one published object by the string of objects intended for all subscribers registered for the given publisher's data:

$$p(\langle data\rangle, i, j) \rightarrow \bigcup_{k,l} s(\langle data\rangle, i, j, k, l)_{here}$$

(for all possible publishers), where i, j are the publisher's IDs and k, l are the IDs of all subscribers registered to receive objects originating from the given publisher. However, this procedure is disadvantageous: any change in the registration of subscriptions would cause necessity of changing these rules.

The function OrderSubscription() is called by subscribers (or alternatively by devices) to sign up to subscribe certain objects from a specific publisher. The controller checks if the both components (the declared publisher and subscriber) are of the correct type, and adds the corresponding entry to the list of subscriptions.

Algorithm 3: Publisher's Code

```
// This function is called by the host device, the new
   member of the components array is created:
function publisher.Start(deviceNum,
 compNum,...)
begin
    self.compType = publisher;
    self.deviceID = deviceNum;
    self.compID = compNum;
    // Setting producing interval (if needed) and
       other implementation-dependent settings.
end

// Called regularly according to the synchroniza-
   tion interval, or when a predefined event occurs:
function publisher.Produce()
begin
    new object;
    object.objType = from_publisher;
    object.data = GetDataFromSensor();
    object.pub_devID = self.deviceID;
    object.pub_compID = self.compID;
    self.exportToMembrane(object);
end
```

We could also add an unsubscribe function that would remove the surplus entry from the list of subscriptions.

In Algorithm 3 we can find two functions as well. The first one initializes the given publisher, sets all the necessary parameters, including the synchronization interval in which the publisher should produce data, if necessary. Publishers do not have to produce at regular intervals, they can be linked e.g. to a specific event in the system, but a function should be added for this reason.

The second function (Produce()) is called when the given publisher has to produce data for a new object. The object is created and exported to the membrane.

Algorithm 4: Subscriber's Code

```
// This function is called by the host device, the new
   member of the components array is created:
function subscriber.Start(deviceNum,
 compNum,...)
begin
    self.compType = subscriber;
    self.deviceID = deviceNum;
    self.compID = compNum;
    // Other implementation-dependent settings.
    while true do
        if ((ObjectFound(&object)) and
          (object.objType == to_subscriber)) then
            self.removeObjectFromMembrane(object);
            self.ProcessData(object.data,
              object.pub_devID,
              object.pub_compID);
        end
    end
end

// Called when the device needs to receive
   a certain type of messages (objects) from
   a specific publisher via this subscriber:
function subscriber.Subscribe(devID, compID)
begin
    controller.OrderSubscription(devID, compID,
      self.deviceID, self.compID);
end
```

Algorithm 4 shows two functions for subscribers. The first one is used to initiate the given subscriber and ensure the subscriber's normal behavior, i.e. when the ordered object appears in the environment, this object is picked up and processed accordingly.

The second function is called by the subscriber (or alternatively by devices) to order subscriptions.

5 Discussion

The indicated membrane structure, rules and additional code are one of the possible solutions, there are other possibilities.

The first possible change is to place the controller at the level of other devices: the controller would not be identified with the skin membrane, but it would have its own membrane inside the skin membrane. The rules would need to be modified, especially the skin membrane would forward all symbols "p" into the controller's membrane and all symbols "s" into other membranes according to the added information.

The advantage of this solution is that the controller could also contain components serving as publishers and subscribers, the disadvantage is the longer path of objects. In practice, this can be imagined as a central device containing a display (subscriber) with a touch screen and buttons to control other devices (publishers), or to manage subscriptions.

If we wanted to use the components in the controller using the solution outlined in the previous sections, we would have to add a membrane for a new "virtual" device with the controller's publishers and subscribers.

Another alternative would be only a two-level structure: a skin membrane representing the controller and inner membranes representing the individual publishers and subscribers. The advantage would be a shorter path of objects between a publisher and a subscriber (with no intermediate stages in the form of device membranes), the disadvantage is the lower clarity of the structure.

In IoT we can encounter paradigms of cloud computing (data is sent to a cloud, the cloud can act as a controller) or edge computing (data is usually preprocessed where it originates, then it can be sent elsewhere). Cloud computing and edge computing can also be included in the model, only this somewhat complicates the path of the object through the membranes, in real world we do not have direct control over some part of this path.

This work was supported by The European Union under European Structural and Investment Funds Operational Programme Research, Development and Education project "Zvýšení kvality vzdělávání na Slezské univerzitě v Opavě ve vazbě na potřeby Moravskoslezského kraje", CZ.02.2.69/0.0/0.0/18_058/0010238.

References

[1] Busi, N.: Causality in Membrane Systems. In Membrane Computing. Springer Berlin Heidelberg (2007) 160–171. ISBN 978-3-540-77312-2.

[2] Datta, S.K., Bonnet, C.: Next-Generation, Data Centric and End-to-End IoT Architecture Based on Microservices. IEEE International Conference on Consumer Electronics – Asia (ICCE-Asia), 2018, pp. 206–212, doi: 10.1109/ICCE-ASIA.2018.8552135.

[3] Dizdarević, J., Carpio, F., Jukan, A., Masip-Bruin, X.: A Survey of Communication Protocols for Internet of Things and Related Challenges of Fog and Cloud Computing Integration. Association for Computing Machinery. New York, NY, USA, 2019, 51(6), 29 p. ISSN 0360-0300.

[4] Hopcroft, J.E., and Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley, 1979.

[5] Kassab, W., Darabkh, K.A.: A–Z survey of Internet of Things: Architectures, protocols, applications, recent advances, future directions and recommendations. Journal of Network and Computer Applications, vol. 163 (2020), ISSN 1084-8045, 49 p., https://doi.org/10.1016/j.jnca.2020.102663

[6] Păun, Gh.: Membrane Computing: An Introduction. Springer, Heidelberg (2002).

[7] Păun, Gh., Rozenberg, G.: A Guide to Membrane Computing. Theor. Comp. Science, vol. 287, issue 1, pp. 73–100 (2002).

[8] Păun, Gh., Rozenberg, A., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press, New York (2010).

[9] Sharma, V., et al.: Managing Service-Heterogeneity using Osmotic Computing. International Conference on Communication, Management and Information Technology (ICCMIT 2017), 7 pages, Warsaw, Poland, 2017. arXiv:1704.04213

[10] The P Systems Website: http://psystems.eu (Accessed June 4, 2020)

[11] Villari, M., et al.: Osmotic computing: A new paradigm for edge/cloud integration. IEEE Cloud Computing 3.6 (2016) 76–83.

[12] Villari, M., et al.: Software Defined Membrane: Policy-Driven Edge and Internet of Things Security. In IEEE Cloud Computing, vol. 4, no. 4, pp. 92–99, July/August 2017, doi: 10.1109/MCC.2017.3791014.