Democratic and Popular Republic of Algeria

Ministry of Higher Education and Scientific Research

University of Relizane

Faculty of Science and Technology

Department of Computer Science

In Partial Fulfillment of the Requirements for the Master's Degree in Computer Science

Title

Formal Verification of a Security Model for a Clinical Information System with Alloy

Submitted by:

Mr. Tabchiche Bilel

Mr. Rouabah Mohamed El Amine

Supervised by:

Mr. Bouadjemi Abdelkrim

Board of Examiners:

Chairman: Mr. AMARA Sofiane

Supervisor: Mr. Bouadjemi Abdelkrim

Examiner: Mr. KEMMAR Omar

Academic Year 2024/2025

Acknowledgements

All praise and thanks be to Allah, by whose grace good deeds are accomplished, and through whose guidance success is achieved.

We extend our deepest gratitude to our beloved families, who have been an unwavering source of strength and support throughout every stage of our journey.

To our mothers, the embodiment of love and sacrifice, whose endless prayers and affection have carried us through challenges.
To our fathers, our role models and pillars of resilience, who instilled in us the values of dedication and perseverance.
To our dear siblings, we thank you for your constant encouragement, patience, and unconditional love.

We would also like to sincerely express our appreciation to our colleagues and friends, whose camaraderie and support have enriched this journey and made it truly meaningful.

Our profound gratitude goes to our esteemed supervisor, **Dr. Bouadjemi Abdelkrim**, whose invaluable guidance, wisdom, and dedication were instrumental in the completion of this work.

Finally, we extend our heartfelt thanks to all faculty members, administrative staff, technicians, and support personnel at the university. Your tireless efforts and commitment have fostered an environment conducive to learning and growth—for this, we are immensely grateful.

Dedication

To my beloved **mother**, whose unwavering love, endless sacrifices, and heartfelt prayers have been the foundation of my journey—this achievement is dedicated to you with all my love and gratitude.

To my dear **father, Abdelkrim**, whose guidance, strength, and constant encouragement have shaped my determination and perseverance—thank you for being my steadfast pillar of support.

To my cherished **siblings**, for their continued encouragement, patience, and belief in me—this moment of pride is shared with you all.

And to my loyal **friends Mohamed,Islem ,Bassel, Younes , Amine and yacine**,
Your sincere companionship and support throughout this academic journey have meant more than words can express. I am deeply grateful to have had you by my side.

Bilel

Dedication

I dedicate this work to my dear mother, whose unconditional love, endless patience, and continuous prayers have been my greatest source of strength.
To my dear father, whose wisdom, support, and unwavering faith in me have guided me every step of the way—this achievement is as much yours as it is mine.
To my brothers and sister, thank you for your encouragement and understanding, and for always standing by my side with unwavering support.
And my dear friends and colleagues,
Your companionship, encouragement, and the moments we shared throughout this journey were truly invaluable. I am sincerely grateful to each and every one of you, especially my friend and brother Ibrahim, who has been a supporter and pillar for me throughout these years. Without his grace, after Allah's will, I would not be here.
And finally, but not least, without Allah's grace and guidance, we wouldn't be here at all, and praise be to Allah forever and ever.

Amine


بسم الله الرحمن الرحيم

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

List of Figues

Table of Tables

List of Tables

Summary

This thesis focuses on the formal verification of a security model applied to a clinical information system using the Alloy language. In the context of the increasing digitalization of healthcare services, protecting sensitive medical data has become a top priority. The first chapter introduces the fundamentals of information systems, particularly in clinical settings, detailing their components, operation, and the security challenges they face. Special emphasis is placed on access control mechanisms, especially the Role-Based Access Control (RBAC) model, which manages user permissions based on their assigned roles. The second chapter presents formal modeling as a rigorous tool for analyzing and specifying system behavior through mathematical, precise, and verifiable methods. Alloy is highlighted as an effective language for modeling and analyzing such systems, thanks to its declarative syntax and its automated analysis tool, the Alloy Analyzer. A comparison of access control models—DAC, MAC, ABAC, and RBAC—allows for an evaluation of their strengths and weaknesses depending on security requirements. The final chapter demonstrates how to model a clinical system's class diagram using Alloy through signatures, facts, predicates, and assertions. The execution results from the Alloy Analyzer validate the system's consistency and help identify potential security breaches. In conclusion, formal verification using Alloy proves to be an effective approach to enhancing the security, confidentiality, and reliability of information systems in the healthcare sector.

ملخص

تتناول هذه الرسالةُ التحقُّقَ الشكليَّ من نموذجٍ أمنيٍّ مُطبَّق على نظامِ معلوماتٍ سريريٍّ باستخدامِ لغة Alloy ومع التوسُّعِ المتزايدِ في رقمنة خدمات الرعاية الصحيّة، أصبحت حمايةُ البياناتِ الطبيةِ الحسّاسة أولويةً قصوى. يستعرضُ الفصلُ الأولُ أُسُسَ نظم المعلومات، ولا سيّما في البيئات السريرية، مُفصّلًا مكوّناتها وآليةَ عملها والتحدّياتِ الأمنية التي تواجهها. ويُسلَّط الضوءُ بوجهٍ خاص على آليات التحكّم في الوصول، وبالأخصّ نموذج التحكّم القائم على الدور (RBAC) الذي يدير صلاحيات المستخدمين بناءً على الأدوار المسندة إليهم. أما الفصلُ الثاني فيُقدّم النمذجةَ الشكلية كأداةٍ صارمة لتحليل سلوك النظام ووصفه رياضيًّا على نحوٍ دقيقٍ قابلٍ للتحقق. وتبرز لغة Alloy كلغةٍ فعّالةٍ لنمذجة هذه الأنظمة وتحليلها بفضل نحوها التصريحي وأداة التحليل الآلي .Alloy Analyzer ويتضمّن الفصل مقارنةً بين نماذج التحكّم في الوصول DAC—و MACو ABACو—RBAC لتقييم نقاط قوّتها وضعفها تبعًا لمتطلّبات الأمان. في الفصل الأخير، تُعرَض كيفيةُ نمذجة مخطط الفئات لنظامٍ سريريٍّ في Alloy عبر التواقيع والحقائق والمُحمَّلات والادعاءات. وتُثبت نتائجُ التنفيذ في Alloy Analyzer اتّساقَ النظام وتساعد في كشف الثغرات الأمنية المحتملة. ختامًا، يبرهن التحقّق الشكلي باستخدام Alloy على فعاليته في تعزيز أمان وخصوصية وموثوقية نظم المعلومات في قطاع الرعاية الصحية.

General Introduction

General Introduction

As the healthcare sector continues to advance toward digitalization, the need to safeguard and ensure the privacy of medical data has become increasingly critical. The healthcare system depends on a continuous flow of sensitive information among patients, healthcare professionals, and medical institutions, making access management a fundamental component in maintaining both confidentiality and security.

Achieving security and privacy in medical systems necessitates the implementation of effective authorization management strategies. This includes the development of policies and procedures to regulate access to medical data, determine user access rights, and define how the data should be handled.

In the healthcare domain, the sensitivity of data ranges from personal health records to treatment and diagnostic information. Therefore, appropriate access control policies must be enforced to regulate access to such data, while also providing mechanisms that ensure only authorized individuals can access it.

By utilizing appropriate technologies—such as identity and access management systems and encryption techniques—the security and privacy of medical systems can be significantly strengthened. When these policies and technologies are properly implemented, a secure and reliable environment for medical data exchange can be established, thereby contributing to more effective and efficient healthcare delivery.

Chapiter 01

the information System in a Clinic,

Introduction

In this chapter, we will explain the concept of an Information System, which is based on the notions of 'system' and 'information'. A company needs information to plan, organize, and allocate resources to ensure the smooth functioning of its activities. For a decision support system to be effective, it must provide access to an increasingly rich informational base. Consequently, a company can only survive if its Information System offers a sufficient set of information that is available, relevant, reliable, precise, and up-to-date, enabling it to act and make decisions efficiently, that is, making the right decisions at the right time.

I.Information system

1. Definition of information system

An information system is a structured set of resources (hardware, software, data, procedures, human resources, etc.) designed to acquire, process, store, transmit, and make information (in the form of data, text, sound, images, etc.) available within and between organizations.

2. Importance of Information System

2.1Data acquisition and entry

Today, information is often processed by computers. In this context, the information must be in a form that is acceptable to machines.

II. Data processing

This involves transforming raw (primary) data into results through operations such as calculations, selection, grouping, formatting, etc.

II.4 Data storage

This refers to storing both raw (primary) and final data in formats that are usable and easily retrievable without errors.

II.4 Data communication

This involves transmitting data to other users.

3. Components of System Information

3.1 Hardware Resources

Physical devices such as servers, computers, networking equipment, and storage devices. Example: A clinic uses servers to store patient data and computers for doctors to access medical records.

3.2 Software Resources
Applications and systems that process data. Examples include Electronic Health Record (HER) software, database management systems (DBMS), and operating systems.

3.3 Data

Raw facts and figures that are processed into meaningful information. In a clinic, data includes patient records, treatment plans, and lab results.

3.4 Procedures

Rules and processes for managing and using the system. Example: Protocols for entering patient data or accessing medical records.

3.5 Human Resources
People who interact with the system, such as IT staff, doctors, nurses, and administrative personnel.[1]

4. Challenges and Issues in Information Systems

4.1 Data Security and Privacy
Protecting sensitive information from breaches. Example: A clinic must ensure patient data is secure to comply with regulations like GDPR or HIPAA.

4.2 Interoperability
Ensuring different systems can work together. Example: A clinic's SIC must integrate with lab systems and pharmacy software.

4.3 Cost Management
Balancing the costs of implementing and maintaining IS with the benefits. Example: A clinic must budget for hardware, software, and IT staff.

4.4 Technological Evolution:
Keeping up with advancements like cloud computing, AI, and big data. Example: A clinic might adopt AI tools to analyze patient data for early disease detection.

5. Types of Information Systems

5.1 Database Management Systems (DBMS)
Software for managing databases. Example: MySQL or Oracle used in clinics to store patient records.

5.2 Management Information Systems (MIS)
Systms that provide reports and insights for decision-making. Example: A clinic uses MIS to track patient admissions and discharges.

5.3 Decision Support Systems (DSS)
Tools that help managers make decisions. Example: A DSS in a clinic might analyze treatment outcomes to recommend the best practices.

5.4 Hospital Information Systems (HIS) or SIC
Centralized systems used in healthcare settings. Example: The SIC in a clinic stores and manages all medical data, enabling seamless coordination between departments.

6. Aspects of an Information System
The IS has two aspects:

6.1_Static aspect (or data aspect)
- Information base.
- Data model (or data structure).

6.2_Dynamic aspect (or processing aspect)
- Flow of information between different actors.
- Chronological and causal evolution of the information[1]

7.Building an Information System
How to create a “good” information system?

Several important steps are essential to building an information system. First, the needs of the organization must be analyzed by identifying the data and processes needed. Next, the system design phase defines the architecture, database, and information flow. The development phase involves programming and testing the system to ensure that it functions properly. Deployment then involves installation and user training of the system. Finally, a maintenance phase is required to correct errors and ensure that the system evolves according to the needs of the organization**.** (2)

II.The information System in a Clinic

1. Definition
The Information and Communication System (Information System in clinic) is the central information platform of a clinic. It stores relevant medical treatment data. The hospital reception staff, inpatient care services, as well as doctors and other specialized departments are the users of the SIC. [4]

2. Specialized Clinic Focused on SI (Information System) in Medical or Healthcare Contexts
It could also refer to a clinic offering consultations and services related to medical information systems or healthcare IT.

For Example, a clinic may provide:

2.1 Consulting services for CIS implementation

Advising hospitals and clinics on how to integrate clinical information systems.

2.2 Training and support

Educating medical staff on using information systems to improve patient care[3]

3. Components of a Clinical Information System (CIS)

3.1 Hardware

The hardware components of a CIS include servers, workstations, and medical devices. These devices help store and process patient data, and ensure real-time access to critical information.

3.2 Software

Software forms the core of the CIS, providing tools for data management, decision support, and communication.

3.2.1 Electronic Health Records (EHR)

A software used to store patient data, enabling doctors to access medical histories, prescriptions, and lab results (Buntin et al., 2011).

3.2.2 Clinical Decision Support Systems (CDSS)

These systems help healthcare providers by offering evidence-based suggestions and reminders, improving decision-making (Hansen et al., 2010).

3.3 People

The people component involves healthcare professionals (doctors, nurses), IT staff, and administrative personnel. They interact with the system to input, access, and analyze patient data.

3.4 Data

Data is the central element in a CIS, consisting of clinical data (patient histories, test results), administrative data (appointments, billing), and medical device data (real-time patient monitoring data).

3.5 Security

Security ensures that patient data is protected from unauthorized access and breaches. CISs employ data encryption, access controls, and audit logs to maintain confidentiality.

3.5.1 Security and Confidentiality

The security and privacy of medical data is critical in a Clinical Information System.

3.5.2 Data Encryption

Encryption protocols are used to protect sensitive data during transmission and while stored in the system.

3.5.3 Access Control

The system ensures that only authorized personnel can access specific patient data, based on their role (e.g., doctor, nurse, administrator).

3.5.4 Auditing and Monitoring

Tools are in place to track and monitor system usage to detect unauthorized access attempts and ensure compliance with regulations like HIPAA (Health Insurance Portability and Accountability Act).[5]

4. Requirements Formulated for the Information System (IS)

4.1 The Rights of Concerned Individuals

The right to data protection is fundamental for the defense of personality rights and the private sphere. It obligates those who handle data to respect the principles of legality and proportionality, and it provides concerned individuals with rights they can directly invoke, which must therefore be secured through the functions of the Health Information System (HIS). The following functions must be integrated into the HIS:

4.1.1 The right to access data

All information about the patient (including personal and health-related data) processed within the scope of treatment should be adequately exportable. This also applies to access and data modification protocols.

4.1.2 The right to rectification and erasure

Data related to a treatment must be erasable or amendable (including erasure and rectification mandates in the referenced systems). Subject to the obligation to offer data to the relevant archives, the data pertaining to a treatment must be erased or anonymized upon the expiration of the retention period.

4.2 Concept of Authorization

4.2.1 Access to User Account Administration

Access to the administration of user accounts should be capable of being limited. The creation of accounts and the management of access authorizations must only be possible through predefined roles.

4.2.2 Locking of User Accounts

It must be possible to lock user accounts for a set period of time. User accounts should be locked after a certain number of failed login attempts using incorrect passwords. Additionally, the system should be able to automatically recognize accounts that have not been used for an extended period and lock them. Moreover, sessions should be automatically disabled after a certain period of inactivity.

4.2.3 Access Authorizations

Access authorizations should be assigned based on roles and functions (e.g., according to the user's role and organizational unit). A user's access should be determined solely by the structural and functional roles they occupy. Within a defined period (removal and retention time), the system must automatically restrict access to data relating to a specific process.
In the event of emergency access to data (when access authorizations are temporarily extended), the reason, date, and time of access, as well as the user's role and identity, must be recorded.
When processing concerns a current or former employee of the institution, specific restrictions on access must be implemented.

