Scalable Computing:
Practice and Experience

Scientific International Journal
for Parallel and Distributed Computing

ISSN: 1895-1767

Universitatea de Vest din Timişoara

Volume 17(3) September 2016

SUBSCRIPTION INFORMATION: please visit http://www.scpe.org

Scalable Computing: Practice and Experience

Volume 17, Number 3, September 2016

TABLE OF CONTENTS

DOI 10.12694/scpe.v17i3.1177
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 17, Number 3, pp. iii–iv. http://www.scpe.org

INTRODUCTION TO THE SPECIAL ISSUE ON RELIABILITY AND SECURITY OF EHEALTH INFORMATION SYSTEMS

With the increasing population and aging society in several countries, healthcare providers aim to enhance the quality of the healthcare services while balancing the risk mitigation and service cost. Therefore, several new information technologies and innovative communication methodologies have evolved to improve the healthcare sector. ICT-based technologies help in decreasing the healthcare system overhead and increasing the quality of healthcare services. These technologies may include biosensors, computer aided diagnosis, Wireless Body Sensor Network (WBSN), mobile health, Radio Frequency Identification (RFID), cloud computing, communication protocols, electronic medical records, big data, and internet of things (IoT). Therefore, the complexity of healthcare systems has increased dramatically during the last two decades. Despite having several approaches developed for testing and verification of healthcare systems, ICT related medical incidents that led into losses of money, time, reputation, and in certain cases, lives, still happen frequently. It is believed that healthcare systems do not get enough testing and verification before being put into use, even though they are considered safety critical systems. This is due to the high cost of testing, short time to marker, and the lack of proper testing and verification techniques in the literature. Design errors, system usage problems, design reliability issues, compliance Issues, system failure and vulnerabilities in eHealth care system can lead to critical conditions or even death.

The special issue publishes three papers that extended from papers presented at IEEE Healthcom 2015, in addition to two new submitted papers. The first paper in this issue by Pervez et al. titled "Improvement Strategies for Device Interoperability Middleware (DIM) using Formal Reliability analysis", where the authors used probabilistic model checker PRISM for analyzing Device Interoperability Middleware (DIM) [1]. The second paper by Bhardwaj and Prasad is titled "PRAVAH: Parameterised Information Flow Control in e-Health". The authors addressed the problem of enforcing information flow control (IFC) in hospital domains in eHealth systems using a parameterised lattice-based IFC framework called PRAVAH [2]. The third paper by Ayache et al. is titled "Analysis and verification of XACML policies in a medical cloud environment". The authors presented a Cloud Policy Verification Service (CPVS) for the analysis and the verification of access control policies specified using XACML [3]. The fourth paper by Sadki and Bakkali is titled "Resolving conflicting privacy policies in m-health based on prioritization". The authors presented resent an approach to resolve the problem of conflicting privacy policies in mobile health environments using AHP (Analytic Hierarchy Process) prioritization technique and reputation mechanism [4]. The fifth paper by Gawanmeh et al. is titled "Formal Analysis of a Microfluidic Device for Blood Cell Separation". The authors used formal analysis in order to formalize and validate the movement of blood cells in a microdevice under different forces for the purpose of cell separation [5]. We would like to thank the editorial board of SCPE for the efforts they made to make this special issue, and all the reviewers for their efforts and feedback.

Kashif Saleem, King Saud University, Saudi Arabia
Amjad Gawanmeh, Concordia University, Montreal, Canada and Khalifa University, UAE

REFERENCES

[1] Usman Pervez, Asiah Mahmood, Osman Hasan, Khalid Latif, and Amjad Gawanmeh, , and Ahmad Alomari. "Improvement Strategies for Device Interoperability Middleware (DIM) using Formal Reliability Analysis." Scalable Computing: Practice and Experience 17 (3), 2016, 150-170.
[2] Chandrika Bhardwaj and Sanjiva Prasad. "PRAVAH: Parameterised Information Flow Control in e-Health." Scalable Computing: Practice and Experience 17 (3), 2016, 171-187.
[3] Meryeme Ayache, Mohammed Erradi, Ahmed Khoumsi, and Bernd Freisleben. "Analysis and Verification of XACML Policies in a Medical Cloud Environment." Scalable Computing: Practice and Experience 17 (3), 2016, 189-205.
[4] Souad Sadki, Hanan EL Bakkali. "Resolving conflicting privacy policies in m-health based on prioritization." Scalable Computing: Practice and Experience 17 (3), 2016, 207-226.
[5] Amjad Gawanmeh, Anas Alazzam, and Bobby Mathew. "Formal Verification of a Microfluidic Device for Blood Cell Separation." Scalable Computing: Practice and Experience 17 (3), 2016, 227-235.

DOI 10.12694/scpe.v17i3.1178
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 17, Number 3, pp. 155–170. http://www.scpe.org

IMPROVEMENT STRATEGIES FOR DEVICE INTEROPERABILITY MIDDLEWARE USING FORMAL RELIABILITY ANALYSIS

USMAN PERVEZ*, ASIAH MAHMOOD*, OSMAN HASAN*, KHALID LATIF† AND AMJAD GAWANMEH‡

Abstract. Ensuring the correctness of middleware that ensures interoperability of various medical devices is one of the biggest challenges in the e-health domain. Traditionally, these Device Interoperability Middleware (DIM) are analyzed using software testing. However, given the inherent incompleteness of testing and the randomness of the user behaviours, the analysis results are not guaranteed to be accurate. Some of these inaccuracies in analysis results could even put human life at risk. In order to overcome these limitations, we propose to use a probabilistic model checker PRISM for analyzing DIM. The proposed approach allows us to rigorously verify reliability properties of the given DIM and thus allows the designers to make appropriate measures to design more reliable systems. For illustration, we formally analyze a middleware that uses the HL7 FHIR and ontology-based description of the devices and a communication protocol to bridge the gap in heterogeneity for dealing with different vendors and incompatible data formats.

Key words: Reliability Analysis, Health Information System (HIS), Device Interoperability Middleware (DIM), Markov Chain, PRISM

AMS subject classifications. 92C50, 92-08, 68Q60, 60J05, 60J25

1. Introduction.

With the fast growing technology in the world, a lot of effort has been put to automate the worksflows, ranging from domestic to industrial workflows, with the aim to enhance the performance of work and shorten the completion time. Hospital workflows have also drawn the attention of the researchers and thus many medical equipments and devices have been developed to perform the work, such as performing medical tests and data capturing, storing, management or transmission. These medical devices and systems have been categorized as Health Information System (HIS) [9] and are increasingly found in almost every hospital now-a-days. Although, the workflows that are composed of HIS systems are better than manual workflows, yet there are many inherit problems, such as device interpretability. Conc univ

Device Interoperability problem refers to the lack of communication between the HIS systems due to the lack of standardization in the manufacturing of medical devices and thus it has become one of the biggest problem that needs to be incorporated in order to setup a hospital workflow or upgrade the existing workflow. The medical devices are diverse in nature and their functionality differs from device to device and the communication mechanism may also be different (e.g., Wifi, Bluetooth or Serial Port). To understand the impact of the interoperability problem, consider a hospital that aims to setup its workflow using HIS systems. This new setup can only be established if we are able to find medical devices that work on the same standards and support the same device integration mechanisms. Similarly, if a hospital aims to upgrade its existing e-health based medical system by adding a new medical device, such as a urine testing device, blood testing device or sugar testing device, then the new medical device must be compatible with the existing electronic environment and must follow the same standard that the other medical devices are following. In case of an unavailability of such a device, the hospital may have to upgrade all of its existing medical devices, which would certainly be a very undesirable solution for most hospitals.

One of the promising solutions to the above-mentioned problem is the development of a middleware that completely resolves the problem by bridging the gap between different standards of the medical devices. For example, if a device using the serial port and ASTM E1394 [7] or its replacement CLSI LIS01 standard [14] required to communicate with the HL7 FHIR based laboratory information system, then the communication can take place by introducing a middleware between them in such a way that it translates the output of the device coming on the serial port to HL7 FHIR compatible data so that it can be received by the laboratory

*School of Electrical Engineering and Computer Science (SEECS), National University of Sciences and Technology (NUST), Islamabad, Pakistan. ((usman.pervez, asiah.mahmood, osman.hasan)@seecs.edu.pk).

†Department of Computer Science, COMSATS Institue of Information Technolgy, Islamabad, Pakistan. (khalid.latif@comsats.edu.pk).

‡Department of Electrical and Computer Engineering, Khalifa University, UAE, and Department of Electrical and Computer Engineering, Concordia University, Montreal, Canada. (amjad.gawanmeh@kustar.ac.ae).

information system. Similarly, the middleware will also be responsible to translate the requests of the laboratory information systems into serial data so that it can be received by the device. This paradigm shift of overcoming the device interoperability problem from standardization of the workflows to the development of the Device Interoperability Middleware (DIM) [6] has shown a significant potential to solve the interoperability problem.

Considering the safety-critical nature of the medicine domain, ensuring the correct functionality of DIM is very important. In particular, if the middleware fails to translate the data from one communication standard to another standard, then this may lead to false results and hence false diagnostic reports of the patients will be produced which is extremely undesirable. Therefore, these medical procedures are considered critical, since faults and errors in the medical system may lead to loss of lives, and in the best cases, loss of money and reputations [16]. Traditionally, the functionality of a middleware is checked by software testing. However, given the enormous number of possible scenarios in these DIM, they cannot be exhaustively tested due to computational power and memory constraints. Thus, the quality of DIM is judged based on a set of test vectors. This kind of incomplete testing of DIM can have serious consequences, including human deaths.

To overcome the above-mentioned inaccuracy limitations of simulations, formal methods have been proposed as a viable solution[19]. They are primarily based on computer-based mathematical analysis methods to model and analyse the given system. A lot of work has been done in the domain of analyzing heathcare systems using formal methods. Some notable examples include the verification of electrocardiogram (ECG) biosensors in event-B [5, 3]. The work is then extended to formalize the rules that reflects the construction of the ECG wave specifications [4, 21]. In addition, reliability analysis of FHIR standard based e-health system was addressed in [34]. Other works include the verification of software components in medical devices [36, 41], ambient assisted systems [24] or healthcare requirements [1] and the verification of collaborative and agent based workflows in healthcare [10, 29]. A formal model for e-Healthcare readiness assessment was also proposed in [38]. Similarly, formal methods have also been used for the verification of system engineering lifecycle where the Communication Sequential Processes (CSP) have been adopted as a formal method language with an aim to formalize the system specifications [33]. Other work related to managing workflow was presented in [32] and [12].

Probabilistic model checking technique, which is a sub domain of formal methods, has also been used for the verification of the healthcare systems that exhibit probabilistic behaviour, such as modeling and verification of the treatment therapies of Tuberculosis and HIV [35]. Some other model based reliability analysis of the systems include the verification and reliability analysis of the software used in medical devices for infusion pump [22]. Moreover, some generic test cases have also been generated for healthcare systems using model based testing [31]. Despite the above-mentioned formal methods work in ascertaining the correctness of healthcare systems, their usage for analyzing the functionality and performance of healthcare systems, like HIS, has been very rare. Similarly, to the best of our knowledge, formal methods have never been used to assess the recently proposed DIM based HIS system.

Given the safety-critical nature of the DIM, it is a dire need to assess its functionality, reliability and performance using formal methods. As a first step towards this direction, we propose to conduct the reliability analysis of DIM using probabilistic model checking. The usage of a probabilistic model checker allows us to capture the natural randomness found in the DIM models. The considered DIM has been developed as a middleware to integrate various medical devices that run on different communication mediums, including serial port, Wifi and bluetooth. This DIM has been installed in different hospitals of Pakistan and it enables automatic up-gradation of the medical systems by adding any new medical device that runs on either of the three communication mediums.

In this paper, we aim to develop a Markovian model of simple DIM based and fault tolerant based DIM medical systems in the language of the PRISM model checker [30], to analyse the reliability and performance of the respective systems. In particular, we use the Markov Decision Processes (MDP) [40] in PRISM to find the probability of occurrence of wrong results (failures) in the considered system having DIM installed. Moreover, we also use Continuous Time Markov Chain (CTMC) [15] to model the real-time workflow of the medical system and evaluate the real-time failure probabilities. The proposed approach provides more accurate results than traditional counterparts due to the exhaustive exploration of a state-based model of the DIM based medical system and allow the designers to find the failures and weaknesses in the underlying system and to make appropriate measures on the basis of these results, in order to make the system more reliable. In addition, this

works extend our previous work in [39] by providing reliability improvement strategies based on probabilistic analysis method conducted in this work. The presented strategies are expected to enhance the probability of success for several workfow operations.

The rest of paper is organized as follows: Section 2 describes some preliminaries about model checking and PRISM to facilitate the understanding of the paper. The considered health information system along with its reliability analysis is described in Section 3. This is followed by the reliability analysis of two versions of the MDP and CTMC based models of the considered health information system in Sections 4 and 5, respectively. Finally, Section 6 concludes the paper.

2. Probabilistic Model Checking and PRISM. Model checking [2] is used to model and verify the systems that exhibit time or decision based behaviour. Some of the examples of these systems include communication protocols and controllers of digital circuits. The given system is first modeled with a finite-state Markovian state machine and the required verification specifications are defined as system properties, which are expressed in temporal logic. The state machine along with its defined properties are then implemented in a model checking tool that verifies either the given properties hold for the system or not. Moreover, if the properties do not hold, the tool also provides the error traces. Based on the size of the given system, the corresponding size of the state machine may also vary i.e., for a large system, its corresponding state machine will also be large. Therefore, for very large systems, the state machines also grow quite large and thus, its verification become impossible with limited resources of memory and time. This problem is termed as the state-space explosion problem and is usually resolved by developing less complex, abstract models, of the system to facilitate analyses. Moreover, to enhance the memory and computational handling power of the model checking tool, several other symbolic and bounded model checking techniques have also been proposed.

Probabilistic model checking [26] is a special branch of model checking that is precisely used for the verification of the systems that exhibit probabilistic behaviour. The properties verified against these systems are also probabilistic. Many probabilistic model checking tools, such as ETMCC [17], VESTA [25], PRISM [30], MRMC [23] and YMER [18], have been proposed and each has its own pros and cons. Among these tools, PRISM best suits our work as it supports the verification of the steady-state probabilities and is also efficient in terms of memory consumptions, whereas YMER and VESTA are less efficient and do not support the verification of steady-state probabilities [27]. PRISM also supports a wide range of models, such as Discrete Time Markov Chain (DTMC), Continuous Time Markov Chain (CTMC) and Markov Decision Process (MDP) and thus has been selected for our work for analysing the reliability and performance of fault tolerant based DIM HIS system [30].

PRISM model checker has its own modeling language, i.e., the PRISM language, in which the underlying system is modeled. A system may have multiple modules. A state at a given time is represented by local variables, which are defined in those modules whereas, the values of all the local variables of those modules represent the overall state of the system. Modules contains a number of instructions and each instruction has its own guarded commands, which defines the behaviour of the system. PRIMS supports various kinds of properties specifications, such as PCTL, LTL and CSL. $S_{\geq 0.99}[\text{“}normal\text{”}]$ is the steady state probability of *normal* state ≥ 0.99. PRISM also supports verification and analysis of time based properties which we use for the time based analysis of Markovian models. These properties are analyzed by associating a certain reward with each state of the model through a reward structure.

2.1. Markov Decision Process (MDP). MDP [8] based modelling is used for the systems where the behaviour of the system changes on the basis of certain decisions. Eac transition in MDP, i.e., from state $\boldsymbol{S}$ to a state $\boldsymbol{S'}$ is based on a probabilistic decision and depends on the present state $\boldsymbol{S}$ of the system. Mathematically, the equation that is used to find the transitional probability of the transition form state $\boldsymbol{S}$ to state $\boldsymbol{S'}$ is represented below,

$$P_a(S, S') = P_r(S_{t+1} = S' | S_t = S, a_t = a) \tag{2.1}$$

where P_r defines the transition probability and $\boldsymbol{a}$ is the action performed by the decision maker. Similarly, the mathematical equation that expresses the system is termed as Transition Probability Matrix $\boldsymbol{P}$ [11], which represents various transition rates from one state to other state. Similarly, the mathematical equation that is

used to calculate the probability of next state is given below

$$P_r(S') = P_r(S) * P \tag{2.2}$$

MDPs are used to evaluate the systems whose behaviour depends on transitional decisions. The corresponding properties of such systems are defined in terms of probability of failures and success and the model and properties are expressed in the language of the PRISM model checker. The PRISM model checker can then be used to verifiy the properties against the system and calculate the overall probabilities of success and failures.

2.2. Continuous Time Markov Chain (CTMC). CTMC [15] models are used for the mathematical modeling of the workflows, in which each event of the workflow is continuous with respect to the time. A CTMC model includes the total number of states $\boldsymbol{S}$, initial probability distribution of states and the transition rate matrix $\boldsymbol{Q}$. The next transition state probabilities are calculated in the CTMC as follows:

$$P'_t = P_t * Q \tag{2.3}$$

Once the given system is modeled with CTMC and is implemented in PRISM, the reliability *properties* are defined according to the needs and are verified to find the results.

2.3. Discrete Time Markov Chain (DTMC). DTMC [37] are used for the mathematical modeling of the workflows in which each event of the workflow is discrete with respect to time. The DTMC also includes the total number of states $\boldsymbol{S}$, initial probability distribution of states and the transition rate matrix $\boldsymbol{P}$, just like in case of CTMC. The next transition state probabilities are calculated in the DTMC as follows:

$$P'_t = P_t * P \tag{2.4}$$

For the reliability analysis of the workflows that exhibit discrete transitional events w.r.t. time, the given system is modeled with DTMC and the intended properties are verified in PRISM.

3. Reliability analysis of a Typical Health Information System. The reliability of a typical Health Information System (HIS) is expected to increase when the DIM is used as a middleware to overcome Device Interoperability problem. To observe this increase in reliability, we first present a manual HIS system, as depicted in the Fig 3.1. which is typically found in the hospitals and evaluate its reliability by using the proposed model checking approach. This system provides the means of communication between various stake holders.

As presented in the figure, when a patient visits the doctor, the doctor examines the patient and refers him to the medical lab in order to undergo medical tests, such as blood test, urine test and glucose test. The lab collects the information of the patient, including his blood sample and generates his bar code. The blood sample is then fed in the medical device which performs the medical test. However, if the blood sample is found to be clotted or of low quantity, the patient's request is rejected. Upon successful completion of the medical test of the blood sample, the test reports are given to the person in charge of delivering them to a pathologist. During this process, the medical device may fail to perform the tests due to some hardware or software failures. Similarly, the reports may get lost while being delivered to the pathologist by the person in charge. Finally, the reports will be delivered to the patient after being successfully examined by pathologist.

The behaviour of the underlying medical system is probabilistic due to the fact that the workflow transitions occur with some probabilities. In order to analyse the reliability of the overall medical system, which is the probability of successful delivery of the medical reports to the patient by the pathologist, the medical system is modeled with MDP. It allows probabilistic decisions by including the appropriate state transition probabilities. The Markov Chain (MC) of the underlying health system is presented in Fig 3.2 and its transitional probabilities, which present the failure and successful probabilities of the transitional events, are depicted in the Table 3.1. These transitional probabilities have been taken based on the statistics reported in [20] [13], and the probability of the reports being lost by the person in charge is considered to be 0.4.

By using the transition probabilities, as mentioned in Table 3.1, we find the probability of success and failure of all the undergoing transitional events of the considered medical system by verifying the properties

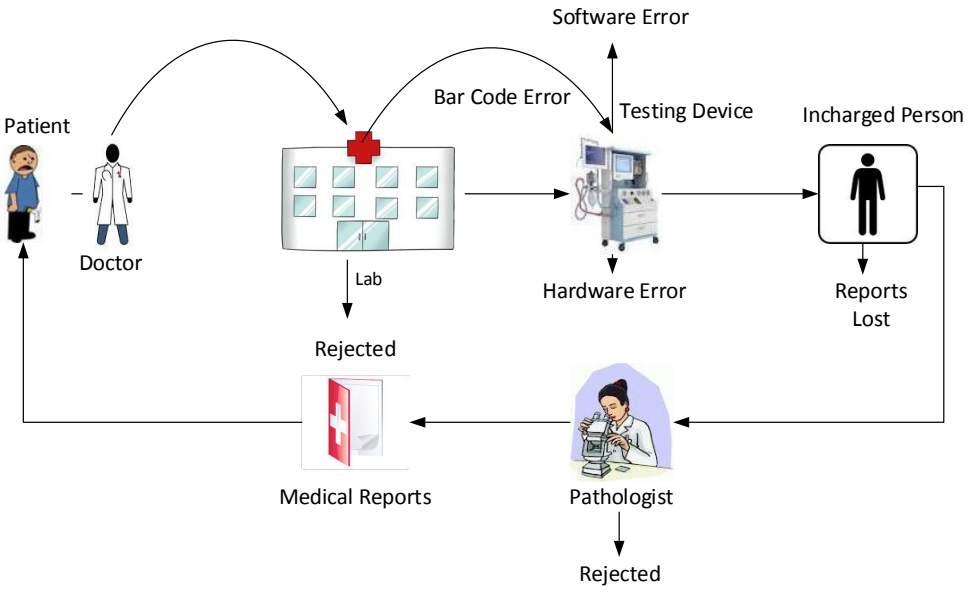


FIG. 3.1. *A Typical Health Information System (HIS)*

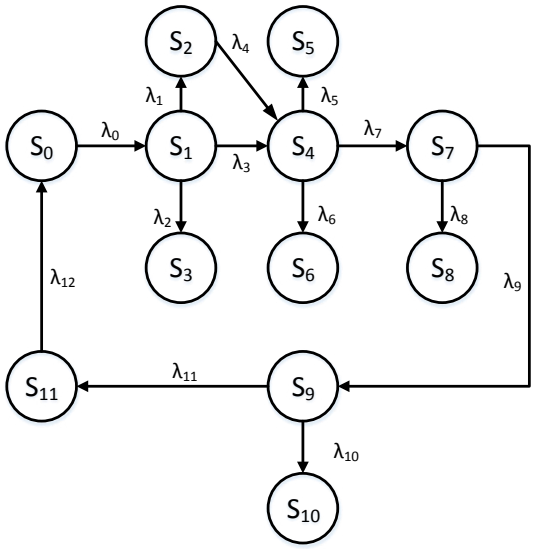


FIG. 3.2. *State Machine of the manual HIS system*

mentioned in Eq 3.1 and 3.2, and the results are presented in the Table 3.2. These results show that the probability of the successful delivery of the medical reports to the patient by the pathologist is 0.41777.

$$Pmax = ?[F\ succ = 1] \tag{3.1}$$

$$Pmax = ?[F\ fail = 1] \tag{3.2}$$

TABLE 3.1
Transitional Probabilities

State Transitions	Probabilities	State Transitions	Probabilities
λ_0	1.0	λ_7	1-λ_5-λ_6=0.93
λ_1	0.02826	λ_8	0.4
λ_2	0.00174	λ_9	1-λ_8=0.6
λ_3	1-λ_1-λ_2=0.97	λ_{10}	0.25
λ_4	1.0	λ_{11}	0.75
λ_5	0.035	λ_{12}	1.0
λ_6	0.035		

TABLE 3.2
Probability of Success and Failure

Lab Rejection	0.002	Human Error	0.637
Bar Code Error	0.047	Human Success	0.557
Device Hardware Error	0.060	Pathologist Rejection	0.239
Device Software Error	0.060	Pathologist Acceptance	0.417
Machine Success	0.928	Successful Delivery	0.417

where ***Pmax*** is the output probability, ***F*** indicates eventually in the future, ***succ*** and ***fail*** are the variables whose values are updated to 1 during the transition from state S_a to S_b. To find the probability of successful medical testing by the machine, the variable ***succ*** is updated during the transition from state S_4 to state S_7. Similarly, to find the probability of failure of delivery of the medical reports to the pathologist by the person in charge, the variable ***fail*** is updated during the transition from state S_7 to S_8. The other probabilities are calculated in the same way.

Reliability analysis of HIS system using MDP allows us to evaluate the general probabilities of success and failure of the underlying system. To increase the depth of evaluation, we use CTMC to model a real-time workflow of the medical system to calculate the probabilistic success as well as failures with respect to time. The CTMC model of the HIS system is just like the MDP model as presented in Fig. 3.2, but with the difference of transitional probabilities. The transitional probabilities are presented in Table 3.3. These probabilities refer to the probability of occurrence of the events with in a time period of 1 hour. For example, the transitional probability λ_0 means that the total number of patients visiting the lab during the time period of 1 hour are 10.

The CTMC model of the above system is implemented in PRISM and the property, as mentioned in Eq 3.3, is verified.

$$P = ? \ [F <= T \ count = K] \tag{3.3}$$

where ***P*** is the output probability, ***F*** indicated future, ***T*** is the time in hours and ***count*** is a variable that acts like a counter which counts the total number of path transitions. This variable is set to count the total number of transitions that occur from state S_9 to S_{11}, which means that the ***count*** variable will count the total number of patient's reports which are successfully delivered to the patients by the pathologist. ***K*** is a variable whose value will be set manually to find the probability of occurrence of ***K*** numbers of count. For testing purposes, the variable ***K*** is set to ***K=1***, and the model is executed for 10 hours, by setting ***F=10*** and results are obtained, as presented in Fig 3.3. From the graph, we conclude that the probability of successful delivery of the medical reports to a single patient by the pathologist during the time period of 1 hour is almost 0.23.

4. Reliability analysis of DIM based Health Information System. After evaluating the reliability of a typical HIS system, we now move on to evaluate the reliability of a DIM based HIS [6] system with the aim to observe the increase in the reliability of the underlying HIS system. The workflow of a DIM based HIS system is presented in the Fig 4.1. This workflow is as same as that of a typical HIS system with the difference that the responsibility of the person, who is in charge of delivering the medical reports to the pathologist, is now

TABLE 3.3
Transitional Probabilities

State Transitions	Probabilities	State Transitions	Probabilities
λ_0	10	λ_7	10-λ_5-λ_6=9.3
λ_1	0.2826	λ_8	4
λ_2	0.0174	λ_9	1-λ_8=6
λ_3	10-λ_1-λ_2=9.7	λ_{10}	2.5
λ_4	10	λ_{11}	7.5
λ_5	0.35	λ_{12}	10
λ_6	0.35		

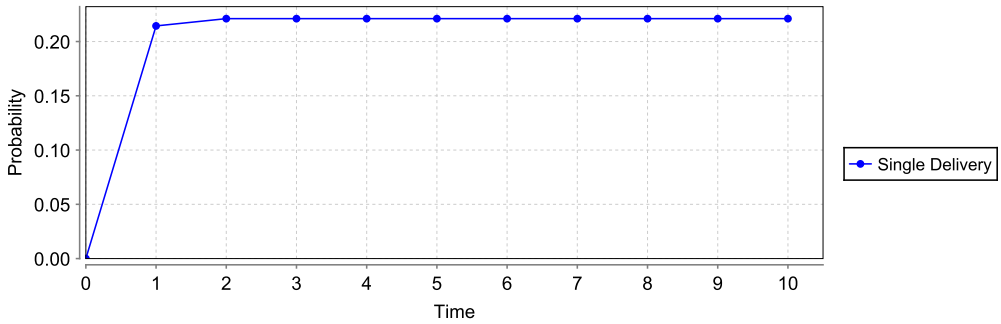


FIG. 3.3. *Probability of successful delivery of a single patient's reports w.r.t time*

performed accurately by the automatic DIM middleware. The DIM in this example is composed of mainly two operations, which include a communication channel and a data mapping. The communication channel, which might be a serial port interface, a WiFi interface or a bluetooth interface, provides the medium to transfer the data taken from the medical machine output to the data mapper. The selection of the communication interface, i.e., serial port, WiFI or bluetooth, depends on the communication standard of the medical machine installed. If the machine has been designed to communicate through a serial port, the serial port interface will be used as a communication channel to communicate the data. Similarly, the wifi interface and the bluetooth interface will be used for the wifi and bluetooth compatible devices, respectively. This automatic selection of the communication interface has significantly resolved the device interoperability problem and thus facilitates new setups as well as easy up-gradation of the existing HIS systems. For example, if the HIS system of a hospital only posses a blood testing medical device, which communicates only by a serial port, then this HIS system can be easily upgraded by adding any new medical device, such as a urine testing medical device, without taking care of the communication standard (i.e., serial port, Wifi, or bluetooth) being followed by the new device. The device mapper finally maps the raw data, as received from the communication channel, to the HL7 standard based diagnostic reports. It has the capability to understand the received raw data regardless of the format of the data i.e., serial port data format, Wifi data format or bluetooth data format. The diagnostic reports are then delivered to the pathologist automatically.

The underlying HIS system is very efficient in terms of automatic reports delivery but it does not guarantee accuracy and perfection. Failures in the system, such as communication failure and data mapping failure, may result into a fatal loss. To avoid these failures, the system must be pre-tested before installation. We propose a reliability evaluation mechanism of the considered system by utilizing MDPs in the proposed methodology. We have developed the MC of the system, presented in the Fig 4.2, and its transitional probabilities are depicted in Table 4.1. These transitional probabilities have been taken based on the statistics reported in [20] [13], whereas the probability of the communication channel failure has been considered to be 0.1.

We verified the reliability of the DIM based HIS system in terms of successful delivery of the diagnostic

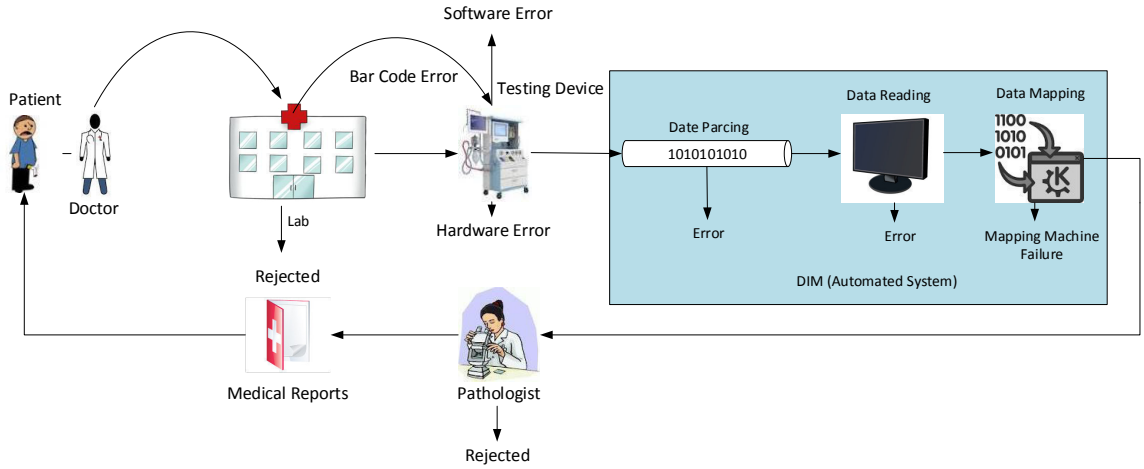


FIG. 4.1. *DIM based HIS System*

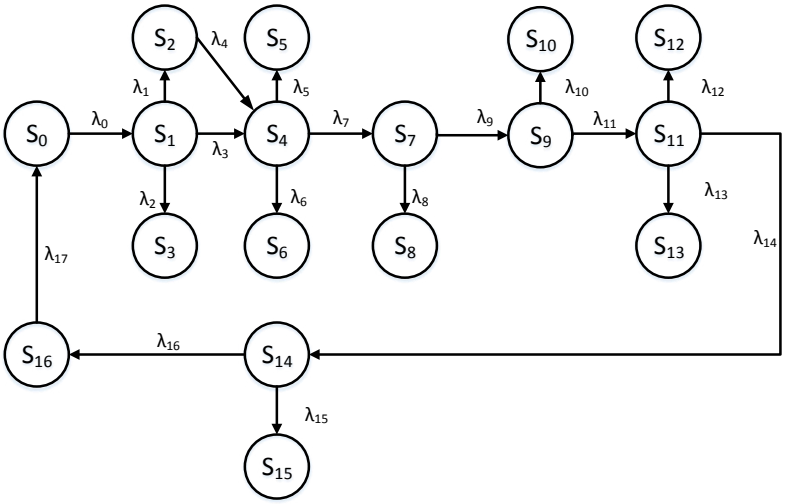


FIG. 4.2. *State Machine of DIM based HIS system*

TABLE 4.1
Transition Probabilities

State Transitions	Probabilities	State Transitions	Probabilities
λ_0	1.0	λ_9	1-λ_8=0.9
λ_1	0.02826	λ_{10}	0.2
λ_2	0.00174	λ_{11}	1-λ_{10}=0.8
λ_3	1-λ_1-λ_2=0.97	λ_{12}	0.065
λ_4	1.0	λ_{13}	0.065
λ_5	0.035	λ_{14}	1-λ_{12}-λ_{13}=0.87
λ_6	0.035	λ_{15}	0.25
λ_7	1-λ_5-λ_6=0.93	λ_{16}	1-λ_{15}=0.75
λ_8	0.1	λ_{17}	1.0

TABLE 4.2
Probability of Success and Failure

Lab Rejection	0.00308	Data Reading Failure	0.29637
Bar Code Error	0.04904	Data Reading Success	0.66843
Device Hardware Error	0.06196	Mapping Machine Error	0.07705
Device Software Error	0.06196	Mapping Algorithm Failure	0.07705
Machine Success	0.92838	Mapping Success	0.58153
DIM Success	0.58153	Pathologist Rejection	0.25784
Communication Failure	0.16465	Pathologist Acceptance	0.43615
Communication Success	0.83554	Successful Delivery	0.43615

TABLE 4.3
Transition Probabilities

State Transitions	Probabilities	State Transitions	Probabilities
λ_0	10	λ_9	10-λ_7=9
λ_1	0.2826	λ_{10}	2
λ_2	0.0174	λ_{11}	10-λ_{10}=9
λ_3	10-λ_1-λ_2=9.7	λ_{12}	0.65
λ_4	10	λ_{13}	0.65
λ_5	0.35	λ_{14}	10-λ_{12}-λ_{13}=8.7
λ_6	0.35	λ_{15}	2.5
λ_7	10-λ_5-λ_6=9.3	λ_{16}	10-λ_{15}=7.5
λ_8	1	λ_{17}	10

report to the patient using PRISM as follows:

$$Pmax = ?[F\ succ = 1] \tag{4.1}$$

The value of the variable ***succ*** is updated to 1 during the transition form state S_{14} to state S_{16}. Similarly, we can find the probabilities of other successful as well as failure transitions by updating the value of the variable **succ** to 1, during those particular transitions. Table 4.2 presents the probability of success as well as failure of all the events in the DIM based HIS. The results indicate that the reliability of the DIM based HIS system, which is the probability of successful delivery of the diagnostic reports to the patient, is 0.43615.

For the reliability evaluation of the real-time workflow of the DIM based HIS system, we developed its CTMC model and analysed the intended property to find the probability of successful delivery of the medical reports to the patient. The CTMC model is just like its MDP model as presented in Fig 4.2, whereas the input transitional probabilities are presented in Table. 4.3. The output results are presented in Fig 4.3. From the graph, we conclude that the probability of successful delivery of the medical reports to a single patient by the pathologist during the time period of 1 hour is almost 0.24.

It has been observed that the MDP as well as CTMC based reliability analysis of the DIM based HIS system is higher than the reliability analysis of the typical HIS system, given in the previous section.

5. Reliability analysis of TMR enabled DIM based Health Information System. Based on the obtained results, as presented in the previous section, the DIM middleware has resulted in increasing the overall reliability of the system, but it has been noticed that this increase in reliability is not significant. Since, the HIS systems are very sensitive and require high accuracy due to the fact that its performance and reliability has a direct effect on the lives of the patients, there is a dire need to increase this reliability up to some acceptable level. For this purpose, we propose some modifications in the DIM system by leveraging upon the strengths of the Triple Modular Redundancy (TMR) mechanism [28].

TMR is a fault tolerating mechanism that is used frequently in safety-critical systems where a single fault in the system may lead to some drastic situations. This mechanism has the capability to tolerate a single

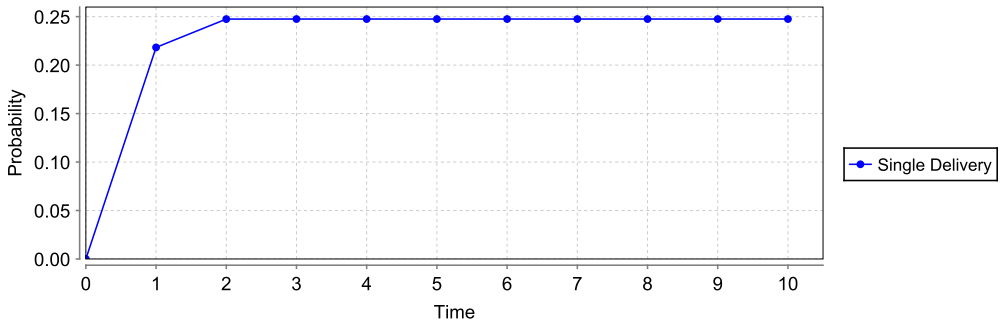


FIG. 4.3. *Probability of successful delivery of a single patient's reports w.r.t time*

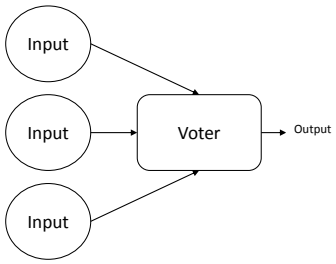


FIG. 5.1. *Triple Modular Redundancy*

fault in the system and thus the reliability of the underlying system increases. A typical TMR mechanism has been presented in Fig 5.1. In TMR, the critical process is performed separately by three identical resources functioning in parallel, whereas the output of all the three resources are fed into a voter. The voter checks these outputs and decides the final output on the basis of a majority voting system. If all the three resources produce the same output, the voter will consider the system flawless and produce the same output, as produced by the resources. However, if any one of the resource produces a different output, due to some unknown fault, in comparison to the remaining two resources, the voter will consider this resource faulty. It will tolerate this fault by masking it and keep the system functioning by producing the output, as produced by the remaining two resources. For the case, when all the resources produce different outputs, the voter will consider the whole system faulty. The MC of a common TMR mechanism is presented in Fig 5.2. In the figure, state 1 shows that all the three resources are functioning properly. State 2 shows that only one system is faulty, however the whole system is still functioning. If any other resource fails from this point, the whole system goes into state $\boldsymbol{F}$. In the figure, the term λ represents the failure rates.

As discussed in the previous section, the communication channel of the DIM middleware can communicate through any of the three communication interfaces i.e., serial port, Wifi and bluetooth. Moreover, only one communication interface is used at a time to communicate the data from medical device output to the data mapping. With the aim to increase the reliability of communication channel in terms of successful communication, we propose to use all the three communication channels in parallel, while using the concepts of TMR mechanism. This proposed modification of the DIM middleware can significantly increases the reliability of the underlying HIS system by enhancing the reliability of the communication channel. This choice would obviously need devices that can communicate via all three communication mediums and is thus the cost of the additional reliability gained. The Markov chain of the modified DIM based HIS system is presented in the Fig 5.3 and its transitional probabilities are presented in the Table 5.1, whereas the failure probabilities of all the communication interfaces are considered to be 0.1.

In Fig 5.3, the state S_4 represents the status of the medical device machine. If the tests are successful, the state machine moves to the state S_7, which indicates that all the three communication channels are functioning

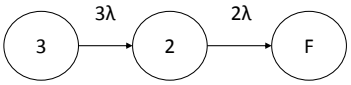


FIG. 5.2. *MC of a TMR mechanism*

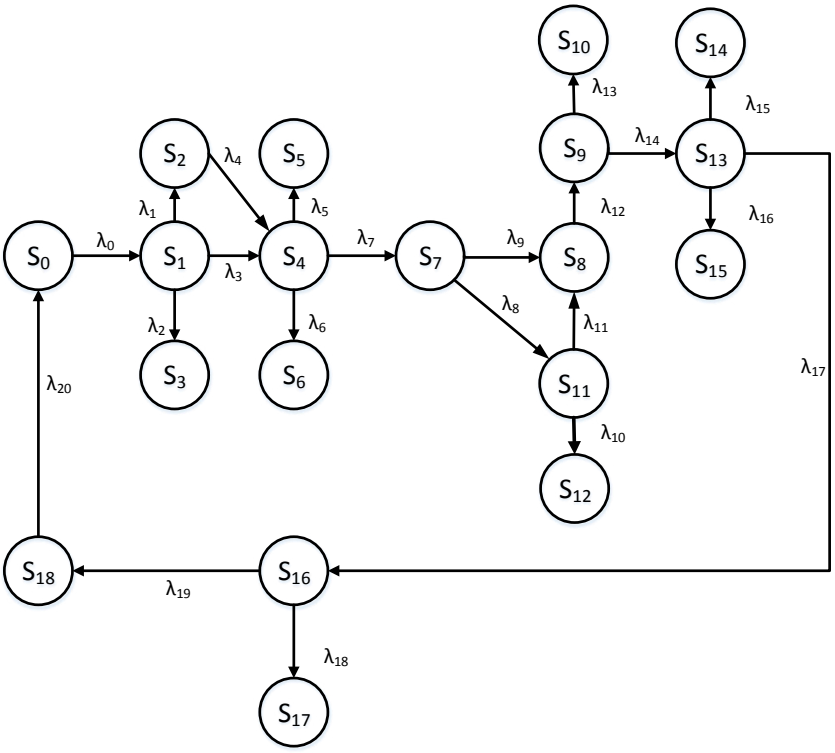


FIG. 5.3. *State Machine of Modified DIM based HIS system*

TABLE 5.1
State Transitional Probabilities

State Transitions	Probabilities	State Transitions	Probabilities
λ_0	1.0	λ_{11}	1-2λ=0.8
λ_1	0.02826	λ_{12}	1.0
λ_2	0.00174	λ_{13}	0.2
λ_3	1-λ_1-λ_2=0.97	λ_{14}	1-λ_{13}=0.8
λ_4	1.0	λ_{15}	0.065
λ_5	0.035	λ_{16}	0.065
λ_6	0.035	λ_{17}	1-λ_{15}-λ_{16}=0.87
λ_7	1-λ_5-λ_6=0.93	λ_{18}	0.25
λ_8	3λ=0.3	λ_{19}	1-λ_{18}=0.75
λ_9	1-λ_8=0.7	λ_{20}	1.0
λ_{10}	2λ=0.2		

TABLE 5.2
Probability of Success and Failure

Lab Rejection	0.00319	Data Reading Failure	0.32056
Bar Code Error	0.0507	Data Reading Success	0.6981
Device Hardware Error	0.06417	Mapping Machine Error	0.08334
Device Software Error	0.06417	Mapping Algorithm Failure	0.08334
Machine Success	0.92838	Mapping Success	0.60738
DIM Success	0.60738	Pathologist Rejection	0.27889
Channel Failure	0.4215	Pathologist Acceptance	0.4555
Communication Failure	0.1023	Successful Delivery	0.4555
Communication Success	0.87263		

TABLE 5.3
State Transitional Probabilities

State Transitions	Probabilities	State Transitions	Probabilities
λ_0	10	λ_{11}	10-2λ=8
λ_1	0.2826	λ_{12}	10
λ_2	0.0174	λ_{13}	2
λ_3	10-λ_1-λ_2=9.7	λ_{14}	10-λ_{13}=8
λ_4	10	λ_{15}	0.65
λ_5	0.35	λ_{16}	0.65
λ_6	0.35	λ_{17}	10-λ_{15}-λ_{16}=8.7
λ_7	10-λ_5-λ_6=9.3	λ_{18}	2.5
λ_8	3λ=3	λ_{19}	10-λ_{18}=7.5
λ_9	10-λ_4-λ_5-λ_6=7	λ_{20}	10
λ_{10}	2λ=2		

successfully. If one of the communication channel fails, the state machine will enter the state S_{11}. Finally, if more than one communication channel fail, the state machine will move to the fail state, which is represented by S_{12}. By using the transitional probabilities, presented in the Table 5.1, we obtained the reliability results which are depicted in the Table 5.2.

For the reliability evaluation of a real-time workflow of the underlying system, we developed its CTMC model and analysed the intended property to find the probability of successful delivery of the medical reports to the patient. The CTMC model is just like its MDP model as presented in Fig 5.3, whereas the input transitional probabilities are presented in Table. 5.3. The output results are presented in Fig. 5.4. From the graph, we conclude that the probability of successful delivery of the medical reports to a single patient by the pathologist during the time period of 1 hour is almost 0.36. These results indicate that the overall reliability of the modified DIM based HIS system has been increased.

6. Reliability Improvement Strategies. As discussed earlier, HIS systems have been increasingly used in many hospitals and are considered to be the fundamental systems of the hospital workflows. Their functionality must be reliable due to the fact that a single fault in the system may lead to a human death. For example, if a blood analyser or a urine testing device produces incorrect results of a particular patient, the results may be mistakenly incorporated as correct results and may eventually cause a serious damage to the patient's health. It is highly recommended to perform the reliability analysis of a workflow as it helps to find the failures and weaknesses in the system and thus some appropriate measures can be taken to incorporate such failures and to increase the overall reliability. In this paper, we conducted the reliability analysis of a typical hospital workflow as well as a DIM based workflow and highlighted the achieved automation and increased reliability in Sections 3 and 4. We further declared in Section 5 that the overall success rate of the workflow can be further increased by adopting the TMR approach within the system and presented the MDP and CTMC based reliability results.

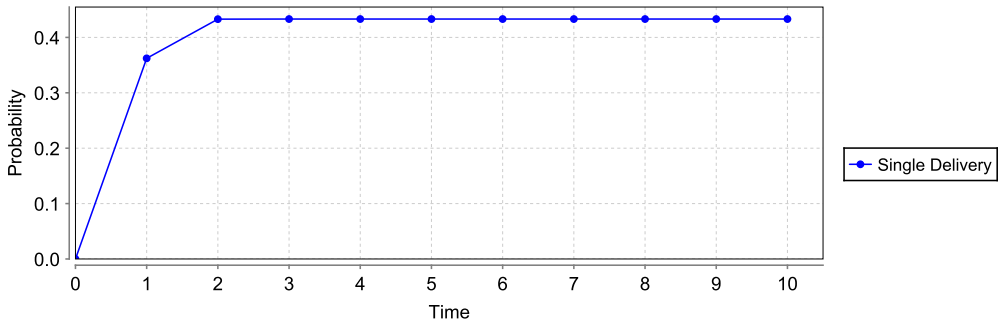


FIG. 5.4. *Probability of successful delivery of a single patient's reports w.r.t time*

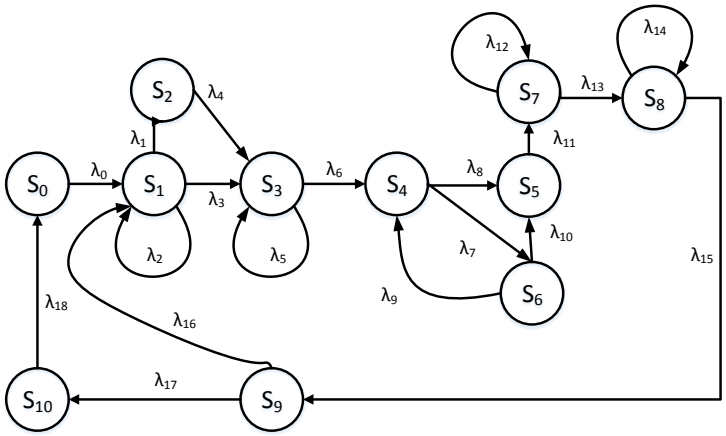


FIG. 6.1. *State Machine*

It has been noted that the overall reliability is not increased up to an acceptable level even after applying TMR approach within the DIM based HIS system and thus there is a dire need to increase this reliability by using some other reliability improvement strategies.

We propose some strategies that have a significant impact on the overall reliability as well as success rate of the hospital workflow and we present a CTMC based reliability analysis of the workflow with these strategies enabled. One of the strategy is to use *verifiers* after every transitional event, such as blood test, urine test, data communication, data reading, data mapping etc., that will verify the correctness of the output result of each event, where a *verifier* can either be an automatic computer system or a human resource. If a *verifier* identifies a mistake, it will notify its event to carry out the process again, and thus the event will eventually produce correct results and will never go into a fail state. Therefore, if all the transitional events produce correct results, the system will never go into a fail state and the overall reliability will greatly increase. Fig 6.1 presents the Markov Chain of TMR enabled DIM based HIS system where no transitional state goes into a fail state due to the applied *verifier*. The other strategy refers to buying highly reliable HIS systems in such a way that their failure rates are very low. For example, if the failure probability of the blood analyzer or urine testing device is very low, it will help to enhance the overall system reliability. Similarly, a highly reliable DIM middleware will greatly improve the overall success rate of the hospital workflow. By using low failure rates, we assume the state transitional probabilities as presented in the Table 6.1, where the failure probability of the communication medium has been assumed to be 0.01. By using these transitional probabilities, we conducted the CTMC based reliability analysis of the workflow, mentioned in the Fig 6.1, and the results are depicted in

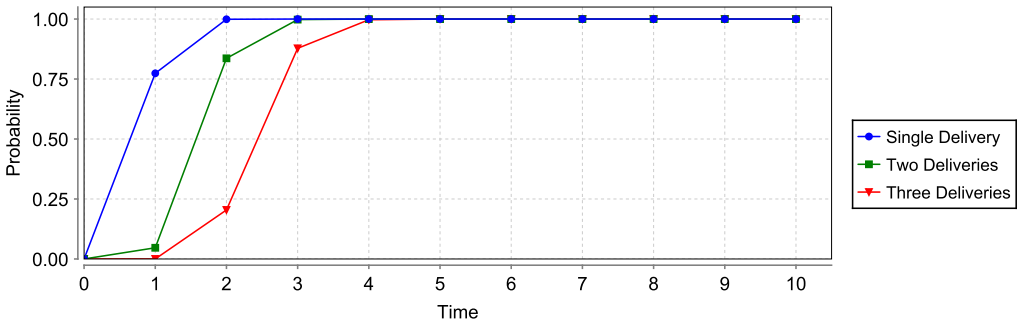


FIG. 6.2. *Probability of Successful Delivery of the Medical Report*

TABLE 6.1
State Transitional Probabilities

State Transitions	Probabilities	State Transitions	Probabilities
λ_0	10	λ_{10}	10-2λ=9.998
λ_1	0.001	λ_{11}	10
λ_2	0.001	λ_{12}	0.002
λ_3	10-λ_1-λ_2=9.998	λ_{13}	10-λ_{12}=9.998
λ_4	10	λ_{14}	0.002
λ_5	0.01	λ_{15}	10-λ_{14}=9.998
λ_6	10-λ_5=9.99	λ_{16}	0.001
λ_7	3λ=0.003	λ_{17}	10-λ_{16}=9.999
λ_8	10-3λ=9.997	λ_{18}	10
λ_9	2λ=0.002		

the Fig 6.2. These results indicate that the overall reliability has been significantly improved. For example, the probability of successfully delivery of the medical reports to a single patient is 0.76 within a time period of 1 hour and this probability increases w.r.t. time, as presented by blue colored graph. Similarly, the probability of successfully delivery of the medical reports to two patients is 0.01 within a time period of 1 hour but this probability considerably increases w.r.t. time, as presented in the green colored graph.

To the best of our knowledge, the underlying DIM based HIS system has not been analysed before using formal methods and the results presented in this paper are accurate and provide detailed information about the system before deployment. On the contrary, the traditional reliability analysis approaches, including numerical methods and simulations, cannot match the rigour and soundness of the results obtained in the presented work.

7. Conclusion. The paper presents a formal reliability analysis of a typical DIM based HIS system using probabilistic model checking technique. The main contribution of the paper includes the MDP and CTMC models development of the traditional HIS system, the DIM based system and the fault tolerant based DIM system and the identification of the corresponding system properties. The analysis is conducted using the PRISM tool and thus the models and properties are implemented for the above-mentioned systems in the language of PRISM. The reliability analysis approach, presented in this paper, was found to be more scalable and accurate compared to the traditional simulation based analysis techniques. We aim to evaluate other DIM middlewares as well using the proposed technique. Similarly, we also aim to find the reliability improvements by using **n** level-redundancy within the DIM HIS system.

