Notes on Relationship of P Colonies to Osmotic Computing and Computer Viruses

Šárka Vavrečková[1]

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

Abstract

P colonies with transferable programs are one of the newer variants of P colonies, which allow the transfer of programs between agents and the environment when certain conditions are met. Code transfer is also known from other concepts: in the concept of Osmotic computing we encounter the transfer of MELs between IoT devices and the cloud, and the long-standing concept of computer viruses uses infecting common applications with appended code. In this paper, the reader will find an outline comparison of these three concepts concerning the possibilities of transferring code fragments or programs.

Keywords

P colony, transferable program, Osmotic computing, MEL, virus, shellcode

1. Introduction

P colonies are a simple computational model based on membrane systems. P colonies were introduced in [1] in 2004 and since then many variations (types of programs, form of environment, etc.) have been developed. Focusing on the basic variant, the environment containing objects of a given type is shared by agents that also contain objects in their internal environment and are equipped with programs consisting of rules. The programs allow the agents to influence not only themselves but also the environment. In paper [2], the concept of P colonies with transferable programs is introduced, where programs can be transferred between an agent and the environment and vice versa.

The principle of traveling programs, or code in general, can be found elsewhere: e.g. in the concept of Osmotic computing or (if we focus on the nowadays very frequent topic of cybersecurity) also in the activity of viruses, shellcode, etc.

Villari et al. in [3] introduce the concept of "Osmotic computing" as a paradigm intended mainly for IoT (Internet of Things) network, the main purpose of which is to increase the accessibility of resources and services in a computer network, including cloud services. *MicroElements* (MELs) are simple entities consisting of programs (MicroServices) and data (MicroData). IoT applications can be decomposited into MELs. The interface between the IoT application and the edge environment (Edge DataCenters), and also between the edge environment and the cloud (Cloud DataCenters), is called *membrane*: MELs can migrate through membranes to where they are needed.

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 in [4] considering the way in which MELs, in particular, can migrate between the cloud and edge resources and focusing more on the Internet of Things. Datta and Bonnet in [5] show the use of MELs in securing connected "smart" cars and other similar devices.

A virus is malicious program code that spreads by infecting available programs with a copy of itself. There is quite a lot of literature and websites about computer viruses, e.g. [6].

According to [7], shellcode can be described as a code injection attack. An attacker inserts a code into an existing chosen file to access the command interface (shell) of the compromised system. Shellcode is one of the ways a virus can work.

In Section 2 we define P colonies, and add a description of the possibility of using transferable programs in P colonies. This concept is explained with examples. In the same section, we follow the definition of the principle of Osmotic computing, where we mainly focus on explaining the possibility of transferring MELs between the involved devices. The last topic of Section 2 is the issue of computer viruses and shellcode. Here we also focus on the process of transmitting or adding new code to infected programs.

In the next two sections, we will try to capture the correlations between P colonies with transferable programs and two other concepts in which computational elements (e.g., program code) are being transmitted: the concept of osmotic computing and the principle of operation of computer viruses and shellcode.

ITAT'23: Information technologies – Applications and Theory, September 22–26, 2023, Tatranské Matliare, Slovakia

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

2. Preliminaries

We assume the reader to be familiar with the basics of the formal language theory and membrane computing [8, 9]. We denote the length of a word w by $|w|$. The empty word is represented by the symbol ε, so $|\varepsilon| = 0$.

2.1. P Colonies

Membrane computing is a framework of parallel distributed processing introduced by Gheorghe Păun in 1998. Information about this paradigm is available in [10], or the bibliography at http://ppage.psystems.eu/ [2023-07-02]. Membrane systems are based on the hierarchical structure of membranes in cells and can be used to model distributed computing. Mathematical models of membrane systems have been called P Systems.

P colonies develop the concept of P systems by enriching the system with agents that evolve activities according to specific programs.

Definition 1 ([2]). *A P colony of capacity k, $k \geq 1$, is a construct*

$$\Pi = (A, e, f, v_E, B_1, \ldots, B_n) \text{ where}$$

- *A is an alphabet, its elements are called objects,*
- *$e \in A$ is the environmental object of the colony,*
- *$f \in A$ is the final object of the colony,*
- *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 itself, i.e. its internal environment: an object a inside the agent is rewritten to the specified object b. The rule can only be applied if at least one occurrence of the object a exists in the internal environment of the agent.

