*International Journal of Innovative Computing, Information and Control*
Volume **21**, Number **4**, August **2025**

ICIC International ©2025 ISSN 1349-4198
pp. **973–986**

# CONFLICT ANALYSIS OF THE CHINESE WALL SECURITY POLICY MODEL USING EVENT-B

Thanh-Binh Trinh[1] and Ninh-Thuan Truong[2,*]

[1]Faculty of Computer Science
Phenikaa University
Nguyen Van Trac Street, Yen Nghia Ward, Ha Dong District, Hanoi 150000, Vietnam
binh.trinhthanh@phenikaa-uni.edu.vn

[2]School of Aerospace Engineering
VNU University of Engineering and Technology
Xuan Thuy, Cau Giay, Hanoi 122045, Vietnam
*Corresponding author: thuantn@vnu.edu.vn

Received December 2024; revised April 2025

Abstract. *The Chinese Wall security policy is a robust framework designed to control access policies within sensitive environments. It establishes barriers between different domains or sets of information to prevent conflicts of interest and unauthorized access. By implementing strict controls and restrictions, this policy ensures that individuals or entities are restricted from accessing information that could potentially lead to conflicts of interest or breach confidentiality. Despite its prevalence, the policy lacks a formal mathematical representation for analyzing and reasoning about its operational aspects and properties. This paper proposes an approach to translate the Chinese Wall security policy model into Event-B constructs, enabling detailed specification and verification of these properties. It then demonstrates how to validate security policy properties using the Rodin/Event-B platform, supported by automated proof obligations for these properties. The authors argue that this mechanism offers a promising alternative for addressing the limitations observed in some previous models, particularly regarding the verification conflicts of interest.*

**Keywords:** Access control, Chinese Wall security policy, Read rule, Write rule, Rodin, Event-B

1. **Introduction.** In today's increasingly interconnected world, where software is widely used in every aspect of human life, it demands high security and safety, especially in organizations where conflicts of interest and confidentiality are critical. To address these problems, various security models have been proposed, each with its strengths and limitations. One of the most well-known models is the Chinese Wall Security Policy Model (CWSPM), also known as the Brewer-Nash model [1], which is specifically designed to address conflicts of interest within organizations, making it particularly valuable in finance, law, and consulting industries. The CWSPM establishes *virtual walls* between different sets of information, preventing users from accessing data that could lead to conflicts with their current assignments. For example, if a user has access to sensitive information from one client, the model restricts access to information from competing clients, mitigating the risk of conflicts of interest.

Although CWSPM effectively prevents conflicts of interest, its implementation and maintenance can be challenging. Accurate classification and labeling of information are crucial to ensuring that the *virtual walls* function as intended. Additionally, the model

DOI: 10.24507/ijicic.21.04.973

lacks a comprehensive mathematical foundation for thoroughly analyzing and reasoning about its operations and properties [2, 3, 4, 5]. This deficiency can make it difficult to identify potential security flaws early in the software development process. Therefore, the CWSPM remains a subject of active research, with ongoing efforts to refine and expand the model to address the increasingly complex challenges of modern information security systems such as cloud security, big data, IoT, cyber forensics, and blockchain security. Addressing this gap through further research could significantly enhance the model's reliability and effectiveness, helping organizations better protect their sensitive information from conflicts of interest.

This paper proposes an approach to formalize the CWSPM using Event-B [6, 7], a formal method known for its strong capabilities in the specification and verification of a software system at the design phase. The approach allows for formal verification of security policies, enhancing reliability through rigorous mathematical proofs, and making it more suitable for large-scale access control systems, strengthening conflict of interest policies in dynamic environments. Additionally, the Rodin platform [8, 9] enhances automation by generating and validating proof obligations, minimizing manual effort, and increasing efficiency in security verification. Through these advantages, CWSPM formalized with Event-B not only strengthens conflict of interest policies but also enhances security verification, automation, and scalability in complex environments.

