The People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
University Abbes Laghrour of Khenchela
Faculty of Science And Technology
Department of Mathematical And Computer Sciences

Thesis

Submitted in Fulfillment of The Requirements For The Degree of Master in Computer Sciences

Option: Software and Distributed Systems Engineering

Integration of Algebra of Communicating Processes in the Design of Fog Systems

Defined By

LAACISSE AFAF

Directed by Dr. MAAROUK TAOUFIK

Jun/10/2023

contents

Abstract

Title: Integration of Algebra of Communicating Processes in the Design of Fog Systems

The rapid growth of fog computing, coupled with the increasing complexity of fog systems, poses significant challenges for their design and analysis. This study explores the integration of Algebra of Communicating Processes (ACP) as a formal modeling and analysis tool for fog systems. The aim is to leverage ACP's algebraic and axiomatic approach to reason about the behavior of fog systems in the context of cloud computing, IoT, edge computing, multi-agent systems, and blockchain technology. The design of fog systems requires understanding the behavior of various components and their interactions. ACP offers a formal framework to model and analyze the behavior of fog nodes, cloud servers, IoT devices, agents, and blockchain components. By abstracting the real behavior of these entities, ACP enables the focus on specific aspects of their interactions, such as communication patterns, concurrency, and synchronization. Using ACP, fog computing architectures can be effectively modeled and analyzed. The processes in ACP represent the different entities in the fog system, while the communication operators capture their interactions. IoT device communication, multi-agent system coordination, fog-cloud interactions, and blockchain-based fog systems can all benefit from the integration of ACP. This integration facilitates the verification of properties like performance, reliability, scalability, synchronization, deadlock avoidance, and fault tolerance. Through the application of ACP, designers and researchers can gain insights into the behavior of fog systems and identify potential issues before implementation. Moreover, the formal nature of ACP allows for rigorous analysis and verification, ensuring the reliability and efficiency of fog systems in various application domains. This research contributes to the advancement of fog computing by providing a formal methodology for system design and evaluation.

Keywords: Cloud, Fog computing, IoT, Edge computing, SPIN, formal verification, Algebra of Communicating Processes (ACP)

ملخص:

العنوان: دمج جبر عمليات الاتصال في تصميم أنظمة الضباب

يشكل النمو السريع لحوسبة الضباب، إلى جانب التعقيد المتزايد لأنظمة الضباب ، تحديات كبيرة لتصميمها وتحليلها. تستكشف هذه الدراسة تكامل جبر عمليات الاتصال (أكب) كأداة رسمية للنمذجة والتحليل لأنظمة الضباب. والهدف من ذلك هو الاستفادة من نهج أكب الجبري والبديهي للعقل حول سلوك أنظمة الضباب في سياق الحوسبة السحابية، وإنترنت الأشياء ، والحوسبة الحافة ، وأنظمة متعددة العوامل ، وتكنولوجيا بلوكشين. يتطلب تصميم أنظمة الضباب فهم سلوك المكونات المختلفة وتفاعلاتها. تقدم أكب إطارا رسميا لنمذجة وتحليل سلوك عقد الضباب والخوادم السحابية وأجهزة إنترنت الأشياء والوكلاء ومكونات بلوكتشين. من خلال استخلاص السلوك الحقيقي لهذه الكيانات ، تمكن أكب التركيز على جوانب محددة من تفاعلاتها ، مثل أنماط الاتصال والتزامن والتزامن. باستخدام أكب ، أبنية الحوسبة الضباب يمكن أن تكون على غرار فعال وتحليلها. تمثل العمليات في مجموعة دول أفريقيا والبحر الكاريبي والمحيط الهادئ الكيانات المختلفة في نظام الضباب ، بينما يلتقط مشغلو الاتصالات تفاعلاتهم. يمكن أن تستفيد اتصالات أجهزة إنترنت الأشياء ، وتنسيق النظام متعدد العوامل ، والتفاعلات بين الضباب والسحابة ، وأنظمة الضباب القائمة على بلوكتشين من تكامل أكب. يسهل هذا التكامل التحقق من خصائص مثل الأداء والموثوقية وقابلية التوسع والمزامنة وتجنب الجمود والتسامح مع الخطأ. من خلال تطبيق أكب ، يمكن للمصممين والباحثين الحصول على نظرة ثاقبة في سلوك أنظمة الضباب وتحديد القضايا المحتملة قبل التنفيذ. وعلاوة على ذلك ، تسمح الطبيعة الرسمية لمجموعة دول أفريقيا والكاريبي والمحيط الهادئ بإجراء تحليل دقيق والتحقق ، مما يضمن موثوقية وكفاءة أنظمة الضباب في مختلف مجالات التطبيق. يساهم هذا البحث في تقدم حوسبة الضباب من خلال توفير منهجية رسمية لتصميم النظام وتقييمه.

الكلمات المفتاحية: سحابة ، والحوسبة الضباب ، تقنيات عمليات ، والحوسبة الحافة ، وتدور ، والتحقق الرسمي ، والجبر من التواصل العمليات (أكب(

I. I. Introduction

A. Context of the study

This study focuses on integrating the Algebra of Communicating Processes (ACP) into the design of fog systems. Fog computing extends cloud computing to the network edge, enabling efficient data processing. However, designing fog systems poses challenges in scalability, concurrency, resource management, dynamic adaptation, security, and verification.

To tackle these challenges, the study proposes incorporating ACP, an algebraic framework for modeling concurrent systems and their communication behaviors. By utilizing ACP, the study aims to formally specify and analyze interactions among fog nodes. ACP offers operators and axioms for describing process composition, sequencing, and parallelism, facilitating rigorous reasoning about system behavior.

Integrating ACP into fog system design brings several benefits. It enables modeling complex communication patterns, resource management strategies, and dynamic adaptation mechanisms. Moreover, it establishes a formal foundation for ensuring system reliability, adaptability, and security. Leveraging ACP, designers gain insights into fog node interactions and can make informed decisions during system development.

In summary, this study aims to integrate ACP into fog system design, addressing challenges related to scalability, concurrency, resource management, dynamic adaptation, security, and verification. By doing so, it enhances the modeling, analysis, and overall design of fog systems.

B. B. Study problem

The study on the integration of the Algebra of Communicating Processes (ACP) in the design of fog systems focuses on the challenges and research questions related to this integration. The study aims to model the behavior of fog systems using ACP, taking into account scalability, concurrency, resource management, dynamic adaptation, security, and verification. The objective is to enhance the design of fog systems by utilizing ACP to ensure their reliability, adaptability, and security.

- Verification of the design of fog systems

C. C. Objectives of the study

The study on the integration of Algebra of Communicating Processes (ACP) in the design of fog systems aimed to explore the applicability of ACP in modeling the behavior of fog systems. It addressed the challenges of scalability, concurrency, resource management, dynamic adaptation, security, and verification in fog system design.

By integrating ACP, the study developed formal models that accurately captured the concurrent and communication aspects of fog systems. It incorporated resource management strategies and dynamic adaptation mechanisms, allowing for optimal utilization of fog resources and adaptive behavior. Additionally, the study utilized ACP to model security protocols and verification techniques, ensuring the integrity and confidentiality of data in fog environments.

Through simulations, experiments, and case studies, the study evaluated the effectiveness of integrating ACP in fog system design. It assessed the impact of ACP on system performance, reliability, adaptability, and security.

Overall, the study's findings contributed to the improvement of fog system design practices and advanced the understanding of fog computing technology. By leveraging ACP, designers can enhance the modeling, analysis, and overall design of fog systems, leading to more efficient and secure deployments in real-world scenarios.

D. E. Plan of the thesis

- **chapter 1:Fog Computing : context and definitions**
    - **a.** Introduction
    - b. Historical
    - c. Fog copmuting
    - d. Cloud vs fog computing
    - e. Fog Computing Reference Architecture
    - f. Models/Architectures
    - g. Fog Computing Environmental Model
    - h. Conclusion
- **chapter 2:Related work.**
    - a. Introduction
    - b. Modeling of a Generic Edge Computing Application Design
    - c. Fog Scenario
    - d. Spin/Promela Scenario
    - e. Algebraic Modeling of a Generic Fog Scenario for Moving IoT Devices
    - f. Topology Framework :
- **chapter 3: An Algebra of Communicating Processes**
    - **a. Introduction**
    - **b. Preliminary**
        - **Definition of a process and basic actions**
        - **Axiom system PA for interleaving or "free" merge**
        - **Semantics of PA .**
    - **c. Differences between ACP and CCS operators**
    - **d. Conclusion**
- **chapter 4:** case study
    - a. Introduction:

b. Transaction: Access to Data
c. Transaction: Modify Data
d. Transaction: Store Data
e. Specification and validation
f. Specification in ACP
g. Specification in Promela
h. Formal verification
i. Conclusion

Chapter1

E. Fog Computing: context and definitions

1. I.Introduction :

Fog Computing is a distributed computing environment that extends the CloudComputing paradigm to fully support the Internet of Things (IoT) vision. It aimsto push the intelligence, processing and storage of data closer to the Edge of thenetwork to provide computer-related services more immediately and near to theinterconnected smart things that form part of the IoT. For this reason, FogComputing is sometimes also referred to as Edge Computing. The two terms areoften used interchangeably; however, there are subtle differences.Fog Computing appears to be the nextbig thing for the IoT vision.Although the Cloud paradigm is highly attractive for data-intensive processingrequirements, there are a number of inherent limitations that the Fog vision aims toresolve: by reducing the Internet/service latency, increasing the context awareness,conserving the network bandwidth, improving the quality of provision, enhancingthe operational efficiency, providing the improved user experience and speeding upthe real-time processing as well as storing of sensitive data close to where the dataare generated. Fog paradigm also provides better support for mobility and the use ofmobile communication [1].

In brief, whereas in Cloud Computing, response times are in minutes and oftendays, data storage periods are more longer time and often permanent, and locationcoverage is too wide and often global; in Fog Computing, response times are inmilliseconds or sub-seconds, data storage periods are transient, and location coverage is local and with increased awareness. These benefits are mainly due to thelocalization of nodes in the Fog architecture that supports vertically isolatedlatency-sensitive applications by providing ubiquitous, scalable, layered andfederated network connectivity. Whereas the Cloud environments are oftengeographically a long way away from the end-user devices .wider Internet bandwidths, the Fog services are much closer to end users with densegeographical distribution. Fog Computing is currently being successfully deployedin numerous commercial and industrial application scenarios that require both Foglocalization and Cloud globalization, including: smart grids, smart homes and cities,connected vehicles, self-drive cars and trains, traffic light system, healthcaremanagement and numerous other cyber-physical systems.

2. II.Historical

Nicolas Tesla said in 1926: "The day when wireless technology will be perfectly applied. The entire Earth will be converted into a huge brain, which is actually the case. All objectswill be particles of a real and rhythmic whole and the instruments that will allow us toDoing this will be incredibly simple compared to our phone, a manwill be able to have one of these instruments in his pocket." This is how it predates connected object environments

IOT : The first "object" connected to the Internet dates back to 1982: it was a beverage dispenser installed at Carnegie-Mellon University in Pittsburgh, Pennsylvania, which indicated the filling level of the device and the temperature of the drinks.

Cloud : The first person to use the term cloud computing was Professor Ramnath Chellappa of the University of Texas at Austin in 1997.

III.Fog copmuting

As fog computing concept is relatively recent, with its first definition presented in 2012[2].

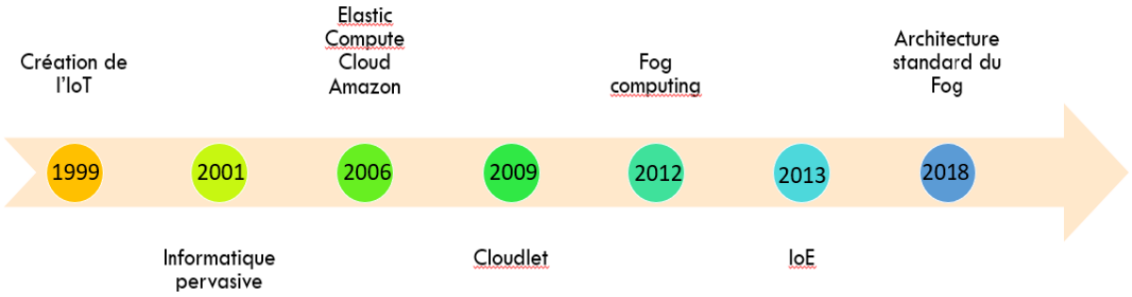


FIGURE 2.1 – Flèche temporelle de l'évolution de l'informatique depuis l'IoT au Fog computing

3. Definitions :

Definition 1 : Fog Computing is the term coined by Cisco that refers to extending cloud computing to an edge of the enterprise's network. Thus, it is also known as Edge Computing or Fogging. It facilitates the operation of computing, storage, and networking services between end devices and computing data centers.

Definition 2 : Fog computing is defined by NIST as 'a layered model for enabling ubiquitous access to a shared continuum of scalable computing resources'. According to NIST, fog computing facilitates the deployment of distributed, latency-aware applications and services, and consists of physical or virtual fog nodes residing between smart end-devices and centralised cloud services.

Definition 3 : Fog Computing is a hyper term that is often used as a substitute to Cloud Computing, though there are differences, as will be illustrated in the discussion below. Fog Computing is a new concept that extends the cloud computing model to the edge of the computing networks, much closer to where the devices, that generate data, exist.

Definition 4 : Fog Computing is defined as a disseminated computing infrastructure in which the applications and services are handled either at the network edge or in a remote data centre cloud.

4. Cloud vs fog computing :

In this subsection, we compare the two models, look into the similarity and differences,and discuss how some of the issues inherent in the Cloud paradigm may be resolved, or at least, minimized by the Fog paradigm. Fog Computing is a distributed computing paradigm that extends Cloud Computing to the edge of the network—as a compliment to the Cloud solution, to adjust to the emerging IoT vision. It facilitates the operation of compute, storage and networking services between end devices and Cloud Computing data centres. Fog Computing typically involves components of an application running both in the Cloud as well as in the Edge devices in the Fog, i.e. in smart gateways, routers or dedicated Fog devices. Refer to Fig 2.2 where the bottom layer has the enterprise smart sensor devices that are accessing resources and compute power from themiddle layer as well as resources and compute power directly from the Cloud; the Fog environment in the middle layer is also linked with the Cloud environment in the top layer.