4.2.4 User Authentication / Access Data

Access to the system must only be allowed through reinforced user authentication, meaning users must authenticate themselves based on at least two criteria.
Passwords must meet current technical requirements (e.g., visibility and reproduction during entry, minimum complexity requirements [length, special characters, numbers], time-limited validity, etc.).

4.3 Interfaces

With the integration of the Health Information System (HIS) into hospital informatics, it becomes necessary to network the databases through the HIS interfaces. To support and maintain the HIS infrastructure, as well as the applications provided by external service providers, interfaces at both the data and technical levels must be created.

For these interfaces, the requirements related to IT security and data protection (ISDP) are very high, especially due to the classification of patient medical data. Therefore, it is essential to draft an ISDP concept that meets the following conditions:

4.3.1 Data Exchange

The data exchange must be clearly defined, including the source, target, affected data, purpose (e.g., aggregation in the target system), and legal basis.

4.3.2 Interface Surveillance

The surveillance of the interface must be described, including login, transfer, and access protocols.

4.3.3 Technical Implementation

The technical implementation of the interface must be outlined, specifying the procedure call, push mechanisms, etc.

4.3.4 Responsibilities
Responsibilities must be clearly defined, including the data controller in the systems that export data and in the systems that import it.

4.3.5 Communication:
The communication process must be specified, including support, coding, and the overall communication process.

4.3.6 User Roles and Rights:
User roles and their rights in the systems that import data must be reviewed and defined.

4.3.7 Data Erasure:
The requirements for the complete erasure of data in all systems must be outlined.

4.3.8 Retention and Erasure Periods:
Retention periods and data erasure times must comply with legal requirements and be respected across all systems.

4.3.9 Data Seed for Rollback
A data seed must be defined for cases of “rollback” or “restore.”

4.3.10 Technical Requirements
All technical requirements must be included in the ISDP concept.

4.3.11 External Access
External access to the data must be encrypted and protected by enhanced authentication methods.

4.3.12 Anonymization and Pseudonymization
Data exported from the HIS must be anonymized or pseudonymized before being used in non-personal processing contexts. For regular or repetitive non-personal processing (e.g., statistics), the HIS should provide a function that allows for the export of data already anonymized or pseudonymized.

4.4 Logging and Access Control

In the field of IT, logging refers to the creation of manual or automated records to answer the following question: "Who, at what time, and by what means processed what data, or respectively, had access to what data?"

Every data processing event must be logged. This includes, but is not limited to, the following activities :

- Reading, modifying, and deleting data.
- System administration activities (e.g., logging all maintenance activities of the Information System [IS]).
- Data export.

- (Re-) activation of blocked data, for example, in medical emergencies.
- Logging failed data access attempts (with automatic notification in case of repeated occurrences).

Protocols contain sensitive information and must be protected accordingly. Only authorized individuals should have access to them.

Protocols must be destroyed once the defined retention periods have expired. The retention period must be determined by the responsible parties and should be based on the necessary duration planned for the data processing.

Protocols should be pseudonymized and used without linking them to the individuals involved.

III.Electronic Health Records (EHRs)

An **Electronic Health Record (EHR)** is a digital version of a patient's paper chart. EHRs are real-time, patient-centered records that make information available instantly and securely to authorized users. They are an integral part of modern healthcare systems, allowing for the efficient management and exchange of patient health information across different healthcare providers and settings.[6]

1. Key Features of EHRs

1.1 Patient Information

EHRs store detailed patient data, including personal information, medical history, diagnoses, medications, immunization status, allergies, and laboratory results.

This information is typically updated and accessed by healthcare providers in real-time to ensure accurate decision-making.

1.2 Centralized Access

Unlike paper-based records, EHRs are stored digitally and can be accessed securely by authorized healthcare professionals from multiple locations (hospitals, clinics, and specialists).

This centralized system allows for better coordination of care, as different doctors, specialists, and healthcare facilities can share up-to-date patient information.

1.3 Integration with Other Systems:

EHRs are often integrated with other health information systems such as Laboratory Information Systems (LIS), Radiology Information Systems (RIS), and Computerized Physician Order Entry (CPOE). This integration allows seamless sharing of lab results, imaging data, and prescription information within the clinical workflow.

1.4 Decision Support:

Many EHR systems come with built-in Clinical Decision Support Systems (CDSS) that provide alerts or reminders based on patient data. These tools help healthcare providers make more informed decisions, reduce errors, and improve patient safety.

1.5 Improved Communication:

With EHRs, communication between healthcare providers is more efficient. Doctors, nurses, and other professionals can easily share patient information, leading to better coordination and reducing the chances of miscommunication. [7]

2. Benefits of EHR

2.1 Improved Patient Care

By having access to comprehensive, up-to-date patient records, healthcare providers can make more accurate diagnoses and treatment decisions, which ultimately improves patient outcomes.

EHRs allow for better management of chronic conditions by keeping detailed records of medications, test results, and follow-up care.

2.2 Reduced Medical Errors

EHRs reduce errors caused by illegible handwriting, missing documentation, or outdated information. Automated alerts for drug interactions, allergies, and lab results help prevent adverse events.

2.3 Increased Efficiency

EHRs streamline administrative tasks, such as appointment scheduling, billing, and patient registration. This leads to faster processing, reduced wait times, and enhanced productivity for healthcare providers.

2.4 Cost Savings

The use of EHRs can lead to cost savings by reducing the need for paper records, storage space, and administrative overhead. Additionally, better clinical outcomes often translate into lower healthcare costs in the long terme

3. Challenges of EHRs

3.1 Data Security and Privacy

Protecting sensitive patient information from breaches and unauthorized access is a major concern. Example: A clinic must comply with regulations like HIPAA to ensure data security.

3.2 Interoperability Issues

Not all EHR systems can communicate with each other, leading to fragmented care. Example: A patient's records from one hospital may not be accessible at another facility.

3.3 High Implementation Costs

Setting up and maintaining EHR systems can be expensive for healthcare organizations. Example: A small clinic may struggle to afford the necessary hardware and software.

3.4 User Training

Healthcare providers and staff need training to use EHR systems effectively. Example: Doctors may resist adopting EHRs if they find the system difficult to use.

3.5 Data Entry Burden

Entering data into EHRs can be time-consuming for healthcare providers.

Example: A doc

tor may spend more time typing notes than interacting with patients.

4. EHRs in Practice: Example of a Clinic

In a clinic, the EHR system serves as the backbone of the SIC (Système d'Information Clinique). It integrates data from various departments, such as reception, laboratory, and pharmacy, to provide a unified view of patient care.

4.1 Reception Staff

Use the EHR to register patients and schedule appointments.

4.2 Doctors

Access patient records to review medical history and prescribe treatments.

4.3 Laboratory Technicians

Upload test results directly into the EHR.

4.4 Patients

Use a portal to view their records and communicate with providers.

5. Future of EHRs

5.1 Artificial Intelligence (AI)

AI can analyze EHR data to predict health outcomes and recommend treatments. Example: An AI tool might identify patients at risk of diabetes based on their records.

5.2 Blockchain Technology

Blockchain can enhance the security and interoperability of EHRs. Example: A patient's records could be securely shared across multiple providers using blockchain.

5.3 Telehealth Integration

EHRs are increasingly being integrated with telehealth platforms to support remote care. Example: A doctor can access a patient's EHR during a virtual consultation.

5.4 Patient-Centered Innovations:

Future EHRs may include more features to engage patients, such as wearable device integration. Example: Data from a patient's fitness tracker could be automatically added to their EHR.[8]

IIII.ACCESS CONTROL

1. Définition

An access control system is a set of components and methods Determine whether legitimate users are participating properly in activities Preconfigured access rights and privileges defined in the security policy Access security.

The system can be used to identify users or programs Authorization to view or use resources in the computing environment.

Two main types of access control: physical access control and logical access control Physical access controls can limit access to campus, buildings, rooms, and computer equipment. Logical access control restricts connections to the network Computer, system files, and data.[9]

2. Access Control Objectives

The objectives of access control are:

- Manage and control access to information resources

person or device.

- Detect unauthorized access.
- Specify the rules to follow for identification, authentication and verification
- Authorization of access to people or equipment.
- Ensure the availability of information by reducing :

✓ Denial of service attack

✓ Propagation of malicious code between computer systems

✓ Application operating or configuration errors

3. Access Control Policy

Constraints that control access to system resources can be

Static or dynamic in nature. We therefore distinguish two types of policies

3.1 Static Access Control Policy

For a given computer system, a static access control policy is Its special feature is that its state does not change with the dynamic evolution of the system.

system since it contains only static constraints. This can be updated Reflect various changes in the organization Assignment within an organization results in an increase in its privileges .

Actions initiated by the user trigger the evaluation of the state policy. Based on the authorization granted by the contract in the current state, If the operation is allowed. In most implementations, the policy Static access control associates users of the system with their authorizations

3.2 Dynamic Access Control Policy

For a given computer system, a dynamic access control policy There are several states regarding the evolution of the system. Authorization Actions are performed based on an assessment of the current state of the system and

Definition of the policy. The current state is updated each time a task is performed. controlled action. The basic version of this type of access control policy uses A history of operations performed by a computer system, updated by the computer system Responsible for implementing the policies. A formal language that supports traces occurrence of events, such as languages based on procedural algebra, which are well suited for Expression of dynamic constraints [11]

4. Access Control Model

There are many models due to performance differences

Requirements of enterprise security policy, two types of policies

Different models have been developed, two different access control models Discretionary :

Classical access control model:

- Discretionary Access Control (DAC) model.
- Mandatory Access Control (MAC) model.
- .Attribute-Based Access Control (ABAC)
- Role-based access control (RBAC) model.

4.1 Discretionary Access Control DAC

DAC for short, Discretionary Access Control Policy is Represented as a series of triples (subject, action, object). Each triple (S,A,O) This means "Subject Sa has the right to perform

action A on object O". This pattern allows. Associate the identity of a principal with a set of permissions on an object. when a subject

Make a request for an object and the operation will be granted if and only if the permission allows it [12]

4.1.1 DAC Policy Basics

The key aspect of a Discretionary Access Control (DAC) policy is its discretionary approach, which is defined by how access permissions are assigned:

A user can choose to grant access rights to different entities for the resources they control. However, they cannot assign permissions they don't have themselves [15].

The individual who owns a resource has complete freedom in determining which other entities can act upon that resource (e.g., read, write, run).

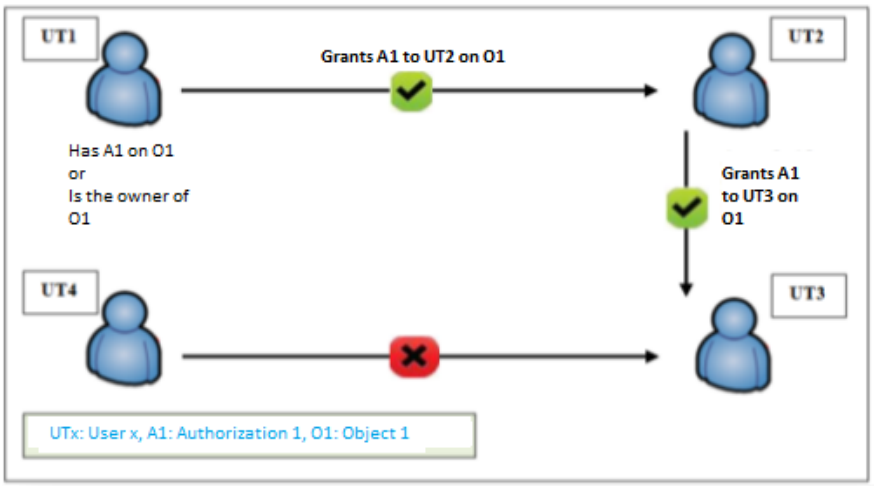

Figure 1.1 Example of a DAC Model (Discretionary Access Control)

4.2 Mandatory MAC Access Control Model

The mandatory MAC control model is implemented to provide Solutions to computer security problems and correction of weaknesses in the Digital-to-analog converter model. The MAC policy is based on delegating access control to independent entities. The existence of this independent entity ensures that the security policy will not be Users of computer systems cannot directly modify [13]

4.2.1 MAC models have two primary approaches

The initial model, called the Bell-LaPadula model, originated for the U.S. Department of Defense. Its chief goal is maintaining data confidentiality within shared mainframe environments.[19]

The second model, called the Biba model, centers on commercial needs and puts primary emphasis on ensuring information integrity.[18]

4.2.2 MAC Policy Explained

In MAC security models, everything (subjects and objects) gets a security level. An object's security level is its classification; a subject's is its clearance.

The security policy is mandatory; it's always in effect and can't be changed. If we're aiming for data confidentiality, the mandatory policy (also known as multi-level security) states:

"Users cannot see data with a classification higher than their clearance, but they can see data classified at their level or lower."[12]

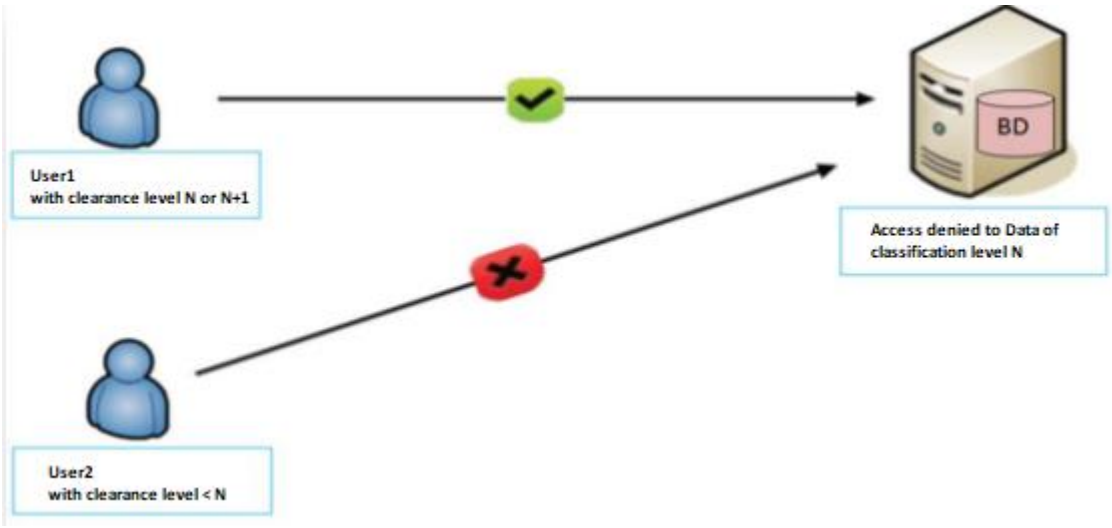

figure 1.2: Example of a MAC Model

4.3 Attribute-Based Access Control (ABAC)

The ABAC (Attribute-Based Access Control) model is based on a set of attributes linked with a claimant or a resource, which are consulted to make access decisions. The attributes may or may not be interrelated.