Unlike previous studies, our approach integrates formal verification into CWSPM using Event-B, ensuring logical consistency and automated validation. The work by Capozucca et al. [2] revisits the CWSPM, refining its semantics and utilizing automated theorem-proving tools to validate security properties. However, their approach does not offer a scalable method for large-scale security enforcement in dynamic access control systems. Arshad et al. [10] propose `XACML2mCRL2`, an automatic transformation of `XACML` policies into `mCRL2` for formal verification, yet it does not directly address conflict of interest policies.

Additionally, Nguyen et al. [11] introduce a robust conflict detection mechanism for `I2NSF` security policies, leveraging Attribute-Based Access Control (ABAC) to mitigate policy inconsistencies. While their model enhances policy conflict resolution, it lacks a formal proof-based validation mechanism that ensures correctness across policy changes. Liu et al. [12] introduce a conflict detection method for ABAC policies, enhancing security by identifying conflicting rules, but their approach does not support formal verification or automated policy validation. Similarly, Anderer et al. [13] optimize Role-Based Access Control (RBAC) using evolutionary algorithms, improving efficiency but lacking a model for conflict of interest resolution.

In contrast, our Event-B-based CWSPM formalization guarantees policy correctness and conflict resolution through theorem proving, making it more resilient to dynamic access control environments. The main contributions of our approach include 1) developing a formal representation of CWSPM, 2) implementing CWSPM in Event-B for precise specification, and 3) using Rodin, an Event-B platform, to analyze conflicts of interest within the CWSPM. The remainder of this paper is structured as follows. Section 2 reviews related work. Section 3 presents an overview of the CWSPM and the Event-B formal method. Section 4 presents our approach to formalizing and translating the CWSPM into Event-B constructs. The experimental results and evaluations of the approach are discussed in Section 5. Finally, Section 6 summarizes the findings and provides suggestions for future research.

## 2. **Related Work.**

The CWSPM has been widely studied as a means to mitigate conflicts of interest in access control environments. Various extensions, conflict detection

mechanisms, and formal verification techniques have been proposed to enhance its effectiveness and applicability. In the following, we will present an overview of relevant works related to policy enforcement, conflict detection, and formal verification.

The original Brewer-Nash model [1] was introduced to restrict access to conflicting datasets in industries like finance, law, and consulting. It dynamically updates user access permissions to prevent conflicts of interest. Capozucca et al. [2] refined CWSPMs semantics and applied automated theorem proving for security validation. However, their approach lacks scalability for dynamic and large-scale security enforcement.

Fehis et al. [4, 5] focused on enhancing the CWSPM by introducing additional layers of granularity in access control decisions. Their work explored the integration of context-aware policies that consider not only the user's access history but also the specific circumstances under which access is requested. This approach aims to improve the model's adaptability to dynamic environments, where traditional static access controls may be insufficient.

Lin [14, 15, 16] extended the CWSPM to better handle scenarios involving large-scale information systems, where the volume and complexity of data pose significant challenges to the model's implementation. This work emphasizes the need for automated tools to support the policy's enforcement, given the increasing demand for scalability and flexibility in modern organizations.

Mohamed et al. [17] proposed a variant of the CWSPM that incorporates elements of Role-Based Access Control (RBAC). This hybrid model leverages the strengths of both CWSPM and RBAC, allowing for more flexible management of user roles while maintaining the conflicts of interest protections inherent in the Chinese Wall framework. Mohamed et al.'s work highlights the potential benefits of combining multiple access control models to achieve a more comprehensive security solution.

Several studies have focused on conflict detection mechanisms to enhance policy enforcement reliability. Nguyen et al. [11] introduced a robust conflict detection framework for I2NSF security policies, integrating Attribute-Based Access Control (ABAC) to mitigate policy inconsistencies. While their model enhances conflict resolution, it does not include formal proof-based validation, which is essential for policy correctness assurance.

Similarly, Liu et al. [12] proposed a conflict detection mechanism for ABAC, focusing on identifying policy inconsistencies. However, their approach lacks formal verification and automated policy validation, limiting its effectiveness in preventing logical conflicts.