Fig. 2.2 Three-layer Cloud–Fog enterprise model

Table 2.3 Fog Computing versus Cloud Computing

	Fog Computing	Cloud Computing
Response time	Milliseconds, sub-seconds	Minutes, days, weeks
Data storage period	Transient	Months and years
Applications	e.g. M2M	e.g. Data analytics
Location coverage	Local	Global

Table 2.3 illustrates the benefits of storage and processing 'closer to home' rather than in a geographically distant Cloud environment. To summarize, the Cloud requires a huge amount of bandwidth, the Internet is inherently unreliable, and wireless networks have limitations. By using Fog Computing, the amount of bandwidth required is much reduced. It allows, essentially, transmitted data to bypass the Internet, keeping it as local as possible, on the smart devices in the Fog environment. The most valuable data may still be transmitted through Cloud networks, but much of the traffic, especially the sensitive data, could be kept

off of those networks, freeing up bandwidth for everyone else using the Cloud. Similar to Cloud Computing, Fog Computing provides storage, compute and applications to be consumed by end-users. However, Fog Computing has a bigger proximity to end-users and bigger geographical distribution [3], as mentioned before. Compared to the Cloud paradigm, Fog Computing emphasises proximity toend-users and client objectives, local resource pooling, latency and the backbone bandwidth reduction to achieve better quality of service (QoS) and Edge analytics/ stream mining, resulting in superior user experience [3]. Thus, Fog Computing extends the Cloud model to the edge of the network to address applications and services that do not fit the paradigm of the Cloud due to technical and infrastructure limitation such as the following [4]:

Applications requiring lower and predictable latency,

• Geographically widely distributed applications and processing,

• Faster mobility and mobile applications,

• Large-scale distributed control systems requiring faster processing.

There are certain inherent issues in Cloud Computing, as discussed before. Fog Computing is highly suited to resolving at least some of these, e.g. reducing the need for bandwidth by not sending every bit of information over Cloud channels and instead aggregating it at certain access points [5]. This type of distributed strategy, in turn, also lowers costs, improves efficiencies, and so the QoS. More interestingly, it is another approach to dealing with the emerging and much popular concept of Internet of Things (IoT).

Fog Computing Reference Architecture

In order to develop a Fog Computing architecture, a collaboration by the name of OpenFog Consortium, comprising the joints of industry (such as Cisco, Dell, Intel and Microsoft), research institutions such as Princeton University and users, was founded in November 2015. This consortium is an independent non-profit organization run under the direction of its board of directors; its committees and working groups are run by its membership. Their deliberations have resulted in, what is now called as the Open Fog Reference Architecture (Open Fog RA) that aims to help business leaders, software engineers and system designers

in developing and maintaining hardware, software and system elements necessary for Fog Computing. The OpenFog RA is built upon a set of eight core principles called Pillars, viz:

• **Security**: safety, trust,

•**Scalability**: scalable performance, capacity, reliability, security, hardware, software,

• **Openness**: composability, interoperability, communication, location transparency,

• **Autonomy**: of discovery, orchestration, management, security, operation, costsavings,

• **RAS**: reliability, availability, serviceability,

• **Agility**: innovation, affordability,

• **Hierarchy,**

• **Programmability**: adoptive infrastructure, efficient deployment, effective operations, enhanced security.

In determining the relevant pillars, ISO/IEC/IEEE standards have been followed. Figure 2.4shows the OpenFog layered architectural logical view of the IoT system from a computational perspective. The hierarchical layers are deployed in the Fog–Cloud model as illustrated in Fig. 2.5. In Fig. 2.5, we note that:

• Enterprise Systems (ESs) in the model designated as {1} reside only in the Fog environment and are independent of the Cloud services.

• Model {2} uses the Cloud for ESs needed for business support at strategic level, e.g. strategic decisions making.

• Model {3} suggests that local Fog infrastructure is used for time-sensitive processing, while the Cloud provision is accessed for operational and business support-related information processing.

• The last model, referred to as {4}, is employed in scenarios such as connected cars etc. The discussion in this section, so far, has referred to an abstract logical higher-level architecture as proposed by the OpenFog RA.

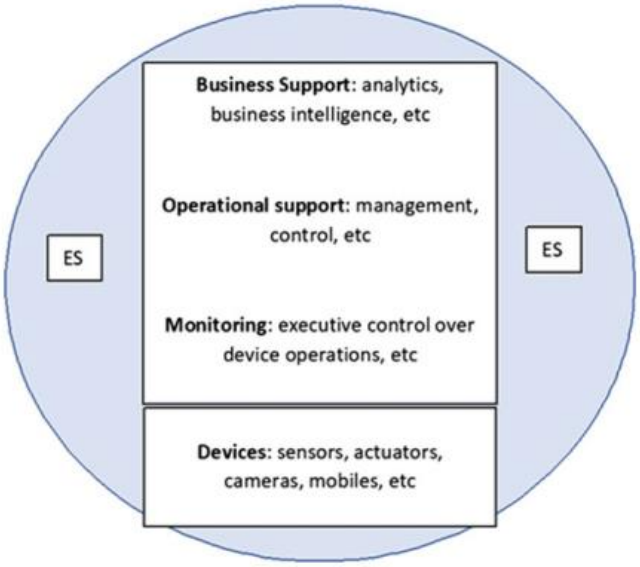


Fig. 2.4 OpenFog layered architectural view—ES refers to enterprise systems for implementation of various systems within the hierarchy

Enterprise systems	Enterprise systems	Enterprise systems	Enterprise systems	Enterprise systems
Business support	Fog .	Cloud .	Cloud .	Cloud .
Operational support	Fog .	Fog .	Cloud .	Cloud .
Monitoring	Fog	Fog	Fog	Cloud
Devices	{1}	{2}	{3}	{4}

Fig. 2.5 OpenFog hierarchy Fog deployment (in Fog–Cloud) models. Adapted from [**7**]

At this point, it is worth noting that the OpenFog RA offers a number of distinct advantages which the OpenFog Consortium refer to as SCALE [6] that are as follows:

- Security: additional security to ensure safe and trusted transactions,
- Cognition: awareness of client-centric objectives to enable autonomy,
- Agility: rapid innovation and affordable scaling under common infrastructure,
- Latency: real-time processing and cyber-physical system control,

• Efficiency: dynamic pooling of resources from participating end-user devices.

5. Models/Architectures :

No standard architectures and tested models have been derived from empirical research. However, some of the Fog Computing architectures have been describedin the following sections for better understanding of the Fog vision.

6. A Generic Fog Computing Architecture

Mind Commerce [7]has outlined a generic Fog Computing architecture that consists of the following elements:

• **IoT** endpoints: which include end devices like sensors, gateways, edge devices, and other physical computing objects that perform the required functions and provide apps that are installed on the end devices. These devices are generally Fog Computing in the IoT Environment connected to a data storage. The IoT endpoints collect the data from other different devices; process some data in real-time in the IP network data processing units. The rest of the data is stored in a Cloud environment for further processing.

• **IP** (Internet Protocol) Network: that provides distributed intelligence and is linked to IoT End-point devices (e.g. sensors) to receive data from and also to the Cloud environment to send required data for storage purposes and future analysis. Purpose is mainly to provide communication links. the IP networkutilises distributed intelligence for supporting intelligence representation e.g. big data mining etc. Distributed Intelligence is considered as a collective intelligence that provides an effective theoretical framework for understanding what humans can achieve and how artefacts, tools, and socio-technical environments can be designed and evaluated to empower human beings.

• **Centralised Control and Mediation Unit**: which is a cloud-based unit in the Cloud environment where the remaining data, which was not processed in the data processing unit, is stored and analysed, as required. This unit is used for storage, queries, and business analytics.

7. Fog Computing Environmental Model

Zhu et al. [8], in their seminal research paper “Improving website performance using edge servers in Fog Computing architecture” structured the Fog Computing Environmental Model. This model has been presented in Fig. 2.6. They have divided the entire Fog Computing environment into three layers as follows:

Centralised Intelligence: Cloud Computing Here, Data Centre Cloud may be employed for services for application hosting, management. The Core Networking and Services are responsible for InternetProtocols (IP), Multiprotocol Label Switching (MPLS), Quality of Service, Multicast, Data Security, Network Services, and Mobile Packet Core functions. Distributed Intelligence: Fog ComputingThis layer is the primary one at the edge level. It may also be called Multi-Service Edge. In this layer, 3G/4G/Long-Term Evolution (LTE), Wi-Fi,

Ethernet, and Programmable Logic Controller (PLC) technologies and other such can be encountered. This is much like the field area network.Distributed Intelligence: End-Point Computing It can alternatively be called the Smart Things Network. This layer consists of several embedding systems, objects and sensors. This indicates that smart and lesssmart things like vehicles and machines are connected via wired and wireless connections.

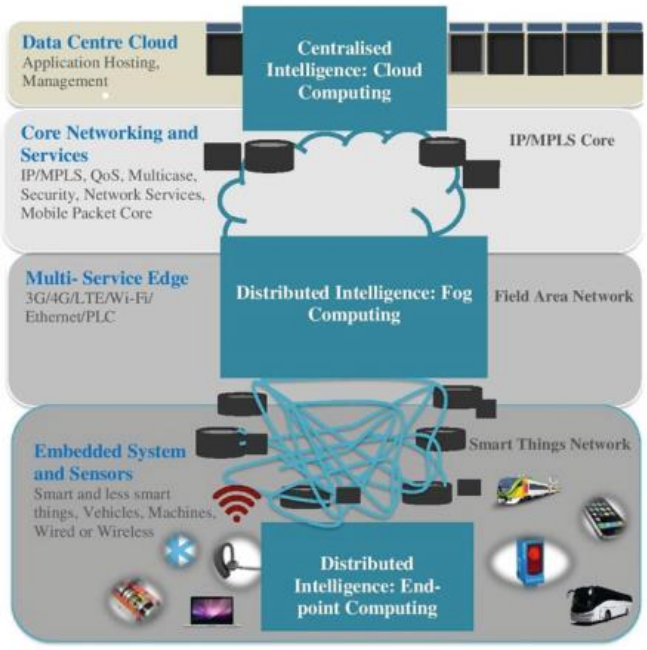


Fig. 2.6 Fog computer architecture (adapted from [10])

F. Conclusion :

Fog Computing-based systems are becoming important in our daily lives. It is amply clear from the above discussions that Fog Computing will, in future, be omnipresent and pervading across all disciplines, social and commercial. Through this, even earthen things are likely to get high calibre computation capabilities,through the inclusion of basic electronic devices and embedded processing capabilities. The beauty of this technology is that all smart devices, including smart phones, already possess this computation technology. The user-oriented applications are already fully loaded in almost all these mobiles. These applications include Google, music, album, movies, camera, calendar, maps, alarms, Facebook, Twitter, navigation, voice speech, Hangouts, voice dialler, WhatsApp, notes, Email and so on. Edge technologies can be used to assist free-view videos by offering virtual view synthesis and related multi-view video incentives. All these applications require local storage and high-speed real-time processing, along with high-speed connected bandwidth to perform edge computing. Even in the case of home automation, smart devices are coming on board with facilities such as computation, residential tracking, sensing and adapting the IoT environments through Ambient Intelligence (AmI), as well as manipulation of databases in real-time situations. All the IT-related companies are developing IoT products and services to interoperable Fog Computing hardware, software, and networking. However, researchers

still have much work to do on the fundamentals of Fogging and its allied applications to develop standard Fogging models and architectures.

Chapter 2

G. An Algebra of Communicating Processes

Introduction*:*

Algebra of Communicating Processes (ACP) is a formal language used to model concurrent and distributed systems. It provides a set of operators to describe the behavior of processes and their communication channels. ACP is widely used in the field of computer science, especially in the design and analysis of software and hardware systems.

Two notable works that have defined ACP are "Algebra of Communicating Processes with Abstraction"[9] and "Process Algebra for Synchronous Communication," both authored by J.A. Bergstra and J.W. Klop. In "Algebra of Communicating Processes with Abstraction," Bergstra and Klop introduced the concept of abstraction to ACP, allowing for the modeling of complex systems with simpler and more manageable representations. This work also established a framework for analyzing the correctness and efficiency of concurrent and distributed systems modeled using ACP.

"Process Algebra for Synchronous Communication,"[10] on the other hand, focuses on the modeling of synchronous communication in ACP. The authors developed a formalism for describing the behavior of synchronous communication channels and provided a set of operators for specifying the interactions between processes that use these channels.

The objective of the authors is to contribute to the theory of concurrency through an algebraic approach. They recognize the importance of understanding the behavior of concurrent systems and processes, particularly with regard to communication. In recent years, various frameworks have been proposed for concurrency, but the authors propose an algebra of processes as a way to systematically study and analyze concurrent and distributed systems.

The authors note that the development of an algebra of processes started with the work of Robin Milner and his calculus of communicating systems (CCS). Milner's goal was to create a calculus with a small number of operators that embody intuitive ideas and provide general expressive power. In particular, he proposed synchronous CCS (SCCS), which is based on four fundamental operators. Milner believed that these four operators could be combined to form a complete calculus, and that the algebraic identities obeyed by these operators could be used as axioms for an algebraic "concurrency" theory, analogous to rings or vector spaces.

Overall, the authors aim to contribute to the development of a systematic and formal approach to the study of concurrent and distributed systems, building on the work of Milner and others. By providing a formal and precise language for describing and analyzing these systems, the authors hope to enable the detection of errors and the verification of system properties, and ultimately to improve the reliability and performance of concurrent and distributed software.

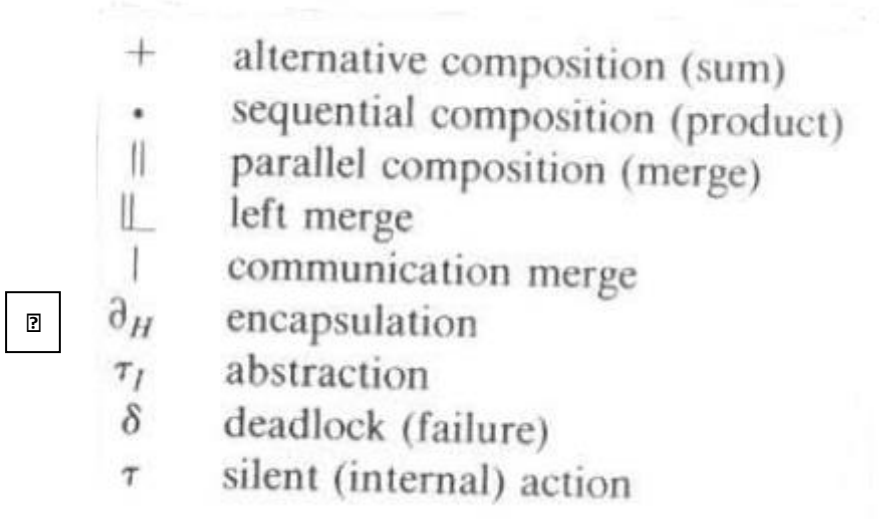
1. Preliminary :

a) Definition of a process and basic actions

Process algebra, the basic elements are processes, and the operations define how processes can be composed or transformed. The operations are defined by a set of axioms or rules, and they can be used to reason about the behavior of processes. Process algebra provides a formal, mathematical framework for reasoning about the behavior of concurrent systems. It is used to model and analyze systems that involve multiple processes that interact with each other. This includes software systems, hardware systems, and systems that involve both software and hardware components [11].

The basic operations of process algebra include composition, parallel composition, hiding, and renaming. Composition allows two processes to be combined into a single process, while parallel composition allows multiple processes to be executed concurrently. Hiding allows certain actions to be hidden from view, while renaming allows actions to be renamed [11].