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. Also, this type of rule can be applied only if the specified objects exist in the given environments.

The checking rules (r_1/r_2) are composed of two rules r_1 and r_2 of any of the previous two types. Only one of these rules will be applied, with the first listed having a higher priority. Only if the first rule cannot be executed, the second rule in order may be executed.

The initial configuration of a P Colony Π 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. Generally, the configuration of a P Colony Π with capacity k is

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

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

Several derivation modes have been defined, differing in the way the rules are selected at each derivation step. In the maximally parallel derivation mode, all agents can work parallely in each derivation step (each agent non-deterministically chooses one of its programs with applicable rules). In the sequential derivation mode, only one (non-deterministically chosen) agent can work in each derivation step. The calculation halts if no agent finds an applicable program.

2.2. Transferable Programs in P Colonies

In [2], the concept of transferable programs for P colonies has been introduced. A transferable program is an ordered pair (program, condition) located inside an agent or the environment. The program can be transferred from the agent to the environment or from the environment to the agent (from the source to the destination), and the stated condition determines under what circumstances this transfer will occur.

The condition can be one of the following two types:

- Object condition: the *destination* must contain certain objects for the program to be transfered (or must not contain when there is the negation symbol in the condition). This type of condition is a set of multisets of objects, the size of the given multisets is equal to capacity of the P Colony.
- Program condition: the *destination* must contain a specified program (or must not, if the negation symbol is present).

Thus, the definition of P colony presented in the previous subsection changes in the part specifying the programs of agents, and it is also necessary to add programs located in the environment to the definition, the rest remains the same.

Example 1. Let Π be a P colony of capacity 2. Capacity is reflected in both the number of objects in the agent

environment and the number of rules in each agent program.

For the illustration, we present here an example program of one of agents, using object condition:

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

This means that if this program is in an agent it can only be transferred to the environment when there are at least two occurrences of the d object in the environment and there is no bg pair. If this program is in an environment, it can be transfered to an agent when the condition holds for the internal environment of that agent.

What happens to the program at the original location if the conditions for transfer are met? According to [2], a program can be classified as *permament*, in this case one copy of the program remains at the original location and the other is transferred. The program remains at the original location (including the "permanent" property), but loses this property at the new location. If a program is not permanent, it is removed at its original location.

In each computation step, an agent can either apply one of its applicable programs or transfer one of programs in or out.[2] This implies that the transfer of programs is actually an analogy of programs (but the action would not be transferable).

Example 2. Let Π be a P colony of capacity 1 with transferable programs. There are two agents:

$$B_1 = \big(e,\ \{\langle e \to a\rangle\,,\ \langle e \to b\rangle\,,\ \langle x \leftrightarrow e\rangle\,,\ \langle y \to b\rangle\}\big)$$
$$B_2 = \big(e,\ \{\langle e \to a\rangle\,,\ \langle e \to c\rangle\,,\ \langle x \leftrightarrow e\rangle\,,\ \langle y \to a\rangle\}\big)$$

In their initial state, these agents do not contain any transferable programs (although they could). We have only the e symbols in the environment at the beginning of the computation. For the purpose of clarity, if we limit reprezentation of the configuration of the system to just the states of the agents and the environment (the configuration of the environment properly includes programs not imported in any agent), the initial configuration is very simple:

$$(e, e, e)$$

There are the following transferable programs in the environment at the beginning of computation:

$$\big\{\ (\langle a \to x\ ;\ \{a\}\rangle)_p\,,\ (\langle b \leftrightarrow e\ ;\ \{b\}\rangle)_p\,,\ (\langle c \to y\ ;\ e \to c\rangle)\ \big\}$$

The first two programs contain object condition, and the third program contains program condition. The p symbol in the subscript indicates permanent transferable programs.