Formal verification methods have been widely explored to ensure security policy correctness. Arshad et al. [10] introduced XACML2mCRL2, a framework that automatically transforms XACML policies into mCRL2 for formal verification. While their approach ensures policy correctness, it does not explicitly address conflict-of-interest scenarios, which are critical in CWSPM-based systems.

In the domain of Role-Based Access Control (RBAC), Anderer et al. [13] optimized RBAC using evolutionary algorithms, improving efficiency. However, their approach does not provide a conflict-of-interest resolution model, making it less applicable to CWSPM-based policies.

Trinh et al. [18] analyzed conflicts of interest in RBAC using Event-B, allowing system designers to systematically specify and verify conflict-of-interest rules, ensuring security correctness at the design phase. By integrating Event-B modeling with Rodin-based validation, the approach enables early detection of policy inconsistencies. However, this study primarily focuses on RBAC-based conflicts and does not address the scalability challenges of the CWSPM in complex environments.

While prior research has refined CWSPM semantics, introduced conflict detection mechanisms, and explored formal verification, most studies lack full integration of theorem

proving into CWSPM. Compared to existing studies, our approach uniquely integrates CWSPM with Event-B, enabling formal verification, automated policy validation, and conflict resolution through theorem proving. Unlike previous research focusing on policy enforcement or detection, our method ensures scalability, correctness, and automated verification using Rodin, making it a robust solution for large-scale access control environments.

## 3. Background.

In this section, we first provide a brief introduction to the CWSPM. Following this, we present an overview of the Event-B formal method and the Rodin platform, which are utilized in the formalization and verification of security policies.

### 3.1. Chinese Wall security policy model.

The CWSPM [1], originating from the field of computer security, provides a framework for controlling access to sensitive information within organizations. The model's name refers to the concept of a "Chinese Wall", a metaphorical barrier designed to prevent the exchange of information between individuals or groups with conflicting interests. In the context of access control policies, the CWSPM aims to restrict access based on past interactions to prevent conflicts of interest or the unauthorized exchange of sensitive information.

To better understand the concept of the CWSPM, let us consider the scenario depicted in Figure 1 [19]. In this scenario, a financial consulting firm organizes its clients into three conflicts of interest (`COI`) classes, each representing a different industry sector, such as `COI-A` (Banking), `COI-B` (Pharmaceuticals), and `COI-C` (Technology). Each `COI` class contains multiple companies. The Chinese Wall security policy is applied to ensuring that consultants within the firm cannot access or transfer sensitive information between companies within the same `COI` class, thereby preventing conflicts of interest.

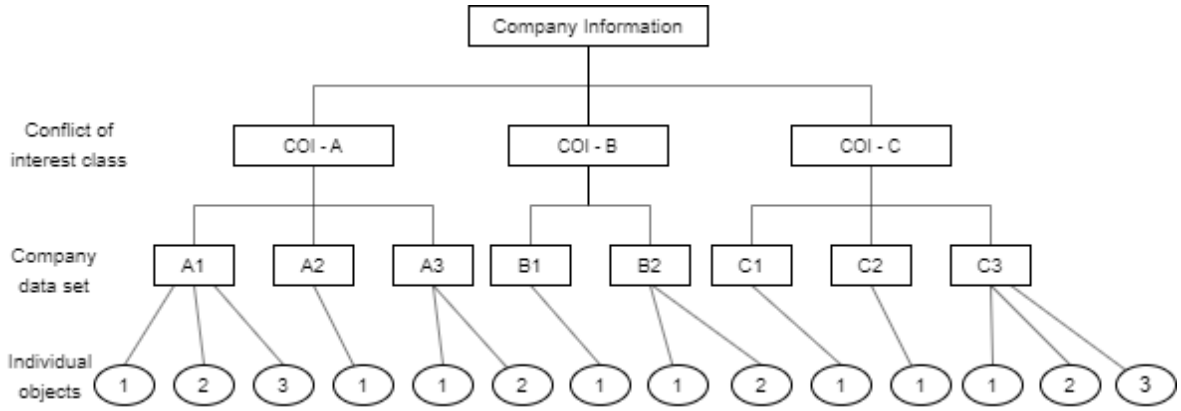


FIGURE 1. Concept of the Chinese Wall security policy model, illustrating the structure and classification of company information according to the `COI` classes and their associated data objects