Definition of the operators +, •, and]] for alternative, sequential, and parallel composition, respectively :



This passage discusses how certain operators in algebraic process calculi (APC), specifically the ACP system, relate to those in the calculus of communicating systems (CCS). The operators +, II, and π have the same meaning in both systems, while multiplication (*) is more general in ACP than in CCS. The operators IL and I are new, and the & operator is similar to NIL in sums but not in products. The operators and π are new but are formally renaming operators in the sense of CCS.

The use of these operators allows for a finite initial algebra specification of the behavior of finite processes. While CCS has hidden operators involved in this specification, ACP includes

IL and I as meaningful operators from an intuitive perspective. The presentation of ACP involves several smaller specifications (PA, PA, ACP) that only involve a subset of the operators.

ACP uses an axiomatic approach, where a set of axioms is given first, and models are investigated next. This is in contrast to CCS, which starts with a model of processes and derives identities in that model as theorems. Through investigations of ACP, the authors have encountered around twenty interesting process algebras, and organizing them as models of an axiomatic theory seems like a sensible approach.

b) Axiom system PA for interleaving or "free" merge

(1) The signature of PA consists of the following ingredients:

$x+y = y+x$	A1
$x+(y+z) = (x+y)+z$	A2
$x+x = x$	A3
$(x+y)\cdot z = x\cdot z+y\cdot z$	A4
$(x\cdot y)\cdot z = x\cdot(y\cdot z)$	A5
$x\|y = x\lfloor y+y\lfloor x$	M1
$a\lfloor x = a\cdot x$	M2
$ax\lfloor y = a(x\|y)$	M3
$(x+y)\lfloor z = x\lfloor z+y\lfloor z$	M4

1 - **a ,b ,c, ... €A** , the set of axiomatic actions (also called **'steps'** or **'events'**). **A** is also referred to as the alphabet. Throughout this paper, we will assume that **A** is finite. (This is done to safeguard the algebraic nature of our considerations — e.g. infinite sums of processes are not considered here.) In the axioms of **PA** , 'a' varies over **A** .

2- x ,y ,z , ... are variables, ranging over the domains of processes (process algebras) which will be con-structed below.

3- binary operators These are:

- + alternative composition, or sum
- . sequential composition, or product
- II parallel composition, or merge
- IL left-merge**.**

The 'main' operators are +,., II. Left-merge IL is an auxiliary operator.

Process expressions or process terms are built from the **a €A** by means of +,., II, IL . Examples of process expressions are:

(a +b).((((a .a) IL b) + (c .d)).e).

The following notational conventions will be employed: xy stands for x .y; outermost brackets are omitted the operator . has the greatest binding power**; x"** stands for **xx…… x** (n times); II and IL hind stronger than +,. So the two process expressions above may be written as

a + b, (a^2 IL b + cd)e.

c) Semantics of PA .

A process algebra is a domain of processes satisfying the axioms of **PA** . The three most important process algebras for **PA** are:

(1) **Aɯ** the initial algebra of **PA**,

(2) **A∞** the graph model of **PA**,

(3) A∞ the standard model of **PA**

It will turn out that these algebras properly extend each other (modulo isomorphism): **Aɯ ¢ A∞ ¢**A∞

d) Algebra of Communicating Processes :

In ACP, communication between processes is represented by the communication merge operator, denoted by the symbol "[". The communication merge operator allows two processes to synchronize on a common communication event. For example, if process A sends a message on a channel, and process B receives the message on the same channel, then we can represent this communication event using the communication merge operator as follows:

A = send(message, channel)

B = receive(message, channel)

C = A [B

The process C represents the synchronization of A and B on the communication event represented by the channel and the message.

Deadlock is a common problem in concurrent systems where processes become blocked and unable to proceed due to circular dependencies. In ACP, deadlock is represented by the deadlock constant 6. The deadlock constant represents a process that is unable to communicate with any other process, and thus is stuck in a blocked state.

To prevent deadlock in ACP, it is important to ensure that processes can always communicate with each other. This can be achieved by careful design of the communication events and by using synchronization primitives such as semaphores and monitors.

In summary, the introduction of the deadlock constant 6 and the communication merge operator [are important features of the Algebra of Communicating Processes. These features allow us to reason about concurrent and distributed systems and to prevent common problems such as deadlock.

e) Introduction of operator c~ n for encapsulation of a process Algebra of Communicating Processes :

The operator "c~ n" is used in the Algebra of Communicating Processes (ACP) as a way to encapsulate a process and make it reusable.

In ACP, processes are defined as sets of actions that can be performed concurrently, and communication between processes is achieved through the synchronization of these actions. The "c~ n" operator allows you to encapsulate a process into a new process, which can then be used as a single entity in the definition of other processes. The syntax for the "c~ n" operator is as follows:

c~ n(P)

where "P" is the process being encapsulated, and "n" is an arbitrary name or identifier for the encapsulated process. The operator creates a new process that behaves exactly like "P", but is now referred to as "n" in the context of other processes.

For example, suppose we have a process "P" that represents a simple calculator:

P = {+,-,*,/}

This process can be encapsulated using the "c~" operator as follows:

c~ calc(P)

Now, "calc" is a new process that behaves like "P" and can be used as a single entity in the definition of other processes. For instance, we can define a new process "Q" that uses "calc" to perform some calculations:

Q = calc(+;*).calc(-;/)

Here, "Q" represents a process that performs addition and multiplication using the "calc" process, and then performs subtraction and division using another instance of "calc".

In summary, the "c~ n" operator in ACP is a useful tool for encapsulating processes and making them reusable. By creating new processes with unique identifiers, you can build complex systems out of smaller, simpler components.

2. Differences between ACP and CCS operators :

ACP (Algebra of Communicating Processes) and CCS (Calculus of Communicating Systems) are two process calculi that use different sets of operators to define and manipulate concurrent processes. Here are some of the main differences between the operators used in ACP and CCS:

1. Basic constructs: In ACP, the basic construct is a process, which is defined as a set of actions that can be performed concurrently. In CCS, the basic construct is an agent, which is defined as a set of actions that can be performed concurrently, as well as a set of actions that can be performed independently.
2. Composition operators: ACP and CCS both have operators for composing processes/agents in different ways. However, the operators used in ACP and CCS are different. ACP has operators such as sequence (.), parallel composition (||), and choice (+), while CCS has operators such as parallel composition (|), restriction (!), and relabelling ([]).
3. Communication operators: ACP and CCS both have operators for communication between processes/agents. However, the communication operators used in ACP and CCS are also different. ACP has operators such as input (a?), output (a!), and replication (!P), while CCS has operators such as input (a?), output (a!), and synchronous communication (;).
4. Equivalence operators: ACP and CCS both have operators for defining process/agent equivalence. However, the equivalence operators used in ACP and CCS are different. ACP has operators such as bisimulation (~~) and weak bisimulation (~~*), while CCS has operators such as trace equivalence (=) and strong bisimulation (~).

In summary, ACP and CCS use different sets of operators to define and manipulate concurrent processes/agents. While both process calculi have operators for composition, communication, and equivalence, the specific operators used in ACP and CCS are different and reflect the different approaches and goals of the two process calculi.

3. Conclusion

Algebra of Communicating Processes (ACP) is a formal language that is widely used in the field of computer science for modeling concurrent and distributed systems. The introduction discusses two notable works that have defined ACP and highlights the importance of understanding the behavior of concurrent systems and processes, particularly with regard to communication. The authors aim to contribute to the development of a systematic and formal approach to the study of these systems, building on the work of Robin Milner and others. By providing a formal and precise language for describing and analyzing concurrent and distributed systems, ACP enables the detection of errors and the verification of system properties, ultimately improving the reliability and performance of concurrent and distributed software.

Chapter 3

Related work.

4. Introduction

In this chapter, we will review two related research studies on edge and fog computing. The first study is titled "Modeling of a Generic Edge Computing Application Design"[12] and was authored by Pedro Juan Roig, Salvador Alcaraz, Katja Gilly, Cristina Bernad, and Carlos Juiz. The second study is titled "Algebraic Modeling of a Generic Fog Scenario for Moving IoT Devices"[13] and was also authored by Pedro Juan Roig, Salvador Alcaraz, Katja Gilly, and Carlos Juiz. These works provide insights into the design and modeling of edge and fog computing systems and their application in Internet of Things (IoT) scenarios.

We will begin by summarizing the main contributions and methodology of each study, followed by a discussion of their similarities and differences. We will then highlight the implications of these works for our own research on edge and fog computing. Finally, we will identify gaps in these studies and explain how our work addresses these gaps to advance the state-of-the-art in edge and fog computing research.

5. Modeling of a Generic Edge Computing Application Design

The first study we will review is titled "Modeling of a Generic Edge Computing Application Design" and was authored by Pedro Juan Roig, Salvador Alcaraz, Katja Gilly, Cristina Bernad, and Carlos Juiz. The paper addresses the challenge of designing edge computing applications that can effectively utilize the resources available at the edge of the network. The authors propose a generic application design model that can be adapted to various edge computing scenarios. They present a methodology for modeling the application components and their interactions, as well as a set of metrics for evaluating the performance of the designed applications.

The paper introduces a generic framework for edge computing, accommodating various sensors, actuators, and network topologies. It emphasizes the publisher/subscriber paradigm using brokers for efficient IoT communication. The block diagram illustrates edge servers as brokers, publishers as sensor-connected end devices, and subscribers as actuator-connected end devices. Sensor data is published to edge servers, which process and forward it to subscribers. If processing fails, data can be sent to the cloud. In fog environments, fog nodes can be inserted between edge servers and the cloud. The behavior of system components is modeled using Abstract Untimed Process Algebra (ACP), capturing relationships and atomic actions. The paper presents ACP models for edge and fog scenarios, incorporating Edge AI with internal CNN functions. Models are verified using Spin and Promela code, along with MSCs. The work provides insights into edge computing dynamics, aiding the analysis of concurrent processes in IoT systems.

6. Fog Scenario :

In Figure 1, the scenario is depicted, which includes five different types of entities: publishers (represented by PUBi), edge servers (represented by EDGEm), subscribers (represented by SUBj), cloud premises (represented by CLOUD), and fog servers (represented by FOGn). This diagram is similar to the edge computing case, with the main differences being the channels connected to and from the fog block and the introduction of CNN for aggregated processing at the fog level.

It seems like you are describing a model or framework related to processing and forwarding data in a hierarchical system, possibly within the context of edge computing. From the information you provided, it appears that there are various entities involved, such as publishers (PUBi), end devices (EDGEm), fog nodes (FOG), cloud (CLOUD), and subscribers (SUBj). Equations (9) to (13) seem to represent the models or relationships for these entities.

Based on the comparisons you mentioned, it seems that Equation (9) is analogous to Equation (1) in some way, Equation (10) corresponds to Equation (2), and Equation (13) relates to Equation (4). Similarly, Equation (11) is similar to Equation (10), but with different data and channels involved. Equation (12) is also similar to Equation (3) but with its own unique characteristics in terms of data and channels.

In this framework, it seems that aggregated processing is performed at the edge level using a CNN (Convolutional Neural Network). The raw data is received from the end devices, and unitary processing is performed at the end devices and the subscribers. The external channels and internal channels within the edge level remain unchanged.

The processed data is forwarded down the hierarchy to reach the appropriate actuator, while aggregated data is forwarded up the hierarchy to a more powerful entity for further processing. Each entity, such as PUBi, EDGEm, FOG, CLOUD, and SUBj, likely has its own role and responsibilities within this hierarchical system.

Please note that without more context or a specific question, it is difficult to provide a more detailed explanation or analysis of the described model. If you have any specific questions or need further clarification, feel free to ask.

$$PUB_i = \left(r_{IN_{p_i}}(d_{p_i}) \cdot \phi(d_{p_i}) \cdot s_{A_i}(d_{p_i}) \right) \cdot PUB_i \quad (9)$$

$$EDGE_m = \Big(r_{A_i}(d_{p_i}) \cdot \Big(s_{B_j}(e_{q_j}) \triangleleft \theta(d_{1_m} \cdots d_{max_m}) \triangleright s_{C_m}(d_{1_m} \cdots d_{max_m}) \Big) + r_{D_m}(e_{q_j}) \cdot s_{B_j}(e_{q_j}) \Big) \cdot EDGE_m \quad (10)$$

$$FOG = \Big(r_{C_m}(d_{1_m} \cdots d_{max_m}) \cdot \Big(s_{D_m}(e_{q_j}) \triangleleft \chi(\bigcup_m \{d_{1_m} \cdots d_{max_m}\}) \triangleright s_{E_n}(\bigcup_m \{d_{1_m} \cdots d_{max_m}\}) \Big) + r_{F_n}(e_{q_j}) \cdot s_{D_m}(e_{q_j}) \Big) \cdot FOG_n \quad (11)$$

$$CLOUD = \Big(r_{E_n}(\bigcup_m \{d_{1_m} \cdots d_{max_m}\}) \cdot \psi(\bigvee_n \{\bigcup_m \{d_{1_m} \cdots d_{max_m}\}_n\}) \cdot s_{F_n}(e_{q_j}) \Big) \cdot CLOUD \quad (12)$$

$$SUB_j = \left(r_{B_{q_j}}(e_{q_j}) \cdot \phi(e_{q_j}) \cdot s_{OUT_{q_j}}(e_{q_j}) \right) \cdot SUB_j \quad (13)$$

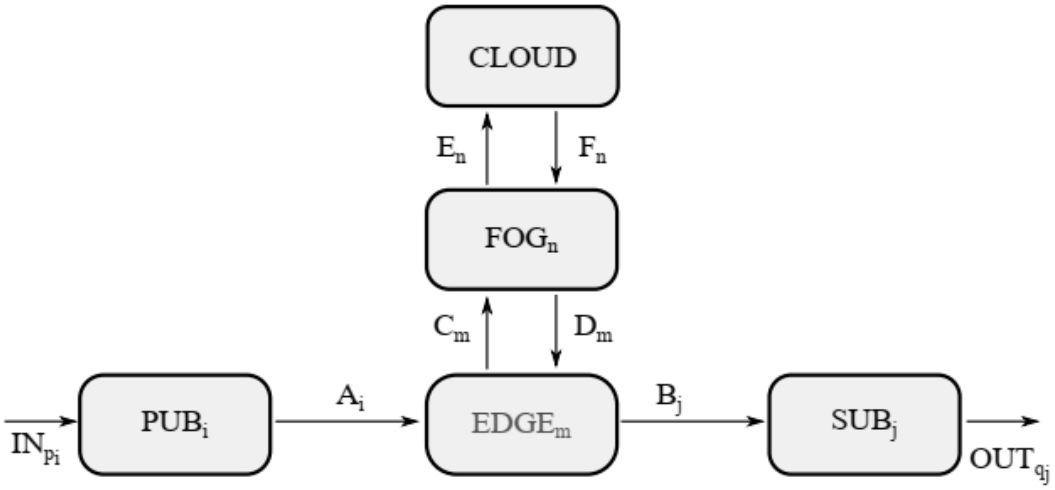


Figure 1 : Model for fog computing

7. Spin/Promela Scenario :

The code starts with defining constants N and INF and the message type MSG.