REFERENCES

[1] A. Gawanmeh, *An Axiomatic Model for Formal Specification Requirements of Ubiquitous Healthcare Systems*, in Consumer Communications and Networking, IEEE, 2013, pp. 898–902.

[2] W. Ahmad, M. Jongerden, M. Stoelinga, and J. van de Pol, *Model checking and evaluating QoS of batteries in MP-SoC dataflow applications via hybrid automata (extended version)*, Centre for Telematics and Information Technology, University of Twente, 2016.

[3] H. Al-Hamadi, A. Gawanmeh, and M. Al-Qutayri, *Theorem proving verification of privacy in WBSN for healthcare systems*, in International Conference on Electronics, Circuits, and Systems, IEEE, 2013, pp. 100–101.

[4] H. Al Hamadi, A. Gawanmeh, and M. Al-Qutayri, *A Verification Methodology for a Wireless Body Sensor Network Functionality*, in Biomedical and Health Informatics (BHI), 2014, pp. 635–639.

[5] H. Al-Hamadi, A. Gawanmeh, and M. Al-Qutayri, *Formalizing Electrocardiogram (ECG) Signal Behavior in Event-B*, in e-Health Networking, Applications and Services (Healthcom), IEEE, 2014, pp. 55–60.

[6] A.Mahmood, F.Ahmed, K.Latif, H.Mukhtar, and A.Raza, *Middleware for Medical Device Interoperability using Ontology-based Description and Mapping*, Technical Report, National University of Sciences and Technology, Pakistan, (2015). http://semr.seecs.nust.edu.pk/downloads/middleware.pdf.

[7] ASTM E1394-97, *Standard Specification for Transferring Information Between Clinical Instruments and Computer Systems*, 1997. http://www.astm.org/Standards/E1394.htm.

[8] Y. Aviv and A. Pazgal, *A partially observed Markov decision process for dynamic pricing*, Management Science, 51 (2005), pp. 1400–1416.

[9] S. Brown, *Networked Health Information System for Monitoring Food Intake*, Oct. 7 2003. US Patent App. 10/605,548.

[10] C. Bertolini, Z. Liu, M. Schaf, and V.Stolz, *Towards a Formal Integrated Model of Collaborative Healthcare Workflows*, in Foundations of Health Informatics Engineering and Systems, vol. 7151 of LNCS, Springer, 2012, pp. 57–74.

[11] R. Cerqueti, P. Falbo, C. Pelizzari, F. Ricca, and A. Scozzari, *A mixed integer linear program to compress transition probability matrices in Markov chain bootstrapping*, Annals of Operations Research, (2016), pp. 1–25.

[12] O. Choudhury, N. Hazekamp, D. Thain, and S. Emrich, *Accelerating Comparative Genomics Workflows in a Distributed Environment with Optimized Data Partitioning and Workflow Fusion*, Scalable Computing: Practice and Experience, 16 (2015), pp. 53–70.

[13] J. Dale and D. Novis, *Outpatient phlebotomy success and reasons for specimen rejection*, Archives of pathology & laboratory medicine, 126 (2002), pp. 416–9.

[14] A. K. David Chou, *LIS01-A2: Specification for Low-Level Protocol to Transfer Messages Between Clinical Laboratory Instruments and Computer System; Clinical and Laboratory Standards Institute (CLSI)*, 2008. http://shop.clsi.org/automation-documents/LIS01.html.

[15] R. P. Dobrow, *Continuous-Time Markov Chains*, Introduction to Stochastic Processes With R, pp. 265–319.

[16] A. Gawanmeh, H. Al-Hamadi, A. Al-Qutayri, S.-K. Chin, and K. Saleem, *Reliability Analysis of Healthcare Information Systems: State of the Art and Future Directions*, in IEEE International Conference on e-Health Networking, Applications and Services, IEEE, 2015, pp. 68–74.

[17] J. M.-K. H. Hermanns, J. Katoen and M. Siegle, *ETMCC: Model Checking Performability Properties of Markov Chains*, in Dependable Systems and Networks, 2003, pp. 673–673.

[18] H. Younes, *Ymer: A Statistical Model Checker*, in Computer Aided Verification, vol. 3576 of LNCS, Springer, 2005, pp. 429–433.

[19] O. Hasan and S. Tahar, *Formal Verification Methods*, Encyclopedia of Information Science and Technology, IGI Global, (2015), pp. 7162–7170.

[20] R. Hawkins, *Managing the pre-and post-analytical phases of the total testing process*, Annals of laboratory medicine, 32 (2012), pp. 5–16.

[21] Hussam Al-Hamadi, A. Gawanmeh, and M. Al-Qutayri, *Formal Validation of QRS Wave within ECG*, in IEEE Int. Conf. on Information and Communication Technology Research, May 2015, pp. 190–193.

[22] J. F. Groote, A. Osaiweran, and J. H. Wesselius, *Analyzing the Effects of Formal Methods on the Development of Industrial Control Software*, in Software Maintenance, IEEE, 2011, pp. 467–472.

[23] M. K. J. Katoen and I. Zapreev, *A Markov reward Model Checker*, in *Quantitative Evaluation of Systems*, 2005, pp. 243–244.

[24] K. Benghazi, M. V. Hurtado, M. L. Rodrguez, and M. Noguera, *Applying Formal Verification Techniques to Ambient Assisted Living Systems*, vol. 5872 of LNCS, Springer, 2009, pp. 381–390.

[25] M. V. K. Sen and G. Agha, *VESTA: A Statistical Model-Checker and Analyzer for Probabilistic Systems*, in Quantitative Evaluation of Systems, 2005, pp. 251–252.

[26] M. Kwiatkowska, G. Norman, and D. Parker, *PRISM: probabilistic model checking for performance and reliability analysis*, ACM SIGMETRICS Performance Evaluation Review, 36 (2009), pp. 40–45.

[27] A. Legay, B. Delahaye, and S. Bensalem, *Statistical Model Checking: An Overview*, in Runtime Verification, H. Barringer, Y. Falcone, B. Finkbeiner, K. Havelund, I. Lee, G. Pace, G. Rou, O. Sokolsky, and N. Tillmann, eds., vol. 6418 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2010, pp. 122–135.

[28] R. Lyons and W. Vanderkulk, *The use of triple-modular redundancy to improve computer reliability*, IBM Journal of Research and Development, 6 (1962), pp. 200–209.

[29] M. Hoogendoorn, M. C. Klein, Z. A. Memon, and J. Treur, *Formal Verification of an Agent-Based Support System for Medicine Intake*, 25 (2009), pp. 453–466.

[30] M. Kwiatkowska, G. Norman, and D. Parker, *PRISM 4.0: Verification of Probabilistic Real-time Systems*, in Computer

Aided Verification, vol. 6806 of LNCS, Springer, 2011, pp. 585–591.
[31] M. Vieira, X. Song, G. Matos, S. Storck, R. Tanikella, and B. B. Hasling, *Applying Model-Based Testing to Healthcare Products: Preliminary Experiences*, in Software Engineering, ACM, 2008, pp. 669–672.
[32] E. Marzini, P. Mori, S. Di Bona, D. Guerri, M. Lettere, and L. Ricci, *A tool for managing the X1. V1 platform on the cloud*, Scalable Computing: Practice and Experience, 16 (2015), pp. 103–120.
[33] O. Faust, U. R. Acharya, and T. Tamura, *Formal Design Methods for Reliable Computer-Aided Diagnosis: A Review*, IEEE Revisions in Biomedical Engineering, 5 (2012), pp. 15–28.
[34] U. Pervez, O. Hasan, K. Latif, S. Tahar, A. Gawanmeh, and M. Hamdi, *Formal Reliability Analysis of a Typical FHIR Standard based e-Health System using PRISM*, in e-Health Networking, Applications and Services, IEEE, 2014, pp. 43–48.
[35] R. Jetley, S. PurushothamanIyer, and P. L. Jones, *A Formal Methods Approach to Medical Device Review*, IEEE Computer Journal, 39 (2006), pp. 61–67.
[36] S. M. Babamir and M. Borhani, *Formal Verification of Medical Monitoring Software Using Z Language: A Representative Sample*, Journal of Medical Systems, 36 (2012), pp. 2633–2648.
[37] B. Sericola, *Discrete-Time Markov Chains*, Markov Chains, pp. 1–87.
[38] S.O. Oio, O. Olugbara, G. Ditsa, M. Adigun, and S. Xulu, *Formal Model for e-Healthcare Readiness Assessment in Developing Country Context*, in Innovations in Information Technology, 2007, pp. 41–45.
[39] U. Pervez, A. Mahmood, O. Hasan, K. Latif, and A. Gawanmeh, *Formal Reliability Analysis of Device Interoperability Middleware (DIM) based E-Health System using PRISM*, in International Conference on E-health Networking, Application Services (HealthCom), IEEE, Oct 2015, pp. 108–113.
[40] K. Ugurlu, *Controlled Markov Decision Processes with AVaR Criteria for Unbounded Costs*.
[41] Z. Daw, R. Cleaveland, and M. Vetter, *Formal Verification of Software-Based Medical Devices considering Medical Guide-lines*, Computer Assisted Radiology and Surgery, 9 (2014), pp. 145–53.

Edited by: Kashif Saleem
Received: Feb 1, 2016
Accepted: May 27, 2016

DOI 10.12694/scpe.v17i3.1179
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 17, Number 3, pp. 171–187. http://www.scpe.org

PRAVAH: PARAMETERISED INFORMATION FLOW CONTROL IN E-HEALTH*

CHANDRIKA BHARDWAJ AND SANJIVA PRASAD†

Abstract. We study the problem of enforcing information flow control (IFC) in eHealth systems. IFC mechanisms allow users to control the release and propagation of sensitive information so that confidential information is not observable to unintended principals while collaborating with other legitimate principals. We describe the methodology for modelling the information flow control requirements in a hospital domain using *Pravah*, a parameterised lattice-based IFC framework. The key advantage of using the parameterised security class lattice is greater precision in stating policies, enhanced usability and a reduced overhead in creating security tags. We can then use type-checking to statically verify that user programs do not violate stated security policies when accessing or manipulating data records. We discuss the main issues in designing the parameterised security class lattice.

Key words: Information flow control, Security, Lattice, Program verification, Parameterised security class.

AMS subject classifications. 03G10, 68N17, 68N18, 68P20, 68Q60

1. Introduction. Consider a patient Puja who visits a doctor Divya in a hospital for a consultation. Nurse Neetu performs the preliminary medical examination, and uploads Puja's vital parameters (taken using various medical devices) to the hospital's Electronic Medical Record (EMR) system. Following this, Dr Divya attends to Puja and enters her diagnosis and notes in the EMR, along with instructions for the nurse. During such *healthcare encounters*[1], data are captured and stored in database tables of the hospital's EMR system, as shown in Table 1.1. Tables 1.3, 1.4 & 1.5 list the base types of a small set of the fields that are filled when a patient visits a doctor in the hospital. The data record related to an encounter is an evolving data structure, which is accessed and modified by different principals and spans the duration of the encounter and beyond.

Security of medical records is a major concern, not merely due to legislative requirements [1, 2, 3] but also since security breaches lead to inefficiencies in medical information systems [4]. Ensuring compliance with the safety, security and reliability requirements of healthcare information systems is best achieved by applying state-of-the-art formal techniques [5]. For instance, Pervez et al. use probabilistic model checking techniques to validate device interoperability in HL7-compliant standards [6, 7]. This paper proposes ideas from programming languages and systems, in particular security and type-checking frameworks, to ensure secure information management in healthcare information systems.

Access control mechanisms (ACMs) are the typical approach for ensuring security of electronic medical records. ACMs protect data by ensuring that they may be read or written only by authorised entities. The focus of ACMs is securing identification and managing identities, entitlements and privileges [8]. While sophisticated ACMs address most security requirements, it may still be possible for a trusted principal to either deliberately (or inadvertently) release or propagate sensitive data. For example, a doctor who has accessed his patients' information (using the function `viewAssigndEncounters` in Example 2) may thereafter accidentally upload these records including the patients' personally identifiable information onto a public server, thus violating patients' data confidentiality.

This problem may be addressed by following the seminal *information flow control* framework (IFC) [9, 10]. IFC extends access control by not only regulating who is allowed to access what data but also the subsequent use of the data accessed. Every data item is tagged with a *security class*. The security classes are organized in a *lattice* with respect to which *permitted information flows* are defined. The security policies define which flows of data from one security class to another are permitted and which are prohibited. By program/system analysis techniques such as type-checking [10, 11], it is possible to track and restrict the release of data as they flow through the system [10, 12]. This end-to-end approach is complementary to that of ACMs, which manages

*This work was supported by a research grant (2012-15) from DeitY, Ministry of Communication and Information Technology, Government of India: *Foundations of Scalable and Trustworthy Last-mile Healthcare*.

†Department of Computer Science & Engineering, Indian Institute of Technology Delhi, India 110016, (chandrika@cse.iitd.ac.in, sanjiva@cse.iitd.ac.in).

[1]We use the term "encounter" to denote the meeting of a patient with a medical care professional, which may have a wider connotation than a medical examination.

TABLE 1.1
EncounterRecords: Data stored in a typical encounter

Eid	Pid	Did	Date	Age	BP val	Diagnosis	Doc-Notes	Instructions For Nurse
91	1	5	02/01/15	33	140/90	Hyper-tension	Patient is hypochon-driac	Ensure AHT/ diuretic is taken in prescribed dose
92	2	6	04/01/15	28	80/65	Hypoten-sion	Pain in hands	-

TABLE 1.2
Classification of data using different security types

Field Name	Eid	Pid	Age	BP val	Diagnosis	Doc-Notes	Instruc-tions Fr Nurse
Base Type	int	int	int	int∗int	str	str	str
Simple Security Class	$\bot$	$\bot$	P	P	P	D	P
Dependent Security Class	$\bot$	$\bot$	$P(pid, \bot)$	$P(pid, eid)$	$P(\bot, eid)$	$D(did, pid, eid)$	$P(\top, eid)$

identities/roles and privileges at the point of accessing data.

Not all data entered in a single record belong to the same *security clearance* level. Therefore, we require appropriate IFC mechanisms to be adopted and integrated into the EMR system to prevent data belonging to a high security class (e.g. doctors' confidential notes) from flowing into any data objects belonging to a low security class (e.g., public files and servers). However, a simple security class lattice may be too coarse to adequately enforce the desired security policies in the eHealth domain.

The main contribution of this paper is describing a methodology for designing more precise information flow control mechanisms in a hospital domain. We present a parameterised security class lattice model, ordered by *permissible* flow relations [9], which helps in enforcing security policies typically required in a hospital. Using a simple example of such *lattices*, we illustrate how the parameterised model can capture the security requirements and permitted flows at a *fine-grained level*. The design of the security classes and the choice of parameters is not trivial, and the lattice has to be carefully designed to permit desired flows while precluding prohibited ones. Our parameterised framework, *Pravah*, is able to account for different relationships, roles and administrative structures obtained in a hospital. This paper extends our earlier work [13] where we used a prototypical hospital scenario to illustrate some of the principles involved in designing the lattice; in particular, we have refined the security class for doctors and have considered more complex lattice structures. We discuss how the lattice may be modified when introducing new classes of principals, to whom restricted or aggregated versions of the data records are to be released.

Subsequent to this introduction, we present in Sect. 2 a simple information flow control framework and motivate the need for parameterisation which supports more precise information flow control policies. In Sect. 3, using a simple example, we illustrate the parametric framework along with detailed explanations of the security lattice, and a discussion of some of the factors governing its design. In Sect. 4, we use the dependent-type IFC techniques of [11] to present some simple type-checked programs that guarantee compliance with the example policies. Sect. 5 presents the extensions to the system to include principals such as nurses and researchers. Finally, we conclude in Sect. 6, discussing related and future work.

2. Information Flow Control in Hospitals. Information flow analysis requires classifying information into different security classes, which are ordered in a pre-order. We start with a simple linear order $\bot < U < P < D < \top$, which expresses the following notion (Fig. 5.1). $\bot$ represents the most permissive class, i.e., public information; U represents information that is accessible to all registered users in a hospital EMR, and P represents all data which are patient-related. D represents the security class in which the information observable to only doctors belongs, while $\top$ is the most restrictive security class (which might even be data that no one can observe). Information is *allowed to flow* from a lower security class to a higher one, while any flow from a

TABLE 1.3
Users: Database table in hospital EMR

FieldName	Base Type	Simple Sec.Class	Dependent Sec.Class
uid	int	$\bot$	$\bot$
name	str	U	$U(uid)$
dob	str	U	$U(uid)$
gender	str	U	$U(uid)$
phone	int	U	$U(uid)$

TABLE 1.4
EncounterRecords: Database table in hospital EMR

Field Name	Base Type	S.Sec. Class	Depndt Class
eid	int	$\bot$	$\bot$
pid	int	$\bot$	$\bot$
nid	int	$\bot$	$\bot$
date	str	P	$P(pid, eid)$
gps	str	P	$P(pid, eid)$
prvsMI	int	P	$P(pid, \top)$
bp_val	int∗int	P	$P(pid, eid)$
medkitid	int	P	$P(\bot, eid)$

TABLE 1.5
Diagnosis: Database table in hospital EMR

Field Name	Base Type	S.Sec. Class	Depndt Class
did	int	$\bot$	$\bot$
pid	int	$\bot$	$\bot$
eid	int	$\bot$	$\bot$
doc_conf	str	D	$D(did, pid, eid)$
prescriptn	str	P	$P(pid, eid)$
instruct-nurse	str	P	$P(\top, eid)$

higher to a lower class is a violation. These security classes allow categorisation of information captured in an encounter according to their confidentiality requirements (see "*simple security classes*" in Tables 1.2, 1.3, 1.4 & 1.5) and enable hospital administrators to specify and enforce security policies such as "*Doctors' confidential notes should not be observable to anyone except doctors*" or "*All patient-specific data captured in an encounter should not be leaked to a public server*".

However, this coarse security lattice does not allow one to specify policies that are precise enough to protect the confidentiality of information of one patient vis-a-vis another. For example, any patient in the hospital who has at least the security level P is allowed to call the function `viewPatientEncounters` (Example 1)[2] for any patient id, thus violating other patients' data confidentiality. Similarly, any doctor in the hospital with security level D can call a function `viewAssigndEncounters` (Example 2) for any doctor id and can not only see clinical and personally identifiable information (PII) of any patient captured in an encounter but can also read any other doctor's confidential notes. The code fragments presented in this paper are written in an ML-like functional language with side effects.

Example 1 Function `viewPatientEncounters` retrieves the list of encounters related to a patient id, from collection `EncounterRecords`.

```
let viewPatientEncounters = λ (pid_a).
  foreach(x in !EncounterRecords) with y = {}
    do let enc = !x in
      if(enc.pid == pid_a) then enc::y else y
```

Example 2 Function `viewAssignedEncounters` retrieves the list of encounters (of various patients) assigned to a doctor id, by simulating a join between collections `EncounterRecords` and `Diagnosis`.

```
let viewAssigndEncounters = λ (uidd).
  (foreach(x in !Diagnosis) with res_x = {} do
    let docnote = !x
    in if(docnote.did == uidd) then
```

[2]Examples are written in a functional language with imperative features as specified in [11] since we type-check our examples using their software.

```
  (foreach (y in !EncounterRecords) with res_y = {} do
  let tuple_enc = !y
  in if(tuple_enc.pid == docnote.pid and tuple_enc.eid == docnote.eid)
then tuple_enc::res_y else res_y )
else res_x )
```

To prevent such confidentiality breaches, hospital administrators should specify and follow policies such as:

P1 a *registered user's* information is observable only to herself but not to other users;

P2 all *patient-specific* information (such as any diagnosis, her name, address, blood-pressure value etc.) is observable to only the patient concerned and all the doctors who are assigned to her; and

P3 a doctor's *confidential notes* are observable to only the doctor herself.

To express such precise policies and enforce them in programs which compute on the sensitive encounter data retrieved from hospital EMRs, we refine the security lattice using the idea of dependent types from [14, 11] and *statically type check the code* for policy violations. The dependent type framework permits security classes to be parameterised over data values encountered at run time. We can therefore formulate permissible flows that are indexed by the data values present in the data records.

Using dependent security types, the security class U is *refined* to $U(uid)$, by splitting the previous class U into n compartments (assuming n users). $U(1)$ now represents the security class of the user with $uid = 1$ and is incomparable with $U(i)$, $\forall i \neq 1$. Each parameter value serves as a selector for information related to that value. By incomparable, we mean that neither $U(j) \leq U(i)$ nor $U(i) \leq U(j)$ holds, $\forall i \neq j$. Apart from the *bona fide* parameter values, two additional fictional parameter values are introduced: $\bot$ and $\top$, where $U(\bot) < U(i)$ and $U(i) < U(\top)$ for any i, refining the original class U into a small lattice. When we don't care about the specific value of parameter, we use index $\bot$: $U(\bot)$ captures the idea of information accessible to any user. When we want to over-approximate on a parameter, we use $\top$: $U(\top)$ is used for e.g., aggregation of information from many (or all) users. Classes indexed with $\bot$ and $\top$ act as connectors in the flow lattice and usually facilitate checking of permissible flows. Note that if we use $\top$ to over-approximate any parameter in a parameterised security class then the flows from that class have to be carefully regulated when designing the lattice to prevent policy violations. While it is possible to develop a more elaborate and fine-grained parameterisation scheme (e.g., indexed with each subset of principals), the flatter and more succinct lattices presented here suffice for most scenarios and policies encountered in the eHealth domain.

Similarly, we index the security classes P and D with patient and doctor ids. Instead of a unary predicate, we choose to parameterise the security class P on both *pid* (patient's *id*) and *eid* (encounter-id), to reflect the functional relationship between a patient and an encounter [13]. In similar fashion, we parameterise the security class D on *pid*, *did* (doctor's *id*) and *eid* to reflect the functional relationship between a doctor, a patient and an encounter. As a result, we get parameterised security classes $P(pid, eid)$ and $D(did, pid, eid)$, where *pid* is patient's *id* and *did* is doctor's *id*. For example, $P(1, 91)$ represents the security class of information related to the patient with $pid = 1$ and the encounter with $eid = 91$ and $D(5, 1, 91)$ represents the security class of information which is related to both the doctor with $id = 5$ and the patient with $id = 1$ and has been captured in the encounter with $eid = 91$. Considering the *eid* while indexing the higher security classes has a twofold advantage: (1) it captures the functional dependency between a patient and an encounter (similarly, between a doctor, a patient and an encounter); (2) it also allows us to capture the joint (though not equal, especially in case of a hospital) readership on the data captured in an encounter, i.e., data $\in P(pid, eid)$ can flow into security class $D(did, pid, eid)$ because doctor (*did*) has been assigned the encounter *eid* of patient (*pid*). These security classes have been used to annotate the types of fields in the database tables of the EMR (see "*dependent security classes*" in Tables 1.2, 1.3, 1.4, & 1.5).

3. The Parameterised Security Class Lattice. To capture the hospital's confidentiality policies, we define a partial order on these new security classes, a minimal example of which is shown in Fig. 3.1. Information is permitted to flow from a lower security class to a higher security class only if a path exists between them in the lattice. Note that it is the policies which essentially determine the structure of the lattice. Observe that the refined classes are not connected in a simply stacked manner with the topmost element of a hitherto lower class connected to the bottom-most element of a hitherto higher class. Several flows are prohibited, in keeping

with the policies. We explain the meaning of the security classes in the lattice below.

3.1. Parameterised Security Class $U(uid)$. This security class allows policy **P1** to be enforced in a hospital EMR.

- $U(uid)$ corresponds to the class of information which is observable to a *registered user* with $id = uid$. For example, if Puja's *uid* is 1 and Priti's *uid* is 2, then Puja's personal data such as her date of birth, medical history, etc. will be tagged with $U(1)$ and these data cannot flow into data objects that are bound to the class $U(2)$. Neither can the data belonging in class $U(1)$ flow into a data object bound to class $U(\bot)$ as they would then be observable by all the *registered users*, which is a violation of policy **P1**.
- $U(\bot)$ represents the security class of data that are observable by all registered users in the hospital, such as public phone numbers of the hospital, the names of specialist doctors, their working hours and consultation fees, etc. The data tagged with $U(\bot)$ class can flow into the data objects bound to higher security classes in the lattice, e.g., $U(1)$ or $U(2)$ (see Fig. 3.1).
- $U(\top)$ represents the security class of data belonging to more than one *registered user* of a hospital, e.g., a list of the names of all the users along with their dates of birth. So, such data should *not be allowed* to flow into lower classes such as $U(1)$, which is observable by patient Puja. Additionally, such data should not be allowed to flow into data objects bound to the $P(\bot, \bot)$ class, which would be observable to all the patients. Such data may only flow into the most restrictive class, i.e. $\top$, from where data cannot flow out.

3.2. Parameterised Security Class $P(pid, eid)$. This security class allows policy **P2** to be enforceable in a hospital EMR.

- $P(pid, eid)$ corresponds to the class of information which is captured in an encounter with $id = eid$ and is specific to a patient with $id = pid$. For instance, information such as Puja's blood-pressure value or the date when the encounter between Puja and Dr Divya took place will be tagged with security class $P(1, 91)$, where 91 is the id of the encounter. These data cannot flow into a data object bound to security class $P(2, 92)$ because data objects tagged with security class $P(2, 92)$ are observable to Priti, and such a flow will violate the policy **P2**. Thus, $P(1, 91) \not\leq P(2, 92)$ in the security lattice (see Fig. 3.1). Similarly, we disallow the flow of information from security class $P(1, 91)$ to the data objects bound to the security class $P(\bot, \bot)$, as those will be observable to all the patients. Security class $P(\bot, \bot)$ is lower than $P(1, 91)$ in the security lattice (see Fig. 3.1).
  The flow of the information from the security class $U(uid)$ to $P(pid, eid)$ is allowed for all eid if $uid = pid$ in the security lattice, i.e., $\forall uid, eid.U(uid) \leq P(uid, eid)$ ensures the policy **P2**.
- $P(\bot, \bot)$ represents the security class of information which is observable to all the patients in a hospital, such as the number for emergencies, the names of specialist doctors, etc. Such information can flow into data objects belonging to higher security classes, e.g., $P(1, \bot)$ or $P(\bot, 91)$ in the lattice (Fig. 3.1).
- $P(pid, \bot)$ represents information which is specific to a patient with $id = pid$ but is not specific to any encounter, e.g., Puja's personally identifiable information (PII). Information from this class can flow into data objects bound to $P(pid, eid)$ but *not* in $P(\bot, \bot)$ as that would then be observable to all patients, violating policies **P1** and **P2**.
- $P(\bot, eid)$ is the security class for information captured in an encounter with $id = eid$ but not specific to any patient, e.g., the id of the medical devices used to measure Puja's vital parameters, *uid* of the nurse who performs the medical examination of Puja, etc. Such information is captured in an encounter record for situations when the doctor wants to confirm the integrity or authenticity of the data that she will use for making decisions and is typically present in metadata received from the machines. Such data can flow into data objects belonging to security class $P(pid, eid)$ but not in $P(\bot, \bot)$, for reasons mentioned above.
- $P(\bot, \top)$ is the security class for information which does *not* belong to any specific patient but is derived from more than one encounter, e.g., the list of identities of medical devices used in a laboratory, etc. In the lattice, $\forall eid, P(\bot, eid) \leq P(\bot, \top)$.
- $P(\top, \bot)$ is the security class for information which has been derived from information belonging to more than one patient but has *not* been captured in any specific encounter. An example of such information

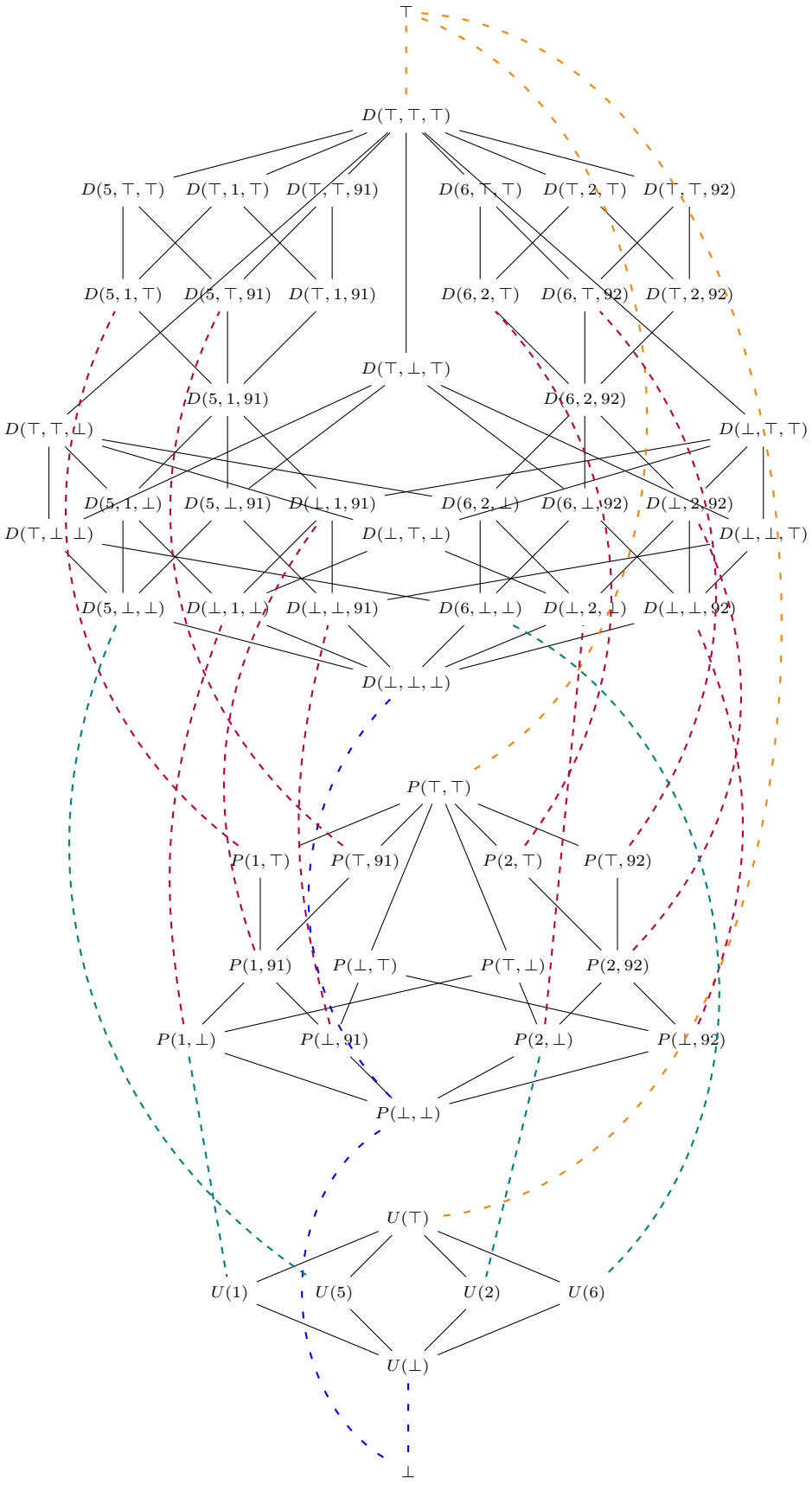


FIG. 3.1. *Minimum example of parameterised security class lattice for hospital domain.*

can be the list of diabetes patients treated in the hospital. In the lattice, $\forall pid, P(pid, \bot) \leq P(\top, \bot)$.

- $P(\top, eid)$ is the security class for the information which has been captured in an encounter with $id = eid$ but may belong to more than one patient. This is a special class, as it is not likely for more than one patient to be involved in the same encounter. As the information from security class $P(pid, eid)$ can flow into the data objects bound to this class, for any pid, this class is more restrictive than $P(pid, eid)$. It is interesting to note that the flow of information from security class $P(\top, \bot)$ to $P(\top, 91)$ is not allowed in the security lattice (Fig. 3.1), which may seem non-intuitive. But if one chooses to allow such a flow, it would enable the flow of information from the security class $P(2, \bot)$ to the security class $D(5, \top, 91)$ and violate the policy **P2**, as the data objects bound to the security class $D(5, \top, 91)$ are observable to the doctors who are not assigned to Priti ($pid = 2$). Therefore, in the security class lattice $\forall eid, P(\top, \bot) \not\leq P(\top, eid)$. Similarly, $\forall pid, P(\bot, \top) \not\leq P(pid, \top)$.
- $P(pid, \top)$ is the security class for information which is derived from data of more than one encounter and belongs to a patient with $id = pid$. This security class is more restrictive than $P(pid, eid)$. For instance, the complete medical profile of Puja can be tagged with security class $P(1, \top)$, which requires higher security than data from a single encounter.
- $P(\top, \top)$ is the security class for the information which consists of data belonging to more than one patient and has been derived from more than one encounter. An example of such information is a list of all the patients treated for a particular disease. Since data objects in this class have information obtained from multiple patient records, therefore, it can only flow to the most restrictive class, i.e., $\top$, from where data is not permitted to flow out.

3.3. Parameterised Security Class $D(did, pid, eid)$. This security class allows policy **P3** to be enforceable in a hospital EMR.

- $D(did, pid, eid)$ corresponds to the class of information captured in an encounter (with $id = eid$) of a patient with $id = pid$ and is specific to a doctor with $id = did$. For instance, information such as Dr Divya's confidential notes about Puja's diagnosis or her mental condition will be tagged with security class $D(5, 1, 91)$, where Dr Divya's uid is 5, Puja's uid is 1 and 91 is the id of the related encounter. These data cannot flow into a data object bound to security class $D(6, 2, 92)$ as it would then be observable to Dr Dipti ($uid = 6$), and such a flow will violate the policy **P2** and **P3**. Thus, $D(5, 1, 91) \not\leq D(6, 2, 92)$ in the security lattice (see Fig. 3.1). These data cannot even flow into a data object bound to security class $D(7, 1, 97)$ as Puja may have visited Dr. Devi (uid=7) only once for cosmetic treatment and such a flow will violate the policy **P3**. Thus, $D(5, 1, 91) \not\leq D(7, 1, 97)$ and $D(5, 1, 91) \not\leq D(7, 1, 91)$ in the security lattice. Similarly, we disallow the flow of information from security class $D(5, 1, 91)$ to the data objects bound to the security class $D(\bot, \bot, \bot)$, as those will be observable to all doctors.
  Note that following hold in the security lattice:
  $\forall pid, did, eid. P(pid, eid) \leq D(did, pid, eid)$. Such a relationship is required to allow the flow of patient-specific information from the security class $P(pid, eid)$ to data objects bound to security class $D(did, pid, eid)$ to ensure the policy **P2**, as the data objects bound to the security class $D(did, pid, eid)$ are observable to the doctor with $id = did$ who is assigned to the patient and needs to see the information captured in an encounter to make clinical decisions.
- $D(\bot, \bot, \bot)$ represents the security class of information which is observable to all the doctors in a hospital, such as the checklist for a surgery etc. Such information can flow into data objects belonging to higher security classes, e.g., $D(5, \bot, \bot)$ or $D(\bot, 1, \bot)$ or $D(\bot, \bot, 91)$ in the lattice (Fig. 3.1).
- $D(did, \bot, \bot)$ represents information which is specific to a doctor with $id = did$ but is not specific to any encounter, e.g., Divya's personal notes. Information from this class can flow into data objects bound to $D(did, \bot, eid)$ and $D(did, pid, \bot)$ but *not* in $D(\bot, \bot, \bot)$ as that class is observable by all doctors and such a flow will violate policy **P3**.
- $D(\bot, pid, eid)$ is the security class for information that is specific to a patient (pid) and is captured in an encounter with $id = eid$ but is not specific to a particular doctor, e.g., Puja's blood pressure value, the id of the medical devices used to measure Puja's vital parameters, etc. Such data can flow into data objects belonging to security class $D(did, pid, eid)$, where did is uid of the doctor who has been

assigned this encounter but not into $D(\bot, \bot, \bot)$, for reasons mentioned above.

- $D(\bot, \top, \top)$ is the security class for information which does *not* belong to any specific doctor but is derived from more than one encounter, e.g., collection of encounter records of patients suffering from HIV. In the lattice, $\forall eid, D(\bot, pid, eid) \leq D(\bot, \top, \top)$.
- $D(\top, \bot, \bot)$ is the security class for information which has been derived from information belonging to more than one doctor but has *not* been captured in any encounter in the hospital, e.g., a discussion amongst doctors which may be about modifying existing checklists and protocols in a hospital. In the lattice, $\forall did, D(did, \bot, \bot) \leq D(\top, \bot, \bot)$.
- $D(\top, pid, eid)$ is the security class for the information which is specific to a patient and has been captured in an encounter with $id = eid$ but may belong to more than one doctor, e.g., the list of opinions of different doctors assigned to a patient. As the information from security class $D(did, pid, eid)$ can flow into the data objects bound to this class, for any did who has been assigned this encounter, this class is more restrictive than $D(did, pid, eid)$. Note that the flow of the information from the security class $D(\top, \bot, \bot)$ to $D(\top, 2, 92)$ is not allowed in the security lattice (Fig. 3.1), which may seem non-intuitive. But if one chooses to allow such a flow, then it would enable the flow of information from the security class $D(5, \bot, \bot)$ to the security class $D(\top, 2, 92)$ which violates the policy **P1** and **P3**, as the data objects bound to the security class $D(\top, 2, 92)$ are observable to the doctors with $uid \neq 5$, i.e., information specific to Dr Divya such as her salary, address and confidential notes will become observable to Dr Dipti. Therefore, in the security class lattice $\forall pid, eid, D(\top, \bot, \bot) \not\leq D(\top, pid, eid)$. Similarly, $\forall did, eid, D(\bot, \top, \bot) \not\leq D(did, \top, eid)$ and $\forall did, pid, D(\bot, \bot, \top) \not\leq D(did, pid, \top)$.
- $D(did, pid, \top)$ is the security class for the information which is derived from the data of more than one encounter of patient (pid) and belongs to a doctor with $id = did$. This security class is more restrictive than $D(did, pid, eid)$. For instance, Divya's notes on the complete medical profile of Puja can be tagged with security class $D(5, 1, \top)$, which requires more security than the notes on data from a single encounter.
- $D(\top, \top, \top)$ is the security class for the information which consists of data belonging to more than one doctor and has been derived from more than one encounter of different patients. Examples of such information can be statistical reports or subjective analyses of patients' diagnosis (& treatment) or doctors' clinical performance, which lead to modifications in the hospital's private protocols. Such information requires a very high security classification in a hospital EMR. Such data cannot flow into any lower security classes as they may be observable to unintended readers. Therefore, it can flow to only the most restrictive class, i.e., $\top$.

TABLE 3.1
Inter-role information flow relations defined for e-Health.

1.	$\forall x, U(x) \rightarrow P(x, \_)$
2.	$\forall x, U(x) \rightarrow D(x, \_, \_)$
3.	$\forall pid, P(pid, \bot) \rightarrow D(\_, pid, \bot)$
4.	$\forall eid, P(\bot, eid) \rightarrow D(\_, \bot, eid)$
5.	$\forall pid, \forall eid, P(pid, eid) \rightarrow D(\_, pid, eid)$
6.	$\forall pid, \forall eid, P(pid, \top) \rightarrow D(\_, pid, \top)$
7.	$\forall pid, \forall eid, P(\top, eid) \rightarrow D(\_, \top, eid)$

4. Typechecked Programs. Our goal is to statically ensure (by typing) the confidentiality of information stored in an EMR. The lattice (Fig. 3.1) described above gives insights into how one can approach *type-checking* the information with security classes in an EMR in a fine-grained manner. In this section we show, with a series of code snippets, how to statically check and enforce the policies (**P1, P2 & P3**) in programs that operate on the data stored in a hospital's EMR, using the parameterised security classes. We use the type-checking framework of [11] to validate the programs. *The dependent types of the programs can be seen as formal proofs of the enforcement of the policies.*

Consider the *dependent security* type annotations in Tables 1.3, 1.4, & 1.5 and the following code snippets annotated with dependent security types:

Example 3 Retrieving encounters of patient with $pid = 2$

```
let viewPatientEncounters = λ(pida: int^⊥) =>
 [ret_type](foreach(x in !EncounterRecords) with y = {}:ret_type do
  let tuple = !x in
  if(tuple.pid == pida) then tuple::y else y)
in let n = 2 in (viewPatientEncounters(n))
```

In Example 3, `EncounterRecords` is a (mutable) collection of references (Table 1.4) and `y` gets the following security type since we are retrieving records with pid value 2: $\sum[pid\colon \bot, eid\colon \bot, nid\colon \bot, date\colon P(2, eid), gps\colon P(2, eid), prvsMI\colon P(2, \top), bp\_val\colon P(2, eid), medkitid\colon P(\bot, eid)]\hat{}\bot$. The presence of security type $P(2, eid)$ ensures that information such as the blood pressure value cannot flow into security classes accessible to principals prohibited by policy **P2**.

Example 4 Retrieving encounters assigned to a doctor.

```
let viewAssignedEncounters = λ(uidd:int^⊥).
 foreach(x in !Diagnosis) with res_x = {} do
  let tuple_doc = !x in if(tuple_doc.did == uidd)
   then foreach(y in !EncounterRecords) with res_y = {} do
    let tuple_enc = !y in
    if(tuple_enc.eid == tuple_doc.eid and tuple_enc.pid == tuple_doc.pid)
    then tuple_enc::res_y else res_y
  else res_x
in let f=first(viewAssignedEncounters(5)) in f
```

In Example 4, function `viewAssigndEncounters()` is used to retrieve the encounters which have been assigned to doctor with $did = 5$ by creating a join between `EncounterRecords` & `Diagnosis` and `f` gets security type $\sum[pid\colon \bot, eid\colon \bot, nid\colon \bot, date\colon P(pid, eid), gps\colon P(pid, eid), prvsMI\colon P(pid, \top), bp\_val\colon P(pid, eid), medkitid\colon P(\bot, eid)]\hat{}\bot$. This list can be accessed only by the assigned doctor because of the permissible flows in the lattice between these patients' parameterised security classes and the doctor's security class.

Example 5 Updating an existing encounter record.

```
let t=first((foreach(x in !EncounterRecords)
 with y={} do let t_enc = !x in
 if(t_enc.pid == 2 and t_enc.eid == 92) then t_enc.bp_val::y else y))
in foreach(x in !EncounterRecords) with _ do
 let t_enc = !x in
  if(t_enc.pid == 2 and t_enc.eid == 92) then let new_rec = [pid=t_enc.pid, eid=t_enc.eid,
    bp_val=t+".00",...]
 in x := new_rec
```

In Example 5, `y` gets security type $P(2, 92)$ since we are retrieving a record with $uid = 2$ and $eid = 92$. To type the record initializing the reference `new_rec`, we need to obtain the type $[pid\colon \bot \times eid\colon \bot \times bp\_val\colon P(2, 92) \times ...]$. But since we know `t` has security level $P(2, 92)$ and $t\_enc.pid = 2$ & $t\_enc.eid = 92$, the assignment $x := new\_rec$ can be deemed secure.

If we change the last conditional to be `if` $t\_enc.pid = 1$, then we would be trying to associate data of security type $P(2, 92)$, value `t`, with the security type $P(1, 92)$ meant for patient with pid 1. Such a flow of information is illegal and hence will not pass type-checking.

Example 6 Function `addFeedbackEncounter` allows a doctor to add her notes/feedback in the `Diagnosis` table.

```
let addFeedbackEncounter = λ(uid_d:int^⊥, pid_d:int^⊥, eid_d:int^⊥).
 foreach(p in viewAssignedEncounters(uid_d)) with _ do
```

```
  if(p.eid == eid_d and p.pid == pid_d) then
  foreach(y in !Diagnosis) with _ do
  let tfeed = !y in
   if(tfeed.eid == p.eid and tfeed.pid == p.pid) then
   let up_rec = [did=tfeed.did,pid=tfeed.pid,eid=tfeed.eid, doc_conf= feedback(p.pid, p.eid, p),...]
   in y := up_rec
in addFeedbackEncounter
```

In Example 6, the types ensure that only the doctor who has been assigned a particular encounter can see the doctor's notes. Function `viewAssignedEncounters` has type $(\prod(uid_d)\colon \bot).F$, where type F is $\sum[pid\colon \bot, eid\colon \bot, nid\colon \bot, date\colon P(pid, eid), gps\colon P(pid, eid), prvsMI\colon P(pid, \top), bp\_val\colon P(pid, eid), medkitid\colon P(\bot, eid)]^*$ (cf. Example 4). As there is no dependency we can refine the function type to be $Int^{\bot} \rightarrow F$ and `p` will have type F. Function `feedback` returns the doctor's personal notes on a particular encounter and has type $\prod u : \bot.\prod e : \bot.\prod r : F.P(u, e)$. So its return type in the call `feedback(p.pid, p.eid, p)` has type $P(p.pid, p.eid)$. Now, to declare the assignment `y:=up_rec` safe, we need to check if `up_rec` has the same type as the type specified for elements of collection `Diagnosis`. So we will need to check if `feedback(p.pid, p.eid, p)` has type $D(tfeed.did, p.pid, p.eid)$.

We know that $p.pid = tfeed.pid$ and $p.eid = tfeed.eid$ and the following holds in the security class lattice (Fig. 3.1): $\forall eid, P(pid, eid) \leq D(\bot, pid, eid)$ and $\forall eid, D(\bot, pid, eid) \leq D(did, pid, eid)$. We can therefore up-classify `feedback(p.pid, p.eid, p)` to security class $D(tfeed.did, p.pid, p.eid)$ where $tfeed.did$ represents the doctor to whom the encounter has been assigned.

Without dependent security types, we would not have fine-grained control over typing and have type $P(\top, \top)$ for `feedback(p.pid, p.eid, p)` due to which
`addFeedbackEncounter` will not type-check in spite of being secure.

5. Extensions to the Lattice. The total order (Fig. 5.1) described in Sect. 2 was minimal in that it considered a very simple ordering of security classes, and only a small class of roles (i.e., doctor and patient) in a hospital encounter. In practice, there are several other kinds of principals involved in a super-speciality hospital. We illustrate the issues encountered when extending the security lattice in different ways.

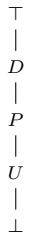


FIG. 5.1. *Total Order for Hospital Domain.*

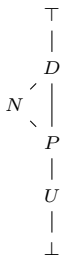


FIG. 5.2. *PreOrder involving Nurses.*

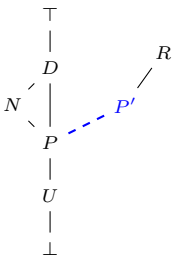


FIG. 5.3. *PreOrder with Nurses and Researchers.*

5.1. Nurses. Nurses play an important healthcare role, especially for in-patients. While most medical data being monitored are accessible to the nurses, a patient may not wish her insurance and payment history [1] to be accessible to them. Likewise, nurses may not want to share their personal information with other nurses, patients, or even doctors. Thus, another security class, i.e., N is added (Fig. 5.2) in the ordering considered in Sect. 2. Security class N represents all the data which are nurse-related.

This security class enables hospital administrators to specify and enforce security policies such as "*Nurses' confidential notes are not observable to patients*" or "*Nurses' personally identifiable information (PII) is not accessible to patients*", as information is not allowed to flow from a higher security class to a lower security class.

Both $P \leq N$ and $P \leq D$ hold in the preorder (Fig. 5.2), due to which all patient related data can flow into security class N. Any nurse in the hospital who has security level N would be able to call a function

TABLE 5.1
Nurse: Database table in hospital EMR

Field Name	Base Type	Sec. Class	Depndt Class
nid	int	$\bot$	$\bot$
qualificatn	str	N	$N(nid, \bot)$
specialisatn	str	N	$N(nid, \bot)$
salary	int	N	$N(nid, \top)$
address	str	N	$N(nid, \top)$
p_review	str	N	$N(nid, \top)$

TABLE 5.2
EncounterRecords: Database table in hospital EMR

Field Name	Base Type	Sec. Class	Depndt Class
eid	int	$\bot$	$\bot$
pid	int	$\bot$	$\bot$
nid	int	$\bot$	$\bot$
date	str	P	$P(pid, eid)$
gps	str	P	$P(pid, eid)$
prvsMI	int	P	$P(pid, \top)$
prvsMTP	int	P	$P(pid, \top)$
bp_val	int∗int	P	$P(pid, eid)$
medkitid	int	P	$P(\bot, eid)$
observatn	str	N	$N(nid, eid)$

`viewAssignedPatients` (Example 7) for any nurse id and can see sensitive information of all the patients, irrespective of the date and time when the patient visited the hospital. As argued earlier, unparameterised security classes do not allow one to specify policies that are precise enough to protect the confidentiality of information. Without parameterisation, the following policies cannot easily be enforced: "*Sensitive information specific to a nurse such as her home address, salary, etc. is not observable by any other nurse or doctor*"or "*A patient's encounter specific information is observable by* only *those nurses who have been assigned the encounter with that patient.*".

To prevent such confidentiality breaches, hospital administrators should specify and follow policies such as:

P2' all *encounter-specific* information of a patient (such as temperature, blood-pressure value, pulse-rate, SpO_2 value, etc.) is observable to only the patient concerned and all the nurses and doctors who have been assigned the encounter with the patient; and

P4 a patient's medical history beyond an encounter is observable to *only* the patient concerned and all the doctors (*not nurses*) who are assigned to her;

P5 a nurse's sensitive information and her confidential notes are observable to her alone.

To express these precise policies and enforce them in programs which compute on the sensitive data retrieved from the hospital EMR, we refine the security class N using both *nid* and *eid*, to reflect the functional relationship between a nurse and an encounter. As a result, we get parameterised security class $N(nid, eid)$, where *nid* is the nurse's id and *eid* the encounter id. For example, $N(3, 91)$ represents the security class of information related to the nurse with $nid = 3$ and the encounter with $eid = 91$. The parameterised security class $N(nid, eid)$ allows policy **P2'**, **P4** and **P5** to be enforceable in the hospital EMR (Fig. 5.4).

5.1.1. Typechecked Programs. We illustrate how parameterised security class $N(nid, eid)$ can enable hospital administrators to enforce security policies **P2'**, **P4**, **P5**.