During the first step of the computation, B_1 uses one of the programs that overwrite the e symbol, for example the first one: $\langle e \to a\rangle$. B_2 also has a choice of two programs, for example $\langle e \to c\rangle$. The configuration has been modified to

$$(a, c, e)$$

There are three transferable programs in the environment. The first one is suitable for B_1, the third for B_2. So the agents are changed as follows:

$$B_1 = \big(a,\ \{\ \langle e \to a\rangle\,,\ \langle e \to b\rangle\,,\ \langle x \leftrightarrow e\rangle\,,\ \langle y \to b\rangle\,,\ (\langle a \to x\ ;\ \{a\}\rangle)\}\big)$$

$$B_2 = \big(c,\ \{\ \langle e \to a\rangle\,,\ \langle e \to c\rangle\,,\ \langle x \leftrightarrow e\rangle\,,\ \langle y \to a\rangle\,,\ (\langle c \to y\ ;\ e \to c\rangle)\}\big)$$

The set of transferable programs in the environment has changed. The third program was not classified as permanent, so it is no longer part of the environment after the transfer.

$$\big\{\ (\langle a \to x\ ;\ \{a\}\rangle)_p\,,\ (\langle b \leftrightarrow e\ ;\ \{b\}\rangle)_p\ \big\}$$

In the next step of the calculation, the agents can use the newly obtained programs, i.e. the state of the agents and the environment has been changed as follows:

$$(x, y, e)$$

B_1 can use its third program, B_2 can use its fourth program, the new state of the agents and the environment is

$$(e, a, x)$$

In the next steps, the agents will use their own programs, import other programs from the environment, or export a transferable program to the environment for use by another agent.

The above example shows how transferable programs behave. Programs can also be transferred in the other direction, i.e. from agents into the environment. This allows an agent to drop a program it does not need.

Another example of using P colonies with transferable programs, which would be more complex to describe, is as follows: Agents are gardeners. Each of them grows a specific plant whose progressive development (growth phase) and condition (healthy, dry, insect infested, etc.) is captured in their status objects. Agents pick up tools (i.e., programs) from the environment as needed (according to these objects), and return unneeded tools.

2.3. Osmotic Computing

In biology and chemistry, osmosis is the process of transferring molecules between two media of different densities across a semipermeable membrane. Osmotic computing carries this principle into the field of technology: here it is about the dynamic migration of microservices between different parts of a computer network.

According to Osmotic computing[4], an application (very often running on an IoT device) can be decomposed into parts called MicroElements (MELs) consisting of MicroServices (code) and MicroData (data). Since IoT devices usually don't have much computing power, it makes sense to move some of the computation (i.e., some of the MELs) elsewhere, such as to another device on the same network or to the cloud. To do this, we need the MEL to be relatively self-contained, independent of other parts of the application, and also to have an interface to enable these transfers and provide the necessary communication.

Definition 2 (according to [11]). The *infrastructure topology* in Osmotic computing can be represented by directed graph $T = (V_T, E_T)$ where V_T is the set of nodes (vertices) and E_T is the set of pars of vertices called links. Each element in V_T is characterized by various computing parameters (CPU, memory etc.), each element of E_T is characterized by various network parameters (latency, bandwidth, etc.).

Osmotic toolkit consists of pre-configured container images intended for various elements in V_T.

Applications in the Osmotic ecosystem can be represented by graph $P = (V_P, E_P)$ where V_P is the set of MELs and E_P is the set of their interconnections. Each MEL in V_P is characterized by the specific resource requirements, constraints and scheduling policies. The edges in the set E_P determine between which elements of V_P MELs can be transferred.

The implementation is based on the principle of SDN (Software Defined Network), i.e. computer network virtualization. Software representation of V_P nodes (MELs) is possible using containers ([11] used the container system Docker for this purpose). Each device (node V_T) can run various number of containers with MELs (nodes V_P) according to its capabilities and MELs requirements.

Containers are actually pre-built images of systems (operating systems running on computers, IoT devices, network devices, etc.) so not just the application itself, but everything in software it needs. The purpose is to simplify the application deployment as much as possible, including configuration. In the case of Osmotic computing, there is also an ecosystem for migration and communication of MELs. A MEL can be moved to a different node V_T (respectively to a different container V_P) if the conditions for it to run, such as memory capacity or CPU performance, are not met in the original location. Another reasons for moving may be to optimize network communication, or to ensure cybersecurity, where we pull MELs from cloud to our own network.