For example, a consultant named Alice is assigned to work with Company A1 in `COI-A`. Under the Chinese Wall policy, Alice is restricted from working with any other companies within the same `COI` class, such as Companies A2 and A3, to prevent conflicts of interest. However, she is permitted to work with companies in different `COI` classes, such as Companies B1 and B2 in `COI-B` or Companies C1, C2, and C3 in `COI-C`, without any restrictions.

The primary goal of the CWSPM is to prevent the leakage of sensitive information from one company to another within the same `COI` class. The model enforces strict access control policies that ensure a consultant, or any individual with access privileges, cannot

have read or write access to multiple companies within the same `COI` class. Formally, the CWSPM defines its access policy through two rules for reading and writing as follows.

**Definition 3.1** (Read Rule)**.** *A consultant $C$ can read an object $O$ if and only if one of the following conditions holds:*

1) *$C$ can read another object $O'$ within the same `COI` class if $O'$ belongs to the same company as $O$ (e.g., both $O$ and $O'$ are within `COI-A` and belong to the same company, such as A1). This ensures that within a `COI` class, $C$ can only access multiple objects if they belong to the same company, preventing conflicts of interest within that company.*
2) *If $C$ has previously accessed any objects $O'$ and all such objects $O'$ belong to different `COI` classes than $O$ (e.g., $O'$ is in `COI-B`, and $O$ is in `COI-A`).*
3) *$O$ is a sanitized object, meaning it has been processed to remove any sensitive information, making it safe for general access.*

**Definition 3.2** (Write Rule)**.** *A consultant $C$ may write to an object $O$ if and only if both of the following conditions hold:*

1) *The consultant $C$ is permitted to read the object $O$ according to the Read Rule.*
2) *For all unsanitized objects $O'$ that $C$ can read within the same `COI` class, $O$ and $O'$ must belong to the same company. This means that $C$ can only write to an object if all other objects that $C$ can read within the same `COI` class are from the same company (e.g., both $O$ and $O'$ are in `COI-A` and belong to A1).*

The key similarity is that both sets of rules aim to maintain the integrity and confidentiality of sensitive information by strictly controlling access across conflicting interests. The Chinese Wall model specifically focuses on preventing conflicts of interest, while the given properties are more generally applicable to multi-level security environments.

3.2. **Event-B and the Rodin platform.** Event-B [6, 7] is a formal method that combines mathematical techniques from set theory and first-order logic, serving as both a notation and method for the formal development of discrete systems. It is an evolution of other formal methods like the B-Method (also known as classical B) [20], offering simplifications that make it easier to learn and more suitable for parallel and distributed reactive system development. Event-B models are described in terms of two basic constructs: *contexts* and *machines*.

The *context* forms the static part of the model, defining essential elements such as carrier sets, constants, and axioms. Carrier sets specify the types of elements (e.g., a set of signals like *SIGNAL* in a railway system), constants represent specific entities (e.g., *RED*, *GREEN*, *YELLOW* signals), and axioms establish logical relationships and constraints among them. Contexts can extend or be extended by other contexts, enabling reuse and ensuring consistency across the model. This static foundation supports the dynamic behaviors modeled by machines in Event-B.

The *machine* represents the dynamic part of the model, encompassing the system's state through variables, ensuring certain properties with invariants, deriving necessary truths with theorems, and modeling behavior with events. For example, in a railway signaling system, a machine might use a variable like *currentSignal* to represent the signal state, with invariants ensuring that the signal does not transition directly from *GREEN* to *RED* without passing through *YELLOW*. Events, structured as *evt = any x where* $G(x, v)$ *then* $A(x, v, v')$ *end*, define the conditions $G(x, v)$ under which state changes occur and the actions $A(x, v, v')$ that update the system state, such as transitioning the signal from *RED* to *GREEN*. The machine thus captures the system's dynamic interactions while ensuring adherence to specified constraints and behaviors.