Example 7 Look up patients' information assigned to a Nurse

```
let viewAssignedPatients = λ(nid_d:int^⊥).
 foreach(x in !EncounterRecords) with res_x={} do
  let t_enc = !x
  in if(t_enc.nid == nid_d) then
     foreach (y in !Users) with res_y={} do
     let t_usr = !y
     in if(t_usr.uid == t_enc.pid) then t_usr::res_y else res_y
    else res_x
in let f = first(viewAssignedPatients(3)) in f
```

In Example 7, function `viewAssignedPatients()` is used to retrieve user-specific details (name, age, gender, etc.) about the patients whose encounter has been assigned to nurse Neetu ($nid = 3$). Here, hospital administrators can enforce policy **P2'** using parameterised security class $N(nid, eid)$, by specifying the following permissible information flow in the lattice: $P(\top, eid) \rightarrow N(\bot, eid)$ (Fig. 5.4). As a result, a nurse can retrieve

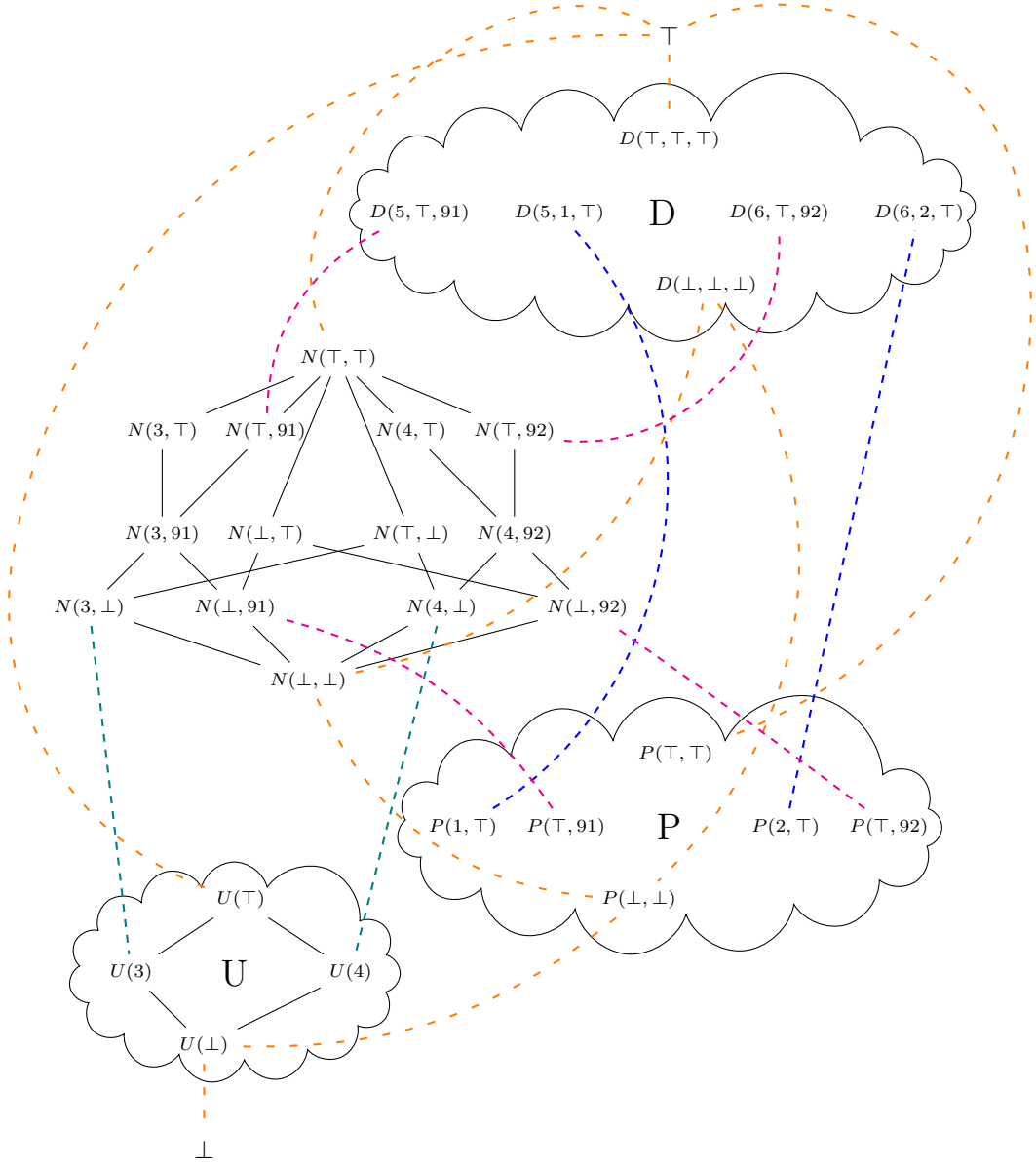


FIG. 5.4. *Example parameterised security class for Nurses.*

the user-specific information of *only* those patients who have been assigned to her by the hospital.

Example 8 Look up performance review specific to a Nurse.

```
let viewReview = λ(uida:int^⊥).
  foreach(x in !Employee) with y = {} do
    let tuple = !x in
      if(tuple.nid == uida) then
        [nid = tuple.nid, p_review = tuple.p_review] :: y
      else y
```

```
in let n = 3 in (viewReview(n))
```

In Example 8, function `viewReview()` is used to view the confidential performance review of nurse Neetu ($nid = 3$). Policy **P5** is enforceable in Examples 8 & 9, if sensitive information of nurses, such as their salary and performance review, is classified in security class $N(nid, \top)$. The return type of function `viewReview()` in Example 8 is $\Sigma[nid : \bot, p\_review : N(3, \top)]\hat{}\bot$ which ensures that the information from performance review of nurse Neetu ($nid = 3$) cannot flow to security classes whose data is observable to other nurses with $nid \neq 3$ or even to any doctor.

Example 9 Look up a salary statement specific to a Nurse.

```
let viewSalary = λ(uida:int^⊥).
  foreach(x in !Employee) with y = {} do
     let tuple = !x in
        if(tuple.nid == uida) then
          [nid = tuple.nid, sal = tuple.sal] :: y
        else y
in let n = 3 in (viewSalary(n))
```

In Example 9, function `viewSalary()` is used to view the salary of nurse Neetu ($nid = 3$). If hospital administrators classify salary in security class $N(nid, \top)$, then the return type of function `viewSalary()` is $\Sigma[nid : \bot, sal : N(3, \top)]\hat{}\bot$ which ensures that no nurse apart from Neetu ($nid = 3$) can access the output of function call `viewSalary(3)`.

Example 10 Look up partial medical history of a Patient.

```
let viewMTPHis = λ(pida:int^⊥).
  foreach (x in !EncounterRecords) with y = {} do
    let t_enc = !x in
      if (t_enc.pid == pida) then
        [pid = t_enc.pid, prvsMTP = t_enc.prvsMTP] :: y
      else y
in let n = 1 in (viewMTPHis(n))
```

In Example 10, function `viewMTPHis()` is used to retrieve the history of MTPs (abortions) of a patient. Patients usually do not want to share such incidents with anyone other than their doctors. Hospital administrators can enforce such security requirements (policy **P4**) by putting such sensitive medical history data in security class $P(pid, \top)$, as these are inaccessible to nurses (Fig. 5.4). The flow $\forall pid, P(pid, \top) \longrightarrow N(\_, eid)$ is *not* allowed in the preorder specifying permissible flows (Fig. 5.4). As a result, the output of function call `viewMTPHis(1)` has type $\Sigma[pid : \bot, prvsMTP : P(1, \top)]\hat{}\bot$, making it inaccessible to any nurse.

Example 11 Look up complete medical history specific to a Patient.

```
let viewCompleteMedHis = λ(pida:int^⊥).
  foreach(x in !EncounterRecords) with y = {} do
     let tuple = !x in
    if(tuple.pid == pida) then
      tuple::y
    else y
in let n = 1 in (viewCompleteMedHis(n))
```

In Example 11, function `viewCompleteMedHis()` is used to retrieve the complete medical history of a patient present in the system. Typically, patients prefer not to share their *complete* medical history with all doctors they meet for consultation and only trust family doctors with their complete history. Hospital administrators need to be able to enforce security policies **P2 & P4** to prevent such information leakage. With appropriate classification of data (see Table 5.2), the return type of function call `viewCompleteMedHis(1)` becomes: $\Sigma[pid : \bot, eid :$

$\bot, nid : \bot, date : P(1, eid), prvsMI : P(1, \top), prvsMTP : P(1, \top), bp\_val : P(1, eid), medkitid : P(\bot, eid)]\hat{}\bot$ which has the patient's medical history including previous Medical Termination of Pregnancy details, classified in security class $P(1, \top)$, making it unobservable to nurses (as explained in the previous example) or any doctor who has not been allowed to observe all encounters of patient Puja ($pid = 1$). Examples 10 & 11 illustrate how parameterised security classes enable hospital administrators to enforce security policy **P4**.

5.2. Researchers. Medicine goes beyond merely treating patients. Medical professionals including doctors, health administrators and researchers require access to large corpora of *bona fide* medical cases for analyses that may help provide better treatments, avoid epidemics, and otherwise improve the delivery of healthcare. For research, authentic details about diagnosis, treatment and prognosis of the patients are also required. But such data sharing between hospitals and researchers would violate most of the security policies discussed till now, i.e., **P1**, **P2**, **P2'**, **P3**, **P4** and **P5**.

Typically, hospitals sign a confidentiality agreement with patients stating that their data will not be given out to third parties without their consent, which is usually taken prior to starting treatment. To protect patient confidentiality, hospitals release medical data for research purposes only after "de-identifying" (anonymising) the patient specific information in a new copy of the data [15]. However, it becomes a non-trivial task for hospital administrators to ensure that the researchers are given access to only certain kinds of information requested, e.g., "*only the medical information which has been captured during a particular time period is shared with a specific researcher*" or "*only the medical information which is related to patients diagnosed with a specific disease is shared with a researcher*". Currently, widely-used practices for this purpose primarily require manual intervention, which is typically error-prone and not sound.

To address this requirement, a security class R is added (Fig. 5.3), in which information observable to only researchers belongs. It is assumed that the hospital administrators will create a copy of patients' data and anonymise these to share with researchers. A trusted function is used to obfuscate all personally identifiable information from this new copy of the medical database. The dashed edge between security class P and security class P' represents that the data flowing from security class P to P' has been anonymised by the trusted function. This security class along with trusted anonymising functions enables hospital administrators to prevent violation of the security policies such as **P1**, **P3**, **P5**, etc.

Any researcher should be able to access all the anonymised information from the new copy of the hospital database. Additionally, a researcher would not like to necessarily share their observations and notes with other users (including other researchers) in the hospital domain. To prevent such confidentiality breaches, hospital administrators need to specify and enforce the following policies:

P6 a researcher can observe *only the non-personally identifiable* information from specific encounters which have been assigned to her,

P7 a researcher's confidential notes and observations are accessible to that researcher *alone*.

To express these precise policies and enforce them in programs which compute on the anonymised sensitive data retrieved from hospital EMRs, we refine the security class P' — which is the same as P except that it is on a different domain, i.e. $P(pid', eid')$, where we have trusted one-way function owf such that:

$$owf(pid) = pid' \text{ and } owf(eid) = eid'.$$

We parameterise the security class R on both rid (researcher's id) and eid' ($owf(eid)$). This gives us security class $R(rid, eid')$ which contains the data specific to some encounter and a researcher. For instance, $R(8, 8475)$ represents the security class of information concerning researcher with $rid = 8$ and the encounter with $eid' = 8475$.

The parameterised security classes $P(pid', eid')$ and $R(rid, eid')$ allow policies **P6** and **P7** to be enforceable in the hospital EMR. These classes allow researchers to access anonymised useful medical data without breaching confidentiality of patient data and disallow researchers from accessing anything below or above the parameterised patient classes.

5.2.1. Typechecked Programs. We illustrate how parameterised security classes $R(rid, eid')$ & $P(pid', eid')$ can enable hospital administrators to enforce security policies **P6** and **P7**.

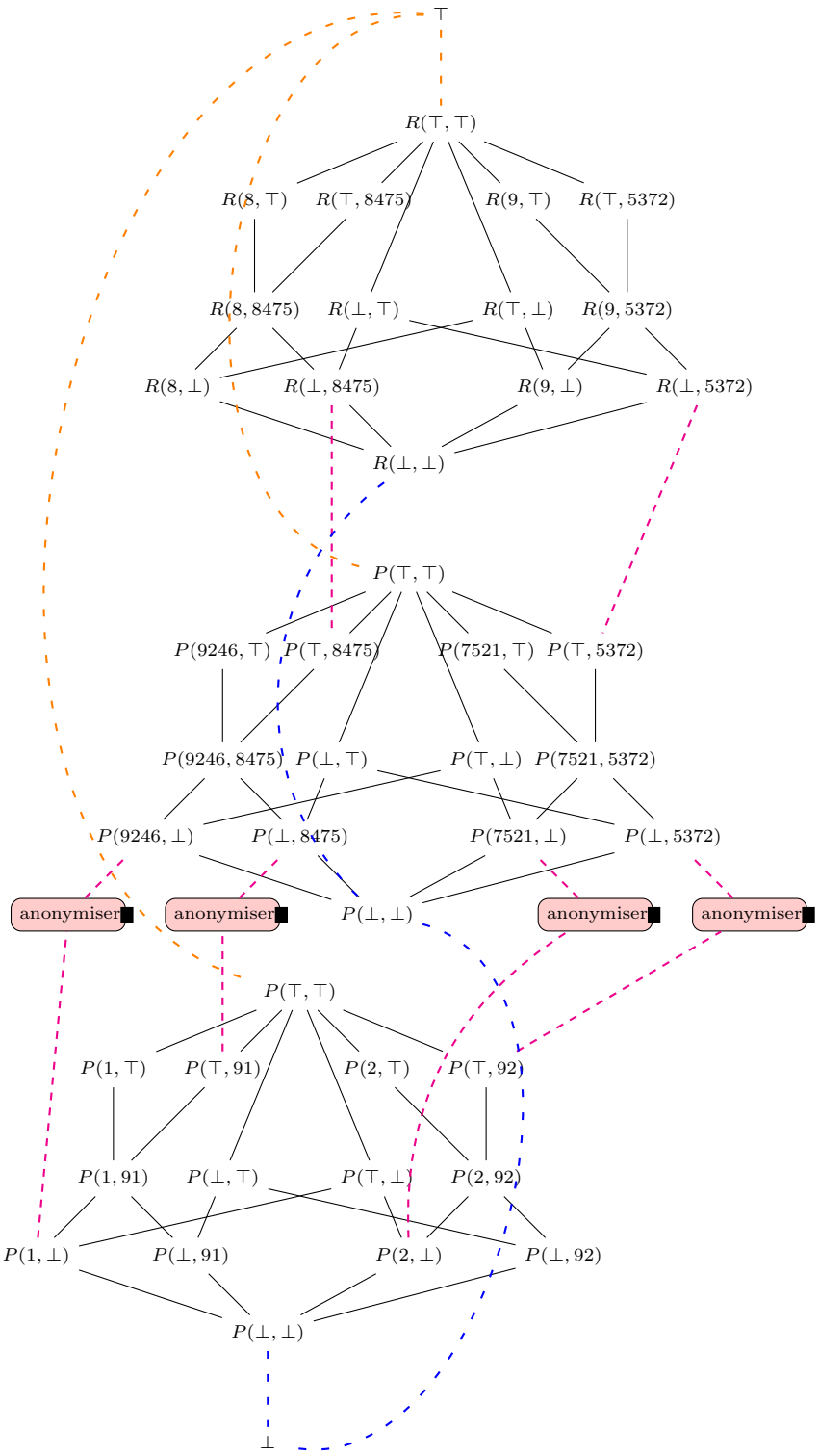


FIG. 5.5. *Example parameterised security classes for Researchers.*

TABLE 5.3
Researcher: Database table in hospital EMR

Field Name	Base Type	Sec. Class	Depndt Class
rid	int	$\bot$	$\bot$
qualificatn	str	R	$R(rid, \bot)$
specialisatn	str	R	$R(rid, \bot)$
salary	int	R	$R(rid, \top)$
address	str	R	$R(rid, \top)$

TABLE 5.4
ResearchRecords: Database table in hospital EMR

Field Name	Base Type	Sec. Class	Depndt Class
eid	int	$\bot$	$\bot$
rid	int	$\bot$	$\bot$
conf_notes	str	R	$R(rid, eid)$
results	str	R	$R(\bot, eid)$

Example 12 Look up data from encounters assigned to Researcher.

```
let viewData = λ(uidr:int^⊥).
  foreach(x in ResearchRecords) with res_x = {}: enc do
    let tuple_r = !x
      in if(tuple_r.rid == uidr ) then
      foreach(y in EncounterRecords) with res_y = {}: enc do
      let tuple_e = !y
        in if(tuple_e.eid == tuple_r.eid) then tuple_e::res_y
           else res_y
     else res_x
in let r = first(viewData(8)) in r
```

In Example 12, function `viewData()` is defined to retrieve all patient-encounter data that are accessible to a researcher by simulating a join between the anonymised copy of collection `EncounterRecords` and the `ResearchRecords` collection. Policy **P6** is enforced in Example 12, as the return type of function call `viewData(8)` is $\Sigma[pid' : \bot, eid' : \bot, nid : \bot, date : P(pid', eid'), gps : P(pid', eid'), bp\_val : P(pid', eid'), medkitid : P(\bot, eid')]\hat{}\bot$ and the output is observable to researcher Richa ($rid = 8$) alone.

Example 13 Look up confidential notes of a Researcher.

```
let viewNotes = λ(uida:int^⊥).
  foreach(x in !ResearchRecords) with y = {} do
     let tuple = !x in
        if(tuple.rid == uida) then
          [rid = tuple.rid, conf_notes = tuple.conf_notes] :: y
        else y
in let n = 8 in (viewNotes(n))
```

In Example 13, function `viewNotes()` is used to view the confidential notes of researcher Richa ($rid = 8$). If hospital administrators classify the confidential notes in security class $R(rid, \top)$, then the return type of function `viewNotes()` is $\Sigma[rid : \bot, conf\_notes : R(8, \top)]\hat{}\bot$ which ensures that no researcher apart from Richa ($rid = 8$) can access the output of function call `viewNotes(8)`. This example illustrates how hospital administrators can enforce policy **P7** and ensure that sensitive information like these confidential notes cannot flow into data objects observable to unauthorised or unintended users in the system.

6. Conclusion. Medical data and metadata require far more sophisticated information management schemes than provided by *access control* mechanisms typically employed in hospital information systems. To ensure the *end-to-end* security/integrity of data, *information flow control* (IFC) mechanisms are required. Earlier work on IFC [9] has been extended in *decentralized* IFC (DIFC) to protect data for different users, each with their individual policy [16] and researchers have proposed an IFC-compliant database in [17]. Naive instantiations of such security lattice frameworks do not provide adequate protection or flexibility in specifying policies. Dependent-types provide a method to index information with specific users (or doctors) and thus provide far more precise and fine-grained control over information flow.

In this paper we have presented a security lattice, and discussed some of the subtleties in the design of

permitted information flows. The examples are simple, but illustrate the principles involved in extending the classification to real HIS systems. In the future, we will explore how the dependent information flow types [11] can be adapted to allow for authorised declassification, along the lines of [16]. Finally, we intend to integrate the systematic tagging of data/metadata [18] into this dependent-type framework, to allow for secure information flow with high usability & minimal overhead in a federated collection of administrative domains where data from different domains are subject to different information flow policies [19].

REFERENCES

[1] U. D. of Health, H. Services, et al., *Summary of the HIPAA privacy rule*, Washington, DC: Department of Health and Human Services, (2003).

[2] M. A. Scholl, K. M. Stine, J. Hash, P. Bowen, L. A. Johnson, C. D. Smith, and D. I. Steinberg, *An introductory resource guide for implementing the health insurance portability and accountability act (HIPAA) security rule*, tech. report, United States, 2008.

[3] U. D. of Health & Human Services et al., *HITECH Act enforcement interim final rule*, US Department of, (2013).

[4] E. Marzini, P. Mori, S. D. Bona, D. Guerri, M. Lettere, and L. Ricci, *A tool for managing the X1.V1 platform on the cloud*, Scalable Computing: Practice and Experience, 16 (2015).

[5] A. Gawanmeh, H. M. N. A. Hamadi, M. Al-Qutayri, S. Chin, and K. Saleem, *Reliability analysis of healthcare information systems: State of the art and future directions*, in HealthCom, 2015, pp. 68–74.

[6] U. Pervez, A. Mahmood, O. Hasan, K. Latif, and A. Gawanmeh, *Formal reliability analysis of device interoperability middleware (DIM) based e-health system using PRISM*, in HealthCom, 2015, pp. 108–113.

[7] U. Pervez, O. Hasan, K. Latif, S. Tahar, A. Gawanmeh, and M. S. Hamdi, *Formal reliability analysis of a typical FHIR standard based e-health system using PRISM*, in Healthcom, 2014, pp. 43–48.

[8] M. Benantar, *Access Control Systems: Security, Identity Management and Trust Models*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[9] D. E. Denning, *A Lattice Model of Secure Information Flow*, Commun. ACM, 19 (1976), pp. 236–243.

[10] D. E. Denning and P. J. Denning, *Certification of Programs for Secure Information Flow*, Commun. ACM, 20 (1977), pp. 504–513.

[11] L. Lourenço and L. Caires, *Dependent Information Flow Types*, in POPL'15, ACM, 2015, pp. 317–328.

[12] M. Krohn, A. Yip, M. Brodsky, N. Cliffer, M. F. Kaashoek, E. Kohler, and R. Morris, *Information Flow Control for Standard OS Abstractions*, in Proceedings of SOSP'07, ACM, 2007, pp. 321–334.

[13] C. Bhardwaj and S. Prasad, *Parametric Information Flow Control in eHealth*, in HealthCom, 2015, pp. 102–107.

[14] P. Martin-Löf, *Intuitionistic Type Theory: Notes by Giovanni Sambin of a series of lectures given in Padova, June 1980*, 1984.

[15] I. C. of Medical Research, *Ethical guidelines for biomedical research on human participants*. `http://icmr.nic.in/ethical_guidelines.pdf`, 2006. Accessed: 2016-05-12.

[16] A. C. Myers and B. Liskov, *A Decentralized Model for Information Flow Control*, SIGOPS Oper. Syst. Rev., 31 (1997), pp. 129–142.

[17] D. A. Schultz and B. Liskov, *IFDB: Decentralized Information Flow Control for Databases*, in EuroSys '13, 2013, pp. 43–56.

[18] C. Bhardwaj, *Systematic Information Flow Control in mHealth Systems*, in Proceedings of Workshop on Networked Healthcare Systems (COMSNETS), 2015, NetHealth'15, IEEE, 2015, pp. 1–6.

[19] S. Prasad, *Designing for scalability and trustworthiness in mhealth systems*, in Proceedings of Distributed Computing and Internet Technology Conference, ICDCIT'15., LNCS 8956, Springer, 2015, pp. 114–133.

Edited by: Amjad Gawanmeh
Received: Feb 1, 2016
Accepted: Jul 3, 2016

DOI 10.12694/scpe.v17i3.1180
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 17, Number 3, pp. 189–205. http://www.scpe.org

ANALYSIS AND VERIFICATION OF XACML POLICIES IN A MEDICAL CLOUD ENVIRONMENT*

MERYEME AYACHE † MOHAMMED ERRADI † AHMED KHOUMSI ‡ AND BERND FREISLEBEN §

Abstract. The connectivity of devices, machines and people via Cloud infrastructure can support collaborations among doctors and specialists from different medical organisations. Such collaborations may lead to data sharing and joint tasks and activities. Hence, the collaborating organisations are responsible for managing and protecting data they share. Therefore, they should define a set of access control policies regulating the exchange of data they own. However, existing Cloud services do not offer tools to analyse these policies. In this paper, we propose a Cloud Policy Verification Service (*CPVS*) for the analysis and the verification of access control policies specified using XACML. The analysis process detects anomalies at two policy levels: a) intra-policy: detects discrepancies between rules within a single security policy (*conflicting rules* and *redundancies*), and b) inter-policies: detects anomalies between several security policies such as *inconsistency* and *similarity*. The verification process consists in verifying the *completeness* property which guarantees that each access request is either accepted or denied by the access control policy. In order to demonstrate the efficiency of our method, we also provide the time and space complexities. Finally, we present the implementation of our method and demonstrate how efficiently our approach can detect policy anomalies.

Key words: Formal Verification, Cloud Computing, XACML Policies, Automata, Completeness, Security Anomaly Detection.

AMS subject classifications. 68Q60, 68Q85, 68M14

1. Introduction.

The use of connected devices (mobiles, sensors, scanners etc.) permits the creation of Electronic Personal Records (EPR) to monitor patients' health states remotely. The EPR of a patient consists of medical histories, diagnoses, medications, immunization dates, etc [11]. A first advantage of an EPR is that it provides accurate, up-to-date, and complete information about patients. Another advantage is that it supports collaborations among different doctors in diverse medical organisations.

Cloud computing offers a suitable platform for such collaborations [1]. For instance, the storage service offered by the Cloud can be considered as a shared pool where medical organisations can store and share their data. A patient's EPR usually contains confidential data and hence each medical organisation needs to define a set of policies to regulate the access to the outsourced data. However, current Cloud solutions do not offer users the ability to define their own policies. To address this issue, we have developed, in previous work, a middleware (denoted *curlX* [6]) that permits the enforcement of users' security policies in *Openstack* [27] (an open source Cloud solution). *curlX* uses XACML [25] (eXtensible Access Control Markup Language) to specify access control policies [7]. Yet, XACML has many limitations in terms of policy anomaly detection [16]. For instance, XACML lacks a mechanism to detect conflicts and redundancies.

In this paper, we present a formal approach based on automata to detect anomalies in XACML policies such as: conflicting rules, redundancies, inconsistencies, and policy similarities; and to verify the completeness property that guarantees that each access request is either accepted or denied by the access control policy. The approach is implemented as a Cloud service denoted *CPVS* (Cloud Policy Verification Service) and integrated into *curlX*. The advantage of the proposed approach is that it detects several discrepancies in the XACML policies using the same formal model, in contrast to other existing approaches which make use of different models to detect distinct anomalies.

The rest of the paper is organised as follows: Section 2 presents related work. In Section 3, we present an overview of XACML policies and automata. Section 4 describes the architecture of *curlX* taking into account the new verification service. In Section 5, we present the procedure to transform XACML policies into automata. Sections 6 and 7 define the approaches to detect anomalies and verify completeness in XACML policies. We calculate the time and space complexities in Section 8. Section 9 discusses the implementation and evaluation of our proposed approach. Finally, Section 10 concludes the paper and outlines areas for future research.

* This work is supported by the BMBF (PMARS Program) and the DAAD (German-Arab Transformation Partnership)
†ENSIAS, Mohammed V University in Rabat, Morocco (meryemeayache@gmail.com; mohamed.erradi@gmail.com)
‡Department of Electrical & Computer Engineering, University of Sherbrooke, Canada (ahmed.khoumsi@usherbrooke.ca)
§Department of Mathematics & Computer Science, Philipps-Universität Marburg, Germany (freisleb@informatik.uni-marburg.de)

2. Related Work. In a collaborative healthcare process, doctors and specialists from different medical organisations share patient's data in order to make a better diagnosis. Due to the current Big data exponential data growth, solutions that store, process [10] and manage medical data are of a great interest. In this direction, cloud computing represents a cost-effective solution for such needs [1, 2]. For instance, Marzini et al. [23] make use of the cloud elasticity to manage basic activities in healthcare scenarios. On the other hand, the usage of cloud computing for medical environments raises several issues such as reliability and security.

Regarding the reliability issue, Gawanmeh et al. [14] present a state of the art review on the verification of reliability in healthcare systems using either simulation-based verification, formal methods such as automata and prism [28, 29], or semi-formal methods. In the work presented in this paper, we focus on the formal verification of the security aspect, especially access control.

Access control protects the system's resources against unauthorized access via a set of policies. Jansen [18] proposed XACML as a policy specification language for cloud applications. Yet, XACML policies may contain conflicting and redundant rules, since XACML policies are sometimes managed by more than one administrator [16]. Moreover, in collaborative applications, the XACML policies are aggregated from collaborative parties which may raise conflicts between rules in different policies.

Several works make use of verification techniques such as model checking in order to detect XACML policy anomalies. For instance, to detect conflicts between rules in a given policy, Martin et al. [22] encode the rules in Coq [8], a tool built specifically for formal theorem proving. A rule is a Coq record with two fields: the first field has the effect type, and the second field contains the srac type that combines the four elements of XACML: subject-resource-action-condition. In order to compare the elements of type srac independently, the authors split them into a defined normal form *DNF*. If two rules have overlap (srac types are identical) with different effects, the rules are then in conflict. Otherwise, if the effects are similar, then the rules are redundant. However, using Coq does not allow the automatic anomaly detection after the insertion of new rules, since their proposed approach does not interact directly with the policies' original format. In contrast, Mourad et al. [24] use the Unified Modelling Language (UML) to detect conflicting and redundant rules prior to their enforcement in the system. However, this technique does not allow completeness verification.

Regarding inter-policy conflict detection, Ramli [30] uses Answer Set Programming (ASP) in order to detect incompleteness, conflicting and unreachable XACML policies. As a limitation of this approach, it is difficult to model XACML expressions dealing with types of attributes that do not belong to AnsProlog [31], such as strings. Huonder [17] proposes another approach to detect and resolve conflicts in XACML policies based on mapping each target to n-dimensional space and overlapping the policies with different effects. The intersection of all dimensions defines an inter-policy conflict. Yet, this technique cannot verify the policy's completeness property.

Besides verification-based techniques, many research efforts consider representing XACML policies as decision trees to detect and resolve conflicts. In this direction, Hu et al. [16] make use of Binary Decision Diagram (BDD). In this work, each XACML attribute is encoded into an atomic boolean expression. The rules are then functions of these expressions. Fisler et al. [12] suggested an extended version of BDD called Multi-Terminal Binary Decision Diagram (MTBDD). Also, Gouglidis et al. [15] transform the XACML policies into Computation Tree Logic (CTL). These tree-based approaches have a main drawback, the state explosion in the decision trees.

A comparison of the proposed approach in this paper with the seven existing methods is presented in Table 2.1. We have adopted the metrics proposed by Li et al. [20]:

1. **Completeness**: it guarantees that any access request has a response by the access control policy: permit or deny.
2. **Policy anomalies** can be divided into two categories, namely intra-policy anomalies (conflicting rules and redundancy) and inter-policy anomalies (inconsistency and similarity):
   - *Conflicting rules*: two rules are in conflict in the same security policy.
   - *Redundancy*: the existence of two rules that have the same effect (permit or deny) such that one of the two rules can be removed without changing the result of the policy.
   - *Inconsistency*: the existence of two or more rules in different policies that are in conflict.
   - *Policy similarity*: two policies can be similar and represented differently.

TABLE 2.1
The Capabilities of the Proposed Access Control Verification Approaches

Approach	Technique	Completeness	Policy anomalies				Flexibility
			Conflicting rules	Redundancy	Inconsistency	Similarity	
[22]	Coq	No	Yes	No	No	No	No
[24]	UML	No	Yes	Yes	No	No	No
[30]	ASP	Yes	No	No	Yes	No	No
[17]	n-dimensions	No	No	No	Yes	No	No
[16]	BDD	No	Yes	Yes	Yes	No	No
[12]	MTBDD	No	No	No	No	Yes	No
[15]	CTL	No	Yes	No	No	No	Yes
Our Proposed method	Automata	Yes	Yes	Yes	Yes	Yes	Yes

3. **Flexibility**: It indicates whether a method can detect anomalies at run-time (i.e. detects if the new inserted rule raises anomalies with the existing rules prior to its enforcement).

Table 2.1 underlines our contributions compared to existing works. The proposed approach uses automata to represent the XACML policies. This formalism, allow us to detect several XACML anomalies (intra and inter policies conflicts) and to verify the completeness property using the same formal model. In addition, the approach has the ability to detect conflicting rules at run-time. In fact, the proposed approach models each security policy with an automaton. To verify if a new rule raises conflicts with the existing ones, the proposed approach consists in applying the synchronous product to the rule's automaton and the policy's automaton. The conflict detection process is then applied to the resulted automaton. Hence, there is no need to integrate the new rule into the policy to do the verification.

3. Preliminaries. The proposed approach consists in verifying XACML (eXtensible Access Control Markup Language) security policies using automata. Therefore, in this section, we define the two concepts: automata and XACML.

3.1. Automata. *Finite state automata* (or briefly automata) are used, for example, for pattern matching in text editors [3], for lexical analysis in compilers, for communication protocol specifications, for language recognition [9], and for firewall design analysis [19]. An automaton can be formally defined by $\mathcal{A} = (\Sigma, Q, q^0, Q^f, \delta)$ where Σ is a finite set of events (also called alphabet), Q is a finite set of states, q^0 is the initial state and $Q^f \subseteq Q$ is a finite set of final states. $\delta : Q \times \Sigma \to Q$ is the transition function, where $\delta(q, \sigma) = r$ means that the execution of the event σ (or the reading of the term σ) from state q leads to state r. $\delta(q, \sigma) = r$ can also be written as $q\sigma r$.

An automaton $\mathcal{A}$ consists of states linked by labelled transitions, and represented by a graph whose nodes and arcs are the states and the transitions of $\mathcal{A}$, respectively. There is one initial state (with a small incoming arrow) and one or more final states (double circled).

In this paper, we use the notation $S = \{\sigma_1, \sigma_2, \ldots, \sigma_p\}$ (it can be also denoted as $S = \{\sigma_1\} \cup \{\sigma_2\} \cup \ldots \{\sigma_p\}$) for a set of events. The notation qSr means that if q is the current state, then every event σ_k from the set S leads to the state r. The arc labelled S, linking q and r, is equivalent to many arcs labelled $\sigma_1, \ldots, \sigma_p$ linking q and r.

A finite event sequence (more briefly, sequence) is accepted by $\mathcal{A}$ if it starts in the initial state q^0 and terminates in one of the final states of $\mathcal{A}$. The language of $\mathcal{A}$, denoted $\mathcal{L}_\mathcal{A}$, is the set of sequences accepted by $\mathcal{A}$.

The rich theory of automata allows us to compose models of systems, behaviours, mechanisms due to the operations that can be performed over automata. For instance, the synchronous products of two automata $\mathcal{A}_1$ and $\mathcal{A}_2$ over the alphabet Σ is an automaton over the alphabet Σ whose language is $\mathcal{L}_{A_1} \cap \mathcal{L}_{A_2}$. This intersection permits us to track the behaviour of the global system (consisting of the two subsystems modelled by A_1 and A_2) in order to detect anomalies and conflicts.

3.2. XACML. XACML (eXtensible Access Control Markup Language) has been widely used as a policy specification language in both academia and industry. Its first version was released by Anderson et al. [4] in 2003 and used in the context of distributed systems [21]. Two years later, OASIS extended the old version and

proposed XACML 3.0 [25]. XACML assumes an architecture containing a PDP (Policy Decision Point), PEP (Policy Enforcement Point) and PAP (Policy Administration Point). The XACML request to access a specific resource is redirected to PEP. The PEP then extracts the attributes from the request and sends them to the PDP which searches in the policy repository for the appropriate policy that matches the request. The PDP then sends a response to the requester, which can be: *Permit*, *Deny*, *Not Applicable* or *Indeterminate*. The first two responses are obvious. *Not Applicable* is applied if no rule or policy matches the request. *Indeterminate* is applied if the system cannot interpret the request due to the lack of attributes or problems of connection. The PAP is responsible to associate the new added rules to the appropriate policy.

The main component of XACML *Policy* is composed of a *Target* that identifies the capabilities that should be exposed by the requester (the targeted resources for example), and some *Rules*. Each *Rule* contains facts (*Subjects*, *Resources*, *Actions* and *Environment*) for access control decisions and an *Effect* that can be either *Permit* or *Deny*.

Policies can be combined using *PolicySet* that specifies the combining algorithms in case if two security policies provoke permit/deny conflicts. XACML offers four combining algorithms:

- permit-override: If at least one policy is evaluated as "*permit*", the integrated output will also be "*permit*".
- deny-override: If at least one policy is evaluated as "*deny*", the integrated output will also be "*deny*".
- first applicable: The result of the combining algorithm is the result of the first policy that evaluates to Permit or Deny.
- only-one-applicable: The result is the one of the only applicable policy. If we have more than one policy, then the result is Not Applicable.

A XACML policy contains hundreds and thousands of rules, which make it difficult to detect policy conflicts directly from the XML file. Yet, identifying conflicts in XACML policies is a primordial task for their designers. In fact, the choice of the combining algorithms relies essentially on the information from conflict diagnosis. The XACML policies may contain two kinds of conflicts: intra-policy (conflict between rules of the same policy) and inter-policy (conflict between rules of several policies defined under the *PolicySet*).

4. CPVS: Cloud Policy Verification Service. Cloud computing offers several services, such as computing, authentication, and storage. These services could support collaborations among different organisations. Yet, such collaborations need to be regulated by a set of access control policies to protect the shared resources. However, the current Cloud architectures do not provide to the users the capability to define their own access control policies (high level control policies). For instance, *Openstack* is a widely used Cloud open source software that offers a storage service via *Swift* [5]. Although the *Swift* component supports fine-grained access control to objects (resources), it remains specific and at a low level of control. To address this issue, we have developed a middleware denoted *curlX* [6] that permits the collaborators to express their own security policies using XACML and enforce them using the cloud primitives such us curl (client url request) library. The user sends a curl request to the middleware asking for accessing a resource stored in *Swift* (see Fig. 4.1). The curl request is of the form:

$\mathbf{curl} - X <PUT|POST> -i - HX - Auth - Token :<TOKEN> -HX - Container - Read :<ACL> <STORAGE - URL>/<container>.$

The authentication token and the storage url are provided by Keystone. Keystone is an authentication service in *Openstack*. It provides each authenticated user by tokens that expire after a specific time delay. The curl request is redirected to the translator component to be transformed into XACML request and redirected to the PEP. After retrieving the main attributes (subjects, resources, and action), PEP sends them to PDP in order to search for an adequate policy, and then decide if the request is permitted or not.

In the first version of *curlX* [7], we did not take into account the analysis of XACML policies. For this purpose, in this paper, we integrate a new component into *curlX* denoted *CPVS* (Cloud Policy Verification Service). Figure 4.1 presents the global architecture of *curlX* after the integration of *CPVS*. It is a policy verification framework that consists of the following four verification modules (see Fig. 4.2):

- Intra-policy anomaly detection: responsible for detecting conflicting and redundant rules in a single security policy.

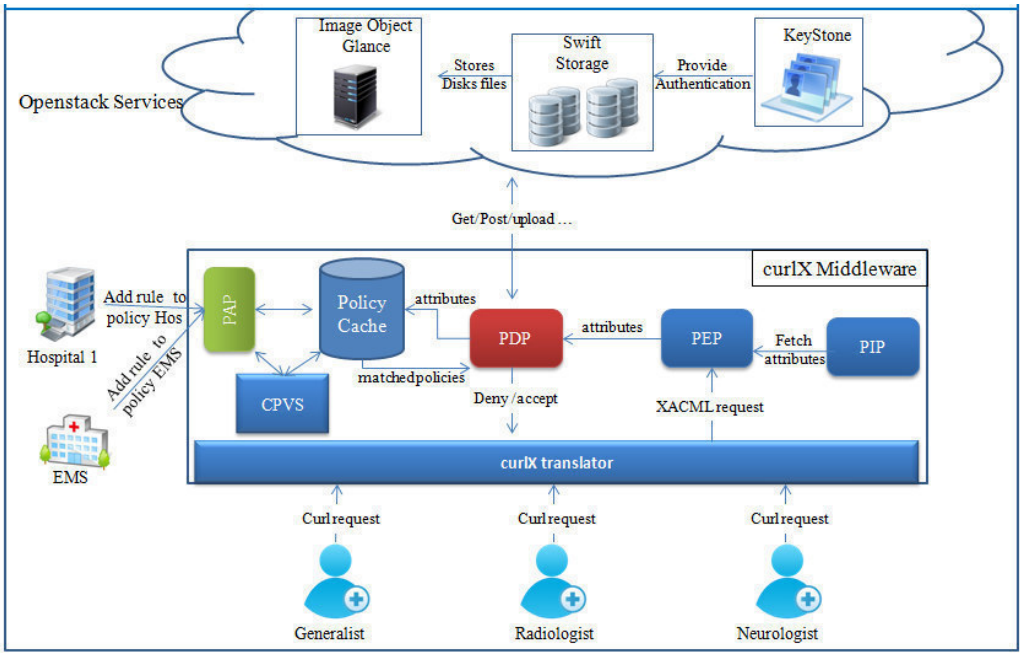


FIG. 4.1. *The global architecture of curlX*

- Inter-policy anomaly detection: if the policy cache contains combined security policies, this module is used to detect inconsistency and similarity between the combined policies.
- flexibility: medical organisations may add a new rule to their security policies. However, this new rule may create a conflict with the existing ones. Hence, this module is responsible for verifying if the new rule generates any conflicts or redundancies prior to its enforcement.
- Policy properties verification: verifies the completeness for each policy stored in the cache.

Each verification module communicates with the policy cache via the *Xparser* sub-module that parses the XACML policies and extracts its components in a hierarchical way.

5. Modelling XACML Policies by Automata.

5.1. Use Case: Stroke Accident. Healthcare organisations provide several services to their patients: emergency services, day procedures, diagnostic services, therapy services, etc. For each service, an organisation may have to produce documents (e.g. personal records, X-ray, brain scan, electroencephalography (EEG), etc). Those documents are typically stored in the organisation's data center. An organisation's data should be accessible by stakeholders from other organisations, so that they can collaborate in an elaborated diagnosis. However, information sharing must be regulated in order to guarantee the integrity and confidentiality of the shared information. This leads to the necessity of having policies regulating the medical data sharing.

Hereafter, we consider a reference scenario of a stroke accident presented by the Moroccan emergency medical service of Rabat [26]. In this scenario, three kinds of medical organisations are involved: two hospitals (H_1 and H_2), one emergency medical service (EMS) and two university hospitals (UH_1 and UH_2). These organisations are involved in a collaborative session (presented by a sequence of accesses) in order to perform an effective diagnosis to the transferred patient. In fact, doctors located in the host hospital (the hospital where the patient has been transferred) can make use of the experiences of specialists located in other medical organisations.

Our proposed scenario is that of a patient that has an accident and is transferred to the nearest hospital, which we call host hospital and denote by H_1. The Moroccan medical system relies on distributed storage: the patient can have his medical file in another hospital H_2. Once the patient is in H_1, the generalist calls EMS and a regulator receives the call and inserts the patient's information in the system. The regulator looks for

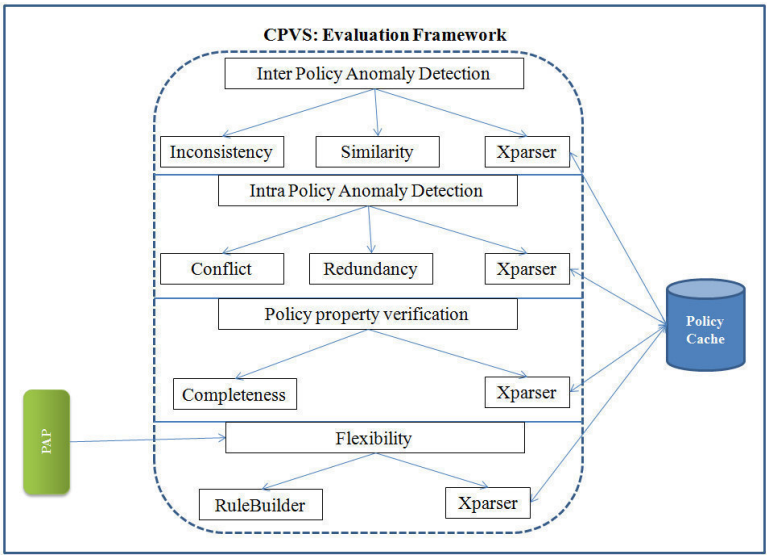


FIG. 4.2. *The four modules of CPVS (Cloud Policy Verification Service)*

the available specialist doctors: radiologists, cardiologists, neurologists etc., in other hospitals or in university hospitals. Once he finds a list of available doctors, the regulator creates a collaborative session. During the collaborative session (whose aim is to elaborate a diagnosis), the specialist doctors may ask the host hospital to provide them with some scans and a part of the medical file of the patient.

Each organisation regulates the access to its own resources by enforcing a set of access control rules. For example, we consider the rule R_1 in a hospital H_1 that permits the radiologists of H_1 to write into the personal record (PR) and the scans of all the patients belonging to this hospital. In the rest of the paper, we adopt three essential notions that are used in the security policies: subjects, objects (resources), and actions. Therefore, in the next subsection we formally describe each one of them. This description is essential for the construction of automata.

5.2. Formal Description of the Collaborating Organisations. Let Org denote the set of organisations involved in the collaborative session. Each organisation has:

- Subjects: they are human resources. Formally, we have:
  Subjects = Doctors ∪ Nurses where
  Doctors = $\bigcup_{x \in Org} doctors_x$ where
  $doctors_x = generalists_x \cup radiologists_x \cup regulators_x \cup cardiologists_x \cup neurologists_x$
- Objects: they are physical and computer resources (hardware, software). For the sake of simplicity, we consider here only the following categories: personal records (PR), scans, audio, lists of available doctors ($listDoctors_x$), and the histories of the collaborative session's discussion ($collSessDisc_x$). We obtain formulas like:
  Objects = $\bigcup_{x \in Org} objects_x$ where
  $objects_x = PR_x \cup scans_x \cup audios_x \cup listDoctors_x \cup collSessDisc_x$

For the rest of the paper, we consider four types of scans: MRI (Magnetic Resonance Imaging), CAT (Computed Axial Tomography), EEG (Electroencephalography) and MRA (Magnetic Resonance Angiogram). For the purpose of illustrating our study, we will consider only the following three categories of medical organisations that are present in most scenarios: hospitals, university hospitals, and emergency medical services. For the sake of simplicity, we will restrict their sets of subjects and objects as follows (Figure 5.1):

1. **Hospitals** H_1, H_2, H_3, ...
   The subjects of a hospital are: generalists, radiologists, and nurses. Formally:

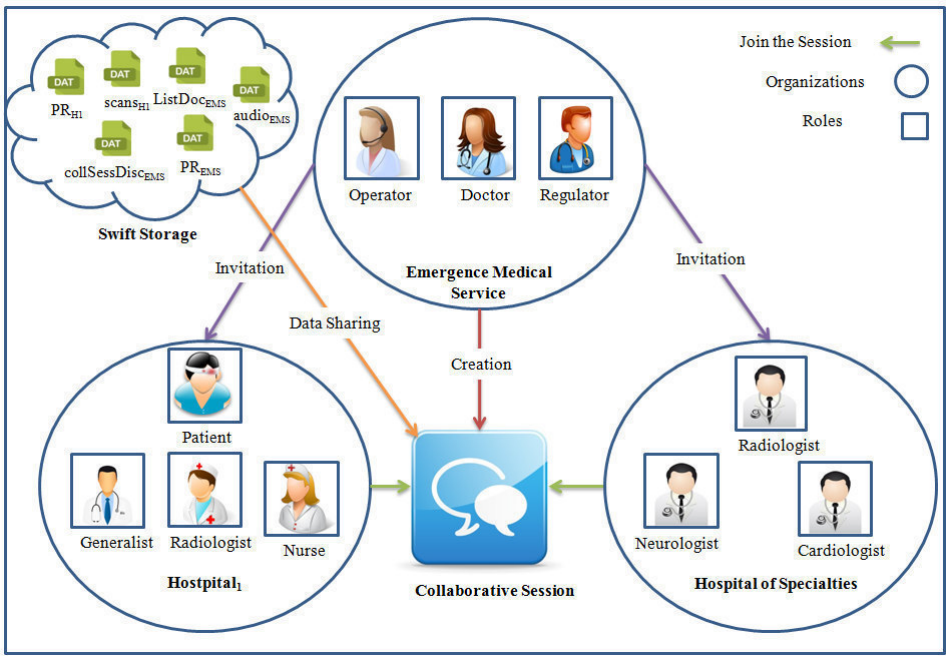


FIG. 5.1. *Organisation involved in the collaborative diagnosis.*

$subjects_{H_i} = generalists_{H_i} \cup radiologists_{H_i} \cup nurses_{H_i}$
The objects of a hospital are: personal records of patients regularly followed by the hospital, or of patients transferred to the hospital in emergency cases; and scans consisting of MRI, MRA, CAT, and EEG. Formally:
$objects_{H_i} = PR_{H_i} \cup MRI_{H_i} \cup MRA_{H_i} \cup CAT_{H_i} \cup EEG_{H_i}$.

2. **University hospitals** UH_1, UH_2, UH_3, ...
The subjects of a university hospital are specialist doctors (for our case, neurologists, cardiologists and radiologists). Formally:
$subjects_{UH_i} = neurologists_{UH_i} \cup radiologists_{UH_i} \cup cardiologists_{UH_i}$.

3. **Emergency medical services** EMS_1, EMS_2,...
The subjects of an emergency medical service are regulators and generalists. Formally:
$subjects_{EMS_i} = generalists_{EMS_i} \cup regulators_{EMS_i}$.
The objects of an emergency medical service are personal records of the hosted people and who passed through the emergency case, audio records, list of the doctors of hospitals and university hospitals, and history of the collaborative session's discussions. Formally:
$objects_{EMS_i} = PR_{EMS_i} \cup audios_{EMS_i} \cup collSessDisc_{EMS_i} \cup listDoctors_{EMS_i}$.

5.3. From XACML Policies to Automata. XACML policies have three levels, namely *PolicySet*, *Policy* and *Rule*. *Rule* is the single entity that describes the particular access control policy. Therefore, in this paper, we focus mainly on the formalisation of *Rule*. *Policy* is the sequence (combination) of several rules. *PolicySet* is the sequence (combination) of two or more policies.

A rule is formally defined by triplet (S, O, a_U), where S is a set of subjects, O is a set of objects, and a_U represents a permission. More precisely, in a_U we have a = Deny or Permit, and U is a set of actions like read and write. The meaning of (S, O, a_U) is:

- if a=Permit, then any action in U applied by a subject of S to an object of O is permitted;
- if a=Deny, then any action in U applied by a subject of S to an object of O is forbidden.

The rule described at the end of Sect. 5.1 can be written as follows: $(radiologist_{H_1}, scans_{H_1} \cup PR_{H_1}, p_{write})$. This gives the right to the radiologists of hospital H_1 to perform the write action on the two categories of objects: the scans and personal records of all patients of hospital H_1.

In XACML, a rule is described by: an *Effect* and a *Target*. The *Effect* can have two values: "*Permit*" and "*Deny*". The *Target* is a combination of *Match* elements. Each *Match* element describes an attribute that a *Request* should match in order to activate a policy. There are four attribute categories in XACML 3.0, namely: (a) subject attribute is the entity requesting the access, e.g., generalist, radiologist, etc; (b) resource attribute is the object or the required data, e.g., EEG, MRA, etc; and (c) action attribute defines the type of access requested, e.g, read, write, delete, etc. The evaluation of the *Match* attributes extracted from the rule permits the evaluation of the request. Even if the request matches one of the rules, the algorithm continues until the last rule in the *PolicySet*.

Our proposed automata-based approach is realized as follows: From the XACML representation of a policy F, we construct an automaton $\mathcal{A}$ that models F, and then our analyses of F are done on $\mathcal{A}$. The automaton $\mathcal{A}$ generated from a policy F has the following characteristics: from the initial state of $\mathcal{A}$, we have several possible paths where each path consists of *a pair* of transitions that leads to a final state associated to a permission a_U (see Sect. 3.1). Each path represents a rule (S, O, a_U) of F as follows: the first and second transitions are labelled S and O respectively, and the reached state is associated to a_U. The set of paths of $\mathcal{A}$ represents therefore a set of rules that constitute F. Table 5.1 indicates how the constituents of a XACML policy are represented in the corresponding automaton.

Table 5.1
How the constituents of a XACML policy are represented in the corresponding automaton

XACML Policies	Finite State Automaton
Rule	Word
Set of subjects and objects	Alphabet
ActionMatch attributes	Actions associated to the final states
SubjectMatch attributes	Labels of first transitions
ResourceMatch attributes	Labels of second transitions

Consider a medical organisation x and its security policy consisting of rules $x_1, x_2, \ldots, x_n$, where n is the number of rules. The construction of the automaton from the policy is done in four steps [19]:

- **Step 1: Attribute extraction from a XACML policy.** Algorithm 1 parses the XACML policy using the function *getDetailPolicy* that extracts rules from the XML file and expresses each one of them formally by a triplet (S, O, a_U). Each rule has: *Effect* (a), *SubjectMatch* (S), *ResourceMatch* (O), *Action* (U).
- **Step 2: Automaton for each rule.** each rule x_i = (S, O, a_U) obtained in Step 1 is described by an automaton with four states x_i^0, x_i^1, and x_i^2 and $x_i^{\#}$, where x_i^0 represents the initial state, x_i^2 and $x_i^{\#}$ are final states. The pair of states x_i^0 and x_i^1 are linked by a transition labelled S, and the pair of transition x_i^1 and x_i^2 are linked by a transition labelled O. The permission a_U is associated to the final state x_i^2. Transitions labelled $\neq$S and $\neq$O link x_i^0 and x_i^1 to $x_i^{\#}$ respectively. $\neq$S denotes the set of subjects of all the collaborative medical organisations, except the subjects of S. $\neq$O denotes the set of objects of the medical organisation owning the policy, except the objects of O. The final state x_i^2 is called *match state*, because it is reached for any request matching the attribute values of the rule x_i, i.e. for any subject s $\in$ S and object o $\in$ O. The final state $x_i^{\#}$ is called *no-match state*, because it is reached if the request does not match the attribute values of x_i. As an example, Fig. 5.2 represents the automaton obtained from the rule presented in the end of Section 5.1.
- **Step 3: Standardize the intervals of the automata.** The automata obtained in Step 2 do not have the same alphabet, the objective here is therefore to rewrite the transitions of the automata so that they have the same alphabet (this rewriting is useful for Step 4). This is realized by partitioning each of the domains of subjects and objects into a set of disjoint sets. Such partitioning permits to express a transition of an automaton as a union of sets of the partition.