Each element of the set V_T is managed by the *Node Manager* component, the whole system is managed by a special element of V_T called *Osmotic Resource Manager*. The Node Manager continuously analyzes the device properties and checks whether the MELs in the corresponding set V_E are satisfied. The Osmotic Resource Manager then balances the workload across the network in cooperation with the Node Managers from each device.[12]

2.4. Virus and Shellcode

The term computer virus is known, at least in a hint, to the general public. It is computer code embedded in a host program that usually performs operations not intended by the author of the program. In addition to these operations, the virus also infects other files, i.e. it inserts its own code (or its functional equivalent) into other reachable files. The malicious code executed by the virus is called *payload*. The payload is usually encrypted or otherwise encoded to make it more difficult to detect. In addition to the payload, another code is embedded in the infected file to guide the computation to the payload and, of course, to decrypt or decode it.[6, 13] Shellcode can be one of the ways a virus can work, i.e. its payload.

Shellcode is, according to [14], the code that allows a hacker to perform his intent (get access to a file, get access to certain data, escalate access privileges, start encryption, etc.). Originally, this code has been used to run a shell (a command environment through which a hacker could e.g. remotely control the system), hence the name, but nowadays the term is more generally understood (shellcodes are also able to run other types of commands or gain higher access privileges).

Definition 3 (according to[7]). Shellcode can be thought of as a sequence of additional commands added to a program, which is executed immediately after the program itself starts and usually lead to the exploitation of some vulnerability in the system or application. A special type of shellcode, called a bind shell, connects to a network port (HTTP or FTP ports are often used) when executed and allows an attacker to establish a remote connection to the device on which it is running.

Infecting a program is usually done by inserting a special sequence of Bytes into the target program and modifying the rest of the code to force the processor to execute the added code at the correct time. And, of course, the purpose is to hide these changes from antivirus programs as much as possible.

If an antivirus finds an infected program, it either moves it to quarantine (it defacto makes the malicious code inaccessible) or it can try to repair the infection, which means the reverse procedure to the one described above.

As mentioned above, the payload of a virus is usually encrypted and there is also a decryption sequence in

the code. While the payload can be very heterogeneous and therefore harder to detect (and it is encrypted), the decryption instructions are easier to detect. *Polymorphic viruses*[13] can modify their decryption sequence as they spread, making detection more difficult. The modification is based on a simple principle: the same computational operation can be performed in many different ways. For example, the result 25 can be reached by $5 * 5$, $20 + 5$, $3 * 6 + 7$, $100/4$, $9 * 7 - 38$. Code instructions can also be put together in various ways: we can add instructions without directly affecting the result, etc.

Antivirus programs use an emulator to detect polymorphic viruses, in which they emulate the process of decrypting a potential locations in a program with an encrypted payload. Attackers have developed a new generation of viruses: metamorphic viruses. *Metamorphic code*[13] does not rely on making copies of the payload, but what was to apply to the polymorphic code for the decryption sequence applies to the payload in the metamorphic code: when spread, a functionally equivalent different code is produced instead of a copy of the payload. The problem of detecting metamorphic viruses is NP-complete[13]. However, not only attackers but also defenders can use metamorphic code. It can be very effective in defending against certain types of attacks, as even defensive tools find it useful to be hidden.

3. P Colonies with Transferable Programs vs. Osmotic Computing Concept

In this section, we focus on the correlations between P colonies with transferable programs and the concept of Osmotic computing.

1) The concept of Osmotic computing can be described using graphs – Definition 2 uses graph T for the infrastructure and graph P for the communication structure of applications. Since not all nodes of the graph T can host MELs and there can be multiple MELs (from V_P) in a single node in V_T, the graphs do not overlap in their nodes.

We can also represent the structure of a P colony with transferable programs using graphs. The graph T for the infrastructure is very simple, the agents are not connected to each other but only to the environment. With the next graph P we can describe the nodes and paths for transferring programs. Since programs can be both at the agents and in the environment, the two graphs practically overlap.

2) In both systems there is some transfer conditionality (programs in the case of P colonies and MELs in the case of Osmotic computing). A P colony program is directly equipped with a transfer condition, this condition concerns either the (non) presence of certain objects or programs in the target. MELs in the Osmotic computing concept have a condition built into the properties of the elements of the set V_P, the transport of a given MEL is possible if these conditions match the equipment of the target node in the set V_T.

3) P colonies are a mathematical model and have been designed to be easy to analyze their computation. Capacity of P colony is determined, which is reflected in the number of objects in each agent's internal environment, and also in the number of rules in the agents' programs. As for transferable programs, the number of rules in such a program must also be equal to the capacity of the P colony.