Communication channels are declared using the chan keyword. The channels include fromSensor, toActuator, Fog2Edge, Edge2Fog, Fog2Cloud, and Cloud2Fog. Each channel is defined with a specific number of message slots (in this case, [14]) and the message structure (mtype, byte, byte) representing the message type and two additional byte parameters.

The code defines four proctype processes: Devices, Edge, Fog, and Cloud. Each process is responsible for executing a specific behavior.

The Devices process simulates the behavior of devices. It receives messages from the fromSensor channel and sends messages to the toActuator channel.

The Edge process simulates the behavior of edge servers. It receives messages from the fromSensor channel and sends messages either to the toActuator channel or the Edge2Fog channel, depending on certain conditions.

The Fog process simulates the behavior of fog nodes. It receives messages from the Edge2Fog channel and sends messages either to the Fog2Edge channel or the Fog2Cloud channel, depending on certain conditions. It also receives messages from the Cloud2Fog channel and sends messages to the Edge2Fog channel.

The Cloud process simulates the behavior of the cloud. It receives messages from the Fog2Cloud channels and sends messages to the Cloud2Fog channels based on certain conditions.

Finally, in the init block, the processes are instantiated and executed using the run keyword.

Algorithm 1 Fog model coded in Promela

```
#define N 2
#define INF 99
mtype = MSG
chan fromSensor[N*N] = [1] of mtype,byte,byte
chan toActuator[N*N] = [1] of mtype,byte,byte
chan Fog2Edge[N*N] = [1] of mtype,byte,byte
chan Edge2Fog[N*N] = [1] of mtype,byte,byte
chan Fog2Cloud[N] = [1] of mtype,byte,byte
chan Cloud2Fog[N] = [1] of mtype,byte,byte
proctype Devices (byte id) {
byte x,y,n=0;
do
:: n<1 -> fromSensor[id] ! MSG(id,INF); n++
:: toActuator[id] ? MSG(x,y)
od
}
proctype Edge (byte id) {
byte x,y;
do
:: if
:: fromSensor[id] ? MSG(x,y) -> if
:: toActuator[id] ! MSG(x,id)
:: Edge2Fog[id] ! MSG(x,y)
fi
```

```
:: Fog2Edge[id] ? MSG(x,y)->toActuator[id] ! MSG(x,y)
fi
od
}
proctype Fog (byte id) {
byte x,y;
do
:: Edge2Fog[id*2] ? MSG(x,y) -> if
:: Fog2Edge[id*2+1] ! MSG(x,id*2+1)
:: Fog2Cloud[id] ! MSG(x,y)
fi
:: Edge2Fog[id*2+1] ? MSG(x,y) -> if
:: Fog2Edge[id*2] ! MSG(x,id*2)
:: Fog2Cloud[id] ! MSG(x,y)
fi
:: Cloud2Fog[id] ? MSG(x,y) -> Fog2Edge[y] ! MSG(x,y)
od
}
proctype Cloud (byte id) {
byte x,y;
do
::Fog2Cloud[0] ? MSG(x,y) -> select(y:N..N+1) -> Cloud2Fog[1]
! MSG(x,y)
::Fog2Cloud[1] ? MSG(x,y) -> select(y:0..1)-> Cloud2Fog[0] !
MSG(x,y)
od
}
init {
byte i;
for (i : 0..(N*N-1))
run Devices (i)
run Edge(i)
for (i : 0..(N-1))
run Fog (i)
run Cloud(0)
}
```

8. Algebraic Modeling of a Generic Fog Scenario for Moving IoT Devices

The second research study we will review is titled "Algebraic Modelling of a Generic Fog Scenario for Moving IoT Devices" and was authored by Pedro Juan Roig, Salvador Alcaraz, Katja Gilly, and Carlos Juiz. The paper addresses the challenge of optimizing the resource utilization of moving IoT devices in fog computing scenarios. The authors propose an algebraic modeling approach that captures the dynamic nature of the devices and their interactions with the fog resources.

The study presents a generic fog scenario model that can be adapted to various IoT applications and environments. The authors also provide a set of algebraic equations that describe the resource allocation process in the fog computing environment. The proposed approach considers various factors, such as the available resources, the location of the devices, and the communication latency, to optimize the resource allocation process.

The paper includes a case study in which the proposed model is applied to a smart transportation scenario. The results show that the proposed approach can effectively optimize the resource utilization and improve the performance of the system compared to existing approaches.

Overall, this work contributes to the development of a systematic approach for optimizing the resource utilization of moving IoT devices in fog computing scenarios and provides insights into the potential benefits of utilizing algebraic modeling techniques in IoT applications.

Atomic actions are basic and indivisible actions that cannot be further broken down into smaller steps. Examples of atomic actions in a computer system include:

Reading and writing data to a file

Sending and receiving messages over a network

Locking and unlocking a resource

Starting and stopping a process or program

Allocating and deallocating memory

Reading and writing data to a database

Opening and closing a file or database connection

Performing arithmetic or logical operations on data.

9. Topology Framework :

The framework selected for the Fog computing environment for moving IoT devices within a wireless domain with the Cloud as a backup solution is composed of 5 layers:

The wireless layer as the lower one: The wireless layer serves as a connection point for moving IoT devices to access their associated virtual machines through the Fog infrastructure.

Figure 2 :

The orchestration layer as the most important one : The orchestration layer is responsible for managing the system's behavior, selecting hosts for VM migration, creating and terminating VMs, and interacting with the Cloud for backup purposes.

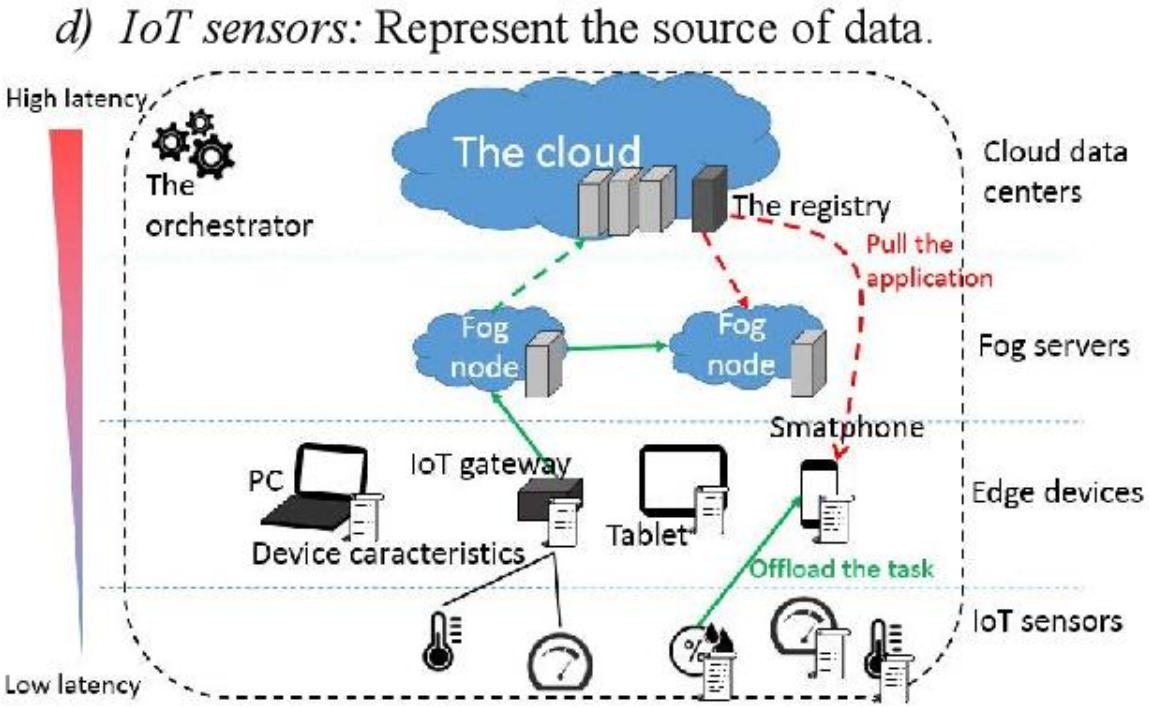


Figure 3 : The proposed multi-tier architecture. [15]

The virtualization layer for managing VMs : The virtualization layer abstracts hardware resources to share them efficiently across multiple VMs and manages the creation, deployment, and resource allocation of VMs.

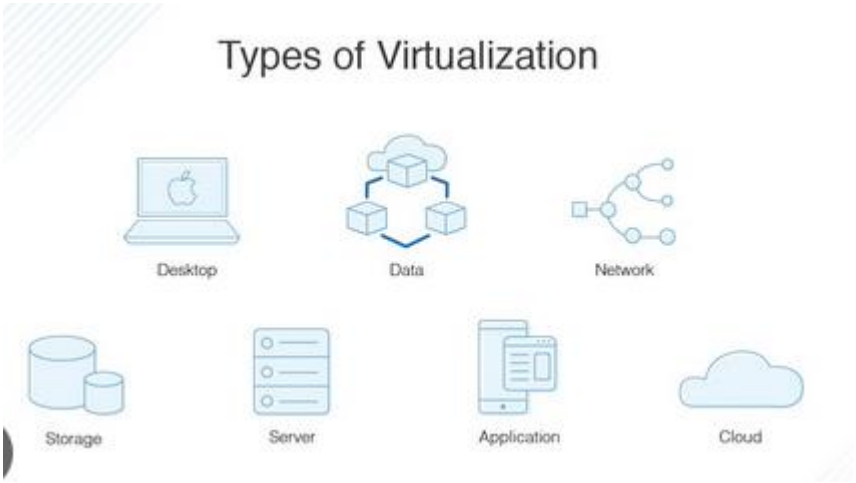


Figure 4 : Types of virtualization

The Fog node layer for allocating resources to VMs : The Fog node layer consists of Fog nodes that support IoT applications, manage resource allocation, load balancing, and fault tolerance to ensure optimal and efficient operation.

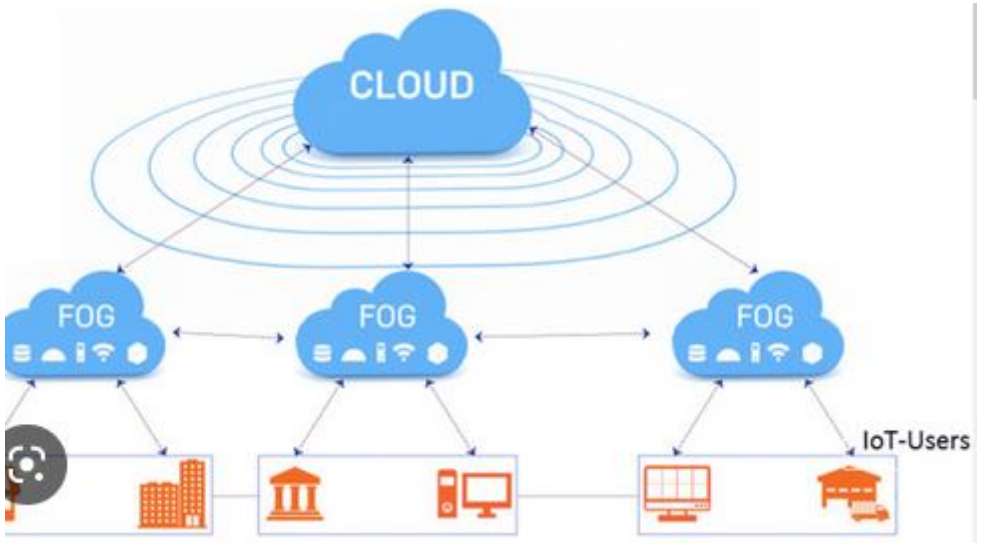


Figure 5 : the Fog node[16]

The Cloud layer for providing backup support : The Cloud layer serves as a backup solution for the Fog environment, ensuring data availability and accessibility in the event of a failur ordisruption, thus enabling system recovery.

The different layers of the Fog infrastructure, including :the host layer, topology layer, cloud layer.

As well as the communication channels between them, with the orchestrator making decisions about VM migration and load balancing policies applied where possible.

Table 1 displays the communication channels used in the model where the orchestrator is the central point making all decisions, starting from the wireless layer, planning at the

orchestrator layer, and executing at the other layers.as wellas to internal channels between layers, two external wireless physical channels are defined: "in" for IoT devices to connect to the Fog domain through wireless relays and "out" for them to disconnect and leave the domain.

Table 1 Communication channels

Channel ID	Direction	Meaning
Wi Q	Incoming	Channel from the wireless relay *Wi* to orchestrator *Q*
QWi	Outgoing	Channel from orchestrator *Q* to the wireless relay *Wi*
QHj	Incoming	Channel from orchestrator *Q* to the host *Hj*
Hj Q	Outgoing	Channel from the host *Hj* to orchestrator *Q*
Hj T	Incoming	Channel from the host *Hj* to Switching topology *T*
T Hj	Outgoing	Channel from Switching topology *T* to the host *Hj*
QC	Incoming	Channel from orchestrator *Q* to the Cloud *C*
C Q	Outgoing	Channel from the Cloud *C* to orchestrator *Q*
in	Inwards	Wireless physical channel coming into wireless relay *Wi*
out	Outwards	Wireless physical channel going out of wireless relay *Wi*

Actions to be Modelled : the six actions required as exhibited in Figure1 at the wireless layer in a Fog computing environment, which are triggered when a moving IoT device comes in or goes out of a wireless relay. These actions are control actions that belong to the control plane, as opposed to the data plane or management plane. The six actions are divided into two categories: actions required when a device enters the Fog domain and actions required when it leaves. The orchestrator is the central element in the system, making decisions that are thenexecuted by the other layers. The communication channels used in the system are also described, with the orchestrator being the main point of communication.

Action Cloud IN :

The modelling of action 1 can be expressed as follows:

ctr1: Select(j) → MoveOut(km) → CloudIn(jm) → MoveIn(jm) → Unbind(km) → Bind(jm)

Where:

Select(j) represents the selection of a suitable host candidate j by the orchestrator Q.

MoveOut(km) denotes the initiation of the migration process of the associated VM km from the Cloud environment to the Fog environment, where the selected host j is located.

CloudIn(jm) represents the migration process from the Cloud environment to the selected host j, resulting in the VM being identified as jm in the Fog environment.

MoveIn(jm) denotes the completion of the migration process from the Cloud environment to the selected host j in the Fog environment.

Unbind(km) represents the removal of the old identifier of the associated VM km in the Cloud environment.

Bind(jm) denotes the association of the new identifier jm of the VM with the selected host j in the Fog environment.