Refinement in Event-B is the process of transforming an abstract model into a more concrete one as development progresses, adding detail and bringing it closer to implementation. For instance, in refining a railway signaling system, an abstract model with signals defined as *ON* or *OFF* can be refined to specify states like *RED*, *YELLOW*, and *GREEN*. New variables and events can be introduced, and existing ones refined, as long as they do not block existing events, often managed with a *variant*, a natural number that decreases with each event. Refinement also allows abstract events to be split into more detailed events or combined into one, such as refining a generic *change_signal* event into specific *turn_green*, *turn_yellow*, and *turn_red* events.

Composition and Decomposition in Event-B provide powerful mechanisms for managing complexity in system development, enabling both independent development and systematic integration of system components. Composition refers to the process of combining multiple components, such as machines or contexts, to form a complete system. This approach allows different parts of the system to be developed and verified independently before integration. On the other hand, Decomposition involves breaking down a complex system into smaller, more manageable subcomponents. This is particularly useful for handling large-scale systems, as it allows developers to focus on individual parts of the system independently.

The Rodin platform [8, 9] is an Integrated Development Environment (IDE) specifically designed for formally modeling and verifying systems using the Event-B formal method. Developed as part of the European RODIN project, the platform provides tools and functionalities to support the entire system development process, including automated proof, refinement, and extensibility through plugins. These features make Rodin a powerful choice for formal system development, particularly in fields where precision and correctness are critical.

## 4. Approach.

This section presents our approach to enhancing the CWSPM, including its formalization, the implementation using Event-B, and an evaluation of the experimental results.

### 4.1. Formalization of Chinese Wall security policy model.

The formalization aims to precisely define the CWSPM and its rules as a step toward translating it into the Event-B formal method. Using Event-B, we can systematically analyze and verify the model's properties, ensuring it meets the desired security requirements and effectively prevents conflicts of interest. To achieve this, we first need to formally define the key components of the CWSPM as in Definition 4.1.

**Definition 4.1** (CWSPM). *Let* $M = (S, C, O, \texttt{COI})$ *be a Chinese Wall security policy model, in which*

- $S = \{S_1, S_2, \ldots, S_n\}$*: The set of individuals (users, principals, subjects, etc.).*
- $C = \{C_1, C_2, \ldots, C_m\}$*: The set of companies.*
- $O = \{O_1, O_2, \ldots, O_k\}$*: The set of data objects where each* $O_i \in O$ *is associated with exactly one company* $C_j \in C$*.*
- $\texttt{COI}(O_i) \subseteq P(C)$*: The set of conflicts of interest where each* $\texttt{COI}(O_i)$ *is a subset of the power set of companies* $P(C)$*.*

With the components of the CWSPM defined, we can now establish the rules that govern access to data objects. Definition 4.2 introduces the Read Rule, which specifies the conditions under which a subject $S$ can be granted permission to read a data object $O_i$.

**Definition 4.2** (Read Rule)**.** *Let $S$ be the set of subjects that can be granted permission to read data object $O_i$, denoted by $canRead(S, O_i)$, defined as follows:*

$$canRead(S, O_i) \triangleq (\exists O_j \in sameCompany(O_i) \land canRead(S, O_j)) \lor (\forall O_j \in \texttt{COI}(O_i) \rightarrow \neg canRead(S, O_j))$$

The Read Rule defines the conditions under which a subject $S$ is allowed to access a data object $O_i$. Specifically, $S$ can read $O_i$ if they have previously accessed another object within the same company as $O_i$ or if $S$ has not accessed any objects within conflicting `COI` classes. This ensures that access is granted only in a way that prevents conflicts of interest, either by keeping access within a single company or ensuring that different `COI` classes do not overlap.