Algorithm 1 Algorithm of Step 1

Input: XACML Policy
Output: S, O, a, U

procedure GETDETAILPOLICY(Policy.xml)
  $document \leftarrow parse(Policy.xml)$
  $root \leftarrow document.getDocumentElement()$ ▷ parses the tags of the xml file
  while $root \neq EndofDocument$ **do**
    for $i \leftarrow 0, nbRootNodes$ **do**
      if $node.getName =$ "$Rule$" and $node.getAttributes \neq null$ **then**
        $a \leftarrow node.getAttributes.getNamedItem($"$Effect$"$)$
        if node.getNodeName = "SubjectMatch" **then**
          $S \leftarrow Attribute.getTextContent$
        else if node.getNodeName = "ResourceMatch" **then**
          $O \leftarrow Attribute.getTextContent$
        else if node.getNodeName() = "ActionMatch" **then**
          $U \leftarrow Attribute.getTextContent$
        end if
      end if
    end for
  end while
  return S, O, a, U
end procedure

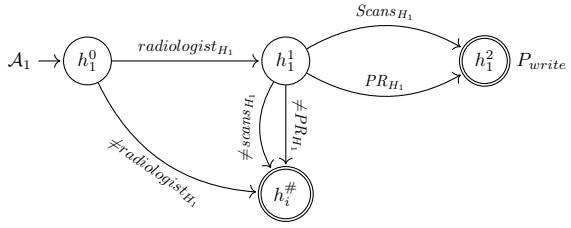


FIG. 5.2. *Automaton $\mathcal{A}_1$ obtained in Step 1 for the rule R_1.*

For example, the automaton of Fig. 5.2 is transformed into the automaton of Fig. 5.3. The set $scans_{H_1}$ has been partitioned into 4 sets $\mathrm{MRI}_{H_1}, \mathrm{MRA}_{H_1}, \mathrm{CAT}_{H_1}$ and EEG_{H_1}, which implies that the transition labelled $scans_{H_1}$ is replaced by four transitions labelled $\mathrm{MRI}_{H_1}, \mathrm{MRA}_{H_1}, \mathrm{CAT}_{H_1}$ and EEG_{H_1}, respectively.

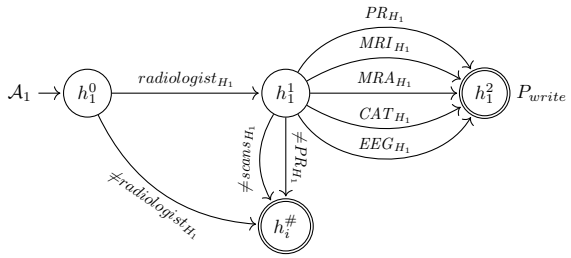


FIG. 5.3. *Automaton $\mathcal{A}_1^*$ obtained from $\mathcal{A}_1$ of Fig. 5.2*

- **Step 4: Product of automata**. In order to model the security policy defined in a XACML policy file, we combine the automata resulting from Step 3 by an operator called *synchronous product*. Hereafter, we consider the policy presented in Table 5.2 as an example. It contains seven rules regulating access to different resources (objects).
  The resulting automaton representing the policy of an organisation x is denoted $\mathcal{A}_x$. Each of its states is a combination of states $(u_1, u_2, \ldots, u_n)$ of the various combined automata, hence each $u_i = x_i^j$ or $x_i^\#$, for $j = 1$ or 2. A final state $x_{\ldots}^2$ may be associated to one or more permissions. For the sake of clarity, a state of $\mathcal{A}_x$ is noted $x_{i_1,i_2,\ldots}^j$, where we indicate only the indices i_k such that $u_{i_k} = x_{i_k}^j$ (i.e. $u_{i_k} \neq x_{i_k}^\#$), for $j = 1$ or 2. For example, the initial state is noted $x_{1,2,\ldots,n}^0$. A state is noted $x^\#$ if all its

TABLE 5.2
Example of a XACML policy

RuleID	Effect	SubjectMatch	ResourceMatch	ActionMatch
R_1	Permit	generalist	PR	read
R_2	Permit	neurologist	EEG	read
R_3	Permit	radiologist	Scans	write
R_4	Deny	radiologist	Scans	write
R_5	Deny	generalist	PR	read
R_6	Permit	neurologist	EEG	read
R_7	Permit	generalist	PR	read

components are $x_i^{\#}$. For example, if we apply the four steps to the policy of Table 5.2, we obtain the automaton of Fig. 5.4.

Let us explain the notation used for match-states, for example the match-state $h^2_{1,5,7}$ associated to permissions (P_{read}, D_{read}, P_{read}). This state is reached by the pair of transitions (generalist, PR), i.e. when a generalist wants to have access to a personal record of a patient. The three indices 1, 5, and 7 mean that this access is matched by the rules 1, 5 and 7. The three permissions are respectively associated to the three indices, i.e.: R_1 and R_7 permit the read access, and R_5 forbids the read access.

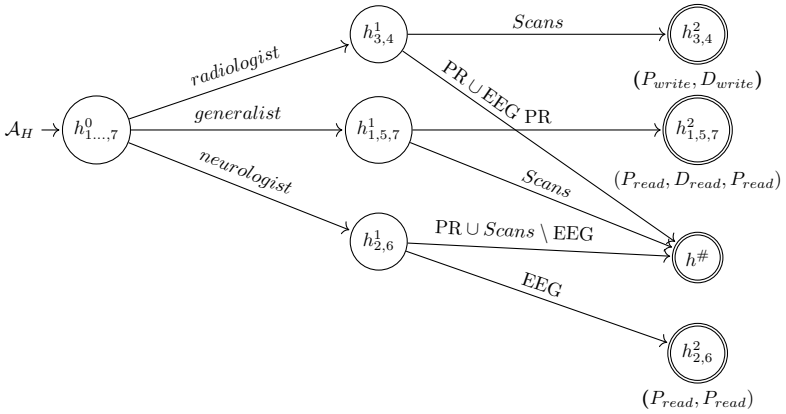


FIG. 5.4. *Automata $\mathcal{A}_H$ Modelling the Security Policy of Table 5.2.*

6. Intra- and Inter-security Policy Anomaly Detection. In a XACML policy, we can specify the policies of several entities (or organisations, like hospitals): the policy of each organisation is represented by *Policy*. *PolicySet* specifies how the different *Policy* components are combined. We study therefore two types of anomalies: intra-policy anomalies that correspond to anomalies between rules of a same policy, and inter-policies anomalies that correspond to anomalies between rules of different policies. In this paper, we consider two types of intra-policy anomalies: redundancy and conflicting rules, and two types of inter-policy anomalies: similarity and inconsistency.

6.1. Intra-policy anomaly detection. Let us first consider intra-policy anomalies of a policy F and show how they are detected by the automaton of F.

6.1.1. Detecting redundant rules. In a policy F, a rule R_j is redundant to a rule R_i if the result of F is not changed by removing R_j and keeping R_i.

PROPOSITION 1. *Consider a policy F and its automaton $\mathcal{A}_F$. A rule R_j is redundant to a rule R_i if for every match-state $x^2_{j_1,j_2...}$ of $\mathcal{A}_F$:*

1. *the match-state has the index j only if it has also the index i, and*
2. *i and j are associated to the same permission.*

Consider, for example, the policy of Table 5.2 and its automaton of Fig. 5.4. Rules 2 and 6 are mutually redundant to each other, because: 1) the indices 2 and 6 are in the state $h^2_{2,6}$ and there is no other state where the indices are not together, and 2) the same permission P_{read} is associated to both indices. Therefore, we can remove either R_2 or R_6 without changing the result of the policy.

6.1.2. Detecting conflicting rules. In a policy F, two rules R_i and R_j are conflicting if they can match the same subjects and objects (s,o) and have different permissions.

PROPOSITION 2. *Consider a policy F and its automaton $\mathcal{A}_F$. Rules R_i and R_j are conflicting if there exists a match-state $x^2_{j_1,j_2\cdots}$ of $\mathcal{A}_F$ such that:*

1. *the match-state has the indices i and j, and*
2. *i and j are associated to different permissions.*

Consider, for example, the policy of Table 5.2 and its automaton of Fig. 5.4. The match-state $h^2_{3,4}$ implies that R_3 and R_4 are conflicting. Intuitively, R_3 permits radiologists to read scans while R_4 forbids it. Also, the match-state $h^2_{1,5,7}$ implies that R_5 is conflicting with R_1 and R_7. Intuitively, R_1 and R_7 permits generalists to read PR while R_5 forbids it.

Algorithm 2 in Appendix A regroups the steps of detecting intra-policy conflicts. The algorithm consists in extracting the final states using the function *getFinalNodes*. The algorithm compares then the permissions associated to each final state, if they are different, the rules are considered as conflicting rules, and otherwise they are redundant.

6.2. Inter-policy anomaly detection. Let us now consider inter-policy anomalies of two policies F_1 and F_2. To detect this type of anomalies, we need here to construct an automaton that combines the automata $\mathcal{A}_{F1}$ and $\mathcal{A}_{F2}$. This is equivalent to consider the global policy F obtained by putting together the rules of F_1 and F_2 and then constructing the automaton $\mathcal{A}_F$ of F.

6.2.1. Detecting similar policies. Two policies F_1 and F_2 are similar if in every situation, the decision of F_1 is similar to the decision of F_2.

PROPOSITION 3. *Consider two policies F_1 and F_2 and the automaton $\mathcal{A}_F$ of the policy F that regroups F_1 and F_2. F_1 and F_2 are similar if in F: every rule of F_1 is redundant to a rule of F_2, and every rule of F_1 is redundant to a rule of F_2.*

Proposition 3 implies that similarity can be verified by detecting redundancy between the rules of the policies using Proposition 1.

6.2.2. Detecting inconsistent policies. Two policies F_1 and F_2 are inconsistent if there exists a situation where they have contradictory (i.e., different) decisions.

PROPOSITION 4. *Consider two policies F_1 and F_2 and the automaton $\mathcal{A}_F$ of the policy F that regroups F_1 and F_2. F_1 and F_2 are inconsistent if in F: there exist a rule of F_1 and a rule of F_2 which are conflicting.*

Proposition 4 implies that inconsistency can be verified by detecting conflicting rules using Proposition 2.

Therefore, the same logic of the two procedures of Algorithm 2 can be applied to detect inconsistency and policy similarity. The only difference is the input of the algorithm: instead of an automaton corresponding to one policy, the input is replaced by a synthesized automaton of two policies (Algorithm 3 in Appendix A).

7. Verification of the Completeness Property. Besides the intra- and inter-security policy anomalies, it is important to assure the evaluation of security properties to guarantee the correctness of access control policies. Most of the existing Cloud verification methods focus mainly on the system behaviour verification and do not take into consideration the security policies. Therefore, designing a dedicated tool that targets the verification of security properties in the Cloud is an important issue to be addressed [13]. In this section, we describe a formal method based on the automata generated in Section 5 to detect and verify the completeness property. Completeness guarantees that each access request is either accepted or denied by the access control policy.

PROPOSITION 5. *A security policy $\mathcal{P}$ is complete if and only if the corresponding synthesized automaton $\mathcal{A}_P$ has no* no-match state.

For instance, the security policy presented in Table 5.2 is incomplete because its corresponding automaton in Fig. 5.4 has a *no-match state* denoted $h^{\#}$. The 3 paths leading to $h^{\#}$ correspond to the following 3 situations:

- A radiologist requests access to a PR or EEG.
- A generalist requests access to a scan.
- A neurologist requests access to a PR or a scan that is not EEG.

Intuitively, the security policy of Table 5.2 does not take any decision in these 3 situations. Algorithm 4 in Appendix A presents the procedure *isComplete* that verifies if the input automaton has a no-match state.

8. Evaluation of Space and Time Complexities. Let n be the number of rules of a policy, and d_1 and d_2 be the numbers of bits to code the subjects and objects, respectively. Hence, the maximum possible number of subjects and objects are 2^{d_1} and 2^{d_2}, respectively. Let $D = d_1 + d_2$. We consider two notions called *great fields* and *small fields* defined by Khoumsi et al. [19]. A *great field* is a field whose domain contains more than n values, and a *small field* is a field whose domain contains at most n values. We then consider two variables: μ, the number of *great fields*; and δ, the sum of the number of bits to code the *small fields*.

By adapting the results of Khoumsi et al. [19] to our context, we obtain Proposition 6 (the proof of this proposition is in Appendix B).

PROPOSITION 6. *The space and time complexities of automata construction and completeness detection are in* $O(n^{\mu+1} \times 2^{\delta})$*, which is bounded by both* $O(n^3)$ *and* $O(n \times 2^D)$.

The bounds of the complexities for the procedures of policy analysis are obtained by multiplying the above values by n. Hence, we obtain the following proposition:

PROPOSITION 7. *The space and time complexities of redundancy and conflict detections are in* $O(n^{\mu+2} \times 2^{\delta})$*, which is bounded by* $O(n^4)$ *and* $O(n^2 \times 2^D)$.

The latter result holds also for detecting similarity and inconsistency between two policies, but by replacing n by $n_1 + n_2$, where n_1 and n_2 are the numbers of rules of the two policies.

As an example, we can consider a policy with $n = 500$ rules where the maximum number of subjects is 256 (so the subjects are coded in 8 bits: $2^8 = 256$) and where the maximum number of objects is 131072 (hence the objects are coded in 17 bits: $2^{17} = 131072$). Hence:

- D = 25 = 8+17 = total number of bits to code the great and small fields.
- $\mu = 1$ = number of great fields: there is one great field which is "objects", because $2^{17} > 500$.
- $2 - \mu = 1$ = number of small fields: there is one small field which is "subjects", because $2^8 \leq 500$.
- $\delta = 8$ = number of bits to code the small field subjects

If we use the expression $O(n^{\mu+1} \times 2^{\delta})$ which depends on the great and small fields, we obtain: $O(n^{\mu+1} \times 2^{\delta}) = O((500^2) \times (2^8)) = O(64$ millions). However, if we use the two expressions $O(n^2)$ and $O(n \times 2^D)$ which do not depend on the great and small fields, we obtain:

- $O(n^2) = O(500^3) = O(125$ millions)
- $O(n \times 2^D) = O(500 \times 2^{25}) = O(16.7$ billions)

From the example, we can conclude that by considering the great and small fields, we obtain a more precise estimation of the complexity. Note that:

- when all the fields are great, we obtain $O(n^{\mu+1} \times 2^{\delta}) = O(n^2)$,
- when all the fields are small, we obtain $O(n^{\mu+1} \times 2^{\delta}) = O(n \times 2^D)$.

From Proposition 6, the time and space complexities of automata construction and completeness detection are upper-bounded by $O(n^3)$ and $O(n \times 2^D)$. Let *Ns* and *No* be the maximum numbers of subjects and objects, respectively. We have $2^D = 2^{d_1} \times 2^{d_2}$, $Ns = 2^{d_1}$ and $No = 2^{d_2}$. Therefore, $O(n \times 2^D) = O(n \times Ns \times No)$. We deduce that our complexities of automata construction and completeness detection exceeds neither the order of the polynomial n^3 nor the order of $n \times Ns \times No$.

With the same reasoning on Proposition 7 we obtain that our complexities of redundancy and conflict detections exceed neither the order of the polynomial n^4 nor the order of $n^2 \times Ns \times No$.

In conclusion, our complexities are at most polynomial in n and linear in *Ns* and *No*.

9. Implementation and Evaluation. We have implemented our policy analysis service *CPVS* (Cloud Policy Verification Service) in Java. This service is integrated into *curlX*, a middleware integrated into *Openstack*. Based on our policy anomaly analysis mechanism, *CPVS* consists of four core components: Inter-policy anomaly detection module, intra-policy anomaly detection module, policy property verification module, and flexibility module. The modules are described in detail in Section 4. *CPVS* makes use of the DOM API provided by the Sun XACML implementation in order to parse the XACML policies and extract the attributes.

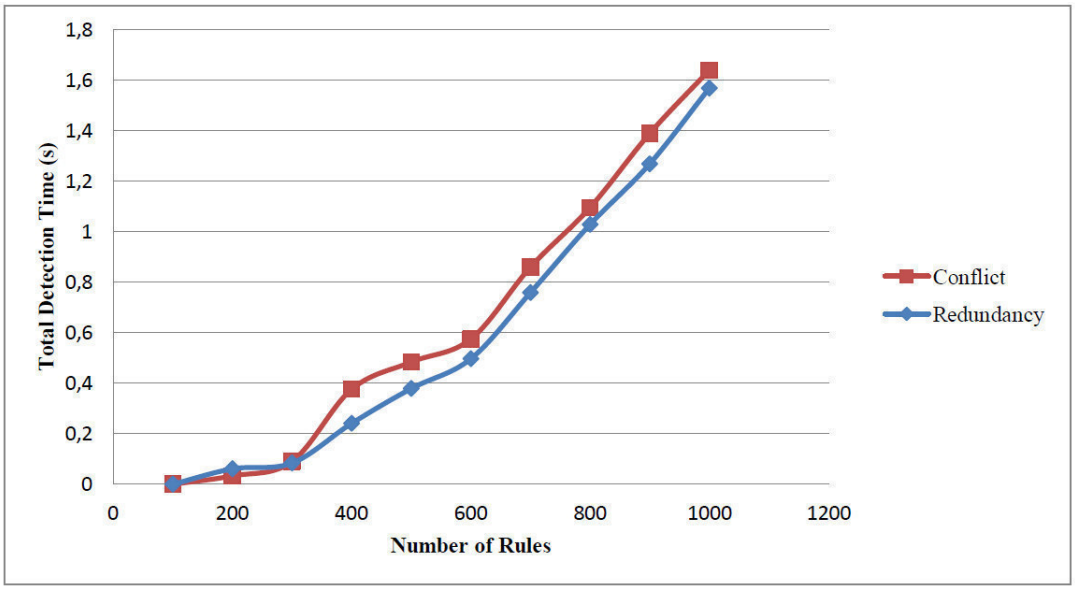


FIG. 9.1. *Performance Improvement for Intra-policy Anomaly Detection*

We have implemented a Domain Specific Language (DSL) to support the construction of automata. In order to evaluate the efficiency and effectiveness of the proposed solution, first we need large policy data sets. Unfortunately, no one has been published due to confidentiality constraints. Hence, we have developed a random policy generator in order to generate a large number of XACML policies.

For scalability, it is also important to note that creating subjects and objects with no semantic relationship (categorisation) is an inefficient approach. It is better to regroup the subjects and objects in subsets or categories to reduce their sizes. For instance, we can have 10 objects: 4 files consisting of prescriptions and 3 scans (*EEG*, *MRI*, *BrainScan*), and 3 documents containing information about the patient. For this example, we have two sub-categories: Scans and PR (prescriptions and documents). In this way, instead of having 10 atomic objects, we have only 2 subsets. This reduces the number of states in the final policy automaton, which then reduces the time of anomaly detection.

We evaluated the efficiency and effectiveness of *CPVS* for synthetic XACML policies using 10 synthetic generated policies. Our experiments were performed on an Intel Core 2 Duo CPU 2.00 GHz with 3.00 GB RAM running on Windows 7. We adopted three types of performance measurement related to intra-policy anomaly detection, inter-policy anomaly detection and incompleteness detection.

The time required by *CPVS* to detect anomalies, such as redundancy and conflicts, depends on the time of parsing and comparing the final states of the automaton. From Fig. 9.1, we can notice that the times of conflict and redundancy detections are quasi equal, which reflects the results of time complexity of Section 8.

Furthermore, we generated synthetic policies consisting of 100 rules, and we combined them using the synchronous product. Figure 9.2 presents the performance of *CPVS* to detect inconsistency and similarity between different set of policies (2, 4 . . . 10).

The verification of completeness, which consists in finding the *no-match* state in the policy's automata, depends on the location of such a state. The performance of such verification is quasi constant (Fig. 9.3). It remains 2 ms for policies that contain 100, 200, 300 and 400 rules, and then it goes to 3 ms for the four other policies.

10. Conclusion. We have proposed a formal approach based on automata to detect XACML policy anomalies and verify the policy completeness. Our proposed approach consists of four steps: (1) extracting attributes from XACML policies; (2) modelling each rule by an automaton; (3) standardising the sets of transitions in automata; and finally (4) forming products of automata to model the security policy. The

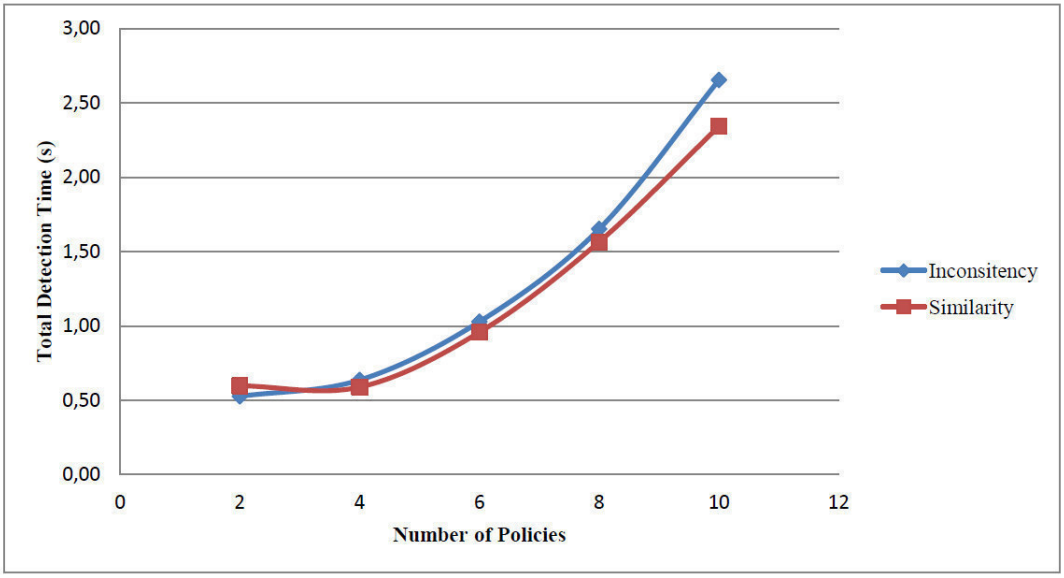


FIG. 9.2. *Performance Improvement for Inter-policy Anomaly Detection*

resulted automaton is used to detect anomalies based on analysing its final states. We evaluated the time and space complexities for anomaly detection. The approach has been implemented in a Cloud service called *CPVS* (Cloud Policy Verification Service) integrated into a middleware denoted *curlX*. The advantage of our approach is that it detects several anomalies in XACML policies at two different levels using the same formal model. In fact, it can detect intra-policy anomalies such as conflicts and redundancies, and inter-policy anomalies such as inconsistencies and similarities.

In future work, we intend to propose a formal approach to resolve the detected anomalies based on a dynamic aspect. The resolution takes into consideration the XACML combining algorithms. Moreover, the concept of delegation is often used in the domain of e-health, so we intend to verify the impact of delegation of roles on the security policies.

Acknowledgements. The authors want to thank Najib Fouhami for his help during the implementation and Yahya Benkaouz for his helpful discussions. This work is supported by the BMBF (PMARS Program) and the DAAD (German-Arab Transformation Partnership).

REFERENCES

[1] B. CALABRESE AND M. CANNATARO, *Cloud computing in healthcare and biomedicine*, Scalable Computing: Practice and Experience, vol. 16, no. 1, pp. 1–18, 2015.
[2] S. P. AHUJA, S. MANI, AND J. ZAMBRANO, *A survey of the state of cloud computing in healthcare*, Network and Communication Technologies, vol. 1, no. 2, pp. 12-19, 2012.
[3] V. ALFRED, *Algorithms for finding patterns in strings*, Algorithms and Complexity, vol. 1, pp. 255-300, 2014.
[4] A. ANDERSON, A. NADALIN, B. PARDUCCI, D. ENGOVATOV, H. LOCKHART, M. KUDO, P. HUMENN, S. GODIK, S. ANDERSON, S. CROCKER, T. MOSES, *Extensible access control markup language (xacml) version 1.0*, OASIS, 2003.
[5] J. ARNOLD, *OpenStack Swift: Using, Administering, and Developing for Swift Object Storage.* " O'Reilly Media, Inc.", 2014.
[6] M. AYACHE, M. ERRADI, AND B. FREISLEBEN, *curlx: A middleware to enforce access control policies within a cloud environment*, in 2015 IEEE Conference on Communications and Network Security (CNS), pp. 771–772, IEEE, 2015.
[7] M. AYACHE, M. ERRADI, AND B. FREISLEBEN, *Access control policies enforcement in a cloud environment: Openstack*, in 2015 11th International Conference on Information Assurance and Security (IAS), pp. 25–30, IEEE, 2015.
[8] F. BLANQUI, *Introduction to the coq proof assistant*, Lecture notes available on https://who. rocq. inria. fr/Frederic. Blanqui, 2013.
[9] H. BOUAMOR, H. SAJJAD, N. DURRANI, AND K. OFLAZER, *Qcmuq@ qalb-2015 shared task: Combining character level mt and error-tolerant finite-state recognition for arabic spelling correction*, in ANLP Workshop 2015, p. 144, 2015.

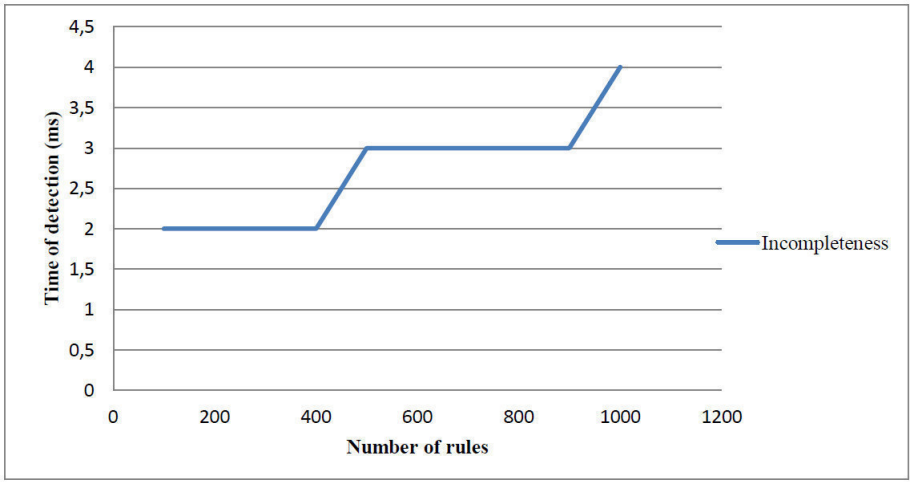


FIG. 9.3. *Performance of the Completeness Property Verification*

[10] O. CHOUDHURY, N. L. HAZEKAMP, D. THAIN, AND S. EMRICH, *Accelerating comparative genomics workflows in a distributed environment with optimized data partitioning*, in 2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), pp. 711–719, IEEE, 2014.

[11] J. A. EVANS, *Electronic medical records system*, July 13 1999. US Patent 5,924,074.

[12] K. FISLER, S. KRISHNAMURTHI, L. A. MEYEROVICH, AND M. C. TSCHANTZ, *Verification and change-impact analysis of access-control policies*, in 27th International conference on Software engineering, pp. 196–205, ACM, 2005.

[13] A. GAWANMEH AND A. ALOMARI, *Challenges in formal methods for testing and verification of cloud computing systems*, Scalable Computing: Practice and Experience, vol. 16, no. 3, pp. 321–332, 2015.

[14] A. GAWANMEH, H. AL-HAMADI, M. AL-QUTAYRI, S.-K. CHIN, AND K. SALEEM, *Reliability analysis of healthcare information systems: State of the art and future directions*, in 2015 17th International Conference on E-health Networking, Application & Services (HealthCom), pp. 68–74, IEEE, 2015.

[15] A. GOUGLIDIS, I. MAVRIDIS, AND V. C. HU, *Security policy verification for multi-domains in cloud systems*, International Journal of Information Security, vol. 13, no. 2, pp. 97–111, 2014.

[16] H. HU, G.-J. AHN, AND K. KULKARNI, *Discovery and resolution of anomalies in web access control policies*, IEEE Transactions on Dependable and Secure Computing, vol. 10, no. 6, pp. 341–354, 2013.

[17] F. HUONDER, *Conflict detection and resolution of xacml policies*, Master's thesis, University of Applied Sciences Rapperswil, 2010.

[18] W. JANSEN, *Cloud hooks: Security and privacy issues in cloud computing*, in 2011 44th Hawaii International Conference on System Sciences (HICSS), pp. 1–10, IEEE, 2011.

[19] A. KHOUMSI, W. KROMBI, AND M. ERRADI, *A formal approach to verify completeness and detect anomalies in firewall security policies*, in Foundations and Practice of Security, pp. 221–236, Springer, 2014.

[20] A. LI, Q. LI, V. C. HU, AND J. DI, *Evaluating the capability and performance of access control policy verification tools*, in Military Communications Conference, MILCOM 2015-2015 IEEE, pp. 366–371, IEEE, 2015.

[21] M. LORCH, S. PROCTOR, R. LEPRO, D. KAFURA, AND S. SHAH, *First experiences using xacml for access control in distributed systems*, in 2003 ACM Workshop on XML security, pp. 25–37, ACM, 2003.

[22] M. ST-MARTIN AND A. P. FELTY, *A verified algorithm for detecting conflicts in XACML access control rules*, in 5th ACM SIGPLAN Conference on Certified Programs and Proofs, Saint Petersburg, FL, USA, January 20-22, 2016, 2016.

[23] E. MARZINI, P. MORI, S. DI BONA, D. GUERRI, M. LETTERE, AND L. RICCI, *A tool for managing the x1. v1 platform on the cloud*, Scalable Computing: Practice and Experience, vol. 16, no. 1, pp. 103–120, 2015.

[24] A. MOURAD, H. TOUT, C. TALHI, H. OTROK, AND H. YAHYAOUI, *From model-driven specification to design-level set-based analysis of xacml policies*, Computers & Electrical Engineering 52, pp. 65-79, 2016.

[25] OASIS, Extensible access control markup language (xacml) version 2.0, 2005.

[26] M. OUZZIF, M. HAMDANI, H. MOUNTASSIR, AND M. ERRADI, *A formal modeling approach for emergency crisis response in health during catastrophic situation*, in International Conference on Information Systems for Crisis Response and Management in Mediterranean Countries, pp. 112–119, Springer, 2014.

[27] K. PEPPLE, *Deploying openstack.* " O'Reilly Media, Inc.", 2011.

[28] U. PERVEZ, O. HASAN, K. LATIF, S. TAHAR, A. GAWANMEH, AND M. S. HAMDI, *Formal reliability analysis of a typical fhir standard based e-health system using prism*, in 2014 IEEE 16th International Conference on e-Health Networking, Applications and Services (Healthcom), pp. 43–48, IEEE, 2014.

[29] U. PERVEZ, A. MAHMOOD, O. HASAN, K. LATIF, AND A. GAWANMEH, *Formal reliability analysis of device interoperability middleware (dim) based e-health system using prism*, in 2015 17th International Conference on E-health Networking, Application & Services (HealthCom), pp. 108–113, IEEE, 2015.

[30] C. D. P. K. Ramli, *Detecting incompleteness, conflicting and unreachability xacml policies using answer set programming*, arXiv preprint arXiv:1503.02732, 2015.

[31] A. Sureshkumar, M. De Vos, M. Brain, and J. Fitch, *Ape: An ansprolog* environment*, in First International Workshop on Software Engineering for Answer Set Programming (SEA), vol. 7, pp. 101–115, 2007.

Appendix A. Algorithms of Anomaly Detecing and Completness Verification.

Algorithm 2 Intra-policy Anomaly Detection

Input: Policy Automaton
Output: Redundancy Set $\mathcal{RS}$ and Conflicts Set $\mathcal{CS}$

procedure IsRedundant(Automaton) ▷ Verify if there are any redundant rules
  $nodes \leftarrow getFinalNodes(Automaton)$ ▷ Extract the final states from the automaton
  while $node.size \neq 0$ and $nodes \neq null$ **do**
    for $i \leftarrow 0, node.size$ **do**
      for $j \leftarrow i+1, node.size$ **do**
        if $node.get(i) = node.get(j)$ **then** ▷ The permissions of the final states are equal
          RS.add(i,j) ▷ Add both rules i and j to the redundant Set
        end if
      end for
    end for
  end while
  return $\mathcal{RS}$
end procedure
procedure hasConflict(Automaton) ▷ Verifies if there are any conflicting rules
  $nodes \leftarrow getFinalNodes(Automaton)$
  while $node.size \neq 0$ and $nodes \neq null$ **do**
    for $i \leftarrow 0, node.size$ **do**
      for $j \leftarrow i+1, node.size$ **do**
        if $node.get(i) \neq node.get(j)$ **then** ▷ The permissions of the final states are not equal
          CS.add(i,j) ▷ Add both rules i and j to the conflict set
        end if
      end for
    end for
  end while
  return $\mathcal{CS}$
end procedure

Algorithm 3 Inter-policy Anomaly Detection

Input: $Policy_1\mathcal{P}_1$, $Policy_2\mathrm{P}_2$
Output: Inconsistency Set $\mathcal{IS}$ and Similarity Set $\mathcal{SS}$

procedure InterPolicyAnalyzer(Automaton) ▷ Detects the inconsistency and the similarity
  $ProductAutomaton \leftarrow generateProductAutomaton(\mathcal{P}_1, \mathcal{P}_2)$ ▷ The automaton of the global policy
  $\mathcal{IS} \leftarrow IsRedundant(ProductAutomaton)$
  $\mathcal{SS} \leftarrow hasConflict(ProductAutomaton)$
  Return $\mathcal{IS}$ and $\mathcal{SS}$
end procedure

Algorithm 4 Verification of the Completeness Property

Input: Policy Automaton
Output: Verification of Completeness (C)

procedure isComplete(Automaton)
  $nodes \leftarrow getFinalNodes(Automaton)$
  while $node.size \neq 0$ and $nodes \neq null$ **do**
    for $i \leftarrow 0, node.size$ **do**
      if $node.get(i) = E_i$ **then** ▷ E_i represents a *non-match state*
        return The policy is not complete
      end if
    end for
  end while
end procedure

Appendix B. Proof of Proposition 6. We use the notation $\Psi_i = \min(2^{d_i};\, n)$. We omit the complexity of Step 1 because it needs a fixed, and finite time O(1)).

B.1. Complexity of Step 2. The space and time complexities to construct one state or one transition of $\mathcal{A}_i$ are in O(1). Each $\mathcal{A}_i$ contains 4 states and a limited number of transitions from each state. Hence, the space and time complexities to construct each $\mathcal{A}_i$ are in O(1). Since we have to construct n automata, the space and time complexities of Step 2 are in O(n).

B.2. Complexity of Step 3. This step consists in replacing each set of objects and subjects by the corresponding transitions. The number of transitions from r_i^j of $\mathcal{A}_i^*$ is bounded by both O(2^{d_j}) and O(n) which means O(Ψ_i). The bound O(2^{d_j}) is because 2^{d_j} is the number of possible values of either subjects or objects, which is necessarily $\geq$ than the number of transitions from r_i^j. Hence, the space and time complexities to construct all the transitions of $\mathcal{A}_i^*$ are in O($\Psi_0 + \Psi_1$). Therefore, the space and time complexities to construct all the $\mathcal{A}_i^*$ are in O($n \times (\Psi_0 + \Psi_1)$).

B.3. Complexity of Step 4. Let us consider the construction of $\mathcal{A}_F$ in Step 4 level by level, where the states of level i are those reached after i transitions from the initial state. At each level i, the transitions links level i-1 to level i. The space and time complexities to construct a state r = $<r_1;\ldots; r_n>$ of $\mathcal{A}_F$ are in O(n), because we need to construct and store the n components of the state. The space and time complexities to construct a transition between two constructed states of levels i and i+1 are in O(1), because we need to store the label of the transition.

Level 0: The unique state is the initial state $r^0 = <q_1^0;\ldots; q_n^0>$. The space and time complexities of its construction are in O(n).

Level 1: Using the same reasoning as in the proof of Step 3, the number of transitions from r^0 is in O(Ψ_0). Hence, the number of states at level 1 is in O(Ψ_0). Therefore, the space and time complexities to construct all the transitions from level 0 to level 1 are in O(Ψ_0), and the space and time complexities to construct all the states at level 1 are in O($n \times \Psi_0$).

Level 2: The number of transitions from each state of level 1 is in O(Ψ_1). Since the number of states of level 1 is in O(Ψ_0), we obtain that the number of states at level 2 and the number of transitions from level 1 to level 2 are in O($\Psi_0 \times \Psi_1$). Therefore, the space and time complexities to construct all the states at level 2 are in O($n \times \Psi_0 \times \Psi_1$), and the space and time complexities to construct all the transitions from level 1 to level 2 are O($\Psi_0 \times \Psi_1$). At each level j, the number of states is also bounded by 2^n, because each state is defined by n 2-value components r_i ($r_i = q_i^j$ or $r_i = E_i$, for $i = 1 \ldots n$). But this bound has no influence due to the assumptions n>D and $2^n > n^2$.

All levels: By adding the complexities of all levels, we obtain that the space and time complexities of constructing $\mathcal{A}_F$ are in O($n \times \Psi_0$)+ O($n \times \Psi_0 \times \Psi_1$).
From $d_i \geq 1$ and $n >$ D, we obtain $\Psi_i \geq 2$, from which we deduce that Ψ_0+($\Psi_0 \times \Psi_1$) $\leq 2\times(\Psi_0 \times \Psi_1)$. Hence, the space and time complexities of constructing $\mathcal{A}_F$ are in O($n \times \Psi_0 \times \Psi_1$).

Associating permissions: It remains to compute complexities of associating permissions to the match states of $\mathcal{A}_F$. The space complexity of associating a permission to a match state of $\mathcal{A}_F$ is in O(1), because we only need to store the permission associated to the state. The time complexity of associating permissions to all match states of $\mathcal{A}_F$ is in O(n), because we may need to consult the n components of the state.

B.4. Total complexity. Since Steps 1 to 3 are less complex than Step 4, we obtain that the space and time complexities for constructing $\mathcal{A}_F$ are in O($n \times \Psi_0 \times \Psi_1$). By definition of μ and δ, we obtain $n \times \Psi_0 \times \Psi_1 = n^{\mu+1} \times 2^{\delta}$, which can be easily shown to be smaller than both n^3 and $n \times 2^D$.

Edited by: Amjad Gawanmeh
Received: Feb 1, 2016
Accepted: Jul 3, 2016

DOI 10.12694/scpe.v17i3.1181
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 17, Number 3, pp. 207–226. http://www.scpe.org

RESOLVING CONFLICTING PRIVACY POLICIES IN M-HEALTH BASED ON PRIORITIZATION

SOUAD SADKI* AND HANAN EL BAKKALI†

Abstract. Mobile health has recently gained a lot of attention. Biological, environmental and behavioral data collected from mobile devices can be analyzed and transmitted directly to the person, family or health professionals for immediate and individualized care. However, due to multiplicity of mobile applications and the heterogeneity of actors involved in patient's care, conflicts among the privacy policies defined by the different actors can take place. Thus, we present in this paper an approach to resolve the problem of conflicting privacy policies in e-health/m-health environments using AHP (Analytic Hierarchy Process) prioritization technique. Conflicts detection and resolution are facilitated by the adoption of S4P formal privacy policy language used as a standardized language. Finally, a case study is suggested to illustrate how our solution can be applied to resolve such conflicts.

Key words: Privacy policy; Privacy preference; Conflicting policies; S4P; AHP

AMS subject classifications. 68N30

1. Introduction. The use of technology and electronic communications in healthcare environments, known as e-health, is significantly enhancing patients' quality of care. In fact, Electronic Health Records (EHRs) infrastructures are more enriched in order that the patient become more engaged with his own care, that way he is gradually moving away from a passive to an active role [22]. Particularly, with the emergence of m-health paradigm, as a sub-segment of e-health, and which refers to the use of mobile technologies such as smartphones and tablets, patients are more and more involved in managing their health using mobile applications or personalized services provided by healthcare organizations. More importantly, the lower cost, immediacy and the availability of mobile technologies allows patients accessing their medical history and easily communicate with their doctors whenever and wherever they are. Furthermore, thanks to mobile devices, it becomes so easy for physicians to download medical records, lab results, medical images, and drug information [1]. However, despite the important role mobile technologies play in enhancing patients' quality of care, they also present tremendous drawbacks including privacy violation. Particularly, with the rising number of actors (healthcare organizations, Cloud providers, external services) involved in patients' care, it becomes even harder to protect sensitive medical data but also to know who can or cannot access, collect or share this data across organizations. Hence, in order to ensure data privacy, the sharing, collection and management of medical data must be regulated using privacy policies [3]. These statements or legal documents contain some or all the ways a party manages users' data i.e. what information is collected, how it is collected and under what circumstances this information is used or stored. These privacy policies are expressed using various languages such as natural languages or formal ones like XACML[13], EPAL[14] or P3P[12]. However, the diversity of domains of application, the fixed vocabularies but also the different level of abstraction make these policies highly heterogeneous leading to conflicting situations [3]. Thus, resolving conflicts among privacy policies is of prime importance. Nevertheless, since data collected or shared via computers or mobile devices can be issued from different sources and can be stored in different locations, it is necessary to standardize the privacy policies defined by the different involved parties in order to take the right actions when a conflict occurs. As stated in [5], resolving this kind of conflicts can be very complex and time-consuming especially when the definition of a privacy policy involves more than one party, and the number of possible shared items as well as the entities that can have access to user's data is not fixed or predefined [5]. Some researchers [4-7] believe that the best technique for solving the problem of conflicting privacy policies is by negotiation. Some other works consider prioritization of one policy with respect to the other policy to be the most preferred technique. Still, to the best of our knowledge there is no mature work that properly addresses the issue of conflicting privacy policies in electronic or mobile health environments. We believe that patients' health condition is of prime importance. Because his privacy has a huge impact on his health and outcomes, he has the right to be notified of every action performed on his data. From this

*Mohammed V University,ENSIAS, Information Security Research Team,Rabat MOROCCO (souad.sadki@um5s.net.ma)
†Mohammed V University,ENSIAS, Information Security Research Team,Rabat MOROCCO (h.elbakkali@um5s.net.ma)

perspective, we suggest an approach to solve the aforementioned issue by prioritizing one policy with respect to the other using AHP technique. Our proposed solution adds a different view to the prioritization-based works by allowing an easy criteria extraction from the policy. Furthermore, to facilitate this task, we adopt the S4P language thanks to its flexibility and numerous advantages among which we specify the distinction between services' privacy policies and users privacy preferences. This distinction facilitates conflicts detection since the language syntax allows the satisfaction checking of a third party privacy policy over user's privacy preferences.

The **key contributions** of our work can be summarized as follows:

1. We propose a privacy-preserving approach for solving conflicts among privacy preferences/policies. This approach takes into account the major privacy-by-design principles.
2. We justify the adoption of the S4P languages compared to other languages. Thus, the use of S4P facilitates the translation and conflict detection tasks.
3. We adopt the AHP technique to prioritize the execution of one policy/preference with respect to the other policy/preference.

The rest of the paper is organized as follows: we describe the main problematic through a motivational example in Section 2. Section 3 presents an overview of AHP technique and S4P language followed by our approach description. Section 4 illustrates the efficiency of our solution in solving conflicting policies through a case study. Section 5 presents related work. In Section 6 we conclude the paper and present future work.

2. Problem statement.

2.1. Background. Mobile users are more and more anxious to get into the technology by downloading different computer-based and mobile applications. However, most of these users avoid reading long, complex and time-consuming privacy policies to well understand what of their data has been collected and how it will be used. Instead, they simply click on the 'I agree' without even paying attention to what they are agreeing to. The challenge is to make it as simple as possible for mobile users to define their privacy preference in a comprehensive way that make them avoid reading the complex privacy policies. As for patients, considered as particular users, since they are more and more integrated in the management of their care via computer-based or mobile applications, the communication of their data to different parties may increase their fear over their privacy. Above this, the heterogeneity of these applications as well as the actors involved in patients' care make it even harder to ensure patients' privacy. In particular, most of these actors may possess a privacy policy written in natural language (generally in English), so it should be translated to another language that could be easily understood and interpreted by the other actors. This translation facilitates the detection of any possible conflicts among these policies. In this work, we focus on the issue of conflicting privacy policies in e-health/m-health environments.

2.2. Problem Illustration. In this section, we describe the main problematic through a motivational example of three privacy policies in conflict. The main entities in our example are: The Patient (P), the Hospital (H) allowing patients to track and access their data via Electronic or Personal Health Records (EHR/PHR) and finally the Cloud Povider (CP) that provides storage services for the hospital. Also, the patient can benefit from Cloud-based mobile health apps as shown in Figure 2.1. Also, we assume that each policy is written in a different privacy policy language.

2.2.1. Policies description. Hospital's policy

We assume that H possesses a policy written in XACML language. An XACML policy consists of header information, an optional text description of the policy, a target, one or more rules and an optional set of obligation expressions [23]. Then, the XACML policy is defined in Figure 2.2 as follows.

Patient's preference. Patient's preference is expressed using P2U language. A P2U policy is formed of eight elements : *POLICY element* indicating information about the policy. A policy is created by a provider for a user and with one or more purpose(s) of use [24]. The PROVIDER element referring to the issuer of the privacy policy. USER element i.e. for whom the privacy policy is about [24]. PURPOSE element data sharing purpose, with whom it was shared, for how long it can be retained, and the kinds of data that is relevant for that purpose [24]. CONSUMER element The entity to whom the policy was addressed [24]. RETENTION element The time period (days) for which data can be retained [24]. DATA-GROUP element The group of data that can be shared.

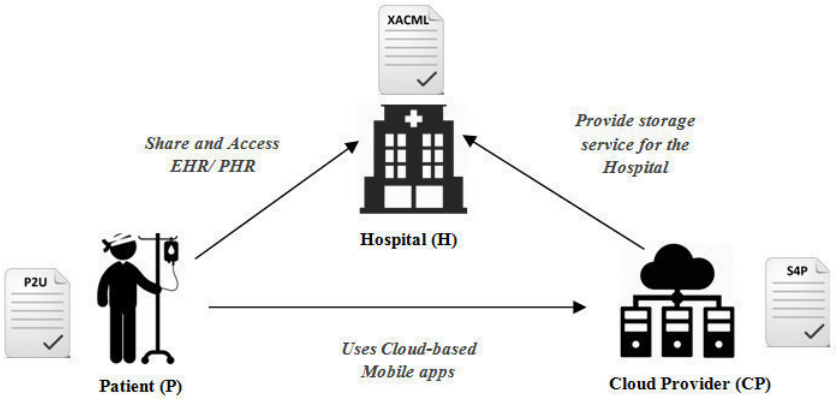


FIG. 2.1. *Ilustrative example architecture*

```
<!-- Permissions specifically for the researcher role -->
<Policy>
<PolicyId="Permissions:specifically:for:the:Researcher:role">
<Rule RuleId="Permission:to:use:medical:data"
Effect="Permit">
<Target>
<Subjects><AnySubject/></Subjects>
<Resources><Medical data/></Resources>
<Actions> <AttributeValue
DataType="&xml;string">Read</AttributeValue></Actions>
<Condition> < purpose="Medical research"/></Condition>
</Target>
</Rule>
</ PolicyId>
</Policy>
```

FIG. 2.2. *Hospital's policy in XACML*

Figure 2.3 presents Patient's privacy preference written in P2U.

Cloud Provider's privacy policy. Using S4P, we assume that Cloud Provider's privacy policy contains the following statement: *CP says CP will save your data for at least 1 year.*

2.2.2. Conflict description. In the first policy, the hospital authorizes any user with the role "Researcher" to use patients' health data for medical purposes whereas patient's policy permits only Health Workers to use/share patient's data (Electronic health record as shown in the attribute DATA) for only medical purposes. Hence the two policies are in conflict since, according to the patient P, researchers are not allowed to use patients' data even for medical reasons. As stated in CP's privacy policy, data can be retained for at least one year while patient's policy indicates that the time of data retention shouldn't exceed 90 days.

The conflict brings out the following concerns:

1. The need for an easy tool allowing the expression of patients' privacy preferences over any action regarding their data.
2. The need for a common privacy language. On one hand, the use of a common language facilitate the communication between the different entities and allows a better understanding of each involved entity's privacy rules on the other hand. Also, this language has to be simple, flexible and reflects both patient's privacy preferences and third parties' policies.
3. The need of a strategy to prevent and resolve any possible conflict.
4. The need of a reference or guidelines indicating the different requirements that should be taken into

```
<POLICY name= "Restrict_access_to_HealthWorker">
  <PROVIDER provID="er60z"/>
  <USER category ="Patient" userID="12" />
  <PURPOSE name="Medical_purpose_only" puID="102">
      <CONSUMER name="HealthWorker"/>
      <RETENTION period="90d" negotiable="TRUE" />
      <DATA-GROUP groupID="ty567" >
        <DATA ref="#medicalrecord" />
      </DATA-GROUP>
  </PURPOSE>
</POLICY>
```

FIG. 2.3. *Patient's privacy preferences in P2U*

account to favorise the execution of a policy, with respect to another one, in case a conflict takes place.

2.3. Overview of privacy laws and regulations. Privacy protection is a shared responsibility between patients, healthcare service providers and any organizations involved in patients' care. However, in order to protect patients' sensitive data from any possible theft or unauthorized use or disclosure, privacy laws and regulations are needed. In US, the Health Insurance Portability and Accountability Act (HIPAA) were created to improve the efficiency and effectiveness of the health care system, by encouraging the development of a health information system by establishing requirements and standards and for the electronic transmission of certain health information [25]. In the same context, the Personal Health Information Protection Act (Canada, 2004) was created with the objective of making patients more engaged with their care and protecting their privacy and the confidentiality of their personal health (PHI) information while facilitating the effective provision of health care by establishing rules for the collection, use and disclosure of PHI [26]. The Enhancing Privacy Protection Act 2012 of Austria defines thirteen privacy principles to protect Australian's personal information, example of these principles topics; Principle 1 open and transparent management of personal information, Principle 6 use or disclosure of personal information; Principle 9 adoption, use or disclosure of government related identifiers [27]. In the EU, the two main directives: the Data Protection Directive 1995/46/EC and the e-Privacy Directive 2002/58/EC [28] regulate the data protection.

3. Prioritization-based approach to resolve conflicting privacy policies. In this section, we describe our prioritization-based approach extending our previous works [2,10]. This work aims to solve the issue of conflicting privacy policies in e-health and m-health environments. We get inspired by the resolution strategy defined in [16] where the prioritization of one policy over another depends on how much that policy is specific in identifying the subject, the object, and the environment to which it is applicable [16]. In fact, in certain cases, it's preferable to prioritize SP's policy execution with respect to patients' preferences. The question is: when and under what circumstances the execution of third parties policies should be prioritized?

To respond to this question, we adopt the multi-criteria decision making AHP technique. The idea is that the execution of a privacy preference depends on the importance/relevance of the criteria extracted from the policies. For instance, let's consider the two following criteria 'purpose of usage' and 'patient's reputation', if the purpose of usage is equal to 'saving patients' life, then the criterion reputation is of lower priority even if it's an important criterion for patient. Also, sometimes access to patients' data is required by Law and the execution of a third party policy is prioritized even if this policy does not match patient's preference. For these reasons, it's of upmost importance to prioritize the execution of a given privacy policy over a privacy preference when a conflict takes place. Next, to formalize patients' and third parties' privacy preferences we adopt S4P language, a formal privacy language for specifying both users' and services' privacy policies.