This sequence of actions describes the migration of a VM associated with an IoT device from the Cloud to a host in the Fog environment. The process involves selecting a suitable host, migrating the VM from the Cloud to the selected host, and updating the identifier of the VM to reflect its new location.

$$Wi = (rin(IoT(km)) \cdot sWi\ Q(ctr1(km)) \cdot rQWi(jm) \cdot Unbind(IoT(km)) \cdot Bind(IoT(jm))) \cdot Wi$$
$$Q = (rWi\ (Qctr1(km)) \cdot Select(km) \cdot sQC(ctr1(km, jm)) \cdot rC\ Q(CloudIn(km, jm)) \cdot sQHj\ (CloudIn(km, jm)) \cdot rHj\ Q(ACK, jm) \cdot sQWi\ (jm)) \cdot Q$$
$$C = (rC\ Q(ctr1(km, jm)) \cdot MoveOut(km, jm) \cdot sC\ Q(CloudIn(km, jm))) \cdot C$$
$$Hj = (rQHj(\ CloudIn(km, jm)) \cdot MoveIn(km, jm) \cdot sHj\ Q(ACK, jm)) \cdot Hj$$

The sequence of events in a timely manner, unchained by the moving IoT device getting into the Fog environment and asking for a Cloud IN action can be obtained by applying the encapsulation operator as follows:

[ctr1: Select(j) → MoveOut(km) → CloudIn(jm) → MoveIn(jm) → Unbind(km) → Bind(jm)]

encapsulated with "IoT device enters Fog environment and requests Cloud IN action"

This can be interpreted as the sequence of events that occur when an IoT device associated with a VM in the Cloud environment enters the Fog environment and requests a Cloud IN action. The encapsulation operator combines the individual actions into a single sequence of events, which occur concurrently as the IoT device enters the Fog environment and requests the Cloud IN action.

$$\partial H(Wi \,||\, Q \,||\, C \,||\, Hj = (\, rin(IoT(km) \cdot cWi\, Q(ctr1(km) \cdot Select(km) \cdot cQC(ctr1(km, jm)) \cdot$$

$$MoveOut(km, jm) \cdot cC\, Q(CloudIn(km, jm)) \cdot cQHj\, (CloudIn(km, jm)) \cdot$$

$$MoveIn(km, jm) \cdot cHj\, Q(ACK, jm) \cdot cQWi(\ jm) \cdot Unbind(IoT(km)\ ) \cdot$$

$$Bind(IoT(\ jm))) \cdot \partial H(Wi \,||\, Q \,||\, C \,||\, Hj)$$

The abstraction operator can be applied to mask internal processes and communications, preserving only the external behavior of the model, including all layers regardless of their involvement in the specific action, as it does not impact the overall outcome.

$$\tau I(\, \partial H(Wi \,||\, Q \,||\, C \,||\, Hj \,||\, T)) = rinIoT(km)\ ) \cdot \tau I\, \partial HWi \,||\, Q \,||\, C \,||\, Hj \,||\, T\ ))$$

Action Migrate IN :

The modelling of action 2 involves the migration of a VM from a current host to a closer host when a moving IoT device connects to a wireless relay Wi, with the orchestrator determining whether a closer host is available and sending a control message to either migrate the VM or acknowledge that no migration is needed.

$$Wi = (rin(IoT(\ jm)\ ) \cdot sWi\, Q(ctr2(\ jm)\ ) \cdot (rQWi\ \ (jm)\ \cdot Unbind(IoT(\ jm)\ ) \cdot Bind(IoT(\ jm))$$
$$+\ rQWi\ (ACK))) \cdot Wi$$

$$Q = (rWi\, Q(ctr2(\ jm)\ ) \cdot (sQHj\ ctr2(\ jm, jm)) \cdot rHjQ(ACK, jm\ ) \cdot sQWi\ (jm\ )$$
$$Select(\ jm) sQWi(ACK) \cdot Q$$

$$Hj = rQHj\ ctr2(\ jm, jm) \cdot MoveOut(\ jm, jm) \cdot sHj\ T\ MigrateOut(\ jm, jm\ \cdot Hj$$

$$Hj = rT\ HjMigrateIn(\ jm, jm) \cdot MoveIn(\ jm, jm) \cdot sHjQACK, jm \cdot Hj$$

$$T = rHj\ T\ MigrateOut(\ jm, jm) \cdot Networking(\ jm, jm) \cdot sT\ HjMigrateIn(\ jm, jm\ \cdot T$$

Here, the processes are happening concurrently. The first part represents the entry of the IoT device into the Fog environment and its request for a Migrate IN action. The second part represents the sequence of actions involved in the migration process, including the selection of a suitable host j', the migration of the VM from the current host j to the new host j', the

establishment of networking connections, and communication between Host j and the IoT device.

By applying the encapsulation operator, the sequence of events captures the concurrent nature of the processes triggered by the moving IoT device's entry into the Fog and its request for a Migrate IN action.

$$\partial H(Wi \,||\, Q \,||\, Hj \,||\, Hj \,||\, T) = (rinIoT(jm)) \cdot cWi\,Q(ctr2(km))$$

$$\cdot(cQHj\,ctr2(jm, jm)) \cdot MoveOut(jm, jm) \cdot cHj\,T(MigrateOut(jm, jm))$$

$$\cdot Networking(jm, jm) \cdot cT\,Hj(MigrateIn(jm, jm)) \cdot MoveIn(jm, jm) \cdot cHjQ(ACK, jm)$$

$$\cdot cQWi(jm) \cdot Unbind(IoT(jm)) \cdot Bind(IoT(jm))\ Select(jm)\ cQWi(ACK)))$$

$$\cdot \partial H(Wi \,||\, Q \,||\, Hj \,||\, Hj \,||\, T)$$

Eventually, abstraction operatormay be applied in order tomask internal processes and internal communications, hence maintaining just the external behaviour of the model, where all layers are to be included, considering that those not participating do not influence the result in any way:

$$\tau I(\partial H(Wi \,||\, Q \,||\, C \,||\, Hj \,||\, Hj \,||\, T)) = rin(IoT(jm)) \cdot \tau I(\partial H(Wi \,||\, Q \,||\, C \,||\, Hj \,||\, Hj \,||\, T))$$

Action Init VM :

Action 3: Create and initialize a VM in the Fog domain

Inputs: IoT device (-), wireless relay Wi

Outputs: Initialized VM identifierIoT device connects to wireless relay Wi without an associated VM (-).

Wi sends a control message ctr3 to the orchestrator Q requesting a host Hj with enough available resources and located closer to Wi.

Upon receiving ctr3, Q processes the message and finds an available host Hj that meets the criteria.

Q sends a control message to Host Hj to initiate the creation and initialization of the VM.

Host Hj processes the message and creates a new VM.

Host Hj initializes the VM with all the required software, configurations, and data.

Host Hj sends an acknowledge message to Q indicating that the VM has been created and initialized successfully.

Q sends a control message to Wi with the identifier of the initialized VM.

Wi processes the message and sends an acknowledge message to the IoT device with the identifier of the initialized VM.

$$Wi = (\ rin(IoT(-)\ ) \cdot sWi\ (Qctr3(-)) \cdot rQWi(\ jm) \cdot BindIoT(\ jm) \cdot Wi$$

$$Q = rWi\ Qctr3(-) \cdot Select(-) \cdot sQHj\ ctr3(\ jm) \cdot rHj\ QACK, jm \cdot sQWi\ jm \cdot Q$$

$$Hj = rQHj\ ctr3(\ jm) \cdot Init(\ jm) \cdot sHj\ QACK, jm \cdot Hj$$

After the application of the encapsulation operator to create a new VM, thus an*Init VM* action:

$$\partial HWi\ ||\ Q\ ||\ Hj = rinIoT(-) \cdot cWi\ Qctr3(-) \cdot Select(-) \cdot cQHj\ ctr3(\ jm)$$
$$\cdot Init(\ jm) \cdot cHj\ QACK, jm \cdot cQWi\ jm \cdot BindIoT(\ jm) \cdot \partial HWi\ ||\ Q\ ||\ Hj$$

Finally, after applying the abstraction operator, the external behaviour prevails:

$$\tau I(\partial H(Wi\ ||\ Q\ ||\ C\ ||\ Hj\ ||\ T\ )) = rin(IoT(-)) \cdot \tau I(\partial H(Wi\ ||\ Q\ ||\ C\ ||\ Hj\ ||\ T\ ))$$

Action Cloud OUT :

Action 4: Migrate VM from Fog to Cloud as IoT device leaves

Inputs: VM identifier jm, host Hj, wireless relay Wi

Outputs: Migrated VM in Cloud

IoT device connected to wireless relay Wi is about to leave the Fog environment.

Wi sends a control message ctr4 to the orchestrator Q, indicating the VM identifier jm and the host Hj where the VM is located.

Q receives ctr4 and initiates the migration process to move the VM from host Hj to the Cloud environment.

Q searches for the Cloud identifier using the process 'Find'.

Q sends a control message to the Cloud C with the Cloud identifier and the VM identifier jm, signaling the migration of the VM from host Hj to the Cloud.

Cloud C processes the message and prepares to receive the VM from host Hj.

Host Hj starts the migration process, sending the VM to the Cloud C.

Cloud C receives the VM from host Hj and finalizes the migration process.

Q receives an acknowledgment from the Cloud C, confirming the successful migration of the VM.

Q sends an acknowledge message to Wi, indicating that the VM has been successfully migrated to the Cloud.

Wi processes the message and acknowledges the IoT device that the VM has been migrated to the Cloud.

The IoT device disconnects from wireless relay Wi, leaving the Fog environment.

Note: The process 'Find' in Step 4 refers to looking up the Cloud identifier where the VM will be migrated to.

$$Wi = sWi\ Qctr4(jm) \cdot rQWi\ km \cdot UnbindIoT(jm) \cdot BindIoT(km) \cdot soutIoT(km) \cdot Wi$$

$$Q = rWi\ Qctr4(jm) \cdot Find(jm) \cdot sQ\ H\ j\ ctr4(jm, km) \cdot rH\ j\ QCloudOut(jm, km) \cdot sQCCloudOut(jm, km) \cdot rC\ QACK, km \cdot sQWi\ km \cdot Q$$

$$H\ j = rQ\ H\ j\ ctr4(jm, km) \cdot MoveOut(jm, km) \cdot sH\ j\ QCloudOut(jm, km) \cdot H\ jC = rQCCloudOut(jm, km) \cdot MoveIn(jm, km) \cdot sC\ QACK, km \cdot C$$

After the application of the encapsulation operator, this is the result of a *CloudOUT* action:

$$\partial HWi\ ||\ Q\ ||\ H\ j\ ||\ C = cWi\ Qctr4(jm) \cdot Find(jm) \cdot cQ\ H\ j\ ctr4(jm, km)$$

$$\cdot MoveOut(jm, km) \cdot cH\ j\ QCloudOut(jm, km) \cdot cQCCloudOut(jm, km)$$

$$\cdot MoveIn(jm, km) \cdot cC\ QACK, km \cdot cQWi\ km \cdot Unbind(IoT(jm))$$

$$\cdot Bind(IoT(km)) \cdot sout(IoT(km))) \cdot (\partial H(Wi\ ||\ Q\ ||\ H\ j\ ||\ C))$$

Eventually, after applying the abstraction operator, the external behaviour is shown:

$$\tau I\ (\partial H(\ Wi\ ||\ Q\ ||\ C\ ||\ H\ j\ ||\ T\ )) = sout\ (IoT(km)) \cdot \tau I\ (\partial H(\ Wi\ ||\ Q\ ||\ C\ ||\ H\ j\ ||\ T\ ))$$

Action Migration OUT :

The modelling for action 5, Migration Out, is as follows:

The IoT device gets disconnected from the wireless relay Wi through physical channel OUT.

Wi sends a control message ctr5 to the orchestrator Q to inform that the IoT device has left.

The orchestrator Q sends a control message ctr6 to the Fog host Hj where the VM associated with the IoT device is running, to request that the VM be migrated to the Cloud.

Hj sends a control message ctr7 to the Cloud orchestrator Qc to request the creation of a new VM in the Cloud to receive the migrated VM from Hj.

The Cloud orchestrator Qc finds an available Cloud host Hc with enough resources and sends a control message ctr8 to Hc to create and initialize the new VM.

Hc sends a control message ctr9 to Hj to initiate the VM migration from Hj to Hc.

Hj sends a control message ctr10 to Q to inform that the VM migration has started.

Q sends a control message ctr11 to Wi to inform that the VM migration has started and that the IoT device can safely disconnect from Wi.

The IoT device disconnects from Wi through physical channel OUT.

The VM migration is completed when Hc confirms to Hj that the new VM is running properly.

Hj sends a control message ctr12 to Q to inform that the VM migration has been successfully completed.

$$Wi = sWi\ Qctr5(jm) \cdot rQWi\ jm \cdot UnbindIoT(jm) \cdot BindIoT(jm) \cdot soutIoT(jm) + rQWi\ ACK \cdot soutIoT(jm) \cdot Wi$$

$$Q = rWi\ Qctr5(jm) \cdot sQ\ H\ j\ ctr5(jm, jm) \cdot rH\ jQACK, jm \cdot sQWi\ jm\ Select(jm)\ sQWi\ ACK \cdot Q$$

$$H\ j = rQ\ H\ j\ ctr5(jm, jm) \cdot MoveOut(jm, jm) \cdot sH\ j\ T\ MigrateOut(jm, jm) \cdot H\ j$$
$$H\ j = rT\ H\ jMigrateIn(jm, jm) \cdot MoveIn(jm, jm) \cdot sH\ jQACK, jm \cdot H\ j$$

$$T = rH\ j\ T\ MigrateOut(jm, jm) \cdot Networking(jm, jm) \cdot sT\ H\ jMigrateIn(jm, jm) \cdot T$$

After the application of the encapsulation operator, this is the result of a *MigrateOUT* action:

$$\partial HWi \,||\, Q \,||\, H\ j \,||\, H\ j \,||\, T = cWi\ Qctr5(km) \cdot cQ\ H\ j\ ctr5(jm, jm) \cdot MoveOut(jm, jm)$$

$$\cdot cH\ j\ T\ MigrateOut(jm, jm) \cdot Networking(jm, jm) \cdot cT\ H\ jMigrateIn(jm, jm)$$

$$\cdot MoveIn(jm, jm) \cdot cH\ jQACK, jm \cdot cQWi\ jm \cdot UnbindIoT(jm)$$

$$\cdot BindIoT(jm) \cdot soutIoT(jm) Select(jm)\ cQWi\ ACK \cdot soutIoT(jm) \cdot \partial HWi \,||\, Q \,||\, H\ j \,||\, H\ j \,||\, T$$

Eventually, abstraction operator to obtain the external behaviour of the model:

$$\tau I\ \partial HWi \,||\, Q \,||\, C \,||\, H\ j \,||\, H\ j \,||\, T =$$

$$soutIoT(jm) + soutIoT(jm) \cdot \tau I\ \partial HWi \,||\, Q \,||\, C \,||\, H\ j \,||\, H\ j \,||\, T$$

Action Kill VM :

The modelling for action 6, Termination of VM, is as follows:

The IoT device gets disconnected from the wireless relay Wi through physical channel OUT.

Wi sends a control message ctr6 to the orchestrator Q to inform that the IoT device has left.

The orchestrator Q sends a control message ctr13 to the Fog host Hj where the VM associated with the IoT device is running, to request the termination of the VM.

Hj receives ctr13 and initiates the termination process by sending a control message ctr14 to the VM to inform it about the termination request.