After defining the attributes used in the system, each attribute is regarded as a discrete value. The values of all the attributes are then compared to a group of values by a Policy Decision Point (PDP) to either grant or deny access.

These types of models are also known as:

Policy-Based Access Control (PBAC)

Claim-Based Access Control (CBAC)

Furthermore, a subject does not need to be previously known by the system. It simply has to authenticate itself and supply its attributes.

However, agreeing on the type and number of attributes to be used in access decisions is a complex task, especially in Cloud Computing environments. This model has not yet been implemented for well-known operating systems.

Finally, it is essential to propose a **security policy** that functions properly with this type of access control model. The security policy is responsible for selecting the relevant attributes used to make access decisions.

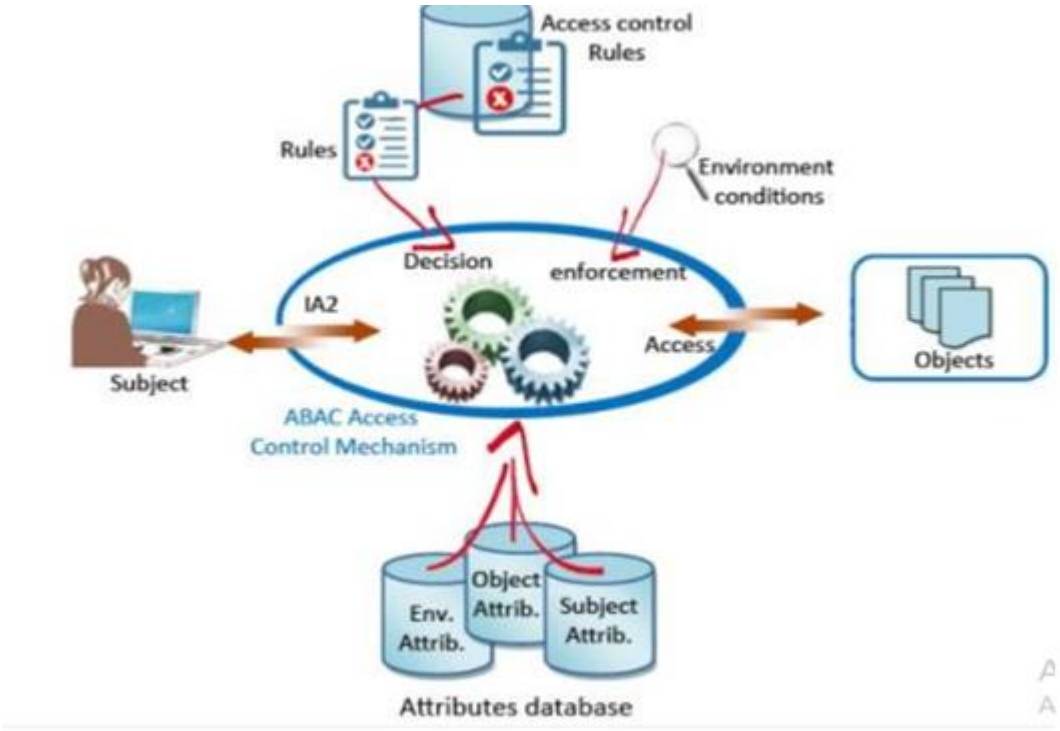

Figure 1.3 : Example of ABAC model

Example: Rule – A user whose role is "Doctor" or "Nurse" of type "Clinic" and who has passed the "HIPAA PRP Rules" certification may read and add notes to a "Patient" type record within the same "GMF" where the "GMF Consent" attribute is set to "Yes".

4.3.1 ABAC Characteristics

The ABAC model is known for several characteristics, including: [51]

Bruno Guay, « GIA et sécurité des applications», Symposium GIA ,1 mars 2017.

- Greater flexibility in decision-making
- Less effort needed to establish static entity-role and role-permission links
- Increased workload at decision time (performance impact)
- Better access control under changing conditions
- More difficult to trace the justification of a decision made at a specific time

- Strongly recommended to document rules in business language
- More difficult to enforce centralized enterprise policies (PBAC)
- Attributes must be associated with objects (metadata)
- Attribute management can be delegated

4.3.2 ABAC Applications

ABAC (Attribute-Based Access Control) can be applied across various levels of an enterprise technology stack, including firewalls, servers, applications, databases, and data layers. It uses attributes to provide context for access decisions.

Key application areas include:

1. **Application Security**
   ABAC policies are technology-neutral and can be reused across APIs, databases, and applications like content management systems, in-house software, and web apps.
2. **Database Security**
   ABAC enables fine-grained, vendor-independent database security such as dynamic data masking, allowing policies like restricting managers to view transactions only within their regions.
3. **Big Data Security**
   ABAC is applicable to Big Data platforms (e.g., Hadoop) to control data lake access with similar attribute-based policies.
4. **File Server Security**
   Windows Server uses ABAC principles through Dynamic Access Control Lists and Security Descriptor Definition Language (SDDL) to control file and folder access based on user and resource metadata.
5. **API and Microservices Security**
   ABAC enforces detailed attribute-based authorization for API endpoints and microservices functions.[52]

4.3.3. Comparison Table : Advantages and Disadvantages

Advantages	Disadvantages
More flexibility in decision-making	More difficult to trace the justification for a decision made at a specific time
Less work to establish static entity-role and role-permission links	Requires associating attributes with objects (metadata)
Better access control in changing conditions	Difficult to perform a pre-event audit and determine the permissions available to a specific user

Table1.1: The advantages and disadvantages of ABAC

4.4 Role-Based Access Control RBAC

The cost of maintaining access control can quickly become prohibitive.

The model mentioned above. Especially in industrial systems, it is possible to access Objects can be completed by hundreds or thousands of agents. Managing these authorizations Causes a combinatorial explosion, slows down the evaluation of access control and makes Granting or revoking rights complex. In addition, the rigidity of the MAC model and The DAC model lacks control and is not really satisfactory for the system In industry, delegation must be able to be delegated, but it must also be controlled. it is in response to these problems, role-based access control, namely role-based access control (RBAC), has emerged. [15]

4.4.1 The principle of the RBAC strategy:

A role-based access control model RBAC is proposed and presented.

The new organization of rights is centered on the concept of roles. A role represents a Function within the organization. Use an intermediary role between the ordering parties Authorizations facilitate and simplify administrative tasks by reducing their number Process works.[14]

illustrates the RBAC mechanism: the subject benefits from the following authorizations Resources are assigned to the roles assigned to them and are themselves associated with a set of authorizations. Since roles are generally much fewer in number than users, and resources, which greatly simplifies the management of authorizations [15]

4.4.2 The three main rules of RBAC

- **Role Assignment:**
- A user can exercise privileges only if a role has been assigned to them.
- **Role Authorization**
- A user's role must be authorized, ensuring that users can only assume roles for which they are permitted.
- **Privilege Authorization**

- A user can exercise certain privileges only if authorized, based on the role assigned to them and its permissions.[60]

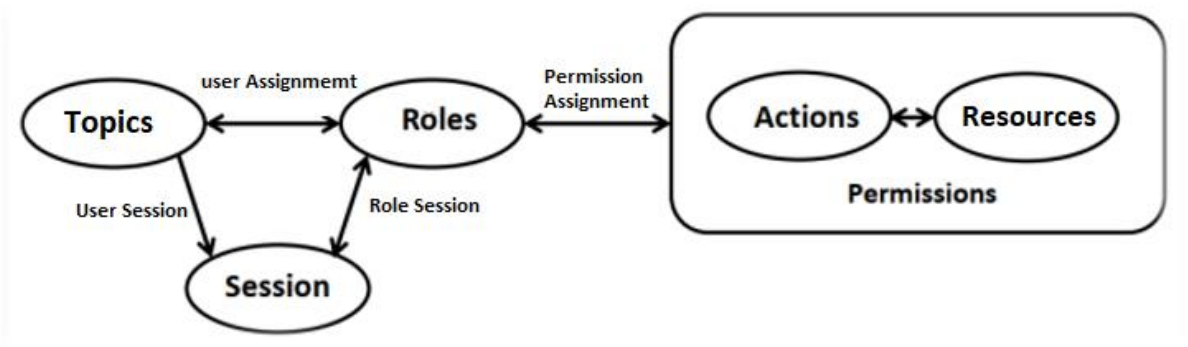

Figure 1.4 : RBAC Model (role-based access control)

Example of RBAC in a Medical System

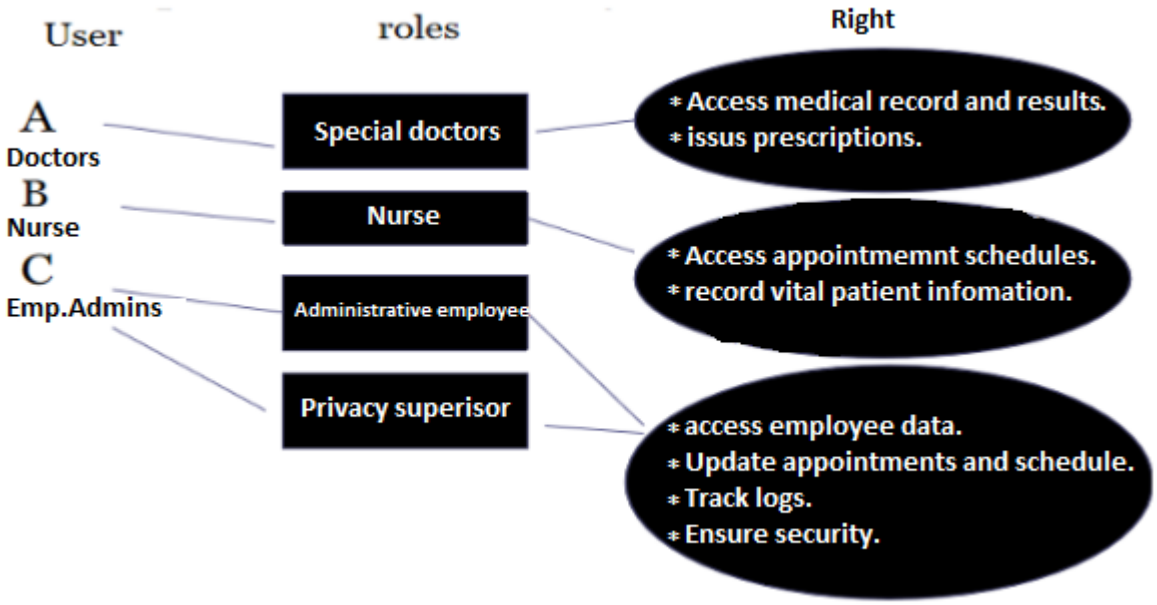

Figure1.5 : Example of RBAC in a Medical System

La figure 1.5 Show a diagram explaining the functioning of a Role-Based Access Control (RBAC) system. This diagram is divided into three main parts. The first part presents the system users, such as doctors, nurses, and administrators. The second part describes the roles assigned to these users, for example, the roles of specialist doctor, nurse, and administrator, with each role linked to specific permissions.
Finally, the third part lists the rights associated with each role. For example, specialist doctors can access patients' medical records and issue prescriptions, while nurses can access appointment schedules and manage patient records.

4.4.3 The basic principles of the RBAC model are as follows

- In the DAC model, permissions are related to low-level operations For example, levels such as read/write operations, in the RBAC model, involve organizational tasks such as “transfer”, “. Buy a plane ticket" or something like that.

- In RBAC, the notion of roles corresponds to job functions. Permissions are granted to roles rather than users. The role is Then distribute to users based on their responsibilities the organization. The same permissions can be assigned to different roles and different Roles can be assigned to the same user.

- RBAC provides a solution to implement measures such as Separation of duties. The principle of separation of duties states that the same task Users cannot perform tasks that can be orchestrated Fraudulent transactions such as "Authorize payment" and " Payment. "This principle is easily guaranteed by RBAC To some extent, it can be said that two roles are mutual Exclusive. Two mutually exclusive roles cannot be assigned to the same role user.

RBAC is widely adopted and applied by enterprises and manufacturers in large organizations. Enterprise Software Trusted Solaris, Windows Authorization Manager, Oracle 9, Sybase, Adaptive Server Microsoft Active Directory, Most commercial DBMS, FreeBSD and Wikipedia implement some or all of Role-Based Model Principles [16]

4.4.4 RBAC Submodels (Family)

The RBAC model specification includes the following submodels (:

- RBAC0 model or "flat model", which presents the basic concepts and relationships i.e. the core of the model.

- **RBAC1** model or "hierarchical model", which uses the RBAC0 model and

The concept of hierarchy between roles is introduced.

- **RBAC2** model or "constrained model", which uses the RBAC0 model and

The notion of constraints is introduced.

- **RBAC3** model or "symmetric model"

which includes RBAC1 and

RBAC2 and considers the interaction between

constraints and hierarchies [14]

These successive improvements illustrate

the general direction of the research

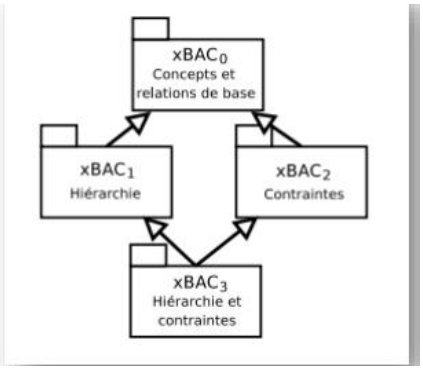

Figure1.6 : RBAC FAMILY

4.4.5 RBAC Core

Figure1.4 presents the core of the RBAC model, which includes the basic elements of the model and the relationships between them. It is defined by the following concepts: Users (USERS), Roles (ROLES), Sessions (SESSIONS), Permissions (PRMS), Operations (OPS), and Objects (OBS), as well as the relations PA (Permission Assignment), UA (User Assignment), session_users, and session_roles.

User

A user is defined as a human being, although the concept can be extended to machines or networks. [62]

Role

A role is a function within an organization, associated with certain authority and responsibility granted to users assigned to that function.

Object

Objects represent containers of information, such as files or directories in an operating system, or rows, tables, and views in a database management system.

Permission

A permission is an approval that allows a user to perform an operation on one or more protected system objects. [62]

Operation

An operation is an action invoked by a user on system objects, such as read, write, and execute. In a database management system, operations may include insert, delete, add, and update.

The relationships in the RBAC model are defined as follows:

UA (User Assignment)

This relation assigns users to roles ($UA \in USERS \times ROLES$). Formally: $User\_assigned(r) = \{u \in USERS \mid (u, r) \in UA\}$.

PA (Permission Assignment)

This relation expresses the assignment of permissions to roles. A role can have many permissions. Formally:
$Permission\_assigned(r) = \{p \in PRMS \mid (p, r) \in PA\}$.