3.1. S4P: A formal privacy Language. In this part, we justify the adoption of S4P as privacy language where we present its syntax and its advantages compared to other privacy policy languages suggested in the literature. In fact, the comparative study [10] we performed on a number of privacy policies including XACML,

P3P, EPAL and other languages suggested in the literature based on a number of criteria, shows that S4P responds to all the considered requirements. S4P language is human-readable, highly Expressive and can support parameterized behaviours, hierarchical data types, recursive relations, and arbitrary constraints [3]. More interestingly, it distinguishes between users' privacy preferences and services' privacy policies and allows the satisfaction checking between the two [3]. In addition, S4P authorizes delegation of authority which has a crucial role in healthcare. For this purpose, we adopt S4P as a privacy language to express both patients' and healthcare providers' policies. Moreover, thanks to S4P syntax and its flexibility in term of expressing the policies whatever the domain of application, S4P in our work will be used as a standardized privacy language.

S4P syntax. Policies and preferences in S4P are presented in a form of assertions and queries [3]. An assertion in S4P is defined as: $\langle$E says f_0 if $f_1 f_n$ where $C\rangle$ where E defines a user and the f_i are facts and C is a constraint on variables occurring in the assertion [3].
An example of an S4P assertion is : 'Alice says x may use data if x will revoke data within t where t$\leq$ 5 years'.

An S4P query q is defined as follows: Q:: == E says f? | c? | $\neg$ q | $q_1 \wedge q_2 \cup q_1 \sqcup q_2$ | $\exists$ x (q)[3]

An example of an S4P query is :'Alice says HP may share Personal Health Information with other healthcare providers?' [10]

Table 3.1 explains the difference between assertions and queries in S4P.

TABLE 3.1
Assertions and queries in S4P[4,10]

	User preferences	**Service Policy**
Permissions	*May-assertions*: User gives permissions	*May-queries* : Service asks for permissions
Promises	*Will-queries*: User asks for promises	Will-assertions: Service gives promises

Conflicts in S4P. As stated in the previous section, S4P allows the satisfaction checking between users' privacy preferences and services' privacy policies. Thus, using S4P, verifying if a patient's preference is in conflict with a service policy become easier. In fact, checking that a policy satisfies a preference consists of two steps. 1) Every behavior declared as possible in the policy must be permitted by the preference. 2)Every behavior declared as obligatory in the preference must be promised by the policy. In other words, the May-queries and Will-queries must be satisfied as indicated in the following condition [3]:

$$A_{pl} \cup A_{pr} \vdash q_m \wedge q_w \tag{3.1}$$

where A_{pr}, A_{pl}, q_m and q_w respectively designate a set of assertions in patient's privacy preferences, a set of assertions in service provider privacy policies, patient will-queries and service may queries [3].

3.2. Prioritisation with AHP. The Analytic Hierarchy Process(AHP) [8,9] is a multi-criteria and well-known decision making technique based on the evaluation of a set of criteria and alternatives to reach a specific goal. AHP returns thus the most relevant alternative with respect to the set of the pre-selected criteria [16]. Using pairwise comparisons, the relative importance of one criterion over another can be expressed using the ranking in Table 3.2. Also, 2, 4, 6, 8 values are used to represent compromise between the cited priorities .

As stated in [15], The AHP method is based on three principles: 1) model structure ; 2) comparative judgment of the criteria and/or alternatives and 3) synthesis of the priorities [15]. Figure 2.2 summarizes the different steps of this technique.

As shown in Figure 3.1, the first step consists on defining the important criteria. In general, in a decision making process, the criteria constitutes users' requirements. These criteria are grouped in a n*n Matrix called the criteria comparison matrix (C) where n is the number of criteria. The matrix C is then fulfilled using Table 3.1 and then normalized. Next, after determining the most relevant criteria and since the quality of the output of the AHP is related to the consistency of the pairwise comparison judgments [16], the next step consists on calculating the Consistency Ratio (CR) indicating if the matrix is completely inconsistent or if the comparison

TABLE 3.2
Fundamental Scale for AHP [16]

Intensity	Definition	Explanation
1	Equal	Two elements contribute equally to the objective
3	Moderate	One element is slightly more relevant than another
5	Strong	One element is strongly more relevant over another
7	Very strong	One element is very strongly more relevant over another
9	Extreme	One element is extremely more relevant over another

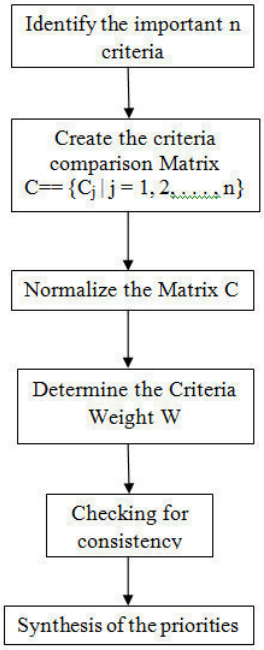


FIG. 3.1. *AHP main steps*

should be recalculated. Finally, the alternatives are evaluated where each alternative is compared with respect to each criteria. The different steps and computing technique are highlighted in the case Study (cf. Sect. 4).

3.3. System architecture and design goals.

3.3.1. System model. In our work, we consider the following entities:

- **Data owner (the patient)**: A patient using health IT (computer-based or mobile applications) to manage his health. In our previous work [2], we classified patients into four main categories as shown in Table 3.3. Thus, a patient can be a Fundamentalist (F), a Pragmatic (P), an Unconcerned (U) or a Should-be-Protected (SPr);
- **Healthcare service providers (Hospital or any entity providing health services)**: in our system a healthcare provider refers to any technology, service or companies offering mobile health apps for their patients;
- **External service providers**: These are entities or companies offering services for healthcare providers including insurance companies, Cloud providers.
- **A trusted third party:** A higher authority complying to privacy laws and standards and playing an intermediary role between patients and SPs. We assume that this party deals with all kind of patients and is responsible for:

-*Translating patients' preferences, HPS's and third parties' policies into Formal policies using S4P*;
-*Suggesting a list of local healthcare providers according to patient's privacy group*;
-*Detecting conflicts among privacy policies.* In fact, since privacy preferences and policies are expressed using a common language, conflict detection is done by applying the rule (1)
-*Extracting the different criteria from S4P policies and preferences in case a conflict occurs.*

TABLE 3.3
Privacy groups [2]

Privacy group	**Description**
Fundamentalist	Patients that distrust third parties to protect their privacy
Pragmatic	Patients who prefer to decide whether they should trust organizations or ask for legal procedures to protect their personal information
Unconcerned	Patients that trust health organizations or any third party to protect their private data.
Should-be-protected	Patients whom health condition does not allow them to make preferences. This group includes children that cant take proper decision and need a guardian or patient badly hurt

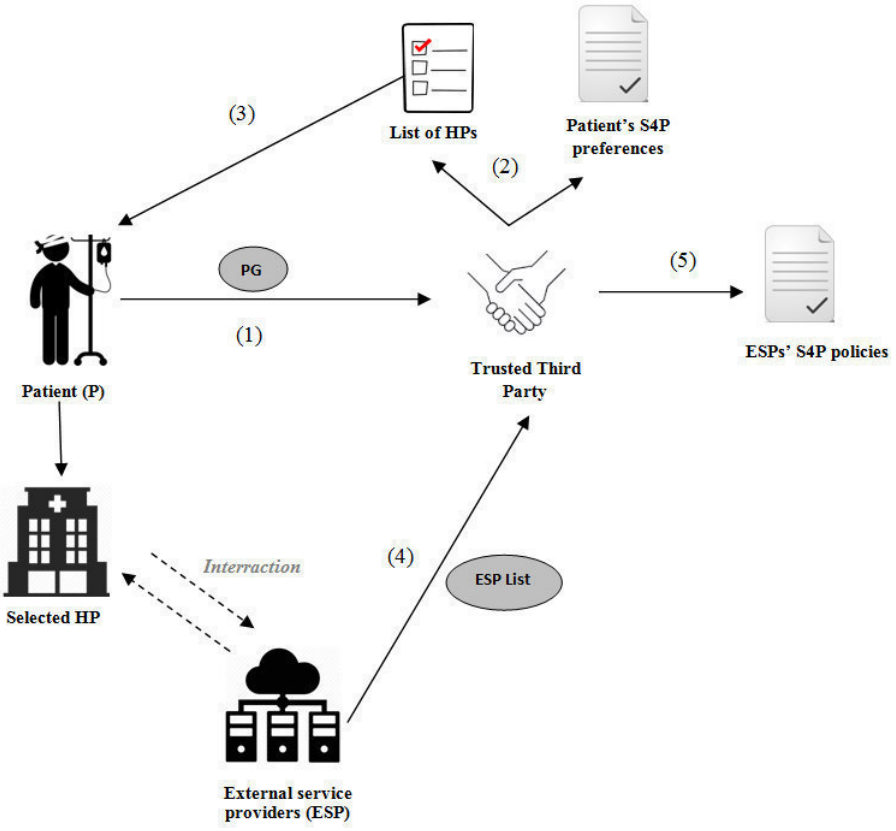


FIG. 3.2. *System architecture and main actors*

The interactions between the four entities are described (see Figure 3.2) as follows:
(1) and (2) : The trusted third party uses patient's privacy group (PG) to formalize his privacy preferences using S4P. As indicated in our previous work (See Figure 3.3) we suggested a Privacy Preserving Approach for Mobile Healthcare (PPAMH) [2] for automatically translating patients' privacy preferences into formal policies.

As shown in the figure, patients are asked to respond to a set of simple questionnaire defined in an intelligent mobile application where the preferences are deduced from patients' answers. The prediction operation is facilitated by grouping the patients into the four groups as stated in the previous section . Then, A set of rules indicating the subjects, the different attributes, constraints, access control decisions that should be taken is then generated and sent to a trusted third party that refers to the generated rules to define the formal privacy preferences. Figure 3.4 presents an overview of the intelligent application. Using the same privacy group and based of the reputation requirement, the TTP selects the local healthcare providers whom privacy policies go along with patient's category. The classification of the HP based on their reputation factor is subject of our future work.

(3) The generated list is then sent to the patient where he selects the HP satisfying his needs (nearest HP for instance). The selected healthcare provider interacts with a number of external entities offering different services.

(4) The list of these external services is then sent to the TTP that seek for their native privacy policies.

(5) In order to check if one of these policies doesn't satisfy patient's privacy preference, ESP's policies expressed in different ways are translated in formal policies using S4P. The idea of having all the policies/ preferences written in the same language facilitates the detection and the prevention of any possible conflict.

It is worth noting that the conflict in our work is provided by other third parties interacting with the healthcare provider 'chosen' by the patient.

3.3.2. Design goals. In order to solve the issue of conflicting privacy policies, we suggest an approach with the privacy-by-design following goals:

- Proactive: That said, in our work we try to prevent the conflict before it happens. In fact, the idea of seeking four healthcare providers responding to patient's need in term of privacy policy reduce the probability of conflict. Also, in case a conflict takes place our solution use this experience to notify the conflicting parties of possible change or improvement in their policy.
- Privacy as the default setting: In the classification of patients we suggested, the should-be-protected-group is considered as a default configuration for patient unable to decide themselves concerning their privacy preferences or patient of the category "Unconcerned".
- Privacy embedded into the design: In every step of the approach we aim to preserve patient's privacy.
- Transparency and visibility: Patient can themselves decide regarding their privacy preferences. As for external services, the fact that they are notified about actions regarding the execution of policies make our solution transparent and visible.
- Respect to user privacy: The integration of the intelligent mobile application allowing a personalized privacy preferences determination and the fact that we select a language distinguishing between patients' policies and users policies make our work patient-centric.

3.4. Application of AHP to determine the prioritized policy/ reference. In this section we answer the question: when and under what circumstances the execution of third parties policies should be prioritized?

The priority of execution in our work is related to:

1. The purpose of usage/disclosure of patients data. Example of purposes: Marketing, research, Law Enforcement, Communication with family, spread of a dangerous disease.
2. Patients' privacy group: In fact, patient's privacy group helps determining the possible important criteria for the patient. For instance for a patient who does not intent to share his information with other parties, we can deduct that the important criteria for this patient are: reputation, time of retention, purpose of usage, collection or divulgation.

In order to resolve a conflict, we consider the following steps:

A. The important criteria identification. Before applying the AHP technique to resolve a possible conflict, it's of up-most importance to determine the different criteria and requirements that should be taken into account. We assume that the definition of these criteria is performed a high authority taking into account the rules and practices defined in privacy laws and regulation. We assume that the considered criteria are: category of data, purpose, reputation, type of organization, time of retention. As stated in the previous section, the classification of patients in four groups in term of privacy preferences plays a crucial role in determining the important criteria defined in Patient's S4P privacy preferences. The structure of S4P policies and preferences

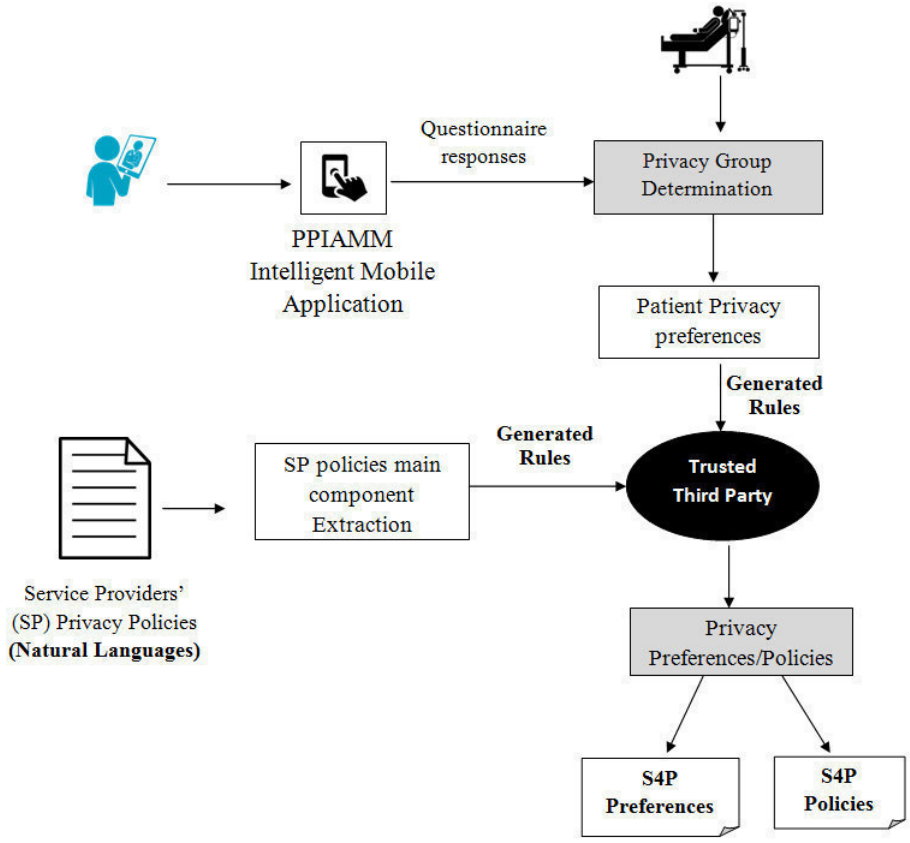


FIG. 3.3. *Privacy prefences policy formalization process [10]*

facilitate the conflict detection as well as the criteria extraction operation as indicated in Table 3.4. We assume that this superior entity is responsible for affecting the initial values (intensity) to the different criteria taking into account patients' condition, rules enforced by law, environment factors etc.

TABLE 3.4
Example of S4P preference/privacy and the associated criteria

Privacy Policy/preference	**Type**	**Extracted criteria**
Alice says **HP** may use Personal Health Information for research purposes?	may-query	Purpose
Alice says **HP** may use Cookies for x if **HP** will revoke Cookies within t where t $\leq$ 1yr	may-assertion	Time of retention
Alice says HP may share Personal Health Information for treatment purposes only [10]	may-assertion	Purpose
Alice says **x** can say **HP** may acess EHR if **x** complies with HIPAA	Delegation of authority	Laws and regulation

B. Creation of the Criteria comparison matrix. A Criteria comparisons Matrix C or a pairwise comparisons matrix is a square matrix which has positive entries and it is reciprocal,i.e., for each element $C_{i,j} = 1/C_{j,i}$ [16].

C. Normalizing the matrix C. Normalizing the matrix means to divide each element in every column by the sum of that column and calculating the average in every column. We obtain the normalized matrix CN.

D. Determination of the most important criteria. We average each row in the normalized matrix CN.

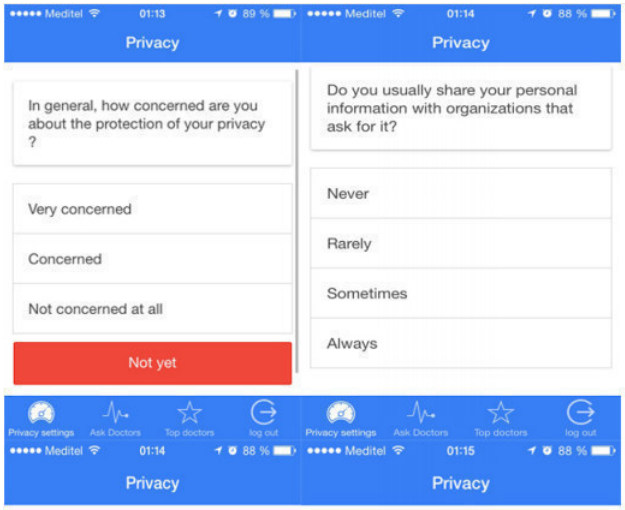


FIG. 3.4. *Example of the PPIMAM intelligent Mobile app*

This average is called "Criteria weight" ,CW. In fact, the highest value of CW constitutes the most important criteria in our design.

E. Checking for consistency. Consistency means that the ranking defined in the Matrix C makes sense. Otherwise, the ranking value should be redefined. In fact, this step requires the computation of a number called CI (consistency index) and then the consistency ratio CR = CI/RI; with a random index called RI, which is a predefined value for each number of criteria. The number 0.1 is the accepted upper limit for CR. If the final consistency ratio exceeds this value, the evaluation procedure has to be repeated to improve consistency. The Algorithm 1 explains the different steps of CR computation and the consistency checking.

Algorithm 1: Consistency checking

input : **C** Comparison Matrix, **n** number of criteria
output: **CS**:Boolean `/* CS= 1 means that C is consistent */`

`/* Calculation of the weight sum vector WS */`
WS = C * CW;
`/* Calculation of the consistency ratio CV */`
CV = WS * (1/CW);
`/* Calculation of the consistency index CI */`
CI = ($\sum CV_{i,j}$-n)/(n-1);
CR = CI/RI;
if $CR < 0{,}1$ **then**
  CS ⟵ 1; `/* The matrice C is consistent */`
end
else
  CS ⟵ 0; `/* The matrice C is inconsistent */`
  Recalculate(C)
end

F. Synthesis of the priorities. Given our two privacy policies, the last step consists on comparing this two alternatives with respect to the n initial criteria. We obtain the matrix AV (alternative value) that will next be multiplied by 1/CW. The final result gives a higher and a lower score. The alternative having the higher value is the policy/preference that should be executed. Thus, as shown in Figure 3.5, we distinguish between

two possible scenarios:

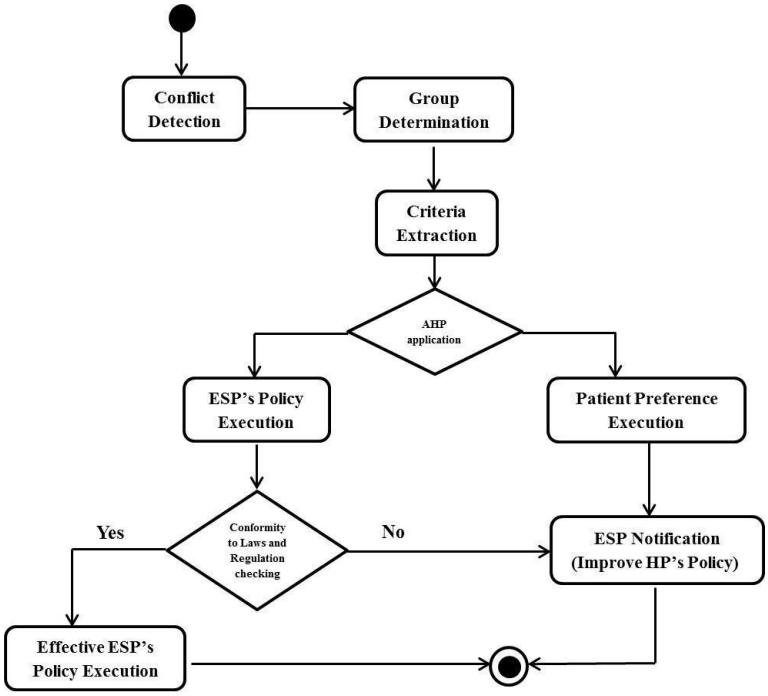


FIG. 3.5. *The approach main steps*

a) The application of AHP indicates that the ESP's privacy policy is executed. In this case TTP verify if this policy is conform to standards and regulations. If it is the case, ESP's policy is executed. Otherwise, TTP notify ESP in question to improve its policy in order to avoid possible future conflicts. It is worth noting that adopting S4P as standard also facilitates the satisfaction checking between ESP's policies and the policies imposed by standards like HIPAA.

b) The application of AHP indicates that Patient's privacy preference is favored. In this case, the ESP is notified of the conflicting situation. In fact, this notification allows third parties to improve their privacy policies statements, especially if the number of conflicts increases continuously.

4. Case Study. In order to illustrate how our approach can be applied to resolve conflicting policies in ehealth/mhealth environments, we consider the conflict scenario suggested in our previous work [10]. As indicated in [10], three main entities are considered:

1) A pediatric medical center in US; Arkansas Childrens Hospital(ACH) [20]; playing the role of a healthcare provider (HP) and which propose a mobile application ,MyACH, to access children's medical history, get information about specific health symptoms and other medical services.

2) CloudTech technologies [21] playing the role of an external service provider that a healthcare provider such as Arkansas can deal with.

3) A 16 years old patient in Arkansas hospital; Bob; whose mother (guardian); Alice; is responsible for any decision or operation regarding his health.

In what follows, our AHP-based approach will be used to resolve the conflict among patient's privacy preferences, healthcare provider's privacy policy and Cloud provider's privacy policy. Since our solution is patient-centric, Bob's privacy preferences (expressed by his guardian) are well defined. Indeed, S4P will be used to formalize bob's privacy preferences and the two providers' privacy policies.

Step 1: Criteria Extraction

The criterion extraction step is preceded by two main steps which are: Privacy/policy formalization and conflict detection.

a) Privacy/policies formalization. Before extracting the different criteria from policies. It is of upmost importance to translate Bob's privacy preferences, ACH hospital's and CloudTech technologies' privacy policies into formal policies using S4P.

Bob's privacy preference: As stated in the previous Section, as the patient group is a key element in our approach, the first step consists on defining this group based on simple questionnaire in a form of intelligent mobile application [2] and that patient is asked to answer. Since Bob is minor, his mother answers these questions in his behalf [10]. We assume that Alice is of the category "Fundamentalist". In other words, she is very strict regarding the sharing, usage and collection of her child. Figure 4.1 presents an example of Bob's privacy preference.

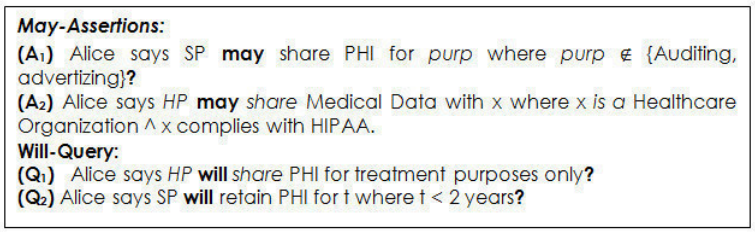


FIG. 4.1. *Bob's privacy preference in S4P [10]*

CloudHealth technologies' privacy policies: CloudHealth technologies' policy are taken verbatim from online CloudHealth technologies' privacy policy. We consider then the following statements:

'We only store data about you for as long as it is reasonably required to fulfill the purposes under which it was first provided by you unless a longer retention period is required or permitted by law' [21].

'We may also use personal information for internal purposes such as auditing, data analysis and research to improve our products' [21].

Table 4.1 shows the translated policies in S4P.

TABLE 4.1
Extract of CloudHealth technologies privacy policy in S4P

English policies	**Translated S4P policies**	**Type**
(A_3)	CloudHealth says CloudHealth will use data for purp where purp ∈ { auditing, data analysis and research }	Will-assertion
(A_4)	CloudHealth says CloudHealth will store personal information for t where t is undetermined	Will-assertion

ACH privacy policy: An extract of ACH Online Privacy policy is taken verbatim [20] and translated into S4P formal privacy polices as shown in Table 4.2.

'We may share some of your PHI with outside people or companies who provide services for us'

'We must disclose your PHI to government authorities that are authorized by law to receive reports of suspected child abuse or neglect involving children or endangered adults'

b) Conflicts detection. As mentioned before, a conflciting situation in S4P means that the may-query in the policy or/and the will-query in the preference are not satisfied where queries are evaluated against the union of the assertions in the policy and the preference [3].

To verify if a conflict takes place, we evaluate each of the three queries Q_1 ,Q_2 ,Q_3 over the union of the will-assertions and may-assertions. i.e. $A_1 \cup A_2 \cup A_3 \cup A_4 \cup A_5$. Table 4.3 presents the result of the evaluation of the queries over the assertions. As shown in Table 4.3, the preference (Q_1) is conflicting with the assertions (A_3) and (A_5). In fact, in the preference (Q_1) Alice expresses her desire of sharing her son's medical data for

TABLE 4.2
Extract of ACH privacy policy in S4P

English policies	**Translated S4P policies**	**Type**
(A_5)	ACH says ACH will share your PHI with TP for purp if TP is a government agency where purp $\in$ { child abuse, neglected children, endangered adults}.	Will-assertion
(Q_3)	ACH says ACH may share PHI with outside services?	May-query

medical purposes only. Whereas in the assertions (A_3) and (A_5) ACH and CloudHealth technologies indicates that data can be shared for other purposes. In what follows, we try to find out which of the three conflicting policies is of higher priority using ACH technique.

TABLE 4.3
Evalution of queries against assertions

Query	Satisfied	**Assertions causing conflict**
(Q_1)	No	(A_3) and (A_5)
(Q_2)	No	(A_4)
(Q_3)	No	(A_2)

c) Criteria Extraction. Obviously, the extraction of different criteria from the policies is facilitated thanks to the structure of S4P policies/preferences (facts, constraint, conditions). Concerning the criteria ranking the patient privacy group will help indicating the important criteria for patient even if it is not mentioned in the preference. For instance, for a Fundamentalist patient (Alice's case), we can deduce that the criterion reputation is to be considered as important even if it is not mentioned in patient's preference. Thus, the criterion reputation will be considered during the comparison. Furthermore, other criteria can be extracted from other conflicting situation apart from (Q_1) ,(A_3) and (A_5). For example, we can deduce the criteria 'Time of retention' from the conflict among Alice query (Q_2) and CloudHealth technologies' policy (A_4).

TABLE 4.4
Extracted Criteria from conflicting policies

Preference/policy	**Extracted criteria**
(Q_1)	Purpose , type of data , **reputation**
(A_3)	Purpose
(Q_5)	Purpose, type of data, type of organization, Laws and regulations

Step 2: Application of AHP to resolve the conflict among (Q_1), (A_3), (A_5)

a) The criteria comparison Matrix creation. Using the ranking defined in Table 3.2, we assume that the relevance of each criterion compared to the other criteria is defined as shown in Table 4.5.

b) Determination of the most important criterion. Using BPMSG AHP priority calculator [19],we obtain the priority vector P as indicated in Table 4.6.

As shown in Figure 4.2 presented using online BPMSG AHP priority calculation system [19], the criterion 'Purpose' has the highest value, so the purpose of usage is of higher priority compared to criteria 'Laws and regulation', 'Data category, 'Reputation', 'Type of organization' and 'Time of retention'.

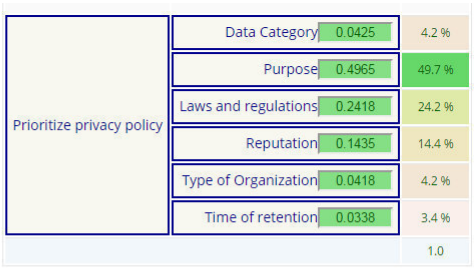
c) Consistency checking. Figure 4.3 indicates that CR=0.048452 $\leqslant 0,1 \Rightarrow$ The matrix is consistent. In other words, the initial value considered for the different criteria were well defined.

TABLE 4.5
Criteria Comparison Matrix C

Criteria	Cat. D	Purp.	Laws and Reg.	Rept.	Tyoe of Org.	TOD.
Category of data	1	1/9	1/7	1/5	1	2
Purpose	9	1	3	5	7	9
Laws and Regulation	7	1/3	1	3	5	7
Reputation	5	1/5	1/3	1	5	5
Type of organization	1	1/7	1/5	1/5	1	1
Time of retention	1/2	1/9	1/7	1/5	1	1

TABLE 4.6
Priority decimal values and priority vector

Criteria	Cat. D	Purp.	Laws and Reg.	Rept.	Tyoe of Org.	TOD.
Category of data	1.0000	0.11111	0.1428	0.2000	1.0000002	2.000
Purpose	9.000	1.000	3.000	5.000	7.000	9.000
Laws and Regulation	7.000	0.3333	1.000	3.000	5.0000	7.000
Reputation	5.000	0.200	0.3333	1.000	5.000	5.000
Type of organization	1.000	0.14285	0.2000	0.200	1.000	1.000
Time of retention	0.5000	0.1111	0.1428	0.200	1.000	1.000
Priority Vector (P)	0.04355	**0.47402**	0.2570	0.1480	0.04277	0.0346



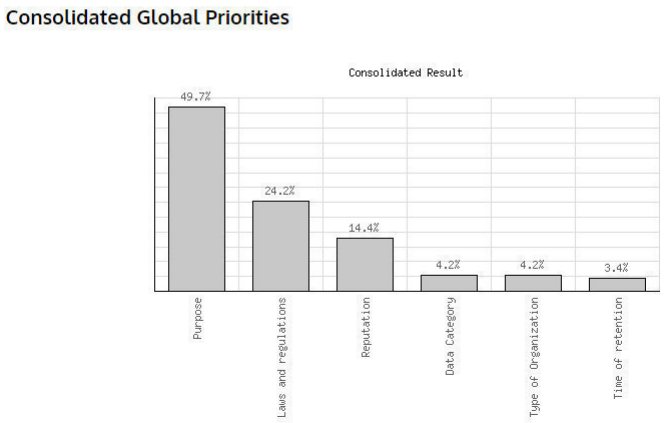


FIG. 4.2. *Criteria Priorities*

FIG. 4.3. *Computation of CR*

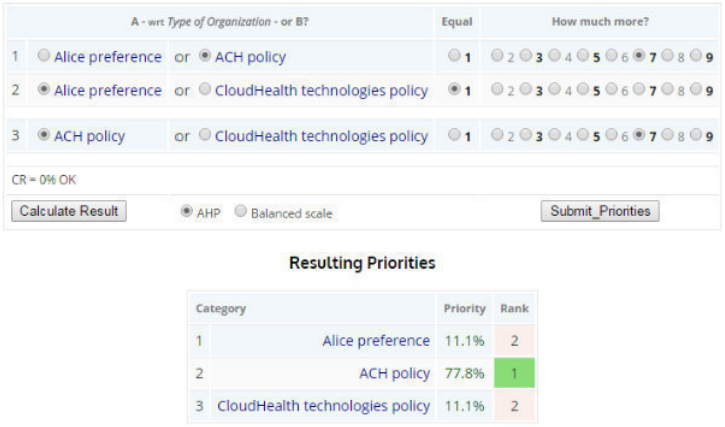


FIG. 4.4. *Comparison of the three policies with respect to the criteria Type of organization*

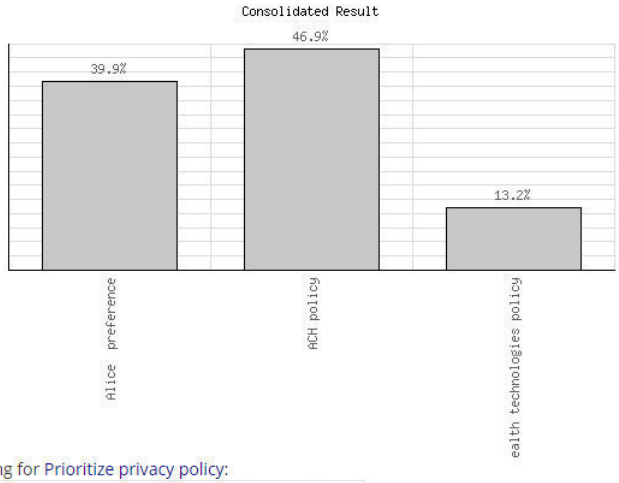
d) Evaluation of the three policies. Now we need to define which of the three policies will be executed. For this purpose we need to evaluate the three alternatives where we compare each of the three S4P policies with respect to the six criteria using AHP technique.

Figure 4.4 presents an example of the comparison of three policies with respect to the criteria "Type of organization" using BPMSG AHP priority calculator system [19]. As shown in Figure 4.4 the criteria type of organization is less important in Alice's preference and CloudHealth technologies' policy than it is in ACH policy.

	Criterion	Node	Glb Priorities	Compare	Alice preference	ACH policy	CloudHealth technologies policy
1.	Data Category	Prioritize privacy policy	4.2%	AHP	0.481	0.405	0.114
2.	Purpose	Prioritize privacy policy	49.7%	AHP	0.458	0.416	0.126
3.	Laws and regulations	Prioritize privacy policy	24.2%	AHP	0.111	0.778	0.111
4.	Reputation	Prioritize privacy policy	14.4%	AHP	0.714	0.143	0.143
5.	Type of Organization	Prioritize privacy policy	4.2%	AHP	0.111	0.778	0.111
6.	Time of retention	Prioritize privacy policy	3.4%	AHP	0.498	0.135	0.367
				Total weight of alternatives:	0.399	0.469	0.132

FIG. 4.5. *Ranking of the three alternatives*

Result for Alternatives



Ranking for Prioritize privacy policy:

	Category	Priority	Rank
1	Alice preference	39.9%	2
2	ACH policy	46.9%	1
3	CloudHealth technologies policy	13.2%	3

FIG. 4.6. *Graphical representation of policies Ranking*

Indeed, if we go back to ACH S4P statements we notice that Government authorities (type of organization) has a strong meaning in the policy. So, we proceed in the same way with the other criteria. Next, using AHP online system, we obtain the result defined in Figure 4.5.

We use the same steps followed in Step 2 to determine the most preferred policy. Finally, as shown in Figure 4.6 ACH policy has a higher priority of 46.9 % compared to Alice preference 39.9 % and CloudHealth Technologies policy 13.2 %.

5. Related Work. In the literature, many privacy-preserving and user-centric works [29, 30, 31, 32, 33, 34, 35, 36, 37] in various domains have been suggested. However, most of these works do not consider the totality of privacy-by-design principles [39]. Also, even if users privacy preferences are taken into account, other

factors such as the multiplicity of actors, the geographic factors or the location of sensitive data can be the cause of privacy violation. For this reason, privacy policies were created regulating the different operations applied to users' sensitive information. As for the medical field, it is important to consider patients as active actors; they have the right to be informed and to take part in any decision regarding the sharing, storage and use of their private and sensitive data. Thus, their privacy preferences need to be properly formalized and taken into account by the different health stakeholders. Nevertheless, these preferences may not be respected by other third parties leading to conflicting situations.

Detecting and solving conflicts among security and privacy policies have recently attracted a lot of attention in different fields. In fact, in order to resolve such conflicts, many approaches and techniques have been adopted. Motivated by the relevance of these works, we developed our approach for healthcare. Indeed, most of the solutions suggested rely on the negotiation technique to resolve such conflicts. Exemplary, authors in [17] suggest a reputation-based approach that makes use of common interest to define other entities having negotiated the same issues in the past, from whom the negotiator can learn the possible offers and counter-offers that could be made to negotiate with the user [17]. In the same context, a negotiation-based approach introducing a policy negotiation point (PNP) between the policy enforcement point (PEP) and the policy decision point (PDP) was suggested in [18] adopting XACML as a privacy language. Other recent negotiation-based works [5-6-7] have been proposed, but noun of them consider the user as an important actor by involving him in the negotiation process. Thus, the challenge is on including the owner of the information (patient in our case) in the decision-making without really forcing him to read complex privacy policies. Furthermore, another work was suggested in [42] describing a policy based authorization infrastructure and a conflict resolution strategy between the different policy decision points. The major particularity of this work is that access to data is always controlled since the users privacy policies are stuck to their data even if this data is shared between Cloud providers or other services [42].

Another category of authors use techniques such as AHP to prioritize the execution of a policy over another. In this context, authors in [16] use a strategy to solve conflicting policies; the prioritization of one policy over another depends on how much that policy is specific in identifying the subject, the object, and the environment to which it is applicable [16]. In the same context, a prototype for solving conflicts in XACML-based e-Health policies was suggested in [6]. Evaluation is done to determine which among the conflicting policies, defines a more specific set of policy elements [6]. In addition to these works, a novel solution for privacy conflict detection and resolution for collaborative data sharing in online social networks was suggested in [43]. The proposed solution considers privacy-sharing tradeoff by quantifying privacy risk and sharing loss [43].

Nevertheless, in order to ensure the effectiveness of these solutions in solving conflicts, they need to be tested before adoption to avoid any possible fault. Indeed, any error in the conception or the usage of a healthcare system may put patients life at risk [44]. From this perspective, a classification of existing testing and verification of healthcare solutions was suggested in [44]. The authors distinguish between three main categories: simulation based methods, formal methods based on mathematics models, and other techniques such as semi-formal methods based on formal syntax and allowing informal semantics [44]. Our work is characterized by adopting a formal language, S4P, to express privacy policies and a formal decision-making methodology, AHP, to resolve the issue of conflicting policies. Still, in order to improve our approach by reducing possible errors in interpreting and translating patients preferences and then extracting the main criteria from these preferences in case a conflict occurs, there is a need for formal validation techniques. For this purpose, model checking can be used. On the other hand, since there are so many actors involved in patients care, most of them with complex architecture, there is a need for formal models that help in verifying and testing the correctness of these systems [45]. In this context, an axiomatic model defining the formal specification requirements for healthcare systems and was suggested in [45]. In particular, with the massive use of Cloud computing services and the emergence of new paradigms such as Big data, healthcare systems are more and more complex. As an example of a Cloud-based work in healthcare, a framework called X1.V1 was suggested in [22] aiming at optimizing the resource utilization on the Cloud by facilitating the exploitation of the cloud elasticity [22]. Thus, using formal methods to enhance the related design issues of Cloud-based systems need to be properly addressed. In this regard, a survey on use of formal methods on testing and verification of Cloud systems was proposed in [50]. Still, and because these systems are considered hybrid where heterogeneous entities collaborate and many

technologies are being used at the same time, there is a lack of a single verification framework for healthcare systems [44].

In addition, it is extremely important to consider the interoperability between the different healthcare actors. For this purpose, health standards like Fast Health Inter-operable Resources (FHIR) [48] have been developed regulating the exchange, integration, sharing and retrieval of electronic medical information and strengthening the reliability of health systems [47]. These standards also need formal methods like model checking to analyze their performance, reliability and functionality [47]. In this regard, a formal probabilistic analysis approach based on the PRISM [46] model checker was proposed in [47] aiming to find the probability of occurrence of wrong results following the FHIR standard. By adopting a formal state-based model for the FHIR, the approach provides more accurate results while allowing additional failures checks and therefore enforcing the reliability of the FHIR standard [47].In the same context, an approach for testing the correctness of device interoperability middleware (DIM) was suggested in [49]. Authors propose to use PRISM model checker to evaluate reliability properties like probability of success and failures and thus take proper measures to design a better DIM. Remarkably, the importance of following health communication standards is also shown in this work where authors use HL7 FHIR standard to illustrate the effectiveness of their proposed solution. Thus, it becomes crucial to follow such standards particularly in distributed environments where data intensive-tasks can be parallelized to improve system performance [51]. Hence, by including FHIR standard into our system design we guarantee an effective and regulated communication between the main actors in our approach especially between the trusted third party and the external service providers. Also, it is of up-most importance to consider compliance to FHIR standard in parallel with privacy conflict checking. That said compliance to FHIR standard need to be classified as crucial criteria in case a conflict take place.

6. Conclusion. In this paper, we presented a technique, applied in e-health/m-health environments, to prioritize the execution of one privacy policy with respect to another when the two policies are in conflict. For this purpose, we adopt the AHP technique and the S4P formal language. This work is an extension of our previous works [2,10] aiming to automatically generate the privacy policies using the notion of *privacy policy group*. The most particularity of our work compared to other AHP-based solutions is the ease of criteria extraction from the policy thanks to S4P syntax structure. Furthermore, the determination of the criteria importance is facilitated by classifying patients into groups in term of privacy preferences. Also, our work respects the major privacy-by-design principles.

As a future work, a comparative study of the most relevant criteria in recent privacy policies mobile health applications will be performed. This study will allow us to develop an automatic mechanism to extract this considered criteria from S4P formal policies. Furthermore, the translation of policies into formal S4P policies will be automated and tested. Finally, the approach will be improved by including an initial step helping patients selecting the most preferred healthcare provider or cloud-based services. For this, the experience of highly reputable parties having offering similar services in the past will be used. The whole system will be evaluated using a formal verification and validation technique.

REFERENCES

[1] B. M Silva, J. Rodrigues, I. De la Torre Díez, M. Lopez-Coronado, *Mobile-Health: A Review of Current State in 2015*, Journal of Biomedical Informatics, June 2015.

[2] S. Sadki, H. El bakkali, *PPAMH: A novel privacy-preserving approach for mobile healthcare*, In Proceedings of the 9th International Conference for Internet Technology and Secured Transactions, IEEE, London December 2014.

[3] L.Y.B. Moritz, M. Alexander, B. Laurent, *AS4P: A Generic Language for Specifying Privacy Preferences and Policies*, Technical report MSR-TR-2010-32, Microsoft Research,2010.

[4] A.A.Datir, A. Sahu,*A Review on Enhancing Privacy Preservation of Web Service through Negotiation Mechanism*,International Journal of Current Engineering and Technology, 2015.

[5] J.M. Such, M. Rovatsos, *Privacy Policy Negotiation in Social Media*,Journal of social and Information Networks December 2014.

[6] A. Lunardelli, I. Matteucci, P. Mori, M. Petrocchi,*A Prototype for Solving Conflicts in XACML-based e-Health Policies*, IEEE 26th International Symposium on Computer-Based Medical Systems (CBMS),2013.

[7] D. Abi Haidar, *Negotiation of sensitive resources using different strategies for policys protection*, Proceedings of the International Conference on Security and management 2013.

[8] T.L. Saaty, *Decision-making with the AHP: Why is the principal eigenvector necessary*. European Journal of Operational Research, 2003
[9] T.L. Saaty, *Decision making with the Analytic Hierarchy Process*. In the International Journal of Services Sciences, 2008.
[10] S. Sadki, H. Bakkali, *A Negotiation-based Approach to Resolve Conflicting Privacy Policies in M-Health*, In International Journal of Information & System Management,2015.
[11] L. Cranor, B. Dobbs, S. Egelman, G. Hogben, J. Humphrey, M. Langheinrich, M. Marchiori, J. Presler-Marshall, M. Reagle, D. Schunter, A. Stampley, R. Wenning *The Platform for Privacy Preferences 1.1 (P3P1.1) Specication*. W3C, 2006
[12] L. Cranor, B. Dobbs, S. Egelman, G. Hogben, J. Humphrey, M. Langheinrich, M. Marchiori, J. Presler-Marshall, M. Reagle, D. Schunter, A. Stampley, R. Wenning *The Platform for Privacy Preferences 1.1 (P3P1.1) Specication*. W3C, 2006
[13] OASIS. *eXtensible Access Control Markup Language (XACML) Version 2.0 Core specification*, 2005
[14] P. Ashley, S. Hada, G. Karjoth, C. Powers, M. Schunter M. *Enterprise Privacy Authorization Language (EPAL 1.2)*. Technical report, IBM, November, 2006
[15] A. Görener, *Comparing AHP and ANP: An Application of Strategic Decisions Making in a Manufacturing Company*, International Journal of Business and Social Science, 2012.
[16] I. Matteucci, P. Mori, M. Petrocchi, *Prioritized execution of privacy policies*, Data Privacy Management and Autonomous Spontaneous Security Volume 7731 of the series Lecture Notes in Computer Science pp 133-145, 2013.
[17] G. Yee, L. Korba, *PThe Negotiation of Privacy Policies in Distance Education*, published in Proceedings. 4th International IRMA Conference. Philadelphia, Pennsylvania, USA, May 18-21, 2003. NRC 44985.
[18] S.Vivying, Y. Cheng, C. Patrick, K. Hung, K. Dickson, W. Chiu, *Enabling Web Services Policy Negotiation with Privacy preserved using XACML*, Proceedings of the 40th Hawaii International Conference on System Sciences - 2007
[19] http://bpmsg.com/academic/ahp_calc.php
[20] Arkansas Children's Hospital. *Joint Notice of Privacy Practices*. http://www.archildrens.org/About-ACH/Privacy-Practices.aspx, revised August 29, 2013.
[21] CloudHeath technologies. *Privacy policy*. http://www.cloudhealthtech.com/privacy. January, 2014
[22] E. Marzini, P. Mori, S. Di Bona, D. Guerri, M. Lettere, L. Ricci, *A Tool for Managing the X1.V1 Platforn on the Cloud*, Scalable Computing: Practice and Experience, Vol. 16, 2015.
[23] OASIS Standard, *eXtensible Access Control Markup Language (XACML) Version 3.0*, 22 January 2013
[24] J. Iyilade, J. Vassileva, *P2U: A Privacy Policy Specification Language for Secondary Data Sharing and Usage*, IEEE Security and Privacy Workshops (SPW), pp. 18-22, 2014
[25] *Health Insurance Portability and Accountability Act of 1996 Public Law 104-191 104th Congress*
[26] *Personal Health Information Protection Act*, 2004, S.O. 2004, c. 3, Sched. A
[27] *Australian Privacy Principles and National Privacy Principles Comparison Guide Summary and analysis of key differences for organisations*, April 2013
[28] European Commission, *A comprehensive approach on personal data protection in the European Union*, Brussels, 4.11.2010
[29] S. Haas, S. Wohlgemuth, I. Echizen, N. Sonehara, G. Müller *Aspects of privacy for electronic health records*. International journal of medical informatics 80, e26e31, 2011
[30] J. Li, *Electronic Health Records and the Question of privacy*. Computer, 2013
[31] K. Renaud, D. Gálvez-Cruz, *Privacy: Aspects, Definitions and a Multi-Faceted Privacy Preservation Approach*, Information Security for South Africa (ISSA), 2010
[32] L. Chen, J.-J. Yang, Q. Wang, Y. Niu, *A framework for privacy-preserving healthcare data sharing*, IEEE 14th International Conference on e-Health Networking, Applications and Services (Healthcom), 2012
[33] L. Pang, M-h. S, S-s. Luo, B. Wang, Y. Xin, *Full privacy preserving electronic voting scheme*, The Journal of China Universities of Posts and Telecommunications, 2012
[34] R. Gajanayake, R. Iannella, T. Sahama, *Privacy Oriented Access Control for Electronic Health Records*, Worldwide Web Conference, 2012
[35] D. Jiazhu, L. Shuangyan, L. Hongxia, *A Privacy-preserving Access Control in Outsourced Storage Services*, Computer Science and Automation Engineering , Vol :3, pp 247-251, 2011
[36] B. Lage Srur, V. Muthukumarasany, *Enhancing Trust on e-Government : A decision Fusion Module*, Third International Conference on Network and System Security, 2009
[37] G. Bella, R. Giustolisi, S. Riccobene, *Enforcing privacy in e-commerce by balancing anonymity and trust*. Computers and Security, 30(8), pp 705-718, 2011
[38] Information and Privacy Commissioner of Ontario, Canada *The Roadmap for Privacy by Design in Mobile Communications: A Practical Tool for Developers, Service Providers, and Users*, 2010
[39] M. Assunta Barchiesi, R. Costa, M. Greco, *Enhancing conflict resolution through an AHP-based methodology*, International Journal of Management and Decision Making, volume 13, issue 1, 2014
[40] E. J. Sobczyk, J. Badera, *The problem of developing prospective hard coal deposits from the point of view of social and environmental conflicts with the use of AHP method*, VERZITA, Tom 29, 2013
[41] L. Antonio Bojórquez-Tapia, *A Continual Engagement Approach through Gis-MCDA: Conflict Resolution of Loggerhead Sea Turtle Bycatch in Mexico*, PA51B-2212, December 2015
[42] D.W. Chadwick, K. Fatema, *A privacy preserving authorisation system for the cloud*, Journal of Computer and System Sciences, Vol 78, pp. 13591373, 2012
[43] H. Hongxin , A. Gail-Joon, J. Jorgensen, *Detecting and Resolving Privacy Conflicts for Collaborative Data Sharing in Online Social Networks*, Proceedings of the 27th Annual Computer Security Applications Conference, pp. 103-112 , 2011

[44] A. Gawanmeh, H. Al-Hamadi, M. Al-Qutayri, S. Chin and K. Saleem, *Reliability analysis of healthcare information systems: State of the art and future directions*,Proceedings of the IEEE 17th International Conference on e-Health Networking, Applications and Services (Healthcom), pp. 68 - 74, 2015
[45] A. Gawanmeh, *An Axionatic Model for Formal Specifictaion requirements of Ubiquitous Healthcare Systems*, Proceedings of IEEE 10th Consumer Communications and Networking Conference (CCNC), pp. 898 - 902, 2013
[46] M. Kwiatkowska, G. Norman, D. Parker, *PRISM 4.0: Verification of Probabilistic Real-time Systems*, In International Conference on Computer Aided Verification, volume 6806 of LNCS, pp. 585591. Springer, 2011
[47] U. Pervez, O. Hasan, K. Latif, S.Tahar, A. Gawanmeh, M-S Hamdi, *Formal Reliability Analysis of a Typical FHIR Standard based E-Health System using PRISM*, Proceedings of the 1st International Workshop on Reliability of eHealth Information Systems - IEEE HEALTHCOM, pp. 43 - 48, 2014
[48] FHIR. http://www.hl7.org/implement/standards/fhir/
[49] U. Pervez, A. Mahmood, O. Hasan, K. Latif, A. Gawanmeh,*Formal Reliability analysis of Device Interoperability Middleware (DIM) based E-health system using PRISM*, 2015 IEEE 17th International Conference on e-Health Networking, Applications and Services (Healthcom): The 2nd International Workshop on Reliability of eHealth Information Systems (ReHIS), pp. 108 - 113, 2015
[50] A. Gawanmeh, A. Alomari, *Challenges in Formal Methods for Testing and Verification of Cloud Computing Systems*, Scalable Computing: Practice and Experience, Volume 16, Number 3, pp. 321332, 2015
[51] O. Choudhury, N. L. Hazekamp, D. Thain, S.J. Emrich, *Acceleratin Comparative Genomics Workfkows in a Distributed Environmnet with Optimized Data Partictioning and Workflow Fusion*, Scalable Computing: Practice and Experience, Volume 16, Number 1, pp. 5369, 2015

Edited by: Amjad Gawanmeh
Received: Feb 16, 2016
Accepted: Jul 15, 2016

DOI 10.12694/scpe.v17i3.1182
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 17, Number 3, pp. 227–235. http://www.scpe.org

FORMAL VERIFICATION OF A MICROFLUIDIC DEVICE FOR BLOOD CELL SEPARATION

AMJAD GAWANMEH*, ANAS ALAZZAM†, AND BOBBY MATHEW†

Abstract. Blood cell separation microdevices are designed in biomedical engineering for separation of cancer cells from blood. The movement of cancer cells particles in a continuous flow microfluidic device is a challenging problem since there are several forces incorporated. For instance, forces due to inertia, gravity, buoyancy, dielectrophoresis and virtual mass are accounted for in this system. Understanding the cell particle movement and behavior at high level of abstraction is necessary in order to avoid fundamental errors in the design of systems that can make use of this behavior. In this paper we use formal analysis in order to formalize and validate the movement of microparticles under DEP forces for blood cell separation microdevice. This is achieved by modeling the dynamic behavior that can predict the trajectory of microparticles as a transition state based system. The model is used to validate the correctness of the microdevice at early stages of the design process.