The VM receives ctr14 and starts the termination process, which may involve saving its current state and releasing any resources it is using.

Once the termination process is completed, the VM sends a control message ctr15 to Hj to confirm its termination.

Hj receives ctr15 and acknowledges the termination by sending a control message ctr16 to Q to inform that the VM has been successfully terminated.

Q sends a control message ctr17 to Wi to inform that the VM has been terminated and that the IoT device can safely disconnect from Wi.

The IoT device disconnects from Wi through physical channel OUT.

The termination of the VM is completed, and the IoT device has successfully left the Fog environment.

$$Wi = sWi\ Qctr6(jm) \cdot rQWi\ - \cdot UnbindIoT(jm) \cdot sout - \cdot Wi$$

$$Q = rWi\ Qctr6(jm) \cdot Find(jm) \cdot sQHj\ ctr6(jm) \cdot rHj\ QACK, - \cdot sQWi\ - \cdot Q$$

$$Hj = rQHj\ ctr6(jm) \cdot Kill(jm) \cdot sHj\ QACK, -\cdot Hj$$

After the application of the encapsulation operator, this is the result of a a *Kil VM* action:

$$\partial HWi \,||\, Q \,||\, Hj = cWi\ Qctr6(jm) \cdot Find(jm) \cdot cQHj\ ctr6(jm) \cdot Kill(jm)$$
$$\cdot cHj\ QACK, - \cdot cQWi\ - \cdot UnbindIoT(jm) \cdot soutIoT, -\cdot \partial HWi \,||\, Q \,||\, Hj$$

Finally, abstraction operator to obtain the external behaviour of the model:

$$\tau I\ \partial HWi \,||\, Q \,||\, C \,||\, Hj \,||\, T = soutIoT(-) \cdot \tau I\ \partial HWi \,||\, Q \,||\, C \,||\, Hj \,||\, T$$

H. Conclusion

In conclusion, this chapter has reviewed two relevant research studies on the design and modeling of edge computing applications and fog scenarios for moving IoT devices. Both works have provided significant contributions to the field, presenting novel approaches and methodologies for addressing the challenges of utilizing edge and fog resources in IoT scenarios.

The first study proposed a generic application design model that can be adapted to different edge computing scenarios, and provided a methodology for modeling application components and evaluating performance. The second study presented an algebraic modeling approach for fog scenarios, which can capture the dynamic nature of moving IoT devices and optimize resource utilization.

These studies have highlighted the potential benefits of edge and fog computing in IoT scenarios and provided insights into the design and modeling of such systems. However, there are still gaps in the existing research that need to be addressed, such as the optimization of resource allocation and the integration of edge and cloud computing.

Chapitre 4

I. Specification and verification of a simplified version of smart classroom..

1. Introduction:

In today's rapidly evolving technological landscape, the concept of a smart classroom has gained immense significance. A smart classroom encompasses a network of embedded sensors and intelligent devices that are self-configured and can be controlled remotely via the Internet. This advanced technological environment aims to create a comfortable and efficient learning space for students, teachers, and parents. Moreover, it offers great potential for individuals with disabilities, as it can assist them in performing tasks that would otherwise require excessive effort or manual assistance. In this chapter, we will delve into a case study on a smart classroom, specifically focusing on its architecture and various transactions that take place within this innovative educational setting.

In this chapter, we focus on creating a simulation of a smart classroom that includes multiple devices, such as a smart thermostat, where users can interact with these devices. The objectives are to define the total number of devices in the classroom, model the possible states of the thermostat, establish communication mechanisms between users and devices, set up processes representing both users and devices, allow users to perform actions such as accessing data and modifying the thermostat's state, and finally, ensure that all processes complete their execution within the main process. Our main goal is to simulate dynamic interactions between users and devices in a smart classroom using the Promela language.

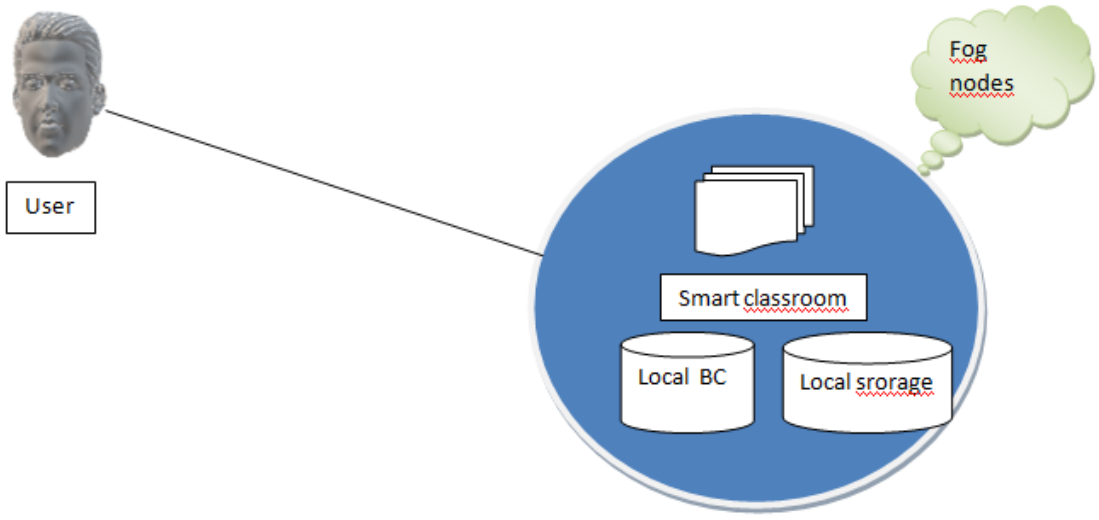


Figure 1 : Classroom architecture

2. Specification and validation

In this section, we provide a specification of the smart classroom using ACP (Algebra of Communicating Processes). The ACP specification captures the behavior and interaction of the different components in the smart classroom, such as the user and the devices. Once the ACP specification is defined, it is then translated into Promela, a language supported by the model checking tool SPIN, for validation and verification purposes. The Promela translation allows us to apply the powerful model checking capabilities of SPIN to analyze the correctness and properties of the smart classroom system. By leveraging this approach, we can ensure that the smart classroom functions as intended, adhering to the specified requirements and exhibiting the desired behavior. The combination of ACP specification and model checking with SPIN provides a rigorous and effective method for verifying the correctness of the smart classroom system.

3. Specification in ACP

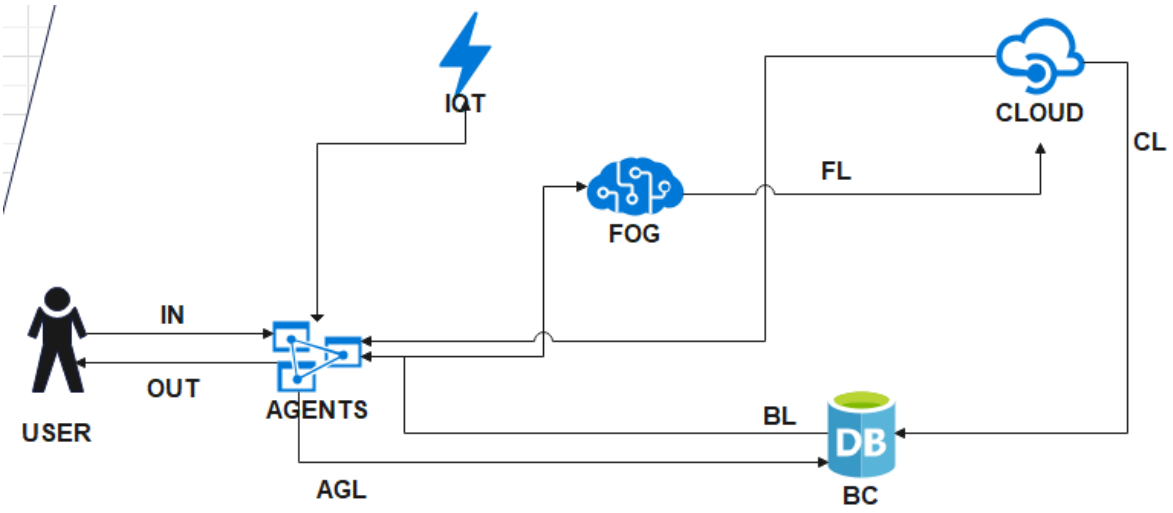


Figure 2. Model for smart class room with fog computing in ACP

1) $USER_u$ = (S_{in}**(R),** ⍉ **(R))** . $USER_u$
2) $AGENTS_a$ = (R_{in}**(R)** , □ **(R)** , S_{agl}**(R))**.$AGENTS_a$
3) BC_b = (R_{agl}**(P)** , S_{agl}**(d)<**$USER_u$**>**S_{bl}**(out))**.BC_b
4) $AGENTS_a$ = (R_{bl}**(R)** , S_{agl}**(P))** . $AGENTS_a$
5) IOT_i**=** (R_{agl}**(P)** , □ **(P)** , S_{agl}**(d))** .IOT_i
6) $AGENTS_a$ = (R_{agl}**(d)** , ⍉ **(d)**) .$AGENTS_a$
7) FOG_f = (R_{agl}**(R)** , ⍉ **(d)** ,S_{fl}**(R))** . FOG_f
8) $CLOUD_c$ = (R_{fl}**(R)** , **CREATE (BC)** ‖S_{cl}**(R))** .$CLOUD_c$
9) $AGENTS_a$**=** (R_{cl}**(d)** , ⍉ **(d)** ,S_{out}**(d))** .$AGENTS_a$
10) $USER_u$ = R_{out}**(d)** . $USER_u$

4. **Encapsulation :**

11/ $\sum_a^{\infty}$ $\sum_u$ $\sum_b$ $\sum_i$ $\sum_f$ $\sum_c$ $=\delta(\ USER_u \| AGENTS_a \| BC_b \| IOT_i \| FOG_f \| CLOUD_c\ )$ **(**S_{in}**(R),** **(R))** **.(**R_{in}**(R)** **,** **(R)** **,** S_{agl}**(R)) . (**R_{agl}**(P)** **,** S_{agl}**(d)<**$USER_u$**>**S_{bl}**(out)).(**R_{bl}**(R)** **,** S_{agl}**(P))** **. (**R_{agl}**(P)** **,** **(P)** **,** S_{agl}**(d)) .(**R_{agl}**(d)** **,** **(d)** **) . (**R_{agl}**(R) , (d) ,**S_{fl}**(R))** **. (**R_{fl}**(R)** **, CREATE (BC)** $\| S_{cl}$**(R)) .(**R_{cl}**(d)** **,** **(d) ,**S_{out}**(d)) .**R_{out}**(d).**$\delta(\ USER_u \| AGENTS_a \| BC_b \| IOT_i \| FOG_f \| CLOUD_c\ )$

5. **Abstraction :**

12/ $\sum_a$ $\sum_u$ $\sum_b$ $\sum_i$ $\sum_f$ $\sum_c$ $=$ $\tau_I(\ \delta(\ USER_u \| AGENTS_a \| BC_b \| IOT_i \| FOG_f \| CLOUD_c)$ $)=S_{in}$**(R).**R_{out}**(d)** **.**$\tau_I(\ \delta(\ USER_u \| AGENTS_a \| BC_b \| IOT_i \| FOG_f)$

Otherwise, the external behavior of the real system may also be expressed by means of ACP

$$\mathbf{X} = S_{in}\mathbf{(R).}R_{out}\mathbf{(d)} \cdot \mathbf{X}$$

6. **Multi**

$\sum_a$ $\sum_u$ $\sum_b$ $\sum_i$ $\sum_f$ $\sum_c$ $=$ $\tau_I(\delta(\ USER_u \| AGENTS_a \| BC_b \| IOT_i \| FOG_f \| CLOUD_c\ )$ $\Leftrightarrow$ **X**

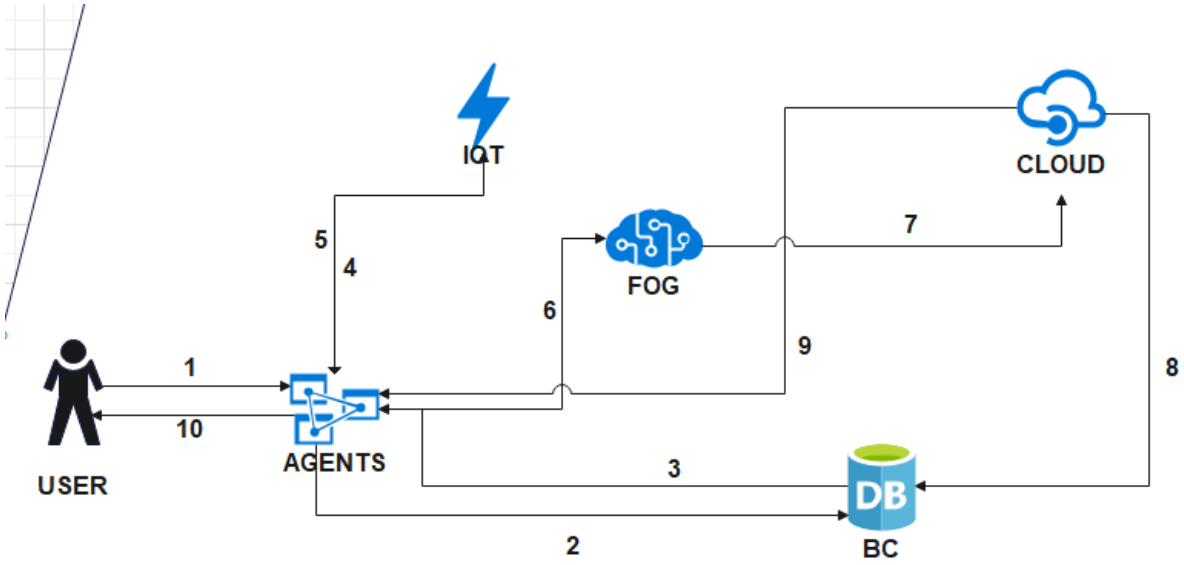


Fig. 3 Topology scheme

Specification in Promela

```
#define NUM_DEVICES 3  // Total number of devices in the smart
classroom

// Definition of possible states of the smart thermostat

#define OFF 0

#define ON 1

// Definition of communication channels between processes

chan request[NUM_DEVICES] = [NUM_DEVICES] of { byte, byte }

chan response[NUM_DEVICES] = [NUM_DEVICES] of { byte, byte }


// Declaration of state variables for the smart thermostat

byte smart_thermostat_status = OFF

byte temperature = 0


// Process representing the user

proctype User() {

    byte device_id = 0  // Device identifier (e.g., thermostat)

    byte action = 0     // Action to perform (e.g., access data)

    byte data = 0       // Data to send (e.g., requested
temperature)


    atomic {

        // Simulating a user request

        request[device_id]!action, data

        // Waiting for a response

        response[device_id]?action, data

        // Processing the response

        // ...

    }
```

```
}

// Process representing a device in the smart classroom (e.g.,
thermostat)
proctype Device(byte device_id) {
byte action = 0  // Received action (e.g., access data)
    byte data = 0    // Received data (e.g., requested temperature)

    do
    :: request[device_id]?action, data ->
        atomic {
            // Processing the request
            if
            :: action == 1 ->  // Access data
                // Simulating data retrieval (e.g., current
temperature)
                data = temperature
                // Sending the response
                response[device_id]!action, data
            :: action == 2 ->  // Modify state
                // Simulating modification of the smart thermostat
state
                smart_thermostat_status = data
                // Sending a confirmation of the modification
                response[device_id]!action, data
            :: action == 3 ->  // Store data
                // Simulating storing data in a local storage device
                // ...
                // Sending a storage confirmation
                response[device_id]!action, data
```

```
            fi
        }
    od
}

// Main process representing the smart classroom
init {
    // Creating user processes
    run User()
    // Creating device processes
    byte device_id = 0
    do
    :: device_id < NUM_DEVICES ->
        run Device(device_id)
        device_id++
    :: else -> break
    od
    // Waiting for the processes to finish execution
do
    :: atomic { skip }
    od
}
```

The Promela code represents a simulation of a smart classroom, consisting of multiple devices such as a smart thermostat. The purpose of this simulation is to model the interactions between users and devices in the classroom.