Session

A session represents an instance of a user's interaction with the system. A user can create or delete a session and activate or deactivate a role within a session. Multiple roles can be activated within the same session, and a user can have multiple concurrent sessions. [62]

session_roles

This relation gives the roles activated in a session. Formally:
$session\_roles(si) = \{r \in ROLES \mid (session\_user(si), r) \in UA\}$.

session_users

This relation gives the set of sessions associated with a user. Formally:
Avail_Session_Perms(s) = ∪ Permission_assigned(r), where r ∈ session_roles(s).

Avail_Session_Perms(s)

The permissions available in a session.

Permission_assigned(r)

All permissions assigned to roles in that session.

4.4.6. The advantages and disadvantages of RBAC

Advantages	Disadvantages
• Provides a neutral/flexible policy. • Supports separation of duties constraints. • Ability to express DAC, MAC, and user-specific policies using role hierarchies and constraints.	• Weak administrative functionality. • Satisfies the principle of least privilege. • Can be easily integrated into current technology

Table1.2 : The advantages and disadvantages of RBAC.[17]

4.4.7 Table of Comparative Analysis of Access Control Model

Criteria	MAC	DAC	RBAC	ABAC
Security authority	Centralized	User	Centralized	Centralized
Access audit	Centralized	User	Centralized	Centralized
Generally	Centralized	User	Centralized	Centralized
Access rights propagation	NO	YES	NO	NO
Information flow control	YES	NO	NO	NO
Multi-domain	NO	NO	NO	NO

Table1.3 : Comparative Analysis of Access Control Model [63]

4.4.8 Combining RBAC and ABAC

Organizations often start by implementing a flat RBAC model because it is easier to configure and maintain. As organizations grow and handle more sensitive data, they realize the need for a more complex access control system. RBAC and ABAC can be used together, with RBAC performing most of the work and ABAC complementing it with finer filtering. [53]

This combined access model is also called RBAC-A. There are three RBAC-A approaches that manage the relationships between roles and attributes: [53]

- **Attribute-Centric**
  A role becomes the name of one of the user's attributes. This resembles a job title. The "role" attribute in such a model is used to mark a set of attributes required for a certain position.
- **Role-Centric**
  Attributes are added to constrain roles. In this model, attributes can reduce the permissions available to a user. This approach strengthens your data security.
- **Dynamic Roles**
  Attributes such as time of day are used to determine the subject's role. In some cases, a user's role may be entirely determined by dynamic attributes.[65]

Conclusion

Security in clinical information systems is essential for protecting patient data and ensuring efficient healthcare operations. Role-Based Access Control (RBAC) helps regulate access by assigning permissions based on user roles, minimizing unauthorized access while maintaining usability. However, effective security requires a balance between restriction and operational efficiency. Formal verification methods, like Alloy, enhance reliability by detecting vulnerabilities in access control policies. As technology evolves, security models must adapt to emerging threats while ensuring seamless integration into healthcare systems. Continuous improvement in access control and verification will strengthen trust, confidentiality, and efficiency in digital healthcare environments.

Chapiter 02

formal modeling,
formal modeling using alloy

I.Formal modeling

Introduction

In the realm of software engineering and system design, the need for precision, consistency, and verification has become more critical than ever. To meet these demands, formal modeling has emerged as a rigorous approach that employs mathematical logic to specify, develop, and verify systems. Unlike informal or semi-formal methods, formal modeling offers unambiguous descriptions, making it easier to detect errors early in the development cycle.

One of the widely recognized tools in this domain is Alloy, a lightweight formal modeling language based on relational logic. Alloy enables designers to model complex systems using a simple, yet expressive syntax, and to automatically analyze these models for consistency and correctness through the Alloy Analyzer. Its balance between expressiveness and automation makes it especially valuable in early-stage design, specification validation, and exploratory analysis.

This introduction sets the foundation for exploring both the theoretical underpinnings of formal modeling and the practical application of Alloy as a tool for modeling and analyzing system behavior.

1. Definition Formal modeling

Formal modeling utilizes math-based methods and notations to portray and examine how systems function and are organized. These models are written in formal languages, which possess specific rules of grammar and meaning. This enables interpretation and processing by verification and validation tools. Unlike informal models, which might employ natural language or diagrams, formal models avoid vagueness. They offer the ability for mechanical analysis, including automatic checks for logical soundness, accuracy, and thoroughness.

2. The Function of Formal Models in System Verification

Formal modeling isn't just for describing things; it also helps us check and prove systems are correct before we build them. Unlike models that are less precise, like those with diagrams or words that can be interpreted in different ways, formal models work with tools like model checkers and theorem provers. This ability greatly lowers the chance of errors in systems where being right is very important, for example, in medical software, finance, space systems, and security.

3. The Concept of Abstraction in Formal Modeling

A crucial aspect of formal modeling is abstraction. Rather than detailing every aspect of a system, a formal model concentrates on important features needed for analysis. For example, when modeling a medical information system, the model might disregard the interface and instead emphasize how patient data is retrieved, modified, and protected. This abstraction helps to isolate key properties, such as privacy or integrity, and confirm whether they're upheld under every condition set by the system's regulations. .[20]

4. Formal Modeling Matters in Critical Systems

Formal modeling's value becomes clear when we assess software errors' potential dangers. An error in a user interface might cause bewilderment, but a misstep in a nuclear reactor's control system could be devastating. Employing mathematical logic, formal modeling lets us demonstrate that specific qualities consistently remain true — such as a safety measure always working, or users never getting data access they shouldn't. Traditional testing, by itself, can't provide this assurance, because it only checks a limited range of situations

5. comparison of Formal, Semi-Formal, and Informal Modeling:

5.1 Informal Modeling

Relies on diagrams, natural language, or ad hoc methods. While useful for quick prototyping, it lacks rigor and is prone to ambiguity.

5.2 Semi-formal Modeling

Uses models that include some formal notation (e.g., UML), but they are not fully mathematically rigorous. These models offer a balance between precision and practicality.

5.3 Formal Modeling

Involves the use of fully mathematical languages to define system behaviors precisely, providing a high degree of confidence in verification.

6.Common Formal Modeling Languages

6.1 Alloy

A simple modeling language based on relational logic. It offers a strong visual aspect and works well for expressing intricate constraints.

6.2 B-Method

Centers around abstract machines, emphasizing iterative refinement to produce code that can be run.

6.3 Petri Nets

Particularly effective for modeling systems that run concurrently, such as workflows or communication rules.

6.4 Z notation

Grounded in set theory and first-order logic, it's well-suited for defining system states and the actions they take

Feature	Alloy	B-Method	Petri Nets	Z Notation
Basis	Relational logic	Abstract machines, refinement	Graph theory (places/transitions)	Set theory + first-order logic
Primary Use	Lightweight modeling + validation	Formal refinement to code	Concurrent/distributed systems	High-level system specification
Visualization	Strong (automatic diagram generation)	Limited (textual focus)	Strong (graphical representation)	Limited (mostly textual)
Tool Support	Alloy Analyzer	Atelier B, ProB,Rodin (for Event-B)	CPN Tools, PetriNet IDE	Z/EVES, CZT, ProofPower-Z
Verification	Simulation + bounded model checking	Proof-based refinement	Reachability, liveness analysis	Theorem proving

Concurrency	Limited (possible but not focus)	Supported via refinement	Excellent (native support)	Possible but not intuitive
Learning Curve	Moderate (declarative syntax)	Steep (refinement requires skill)	Moderate (visual but nuanced)	Steep (heavy math dependency)
Industrydoption	Academia + small-scale projects	Critical systems (rail, aviation)	Manufacturing, protocols	Aerospace, defense, legacy systems

Table 2.1 : Defferent between Formal Modeling Languages

7. Multiple Real-World Examples of Formal Modeling

7.1 Medical Data Systems (Alloy)

Alloy can be used to model access restrictions within a hospital system.

for example, define that only doctors can modify patient diagnoses, while nurses have view-only access. The Alloy Analyzer can then determine if any of these rules might be broken based on the model. .[20]

7.2 Train Control Systems (B-Method)

In France, the B-method has been utilized to formally model and verify train interlocking systems. These systems guarantee that only one train is on a particular track segment, thus preventing accidents. The model assures that mutually exclusive operation and safety measures are enforced in every possible case.

7.3 Cryptography Protocols (TLA+)

Protocols like SSL or customized authentication systems can be modeled with TLA+. This allows for formal verification of security attributes like message confidentiality, proper authentication, and protection against replay attacks

7.4 Smart Card Software (Z notation)

Companies such as Mondex have applied Z notation to model smart card operations, verifying that money cannot be created or lost due to a software fault. This formal specification was important for achieving ITSEC certification, which validates high-level security.

8. Library Management Systems (Alloy)

A model can enforce limits, such as ensuring that a member can only borrow a certain number of books concurrently. The Alloy tool validates this restriction throughout all possible situations, guaranteeing consistency and adherence to set policies.

. Challenges and Limitations of Formal Modeling

While formal methods are powerful, they come with challenges:

8.1 Complexity

Developing a formal model can be time-consuming and require specialized knowledge.

8.2 Scalability

Some formal methods struggle to scale to large or complex systems.

8.3 Usability

Formal methods may not be accessible to all developers, particularly those without a background in mathematics or formal logic.

8.4 Tooling

The tools and techniques for formal verification are still developing, and there may be limitations in support for certain models or systems.

II.Formel modeling using Alloy languages

Alloy

Alloy is a modeling language for structures and a set of tools to analyze them. It was developed at the Massachusetts Institute of Technology (MIT) by a software design group led by Daniel Jackson [21]. It is based on first-order logic and inspired by the Z language [22], allowing users to model system structures using relations. Alloy is declarative in nature, meaning it defines relations, constraints, and properties to describe systems rather than prescribing how to implement them. An Alloy model is essentially a set of constraints that produces a set of possible structures (1, 1.1, 1.2).

Alloy is supported by the Alloy Analyzer, a tool that offers semantic analysis and both textual and visual representations of the models. To analyze models and verify their consistency and the validity of their properties, the Alloy Analyzer employs SAT solvers (implementations of Boolean satisfiability theory) [23]. It offers two primary types of analysis:
• Simulation (command: *run*): used to obtain an instance of the model that satisfies all constraints.
• Validation (command: *check*): used to verify whether a theorem holds true for all possible instances of the model [21] (1.3**).**

1.4 Characteristics of Alloy

Alloy [24] has several features that distinguish it from other modeling languages and methods.

1.4.1 Limited Scope of Verification

To analyze the model, we must specify the boundary parameters. Therefore, the tests can only

be executed within this boundary to ensure that all counterexamples are generated within the specified limit.

1.4.2 Declarative Model

The Alloy system is declarative in the sense that it answers the question "How do we know when X has occurred?" In contrast to operational or imperative modelers who ask: "How can we reach X?"

1.4.3 Automatic Analysis

Unlike other declarative specification languages such as Z and UML [25] [26], Alloy models can be automatically analyzed to generate examples and counterexamples of the modeled system.

1.4.4 Structured Data

Alloy supports complex data structures such as trees [27].

1.5 Structure of Alloy

An Alloy model consists of:

1.5.1 Signatures

Signatures represent units of a system that can model aspects of the real world. The "sig" notation declares a base type. The keyword "abstract" before the signature means that the signature cannot be instantiated. Since the notion of a signature is similar to that of a class in object-oriented languages and the notation is similar, we use the concatenation operator '.' to access signature attributes. The keyword "extends" allows a signature to inherit from another signature.

Example:

Alloy code

CopierModifier

```
abstract sig policy {}

sig policy1 extends policy {}
```

The code above represents a signature policy1 that inherits from the abstract signature policy [27].

1.5.2 Facts

The keyword "fact" comes before facts. Facts do not require arguments and specify restrictions that apply to the signature throughout execution. An important feature of facts is that they are always true.

Example:

Alloy code

CopierModifier

```
fact {Fsobject in Root*.connects}
```

The fact above means that a file system is always connected [27].

1.5.3 Predicates

The keyword "pred" precedes the predicate. A predicate introduces a constraint that can be evaluated as true or false, unlike a fact which is always true. A predicate takes arguments and returns

a boolean variable.

Example:

Alloy code

CopierModifier

```
pred conflict (policya: policy) {
  (policya.subject.clearance =< policya.resource.classification and
   policya.action = read)
}
```

The predicate “conflict” returns true if a subject is allowed to read an object that has a classification higher than its clearance [27].

1.5.4 Assertions

Assertions are preceded by the keyword "assert". Assertions do not require parameters; they define constraints that the system must satisfy. The Alloy Analyzer tool displays counterexamples if it finds any instance that fails the assertion.

Example:

Alloy code

CopierModifier

```
assert oneRoot {one d: Directory | no d.parent}
```

The assertion oneRoot above indicates that there exists a single directory without a parent directory (the root directory), to indicate that there is only one root directory in the file system [27].

1.5.5 Functions

Functions are preceded by the word "fun". It takes an argument and returns a typed value.

Example:

Alloy code

CopierModifier

fun Mother (p : Person) : Woman {p.father}

The function Mother receives a person as an argument and returns a value of type Woman [28].

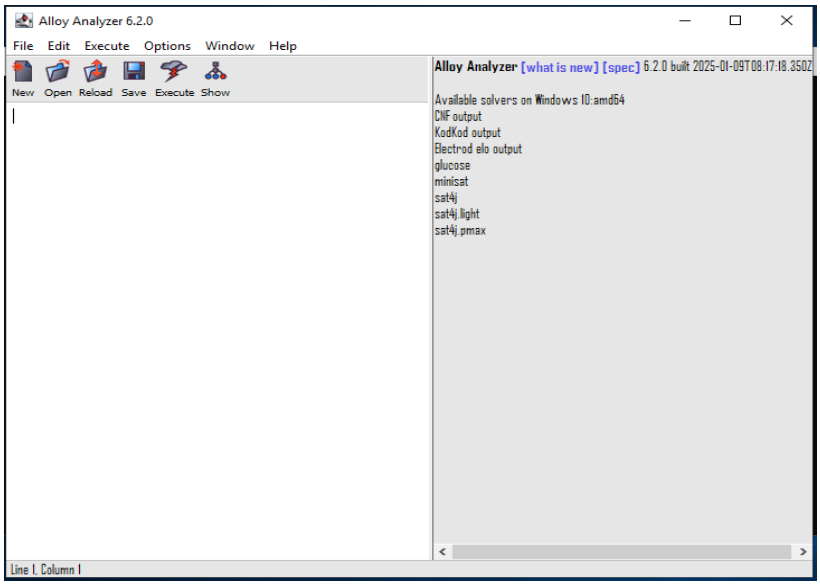

Figure 2.1 alloy analyzer

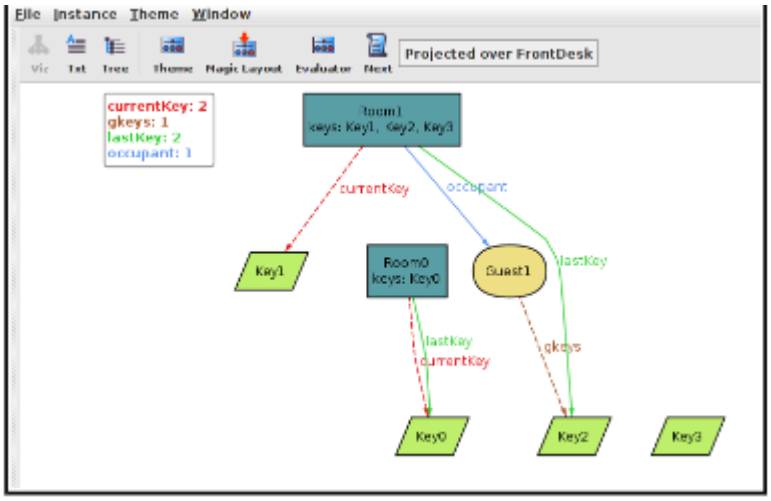

Figure 2.2 Alloy Analyzer's instance and counter example viewer.