Key words: Reliability Analysis, Medical System, Microfluidic Device, Blood Cell Separation

AMS subject classifications. 92C17, 92C50, 92-08, 68Q60

1. Introduction. Dielectrophoresis (DEP) is the phenomenon in which neutral but polarizable particles, dispersed in a medium, transverse when subjected to a non-uniform electric field. The particles transverse towards either the field maxima or minima; the preference of maxima/minima depends on electrical properties, specifically conductivity and permittivity, of the microparticles and medium as well as applied frequency. DEP is used very frequently in order to design techniques that are used for separating microparticles in a heterogeneous mixture. In these microdevices, the sample containing microparticles is subjected to a DEP field, applied normal to the direction of flow, thereby repelling the microparticles away from the electrodes. In most microfluidic devices the DEP field is generated in the vertical direction and this leads to the microparticles being distributed along the height when subjected to DEP. The height to which each microparticle is repelled depends on the properties such as conductivity, permittivity and density of the medium and the microparticle itself.

Existing microparticles separation techniques are usually validated through a set of experimental data, and then results are compared to the theoretical model. Simulation however, cannot provide full coverage for complex systems, since huge number of test cases are needed. Therefore, other complementary testing and verification techniques such as formal method are often used. Formal methods, in particular, model checking, involve a systematic analysis that is based on mathematical reasoning to verify that design specifications comprehend certain design requirements. They have been successfully used for the precise analysis of various complex systems [10], therefore, they can be efficiently used to validate the separation of microparticles in a continuous flow microdevice at high level of abstraction. NuSMV model checking method is used in this work in order to formalize and validate the movement of blood cells under DEP in a microfluidic device employing dielectrophoresis for purposes of blood cell separation.

Model checking [7] (or sometimes called property checking) is a formal verification technique that verifies whether a model of a system meets a given specification. The method provides exhaustive coverage for the system and is conducted automatically. It examines all possible system states in a brute-force manner in order to show that a given system model truly satisfies a certain property. Model checking approach has a limitation related to the size of system states that can be checked, therefore, abstraction methods are used to enable the verification of complex and large systems. In this, paper we use the NuSMV model checker, which provides cutting-edge formal verification methods based on optimized techniques. It supports both temporal model checking including CTL and LTL temporal logics, and safety assessment, and has been used in several industrial contexts. Therefore, we will use NuSMV [13] in order to model and verify the microdevice used for blood cell separation.

*Department of Electrical and Computer Engineering, Khalifa University, UAE. and Department of Electrical and Computer Engineering, Concordia University, Montreal, Canada. (amjad.gawanmeh@kustar.ac.ae).
†Department of Mechanical Engineering, Khalifa University, UAE.

This paper extends the work in [19] by using model checking for the formal analysis of the movement of blood cells in microdevices within a heterogeneous mixture of human blood for cell separation. This formalization helps in identifying certain features about the behavior of these blood cells, for instance, whether they are affected by specific design parameters, such as microchannel height, while in the transient state. This model also provides an early understanding of the behavior of the system at high level of abstraction, and therefore can provide early feedback to the designers of DEP microdevice.

2. Related Work. The first attempt for modeling the trajectory of cells in a DEP microfluidic device, employing IDT electrodes, is credited to Huang *et al.* [22]. In this work the authors developed an analytical static model for determining the levitation height of cells subjected to nDEP in a microfluidic device. As with static models the model developed by Huang *et al.* is valid only under steady state conditions. The model accounts for forces due to DEP, gravity, buoyancy and hydrodynamic lift. For purposes of developing an analytical equation the force due to DEP is modified from its original form, in terms of gradient of the square of the magnitude of electric field, to a function in applied voltage (RMS). They did not consider the forces due to inertia, drag and virtual mass. The authors experimentally validated this model using data from their own experiments.

Kralj *et al.* [24] modeled the trajectory of microparticles in a continuous flow nDEP based microfluidic device, with IDT electrodes, they developed for sorting of microparticles. The electrodes are located at the bottom of the microchannel but aligned at an angle to side walls of the microchannel. The purpose of this microdevice is to sort the microparticles in the lateral direction, i.e. along the width of the microchannel. Unlike traditional DEP microdevices where microparticles are sorted based on material properties alone, the microdevice of Kralj *et al.* can achieve sorting based on the size of microparticles as well. This is because the sorting is achieved in the lateral direction rather than in the vertical direction. The authors modeled the trajectory of microparticles only in the lateral direction for which they considered the forces due to drag and DEP. In this model the electric field is approximated using a trigonometric function for realizing an analytical equation of the trajectory of the microparticles. Crews *et al.* [14] carried out a numerical study of the IDT electrodes for the purpose of developing an approximate mathematical equation of the gradient of square of the magnitude of electric field inside the microchannel for use in estimating the force due to DEP. This equation is a function of electrode and gap length; in addition, it implicitly accounts for the height of the microchannel. According to Crews *et al.* (2007) their equation is applicable only for voltages lower than 8 V (peak-to-peak) as well for electrode/gap lengths smaller than $80\mu m$.

Cao *et al.* [11] developed a model for describing the levitation of microparticles in a DEP microfluidic device employing IDT electrodes. The work only considered the forces due to DEP, gravity, buoyancy and drag in their model. The influence of electrothermal flow on the trajectory of microparticles is accounted though the drag force. Leu and Weng [26] developed an analytical equation for the predicting the levitation height of microparticles in a DEP-FFF microdevice. They considered the forces due to DEP, gravity and buoyancy. In this model they used an existing analytical equation for electric field for calculating the square of the magnitude of electric field.

Neculae *et al.* [30] carried out a numerical study of the trajectory of nanoparticles subjected to DEP, both pDEP and nDEP, in a continuous flow microfluidic device with IDT electrodes. The electric potential inside the microchannel is determined numerically. This is followed by the calculation of the nanoparticle trajectory by equating the forces associated with drag and DEP; the force due to DEP is approximated using a mathematical expression to simplify the calculation. The authors did not consider the influence of forces due to inertial, gravity and buoyancy. In addition, they observed that a specific nanoparticle, when subjected to nDEP, translated towards the same final location irrespective of its initial location along the height of the microchannel. Lam *et al.* [25] developed a three dimensional model of the trajectory of cells in a microfluidic device employing DEP for purposes of sorting cells; the model is numerically solved and experimentally validated. The work in [5], used a microfluidic device that employs interdigitated electrodes, on the bottom surface of the microchannel, for separation of cancer cells from blood.

To overcome the above-mentioned inaccuracy limitations of simulations, formal methods have been proposed as a viable solution[20]. They are primarily based on computer-based mathematical analysis methods to model and analyse the given system. A lot of work has been done in the domain of analyzing heathcare systems

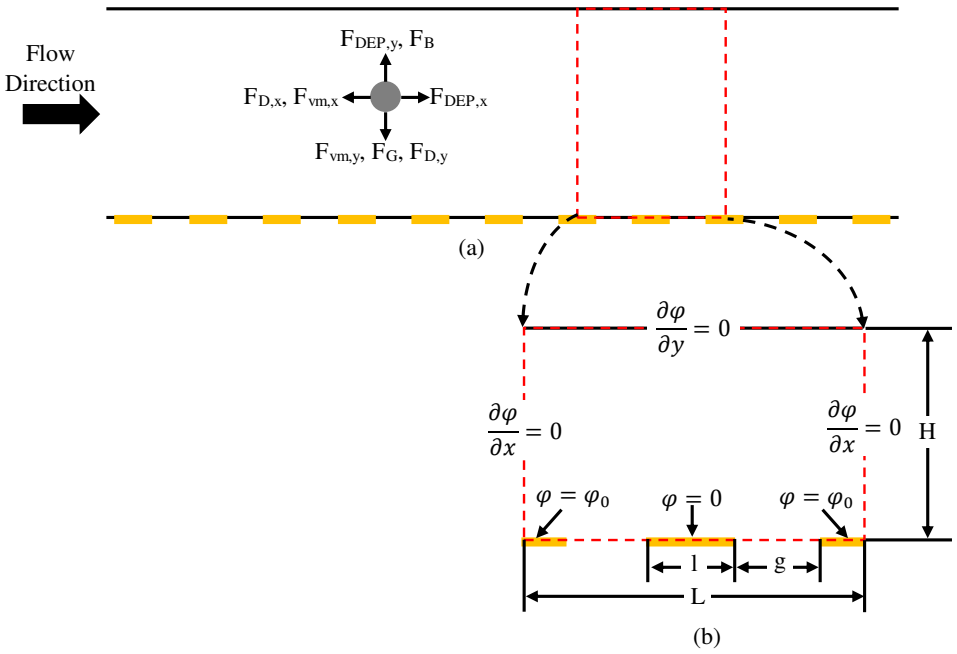


FIG. 3.1. *(a) Schematic of a section of the microchannel embedded with electrodes and forces acting on the microparticle; dashed line represents the repeating unit with respect to the electrodes; (b) detailed schematic of repeating unit with boundary conditions.*

using formal methods. Some notable examples include the verification of electrocardiogram (ECG) biosensors in event-B [2, 1]. The work is then extended to formalize the rules that reflects the construction of the ECG wave specifications [3, 4]. In addition, reliability analysis of FHIR standard based e-health system was addressed in [31, 32]. Other works include the verification of software components in medical devices [6, 16] and [29, 15], ambient assisted systems [8] or healthcare requirements [17] and the verification of collaborative and agent based workflows in healthcare [9, 21]. Other work related to managing medical wrokflow was presented in [28] and [12].

Medical devices are considered critical, since faults and errors in the medical system may lead to loss of lives, and in the best cases, loss of money and reputations [18]. To the best knowledge of the authors, this work presents the only effort at dynamic modeling of the trajectory of microparticles in a microdevices at high level of abstraction. The model presented in this work includes all forces relevant to the translation of microparticles in a microdevice, including that due to inertia, drag, DEP, virtual mass, gravity and buoyancy. In contrast to a static model a dynamic model is necessary for relating the time duration associated with the blood cell to reach the steady state position as well as the corresponding axial displacement. Moreover, the presented model can be used for parametric study of the microdevices with different positions of electrodes, which can be beneficial to designers of such microdevice.

3. Specification of Blood Cell Separation Device. Existing microparticles separation techniques are usually validated through a set of experimental data, and then results are compared to the theoretical model. In previous work, we used a microfluidic device that employs interdigitated electrodes, on the bottom surface of the microchannel, for separation of microparticles. We have also derived mathematical models for modeling movement particles in microchannels under the influence of DEP. Figure 3.1 represents the schematic of the microchannel considered in this work, where the IDT electrodes are located on the bottom surface of the microchannel. Under steady state conditions, the electric field and electric potential inside this repeating unit are provided by Khoshmanesh *et al.* 2011 [23] and Zhang *et al.* 2010 [33]. The system specifications assumes

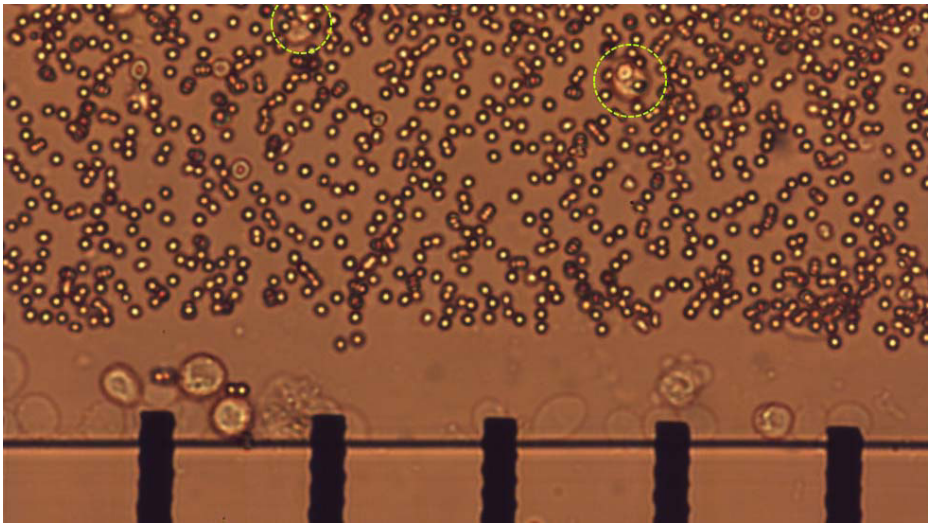


FIG. 3.2. *Blood cell separation image from microdevice shows that certain cancer cells could not be captured.*

that there are no electric charges inside the microchannel. The force due to DEP depends on the gradient of the Electric field and thus the need for non-uniform electric field. The force due to DEP also depends on the radius and electrical properties of the microparticles, the electrical properties of the medium and the applied frequency. The forces in Figure 3.1 are defined as drag F_D, DEP force F_{DEP}, virtual mass force F_{vm}, gravity force F_G and buoyancy force F_B.

Several assumptions were considered in this model, most were were initially provided by Loth (2000) [27]. First, blood cell particles are assumed to be spherical and rigid. It is also assumed that there is only one way coupling between particles and the medium as well as between particles and the electric potential. The particles are also assumed to be much smaller than the depth of the microchannel. There are no interactions between different particles, and no interactions between particles and the wall. Finally, the microparticle is not subjected to rotation about its axis while in transition.

According to Figure 3.1, a particle flows in the blood is affected by eight different forces. Some of these forces are of constant value, such as, F_G, other forces are imposed by the fluid flow, such as F_{vm}. The main force that will be used to guide the particle into the separation path is enforced through F_{DEP}. In our formal analysis, we assume that F_{DEP} is controlled by the separation device, and the rest are controlled by other factors, such as blood flow and location of the blood cell particle. In the next section, we provide formal modeling and analysis for the separation system.

Figure 3.2 shows a view of separated blood cell from the flow. We observed that some infected blood cells could not be captured in the separation process. Therefore, in the next step, we intend to conduct performance analysis on the system, that will help providing statistical information about the ability of the microdevice to capture infected cells. This can help in improving the performance of the microdevice by changing certain design parameters such as Electric field strength, frequency or electric cathode position and dimensions.

4. Formal Analysis of Microparticle Separation Device. Formal methods has proved to provide a complete coverage, and are becoming fundamental for the certification of such different types of systems. Model checking [7] or property checking is a formal verification technique where we check exhaustively and automatically whether a model of a system meets a given specification. Model checking examines all possible system states in a brute-force manner in order to show that a given system model truly satisfies a certain property. The main drawback of model checking approach is the limited size of number of system states that can be checked, therefore, abstraction methods are used to enable the verification of complex and large systems. In this paper we use the NuSMV model checker, which provides cutting-edge formal verification methods based on optimized techniques. It supports both temporal model checking including CTL and LTL temporal logics,

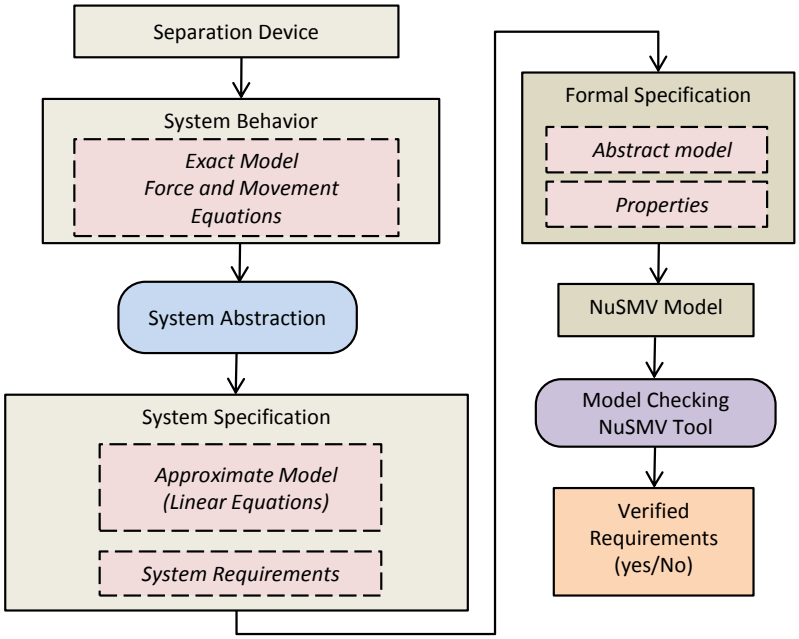


FIG. 4.1. *Separation Device Formal Verification Methodology*

and safety assessment, and has been used in several industrial contexts. Therefore, we will use NuSMV in order to model and verify the microparticle separation device.

Figure 4.1 illustrates the verification method for blood cell separation systems, where the behavior of the given system is usually described using an exact model with a set of system equations that describes particle movement and applied forces. In order to conduct formal reliability analysis of such a system, we first apply abstraction in order to simplify the system behavior. This results in a linear model of the system with a set of system requirement to be satisfied. It shall be noted here that system approximation is not intended to be used for design purposes, but only for the transformation of the system into a model that can be formally analyzed. Then, a NuSMV model is obtained, along with the desired system properties. The underlying verification tool is used in order to validate whether the requirement is satisfied or not.

In order to be able to describe the system under verification using NuSMV language, the system needs to be described as a transition based system, where variables are abstracted within finite integer ranges. This abstraction should be in conformance with the above system specifications. The blood cell separation system is described using several state variables that represent the particle behavior and characteristics including, its position, speed and forces that affect its movement. In fact, all variables are normalized with regard to the formal analysis model. First we define the set of ranges for variables of the blood cell septation system in NuSMV as shown below, where $MAXF1$, for instance, represents the upper bound of $FDEp$ force in both x and y directions.

```
MODULE main
VAR
  FDEPy : 0.. MAXF1;      FDEPx : 0.. MAXF1;
  Fvmx  : 0.. MAXF2;      Fvmy  : 0.. MAXF2;
  FDx   : 0.. MAXF3;      FDy   : 0.. MAXF3;
  posx  : 0.. MAXPx;      posy  : 0.. MAXPy;
  FB    : 0.. MAXF4;      PSx   : 0.. MAXSx;
```

Next, we model the state variables of the separation device. First we show how F_{DEP_x} and F_{DEP_y} are updated by changing the force level incrementally or detrimentally according to the microparticle movement. This models the application of Electrical field on the fluid flow, where value of the electrical field change according to the distance inside the fliud device.

```
next(FDEPy) := case
 (posx > POSxT & posy < TARyL & FDEPy < MAXF1 - 1):
       FDEPy + 2;
 (posx > POSxT & posy < TARyL ) : MAXF1;
 (posy < TARyL & FDEPy < MAXF1) : FDEPy + 1;
 (posx > POSxT & posy > TARyH & FDEPy > 2):
       FDEPy - 3;
 (posx > POSxT & posy > TARyH ) : 0;
 (posy > TARyH & FDEPy > 0)     : FDEPy - 1;
 (FDEPy < MAXF1) : FDEPy + 1;
 TRUE            : FDEPy;
esac;

next(FDEPx) := case
 (posy >= TARyL &  posy <= TARyH) : FDEPx;
 ((posy < TARyL |  posy <= TARyH) & posx > POSxT &
       FDEPx > 2) : FDEPx - 2;
 ((posy < TARyL |  posy <= TARyH) & FDEPx > 0) :
                    FDEPx - 1;
 (FDEPx > MAXF1 / 2) : FDEPx - 1;
 TRUE            : FDEPx;
esac;
```

Next, we show how the particle speed in both directions, x and y are calculated based on the total forces enforced on the particle, where C represent a constant related to the particle mass and force normalization.

```
next(PSx) := case
 (PSx + (FDEPx - Fvmx - FDx)*C) < MAXSx &
     (PSx + (FDEPx - Fvmx - FDx)*C) > 0 :
              PSx + (FDEPx - Fvmx - FDx)*C;
 (PSx + (FDEPx - Fvmx - FDx)*C) >= MAXSx:
             MAXSx;
 (PSx + (FDEPx - Fvmx - FDx)*C) <= 0 : 0;
 TRUE          : PSx;
esac;

next(PSy) := case
 (PSy + (FDEPy + FB - Fvmy - FDy - FG)*C) < MAXSy &
   ((FDEPy + FB - Fvmy - FDy - FG)*C) > 0 :
            PSy + (FDEPy + FB - Fvmy - FDy - FG)*1;
 (PSy + (FDEPy + FB - Fvmy - FDy - FG)*C) >= MAXSy:
            MAXSy;
 (PSy + (FDEPy + FB - Fvmy - FDy - FG)*C) <= 0 : 0;
 TRUE          : PSy;
esac;
```

Next, we show how the particle coordinates are calculated based on the speed of the particle in both directions, where DX represent the difference in distance from previous location, and is equal to particle speed multiplied by time unit.

```
next(posx) := case
  (posx + DX < MAXPx & posx + DX > 0) : posx + DX;
  posx + DX >= MAXPx  : MAXPx;
  TRUE          : 0;
esac;

 next(posy) := case
  (posy + DY < MAXPy & posy + DY > 0) : posy + DY;
  posy + DY >= MAXPy  : MAXPy;
  TRUE          : 0;
 esac;
```

In order to validate the model of the particle movement considered in this work, we formalize a property about particle separation in the microdevice as follows:

Property. *The microparticle will eventually move in the* x *direction under the application of a positive electric field. The particle will be successfully separated in it hits the target x coordinate between upper threshold y_h and*

lower threshold $y - l$ on the y-coordinate. This property is modeled in NuSMV as follows:

```
SPEC AG((posx = TARx) ->   posy >= TARyL & posy <= TARyH)
```

In order to conduct analysis on the microdevice, we assumed discrete sets of possible values for particle parameters, and we also assumed that all parameters are normalized to one. Table 4.1 below shows the results of the analysis of microparticle movement using different parameters for particles and results of verification, where separation is successful or unsuccessful. The value $True$ illustrates that the particle with the given parameters will be separated successfully, while the value $False$ states the opposite. P_{y_0} represents particle initial position in the y-coordinate, P_{x_0} is assumed to be 0 for all particles, since it is the beginning of the device microchannel where the fluid flows. PS_{x_0} and PS_{y_0} represents particle initial speed in x and y coordinates, respectively. Table 4.2 shows the results for different location of the separation unit. The results shows that the separation success depends on several issues, first the location of the separation unit, second the particle initial position and speed in both directions. In addition, the results show that separation will be unsuccessful only if particles enter the flow at specific positions with certain initial speeds. While the used model checking framework can show correctness of the property for all given values, it cannot be used to provide statistical analysis about the behavior of the microdevice. This is an open issue that can be addressed at later stage of the work.

TABLE 4.1
Analysis of particle separation at first location

P_{y_0}	PS_{x_0}	PS_{y_0}	Separation
$\{0, 0.01, \ldots, 0.60\}$	$\{0, 0.1, \ldots, 1\}$	$\{-1.0, -0.8, \ldots 1.0\}$	True
$\{0.61, 0.62, \ldots, 0.80\}$	$\{0, 0.2\}$	$\{-1.0, -0.8, \ldots 1\}$	False
$\{0.61, 0.62, \ldots 0.80\}$	$\{0.3, 0.4, \ldots, 1\}$	$\{-1.0, -0.8, \ldots, 1.0\}$	True
$\{0.81, 0.82, \ldots, 1\}$	$\{0.0, 0.1\}$	$\{-1.0, -0.8, \ldots, 1.0\}$	True
$\{0.81, 0.82, \ldots, 1\}$	$\{0.2\}$	$\{-1.0, -0.8, \ldots, 0.8\}$	False
$\{0.81, 0.82, \ldots, 1\}$	$\{0.2\}$	$\{1.0\}$	True
$\{0.81, 0.82, \ldots, 1\}$	$\{0.3, 0.4, \ldots, 0.6\}$	$\{-1.0, -0.8, \ldots, 1.0\}$	True
$\{0.81, 0.82, \ldots, 1\}$	$\{0.7, 0.8, \ldots, 1.0\}$	$\{-1.0, -0.8, \ldots, 1.0\}$	True

TABLE 4.2
Analysis of particle separation at second location

P_{y_0}	PS_{x_0}	PS_{y_0}	Separation
$\{0, 0.01, \ldots, 1.0\}$	$\{0, 0.1\}$	$\{-1.0, -0.8, \ldots 1.0\}$	True
$\{0, 0.01, \ldots, 1.0\}$	$\{0.2\}$	$\{-1.0, -0.8, \ldots, 0.4\}$	False
$\{0, 0.01, \ldots, 1.0\}$	$\{0.2\}$	$\{0.6, 0.8, 1.0\}$	True
$\{0, 0.01, \ldots, 1.0\}$	$\{0.3, 0.4, \ldots, 0.7\}$	$\{-1.0, -0.8, \ldots, 1.0\}$	True
$\{0, 0.01, \ldots, 1.0\}$	$\{0.8\}$	$\{-1.0, -0.8, \ldots, 0.8\}$	True
$\{0, 0.01, \ldots, 0.2\}$	$\{0.8\}$	$\{-1.0\}$	False
$\{0.21, 0.22, \ldots, 1.0\}$	$\{0.8\}$	$\{-1.0\}$	True
$\{0, 0.01, \ldots, 1.0\}$	$\{0.9, 1.0\}$	$\{-1.0, -0.8, \ldots, 1.0\}$	True

5. Conclusion and Discussion. Existing microparticles separation techniques are used frequently within health domain for cancer cell separation in human blood. These techniques are usually validated through a set of experimental data, and then results are compared to the theoretical model. However, there is a lack of modeling the behavior of these particles within microfluidic device at high level of abstraction, where the movement of cells and particles in microchannels under the influence of DEP is affected by several factors and forces such as inertia, gravity, buoyancy, dielectrophoresis and virtual mass. In this work, we formalize the movement of microparticles under DEP in a microfluidic device using NuSMV model checker in order to

provide formal analysis for the microdevice. We first modeled the system specifications including all types of forces in NuSMV language. This is achieved by modeling the dynamic behavior of the microdevice as a state based system. We then formalized the correct separation of blood cells as a CTL property. Finally, we used NuSMV tool to analyze the behavior of the system for different sets of parameters. The results show that the microdevice will successfully separate large portion of particles, while at the same time, there will be specific scenarios where cells with particular parameters might not be separated successfully.

As a future work, we intend to extend the work using probabilistic model checking which can provide statistical analysis about the behavior of the microdevice. This model can provide early understanding of the behavior of the system at high level of abstraction, and therefore can help to validate several aspects of the design which is beneficial to designers of similar microdevices at early stages of the design process.

REFERENCES

[1] H. Al-Hamadi, A. Gawanmeh, and M. Al-Qutayri. *Theorem proving verification of privacy in wbsn for healthcare systems.* In International Conference on Electronics, Circuits, and Systems (ICECS), pages 100–101. IEEE, 2013.

[2] H. Al-Hamadi, A. Gawanmeh, and M. Al-Qutayri. *Formalizing electrocardiogram (ECG) signal behavior in event-b.* In International Conference on e-Health Networking, Applications and Services (Healthcom), pages 55–60. IEEE, Oct 2014.

[3] H. Al-Hamadi, A. Gawanmeh, and M. Al-Qutayri. *A verification methodology for a wireless body sensor network functionality.* In IEEE-EMBS International Conference on Biomedical and Health Informatics, pages 635–639. IEEE, June 2014.

[4] H. Al-Hamadi, A. Gawanmeh, and M. Al-Qutayri. *Formal validation of QRS wave within ECG.* In IEEE Int. Conf. on Information and Communication Technology Research, pages 190–193, May 2015.

[5] A. Alazzam, B. Mathew, M. Abutayeh, A. Gawanmeh, and S. Khashan. *Modeling the trajectory of microparticles subjected to dielectrophoresis in a microfluidic device for field flow fractionation.* Chemical Engineering Science, 138:266–280, July 2015.

[6] S. M. Babamir and M. Borhani. *Formal Verification of Medical Monitoring Software Using Z Language: A Representative Sample.* Journal of Medical Systems, 36(4):2633–2648, 2012.

[7] C. Baier and J.P. Katoen. *Principles of Model Checking.* The MIT Press, Cambridge, MA, USA, 2008.

[8] K. Benghazi, M. V. Hurtado, M. L. Rodrguez, and M. Noguera. *Applying Formal Verification Techniques to Ambient Assisted Living Systems.* volume 5872 of *LNCS*, pages 381–390. Springer, 2009.

[9] C. Bertolini, Z. Liu, M. Schaf, and V.Stolz. *Towards a Formal Integrated Model of Collaborative Healthcare Workflows.* In Foundations of Health Informatics Engineering and Systems, volume 7151 of *LNCS*, pages 57–74. Springer, 2012.

[10] P. Boca, J. Bowen, and J. Siddiqi. *Formal methods: state of the art and new directions.* Springer, 2010.

[11] J. Cao, P. Cheng, and F. Hong. *A numerical analysis of forces imposed on particles in conventional dielectrophoresis in microchannels with interdigitated electrodes.* Journal of Electrostatics, 66(11):620–626, 2008.

[12] O. Choudhury, N. Hazekamp, D. Thain, and S. Emrich. *Accelerating comparative genomics workflows in a distributed environment with optimized data partitioning and workflow fusion.* Scalable Computing: Practice and Experience, 16(1):53–70, 2015.

[13] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. *Nusmv 2: An Opensource tool for Symbolic Model Checking.* In Computer Aided Verification, pages 359–364. Springer, 2002.

[14] N. Crews, J. Darabi, P. Voglewede, F. Guo, and A. Bayoumi. *An analysis of interdigitated electrode geometry for dielectrophoretic particle transport in micro-fluidics.* Sensors and Actuators B: Chemical, 125(2):672–679, 2007.

[15] P. Curzon, H. Thimbleby, and P. Masci. *Early identification of software causes of use-related hazards in medical devices.* In Proceedings of the 5th EAI International Conference on Wireless Mobile Communication and Healthcare, pages 12–14. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2015.

[16] Z. Daw, R. Cleaveland, and M. Vetter. *Formal Verification of Software-Based Medical Devices considering Medical Guide-lines.* Computer Assisted Radiology and Surgery, 9(1):145–53, 2014.

[17] A. Gawanmeh. *An axiomatic model for formal specification requirements of ubiquitous healthcare systems.* In *Consumer Communications and Networking Conference (CCNC)*, pages 898–902. IEEE, 2013.

[18] A. Gawanmeh, H. Al-Hamadi, M. Al-Qutayri, S.K. Chin, and K. Saleem. *Reliability analysis of healthcare information systems: State of the art and future directions.* In IEEE International Conference on e-Health Networking, Applications and Services, pages 68–74. IEEE, 2015.

[19] A. Gawanmeh, A. Alazzam, B. Mathew, M. Abutayeh, and H. Jin Sung. *Formalizing the movement of microparticles in a continuous flow microfluidic device for field flow fractionation.* In IEEE International Conference on e-Health Networking, Applications and Services, Healthcom, pages 114–119. IEEE, 2015.

[20] O. Hasan and S. Tahar. *Formal verification methods.* Encyclopedia of Information Science and Technology, IGI Global, pages 7162–7170, 2015.

[21] M. Hoogendoorn, M. C. Klein, Z. A. Memon, and J. Treur. *Formal Verification of an Agent-Based Support System for Medicine Intake.* 25:453–466, 2009.

[22] Y. Huang, X.B. Wang, F. Becker, and P.R. Gascoyne. *Introducing dielectrophoresis as a new force field for field-flow*

fractionation. Biophysical Journal, 73(2):1118–1129, 1997.
[23] K. Khoshmanesh, S. Nahavandi, S. Baratchi, A. Mitchell, and K. Kalantar-zadeh. *Dielectrophoretic platforms for bio-microfluidic systems.* Biosensors and Bioelectronics, 26(5):1800–1814, 2011.
[24] J. G. Kralj, M. Lis, M. Schmidt, and K. Jensen. *Continuous dielectrophoretic size-based particle sorting.* Analytical chemistry, 78(14):5019–5025, 2006.
[25] Y. C. Lam, S. H. Ling, W. Y. Chan, and K. S. Chian. *Dielectrophoretic cell motion model over periodic microelectrodes with unit-cell approach.* Microfluidics and Nanofluidics, pages 1–13, 2014.
[26] T.S. Leu and C.Y. Weng. *Studies of particle levitation in a dielectrophoretic field-flow fraction–based microsorter.* Journal of Micro/Nanolithography, MEMS, and MOEMS, 8(2):021106–021106, 2009.
[27] E. Loth. *Numerical approaches for motion of dispersed particles, droplets and bubbles.* Progress in Energy and Combustion Science, 26(3):161–223, 2000.
[28] E. Marzini, P. Mori, S. Di Bona, D. Guerri, M. Lettere, and L. Ricci. *A tool for managing the x1. v1 platform on the cloud.* Scalable Computing: Practice and Experience, 16(1):103–120, 2015.
[29] P. Masci, A. Ayoub, P. Curzon, M. Harrison, I. Lee, and H. Thimbleby. *Verification of interactive software for medical devices: PCA infusion pumps and FDA regulation as an example.* In Proceedings of the ACM Symposium on Engineering Interactive Computing Systems, pages 81–90. ACM, 2013.
[30] A. Neculae, C. Biris, M. Bunoiu, and M. Lungu. *Numerical analysis of nanoparticle behavior in a microfluidic channel under dielectrophoresis.* Journal of Nanoparticle Research, 14(10):1–12, 2012.
[31] U. Pervez, O. Hasan, K. Latif, S. Tahar, A. Gawanmeh, and M. S. Hamdi. *Formal reliability analysis of a typical fhir standard based e-health system using prism.* In International Conference on e-Health Networking, Applications and Services (Healthcom), pages 43–48. IEEE, 2014.
[32] U. Pervez, A. Mahmood, O. Hasan, K. Latif, and A. Gawanmeh. *Formal reliability analysis of device interoperability middleware (DIM) based E-Health system using PRISM.* In International Conference on E-health Networking, Application Services (HealthCom), pages 108–113. IEEE, Oct 2015.
[33] C. Zhang, K. Khoshmanesh, A. Mitchell, and K. Kalantar-Zadeh. *Dielectrophoresis for manipulation of micro/nano particles in microfluidic systems.* Analytical and bioanalytical chemistry, 396(1):401–420, 2010.

Edited by: Kashif Saleem
Received: Feb 29, 2016
Accepted: May 15, 2016

DOI 10.12694/scpe.v17i3.1183
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 17, Number 3, pp. 237–249. http://www.scpe.org

SOLVING THE TABLE MAKER'S DILEMMA ON CURRENT SIMD ARCHITECTURES

CHRISTOPHE AVENEL[†§] PIERRE FORTIN[†] MOURAD GOUICEM[†‡] AND SAMIA ZAIDI[†]

Abstract. Correctly-rounded implementations of some elementary functions are recommended by the IEEE 754-2008 standard, which aims at ensuring portable and predictable floating-point computations. Such implementations require the solving of the Table Maker's Dilemma which implies a huge amount of computation time. These computations are embarrassingly and massively parallel, but present control flow divergence which limits performance at the SIMD parallelism level, whose share in the overall performance of current and forthcoming HPC architectures is increasing. In this paper, we show that efficiently solving the Table Maker's Dilemma on various multi-core and many-core SIMD architectures (CPUs, GPUs, Intel Xeon Phi) requires to jointly handle divergence at the algorithmic, programming and hardware levels in order to scale with the number of SIMD lanes. Depending on the architecture, the performance gains can reach 10.5x over divergent code, or be constrained by different limits that we detail.

Key words: floating-point arithmetic, Table Maker's Dilemma, SIMD, control flow divergence, OpenCL

AMS subject classifications. 68M07, 68W10

1. Introduction. Since 1985, the IEEE 754 standard specifies the implementation of floating-point operations in order to have portable and predictable numerical software. Its latest revision [15, 4] recommends the correct rounding of some elementary functions, like log, exp and the trigonometric functions. Since such functions are transcendental, one cannot evaluate them exactly but have to approximate their evaluation. However, it is hard to decide which intermediate precision is required in the function implementation to guarantee a correctly rounded result: the rounded evaluation of the approximation must be equal to the rounded evaluation of the function with infinite precision. This problem is known as the *Table Maker's Dilemma* or TMD (see [25], chapter 12: Solving the Table Maker's Dilemma).

Solving the TMD involves finding the *hardest-to-round* arguments of the function [25], that is to say the arguments requiring the highest precision to be correctly rounded when the function is evaluated at. This precision guaranteeing the correct rounding for all arguments is named the *hardness-to-round* of the function [25]. The hardest-to-round cases can be found by *exhaustive search*, which implies to browse each floating-point number in the domain of definition of the function. This approach is however prohibitive since it leads to a $O(2^p)$ operation count when considering precision-p floating-point numbers as arguments.

In order to speed up the search for hardest-to-round arguments, the Lefèvre algorithm [21] uses local affine approximations of the targeted function. The domain of definition of the function is split into several domains D_i and an affine approximation of the function is computed for each D_i. Thanks to this affine approximation, one can isolate *hard-to-round cases* (HR-cases, see [25, 23]) with a $O(p^2)$ operation count for a domain D_i with precision-p floating-point numbers. The hardest-to-round cases are then found among the HR-cases with a localized exhaustive search. Higher degree approximations have been introduced since (SLZ algorithm [29]) in order to further reduce the asymptotic operation count for large values of p. However quadruple precision ($p = 113$) is still currently out of reach. We thus focus in this article on the double precision format ($p = 53$), for which the Lefèvre algorithm is as efficient as the SLZ algorithm in practice [25, 6]. Moreover, the Lefèvre algorithm has already been used to generate all known hardness-to-round in double precision [25], and it offers fine-grained parallelism which is suitable for massively parallel architectures like GPUs [8, 10]. We therefore study the Lefèvre algorithm here.

Even if the Lefèvre algorithm makes it possible to compute the hardness-to-round of elementary functions, it remains very computationally intensive. For example, it requires around five years of CPU time for the exponential function over all double precision arguments. Moreover, even if the hardest-to-round cases of some functions in double precision are known [25], this is still not the case for about half of the univariate functions

[†]Sorbonne Universités, UPMC Univ Paris 06, CNRS, UMR 7606, LIP6, F-75005, Paris, France
[‡]LIRMM, CNRS/Université Montpellier 2, UMR 5506, Montpellier, France
[§]Centre for Image Analysis, Uppsala University

recommended by the IEEE standard 754-2008. Furthermore, some scientific computations may require correctly-rounded implementations of other elementary functions, of specific compositions of elementary functions or even of elementary functions using non-standard formats or precisions. Being able to find the hardness-to-round of any elementary function in double precision in a reasonable amount of time would therefore be very useful.

In practice, both the affine approximation generation and the HR-case search are independent among the D_i domains. This data-parallel algorithm is thus embarrassingly and massively parallel which suits well to multi-core and many-core parallel architectures. However, such architectures rely heavily on SIMD (Single Instruction Multiple Data) parallelism. As far as CPUs are concerned, such parallelism is increasingly important in the overall CPU performance since the SIMD vector width has been constantly increasing from 64 bits (MMX [16] and 3DNow! [1]) to 128 bits (SSE [17], AltiVec [5]), then to 256 bits (AVX [18]), and to 512 bits on the Intel Xeon Phi as well as in the forthcoming AVX-512 instruction set [19]. As far as GPUs are concerned, they are now widely used in HPC and also present a partial SIMD execution since multiple GPU threads are processed in a SIMD fashion by groups of 32 or 64 threads. This increasing SIMD parallelism offers indeed important performance gains at a relatively low hardware cost. But efficiently exploiting such parallelism requires "regular" algorithms where the memory accesses and the computations are similar among the lanes of the SIMD vector unit. Unfortunately, as presented in [8], the original HR-case search of Lefèvre algorithm (*Lefèvre HR-case search*) presents multiple sources of control flow divergence which limit its performance on GPUs.

In this paper, we show that efficiently solving the TMD on various multi-core and many-core SIMD architectures (CPUs, GPUs, Intel Xeon Phi), and scaling performance with the number of SIMD lanes, requires to jointly handle this divergence at multiple levels: algorithm, programming and hardware. We start by describing a regular HR-case search algorithm, first presented in [9, 10], which has been shown to drastically reduce divergence in the execution flow on NVIDIA GPUs [9, 10]. We then extend the deployment of this HR-case search on other SIMD architectures: CPUs and the Intel Xeon Phi. We first compare C programming with the SPMD-on-SIMD (*Single Program Multiple Data*) programming model [26, 7], and show that the SPMD-on-SIMD model is well-suited for vectorizing the HR-case search on CPUs and on the Xeon Phi. Moreover, thanks to OpenCL this programming model enables code portability on various architectures, including AMD GPUs. Secondly, we present a survey of the deployment of this OpenCL implementation on various current SIMD architectures. We detail the performance results depending on how divergence is handled at the hardware level, and on the discrepancy between control flow (static) divergence and execution flow (dynamic) divergence. We obtain performance gains up to 10.5x on some architectures, and specify the performance bottlenecks on the other architectures. Finally, we present a performance comparison of these architectures for solving the TMD.

As far as related work is concerned, general solutions have been proposed to handle divergence on SIMD architectures, at the hardware level [3, 12, 24] as well as at the software level [11, 13, 31, 28]. We target here currently available hardware, and our HR-case searches offer very fine computation grains: the overhead of software solutions to handle divergence would be too high here (see [8] for example). Up to our knowledge, there is no other specific work to reduce the SIMD divergence when solving the TMD. The reference C code of V. Lefèvre [21] is a CPU scalar code that can target multi-core and distributed multi-processor architectures, but does not exploit SIMD parallelism within each CPU core. It can be noticed that another implementation to solve the TMD has been designed for FPGA architectures [6], but this implementation relies on the exhaustive search.

In the rest of this paper, Sect. 2 introduces the Table Maker's Dilemma and Lefèvre algorithm. In Sect. 3 we present our regular algorithm for the HR-case search which reduces divergence in the execution flow. In Sect. 4 we compare two programming models to enable the CPU vectorization of the HR-case search code, and show the relevance of the SPMD-on-SIMD programming model. Section 5 presents performance results, detailed analyzes on various SIMD architectures, and the performance comparison among these architectures. Finally, concluding remarks will be presented in Sect. 6.

2. The Table Maker's Dilemma and Lefèvre algorithm. Floating-point arithmetic aims at approximating real arithmetic. It uses the property that any real number α can be uniquely represented in base β scientific notation as $\alpha = m \times \beta^e$, with $1 \leq m < \beta$ the *significand* of α and $e \in \mathbb{N}$ the *exponent* of α. A precision-p floating-point (abbreviated FP_p) number format will then represent real numbers with e in a specific range and m with a finite precision of p digits. Hence every real number α is either exactly representable as an

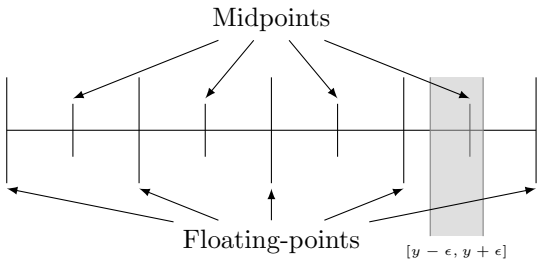


FIG. 2.1. *Example of undetermined correct rounding for a value y computed with precision ϵ in the case of rounding to nearest, where the rounding breakpoints are the midpoints of floating-point numbers.*

FP_p number, or is not, in which case it will be approximated using a *rounding function*.

When evaluating a function at a precision-p floating-point argument, the exact result is commonly not a precision-p floating-point number. The simple example of the inverse function $1/x$ evaluated at x different than a power of 2 (e.g. $x = 10$) yields an infinite sequence of digits in the fractional part of the result if represented in binary format. Hence, a typical implementation of a mathematical function will have to approximate the exact mathematical result $f(x)$ by $\hat{f}(x)$ with precision ϵ, and then round this approximation to p bits of precision. However, if for some argument x, $\hat{f}(x)$ is at a distance less than ϵ to a rounding breakpoint (where the result of the rounding function changes), it is impossible to determine the correct rounding of $f(x)$ from $\hat{f}(x)$ as illustrated in Fig. 2.1. Such an argument x is called a (p, ϵ) *hard-to-round* case (abbreviated as HR-case).

Given a function f, a rounding function, and a FP_p format, the *Table Maker's Dilemma* is hence defined as finding the necessary accuracy ε such that both $f(x)$ and an approximation $\hat{f}(x)$ with accuracy ε round to the same FP_p number for every FP_p number x in the definition domain of the function f. The largest ε verifying this property is called the *hardness-to-round* of f at precision p.

To find this hardness-to-round with a better complexity than exhaustive search, Lefèvre algorithm relies on a three step methodology based on searching (p, ϵ) HR-cases [21]:

- fix a "convenient" ϵ using probabilistic assumptions [25],
- find (p, ϵ) HR-cases with *ad hoc* methods,
- find the hardest-to-round case among the (p, ϵ) hard-to-round cases.

There are two key-points in that methodology. First, the statistical assumption states that the probability of an argument being an HR-case decreases exponentially with the precision of the approximation. This implies that the hardness-to-round of an elementary function evaluated to a precision-p is likely to be around 2^{-2p} [25], and that there are few $(p, 2^{-2p})$ HR-cases. This can dramatically reduce the use of exhaustive search, which is time consuming. But more importantly, the second key-point is that we can search for HR-cases in polynomial time in the format size, against an exponential time for the exhaustive search. This efficient HR-case search is obtained by using polynomials and the following two steps.

- *The generation of polynomial approximations*: we generate many local polynomial approximations P_i of the function f over independent domains D_i, with error $\epsilon_{approx} \approx \epsilon$.
- *The (p, ϵ') HR-case search*: for each polynomial approximation P_i, we search for (p, ϵ') HR-cases of P_i, which are the (p, ϵ) HR-cases of f, with $\epsilon' = \epsilon + \epsilon_{approx}$.

These two steps are massively parallel over the domains D_i since these numerous domains can all be processed independently. However, while the polynomial approximation generation has a regular control flow [9], the HR-case search presents divergence issues when executed on SIMD architectures. Moreover, the HR-case search is the most time consuming step when solving the TMD. In the rest of this paper, we will therefore focus on this HR-case search and its SIMD execution on various HPC architectures.

The HR-case search on the polynomials P_i is done using an isolation strategy [22] as described in Algorithm 1. An HR-case existence test `Exists?` is executed on each domain D_i. It returns `True` if there potentially exists an HR-case in the tested domain (false positives are possible), or `False` otherwise. If the test succeeds (that is to say, there might be an HR-case in the tested domain D_i), we split D_i into κ sub-domains $D_{i,j}$, upon which

we repeat the HR-case existence test. For each of these sub-domains succeeding the HR-case existence test, we finally perform exhaustive search. In [22], Lefèvre tested other variants of this strategy and concluded that in practice the most efficient strategy was this three phases refinement. This is mainly due to the fact that the number of arguments succeeding the existence test can be roughly predicted, and that the amount of time spent in the HR-case existence test has to be balanced with the amount of time spent in exhaustive search [9, 22].

Algorithm 1: Lefèvre three phases isolation strategy for HR-case search.

```
1  foreach P_i over its domain D_i do
2      if Exists?(P_i, ε') then                                   /* Phase 1 */
3          (D_{i,1}, D_{i,2}, ..., D_{i,κ}) := SplitDomain(D_i, κ);
4          (P_{i,1}, P_{i,2}, ..., P_{i,κ}) := RefineApprox(P_i, κ);
5          foreach P_{i,j} over its domain D_{i,j} do
6              if Exist?(P_{i,j}, ε') then                        /* Phase 2 */
7                  ExhaustiveSearch(P_{i,j}, ε') ;                /* Phase 3 */
8              end
9          end
10     end
11 end
```

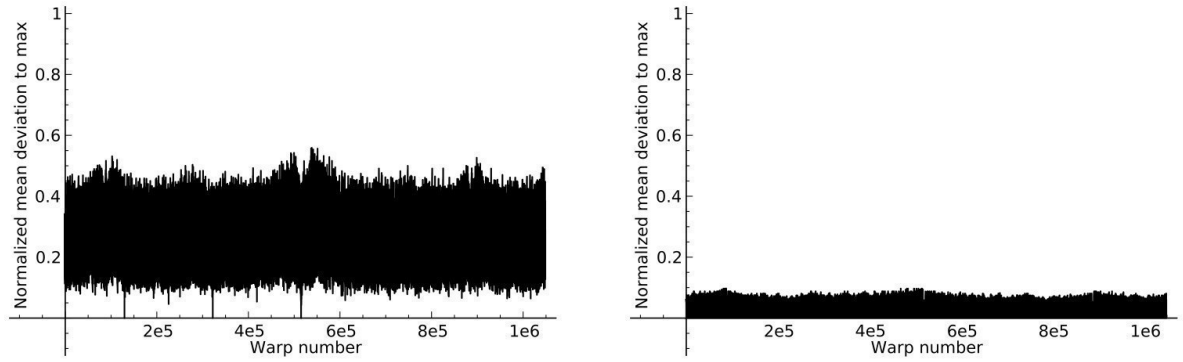
Lefèvre HR-case existence test takes as argument a degree one polynomial P_i or $P_{i,j}$. As we do not need the dynamic range of floating-point numbers, we use fixed-point arithmetic to avoid rounding errors, and we apply a suitable change of variable to write P_i or $P_{i,j}$ as a polynomial $b - a \cdot x$, while representing only the 64 bits after the p^{th} bit of the significands of a and b as 64-bit integers. Hence we also consider $x \in \mathbb{N}$. This HR-case existence test then returns a lower bound on the distance between the values of $b - a \cdot x$ for $x < N$ and the rounding breakpoints, with N the number of arguments to test in D_i or $D_{i,j}$. This is achieved by computing the continued fraction expansion of a with the Euclidean algorithm, and a particular decomposition of b in the sequence of partial remainders. Comparing this lower bound to ϵ', we can then determine whether there is potentially a (p, ϵ') HR-case in the domain or not. Lefèvre existence test is presented in Algorithm 2 and is explained more thoroughly in [9, 10].

Algorithm 2: Lefèvre HR-case existence test algorithm.

```
   input : b − a · x, ε', N
1  initialisation: p ← {a};  q ← 1 − {a};  d ← {b};
                   u ← 1;    v ← 1;
2  if d < ε' then return True;
3  while True do
4      if d < p then
5          k = ⌊q/p⌋;
6          q ← q − k ∗ p; u ← u + k ∗ v;
7          if u + v ≥ N then return False;
8          p ← p − q; v ← v + u;
9      else
10         d ← d − p;
11         if d < ε' then return True;
12         k = ⌊p/q⌋;
13         p ← p − k ∗ q; v ← v + k ∗ u;
14         if u + v ≥ N then return False;
15         q ← q − p; u ← u + v;
16     end
17 end
```

3. A regular algorithm for the HR-case search. In [8], we underlined a problem in Lefèvre algorithm execution on GPU architectures: the execution flow is highly divergent from one thread to another. There are three sources of divergence in Algorithm 2:

- the main unconditional loop, whose number of iterations depends on the value of the arguments;



(a) Lefèvre HR-case search (b) Regular HR-case search

FIG. 3.1. *Normalized mean deviation to the maximum of the number of main loop iterations per CUDA warp, on NVIDIA GPUs, among the* 2^{20} *CUDA warps required for the* exp *function in the domain* $[1; 1 + 2^{-13}]$.

- the main conditional statement, whose scope contains all the intructions within the main loop;
- and finally the divisions, which are computed using a hybrid implementation with a tunable parameter `LOGMS`. If we compute p/q and $p > 2^{\texttt{LOGMS}}q$, we call the division instruction, otherwise it is more efficient to use a loop to compute the quotient by repeated subtractions. As divisions operands are 64-bit integers, `LOGMS` $= 64$ implies all quotients are computed using repeated subtractions, and `LOGMS` $= 0$ implies all quotients are computed using the division intruction.

Even though the hybrid divisions affect the control flow, they do not lead to strong divergence issue at runtime since almost all the computed quotients are expected to be small in practice [9]. However when processing multiple instances of the Lefèvre HR-case existence test in parallel on GPUs, the main conditional statement and the main loop have a strong performance impact because of the partial SIMD execution of GPUs. Both are induced by conditioning the computation of the quotients of the continued fraction of a by the value of b. To our knowledge there is no *a priori* information on the number of loop iterations or on the branch executed at each iteration that would enable us to statically reorder the domains D_i in order to decrease this divergence. We also tried to use software solutions to reduce the impact of the loop divergence [8]: no performance gain was obtained because the computation is very fine-grained.