The smart classroom includes a total number of devices defined by the constant NUM_DEVICES. In this example, NUM_DEVICES is defined as 3.

The possible states of the smart thermostat are defined by the constants OFF (0) and ON (1).

The code uses communication channels between processes to enable the interactions. Two arrays of channels are defined: request and response. Each array contains NUM_DEVICES channels, which are communication channels for sending and receiving data between processes.

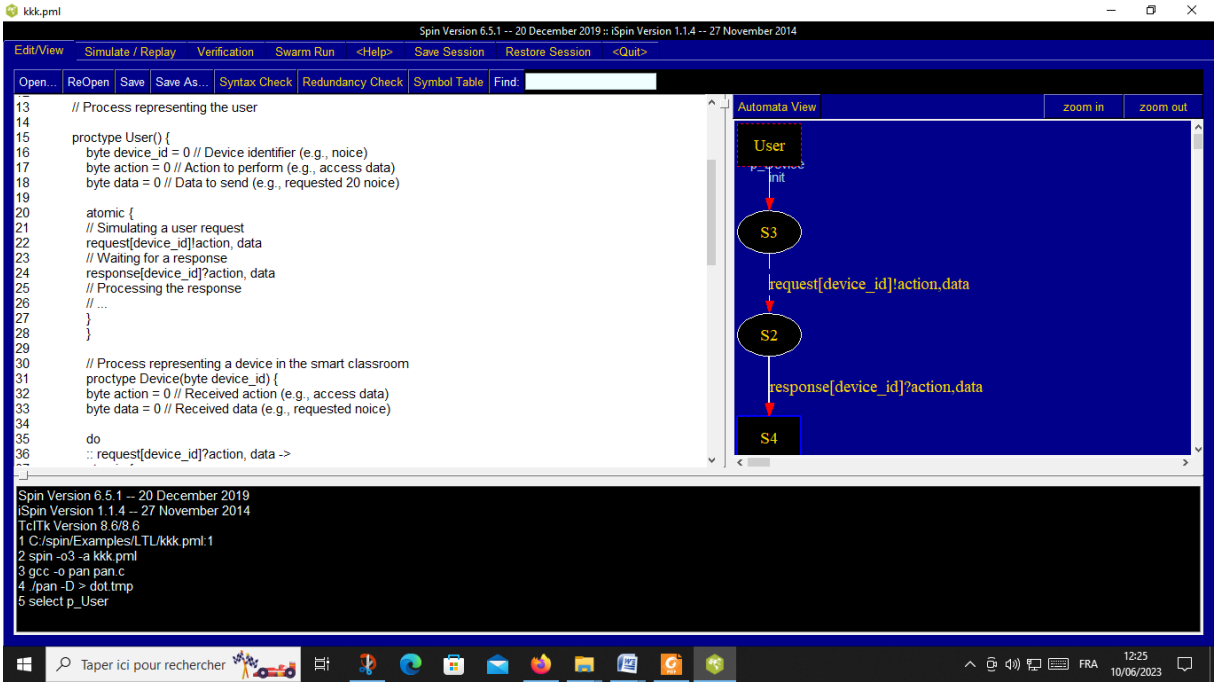
The User process represents a user in the classroom. It has a device identifier (device_id), an action to perform (action), and data to send (data).

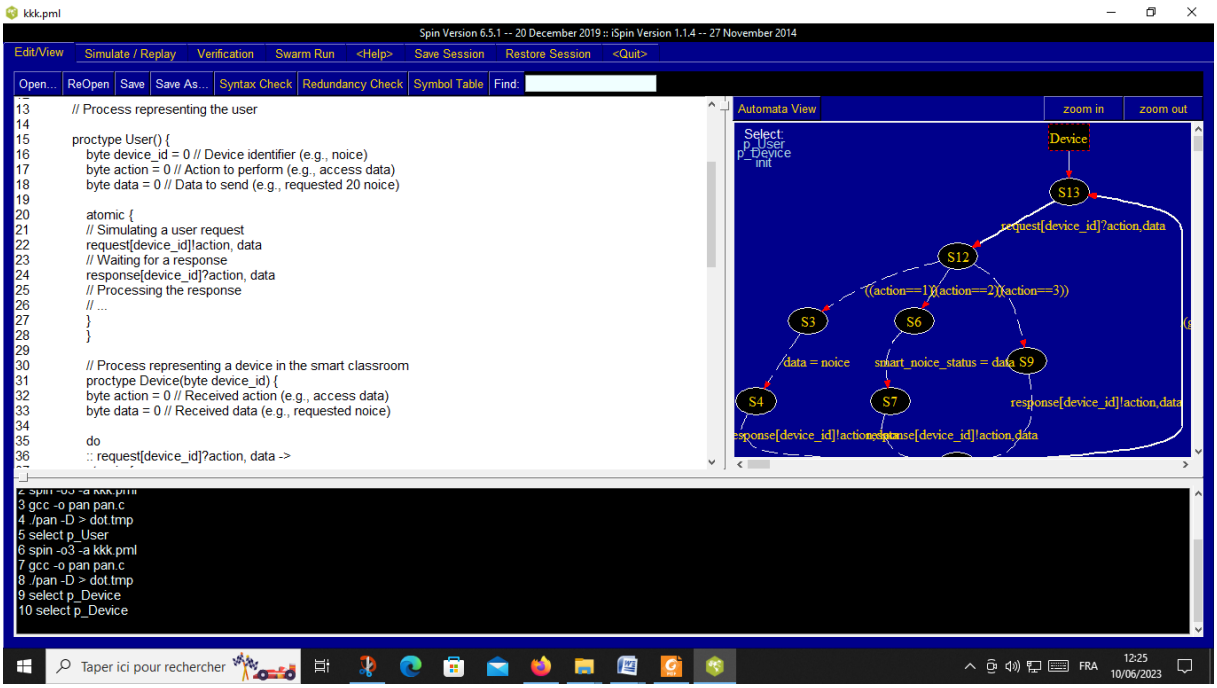
In an atomic section, the User process sends a request using the request[device_id] channel with the specified action and data. Then, it waits for a response using the response[device_id] channel. Once the response is received, the process can process the response.

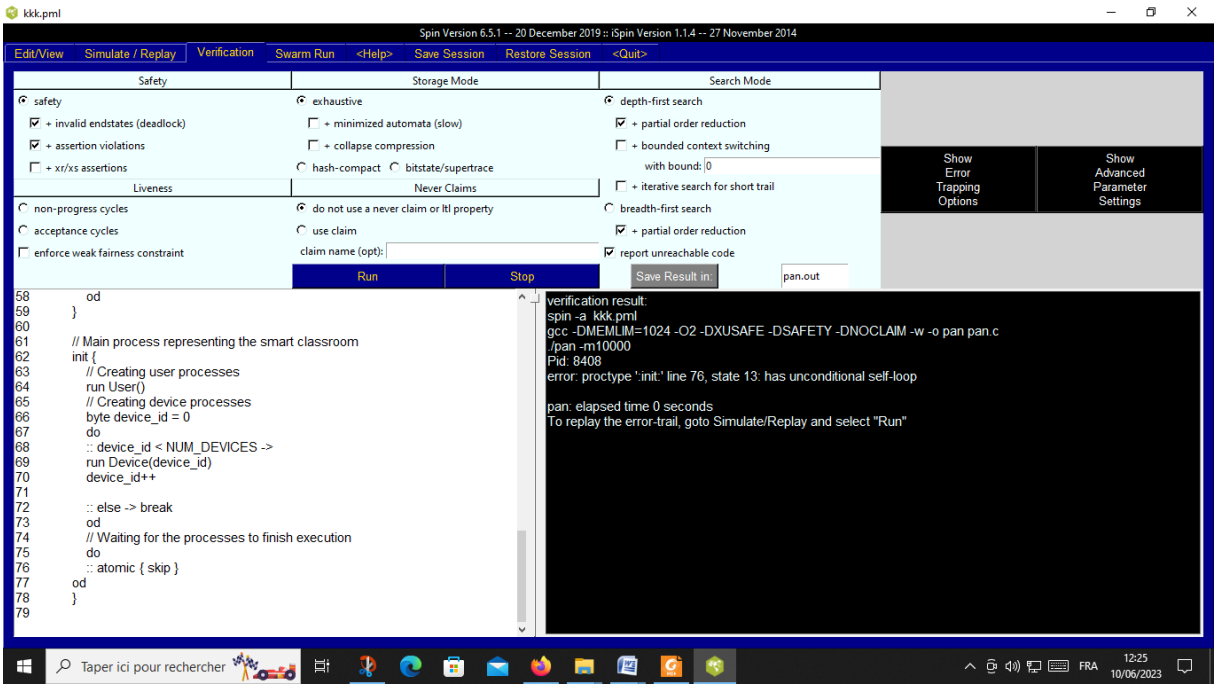
The Device process represents a device in the classroom, such as the thermostat. It also has a device identifier (device_id), a received action (action), and received data (data).

In an infinite loop (do), the Device process waits for a request using the request[device_id] channel. When a request is received, the process handles the request based on the specified action. If the action is equal to 1, the process accesses the data, simulating retrieving the current temperature (temperature). Then, it sends the response using the response[device_id] channel. If the action is equal to 2, the process modifies the state of the smart thermostat using the received data. Then, it sends a confirmation of the modification using the response[device_id] channel. If the action is equal to 3, the process stores the data in a local storage device (not specified in the code). Then, it sends a storage confirmation using the response[device_id] channel.

The init process is the main process representing the smart classroom. It initializes the User and Device processes by executing run User() and run Device(device_id) respectively. It then iterates over device_id until it reaches NUM_DEVICES, executing run Device(device_id) for each device identifier. Finally, it waits for all processes to finish using an empty and atomic do loop.


kkk.pml
Spin Version 6.5.1 -- 20 December 2019 :: iSpin Version 1.1.4 -- 27 November 2014
Edit/View
Simulate / Replay
Verification
Swarm Run
<Help>
Save Session
Restore Session
<Quit>
Open...
ReOpen
Save
Save As...
Syntax Check
Redundancy Check
Symbol Table
Find:
13 // Process representing the user
14
15 proctype User() {
16 byte device_id = 0 // Device identifier (e.g., noice)
17 byte action = 0 // Action to perform (e.g., access data)
18 byte data = 0 // Data to send (e.g., requested 20 noice)
19
20 atomic {
21 // Simulating a user request
22 request[device_id]!action, data
23 // Waiting for a response
24 response[device_id]?action, data
25 // Processing the response
26 // ...
27 }
28 }
29
30 // Process representing a device in the smart classroom
31 proctype Device(byte device_id) {
32 byte action = 0 // Received action (e.g., access data)
33 byte data = 0 // Received data (e.g., requested noice)
34
35 do
36 :: request[device_id]?action, data ->
Automata View
zoom in
zoom out
User
init
S3
request[device_id]!action,data
S2
response[device_id]?action,data
S4
Spin Version 6.5.1 -- 20 December 2019
iSpin Version 1.1.4 -- 27 November 2014
TclTk Version 8.6/8.6
1 C:/spin/Examples/LTL/kkk.pml:1
2 spin -o3 -a kkk.pml
3 gcc -o pan pan.c
4 ./pan -D > dot.tmp
5 select p_User
Taper ici pour rechercher
FRA
12:25
10/06/2023


kkk.pml
Spin Version 6.5.1 -- 20 December 2019 :: iSpin Version 1.1.4 -- 27 November 2014
Edit/View
Simulate / Replay
Verification
Swarm Run
<Help>
Save Session
Restore Session
<Quit>
Open...
ReOpen
Save
Save As...
Syntax Check
Redundancy Check
Symbol Table
Find:
13 // Process representing the user
14
15 proctype User() {
16 byte device_id = 0 // Device identifier (e.g., noice)
17 byte action = 0 // Action to perform (e.g., access data)
18 byte data = 0 // Data to send (e.g., requested 20 noice)
19
20 atomic {
21 // Simulating a user request
22 request[device_id]!action, data
23 // Waiting for a response
24 response[device_id]?action, data
25 // Processing the response
26 // ...
27 }
28 }
29
30 // Process representing a device in the smart classroom
31 proctype Device(byte device_id) {
32 byte action = 0 // Received action (e.g., access data)
33 byte data = 0 // Received data (e.g., requested noice)
34
35 do
36 :: request[device_id]?action, data ->
Automata View
zoom in
zoom out
Select:
p_User
p_Device
init
Device
S13
request[device_id]?action,data
S12
((action==1)||(action==2)||(action==3))
S3
S6
data = noice
smart_noice_status = data
S9
S4
S7
response[device_id]!action,data
3 gcc -o pan pan.c
4 ./pan -D > dot.tmp
5 select p_User
6 spin -o3 -a kkk.pml
7 gcc -o pan pan.c
8 ./pan -D > dot.tmp
9 select p_Device
10 select p_Device
Taper ici pour rechercher
FRA
12:25
10/06/2023



This Promela specification represents a simplified version of the smart classroom with three devices (e.g., thermostat, surveillance camera, etc.) and one user. Each device receives requests from the user and performs corresponding actions, such as accessing data, modifying state, and storing data. The communication channels request and response facilitate the interaction between the user and the devices, enabling seamless communication and coordination within the smart classroom environment. The specification captures the essential functionality and behavior of the smart classroom system, laying the foundation for validation and verification using model checking techniques.

7. Formal verification

The following propertiesserve as examples of the types of properties that can be verified using model checking techniques. Depending on the specific requirements and goals of the smart classroom system, additional properties related to safety, performance, or reliability can be defined and checked using the Promela specification.