2.Commands and Scope

2.1 Commands

The checks to be performed by the Alloy analyzer are declared using the run and check commands.

2.2 The run command

The run command instructs the Alloy Analyzer to search for an instance satisfying a given constraint; such instances are called examples. These constraints are generally expressed by predicates (formulas). For a more formal definition, given an Alloy model if:

— φmodel is the model formula, defined by the conjunction of the implicit constraints related to the declaration of signatures and fields, inheritance and multiplicity constraints, as well as the facts of the model;

— φrun is the formula of the run command.

Then, the command instructs Alloy Analyzer to verify the satisfiability of the formula φmodel ∧ φrun, that is, to check if there exists an interpretation structure M such that M φmodel ∧ φrun.

The run command is used in particular to verify the consistency of a model, that is, that the facts are not contradictory. Thus, consistency is generally the first command executed on a model to ensure that it is instantiable.

2.3 The check command

The check command instructs the Alloy Analyzer to verify whether an assertion is valid. More precisely, Alloy Analyzer searches for instances that would not satisfy the considered assertion. Such instances are called counterexamples.

Based on the above reasoning, the check command is formally defined as follows. Given an Alloy model where φmodel is the model formula and φcheck is the formula for the check command, the command instructs the model checker to verify the validity of the formula φmodel ⇒ φcheck, meaning M φmodel ⇒ φcheck for every interpretation structure M. By negating this, it amounts to verifying that there's no interpretation structure M such that M φmodel ∧ ¬φcheck.

2.4 Scope

Besides predicate or assertion names, the Alloy Analyzer also accepts scopes for each signature in the model.

Definition: A scope sets the maximum or the exact number of atoms that will be taken into account for each signature in a model's instance. It is introduced in the command by the keyword for. Moreover, to specify that a scope is exact, the expression but exactly is used.

Note 4 (Default scope). If no scope is specified, the Alloy Analyzer attributes a default scope of three to all top-level signatures [37].

Example 3: In the Hotel model of figure 1, the predicate consistency specifies the coherence that is verified by the run consistency command. In the execution of this command, we will consider at most four hosts, exactly four keys, and exactly three rooms. Furthermore, for the check NoBadEntry command, Alloy Analyzer will consider three atoms for each signature.

2.5 Verification

Alloy is based on FOR, which is an undecidable logic. Therefore, there is no verification method to prove the validity of an assertion. However, Alloy offers an analysis method called instance finding [37], [38], [39], [40] which consists of exhaustively analyzing an assertion within a finite instance domain defined by the scopes of the signatures. If a counterexample is found in this domain, then the assertion is not valid. However, the absence of counterexamples in this domain does not mean that the assertion is true, as it could be false in a larger domain.

Alloy, thus, focuses not on proving the assertion, but on finding counterexamples that invalidate the assertion. This instance finding technique used by Alloy is well-suited for analyzing invalid assertions because counterexamples are generated quickly (in smaller domains). Generally, Alloy's effectiveness relies on the small scope hypothesis:

Small Scope Hypothesis: Most bugs have a small counterexample. In other words, if an assertion is invalid, it probably has a small counterexample. [37]

An implication of this hypothesis is that for an assertion, if analyzed on very small models, it is highly probable that a counterexample will be found for this assertion.

3.Why using Alloy?

Below, we highlight the reasons that led to our choice of Alloy:

– It supports a wide range of properties, including invariants, user-defined assertions, LTL [31], and CTL [32] formulas with fairness constraints.

– The language's expressiveness is sufficient to represent a UML model along with OCL constraints almost directly, as demonstrated by the UML2Alloy tool [9]. This aspect is particularly important for the Model-Driven Development (MDD) community.

– Alloy's logic is very generic and doesn't force users into a specific style of specification for modeling and verifying reactive systems.

– It allows specifying the SAT solver to be used for problem-solving, enabling us to benefit from the ongoing advancements in verification time for SAT solving.

– It has the capability to produce the "unsatisfiable core" (UnSAT) when it fails to find an instance for a given problem (i.e., the set of clauses for which no satisfying instance exists). Thus, when evaluating a property fails, it is possible to provide explanations for the reason for this failure based on the problematic clauses.

– It has a graphical tool and a Java API to seamlessly integrate the analysis and verification process into a BPMS.

– It boasts a large and active community, with continuous development from the MIT team.

– Because it is based on first-order logic, implementing the formalization described in Chapter 3 can be done in a (nearly) direct manner.

– Many extensions are offered by the community, such as: (1) DynAlloy [33], a dynamic version of Alloy offering operational constructs to model the execution of operations and reason about execution traces. (2) Moolloy [34], an extension implementing a Guided Improvement Algorithm (GIA) to solve multi-objective optimization problems. (3) Kelloy [35], a tool to verify Alloy specifications on infinite domains by translating them into the input language of the KeY theorem prover [6].

These tools will, in the future, offer the potential to enhance the Alloy4PV framework in an almost "free" way. For instance, Moolloy would allow solving properties like "what is the fastest execution to run activity A", when Alloy is only able to answer the satisfiability question such as "which path executes activity A", without the notion of optimization of the solution found.

Therefore, to implement the formalization presented in Chapter 3, we use the Alloy language. Alloy's growing popularity as a formal method for the MDD community stems from its object-oriented notation based on simple relational logic, incorporating the concepts of abstraction and inheritance. This notation shares many similarities with UML enhanced with OCL constraints, while offering a powerful automatic analysis tool based on a SAT solver, making the power of formal verification available to software engineers.

4. Modeling Behavior in Alloy

System changes are described by sequences of executions involving basic events.

4.1 Modeling Time

Alloy works with FOR, where behavioral properties cannot be specified inherently. To analyze dynamic systems, Alloy employs well-known idioms [35] that introduce a Time signature into the model. Time atoms are ordered, representing potential time points. This order (use/ordering) is defined by a smallest element (first), a largest element (last), and a partial function 'next' that computes an atom's successor.

Alloy provides two idioms for defining variable fields, with an idiom being a standard way to represent a common construct.

global state: The Time signature encompasses all variable fields in the model;

local state: Variable fields exist within the same signatures as static fields. In this instance, an extra column representing time is added to variable fields, with the join on this column indicating a field's value at each time point.

Example 1 ;. The Hotel model is defined using the local state idiom with the Time signature, thus the variable fields currentKey, lastKey, occupant, gkeys each have an added Time

column.

```
fact traces {
  init [ first ]
  all t: Time-last | let t' = t.next {
    some g: Guest, r: Room, k: Key |
      entry [t, t', g, r, k]
      or checkin [t, t', g, r, k]
      or checkout [t, t', g]
  }
}
assert NoBadEntry {
  all t: Time, r: Room, g: Guest, k: Key {
    let t' = t.next,
      o = FD.occupant.t[r]{
      entry [t, t', g, r, k]
      and some o ⇒ g in o
  }
}
pred consitence {}
run consistence for 4 but exactly 4 Key,
  exactly 3 Room
```

```
fact NoIntervening {
  all t: Time-last {
    let t' = t.next, t'' = t'.next{
      all g: Guest, r: Room, k: Key {
        checkin [t, t', g, r, k]
        ⇒ (entry [t', t'', g, r, k] or no t'')
          or entry [t, t', g, r, k]
            and some o ⇒ g in o }}
  }
}
check NoBadEntry
check NoBadEntry for 5 but exactly 5 Key,
  10 Time
```

FIGURE 2.3 ; model alloy in hotel

4.2 Modeling Events

Event modeling in Alloy follows a common pattern, but this pattern isn't rigid.

Alloy's flexibility allows designers to customize patterns as needed. Two common patterns are used in Alloy to specify system events.

Predicate Pattern: In this pattern, system events are defined by predicates that relate two consecutive points in time. The model's state transitions are determined by a relationship between guards and postconditions.

The predicates' guards specify the state prior to the event's execution (the conditions under which the operation is permitted), while the postconditions describe the system's state (the effects of the operation) after the event has occurred.

Example 2. In the Hotel model from picture 1, the starting states fulfill the predicate `init`. This predicate outlines the initial setup's constraints: no key for each host, (no occupant in any room, and for every room, the key currently in the lock also matches the last one registered at reception. The subsequent execution trace indicates a state result from running at least one of the operations: checkin, entry, and checkout.

```
abstract sig Event {
  pre, post : Time,
  g: Guest
   }
abstract sig RoomKeyEvent extends Event {
room: Room ,
key: Key ,
 }
sig checkout extends Event{}  {
  let occ = FD.occupant {
    some occ.pre.guest
    occ.post = occ.pre − Room →guest
}
```

```
sig entry extends RoomKeyEvent { } {
  key in guest.keys.pre
  let ck = room.currentKey{
    (k = ck.pre and ck.post = ck.t) or
    (key = nextKey[ck.pre, room.keys] and
          ck.post = key)
}
sig checkin  extends RoomKeyEvent { }{
  guest.keys.post = guest.keys.pre + key
    let occ = FD.occupant {
    no occ.pre [room]
    occ.post = occ.pre + room → guest
  }
  let lk = FD.lastKey {
    lk.post = lk.pre ++ room → key
    key = nextKey [lk.pre [room], room.keys]
 }
 }
```

```
fact traces {
  init  [ first ]
  all t: Time−last | let t' = t.next {
    some e: Event {
      e.pre = t and e.post = t'
// Conditions du cadre a la Reiter
      currentKey.t != currentKey.t'
        ⇒ e in entry
      occupant.t != occupant.t'
        ⇒ e in checkin + checkout
      lastKey.t != lastKey.t
        ⇒ e in checkin
  }
}
assert NoBadEntry {
  all e: entry {
    o = FD.occupant.(e.pre) [e.room]|
      some o ⇒  e.guest in o
  }
}
```

```
fact NoIntervening {
  all c: checkin {
    c.post = last
    e.room = c.room
    e.guest = c.guest
    }
}

pred consitence {}
run consistence for 4
  but exactly 4 Key,
  exactly 3 Room , 6 Event
check NoBadEntry for 5
  but exactly 5 Key, 10 Time
```

5. Execution Trace

The system's evolution is specified by defining the allowed states during execution, known as the execution trace. In Alloy, a trace is defined by a universal quantification over the atoms in the signature representing time, along with a constraint. This constraint dictates that a state transition must be triggered by at least one operation's execution. Therefore, a state in the model is either an initial state (at the first-time step), or a state that arises from the execution of at least one operation.

Consequently, temporal property analysis in Alloy is conducted on traces, bounded by the number of atoms within the time-representing signature. As a result, Alloy is less suited for analyzing properties on infinite traces, such as liveness. However, it performs well for analyzing safety properties.

Nevertheless, [36] presents a technique for verifying behavior on unbounded traces that incorporate a lasso loop, characterized by a relationship between the final state and a prior state.

6.Framework Conditions

It's essential to monitor the progression of variable fields to ensure the model aligns with the actual system. This monitoring is achieved through the framework conditions.

Frame conditions determine how operations affect variable fields.

These conditions can specify which variable fields will remain unchanged after an operation. If so, they are included within the operation's definition.

Alternatively, they can state which operations are permitted to alter a given variable field; then, the conditions are defined separately from the operations, using extra facts. These are also known as frame conditions in the style of Reiter.

For example, in Hôtel, frame conditions are incorporated within the operations. Specifically, the predicates noRoom Change Except and noGuest Change Except [l:2] in Figure1 describe modifications of the variable fields within the Room and Guest signatures, respectively. These predicates express the frame conditions for those fields.

Within the checking operation, the frame conditions for variable parameters are expressed by the conjunction of these two predicates. Consequently, when the checkingoperation runs, no room key changes, and, except for guest g who adds key k to their key ring, no other guest changes their key ring. The key ring of guest g and the current key of room r are then modified.

Conclusion

Formal modeling plays a crucial role in the development of reliable and error-free systems. By using mathematical logic and formal specification techniques, it enables developers to define system behavior precisely and verify properties before implementation. This rigorous approach minimizes ambiguity, reduces the risk of bugs, and enhances overall system quality—especially in safety-critical domains such as aerospace, finance, and healthcare.

Among the tools that support formal modeling, **Alloy** stands out for its simplicity, expressiveness, and practical utility. Unlike traditional formal methods that often require deep expertise in formal logic, Alloy offers a lightweight and user-friendly syntax, making formal modeling more accessible to engineers and designers. Through its **Alloy Analyzer**, it provides automated support for model checking, allowing users to explore scenarios, test assumptions, and detect inconsistencies within system models.

Modeling with Alloy encourages early validation of specifications, fosters a deeper understanding of system structures, and supports exploratory design. Its combination of precision and ease of use makes it ideal for both academic research and real-world software engineering projects.

In conclusion, formal modeling—especially with Alloy—empowers designers to build better systems by uncovering hidden flaws and validating complex interactions early in the design process. As systems grow in complexity and criticality, the adoption of such formal techniques becomes not just beneficial, but essential.

Chapter 03

Modeling class diagram with alloy

Introduction

Functional models describe clinical processes and the interaction of an information system with its environment. In this work, we class diagram to represent system functionality and depicts classes, which include both behaviors and states, with the relationships between the classes between clinic system.

In this work, we employ a **class diagram** to illustrate the relationships between database tables, clarifying their structure and functionality. This diagram helps define the system's data model, including access permissions and user roles.

To rigorously define and validate the system's structure, we use **Alloy**, a lightweight formal specification language designed for modeling and verifying software designs. Alloy enables precise representation of structural relationships, enforcement of integrity constraints, and automated consistency checking. By modeling the database schema and its associated logic, we ensure the design is both robust and error-free. Furthermore, we demonstrate how the class diagram's logic translates into Alloy, ensuring the system's intended behavior is accurately captured and formally verifiable. [46]

1.Class Diagram

A class diagram is a static model that shows the classes and the relationships among classes that remain constant in the system over time. The class diagram depicts classes, which include both behaviors and states, with the relationships between the classes [47]. Figure 3.1 below shows class diagram of clinic system.