To highlight the impact of loop divergence during Lefèvre existence test execution, we introduced in [8] an indicator named the *normalized mean deviation to the maximum*. When processing concurrently n independent instances of a divergent loop on a SIMD unit with n lanes, the number of loop iterations issued in total is the maximum number of loop iterations issued among all the lanes of the SIMD unit. This indicator aims thus at giving the average percentage of loop iterations for which a lane remains idle during the SIMD execution. More formally, we denote ℓ_i the number of loop iterations to issue for the lane i and we number the lanes within a SIMD vector from 1 to n. If $\ell = \{\ell_i, i \in [\![1, n]\!]\}$, the Normalized Mean Deviation to the Maximum (NMDM) is defined as

$$\mathrm{NMDM}(\ell) = 1 - \frac{\mathrm{mean}(\ell)}{\max(\ell)}.$$

In Fig. 3.1(a), we measured the NMDM of the main unconditional loop of Lefèvre HR-case search execution on a NVIDIA GPU ($n = 32$) on a set of domains D_i for the exponential function. We can see that the NMDM is uniformly high with an average NMDM of 25.6%, which means that a SIMD lane remains idle on average 25.6% of the number of loop iterations issued on its SIMD unit. This divergence in Lefèvre HR-case search is mainly due to the fact that the algorithm goes from the subtraction-based Euclidean algorithm to the division-based one depending on the value of b.

In [9, 10], we proposed a new HR-case existence test which presents a regular execution, as illustrated in Algorithm 3. This is enabled by getting rid of the dependence between the computation of the continued

Algorithm 3: Regular HR-case existence test algorithm.

```
   input  : b − ax, ϵ′, N
1  initialisation:  p ← {a};   q ← 1;   d ← {b};
                    u ← 1;     v ← 0;
2  while True do
3  |  k = ⌊q/p⌋;
4  |  q = q − k ∗ p; u = u + k ∗ v;
5  |  d = d  mod p;
6  |  if u + v ≥ N then return d > ϵ′;
7  |  k = ⌊p/q⌋;
8  |  p = p − k ∗ q; v = v + k ∗ u;
9  |  if d ≥ p then
10 |  |  d = d − p  mod q;
11 |  end
12 |  if u + v ≥ N then return d > ϵ′;
13 end
```

fraction expansion of a and the value of b. It first turns the unpredictable main conditional statement of Lefèvre algorithm into a deterministic test, which can be removed by unrolling two loop iterations as in Algorithm 3. And second, a full quotient of the Euclidean algorithm is entirely computed at each loop iteration in the regular HR-case search, which is not the case in the Lefèvre existence test. As the number of quotients to compute is almost constant from one domain D_i to the next, we reduce the mean NMDM per SIMD computation on NVIDIA GPUs from 25.6% to 0.1% (cf. Fig. 3.1(b)). However, even though the execution flow is now regular, the control flow in the source code remains divergent: the algorithm still exhibits the main unconditional loop, a few conditional statements, inner loops for the hybrid implementation of the divisions, and outer loops (over the D_i or $D_{i,j}$ domains as presented in Algorithm 1).

In practice, such regular existence test offers performance gains on NVIDIA GPUs up to 3.4x over Lefèvre existence test [9, 10]. When comparing an high-end hex-core CPU with an high-end NVIDIA GPU, the GPU deployment delivers a 6.6x speedup for the regular existence test. Such speedup is mainly due to the lack of SIMD computations on the CPU. That is why we will now consider the vectorization of the two existence tests on CPUs and on Xeon Phi.

For the sake of shortness, we will name in the remainder of this paper *Lefèvre HR-case search* the combination of the isolation algorithm with Lefèvre existence test, and *regular HR-case search* the combination of the isolation algorithm with the regular existence test.

4. The relevant programming paradigm. In order to deploy the HR-case search on CPUs and on the Xeon Phi, one could first consider to rely on C programming. We thus start by considering the vectorization of the reference scalar C code implementing the Lefèvre and regular HR-case searches [9, 10].

4.1. Vectorization with C compiler. When considering the vectorization of a C program, one can use several programming paradigms.

Manual SIMD programming with intrinsics is a first possibility. However, this is generally a tedious task which requires for example array padding and which leads to non-portable code: the program must be re-written when moving to another SIMD instruction set or to another vector width. In the case of the HR-case search, this would be an especially tedious task because of the multiple nested `while` loops and conditional branches. Each one is a divergence source which implies a different mask to handle this divergence on CPU SIMD units. Therefore, with intrinsics we would have to set and update all these masks in the source code which represents a very important programming effort.

Another possibility is to rely on the C compiler. Automatic vectorization is provided for example in `icc` (Intel C/C++ Compiler) and `gcc` (GNU C Compiler). The programmer can let the compiler perform the dependency analysis of the targeted loop, in order to determine whether the loop is parallel or not, hence vectorizable or not. This dependency analysis is however limited by the compiler capacity [20]. Therefore, some compilers support compiler directives, which enable the programmer to indicate (and ensure) that the loop is parallel: no dependency analysis is then required by the compiler. Such compiler directives are available in `icc`

(`#pragma simd`), and have recently been standardized in the last versions of OpenMP (OpenMP 4.X).

As far as the HR-case search is concerned, we aim at vectorizing multiple iterations of the outermost loop which browses 2^{25} D_i domains. We use here `icc` (version 15.0.2).

4.1.1. Exhaustive search. In order to start with a simpler code, we first consider only the exhaustive search (phase 3): phases 1 and 2 are here removed from the code. In the reference C code, the original implementation used to rely on two inner `do..while` loop for each D_i domain: the 2^{15} arguments of the D_i domain being browsed as 8 sub-domains of 2^{12} arguments (to match phase 2). In order to minimize the number of nested loops, and to ease the compiler vectorization, these two `do..while` loops have been merged in one single `for` loop.

When considering automatic vectorization of a loop nest without compiler directive, the compiler starts with the innermost loop [20], which corresponds here to this new inner `for` loop within the outer `for` loop over the D_i domains. However, as the exhaustive search is performed using the tabulated differences algorithm [21], this inner `for` loop presents `flow` and `anti` data dependencies among its iterations: the next polynomial evaluation is computed from the current one. This used to prevent former versions of `icc` from vectorizing the inner loop and hence the outer loop. The latest version of `icc` (15.0.2) can override this vector dependency on the inner loop, and attempt to vectorize the outer loop. But, this outer loop vectorization fails due to an `output` dependency. Indeed, the 2^{25} D_i domains provided as input lead in practice to very few HR-cases (e.g. 243 for the first set of 2^{25} D_i domains). These HR-cases are written consecutively in memory thanks to a counter incremented each time an HR-case is found, which results in an `output` data dependency between successive iterations.

When considering vectorization hinted by compiler directives (here `#pragma simd` on the outer loop), either the compiler does generate vector code, which leads to wrong results because of this `output` data dependency, or the compiler detects the dependency and refuses the vectorization.

4.1.2. Complete HR-case search. We now consider the complete HR-case search with the three phases and study the regular HR-case search. In order to enable the vectorization [20], we had to strongly rewrite our C code. Function calls from the loop bodies have first been replaced by preprocessor macros. The corresponding `return` statements have been replaced by boolean tests: this ensures one single entry and one single exit in each loop [20]. Likewise, `goto` statements among the different phases have been replaced by boolean tests.

When considering the complete HR-case search, the compiler faces the same data dependencies as with the exhaustive search only. Moreover, there are in phases 1 and 2 inner `while` loops with unknown iteration numbers, which cannot be vectorized and can thus prevent the outer loop vectorization. All this leads to the same conclusion: the outer loop vectorization fails without compiler directives. When forcing vectorization with compiler directives, the compiler can manage to vectorize the code but this results again in wrong results because of the `output` dependency.

It has to be noticed that the same conclusions would also apply to the Lefèvre HR-case search which presents the same data dependencies and also outputs its HR-cases consecutively.

4.2. Implicit vectorization in OpenCL. Another possibility to exploit the SIMD units is to rely on the SPMD-on-SIMD (*Single Program Multiple Data*) programming model [26, 7]. All computations are written as scalar ones and it is up to the compiler to merge such scalar computations in SIMD instructions. The main advantages are the ease of programming and the portability: the programmer needs neither to write the specific SIMD intrinsics for each architecture, nor to know the vector width, nor to implement data padding with zeros according to this vector width. The vector width will indeed be determined only at compile time (depending on the targeted hardware). Moreover, like compiler directives, no data dependency analysis is required by the compiler: it is up to the user to ensure that the scalar computations can be processed correctly in parallel. Such programming paradigm is increasingly used in HPC: first on GPUs with CUDA and then on various compute devices with OpenCL. On CPU, such programming model is available in OpenCL (OpenCL implicit vectorization), as well as in the Intel SPMD Program Compiler (`ispc`) [26]. We choose here OpenCL over `ispc` since OpenCL enables us to maintain one single source code for both CPUs and GPUs, and to target other GPUs like the AMD ones. On multi-core CPUs, we use the Intel OpenCL SDK[1] which provides OpenCL

[1] See: `https://software.intel.com/en-us/intel-opencl`

TABLE 5.1
Times in seconds for both HR-case searches over I_0 on one NVIDIA C2070 GPU.

HR-case search	Lefèvre		Regular	
	CUDA	OpenCL	CUDA	OpenCL
Phase 1	0.258	0.246	0.074	0.069
Phase 2	0.006	0.006	0.010	0.008
Phase 3	0.001	0.001	0.003	0.003
Total	0.265	0.253	0.086	0.081

implicit vectorization while supporting conditional statements as well as `while` loops in the OpenCL kernels [27]. It has to be noticed that OpenCL also provides parallelism at the thread level in order to exploit multi-core processors in shared memory. This is however not a key-point here since such parallelism is straightforward to implement in the HR-case search [10].

Our OpenCL kernels are thus a translation of our CUDA kernels [8, 9]. The OpenCL implementation therefore uses the same code structure as in the CUDA implementation [8] where the three phases of the HR-case search have been separated in three distinct GPU kernels. Like in CUDA, atomic operations are used in OpenCL to consecutively write the outputs of each phase in memory. This includes the HR-cases resulting from phase 3 and leading to the `output` dependency with the C compiler vectorization. Here this issue is easily solved thanks to the SPMD-on-SIMD programming model, and these atomic operations are the only synchronizations required among the work-items. We thus emphasize that the HR-case search of the Table Maker's Dilemma fits naturally with the SPMD-on-SIMD programming model: each work-item processes one (or a few) D_i domain(s), and only a few atomic operations are required for correct work-item synchronization. We then fully exploit the data parallelism of this massively parallel application to process concurrently the numerous work-items on the SIMD units (as well as on all the available CPU cores).

However, as far as performance is concerned, the divergence in the SIMD processing of consecutive D_i domains will be a key-factor and will impact performance differently depending on the algorithm (Lefèvre or regular HR-case search) and on the underlying hardware, as detailed in the next section.

5. Performance results of HR-case searches on various current SIMD architectures. For the following performance results, each OpenCL implementation is tuned in order to determine, for each OpenCL kernel, the optimal value for the work group size, for the number of intervals processed by each work-item, and for the LOGMS value (for phases 1 and 2, cf. Sects. 2 and 3).

All the results given in this section are issued from the HR-case search on the *exp* function for double precision. Except otherwise mentioned, all tests are performed over the 1024 first intervals $I_{0..1023} = [1; 1+2^{-3}[$ of the binade $[1; 2[$ ($I_{0..1023}$ containing 2^{50} *doubles*). The parameter tuning has been performed only on the interval $I_0 = [1; 1+2^{-13}]$ (containing 2^{40} *doubles*).

5.1. GPUs and SIMD width impact. We first present performance results of our OpenCL implementation on both NVIDIA and AMD GPU architectures. Table 5.1 shows that for both HR-case searches the performance of our OpenCL implementation matches, and even slightly outperforms, the one of our original CUDA code on NVIDIA GPUs. This validates the choice to move from CUDA to OpenCL even on NVIDIA GPUs.

We now consider in Fig. 5.1 performance results on various high-end GPUs. On one NVIDIA C2070, the regular HR-case search delivers a 2.7x performance gain over the Lefèvre HR-case search thanks to its regular execution flow. On a newer NVIDIA GPU (K20c),the two HR-case searches are 2.2 or 2.3 times faster compared to their execution on the C2070, which shows that our HR-case search GPU implementation scales well on the newer Kepler GPU architectures which offers a much higher number of GPU cores. Besides, the performance gain of the regular HR-case search over the Lefèvre HR-case search is almost the same. The SIMD width is indeed the same in both Fermi (C2050) and Kepler (K20c) architectures: work-items are processed in a SIMD fashion by groups of 32.

Starting from the Southern Islands family (which includes our Radeon HD 7970), AMD GPUs present a scalar architecture (*Graphics Core Next* - GCN) that enables the programmer to reach best performance with

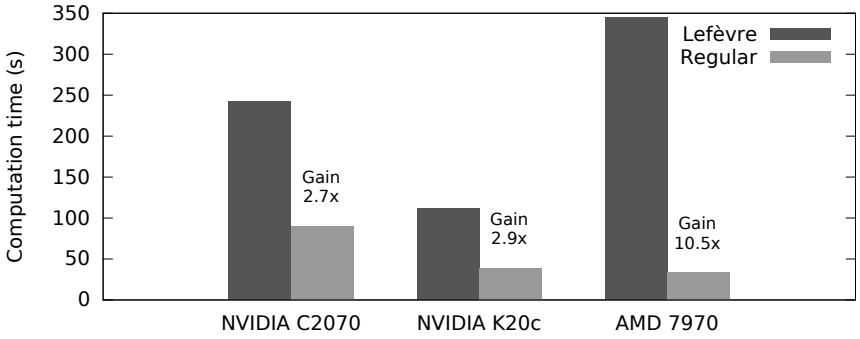


FIG. 5.1. *HR-case searches computation times over $I_{0..1023}$ on various GPUs, using OpenCL from CUDA 7.5 (352.68 driver) and from the AMD SDK 2.9.1 (Catalyst Omega 14.12 driver).The performance gains on top of the bars correspond to the Lefèvre over regular ratios.*

scalar work-items. Contrary to previous AMD GPU generations, no explicit vector programming is required. On these scalar GPUs, the ALUs are arranged in four SIMD arrays consisting of 16 processing elements each [2] (using OpenCL terminology), and work-items are processed in a SIMD fashion by groups of 64. Dynamic divergence among work-items can therefore have an even more detrimental impact on AMD GPUs than on NVIDIA GPUs. This explains that the gain of the regular HR-case search over the Lefèvre's one is more important on AMD Radeon HD 7970 GPU than on NVIDIA GPUs. Comparing this AMD 7970 and the NVIDIA K20c, whose hardware compute powers are similar[2], one can see that only thanks to the regular HR-case search similar application performance can be achieved on these two GPUs.

5.2. AVX and SSE CPUs. We now consider the OpenCL deployment of the two HR-case searches on standard CPUs with either SSE4.2 or AVX2 SIMD instruction sets. SSE4.2 vector units can process two 64-bit integers in a SIMD fashion, whereas AVX2 ones can process four 64-bit integers. There are however multiple issues when considering the deployment of the HR-case searches on such SIMD instruction sets.

The first issue lies at the hardware level, where divergence among SIMD lanes is handled differently on CPU and on GPU [14]. When the control flow diverges on a GPU SIMD unit, a mask register is set according to the condition evaluation: each processing element then either performs the following instruction or remains idle. A stack of mask registers is used to handle nested divergence levels. This is handled dynamically by the GPU hardware, which can then skip at runtime branches where all processing elements would be idle (e.g. for a `if-then-else` statement: when all mask bits are zero the `then` branch can be skipped, and when all mask bits are one the `else` branch can be skipped). When control flows diverge within a CPU SIMD unit, mask registers are also used to handle divergence among the SIMD lanes. On AVX and SSE however, all computations are always performed by all the SIMD lanes. The masks are used to prevent committing results in memory for computations that should not have been performed (*predication*). Moreover, on CPUs all this is handled explicitly in software by the compiler. This implies a general overhead compared to the GPU hardware management, and can also be crucial for the SIMD performance of our specific application. Both HR-case searches show indeed important static divergence (at compile time, in their control flow), but the regular HR-case search presents low dynamic divergence (at runtime, in its execution flow). This low execution flow divergence can thus be handled efficiently by the GPU hardware, while the CPU compiler has to set all the required masks for predication according to the control flow divergence of the source code. As far as masks with all zeros or all ones are concerned, it has to be noticed that recent work can detect these cases at runtime in order to avoid using code with predication when possible on CPUs [30].

The second issue with the vectorization of the HR-case search on x86 CPUs is the lack of vector integer

[2]We are not aware of the exact 64-bit integer compute power of these GPUs, but their floating-point peak performances are similar : 3520 SGflop/s (single precision) and 1170 DGflop/s (double precision) for the K20c, against 3789 SGflop/s and 947 DGflop/s for the 7970.

division instruction in SSE, in AVX2 [18] and even in the forthcoming AVX-512 [19]. The compiler therefore uses the scalar integer division instruction, or emulates the vector integer division: e.g. with vector subtraction instructions or with optimized intrinsics such as `_mm_div_epu64`, `_mm256_div_epu64` or `_mm512_div_epu64` (from the Intel Short Vector Math Library - SVML).

As far as our performance tests are concerned, the AVX2 server hosts an Intel Xeon E3-1275 v3 CPU, with 4 physical cores running at 3.50 GHz and 2-way SMT, and we use on this server the Intel SDK for OpenCL 2016 and the OpenCL Runtime 15.1. The SSE4.2 server hosts two Intel Xeon E5-2660 CPUs, totalizing 16 physical cores running at 2.20 GHz with 2-way SMT (and using SSE4.2 for integer SIMD operations), as well as a Xeon Phi coprocessor: we use here the Intel SDK for OpenCL 2016 with the OpenCL Runtime 14.2 (latest version for Xeon Phi coprocessors). On AVX and SSE, the OpenCL compiler relies on a heuristic[3] to determine if it is worth generating vector code. We use here the `CL_CONFIG_CPU_VECTORIZER_MODE` environment variable to explicitly force or prevent the OpenCL implicit vectorization.

Figure 5.2(a) shows the performance results of both HR-case searches on the AVX2 server with both vector and scalar codes generated. When inspecting the assembly code generated by the OpenCL compiler, one can see that the vector codes contain AVX2 vector instructions for additions, multiplications and subtractions but only scalar 64-bit integer divisions. No vector division, such as `_mm256_div_epu64`, are generated. Along with the overhead required for masking, this leads to the vectorized versions being slower than the scalar ones. When comparing the regular and Lefèvre HR-case searches, the regular HR-case search is then 1.5x faster in scalar mode, and 1.4x faster in SIMD mode, which nevertheless shows the benefit of the regular HR-case search on CPUs.

The same conclusions apply to the SSE4.2 server[4] (cf. Fig. 5.2(b)): the use of scalar 64-bit integer division and the cost of masking inhibit SIMD performance gains. On the SSE4.2 server, the regular HR-case search delivers in the end the same performance gains over the Lefèvre one as on the AVX2 server.

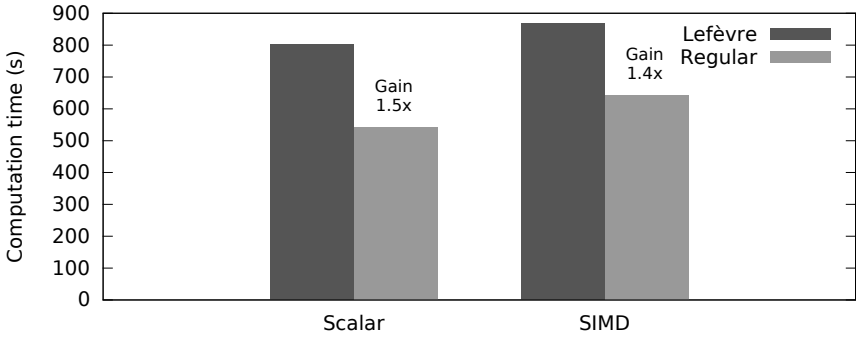
As far as the SIMD division issue is concerned, it can be noticed that we also tried to use `LOGMS` = 64 (cf. Sect. 3) instead of the optimal `LOGMS` in order to implement divisions with SIMD subtractions and thus try to avoid this issue: the scalar execution time will not be optimal, but the SIMD speedup could improve performance in the end. This however only leads to the vector versions being almost as fast as the scalar ones on both servers, the overhead of masking still inhibiting SIMD performance gains, and this thus results in an overall performance loss.

5.3. The Xeon Phi coprocessor. Since masking hinders SIMD performance gains on CPUs, we now target a Xeon Phi coprocessor (Knights Corner 5110P, with 60 cores with 4-way SMT at 1.053 GHz). Indeed the overhead of masking for SIMD control flow divergence is lower on Xeon Phi than on AVX2 or SSE4.2 CPUs since all Xeon Phi SIMD instructions directly support a 16-bit mask to control which lanes are active or not during the instruction execution. This avoids the predication required for SSE or AVX, but still requires a software management of the masks. However, there are no 64-bit integer SIMD arithmetic operations on the current Xeon Phis. The compiler must therefore emulate these 64-bit SIMD operations with 32-bit integer SIMD operations.

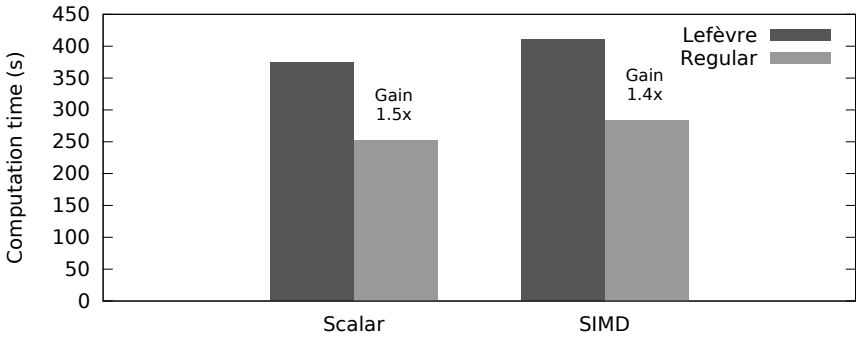
As far as SIMD 64-bit integer division is concerned, the assembly SIMD code generated from the OpenCL kernel shows both scalar 64-bit integer divisions and calls to SVML 64-bit integer functions. Since we are unable to determine which ones are indeed executed at runtime, we performed some micro-benchmarks of the 64-bit integer division on the Xeon Phi as presented on table 5.2. All tests on the Xeon Phi have been performed with the Intel SDK for OpenCL 2016 with the OpenCL Runtime 14.2 and the Intel Manycore Platform Software Stack 3.4. Scalar code is obtained thanks to the `CL_CONFIG_USE_VECTORIZER` environment variable. One can see that the vectorized OpenCL kernel offers the same SIMD speedup and performance as the C+SVML code. Even if our HR-case search kernels are thus likely to also use this SVML 64-bit integer division, the SIMD speedup for this operation is actually low: only up to 2.7x, whereas we have obtained SIMD speedups between 7.2x and 7.7x for 64-bit additions and multiplications (8x being the maximum theoretical speedup for 64-bit SIMD operations on the Xeon Phi). This shows that the SIMD 64-bit integer division is currently a potential bottleneck for SIMD performance on the Xeon Phi.

[3] See: `https://software.intel.com/en-us/node/540483`

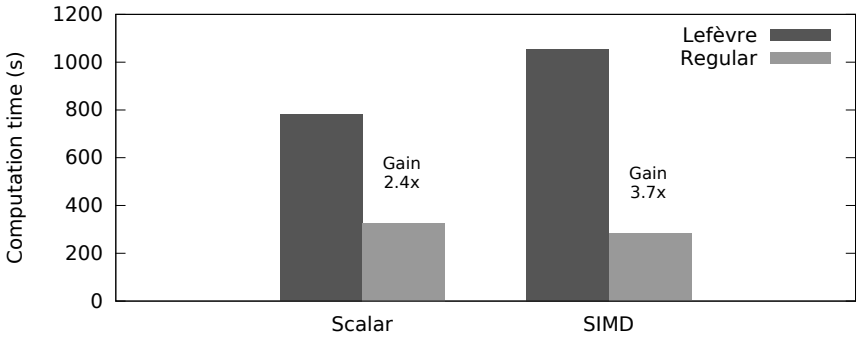
[4] Computation times are smaller on the SSE4.2 server than on the AVX2 server because of its higher number of CPU cores.



(a) AVX2 server



(b) SSE4.2 server



(c) Xeon Phi coprocessor

FIG. 5.2. *Performance results over $I_{0..1023}$ for Lefèvre and regular HR-case searches, in scalar and SIMD modes.*

Figure 5.2(c) shows the performance results of our HR-case searches on the Xeon Phi. Thanks to the more efficient masking on the Xeon Phi, we notice a 13% performance gain for SIMD code over scalar code with the regular HR-case search. This is an improvement over SSE and AVX CPUs, but the SIMD gain is very low: this can be explained by the emulation of 64-bit SIMD operations by 32-bit SIMD operations (while 64-bit scalar operations are not emulated), by the SIMD 64-bit integer division performance and by the software management of the masks. Again, using `LOGMS=64` to avoid the SIMD division issue does not improve these performance results. The SIMD version of the Lefèvre HR-case search shows however a performance loss with respect to its scalar version, probably due to its higher dynamic divergence. When comparing the two HR-case searches, the SIMD regular HR-case search is 3.7x faster than the SIMD Lefèvre HR-case search, and 2.8x faster than the

TABLE 5.2
Micro-benchmarks of the 64-bit integer division on the Xeon Phi. Computations are repeated 10^6 times on two input arrays of 1024 64-bit integers, with a scalar C code, a SIMD C+SVML code and an OpenCL kernel (vectorized or not).

	C [+ SVML]	OpenCL
Scalar	97.9 s	94.2 s
SIMD	36.7 s	39.9 s
SIMD speedup	2.7x	2.4x

scalar Lefèvre HR-case search on this architecture. This once again shows the interest of the regular HR-case search.

5.4. Architecture comparison. Thanks to the OpenCL portability, we can now compare the following different architectures for solving the TMD: the NVIDIA K20c and AMD 7970 GPUs, the SSE4.2 CPU server[5] and the Intel Xeon Phi. We first emphasize that the maximum power comsumptions are nearly the same: 225W for the K20c GPU, 250W for the 7970 GPU, 190W of TDP for the SSE4.2 server (with two Intel Xeon E5-2660 CPUs), and 225W of TDP for the Xeon Phi. In order to compute the 1024 intervals $I_{0..1023}$, with the regular HR-case search which performs best on all architectures, both GPUs require less than 40 s, whereas the SSE4.2 server and the Xeon Phi require at least 250 s. This 6.25x performance gap is clearly due to the inefficient SIMD execution of the HR-case search on the SSE4.2 CPUs and on the Xeon Phi.

6. Conclusion. In this paper, we have shown that handling efficiently the divergence on SIMD architectures for the most time consuming step of the Table Maker's Dilemma solving requires to use regular algorithms, with the relevant programming model, but also depends on the hardware. Using algorithmic changes, we can strongly reduce the divergence in the conditional statements within the main loop, as well as reduce the execution flow divergence on this main loop. Using OpenCL with its SPMD-on-SIMD programming model and its implicit vectorization feature, our massively parallel algorithm can be easily implemented and deployed on various GPUs and CPUs, as well as on the Intel Xeon Phi coprocessor.

Compared to the previous CUDA implementation, our OpenCL implementation shows similar performance gains (2.9x) for our regular algorithm on NVIDIA GPUs. This regular algorithm is even more decisive on AMD GPUs, with 10.5x performance gains, since their SIMD execution width is larger. However, when considering the SIMD units of CPUs and of the Xeon Phi, we show no or low performance gains for the SIMD execution over the scalar one. This is due the SIMD integer division implementation, to the lack of SIMD 64-bit integer instructions on the Xeon Phi, as well as to the static software handling of divergence on CPUs. This latter implies indeed an overhead compared to the dynamic hardware handling of divergence on GPUs, and cannot take full advantage of our regular algorithm, which presents important divergence in its control flow, but low divergence in its execution flow. However, it has to be noticed that the regular HR-case search still offers performance gains ranging between 1.5x and 2.8x on CPUs and on the Xeon Phi.

Currently, the solving of the Table Maker's Dilemma is thus more efficiently performed on GPU architectures due to their divergence handling. However, more efficient emulations of the SIMD integer division could be possible, and the forthcoming Xeon Phi processors (Knights Landing with AVX-512 SIMD instruction set) will support SIMD 64-bit integer instructions while maintaining a lower masking overhead than CPUs. This could lead to better SIMD performance gains on this architecture, especially for our regular algorithm.

Finally, it can also be noticed that FPGA vendors like Xilinx and Altera now support OpenCL. Our OpenCL implementation could thus straightforwardly be deployed on FPGA, tailoring the FPGA hardware to our algorithm. The performance of this FPGA deployment could then be compared with GPU and CPU deployments, as well as with other FPGA implementations for solving the TMD.

Acknowledgments. This work was supported by the TaMaDi project of the French ANR (grant ANR 2010 BLAN 0203 01). The authors would like to thank A. Zaks and A. Narkis from Intel for helpful discussions on the Xeon Phi and on OpenCL. The authors would also like to thank Pierre-Emmanuel Le Roux (LIP6) for

[5]The AVX2 server has only 4 CPU cores, and no speedup is obtained on CPU thanks to vectorization (in AVX2 as in SSE4.2): we therefore only consider the SSE4.2 server here.

managing the compute servers, and the SFPN specialty (numerical security, reliability and performance) of the master in computer science at *Université Pierre et Marie Curie* for providing access to the AVX2 server.

REFERENCES

[1] AMD, *3DNow! Technology Manual,* 2000.
[2] AMD, *Accelerated Parallel Processing OpenCL Programming Guide,* revision 2.7, November 2013.
[3] N. Brunie, S. Collange and G. Diamos, *Simultaneous Branch and Warp Interweaving for Sustained GPU Performance,* in Proceedings of the International Symposium on Computer Architecture (ISCA'12), 4960, 2012.
[4] M. Cornea, IEEE 754-2008 Decimal Floating-Point for Intel, ARITH, IEEE Symposium on Computer Arithmetic, 2009, pp. 225-228.
[5] K. Diefendorf, *Altivec extension to Power PC accelerates media processing,* 2001.
[6] F. de Dinechin, J.-M. Muller, B. Pasca and A. Plesco, *An FPGA architecture for solving the Table Maker's Dilemma,* in Proceedings of the 22nd IEEE International Conference on Application-Specific Systems, Architectures and Processors, 187194, 2011.
[7] B. Gaster, L. Howes, D.R. Kaeli, P. Mistry and D. Schaa, *Heterogeneous Computing with OpenCL: Revised OpenCL 1.2 Edition,* Newnes, 2012.
[8] P. Fortin, M. Gouicem and S. Graillat, *Towards solving the Table Maker's Dilemma on GPU,* in Proceedings of the 20th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, pp. 407-415, 2012.
[9] P. Fortin, M. Gouicem and S. Graillat, *GPU-accelerated generation of correctly-rounded elementary functions,* ACM Transactions on Mathematical Software (to appear).
[10] P. Fortin, M. Gouicem and S. Graillat, *GPU-accelerated generation of correctly-rounded elementary functions,* Research Report hal-00751446 v2, `https://hal.archives-ouvertes.fr/hal-00751446`
[11] S. Frey, G. Reina and T. Ertl, *SIMT Microscheduling: Reducing Thread Stalling in Divergent Iterative Algorithms,* in Proceedings of the 20th Euromicro International Conference on Parallel, Distributed and Network-based Processing, 399406, 2012.
[12] W. W. L. Fung, I. Sham, G. Yuan and T. M. Aamodt, *Dynamic Warp Formation: Efficient MIMD Control Flow on SIMD Graphics Hardware,* ACM Trans. Archit. Code Optim. 6(2), Article 7, 2009.
[13] T.D. Han and T.S. Abdelrahman, *Reducing branch divergence in GPU programs,* in Proceedings of the Fourth Workshop on General Purpose Processing on Graphics Processing Units, 3:13:8, 2011.
[14] J.L. Hennessy, D.A. Patterson, *Computer Architecture, Fifth Edition: A Quantitative Approach,* The Morgan Kaufmann Series in Computer Architecture and Design, 2011
[15] IEEE Computer Society, *IEEE Standard for Floating-Point Arithmetic,* 2008.
[16] Intel Developer Services, *MMX Technology Technical Overview,* 1996.
[17] Intel, *Intel SSE4 Programming Reference,* Reference number: D91561-003, 2007.
[18] Intel, *Intel Architecture Instruction Set Extensions Programming Reference,* Number: 319433-012A, 2012.
[19] Intel, *Intel Architecture Instruction Set Extensions Programming Reference,* Number: 319433-024, 2016.
[20] Intel, *A Guide to Vectorization with Intel C++ Compilers,* 2012
[21] V. Lefèvre, J.-M. Muller and A. Tisserand, *Toward correctly rounded transcendentals,* IEEE Transactions on Computers, 47(11), pp. 1235-1243, 1998.
[22] V. Lefèvre, *New Results on the Distance between a Segment and $\mathbb{Z}^2$. Application to the Exact Rounding,* Proceedings of the 17th IEEE Symposium on Computer Arithmetic, pp. 68-75, 2005.
[23] L. D. McFearin, D. W. Matula, *Generation and Analysis of Hard to Round Cases for Binary Floating Point Division,* ARITH, IEEE Symposium on Computer Arithmetic, 2001.
[24] J. Meng, D. Tarjan and K. Skadron, *Dynamic Warp Subdivision for Integrated Branch and Memory Divergence Tolerance,* in Proceedings of the International Symposium on Computer Architecture (ISCA'10), 2010.
[25] J.-M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé and S. Torres, *Handbook of floating-point arithmetic,* Springer, 2010.
[26] M. Pharr and W.R. Mark, *ispc: A SPMD compiler for high-performance CPU programming,* In Innovative Parallel Computing (InPar), pp. 1-13, 2012.
[27] N. Rotem, *Intel OpenCL Implicit Vectorization Module,* 2011 LLVM Developers' Meeting
[28] T. Schaub, S. Moll, R. Karrenberg and S. Hack, *The Impact of the SIMD Width on Control-Flow and Memory Divergence,* ACM Trans. Archit. Code Optim., 11(4), 2015.
[29] D. Stehlé, V. Lefèvre and P. Zimmermann, *Searching Worst Cases of a One-Variable Function Using Lattice Reduction,* IEEE Trans. Comput., 54, 340346, 2005
[30] S. Timnat, O. Shacham and A. Zaks, *Predicate Vectors If You Must,* Workshop on Programming Models for SIMD/Vector Processing, 2014, colocated with the 19th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP), 2014
[31] E. Z. Zhang, Y. Jiang, Z. Guo and X. Shen, *Streamlining GPU Applications On the Fly: Thread Divergence Elimination through Runtime Thread-Data Remapping,* in Proceedings of the 24th ACM International Conference on Supercomputing (ICS '10), 2010.

Edited by: Dana Petcu
Received: Apr 7, 2016
Accepted: May 2, 2016

DOI 10.12694/scpe.v17i3.1184
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 17, Number 3, pp. 251–270. http://www.scpe.org

COMMUNICATION-AWARE APPROACHES FOR TRANSPARENT CHECKPOINTING IN CLOUD COMPUTING

SAMY SADI* AND BELABBAS YAGOUBI†

Abstract. Checkpoint/Restart or checkpointing is a fault tolerance technique which consists on taking frequent snapshots of an application, so that, in the event of a failure, the application's state can be restored and the application's execution continued without necessarily restarting it. The advent of Cloud Computing brought new challenges with regard to this technique as Fault Tolerance needs to be supplied transparently in environments running highly heterogeneous applications. In this context, we propose two new fully transparent checkpointing approaches. Both approaches use communication-induced checkpointing and guarantee a consistent view of the applications with regard to the outside world process. The first approach is uncoordinated and creates checkpoints for applications independently. The second approach is coordinated, and applications are first grouped into clusters before the checkpointing process is started. We have compared the proposed approaches with state of the art approaches. The results show that our approaches perform better when considering the communication latencies, and the overhead on the execution of the Virtual Machines.

Key words: Cloud Computing, Fault Tolerance, Uncoordinated Checkpointing, Coordinated Checkpointing, Dynamic Clustering, Performance Evaluation, Simulation

AMS subject classifications. 68M14, 68M15, 68M20, 68U20

1. Introduction.

Over the past few years, Cloud Computing [2, 17] has been adopted increasingly as the key solution to satisfy the ever-growing need for computing, storage and networking resources. Among its different strengths and promises, Cloud Computing allows users to easily deploy their applications and control running costs while delegating the hassle of infrastructure management and quality of service enforcement to the provider. This one should, for its part, use different techniques to guarantee resources availability and to offer a secure and fault tolerant environment for users.

With regard to fault tolerance, the Cloud provider's task is to ensure that, in the event of a failure, there is no data loss and deployed applications can continue to run flawlessly. This property is very important as failures are not uncommon in the Cloud. In fact, failures are rather a rule than an exception due to the high number of machines and the frequent use of commodity hardware [24]. To illustrate this point, a previous study characterising failures in the Cloud estimated that a proportion of 8% of the machines can expect to see at least one failure each year [24].

In this context, many research efforts have been devoted to address the issue of failures and to make Clouds more reliable. These efforts found strong existing foundations as fault tolerance has already been addressed for decades in computer systems, especially in High Performance Computing (HPC) systems where the average job length is high and job resubmissions costly. Therefore, research on fault tolerance in Cloud Computing mainly focused on adapting and extending existing techniques, while taking into account Cloud specificities. For instance, much effort has been put forth to design transparent and application agnostic fault tolerance techniques that could be leveraged directly by the provider [6, 15, 20].

Among these, checkpointing [8], sometimes referred to as checkpoint/restart, enables applications to continue running even after the occurrence of a failure. This is achieved by taking frequent snapshots (or checkpoints) of the running application's state, which are saved on secondary hardware. Thus, if the primary machine where the application is running fails, the application's state is not lost and a secondary machine can be used to restore the application and continue its execution.

Despite the apparent simplicity of this process, two main issues remain. The first issue arises when considering the efficiency of the checkpointing process, as it should induce a low overhead on the application's execution and on the application's communications. The second issue arises when considering the behaviour of the checkpointed application, as it should not display any inconsistencies to other applications due to the checkpointing process.

*University of Oran1 Ahmed Ben Bella, Department of Computer Science, Oran, Algeria, (samy.sadi.contact@gmail.com)
†University of Oran1 Ahmed Ben Bella, Department of Computer Science, Oran, Algeria, (byagoubi@gmail.com)

In this paper, we address both of these issues by proposing two fully transparent checkpointing approaches for Cloud Computing environments. Both of the approaches are communication-aware as they guarantee a consistent view of the checkpointed applications and they induce a minimal overhead on the communications.

The first proposed approach is uncoordinated. It takes checkpoints for each application separately. The second approach is coordinated. It orchestrates the checkpoint creation among multiple applications after automatically grouping them into clusters.

We have simulated the proposed approaches, and compared them with three other state of the art approaches. The results show that our approaches are better when taking into account both the overhead on the communications and on the execution of the checkpointed applications.

The organisation of the paper is as follows. In the next section, we review existing research on checkpointing. Next, we present our uncoordinated and coordinated checkpointing approaches in Sect. 3 and in Sect. 4 respectively. After that, in Sect. 5, we describe the evaluation of the two proposed approaches and their comparison with three other state of the art approaches. Finally, we conclude and give a preview of our future work in Sect. 6.

2. Related Work. Checkpointing has been extensively addressed in the literature, and much work has been done to improve existing checkpointing approaches and to reduce the checkpointing overhead while guaranteeing an acceptable level of fault tolerance.

A part of these works have undertaken to determine the optimum checkpointing frequency [26, 23]. They have established that a random selection of the checkpointing frequency leads to poor performances, as high values generate high overhead on the application's execution and low values induce poor fault tolerance.

Another part of the literature focused on reducing checkpoints' size by using several means. One of them is data compression [11], which comes with an added overhead during the creation of the checkpoint. Another is incremental checkpointing [1, 9], which relies on identifying and saving the changed state between two consecutive checkpoints instead of saving the whole application's state.

From the implementation perspective of the checkpointing process, three main categories of approaches are described in the literature: application level, user level and system level [7]. The two first categories regroup approaches that need access to the application's code or to the system's libraries. The latter category regroups approaches that are fully transparent. These are preferred in Cloud Computing environments since the control of the providers over the applications is limited.

The literature abounds with examples of system level checkpointing approaches. Most of them rely on the virtualisation layer to easily create checkpoints for whole operating systems [4, 6, 20, 25]. Each operating system being confined in a Virtual Machine (VM), the content of the checkpoints can easily be determined by monitoring memory changes, disk operations and network operations. The checkpoints can then be saved in a secondary machine [6], in memory [25] or in a dedicated checkpoint repository [20].

Our first contribution in this paper consists on extending and improving the existing system level checkpointing system Remus [6]. The choice of Remus has been motivated by two main points. Firstly, Remus offers notable performances as it enables high-frequency checkpointing and can generate checkpoints as often as every 25ms [6]. Secondly, Remus is one of the few checkpointing systems that ensure the consistency of input/output (I/O) communications by using an output commit mechanism (also used in [19]), which is all the more important to ensure a correct execution of the applications. Further description of Remus and of our contribution are given in Sect. 3.

Our second and main contribution in this paper is given in Sect. 4. As many other existing works in the literature, we have addressed the checkpointing coordination issue. However, unlike the majority of the existing contributions which propose application and user level approaches [8], we propose a system level approach. This ties in with many similar efforts including: VCCP [21], VNSnap [13] and ATCCp [18].

VCCP [21] is a virtual cluster checkpointing system that enables checkpointing coordination among multiple VMs. It uses a blocking algorithm to orchestrate the checkpointing process and the recovery process in a virtual cluster. Both of these processes are initiated by the head node of the virtual cluster and require a reliable FIFO data transmission channel.

VNSnap [13] is another system that enables coordinated checkpointing. It uses a non blocking process where much of the checkpointing takes place while the VM is not suspended. For the coordination part, the

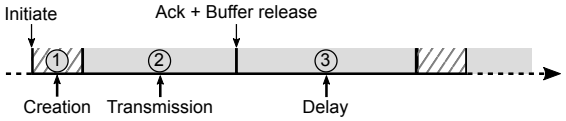


FIG. 3.1. *Checkpointing process of Remus*

authors adapted an existing and non-blocking global snapshot algorithm [16] and showed its applicability.

On a different note, ATCCp, an intra-server coordinated checkpointing approach, has been proposed in [18]. Checkpointing coordination is achieved by taking checkpoints at the same moment based on the VMs' clocks. The main issue raised by the authors is related to the fact that clocks can deviate and thus are not reliable enough. They propose to overcome the issue by using message logging and by piggybacking extra information in the VMs' communications.

There are many differences between our approach and the previously described approaches. The main difference is that previous approaches assume that VMs are grouped in predefined and static virtual clusters, whereas our approach dynamically groups VMs into different clusters. Considering the highly heterogeneous nature of the applications deployed in the Cloud, this improves the transparency of our approach and enhances its performances. Furthermore, while other approaches in the literature only take into account communications of the VMs inside the same virtual cluster, our approach also considers the communications with other systems, whether it is a VM in another cluster or a system outside the Cloud. Such communications are referred in the following as communications to the outside world process (OWP).

3. Improving Remus Using Communication-Induced Checkpointing. Remus [6] is a prominent system level checkpointing approach which enables high-frequency checkpointing while guaranteeing a moderate overhead on the running applications. It is one of the few system level approaches in the literature that keep consistent communications in a faulty environment. However, the output commit mechanism employed to this extent causes an additional delay for communications which is directly dependent on the checkpointing frequency. In fact, to keep reasonable communication latencies, the checkpointing frequency should be high enough which in turn may degrade applications' performances.

In this section, we address this particular issue by using communication-induced checkpointing where we define two types of checkpoints: regular periodic checkpoints, and forced communication-induced checkpoints.

We start by giving an overview of Remus. Then we present current issues with Remus and we bring into view the relationship between the communication latencies and the checkpointing frequency. Finally, we present our contribution.

3.1. Overview of Remus. Remus [6] is a system level checkpointing approach that offers a high degree of fault tolerance. It associates for each VM on a primary machine, a backup VM on a secondary machine which is frequently updated to be near to identical to the primary VM. When the primary VM fails, the secondary VM is started and becomes the new primary VM. The recovery process is totally transparent to the user, as even network addresses of the failed VM are reassigned to the newly started VM.

The checkpointing process defined by Remus is illustrated in Fig. 3.1. It consists of multiple sequences of three stages. During the first stage, the VM is paused and a checkpoint file is created by including the execution state of the VM and all modified memory pages since the last checkpoint. Next, in the second stage, the VM is running speculatively and all network output is buffered. During this stage, the checkpoint file is transmitted to the secondary machine. Once the checkpoint is fully transmitted and acknowledged, all network output that has been buffered before the first stage is released and sent asynchronously to the outside world process (OWP). Finally, during the third stage, the VM continues to run speculatively for a given delay depending on the checkpointing frequency.

The speculative execution used by Remus is an important feature which guarantees that no state of the VM is made externally visible until it is successfully checkpointed. The mechanism used is similar to the output commit mechanism also used in [19]. It relies on buffering the network output of a VM until a checkpoint containing the state that has generated this output is acknowledged. This is illustrated in Fig. 3.2. In this example, input communication data (IN1 and IN2) is immediately transmitted to the VM, whereas the VM's

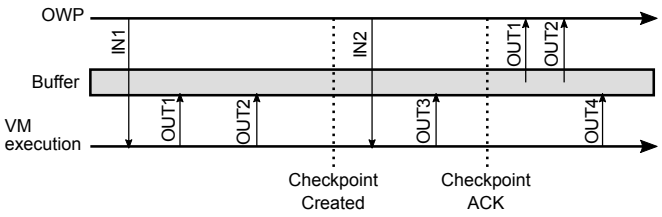


FIG. 3.2. *Time diagram displaying the output commit mechanism used by Remus [6]*

output is first buffered and is released only once a corresponding checkpoint has been acknowledged. Particularly, the output data which is generated after the checkpoint creation (OUT3 and OUT4) is not released immediately, but is released only after another checkpoint is created and acknowledged. This ensures that, when recovering from a failure, the secondary VM will not re-generate the same output.

With regard to disk operations, these are immediately and asynchronously sent to the secondary machine where they are buffered and applied once a checkpoint has been acknowledged. If the primary VM fails before the checkpoint is fully transmitted then the buffered disk operations are discarded.

3.2. Issues with Remus. The main issues with Remus come from the output commit mechanism which is necessary to keep a consistent execution of the VM, but induces an extra-latency on the communications and potentially a notable packet loss.

The extra-latency is due to communications' buffering, which retains the VM's output until a committed (i.e. acknowledged) checkpoint is produced. This aspect incites to use higher checkpointing frequencies to reduce the retention delay and attenuate the latency. However, this also comes at the expense of a higher overhead on the VM's execution.

The other issue concerns the loss of network packets which can affect both the incoming and the outgoing communications, and can happen both during the normal execution of the VM or after a failure.

Loss of input packets during the execution of the VM can happen because of the previously described extra-latency. When the VM takes too long to answer a request, the OWP may assume that the request (i.e. input) is lost and that it has not reached the VM.

Additionally, after a failure, any input that has been sent since last committed checkpoint is lost. Indeed, at the moment of the failure, the secondary VM's state is identical to the primary VM's state when the last committed checkpoint has been created. Thus, it does not reflect the alterations brought by new input.

Likewise, failures can also cause output packet loss. This happens when a failure occurs after the second stage of the checkpointing process when buffered output is being released. The failure will interrupt the output release process and will prevent some packets to be sent. Moreover, when recovering, Remus will not try to resend those packets because last acknowledged checkpoint assumes that they are already transmitted.

A potential solution to the previous issues, when caused by failures, can be envisioned by using packets duplication and logging mechanisms on the secondary machine. However, this will induce a higher overhead on the VM's execution during the checkpointing process. Moreover, since failures are uncommon, and since network communications are usually considered unreliable, Remus can rely on application level protocols (e.g. TCP) to handle packet loss when necessary. Thus, and in light of these elements, addressing failure-induced communication issues is usually worthless. However, communication issues introduced by the checkpointing process during the failure-free execution of the VM should be addressed. These have more frequent consequences and can highly deteriorate the quality of service perceived from the OWP.

3.3. Contribution. In our contribution, we focus on reducing the network communication latencies which are introduced by the checkpointing process during the failure-free execution of the VM. In particular, we enable to reduce the checkpointing frequency, and thus the checkpointing overhead on the VM's execution, with a moderate effect on the communication latencies.

To this extent, we introduce a new type of checkpoint, which we refer to as *forced checkpoint* and comes in addition to regular and periodic checkpoints taken by Remus. This new checkpoint is communication induced, and is taken after the checkpointed VM generates output. Thus, and as opposed to Remus, a checkpoint is

initiated as soon as possible when there is pending buffered output. This new checkpoint addresses the previous issues regarding communication latencies, and packet loss due to communication latencies.

Because taking forced checkpoints immediately after output is available may overburden the VM's execution, we set a small delay α before taking a forced checkpoint. Moreover, before taking a forced checkpoint, we ensure that we do not interfere with another checkpoint which is being created or transmitted. Finally, before initiating the forced checkpoint, we ensure that there is still buffered output waiting to be released. In fact, it is possible that another checkpoint has released the buffered output while waiting for the α delay.

In sum, we define the following steps for taking a forced checkpoint:

1. Wait for buffered output;
2. Wait for a specific time (a delay, denoted by α);
3. If in stage 1 (i.e. checkpoint creation) or in stage 2 (checkpoint transmission) of the checkpointing process (cf. Fig. 3.1) then wait for stage 3;
4. Make sure that there is still buffered output waiting to be released, otherwise go back to the first step;
5. Stop current checkpointing (i.e. stage 3), and immediately initiate a new checkpoint starting from stage 1 (cf. Fig. 3.1);
6. Go back to the first step.

The selection of the α parameter should be done by taking into account two factors. First, it should be big enough to not interfere with the VM execution. Secondly, it should be short enough to minimise communication latencies. In particular with regard to this last point, the α parameter should be set so that the VM response time is short enough to avoid packet loss. If we assume that the OWP will wait for the delay θ before considering that a packet is lost, then the VM response time should be smaller than θ.

On another note, the VM response time after buffering output depends on the α parameter, the checkpoint creation delay δ and the delay for transferring the checkpoint to the secondary machine. This last delay can be computed knowing the checkpoint file size S and the available bandwidth B when transferring the checkpoint from the primary to the secondary machine.

We obtain the following inequation when bounding the response time with the θ parameter:

$$\text{ResponseTime} = \delta + \frac{S}{B} + \alpha \leqslant \theta \tag{3.1}$$

Thus, we obtain the following upper bound for the α parameter to avoid packet loss:

$$\alpha_{\max} = \theta - \delta - \frac{S}{B} \tag{3.2}$$

We define no lower bound for α. Clearly, lower values will give smaller communication latencies but they will also generate higher overhead on the VM's execution. Similarly, higher values will generate lower overheads but higher latencies. A compromise would be to set:

$$\alpha = \frac{1}{2} \cdot \alpha_{\max} \tag{3.3}$$

4. System Level Coordinated Checkpointing Approach. In this section, we present our approach for system level checkpointing coordination. We extend our approach described in Sect. 3 by using a dynamic clustering approach and by coordinating the checkpoint creation at each cluster level.

This approach eliminates the latencies introduced by the checkpointing process for intra-cluster communications. Moreover, when compared with other approaches in the literature, our approach brings two novelties. First, our dynamic clustering approach enables to define optimum cluster sizes which will reduce the periodic checkpointing overhead while keeping a moderate effect on communication latencies to the OWP. Secondly, we do not only focus on keeping a consistent execution state inside the same cluster. We also strive to keep a global consistent state with regard to the OWP.