The concept of Osmotic computing is more free in this respect, it is heterogeneous in principle. The analogue of the agents from P colonies are the devices in the network hosting MELs, which can be very different from each other in their parameters, and the analogue of the transmitted programs are just the MELs. MELs can be of varying complexity (longer or shorter programs). On the other hand, there is also a similarity: an agent of a P colony may contain different amounts of programs, a device in the Osmotic computing concept may contain various amounts of MELs. In both cases, the number of MELs can be changed dynamically.

4) Programs in P colonies use three types of rules, as follows from Definition 1. Their task is to work with objects (transformation or transfer). MELs, on the other hand, are composed of instructions prescribed by implementations for a given host type (operating system and processor on a given device). Thus, the set of usable instructions varies but is certainly very large (tens, hundreds). These instructions may also work with data (analogous to objects), but they may also be other types of tasks.

5) Although the rules in the P colony do not directly create new symbols, the e object is available in any quantity. Therefore, the quantity of other objects within the environment can be freely increased simply by transforming the environment object into some other object. MELs typically run in an IoT device (or, for example, in cloud). IoT devices are very often composed of sensors, and sensors are typically the source of data. Thus, MELs can generate data without the need for any auxiliary object. However, the impacts are similar. The reverse operation (removal or consumption of objects or data) is also implemented in P colony by transforming an object into an e object; for MELs, this procedure is straightforward.

6) The concept of Osmotic computing is rather intended to be centralized: each node in V_E has its own Node Manager, and there is an Osmotic Resource Manager in

the network managing the whole system. In contrast, P colonies are decentralized in principle.

4. P Colonies with Transferable Programs vs. Computer Viruses

Next, we focus on the similarities between P colonies with transferable programs and the operation of viruses and shellcode.

1) In the case of P Colony, the transfer of programs can be considered as an alternative activity. An agent can either execute one of its programs, or receive a new program from the environment, or divest itself of one of its programs. In the Osmotic computing concept, the migration of MELs can also occur repeatedly and can be seen as an alternative activity on the devices involved.

In the case of a virus, the program is infected once (unless it is a massive infection with a wide range of different viruses), and the opposite operation (removal of malicious code by the antivirus) is not expected to be repeated often. These operations are not considered an alternative to other activities.

2) Another significant difference is the contextuality of operation. P colonies with transferable programs have a certain type of context built into the program transfer conditions, consisting of the presence or absence of objects or programs in the target. The context for viruses and the shellcode embedding mechanism is much more strict: each shellcode must be customized for the target system (it must be made up of instructions that the target system understands), and in addition, the application that is to host the added code must be modified in multiple places (it is not enough just to add a new section of code e.g. to the end of the given file). There is no space for details in this paper (see [13], e.g. hackers use the buffer overflow mechanism, which requires modifications at several places in addition to inserting new code).

3) The next criterion being compared is similar to criterion #3 and #4 from the previous section. Also, the structure of virus code is much less strict than the structure of agents in the P colony. It depends on what exactly the shellcode (or other type of virus payload) is supposed to do, and its length and complexity depends on that. There are also different approaches to writing shellcode. The best known type probably is Aleph1[14], but there are many others. The situation is further complicated by polymorphic and metamorphic viruses, where there is a large variability even within the same virus family.

If we wanted to simulate infecting agents with malicious code in a P colony with transferable programs, it would not be so complicated. It would be enough to insert one or more programs containing, for example, rules of type $a \rightarrow e$ for various symbols $a \in A$ with a suitable condition (here the condition would prescribe the existence of the symbol a in the agent's internal environment). If an agent were infected by such a rule, it would lose its internal state.

4) Similar to the concept of Osmotic computing, for viruses, it is not so important what (numerical) result is reached at the end of the computation, but the process itself is significant, unlike P colonies.

5) Infecting an application with malicious code does not usually mean that part of the original application code is lost (although this is not impossible) but rather that something new is added. Overwriting occurs elsewhere rather than inside the application itself, for example, data on disk is encrypted or deleted. If we are looking for an equivalent in the transferable programs in P colonies, it would be a transferable program containing rules that affect the environment (in P colony, an agent cannot affect another agent directly, except by planting a program through the environment).

6) If we focus on the activity of antivirus, simulating its operations in a P colony with transferable programs would consist in enriching the environment with "disinfecting" programs. If an agent is disabled by a malicious program (e.g. such that its environment consists only of objects for which it has no programs), then in the next derivation step it imports a suitable disinfecting program whose purpose is to repair its internal environment.