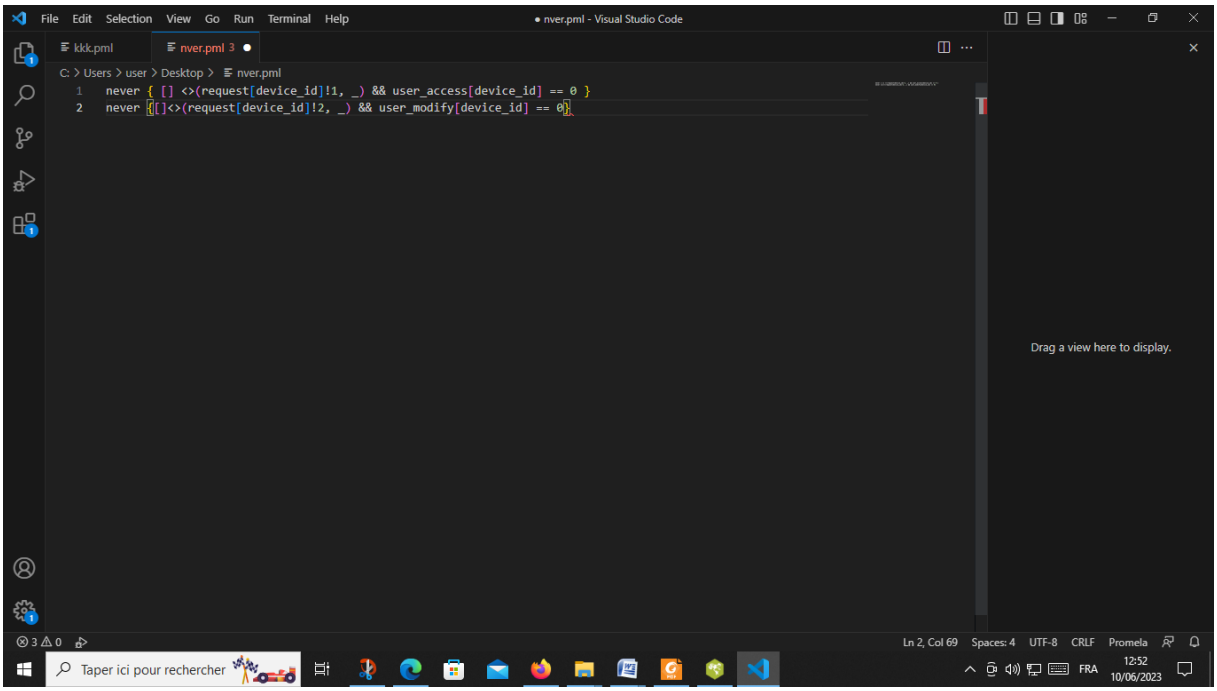
8. Property 1: Access Control

Property: "Only authorized users can access and modify the smart classroom devices."

Description: This property ensures that the system enforces appropriate access control measures, allowing only authorized users to access and modify the devices in the smart classroom. It prevents unauthorized access and modifications, protecting the integrity and security of the system.

```
never {
 // Unauthorized access
 []<>(request[device_id]!1, _) && user_access[device_id] == 0
```

```
}
never {
 // Unauthorized modification
 []<>(request[device_id]!2, _) && user_modify[device_id] == 0
}
```

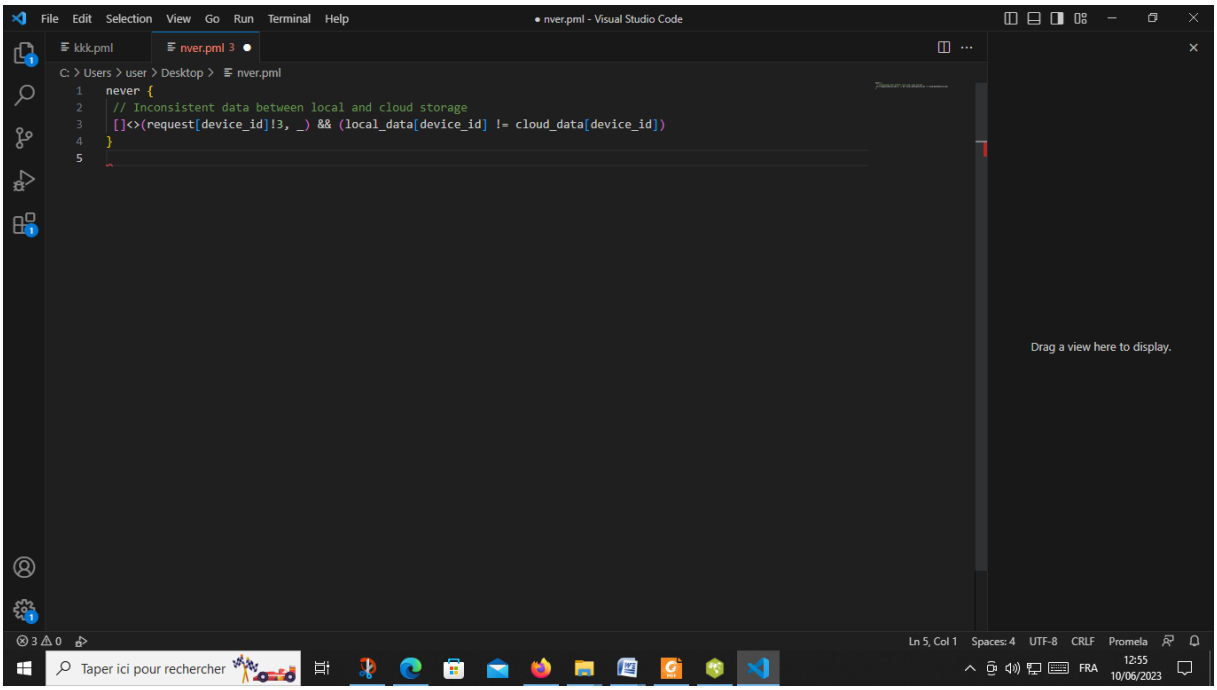


9. Property 2: Data Consistency

Property: "Data stored in the local storage and cloud storage of the smart classroom remains consistent."

Description: This property verifies that the data stored in both the local storage and cloud storage of the smart classroom remains consistent and synchronized. It ensures that any updates or modifications to the data are correctly propagated and reflected in both storage locations, preventing data inconsistencies or discrepancies.

```
never {
 // Inconsistent data between local and cloud storage
 []<>(request[device_id]!3, _) && (local_data[device_id] != cloud_data[device_id])
}
```

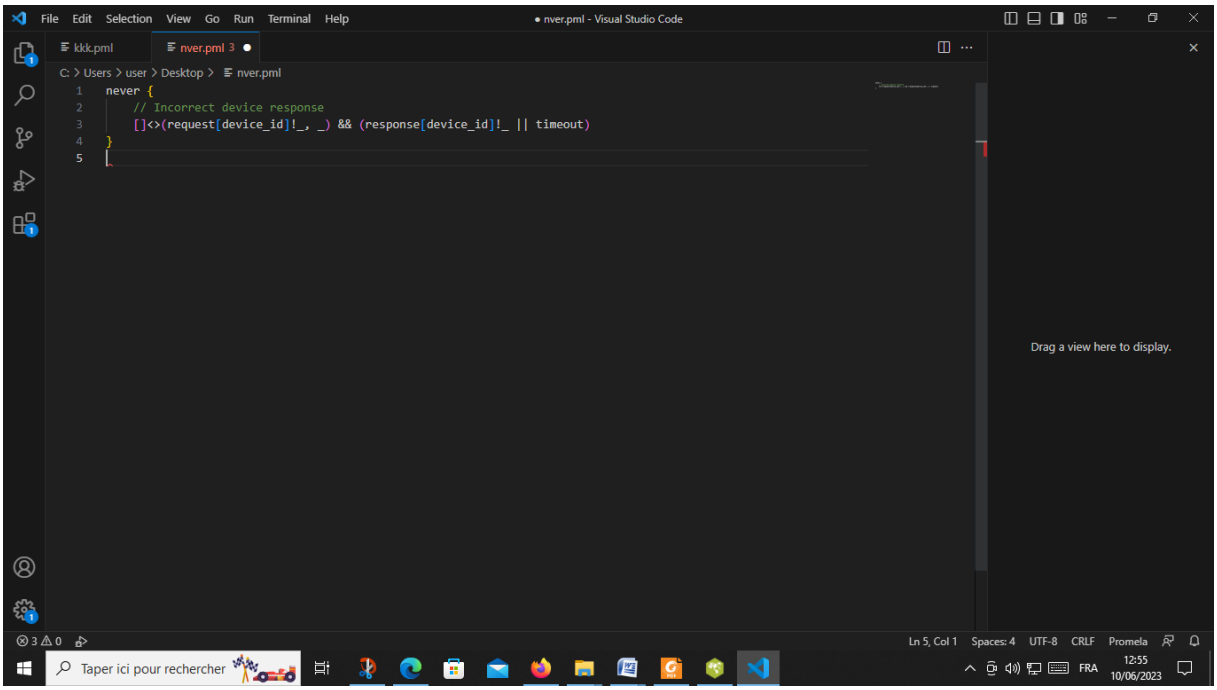


10. Property 3: Device Interaction

Property: "Devices in the smart classroom communicate and respond correctly to user requests."

Description: This property validates that the devices in the smart classroom interact and communicate as expected with the user. It ensures that the devices receive and respond to user requests accurately, providing the intended functionality and behavior. This property helps detect any communication failures, synchronization issues, or incorrect device responses within the system.

```
never {
  // Incorrect device response
  []<>(request[device_id]!_, _) && (response[device_id]!_ || timeout)
}
```



J. Conclusion:

The case study on the smart classroom provides valuable insights into the potential of technology in transforming traditional learning environments. The architecture outlined in the study highlights the key components involved, such as users, devices, fog nodes, local storage, and local blockchain. These components work in tandem to facilitate seamless communication, data analysis, and service delivery within the smart classroom.

Additionally, the study presents three essential transactions: access to data, modification of data, and storage of data. These transactions exemplify how users and devices interact within the smart classroom ecosystem, enabling real-time access to information, efficient data management, and remote control over classroom devices.

II. General Conclusion and Perspectives:

This dissertation has covered four chapters, each addressing different aspects of Fog Computing and its practical applications. The chapters encompassed a wide range of topics, including the context and definitions of Fog Computing, related work in the field, algebraic modeling of Fog scenarios, and a case study on the specification and verification of a classroom environment.

In Chapter 1, we provided an overview of Fog Computing, discussing its context and key definitions. We highlighted the advantages of Fog Computing over traditional Cloud Computing and addressed the specific challenges associated with implementing Fog Computing.

Chapter 2 focused on related work in the field. We described two specific studies: the modeling of a generic edge computing application design and the algebraic modeling of a generic Fog scenario for moving IoT devices. These studies provided valuable insights into the practical aspects of implementing Fog Computing solutions.

In Chapter 3, we delved into the algebra of communicating processes. We explored the theoretical foundations of process algebra and its application in modeling and analyzing communication systems, which is crucial for understanding the behavior of complex Fog Computing architectures.

Finally, in the last chapter, we presented a detailed case study that involved specifying a classroom environment using the ACP (Algebra of Communicating Processes) model. We then translated the specification into Promela and verified it using the SPIN model checker. This case study demonstrated the practical application of formal methods in ensuring access control, data consistency, and device interaction in a Fog Computing environment.

Looking ahead, the field of Fog Computing is rapidly evolving, and there are several promising avenues for future research. Firstly, further exploration of novel architectural designs and protocols that enhance the scalability, reliability, and security of Fog Computing systems is warranted. Additionally, the development of advanced tools and techniques for modeling, analysis, and verification of Fog Computing architectures can contribute to the robustness and dependability of these systems. Furthermore, investigations into energy-

efficient resource allocation and management strategies in Fog Computing environments can lead to significant improvements in sustainability and cost-effectiveness.

In conclusion, this dissertation has provided a comprehensive understanding of Fog Computing, explored related works, introduced the algebra of communicating processes, and presented a practical case study. It is our hope that this work contributes to the advancement of Fog Computing research and inspires further investigations into the exciting possibilities offered by this emerging paradigm.

III. Bibliography

[1] Stojmenovic, I., Wen, S., Huang, X., & Luan, H. (2016). An overview of fog computing and its security issues. *Concurrency and Computation: Practice and Experience*, *28*(10), 2991-3005.

[2] Bonomi Flavio, Milito Rodolfo, Zhu Jiang, and Addepalli Sateesh. 2012. Fog computing and its role in the Internet of Things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing (MCC'12)*. ACM, New York, NY. 13–16. DOI: DOI: https://doi.org/10.1145/2342509.2342513

[3] Cisco (2013) Fog computing, ecosystem, architecture and applications. Cisco ReportRFP-2013-078

[4] Cearley DW (2010) Cloud computing: key initiative overview. Gartner Report, 2010

[5] Banafa A (2014) What is fog computing? https://www.ibm.com/blogs/Cloud-computing/2014/08/Fog-computing/

[6] OpenFog (2017) OpenFog reference architecture for fog computing. https://www.openFogconsortium.org/wp-content/uploads/OpenFog_Reference_Architecture_2_09_17-FINAL-pdf.

[7] Shariffdeen R (2017) Fog computing vs edge computing. https://medium.com/@rshariffdeen/edge-computing-vs-Fog-computing-5b23d6bb049d

[8] Cearley DW (2010) Cloud computing: key initiative overview. Gartner Report, 2010

[9] J.A. Bergstra, J.W. Klop, Process algebra for synchronous communication, Information and Control, Volume 60, Issues 1–3, 1984, Pages 109-137,ISSN 0019-9958, https://doi.org/10.1016/S0019-9958(84)80025-X.

[10] J.A. Bergstra, J.W. Klop,Algebra of communicating processes with abstraction,Theoretical Computer Science,Volume 37,1985,Pages 77-121,ISSN 0304-3975,https://doi.org/10.1016/0304-3975(85)90088-X.

[11] J. C. M. BAETEN,T. BASTEN ,M. A. RENIERS - Process Algebra: Equational Theories of Communicating Processes, 2010.

[12]Roig, P.J.; Alcaraz, S.; Gilly, K.; Bernad, C.; Juiz, C. Modeling of a Generic Edge Computing Application Design. *Sensors* 2021, *21*, 7276. https://doi.org/10.3390/s21217276.

[13]Roig, P.J., Alcaraz, S., Gilly, K., Juiz, C. Algebraic Modelling of a Generic Fog Scenario for Moving IoT Devices. In: Thampi, S.M., Lloret Mauri, J., Fernando, X., Boppana, R., Geetha, S.,

Sikora, A. (eds) Applied Soft Computing and Communication Networks. Lecture Notes in Networks and Systems, vol 187. Springer, 2021, Singapore. https://doi.org/10.1007/978-981-33-6173-7_1.

[14]Padua D (2011) Encyclopedia of parallel computing. Springer, Heidelberg. https://doi.org/10.1007/978-0-387-09766-4.

[15] Mechalikh, C., Taktak, H., & Moussa, F. (2019). A Scalable and Adaptive Tasks Orchestration Platform for IoT. 2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC), 1557-1563.

[16]Daoud, W.B., Obaidat, M.S., Meddeb-Makhlouf, A. et al. TACRM: trust access control and resource management mechanism in fog computing. Hum. Cent. Comput. Inf. Sci. 9, 28 (2019). https://doi.org/10.1186/s13673-019-0188-3