In the following, we first start by giving a general overview of our approach and the motives behind clustering and checkpointing/recovery coordination. Next, we present the system design used in our approach and we describe the role of each component in the system. After that, we describe the dynamic clustering process. Then, we present our approach for checkpointing coordination. Finally, we describe the recovery process.

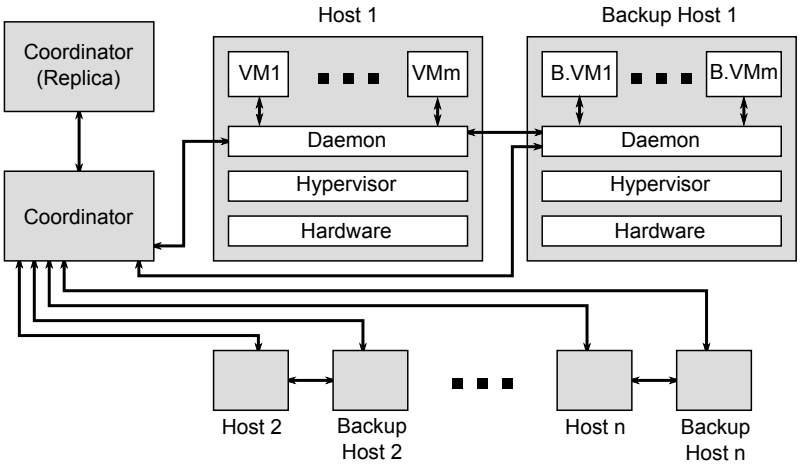


FIG. 4.1. *System Design used in the proposed approach*

4.1. Overview of the proposed approach. Guaranteeing a consistent global state for applications during the checkpointing process and after recovery in the event of a failure is an important property. The uncoordinated checkpointing approach presented in Sect. 3 provides this property by using a buffering mechanism which ensures that no state of the VM is made externally visible until a checkpoint is created. This is a viable solution, but has many drawbacks on the communications. In particular, using the proposed communication-induced checkpoints can indeed potentially reduce communication latencies but it will also cause important overheads for communication-intensive VMs.

To address these issues, we propose to improve the previous approach by using two techniques: clustering and checkpointing/recovery coordination. In the first technique, we propose to identify highly-coupled VMs and to group them into the same cluster. In the second technique, we propose to coordinate the checkpointing process and the recovery process among all VMs belonging to the same cluster. In particular, creating a checkpoint for a VM in a given cluster will imply creating a checkpoint for all VMs in that cluster. Similarly, if a VM in a given cluster fails, then all VMs in that cluster are rolled back accordingly to the previous checkpoint.

As a result, communications inside each cluster can be immediately delivered without introducing inconsistencies or extra-latencies. However, VMs' communications to the OWP (i.e. VMs in other clusters) still have to be buffered. Therefore, we keep the same output commit mechanism, also used in the uncoordinated checkpointing approach presented in Sect. 3, for communications to the OWP.

4.2. System Design. Our approach relies on two important features: automatic clustering of VMs and checkpointing/recovery coordination for VMs belonging to the same cluster. This entails the use of a coordinator component to manage these two processes. Moreover, another component is also needed on each machine to handle the checkpointing process for each VM, and to handle the communications with the coordinator component. The complete system design is displayed in Fig. 4.1, and the role of each component is given in the following.

4.2.1. The coordinator. The coordinator component has four different roles:

- **Information gathering:** it collects the information provided by the daemon component on each machine, which is necessary for the clustering process.
- **Clustering:** based on the collected information, VMs are grouped into different clusters. Then, the daemon component in each machine is informed about the clustering. This is necessary so that the daemon can differentiate between intra-cluster communications which can be immediately delivered, and communications to the OWP which have to be buffered.
- **Checkpointing coordination:** it manages the checkpointing process inside different clusters. The checkpointing process can either be initiated by this component to conform to a given checkpointing frequency, or it can be initiated after a request of the daemon component.

- **Recovery coordination:** it manages the recovery process inside different clusters. This process is initiated by the daemon component after a failure.

4.2.2. The coordinator replica. This component is a substitute for the coordinator component. It is used after a failure of the coordinator component. It has two different roles:

- **Replication of the coordinator:** frequently enough, this component copies the information gathered by the coordinator. It also keeps a copy of the clustering state which is transmitted by the coordinator after the clustering process. The process for transmitting the clustering state to this component and to the daemon components is described further in this paper. This process ensures that the clustering information is consistent among all components even after the occurrence of a failure.
- **Detection of the failure of the coordinator:** using a heartbeat failure detection technique. Once the failure is detected, this component sends a message to all daemon components to inform them about the failure. Then, this component becomes the new coordinator, and another coordinator replica component is created on another machine.

4.2.3. The Daemon. This component runs on top of the hypervisor and provides host functionality for the checkpointing and the recovery processes. It is present in both the primary host and the backup host, and has the following roles:

- **Probing:** this component collects different information on VMs' communications. These are transmitted to the coordinator component and are used during clustering process.
- **Communication buffering:** this component analyses outgoing VM's communications and automatically buffers communications to the OWP. To do so, it associates a separate buffer for each VM.
- **Checkpointing:** this component supervises the checkpointing process for VMs on the host level. This includes checkpoint creation and checkpoint transmission to the backup host. Furthermore, this component should also communicate with the coordinator component and initiate the checkpointing process when asked to. It can also request a checkpoint creation for a cluster by sending a request to the coordinator. More details about the checkpointing process is given in Sect. 4.4.
- **Failures detection:** this component uses a heartbeat failure detection technique to detect a failure of the backup host or of the primary host. If a failure of the backup host is detected, then another backup host is selected and this component initiates a checkpointing process by sending a request to the coordinator component. If a failure of the main host is detected, then a recovery process is initiated by sending a request to the coordinator component. After the recovery, the backup host becomes the new primary host, and another backup host is assigned to the primary host. The recovery process is further described in this paper.

4.3. Dynamic Clustering. The dynamic clustering process consists on attributing a cluster for each VM being checkpointed. The objective during this process is to generate a set of clusters that reduces communication latencies and the overhead which is due to the checkpointing and the recovery process inside each cluster.

In the following, we first formalise the clustering problem and we give the cost function to optimise during the clustering. Then, we give a heuristic to solve the clustering problem. Finally, we describe the steps taken by the coordinator component during the clustering process which ensures that no inconsistent checkpoints are created even if the clustering configuration is changed.

4.3.1. Problem Definition and Cost Function. The clustering problem consists on attributing for a set of n VMs $\mathcal{V} = \{vm_0, vm_1, \ldots, vm_n\}$, a partition $\mathcal{C} = \{c_0, c_1, \ldots, c_m\}$ such that:

$\bigcup_{c_i \in \mathcal{C}} = \mathcal{V}$ and

if $c_i, c_j \in \mathcal{C}$ and $c_i \neq c_j$ then $c_i \cap c_j = \emptyset$

We associate for each cluster c_i, a cost function $f(c_i)$ which is computed based on the checkpointing cost of the cluster, noted $h(c_i)$, and the recovery cost of the cluster, noted $r(c_i)$. Such that:

$$f(c_i) = h(c_i) + r(c_i) \tag{4.1}$$

We do not formulate the communication latencies which are due to the checkpointing process directly as part of the cost function $f(c_i)$. In fact, and according to the definition of our checkpointing approach,

communication latencies are due to communications to the OWP which are buffered and only released after a checkpoint is taken. Moreover, every time communications are buffered, a communication-induced checkpoint is scheduled and taken to minimise communication latencies. As a consequence, communication latencies are directly correlated to the checkpointing frequency: the more often we take communication-induced checkpoints, the more often communications are retained and the more important are the induced communication latencies. Thus, by reducing the frequency of communication-induced checkpoints (i.e. the checkpointing cost), we also reduce communication latencies.

Accordingly, we define the cost function for a partition $\mathcal{C}$ as follows:

$$F(\mathcal{C}) = \sum_{c_i \in \mathcal{C}} f(c_i) \tag{4.2}$$

We also define the optimum clustering as a partition $\mathcal{C}$ which minimises the cost function (4.2).

With regard to the checkpointing and the recovery costs for a cluster c_i, they are both computed based on the time lost which is introduced by the checkpointing process on the VMs' execution inside the cluster, and the time lost which is due to the recovery process. These two values are computed for a delay $\mathcal{T}$ which can be arbitrarily set.

The time lost due to the checkpointing process consists of the sum of the checkpointing overheads induced for each VM in the cluster during the chosen delay $\mathcal{T}$. Assuming the checkpointing frequency for the cluster c_i during the delay $\mathcal{T}$ is $v(c_i)$, and assuming δ_{vm_a} is the the checkpointing overhead corresponding to the VM vm_a, then:

$$h(c_i) = v(c_i) \cdot \sum_{vm_a \in c_i} \delta_{vm_a} \tag{4.3}$$

Because the checkpointing process is coordinated for all VMs inside the cluster, the checkpointing frequency $v(c_i)$ corresponds to the highest VM checkpointing frequency in the cluster. Besides, checkpoints are either initiated by the coordinator component given a frequency ν_{vm_a} for each VM vm_a, or by a daemon component after buffering communications. In such case, the checkpointing frequency is determined by the communication frequency of the VMs to the OWP.

We note μ_{vm_a,vm_b} the communications frequency from vm_a to vm_b during the delay $\mathcal{T}$. Accordingly, we can estimate the checkpointing frequency for a cluster c_i as follows:

$$v(c_i) = \max_{vm_a \in c_i} \left(\nu_{vm_a}, \max_{vm_b \in \mathcal{C} - c_i} \mu_{vm_a,vm_b} \right) \tag{4.4}$$

The time lost which is due to the recovery process can be estimated, for its part, knowing the failures frequency in the cluster. Assuming λ as the average failure frequency for a VM during the delay $\mathcal{T}$, and assuming m the number of VMs in the cluster c_i, the failure frequency for a cluster c_i during the same delay is then $m \cdot \lambda$.

After a failure in the cluster, all VMs in the cluster are rolled back to the previous checkpoint. Assuming that the failure happens on average at the middle way between two checkpoints, we can estimate the time lost due to a failure for each VM to be half the checkpointing interval. Thus, the time lost in the cluster after one recovery, which is the sum of time losses for each VM, can be computed as follows:

$$r_{\text{single failure}}(c_i) = m \cdot \frac{\mathcal{T}}{2 \cdot v(c_i)} \tag{4.5}$$

Finally, we can estimate the time lost due to the recovery process as follows:

$$r(c_i) = m^2 \cdot \lambda \cdot \frac{\mathcal{T}}{2 \cdot v(c_i)} \tag{4.6}$$

4.3.2. Heuristic for solving the clustering problem. The clustering problem is an instance of the set partitioning optimisation problem which has been proven to be NP-hard [3]. Thus, unless $\mathcal{P} = \mathcal{NP}$, there is no algorithm that runs in polynomial time and brings an exact solution to the problem. However, it is possible to approximate the optimum solution using appropriate heuristics. In the following, we propose a $\mathcal{O}(n^3)$ algorithm that uses a greedy strategy to solve the clustering problem (see Algorithm 1).

The proposed algorithm receives a list of VMs as input and generates a partition containing the clusters as output. It operates as follows. Firstly, it creates an initial partition where each VM is placed in its own cluster. Then it proceeds by stages in which clusters are merged two by two until no more clusters can be merged. In each stage, the algorithm computes the cost gain after merging two clusters by computing the difference between the cost induced by the clusters separately and the cost induced by the two clusters when merged. The two clusters that generate the highest cost difference if merged, are then combined.

Algorithm 1 A greedy algorithm for the clustering problem

INPUT: Virtual Machines $\mathcal{V} = \{vm_0, vm_1, \ldots, vm_n\}$
INPUT: The cost function $f()$ which computes the cost corresponding to a cluster as previously defined
OUTPUT: The partition $\mathcal{C}$ of $\mathcal{V}$ which contains the clusters

```
1: ▷ Initiate clusters
2: C ← ∅
3: for all vm_a ∈ V do
4:     C ← C ∪ {{vm_a}}
5: ▷ Start merging clusters
6: repeat
7:     toRemove ← ∅
8:     merged ← ∅
9:     a ← 0
10:    for all c_i ∈ C do
11:        f_i ← f(c_i)
12:        for all c_j ∈ C − {c_i} do
13:            f_j ← f(c_j)
14:            f_t ← f(c_i ∪ c_j)
15:            d ← (f_i + f_j) − f_t                ▷ Computes the cost gain
16:            if d ≥ a then
17:                a ← d
18:                toRemove ← {c_i, c_j}
19:                merged ← c_i ∪ c_j
20:    if toRemove ≠ ∅ then
21:        C ← (C − toRemove) ∪ {merged}
22: until toRemove = ∅
23: return C
```

4.3.3. Clustering process. The clustering process can be started at any moment during the lifetime of a VM by the coordinator component. Initially, each VM is attributed its own cluster. Next, after that enough information is gathered about the VMs' communication profiles by the daemon components, a clustering process is engaged and another clustering configuration is defined. The clustering process can be started whenever important changes on the VMs' communication profiles are detected.

The role of the clustering process is to ensure that the new clustering configuration is correctly transmitted to all daemon components and to the coordinator replica. Additionally, this process should not generate any inconsistencies with regard to the created checkpoints nor to the communications.

We propose, to this extent, a blocking process, in which all VMs are first paused before the clustering configuration is transmitted and applied. This process comprises the following steps:

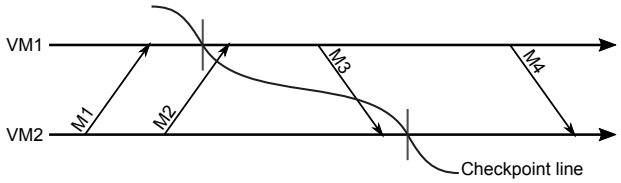


FIG. 4.2. *Example of the state of the communication channels before, during and after checkpointing for two VMs*

1. If there is any ongoing checkpointing or recovery process then wait for it to finish;
2. The coordinator component sends a message to all daemon components to pause the execution of all VMs;
3. Once all VMs are paused, the new clustering configuration is sent to all daemon components and to the coordinator replica component;
4. While the VMs are still paused, a checkpointing process is initiated for all new clusters;
5. Once all checkpoints have been taken, a message is sent to all daemon components to release buffered output to the OWP and to resume VMs' execution.

4.4. Checkpointing process. The checkpointing process is coordinated among all VMs belonging to the same cluster. This process can be initiated in two different ways, either by the coordinator component, or after a request made by the daemon component. The coordinator component initiates the checkpointing process periodically according to a given frequency. The daemon component, for its part, requests to initiate the checkpointing process after buffering output for a VM.

Checkpoints requested by the daemon are communication-induced and have the same role as the forced checkpoints previously described in Sect. 3. They are initiated to minimise the packets retention delay in the buffer, and thus, to mitigate communication latencies to the OWP. As previously noted, these checkpoints should not be initiated immediately after communications are buffered. Instead, there must be a certain delay between the moment when communications are buffered, and the moment when the daemon component requests for a checkpoint. This delay can be computed using the same method described for our uncoordinated approach (cf. Eq. (3.3)).

After the checkpointing process is initiated for a given cluster, it is supervised by the coordinator component. This component requests a checkpoint for each VM in the cluster by sending a message to the appropriate daemon component. During this process, the coordinator must ensure that the resulting checkpoints define a consistent global snapshot [5].

In this context, there are many works in the literature that propose global snapshot algorithms [16, 14, 10]. These algorithms try to ensure two properties. First, a message issued by a VM which has already recorded its state (i.e. taken a checkpoint), should not be delivered to the destination VM until it has recorded its own state (e.g. M3 in Fig. 4.2). Secondly, after recording its state, a VM should also record all in-transit messages which were sent by other VMs before they recorded their own state (e.g. M2 in Fig. 4.2).

The first property is all the more important to guarantee a consistent state after recovering from a failure. This property ensures that if we have recorded a message reception in the checkpoint of a VM, then we also have recorded the corresponding message emission in the checkpoint of the VM that has sent the message [5]. Put differently, this also means that there is no recorded state (e.g. message reception) which depends on a future and un-recorded state (e.e. message emission).

The second property is necessary to avoid the loss of in-transit messages which can happen after recovering from a failure. This property is not critical when the communication channels are assumed unreliable. In such a case, a message loss which is due to a VM failure cannot be distinguished from a message loss which is due to a failure in communication channels. Hence, process level protocols (e.g. TCP) can be used to recover such messages if necessary.

In our work, we have assumed unreliable communication channels. Thus, we only strive to honour the first property as it is essential to ensure the consistent execution of VMs after recovering from a failure. In contrast, the second property is less important because its only role is to prevent the loss of in-transit messages after the

occurrence of a failure, which is rather not frequent.

In our approach, the snapshot algorithm we use is a simplification of the existing Mattern's algorithm [10]. We chose this algorithm because it is non-blocking and does not require FIFO communication channels. Our contribution consists on adapting this algorithm to our approach. In particular, we assume that the checkpointing process can only be initiated by the coordinator component, and we allow the loss of in-transit messages.

In the following, we give a short description of the Mattern's global snapshot algorithm. Then we describe our simplified version of this algorithm. Finally, we give our approach for coordinated checkpointing.

4.4.1. Overview of Mattern's snapshot algorithm. This algorithm [10] is an extension of Chandy and Lamport's snapshot algorithm [5] when non-FIFO communication channels are used. It can be used to create a consistent snapshot of a virtual cluster including the state of all communication channels (i.e. including in-transit messages). To this end, it uses a colouring principle in which a VM is either white or red. This colour is then piggybacked to all outgoing messages before they are sent to other VMs in the cluster.

Initially, the white colour is attributed to all VMs in the cluster. Any VM in the cluster can then initiate the global snapshot process. First, the initiator VM will take a local checkpoint and turn red. Next, it will send a control message to all other VMs in the cluster. Once a white VM receives the control message, it will take a local checkpoint and turn red in its turn.

In the meanwhile, the execution of VMs is not blocked and messages are continually exchanged. However, particular actions are taken when a white VM receives a red message, or when a red VM receives a white message.

First, if a white VM receives a red message, then the message is buffered and is only processed after the VM has turned red in its turn. This way, the first property which was previously discussed is guaranteed. Indeed, a message reception cannot be recorded in a checkpoint if the state that has generated it is not also recorded in a checkpoint.

Secondly, if a red VM receives a white message, then the message is identified as an in-transit message and is saved in the checkpoint as part of the state of the communication channels. To detect when there is no more in-transit messages, a counter is piggybacked in addition to the message's colour in all exchanged messages. This way, after the termination of the snapshot algorithm, all in-transit messages are known and the previously discussed second property is satisfied.

4.4.2. Simplifications. We simplify the previous algorithm by discarding in-transit messages. In other words, we do not save white messages as part of the checkpoint when received by red VMs.

As a result, the snapshot algorithm can be terminated as soon as all VMs in the cluster turn red. Thus, we do not need to keep track of the count of transmitted/received messages nor do we need to transmit this information.

This simplification will induce the loss of in-transit messages after a failure. However, as previously stated, this does not cause any inconsistencies because we have assumed lossy communication channels.

4.4.3. Proposed approach. In our approach, we use the previously described simplified version of the Mattern's algorithm [10] to make sure that the checkpoints created for VMs in a given cluster are globally consistent. Our contribution consists on defining a new approach based on this algorithm which, in addition to guaranteeing globally consistent checkpoints, also brings the following functionality:

- the management of errors during the creation of local checkpoints;
- the detection of the termination of the global snapshot algorithm;
- and the support for an output commit mechanism for communications to the OWP.

To achieve these goals, we rely on the cooperation of the coordinator component and the daemon component. For each component, we define a set of variables which are necessary for the checkpointing process to function, and a set of steps which are taken as part of the checkpointing process. These steps may involve the exchange of control messages between components, which is done using a failure-free communication protocol (e.g. TCP).

We detail these steps and the role of each component during the checkpointing process in the following.

The coordinator component. It orchestrates the checkpointing process inside each cluster. We do not make any assumption on the number of clusters being checkpointed at the same moment. Multiple clusters may engage in the checkpointing process at the same moment or at different moments without interfering. However,

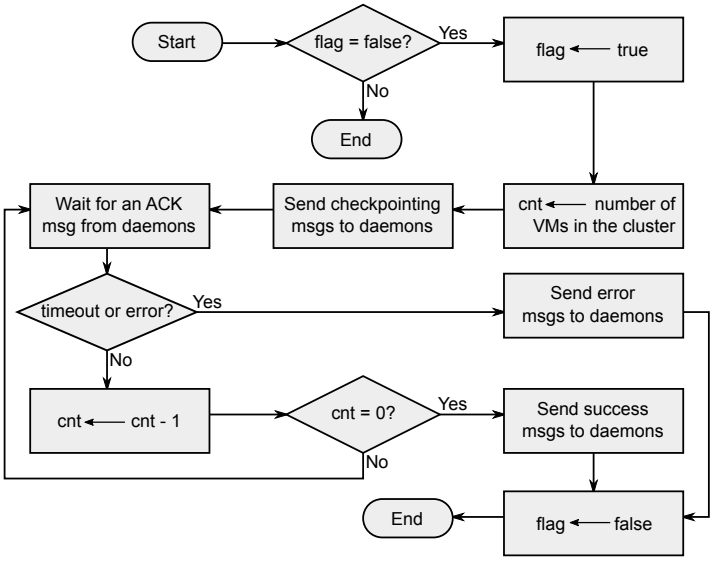


FIG. 4.3. *Flow chart displaying the checkpointing process steps taken by the coordinator component for a cluster*

only one checkpointing process is active for the same cluster at the same moment. To identify a cluster, the coordinator component relies on the unique identifier which was assigned to it during the clustering process.

The steps taken by the coordinator component during the checkpointing process are displayed in Fig. 4.3. During these, the coordinator first sends a control message to all daemon components corresponding to the VMs in the current cluster asking them to initiate the checkpointing process locally. Then the coordinator component waits for acknowledgement messages or error messages. When the coordinator receives the same number of acknowledgement messages as the number of VMs in the cluster, it can assert that all VMs in the cluster have been checkpointed and a success termination message is sent to all daemon components in the cluster. Otherwise, if the coordinator receives an error message, or if, after a given delay, the coordinator does not receive all acknowledgement messages, then an error is assumed and an error termination message is sent to all daemon components in the cluster.

The daemon component. It manages the checkpointing process at the host level. One or multiple VMs may be checkpointed at the same moment without interfering. To this extent, this component keeps a separate dataset per VM which includes the following:

- the cluster identifier of the VM;
- the corresponding backup host where the VM's checkpoint is saved;
- the communications' buffer, which is used to store outgoing communication packets before they are released to the OWP;
- the epoch value which is associated to the communications' buffer and is used to determine when packets are released to the OWP;
- the colour of the VM, which is necessary for the global snapshot algorithm to function;
- another communications' buffer, which is used during the checkpointing process when a white VM receives red communication packets.

By default, all outgoing communications of a VM to the OWP are buffered. Communications of VMs inside the same cluster are not suspended. However, the colour of the sending VM is piggybacked to each communication packet before it is transmitted to another VM in the same cluster. This can be achieved transparently through network virtualisation (e.g. using Violin [12]).

When receiving communication packets, the VM checks the colour of the sending VM. If the colour of the VM is white and the colour of the sending VM is red, then the received packet is buffered until the VM becomes red.

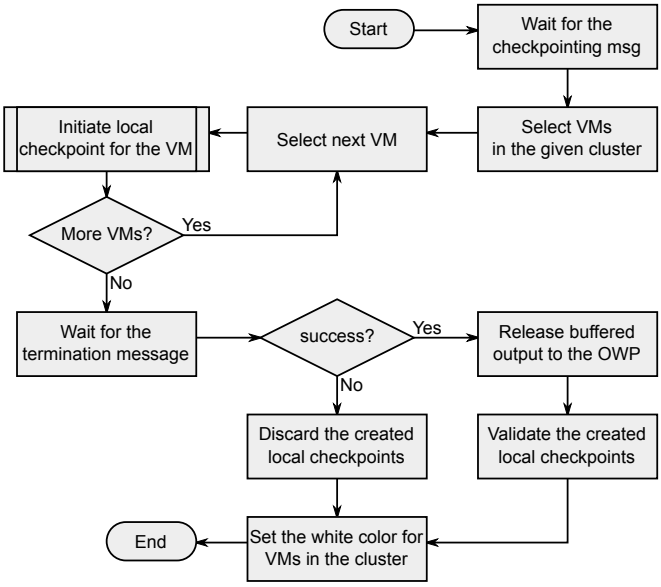


FIG. 4.4. *Flow chart displaying the checkpointing process steps taken by the daemon component for a cluster*

After that a checkpoint is initiated for a given cluster, the daemon component will concurrently create a local checkpoint for each VM in the current host which belongs to the cluster being checkpointed.

The steps which are globally taken (for all VMs in the current host and current cluster) by the daemon component are displayed in Fig. 4.4. They mainly consist on initiating the local checkpoint creation for all VMs in the current cluster, and to wait for the termination message from the coordinator component. If the daemon component receives a success termination message, then it validates all created checkpoints and, for each VM in the current cluster, it releases any buffered output since last epoch to the OWP. However, if an error termination message is received, then the local checkpointing process is aborted and all checkpoints resulting from the current process are discarded.

The steps taken for each VM by the daemon component are displayed in Fig. 4.5. First the daemon component pauses the VM, creates the checkpoint file and updates the colour and the epoch associated with the VM. Next, the VM is resumed and, because the VM is now red, any incoming red message which was previously buffered is released. Finally, the checkpoint file is transmitted to the backup host, and either an acknowledgement message or an error message is transmitted to the coordinator component.

4.5. Recovery process. The recovery process is managed by the coordinator component inside each cluster. This process is initiated after that a failure is detected by a daemon component, and that a request is formulated and sent to the coordinator component.

Once the recovery process is initiated for a given cluster, no further checkpointing is possible for the cluster until the recovery process is finished. However, if the coordinator process is already engaged in a checkpointing process while receiving a recovery request, then the checkpointing process is first aborted before initiating the recovery process. This has the same consequences as if an error was reported by a daemon component during the checkpointing process.

The steps taken by the coordinator component during the recovery process are the same as those taken during the checkpointing process (cf. Fig. 4.3) with two notable differences. Firstly, a recovery message is sent to the daemon components instead of a checkpointing message. Secondly, the recovery message is sent to the daemon components corresponding to the secondary VMs (i.e. backup VMs) in the current cluster, and not to the daemon components corresponding to the primary VMs.

Consequently, the recovery process is locally handled by the daemon component on the backup host. After receiving the recovery request, the daemon component on the backup host will initiate the recovery process for

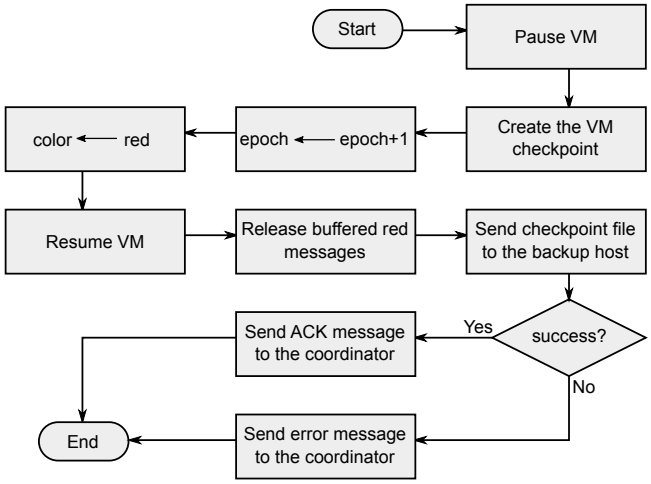


FIG. 4.5. *Flow chart displaying the checkpointing process steps taken by the daemon component for a single VM*

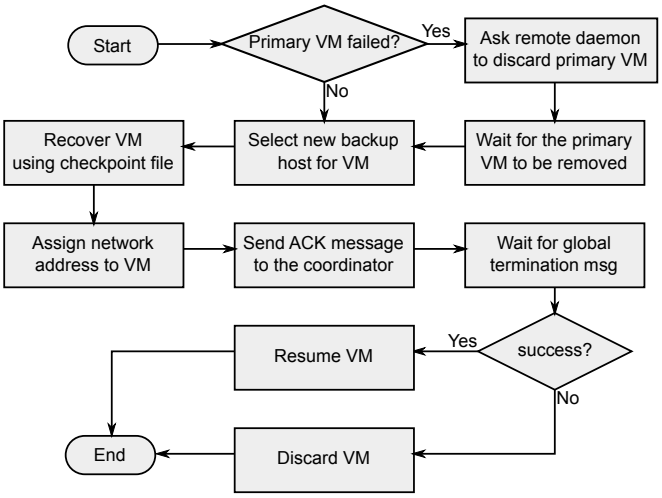


FIG. 4.6. *Flow chart displaying the steps taken by the daemon component when recovering a single VM*

all VMs in the current cluster for which it has a saved checkpoint. This process is illustrated in Fig. 4.6.

For each VM, the daemon component will use the backup host as its new primary host and another backup host is selected for the VM. This way, the checkpoint file is immediately available for recovery, and there is no need to transmit the checkpoint file through the network.

With regard to the old primary VM, if it has not failed, then a message is sent to the corresponding daemon component to discard the VM and any related state (i.e. communication buffers). The network address of the VM is then reattributed to the new VM that results from the recovery process.

After that the old primary VM is discarded, and after that the new VM has been recovered using the checkpoint file, an acknowledgement message is sent to the coordinator component. However, the recovered VM is not immediately started. This guarantees that the recovered VM is not affected by other running (and not yet recovered) VMs in the cluster, which, in turn, guarantees a consistent global state for the cluster after recovery.

The termination of the recovery process is signalled by the coordinator component when it receives an error message or when it receives an acknowledgement message for each VM in the cluster. After a successful termination, the execution of all recovered VMs is resumed and further checkpointing is possible. However, if an

error happens, then the cluster is put under an erroneous state and any previously recovered VM is discarded. The coordinator component can then initiate another recovery of the cluster if necessary.

5. Evaluation. In this section, we present the performance measurements of the two approaches presented in this paper. To this extent, we have simulated and compared both the presented approaches, and three other state-of-the-art approaches: one uncoordinated (Remus [6]) and two coordinated (VCCP [21] and VNSnap [13]).

The simulations were performed using the ACS [22] simulator. ACS is an open source discrete event simulator which enables Cloud Computing simulation and performance evaluation of different aspects of the Cloud. Our choice for ACS has been motivated by two points. First, it already implements many aspects of fault tolerance simulation, which greatly reduces implementation efforts. Secondly, it offers high performance simulations both in terms of memory consumption and simulation time.

During the simulations, we have focused on three different measures to evaluate the performances of a given approach. The first and the second measures are, respectively, the average communications' latency and the percentage of packets loss during communications. These two values are used to evaluate the communications' handling aspect of the simulated approaches. The third measure is the added execution time of submitted jobs relatively to their initial length. The extra-time is due to the checkpointing process and the failures. It is a good indicator of an approach's ability to introduce low overheads during the checkpointing process and the recovery process.

In the following, we first start by describing the different values that were used as simulation input. Next, we compare simulated approaches based on the previously described communication measures. Finally, we compare simulated approaches based on the added overhead on the jobs' execution time.

5.1. Simulation input. There are many simulation inputs that can interfere with the checkpointing and the recovery processes, and consequently, with a VM's execution. For each of these inputs, we need to cover the largest possible set of values during the simulations to draw trustworthy conclusions on the performances of a given approach.

However, the number of simulations per approach grows rapidly as the number of tested inputs grows. Therefore, to limit the number of simulations, it is necessary to limit the values tested for each input. To this extent, we use empirical data, when available, to set input values. For other inputs, we try to chose the most representative values.

5.1.1. Single-valued simulation inputs. The values of these inputs are set based on empirical data. We define one unique value for each of them and for all simulations. These inputs include: the average job length, the checkpointing overhead, the recovery overhead and the average networks links latency.

The job length is randomly generated based on a mean value of 100 hours. At first glance, this value may seem too high. However, it allows us to emphasise the efficiency of each approach. Besides, since all tested approaches use VM-level checkpointing, and considering the fact that a VM's lifetime can be relatively high, this allows us to simplify the simulations by assigning exactly one job per VM.

The checkpointing overhead is also randomly generated based on empirical data. It defines the checkpoint file creation delay (randomly chosen using a mean value of 200 milliseconds), the checkpoint file size (randomly chosen using a mean value of 30MiB) and the available bandwidth for transferring the checkpoint (set to 100MiB/s).

The recovery overhead defines the delay for recovering a VM using a checkpoint file. It is randomly chosen based on a mean value of 300 milliseconds. We assume that the host where the checkpoint file is located is the same host that is used when recovering a VM. This is the behaviour that has been defined by Remus and our approaches. Other tested approaches do not give explicit directives on this matter.

The network link latency is the average latency for message communications when no checkpointing approach is used. This value is randomly chosen for each communication based on a mean value of 10 milliseconds which is commonly observed in medium to large-scale data centres.

5.1.2. Multi-valued simulation inputs. Added to the previous inputs, we define five other inputs which take three different values each (cf. Table 5.1).

The cluster size (noted In_0) defines the number of VMs initially submitted when a simulation starts. For the VCCP and VNSnap approaches, this defines also the cluster size.

TABLE 5.1
Simulation input values

Input	Description	Values		
In_0	Cluster Size	30	80	200
In_1	Failures rate	low $mttf$=8000h	medium $mttf$=1500h	high $mttf$=500h
In_2	Checkpointing frequency	low $3/h$	medium $12/h$	high $60/h$
In_3	Communications' frequency	low $10/h$	medium $500/h$	high $3000/h$
In_4	AVG Interlocutors Percentage	5%	20%	60%

The failures rate (noted In_1) defines the number of failures generated per host. Each failure rate corresponds to a mean time to failure value (MTTF). This value is used during the simulations to randomly generate failures following an exponential distribution.

The checkpointing frequency (noted In_2) defines the average number of checkpoints taken for each VM in a given period of time. The values presented in Table 5.1 are the average values, and the final checkpointing frequency for each VM is randomly chosen given that average.

The communications frequency (noted In_3) defines the average number of communication messages sent by a VM during a given period of time. This is also an average value, and final values are randomly set during the simulation.

Finally, the average interlocutors percentage (noted In_4) is used to define the number of remote VMs that a given VM can communicate with during the simulation. This percentage is relative to the number of VMs initially submitted.

5.1.3. Other considerations on simulation input. The VCCP and the VNSnap approaches do not take into account communications to the OWP. Consequently, when using these approaches, we have assumed that all VMs belong to the same cluster and that no communications to the OWP are possible.

For our coordinated approach, we assumed that communication profiles are already available when the simulation starts. Consequently, the VMs are automatically grouped into different clusters during the initialisation of the simulation.

With regard to the number of simulations, it can be computed after generating all possible values combinations for simulation inputs. This results in a total of 1215 simulations, which corresponds to 243 simulations per approach.

During the simulations, we have assumed that there is always a sufficient number of machines for hosting the VMs and their backups. The VMs are initially placed using a random-fit placement policy given a fixed set of available hosts. Then, each time a new host is needed, the same policy is used again.

Regarding number generation, we use the same random seed for all simulations. Thus, given the same input, same random values are generated and we can guarantee the correctness of results when comparing two approaches.

5.2. Evaluation of communication latencies and packets loss. Simulation results regarding packets loss and communication latencies are summarised in Table 5.2. The mean and median values displayed in this table, have been computed for each approach after all simulations.

Over the 1215 simulations, the VCCP and VNSnap approaches were the most efficient in terms of communication latencies and packets loss. The reason for that, is that these approaches do not take into account communications to the OWP. Consequently, communication packets do need to be retained and no extra-latencies are induced.

The next most efficient approach is our coordinated approach. This approach introduced a relatively small network latency and a negligible rate of packets loss which is mostly due to the recovery process.

Our uncoordinated approach also caused relatively low communication latencies. However, they are higher than the latencies induced by our coordinated approach. This is because our coordinated approach does not need to buffer intra-cluster communications, and thus, does not introduce extra-latencies for these communications.

TABLE 5.2
Summarised simulations results for communication latencies and communication packets loss

Approach	Packets loss		Communication latency	
	Mean	Median	Mean	Median
Our coordinated approach	0.31%	0.00%	1.73s	1.76s
Our uncoordinated approach	0.00%	0.00%	2.23s	2.26s
Remus	91.20%	93.00%	2.46s	2.54s
VCCP	0.00%	0.00%	0.01s	0.01s
VNSnap	0.00%	0.00%	0.01s	0.01s

TABLE 5.3
Correlation coefficients of simulation input with communication packets loss

Approach	In_0	In_1	In_2	In_3	In_4
Our coordinated approach	0.04	0.00	-0.08	0.00	0.26
Our uncoordinated approach	0.00	0.00	0.00	0.00	0.00
Remus	-0.20	0.00	**-0.87**	0.00	0.00
VCCP	0.00	0.00	0.00	0.00	0.00
VNSnap	0.00	0.00	0.00	0.00	0.00

Remus induced the highest rate of packets loss and caused high messages latency. The packets loss was due to communication timeouts. This was predictable as Remus retains outgoing communications until a checkpoint is committed, and it does not implement any strategy to reduce communication latencies.

5.2.1. Correlation with simulation input. We have used the Pearson product-moment correlation coefficient (PPMCC) to measure the correlation of simulation input with the communications' packets loss rate, and with the communication latencies.

The PPMCC is used in statistics as a measure of linear correlation between two variables. It takes its values in the range $-1 \cdots +1$, where $+1$ indicates a perfect correlation, 0 indicates no correlation and -1 a total negative correlation.

The PPMCC values corresponding to the rate of communication packets loss are displayed in Table 5.3, and the PPMCC values corresponding to communication latencies are displayed in Table 5.4.

With regard to the rate of communication packets loss, we see no important correlation with any of the inputs when using an approach other than Remus.

For Remus, there is an non-negligible negative correlation of the rate of communication packets loss with the checkpointing frequency (In_2). This is due to the fact that Remus releases output for a VM only after a checkpoint is committed. Thus, a more frequent checkpointing induces lower output retention delays, which in turn induces lower communication latencies. Consequently, there is a lower rate of packets loss which is due to communication timeouts.

With regard to communication latencies, we see an important negative correlation with the checkpointing frequency (In_2) for Remus, and with the communications' frequency (In_3) for our approaches.

The correlation of communication latencies with the checkpointing frequency when using Remus has already been discussed, and is due to the fact that Remus releases network output only after a checkpoint is committed.

For our approaches, the correlation of communication latencies with the communications' frequency is due to the fact that the checkpointing frequency is in part determined by the communications' frequency. In fact, a communication-induced checkpoint is scheduled each time network output is buffered. Thus, by increasing the communications' frequency, we also increase the checkpointing frequency, and we reduce the retention delay of network output.

5.3. Evaluation of jobs' execution time. For each simulation, we have computed the average jobs' completion time for all VMs. This value was then used to compute the added execution time which is due to the checkpointing process and the failures, based on the average jobs' length when the simulation was initialised. Finally, we have estimated the percentage of added execution time for each simulation.

The results regarding the added execution time induced by each approach are summarised in Table 5.5. This table contains the mean and median values for each approach after running all simulations.

TABLE 5.4
Correlation coefficients of simulation input with communication latencies

Approach	In_0	In_1	In_2	In_3	In_4
Our coordinated approach	0.07	0.04	0.19	**-0.88**	0.12
Our uncoordinated approach	-0.16	0.00	-0.07	**-0.93**	0.00
Remus	0.03	0.02	**-0.84**	0.14	0.00
VCCP	0.00	0.00	0.00	0.00	0.00
VNSnap	0.00	0.00	0.00	0.00	0.00

TABLE 5.5
Summarised simulations results for added execution time

Approach	Added execution time	
	Mean	Median
Our coordinated approach	1.30%	1.25%
Our uncoordinated approach	1.51%	1.44%
Remus	0.22%	0.12%
VCCP	6.94%	6.15%
VNSnap	1.76%	1.77%

Over the 1215 simulations, Remus offered the best value for the average job completion time. This is due to two reasons. First, Remus does not take forced communication-induced checkpoints as we do in our approaches. Secondly, Remus does not assume any clustering for checkpointed VMs. Thus, it does not induce any extra-overhead to orchestrate the checkpointing or the recovery process inside a cluster. Moreover, failures are less costly. In fact, when using Remus, only the failed VM needs to be recovered. Conversely, when using a coordinated approach, all VMs belonging to the same cluster have to be recovered each time a VM in the cluster fails.

The second most efficient approach is our coordinated approach. This approach ensures that an appropriate clustering of VMs is done to minimise the execution time of VMs. However, the frequent communication-induced checkpoints causes extra-overhead which makes it less efficient than Remus when considering execution time.

Our uncoordinated approach was the next most efficient approach. This approach, as for Remus, does not induce extra-overhead to orchestrate the checkpointing or the recovery process inside a cluster.

VNSnap is the fourth most efficient approach. It performed worse than previous approaches because the checkpointing process needs to be orchestrated over all submitted VMs. Additionally, each time a failure happens, all VMs need to be recovered.

VCCP was the worse performing approach. Added to the disadvantages of VNSnap on the execution time of VMs, this approach also uses a blocking checkpointing process which highly burdens the VMs' execution.

5.3.1. Correlation with simulation input. As for packets loss and communications' latency, we have used the PPMCC values to estimate the correlation of execution time with simulation input. The coefficients are displayed in Table 5.6.

For our approaches, we see an important correlation of execution time with the communications' frequency (In_3). This correlation can be anticipated because the communications' frequency determines the number of communication-induced checkpoints taken when using our approaches, and a high number of checkpoints induces a high overhead on the execution time. Additionally, the low correlation with the checkpointing frequency (In_2). indicates that most of the checkpoints taken by our approaches are communication-induced.

For Remus, VCCP and VNSnap, there is a high correlation of execution time with the checkpointing frequency (In_2). This is predictable as more frequent checkpointing induces a higher overhead on the execution of the VMs.

Besides, we see a non-negligible correlation of execution time with the number of submitted VMs (In_0) when using VCCP and VNSnap. This can be explained by the fact that, the checkpointing and recovery processes are more costly as the cluster size is bigger.

6. Conclusion and Future Work. In this paper, we have presented two new fully transparent checkpointing approaches for Cloud Computing environments. Unlike many approaches in the literature, our approaches

TABLE 5.6
Correlation coefficients of simulation input with the added execution time

Approach	In_0	In_1	In_2	In_3	In_4
Our coordinated approach	0.06	0.00	0.17	**0.96**	0.06
Our uncoordinated approach	0.08	0.00	0.11	**0.98**	0.04
Remus	0.17	0.03	**0.82**	0.00	0.00
VCCP	**0.28**	0.00	**0.84**	0.00	0.00
VNSnap	**0.32**	0.01	**0.84**	0.00	0.00

strive to keep a consistent view of checkpointed VMs from the outside world process. This is achieved by using an output commit mechanism which buffers communications until a checkpoint is committed. Our main contribution in this matter was to define a process for taking forced communication-induced checkpoints to reduce the retention delay of buffered communications. Additionally, we also defined a non-blocking coordinated checkpointing approach, which automatically groups VMs into clusters to reduce inter-VM communication latencies.

The comparison of the two proposed approaches with state of the art approaches (Remus, VCCP and VNSnap) shows that our coordinated approach is the best performing approach when considering both the execution time of the VMs and the communication latencies. Our uncoordinated approach, is next best performing approach when considering those two metrics.

When considering the execution time, Remus was better. However, the packets loss rate is excessively important.

When considering communication latencies, VCCP and VNSnap were better. However, these two approaches do not take into account communications to the outside world process. Thus, they are unusable in a Cloud environment if the consistency of the VMs' execution towards the outside world process is an issue.

Finally, as future work, we aim to implement the two proposed approaches on top of the Xen hypervisor to evaluate their efficiency and feasibility on a real Cloud environment.

REFERENCES

[1] S. AGARWAL, R. GARG, M. S. GUPTA, AND J. E. MOREIRA, *Adaptive incremental checkpointing for massively parallel systems*, in Proceedings of the 18th annual international conference on Supercomputing, ACM, 2004, pp. 277–286.

[2] M. ARMBRUST, A. FOX, R. GRIFFITH, A. D. JOSEPH, R. KATZ, A. KONWINSKI, G. LEE, D. PATTERSON, A. RABKIN, I. STOICA, ET AL., *A view of cloud computing*, Communications of the ACM, 53 (2010), pp. 50–58.

[3] E. BALAS AND M. W. PADBERG, *Set partitioning: A survey*, SIAM review, 18 (1976), pp. 710–760.

[4] K. CHANCHIO, C. LEANGSUKSUN, H. ONG, V. RATANASAMOOT, AND A. SHAFI, *An efficient virtual machine checkpointing mechanism for hypervisor-based hpc systems*, in High Availability and Performance Computing Workshop, 2008.

[5] K. M. CHANDY AND L. LAMPORT, *Distributed snapshots: determining global states of distributed systems*, ACM Transactions on Computer Systems (TOCS), 3 (1985), pp. 63–75.

[6] B. CULLY, G. LEFEBVRE, D. MEYER, M. FEELEY, N. HUTCHINSON, AND A. WARFIELD, *Remus: High availability via asynchronous virtual machine replication*, in Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation, San Francisco, 2008, pp. 161–174.

[7] I. P. EGWUTUOHA, D. LEVY, B. SELIC, AND S. CHEN, *A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems*, The Journal of Supercomputing, 65 (2013), pp. 1302–1326.

[8] E. N. ELNOZAHY, L. ALVISI, Y.-M. WANG, AND D. B. JOHNSON, *A survey of rollback-recovery protocols in message-passing systems*, ACM Computing Surveys (CSUR), 34 (2002), pp. 375–408.

[9] K. B. FERREIRA, R. RIESEN, P. BRIDGES, D. ARNOLD, AND R. BRIGHTWELL, *Accelerating incremental checkpointing for extreme-scale computing*, Future Generation Computer Systems, 30 (2014), pp. 66–77.

[10] R. GARG, V. K. GARG, AND Y. SABHARWAL, *Scalable algorithms for global snapshots in distributed systems*, in Proceedings of the 20th annual international conference on Supercomputing, ACM, 2006, pp. 269–277.

[11] D. IBTESHAM, D. ARNOLD, K. B. FERREIRA, AND P. G. BRIDGES, *On the viability of checkpoint compression for extreme scale fault tolerance*, in Euro-Par 2011: Parallel Processing Workshops, Springer, 2012, pp. 302–311.

[12] X. JIANG AND D. XU, *Violin: Virtual internetworking on overlay infrastructure*, in Parallel and Distributed Processing and Applications, Springer, 2005, pp. 937–946.

[13] A. KANGARLOU, P. EUGSTER, AND D. XU, *Vnsnap: Taking snapshots of virtual networked infrastructures in the cloud*, Services Computing, IEEE Transactions on, 5 (2012), pp. 484–496.

[14] A. D. KSHEMKALYANI, M. RAYNAL, AND M. SINGHAL, *An introduction to snapshot algorithms in distributed computing*, Distributed systems engineering, 2 (1995), p. 224.

[15] A. Litvinova, C. Engelmann, and S. L. Scott, *A proactive fault tolerance framework for high-performance computing*, in Proceedings of the 9th IASTED International Conference, vol. 676, 2009, p. 105.
[16] F. Mattern, *Efficient algorithms for distributed snapshots and global virtual time approximation*, Journal of Parallel and Distributed Computing, 18 (1993), pp. 423–434.
[17] P. Mell and T. Grance, *The nist definition of cloud computing*, National Institute of Standards and Technology, 53 (2009), p. 50.
[18] B. Meroufel and G. Belalem, *Adaptive time-based coordinated checkpointing for cloud computing workflows*, Scalable Computing: Practice and Experience, 15 (2014).
[19] J. Nakano, P. Montesinos, K. Gharachorloo, and J. Torrellas, *Revivei/o: Efficient handling of i/o in highly-available rollback-recovery servers*, in High-Performance Computer Architecture, 2006. The Twelfth International Symposium on, IEEE, 2006, pp. 200–211.
[20] B. Nicolae and F. Cappello, *Blobcr: efficient checkpoint-restart for hpc applications on iaas clouds using virtual disk image snapshots*, in Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, ACM, 2011, p. 34.
[21] H. Ong, N. Saragol, K. Chanchio, and C. Leangsuksun, *Vccp: A transparent, coordinated checkpointing system for virtualization-based cluster computing*, in Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on, IEEE, 2009, pp. 1–10.
[22] S. Sadi and B. Yagoubi, *Acs-advanced cloud simulator: A discrete event based simulator for cloud computing environments*, in Proceedings of the 2nd International Conference on Networking and Advanced Systems, 2015, pp. 11–16.
[23] S. Sadi and B. Yagoubi, *On the optimum checkpointing interval selection for variable size checkpoint dumps*, in Computer Science and Its Applications, Springer, 2015, pp. 599–610.
[24] K. V. Vishwanath and N. Nagappan, *Characterizing cloud computing hardware reliability*, in Proceedings of the 1st ACM symposium on Cloud computing, ACM, 2010, pp. 193–204.
[25] L. Wang, Z. Kalbarczyk, R. K. Iyer, and A. Iyengar, *Vm-μcheckpoint: Design, modeling, and assessment of lightweight in-memory vm checkpointing*, Dependable and Secure Computing, IEEE Transactions on, 12 (2015), pp. 243–255.
[26] J. W. Young, *A first order approximation to the optimum checkpoint interval*, Communications of the ACM, 17 (1974), pp. 530–531.

Edited by: Dana Petcu
Received: Apr 17, 2016
Accepted: May 2, 2016

AIMS AND SCOPE

The area of scalable computing has matured and reached a point where new issues and trends require a professional forum. SCPE will provide this avenue by publishing original refereed papers that address the present as well as the future of parallel and distributed computing. The journal will focus on algorithm development, implementation and execution on real-world parallel architectures, and application of parallel and distributed computing to the solution of real-life problems. Of particular interest are:

Expressiveness:

- high level languages,
- object oriented techniques,
- compiler technology for parallel computing,
- implementation techniques and their efficiency.

System engineering:

- programming environments,
- debugging tools,
- software libraries.

Performance:

- performance measurement: metrics, evaluation, visualization,
- performance improvement: resource allocation and scheduling, I/O, network throughput.

Applications:

- database,
- control systems,
- embedded systems,
- fault tolerance,
- industrial and business,
- real-time,
- scientific computing,
- visualization.

Future:

- limitations of current approaches,
- engineering trends and their consequences,
- novel parallel architectures.

Taking into account the extremely rapid pace of changes in the field SCPE is committed to fast turnaround of papers and a short publication time of accepted papers.

INSTRUCTIONS FOR CONTRIBUTORS

Proposals of Special Issues should be submitted to the editor-in-chief.

The language of the journal is English. SCPE publishes three categories of papers: overview papers, research papers and short communications. Electronic submissions are preferred. Overview papers and short communications should be submitted to the editor-in-chief. Research papers should be submitted to the editor whose research interests match the subject of the paper most closely. The list of editors' research interests can be found at the journal WWW site (`http://www.scpe.org`). Each paper appropriate to the journal will be refereed by a minimum of two referees.

There is no a priori limit on the length of overview papers. Research papers should be limited to approximately 20 pages, while short communications should not exceed 5 pages. A 50–100 word abstract should be included.

Upon acceptance the authors will be asked to transfer copyright of the article to the publisher. The authors will be required to prepare the text in LaTeX 2ε using the journal document class file (based on the SIAM's `siamltex.clo` document class, available at the journal WWW site). Figures must be prepared in encapsulated PostScript and appropriately incorporated into the text. The bibliography should be formatted using the SIAM convention. Detailed instructions for the Authors are available on the SCPE WWW site at `http://www.scpe.org`.

Contributions are accepted for review on the understanding that the same work has not been published and that it is not being considered for publication elsewhere. Technical reports can be submitted. Substantially revised versions of papers published in not easily accessible conference proceedings can also be submitted. The editor-in-chief should be notified at the time of submission and the author is responsible for obtaining the necessary copyright releases for all copyrighted material.