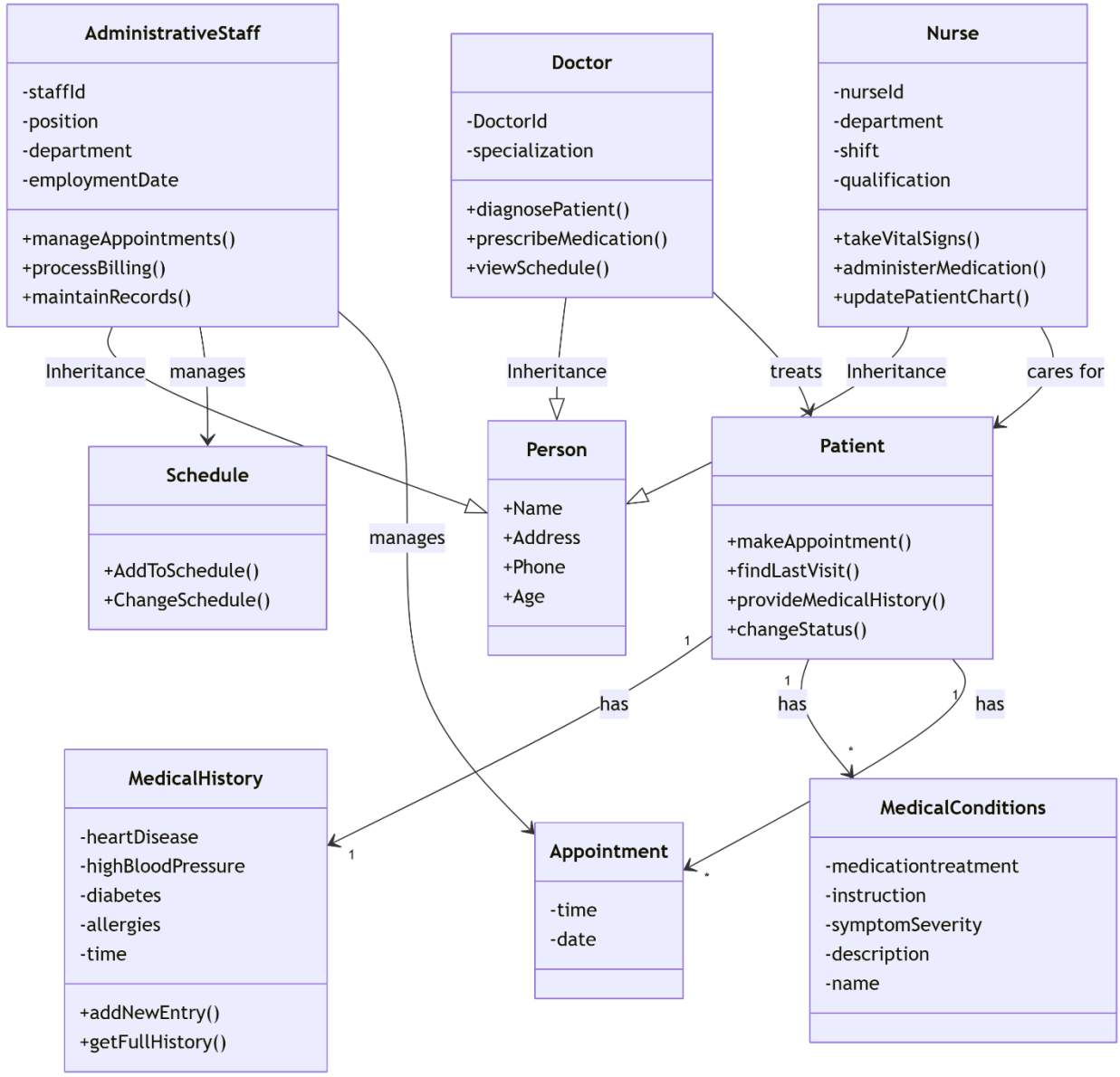

Figure 3.1 Class Diagram for Clinic System

2. Modeling in Alloy

2.1 Class Diagram to alloy

to model this class diagram in Alloy. we should understanding the components and relationships shown in the class diagram **figure (1)** step-by-step guiding to transform class diagram into an Alloy model of clinic system. [48]

2.1.1 Analyze Class Diagram Components

Identify core entities and their relationships from the diagram.

Core Entities (Main Classes)

Entity	Type	Description	Alloy Mapping Strategy
Person	Abstract	Base class for all human actors in the system	abstract sig with common fields
AdministrativeStaff	Concrete	Handles appointments, billing, and records	sig extends with staff-specific fields
Doctor	Concrete	Medical professionals with specializations	Sig extends + specialization field
Patient	Concrete	Healthcare recipients with medical histories	sig extends + medicalHistory relation
MedicalCondition	Abstract	Clinical conditions (heart disease, diabetes, etc.)	Hierarchy of one sig extensions
MedicalHistory	Concrete	Temporal record of patient conditions/treatments	sig with Time-indexed entries

Appointment	Concrete	Scheduled interactions between patients and doctors	sig linking Patient + Doctor + Time

Table 3.1 core entities of main classes

2.1.2 Supporting Entities

Entity	Type	Purpose	Modeling Approach
Status	Enum	Patient lifecycle states (Active/Inactive/Discharged)	abstract sig + one sig extensions
Time	Value	Temporal tracking for appointments/history entries	util/ordering[Time] for sequencing
Department	Enum	Organizational units (Cardiology, Pediatrics, etc.)	abstract sig + one sig extensions

Table 3.2 supporting entities using

3. Create Basic Signatures (Classes) :

Map each class to an Alloy signature.

Alloy part code

```
abstract sig Person {}  // Base class for all human actors
sig Patient extends Person {}  // Patient specialization
sig Doctor extends Person {}   // Medical professional
sig Nurse extends Person {}    // Nursing staff
```

Signatures **(sig)** define the core classes. **abstract sig** Person ensures all person types share common attributes (added later).

4. Add Attributes

Convert class attributes to signature fields.

Alloy part code

```
abstract sig Person {
name: one Name, // Each person has exactly one name
address: one Address, // One address (no nulls)
phone: lone Phone, // Mandatory phone number
age: one Age // Age is required }
```

5. Model Relationships

Represent associations between classes.

Alloy part code

```
sig Appointment {
   patient: one Patient,  // 1-to-1 relationship
   doctor: one Doctor,    // 1-to-1 relationship
   time: one Time
}
sig MedicalHistory {
   patient: one Patient,       // Each history belongs to one patient
   conditions: set Condition    // 1-to-many relationship
```

6. Handle Inheritance

Implement class hierarchies.

Alloy part code

```
abstract sig Staff extends Person {}  // Intermediate abstract class
```

Purpose: Creates an **abstract** category for all staff members.

```
sig Doctor extends Staff {}          // Doctor inherits from Staff
```

Purpose: Defines doctors as a specialized staff type

```
sig Nurse extends Staff {}           // Nurse inherits from Staff
```

Purpose: Defines nurses as another staff subtype.

```
one sig Active, Inactive, Discharged extends Status {} // Patient states
```

abstract sig defines general types.

7. Model Methods as Predicates

Purpose: Convert class methods to executable predicates.

Alloy part code

```
pred scheduleAppointment[p: Patient, d: Doctor, t: Time] {
   some a: Appointment {  // Create new appointment
      a.patient = p
      a.doctor = d
      a.time = t }}
```

1. Takes a patient, doctor, and time as input
2. Creates/finds an appointment (some a)
3. Connects them through fields
4. Acts like a method that enforces valid scheduling rules

This declaratively states: "An appointment exists that connects this patient, doctor, and time." No implementation details - just the essential constraints.

8. Add Constraints (Facts)

Enforce system invariants.

Alloy part code

```
fact NoTimeTravel {
   // Appointments must be in the future
   all a: Appointment | a.time > currentTime
}
fact UniqueStaffIDs {
   // No two staff members share same ID
   no disj s1, s2: Staff | s1.staffId = s2.staffId}
```

- **NoTimeTravel Fact:**
  - Ensures appointments can't be scheduled in the past
  - a.time > currentTime means every appointment's time must be after the current system time
  - Prevents invalid/backdated appointments
- **UniqueStaffIDs Fact:**
  - Guarantees each staff member has a unique ID
  - **no disj** s1, s2 means no two distinct staff members
  - s1.staffId = s2.staffId prevents ID duplication

Key Points:

- Both are system-wide constraints (always enforced)
- First maintains temporal validity
- Second ensures data integrity for staff records

9. Create Helper Signatures

Define enums and value types.

Alloy part code

```
abstract sig Department {}  // Enum-like definition
one sig ER, ICU extends Department {}
```

Creates an enum-like structure for hospital departments and **ER** and **ICU** are the only possible department values and and **one sig** ensures single unique instances (ER ≠ ICU)

```
abstract sym Status {}      // Patient status enum
one sig Active, Discharged extends Status {}
```

Defines possible patient states as an enum and **Active** and **Discharged** are mutually exclusive statuses and **one sig** guarantees these are distinct constants

Key points:

- Both model fixed sets of options (like enums in programming)
- **abstract sig** prevents direct instantiation
- **one sig** creates single immutable instances
- Used for type-safe attribute values in the clinic system

10. Add State Management

alloy part code

```
sig Clinic {
   patients: set Patient, // All patients in the clinic
   appointments: set Appointment // All scheduled appointments
}
```

Represents the clinic as a whole and Tracks all patients and appointments in the system

```
pred admitPatient[c, c': Clinic, p: Patient] {
   c'.patients = c.patients + p  // State transition
}
```

Models a state change where:

- c = current clinic state
- c' = next clinic state after admission
- p = patient being admitted

- The operation adds the patient to the clinic's patient set

Key points :

- Shows state transitions (before/after admission)
- c and c' represent different time states
- Simple but powerful way to model system changes

11. Validate with Assertions

alloy part code

```
pred validSystem {
   all p: Patient | some p.medicalHistory //(1)
   all a: Appointment | a.patient != a.doctor //(2)
   all mh: MedicalHistory | some mh.conditions //(3)
}
check checkValidSystem {
   validSystem
} for 5
```

In this Alloy part code validates core healthcare system rules:

1. **ValidSystem Predicate enforces 3 key requirements:**
   - Every patient must have medical history (some p.medicalHistory)
   - No doctor can be their own patient (a.patient ≠ a.doctor)
   - All medical records must document conditions (some mh.conditions)
2. **Verification Command:**
   - **Check checkValidSystem** tests these Rules
   - Examines all possible system states with up to 5 instances (for 5)

- Searches for violations (counterexamples)

In Simple Terms:

"Verify the clinic system always maintains: (1) complete patient records, (2) no self-treatment, and (3) properly documented conditions." Alloy will either confirm these always hold or find breaking cases.

12. Run Verification

alloy part code

```
pred showExample {
    #Patient > 2
    #Doctor > 1
    #Nurse > 1
    #AdministrativeStaff > 1
    #Appointment > 3
    some p: Patient | #p.medicalHistory > 1
    some n: Nurse, p: Patient | takeVitalSigns[n, p]
}
run showExample for 5 but 5 Person, 8 Appointment, 6 MedicalHistory, 5 Time
```

This Alloy code generates a sample clinic scenario with:

1. **Minimum Requirements**:
   - At least 3 patients, 2 doctors, 1 nurse, and 1 admin staff
   - Minimum 4 appointments
   - At least 1 patient with multiple medical history entries
   - At least 1 nurse taking vital signs for a patient
2. **Execution Command**:
   - **run showExample** creates valid instances matching these conditions
   - Limits to maximums (for 5 but...):
     - 5 Persons total
     - 8 Appointments
     - 6 MedicalHistory records
     - 5 Time points

Purpose:

Generates a realistic test case showing how all components interact, while keeping the example small enough to analyze. Helps verify the model works as intended.

Keys Conversion Rules :

CLASS DIAGRAM ELEMENT	ALLOY EQUIVALENT
Class	sig
Inheritance	extends
Attribute	Field in signature
1-to-1 Relationship	one
1-to-Many Relationship	set
Method	pred
Composition	Nested signature with ownership fact
Enumeration	abstract sig with one sig subtypes

The Alloy model for clinic system class diagram

```
// Step 1: Basic Types and Orderings
open util/ordering[Time] as TO
sig Time {}
sig Name, Address, Phone, Age, Text {}

// Step 2: Department and Shift Structures
abstract sig Department {}
one sig Cardiology, Pediatrics, Oncology, Emergency, GeneralPractice extends Department {}

abstract sig Shift {}
one sig MorningShift, AfternoonShift, NightShift extends Shift {}

// Step 3: Unified Position Hierarchy (FIXED DUPLICATES)
abstract sig Position {}
one sig Receptionist, BillingSpecialist, RecordsClerk extends Position {}

abstract sig MedicalQualification {}
one sig RN, LPN, NP extends MedicalQualification {}

// Step 4: Specializations (SINGLE DEFINITION)
abstract sig MedicalSpecialization {
   department: one Department
}
one sig Cardiologist extends MedicalSpecialization {} { department = Cardiology }
one sig Pediatrician extends MedicalSpecialization {} { department = Pediatrics }
one sig Oncologist extends MedicalSpecialization {} { department = Oncology }
one sig GeneralPractitioner extends MedicalSpecialization {} { department = GeneralPractice }
```

```
// Step 5: Core Person Signature
abstract sig Person {
   name: one Name,
   address: one Address,
   phone: one Phone,
   age: one Age
}

// Step 6: Administrative Staff
sig AdministrativeStaff in Person {
   staffId: one Int,
   position: one Position,
   adminDept: one Department,
   employmentDate: one Time
}

// Step 7: Doctor Implementation
sig Doctor in Person {
   doctorId: one Int,
   specialization: one MedicalSpecialization,
   schedule: set Appointment
}

// Step 8: Nursing Staff
sig Nurse in Person {
   nurseId: one Int,
   nurseDept: one Department,
   shift: one Shift,
   qualifications: set MedicalQualification
}
```

```
// Step 9: Medical Components
sig Description, SymptomSeverity, Medication, Institution {}

abstract sig MedicalCondition {
    name: one Name,
    description: one Description,
    symptomSeverity: one SymptomSeverity
}
sig HeartDisease, HighBloodPressure, Diabetes, Allergies extends MedicalCondition {}

sig MedicalHistory {
    conditions: set MedicalCondition,
    treatment: set Medication,
    instructions: set Text,
    recordedTime: one Time,
    entries: set HistoryEntry
}

sig HistoryEntry {
    entryTime: one Time,
    details: one Text
}

sig MedicalConditions {
    medicationTreatment: set Medication,
    institution: one Institution,
    symptomSeverity: one SymptomSeverity,
    description: lone Text,
    conditionName: one Text
}
```

```
// Step 10: Patient System
abstract sig Status {}
one sig Active, Inactive, Discharged extends Status {}

sig Patient in Person {
    medicalHistory: set MedicalHistory,
    status: one Status,
    appointments: set Appointment,
    conditions: set MedicalConditions
}

sig Appointment {
    patient: one Patient,
    doctor: one Doctor,
    time: one Time,
    date: one Time
}

// Step 11: Operational Predicates
pred makeAppointment[p: Patient, d: Doctor, t: Time, dt: Time] {
    some a: Appointment | {
        a.patient = p
        a.doctor = d
        a.time = t
        a.date = dt
        a in p.appointments
        a in d.schedule
    }
}
```

```
pred takeVitalSigns[n: Nurse, p: Patient] {
    n.nurseDept = p.appointments.doctor.specialization.department
    some e: HistoryEntry | e in p.medicalHistory.entries
}

pred manageAppointments[s: AdministrativeStaff, a: Appointment] {
    s.adminDept = a.doctor.specialization.department
}

pred administerMedication[n: Nurse, p: Patient, m: Medication] {
    m in p.conditions.medicationTreatment
    some e: HistoryEntry | e in p.medicalHistory.entries
}

// Step 12: System Constraints
fact UniquePersonDetails {
    no disj p1, p2: Person |
        p1.name = p2.name and
        p1.address = p2.address and
        p1.phone = p2.phone and
        p1.age = p2.age
}

fact MedicalHistoryOwnership {
    all mh: MedicalHistory | one p: Patient | mh in p.medicalHistory
}

fact NoSelfTreatment {
    all a: Appointment | a.patient != a.doctor
}
```

```
// Step 13: Validation and Example
pred validSystem {
    all p: Patient | some p.medicalHistory
    all a: Appointment | a.patient != a.doctor
    all mh: MedicalHistory | some mh.conditions
}

check checkValidSystem {
    validSystem
} for 5

pred showExample {
    #Patient > 2
    #Doctor > 1
    #Nurse > 1
    #AdministrativeStaff > 1
    #Appointment > 3
    some p: Patient | #p.medicalHistory > 1
    some n: Nurse, p: Patient | takeVitalSigns[n, p]
}

run showExample for 5 but 5 Person, 8 Appointment, 6 MedicalHistory, 5 Time
```