7) P colonies are decentralized in principle. The system of computer viruses is decentralized too. But, on the other hand, the operation of antivirus, which is based on a similar principle (finding a target and working with its code), is centralized.

5. Conclusions

P colonies are motivated by biological processes (simple cell, passing of nutrients across membranes, etc.) but are intended from the beginning as a mathematical model. The concept of Osmotic computing is motivated by processes in biology and chemistry (membrane as a communication interface) but from the beginning they are intended to be used in practice, in technology. While in the computational model of P colonies, f is defined as the final object and the end of the computation is assumed (the number of objects f is considered as the result of the computation), the concept of Osmotic computing does not assume any end of the activity, the important thing is the process itself. This is not only a significant difference between the two systems, but it also complicates somewhat the comparison of other parameters of these systems.

The design of a virus and shellcode in general is also motivated by an ending (infecting other programs and selected malicious actions in the form of encrypting content, taking control, stealing data, etc.), but this ending cannot be simply represented by numbers.

In conclusion, some similarities can be found between the three systems (the possibility of code transfer), other similarities always exist in the particular pair being compared. Therefore, to a limited extent, there are possibilities to simulate certain operations of one system in another, as we showed in the previous section on the comparison of P colonies with transferable programs and viruses.

Colonies with transferable programs have not yet been studied in detail; in fact, there is no precise definition. For example, it is not clear from [2] whether the action of importing/exporting a program is an alternative for agents to using some other program, or whether this import/export can be performed in the single step at the same time as the use of another program. In this paper, we assume the first option.

Therefore, future research can be targeted in the following direction: to establish a precise and detailed definition of P colonies with transferable programs, to investigate the behavior of the system in specific situations and with specific parameters, taking into account the impact of the use of transferable programs, the impact of marking programs as permanent, etc.

Another interesting direction of research is the possibility of using P colonies with transferable programs to simulate various systems and processes, e.g. in the area of security (distributing program updates, managing encryption algorithms,...).

References

[1] 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.

[2] 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.

[3] M. Villari, M. Fazio, S. Dustdar, O. Rana, R. Ranjan, Osmotic computing: A new paradigm for edge/-cloud integration, IEEE Cloud Computing 3 (2016) 76–83.

[4] V. Sharma, K. Srinivasan, D. N. K. Jayakody, O. Rana, R. Kumar, Managing service-heterogeneity using osmotic computing, in: International Conference on Communication, Management and Information Technology (ICCMIT 2017), Warsaw, Poland, 2017.

[5] S. K. Datta, C. Bonnet, Next-generation, data centric and end-to-end iot architecture based on microservices, in: IEEE International Conference on Consumer Electronics – Asia (ICCE-Asia), 2018, pp. 206–212. doi:10.1109/ICCE-ASIA.2018.8552135.

[6] D. Ferbrache, A Pathology of Computer Viruses, Springer Science & Business Media, 2012.

[7] P. Liguori, E. Al-Hossami, D. Controneo, R. Natella, B. Cukic, S. Shaikh, Can we generate shellcodes via natural language? an empirical study, Automated Software Engineering 29 (2022). doi:https://doi.org/10.1007/s10515-022-00331-3.

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

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

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

[11] A. Buzachis, D. Boruta, M. Villari, J. Spillner, Modeling and emulation of an osmotic computing ecosystem using osmotictoolkit, in: 2021 Australasian Computer Science Week Multiconference, ACSW '21, Association for Computing Machinery, New York, NY, USA, 2021. URL: https://doi.org/10.1145/3437378.3444366. doi:10.1145/3437378.3444366.

[12] M. Nardelli, S. Nastic, S. Dustdar, M. Villari, R. Ranjan, Osmotic flow: Osmotic computing + iot workflow, IEEE Cloud Computing 4 (2017) 68–75. doi:10.1109/MCC.2017.22.

[13] W. Wong, M. Stamp, Hunting for metamorphic engines, Journal in Computer Virology 2 (2006) 211–229. doi:10.1007/s11416-006-0028-7.

[14] S. Harris, A. Harper, C. Eagle, J. Ness, Gray Hat Hacking, Second Edition, McGraw-Hill Professional, 2008.