Figure 3.2 Alloy model Code for clinic system

Analysis of Alloy Model Execution and Instance Generation

1. Explanation of Execution Output

We executed the code using the Alloy Analyzer to run the showExample predicate under specific scope constraints, and here is the result **Figure 3.2** : The Analyzer successfully generated a valid instance that satisfies all defined constraints, confirming the logical consistency of the model. Below is a breakdown of what the execution output means.

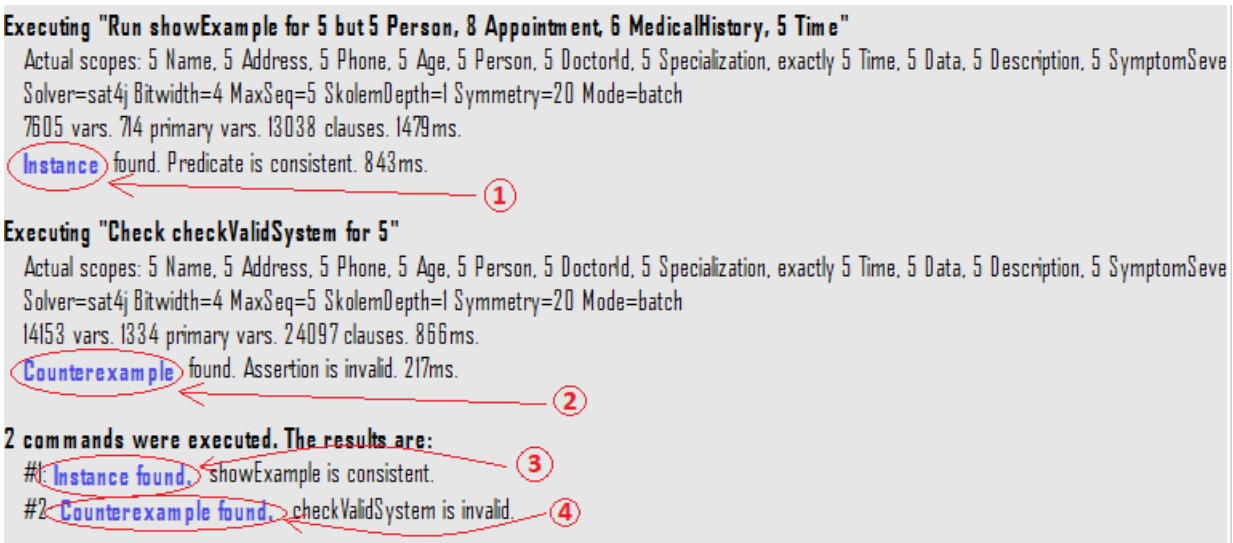

Figure 3.3 result after the Execution

We executed the command:

alloy part code

run showExample for 5 but 5 Person, 8 Appointment, 6 MedicalHistory, 5 Time.

This informations means Alloy was asked to find an example (instance) of our model that satisfies the showExample predicate, with the following scope limits:
5 Person: Only up to 5 people total (includes patients, doctors, etc.)
8 Appointment: Up to 8 appointments allowed
6 MedicalHistory: Up to 6 entries in total for medical history
5 Time: Exactly 5 distinct time points used

Other types like Name, Address, Status, etc., also have upper bounds or specific constraints (e.g., exactly 1 Active, exactly 1 Inactive, exactly 1 Discharged)

So **Alloy Analyzer** performed two main tasks in our example, using **automated reasoning** (via the SAT solver *sat4j*) to analyze your model. Here's a breakdown of what happened:

1. First Command: Instance Finding (showExample**)**

Goal: Check if the model is consistent (i.e., if there exists at least one valid configuration where all constraints hold).
In this step will understand what Alloy Did:

- **Scope Handling**: Alloy assigned exact bounds to each type (e.g., 5 Name, 5 Person, exactly 1 Active for Status, etc.).
- **SAT Solving**: It translated your model into a Boolean satisfiability problem (SAT) with:
  - **7,605 variables**, **7,114 primary variables**, and **13,038 clauses**.
  - The solver (*sat4j*) searched for a solution where all constraints are satisfied.
- **Result**: Found a valid instance in **843ms**, proving your model is **consistent** (no contradictions).

Key Insight: This means your model is logically possible—there exists at least one system state where all your defined rules hold.

2. Second Command: Assertion Checking (checkValidSystem**)**

Goal: Verify if the property checkValidSystem holds **for all possible instances** within the given scopes.
Also In this step understand what Alloy Did:

- **Adjusted Scopes**: Some scopes changed slightly (e.g., MedicalHistory reduced from 6 to 5, Appointment from 8 to 5).
- **Larger SAT Problem**: This check was more complex, with:
  - **14,153 variables**, **1,334 primary variables**, and **24,097 clauses**.
- **Result**: Found a **counterexample** in **217ms**, proving the assertion is **invalid**.

Key Insight: The property checkValidSystem is **not universally true**. There exists at least one valid system state where it fails.

2.Visual Interpretation of an Alloy Instance :

The **Alloy Analyzer** provides a visual snapshot of a possible scenario that satisfies all the constraints defined in the model. This walkthrough helps interpret the graphical

elements—such as objects, relationships, and time ordering—offering insights into how the system behaves under certain conditions.[49]

In those images below shows visual instance generated by **the Alloy Analyzer** that follows all your model's rules when we click in “1”,”2”,”3”,”4” As shown in **figure 3.2** above, each box is an object like a patient or doctor, and the arrows show their relationships. It helps you see if the system behaves the way you expect.

From click 1 of Executing "Check checkValidSystem for 5":

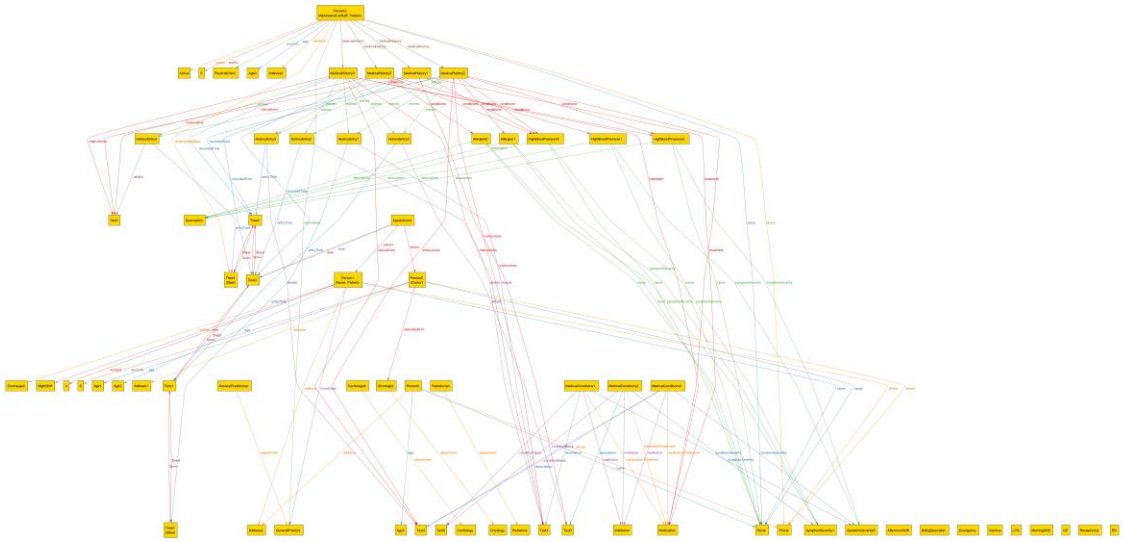

Figure 3.3 visual instance generated by the Alloy Analyzer of Executing "Check checkValidSystem for 5"

From click 2 of Executing "Run showExample for 5 but 5 Person, 8 Appointment, 6 MedicalHistory, 5 Time":

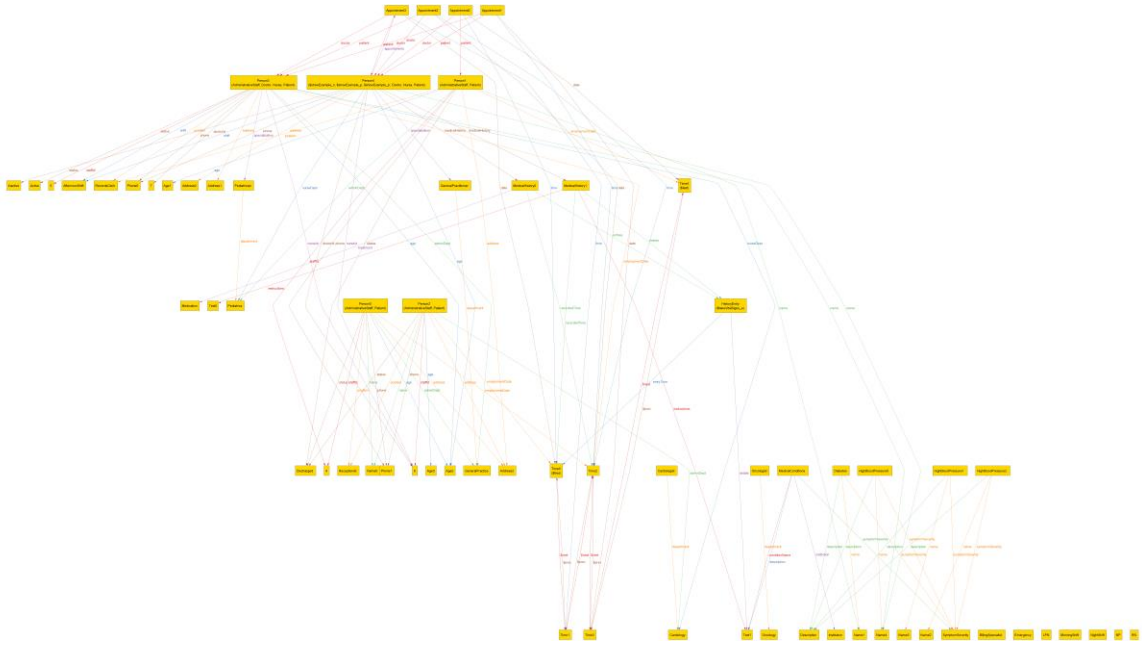

Figure 3.4 visual instance generated of Executing “Run showExample …”

Counter Example from click 3:

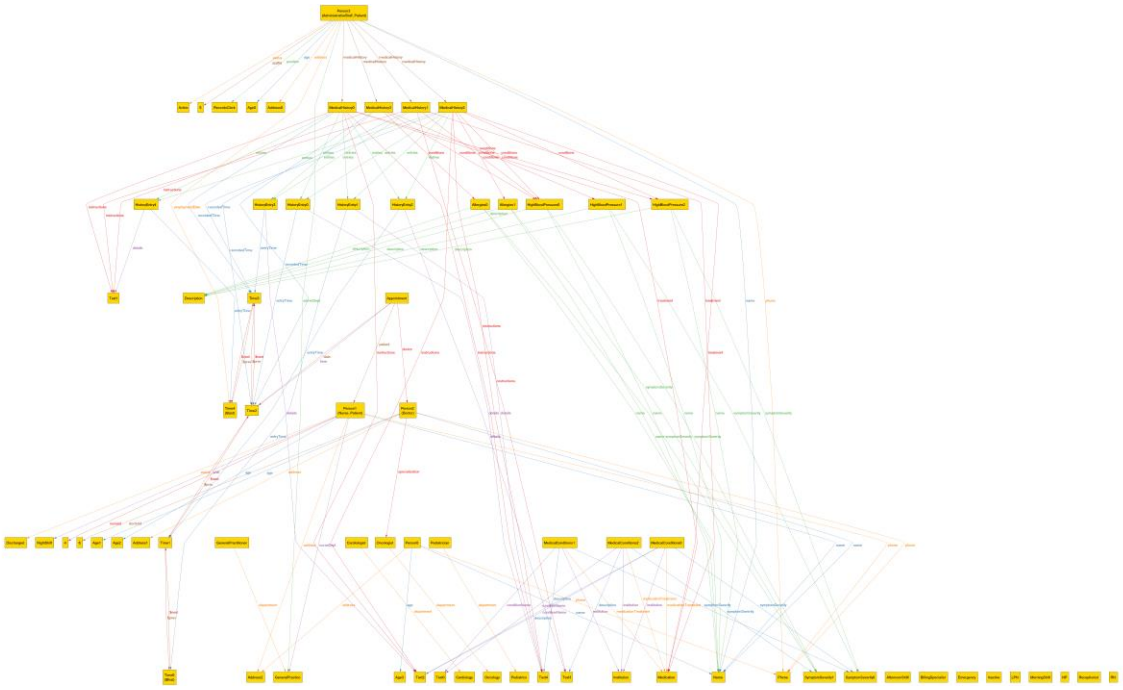

Figure 3.5 visual instance generated by the Alloy Analyzer from click 3

Instance from click 4:

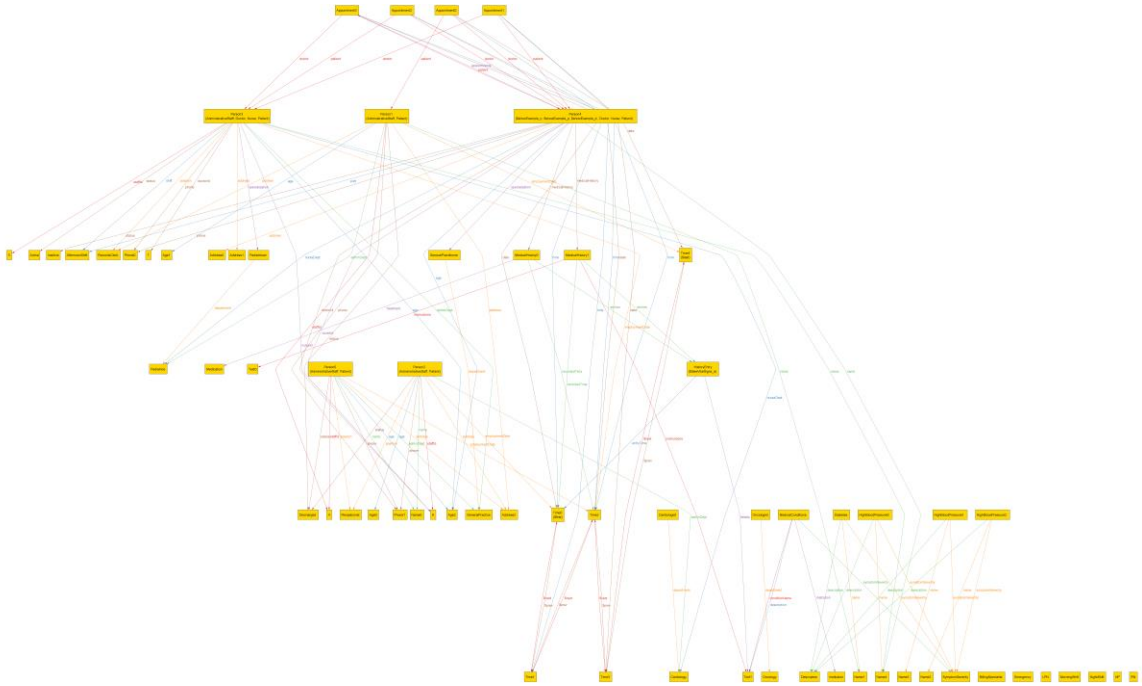

Figure 3.6 visual instance generated by the Alloy Analyzer from click 4

Understanding the Alloy Analyzer Output: A Visual Walkthrough

Signatures and Atoms (Yellow Boxes):

Each yellow box represents an atom, or an instance of a signature (i.e., a concrete object created from your model).

Examples:

Person0, Person1, Person2: instances of the Person sig (some are also instances of Doctor, Patient, etc.).

Appointment0, Appointment1, etc.: instances of the Appointment sig.

Colored Arrows (Relations)

Each colored arrow represents a field or relation between atoms.

Examples:

appointments: arrows from a Patient to several Appointments.

doctor, patient, and time: inside each Appointment, these show who is involved and when.

medicalHistory: from a Patient to MedicalHistory entries.

conditions: from a MedicalHistory to MedicalConditions (like Diabetes).

status: maps Patients to their Status (e.g., Active, Discharged).

$next/$prev: defines temporal ordering of Time using util/ordering.

Multiple Roles in One Person:

Note how:

Person2 is marked as:

($showExample, _p, Applicant, Doctor, Patient)

This means that in the current solution:

It's a patient

It’s also a Doctor

Also an Applicant

Also satisfies your predicate showExample

That’s why the warning a.patient != a.doctor was previously redundant — the sets were overlapping.

MedicalHistory and Conditions :

MedicalHistory0 and MedicalHistory1 belong to Person2 and Person1.

They contain conditions like HighBloodPressure, Allergies.

Treatments and instructions (like MedicationTreatment, Instruction) are also linked.

Interpretation

This instance satisfies your constraints such as:

Multiple patients

More than 3 appointments

At least one patient has more than one medical history

No self-treatment (a.patient != a.doctor)

Conclusion

Transforming a class diagram into an Alloy model involves a structured approach that ensures accurate representation of classes, relationships, and constraints in a formal, analyzable way. By following the step-by-step process—converting classes to signatures, attributes to fields, methods to predicates, and constraints to facts—we successfully modeled a healthcare system with entities like Patient, Doctor, Appointment, and MedicalHistory.

Key achievements of this modeling effort include:

Precise Representation: The Alloy model captures inheritance (e.g., Person → Doctor/Patient), associations (e.g., Appointment links Patient and Doctor), and business rules (e.g., no double bookings).

Verifiability: Using assertions and the Alloy Analyzer, we can automatically check for inconsistencies or undesired behaviors.

Scalability: The model can be extended with additional features (e.g., billing, prescriptions) while maintaining logical consistency.

The image generated by the Alloy Analyzer visually represents an example Instance that satisfies all the model's constraints. It helps verify that relationships are correctly formed and allows easy tracing of entities such as patients, their appointments, and medical histories.

This approach demonstrates how formal methods like Alloy can bridge the gap between conceptual class diagrams and executable specifications, enabling early detection of design flaws.

Future work could include:

Adding temporal logic for appointment scheduling

Modeling security constraints (e.g., patient data access)

Integrating with real-world healthcare data standards

By leveraging Alloy's relational logic and automated analysis, we've created a robust foundation for verifying and refining the healthcare system's design before implementation.

General Conclusion

General Conclusion

Optimizing permissions within a medical system presents a complex challenge. The protection of sensitive medical data is of paramount importance. Implementing robust access controls—such as strong authentication mechanisms, identity and access management (IAM), and user activity monitoring—is a fundamental first step toward safeguarding confidential information.

The selection of a suitable programming language for developing medical applications plays a critical role in ensuring the system's robustness, security, flexibility, and maintainability. Choosing a language that meets the stringent security requirements of the healthcare sector is essential for protecting patients' sensitive data from potential breaches.

Building on these foundational elements, we propose an approach for optimizing permissions within medical systems. This approach not only ensures compliance with the highest security standards but also delivers a seamless and efficient user experience, thereby reinforcing the protection of patient information.

By integrating strict security protocols, embracing advanced technologies, and making informed decisions in selecting development tools, this approach fosters the creation of IT environments that both safeguard patient privacy and enhance healthcare accessibility. Ultimately, such measures contribute to improved care quality and strengthened trust in digital medical systems.

References

References

1/Laudon, K. C., & Laudon, J. P. (2020). ***Management Information Systems: Managing the Digital Firm***. Pearson.

2/ O'Brien, J. A., & Marakas, G. M. (2011). *Introduction to Information Systems*. McGraw-Hill.

3/ Rainer, R. K., & Cegielski, C. G. (2014). *Introduction to Information Systems: Supporting and Transforming Business*. Wiley.

4/ World Health Organization (WHO). (2016). *Health Information Systems*. Retrieved from https://www.who.int (Accessed on 25/02/2025)

5/ HIPAA Journal. (2023). *What is HIPAA Compliance?* Retrieved from https://www.hipaajournal.com (Accessed on 25/02/2025)

6/ HealthIT.gov. (2023). *What is an Electronic Health Record (EHR)?* Retrieved from https://www.healthit.gov (Accessed on 25/02/2025)

7/ World Health Organization (WHO). (2016). *Electronic Health Records*. Retrieved from https://www.who.int (Accessed on 25/02/2025)

8/ HIPAA Journal. (2023). *What is HIPAA Compliance?* Retrieved from https://www.hipaajournal.com (Accessed on 25/02/2025)

9/ Thion, R. (2008). *Structuration relationnelle des politiques de contrôle d'accès : Représentation, raisonnement et vérification* [Doctoral dissertation, Institut National des Sciences Appliquées de Lyon].

11/ Embe Jiague, M. (2012). *Mise en œuvre de politiques de contrôle d'accès formelles pour des applications basées sur une architecture orientée services* [Doctoral dissertation, Université de Sherbrooke].

12/ Gabillon, A. (2013). *Contrôler les accès aux données numériques*. *La Revue de l'Électricité et de l'Électronique*, Société de l'Électricité, de l'Électronique et des Technologies de l'Information et de la Communication. https://hal.archives-ouvertes.fr/hal-02108021

13/ Blanc, M. (2006). *Sécurité des systèmes d'exploitation répartis : Architecture décentralisée de métapolitique pour l'administration du contrôle d'accès obligatoire* [Doctoral dissertation, Université d'Orléans]. https://tel.archives-ouvertes.fr/tel-00460610

14/ Cheaito, M. (2012). *Un cadre de spécification et de déploiement de politiques d'autorisation* [Doctoral dissertation, Université Toulouse III – Paul Sabatier].

15/ Tran Van, P. (2018). *Partage de documents sécurisés dans le Cloud Personnel* [Doctoral dissertation, Université Paris-Saclay]. https://tel.archives-ouvertes.fr/tel-01779315

16/ Thion, R. (2008). *Structuration relationnelle des politiques de contrôle d'accès : Représentation, raisonnement et vérification* [Doctoral dissertation, Institut National des Sciences Appliquées de Lyon].

17/ Younis, A. Y., et al. (2014). An access control model for cloud computing. *Journal of Information Security and Applications*.

18/ Brancourt, C. (2015). *Le contrôle de droit d'accès et la sécurité de vos systèmes*. Retrieved from https://www.synbioz.com/blog/tech/autorisation-et-droits-d-acces

19/ Dev-Code Lab. (2017). *Le top 10 de l'OWASP 2017 : les risques les plus critiques pour la sécurité des applications web*. Retrieved from https://lab.dev-code.fr/le-top-10-de-lowasp-2017-les-risques-les-plus-critiques-pour-la-securite-des-applications-web/

20/ Attiogbé, J. C. (2011). *Introduction à la modélisation formelle*. Université de Nantes.

21/ Jackson D, Software Abstraction, logic langage, and analysis, MIT Press, 2012.

22 Jackson D, Alloy: a lightweight object-modeling notation, ACM Transaction on software Engineering and Methodology (TOSEM), 2002.

23/ S. Malik and L. Zhang, Boolean Satisfiability from theoretical hardness to pratical success. Communication of the ACM, 2009..

24/ http://alloy.mit.edu 2, 10-03-2021.

25/ K. Biba., Integrity Consideration for Secure Computer Systems". The MITRE Corporation, 31-53. 1977.

26 www.uml.org, 2004-2021.

27 . Boulares S, Validation des politiques de sécurité par rapport aux modèles de contrôle d'accès-Département d'informatique et d'ingénierie-Université du Québec en Outaouais, Aout 2010.

28/ Valmari, A. (1998). The state explosion problem. In *Lectures on Petri Nets I: Basic Models* (pp. 429–528). Springer..

31/ Cunha, A. (2012). Bounded model checking of temporal formulas with Alloy. *arXiv preprint arXiv:1207.2746*.

32/ Vakili, A., & Day, N. A. (2012). Temporal logic model checking in Alloy. In *Abstract State Machines, Alloy, B, VDM, and Z* (pp. 150–163). Springer.

33/ Frias, M. F., López Pombo, C. G., Baum, G. A., Aguirre, N. M., & Maibaum, T. S. E. (2005). Reasoning about static and dynamic properties in Alloy: A purely relational approach. *ACM Transactions on Software Engineering and Methodology, 14*(4), 478–526.

34/ Jackson, D., Estler, H.-C., & Rayside, D. (2009). *The guided improvement algorithm for exact, general-purpose, many-objective combinatorial optimization*. MIT Computer Science and Artificial Intelligence Laboratory.

35/ Jackson, D. (2003). Alloy: A logical modelling language. *ZB, 2651*, 1.

36/ Cunha, A. (2014). Bounded model checking of temporal formulas with Alloy. In *International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z* (pp. 303–308). Springer.

37/ Jackson, D. (2003). Alloy: A logical modelling language. *ZB, 2651*, 1.

38/ Jackson, D. (2006). *Software Abstractions* (Vol. 2). MIT Press.

39/ Torlak, E., & Dennis, G. (2006). Kodkod for Alloy users. In *First ACM Alloy Workshop*, Portland, Oregon.

40/ Jackson, D. (2012). *Software Abstractions: Logic, Language, and Analysis*. MIT Press.

41/ Torlak, E., & Jackson, D. (2007). Kodkod: A relational model finder. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (pp. 632–647). Springer.

42/ Frias, M. F., López Pombo, C. G., Baum, G. A., Aguirre, N. M., & Maibaum, T. S. E. (2003). Taking Alloy to the movies. In *International Symposium of Formal Methods Europe* (pp. 678–697). Springer.

43/ Frias, M. F., López Pombo, C. G., Baum, G. A., Aguirre, N. M., & Maibaum, T. S. E. (2005). Reasoning about static and dynamic properties in Alloy: A purely relational approach. *ACM Transactions on Software Engineering and Methodology, 14*(4), 478–526.

44/ Frias, M. F., Galeotti, J. P., López Pombo, C. G., & Aguirre, N. M. (2005). DynAlloy: Upgrading Alloy with actions. In *Proceedings of the 27th International Conference on Software Engineering, ICSE '05* (pp. 442–451). ACM.

45/ Frias, M. F., López Pombo, C. G., Galeotti, J. P., & Aguirre, N. M. (2007). Efficient analysis of DynAlloy specifications. *ACM Transactions on Software Engineering and Methodology, 17*(1), 4.

46/ Craig, L. (2002). *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Unified Process* (2nd ed.). Prentice-Hall.

47/ Alan, D., Barbara, H., & David, T. (2010). *System Analysis and Design with UML* (3rd ed.). John Wiley & Sons.

48/ Ziniewicz, P., Malinowski, P., & Mnich, S. Z. (2010). Clinic department information system development. *Studies in Logic, Grammar and Rhetoric, 21*.

49/ Joseph, S. (2002). *Sams Teach Yourself UML in 24 Hours* (3rd ed.). Sams Publishing.

50/ Alloy Tools. (n.d.). *Alloy tutorials*. Retrieved from https://alloytools.org/tutorials

52/ Wikipedia. (n.d.). *Attribute-based access control*. Retrieved March 21, 2020, from https://en.wikipedia.org/wiki/Attribute-based_access_control

60/ Gabillon, A. (2013). *Contrôler les accès aux données numériques*. La Revue de l'Électricité et de l'Électronique. https://hal.archives-ouvertes.fr/hal-02108021

62/ American National Standards Institute. (2004). *American National Standard for Information Technology – Role-Based Access Control (ANSI INCITS 359-2004)*.

63/ Abahmane, O. (2015). *Contrôle de flux d'informations basé sur la granularité* [Master's thesis, Université du Québec en Outaouais].

65/ Ekran System. (2020). *RBAC vs ABAC*. Retrieved April 7, 2020, from https://www.ekransystem.com/en/blog/rbac-vs-abac