With the Read Rule in place, we can now establish the conditions for writing to data objects. Definition 4.3 introduces the Write Rule, which outlines when a subject $S$ can be granted permission to modify a data object $O_i$.

**Definition 4.3** (Write Rule)**.** *Let $S$ be the set of subjects that can be granted permission to write to a data object $O_i$, denoted by $canWrite(S, O_i)$, defined as follows:*

$$canWrite(S, O_i) \triangleq canRead(S, O_i) \land (\forall O_j \notin sameCompany(O_i) \rightarrow \neg canRead(S, O_j))$$

The Write Rule specifies that a subject $S$ can only write to a data object $O_i$ if they are already permitted to read $O_i$ and have not accessed any other unsanitized objects outside the company to which $O_i$ belongs. This rule ensures that writing permissions are restricted to prevent conflicts of interest by confining modifications within a single company and maintaining data integrity across `COI` classes.

While the Write Rule ensures that modifications to data objects are carefully controlled to prevent conflict of interest, time-based restrictions add another critical dimension to access control. Definition 4.4 introduces the concept of time frames, which further limits when a subject can read, write, or delete a data object, ensuring that access is not only secure but also time-bound.

**Definition 4.4** (Time Frames)**.** *Access to data objects is restricted based on predefined time intervals, ensuring that access permissions are valid only within specified periods. Let $timeFrames(o) \triangleq \langle start, end \rangle$ denote the time frame during which the object can be accessed. Let $currentTime \in \mathbb{N}$ be the current time. A subject $s \in S$ can perform an operation (read, write, delete) on a data object $o \in O$ only if the current time falls within the object's time frame $isWithinTimeFrames(o, t) \triangleq t \in timeFrames(o)$.*

This definition introduces the concept of time frames for accessing data objects. Access to data objects is restricted based on predefined time intervals, ensuring access permissions are only valid within specified periods. A subject $S$ can perform operations such as read, write, or delete on a data object $O$ only if the current time falls within the object's time frame.

From the definitions above, we have the following corollaries.

**Corollary 4.1.** $\forall (O_i, O_j) \mid (O_i \in sameCompany(O_j)) \rightarrow (O_j \in sameCompany(O_i) \land O_j \notin \texttt{COI}(O_i))$

**Proof:** According to the definition, $O_j \in sameCompany(O_i)$ means both $O_i$ and $O_j$ belong to the same company. By the definition of $sameCompany(O_i)$, if $O_i$ and $O_j$ are in the same company, then $O_i \in sameCompany(O_j)$. Additionally, if $O_j \in sameCompany(O_i)$, $O_j$ cannot belong to a `COI` class with $O_i$ because objects in the same company cannot be in conflict of interest. □

**Corollary 4.2.** $\forall(O_i, O_j) \mid (O_j \in \texttt{COI}(O_i)) \rightarrow (O_i \in \texttt{COI}(O_j))$

**Proof:** By definition, $O_j \in \texttt{COI}(O_i)$ means that $O_j$ is in the COI class with $O_i$. Since $\texttt{COI}(O_i)$ is a symmetric relation (if $O_j$ is in conflict with $O_i$, then $O_i$ is in conflict with $O_j$), it follows that $O_i \in \texttt{COI}(O_j)$. □

**Corollary 4.3.** $\forall(S, O_i, O_j)|(canRead(S, O_i) \wedge canRead(S, O_j) \wedge O_i \in sameCompany(O_j))$ $\rightarrow (canWrite(S, O_i) \wedge canWrite(S, O_j))$

**Proof:** If a subject $S$ can read both $O_i$ and $O_j$ and $O_i$ and $O_j$ are from the same company, then by Definition 4.3 (Write Rule), $S$ can be granted write access to both objects. This is because Definition 4.3 requires two conditions for write access: $S$ must have read access to $O_i$ and must not have read access to any object outside the same company as $O_i$. Since $O_i \in sameCompany(O_j)$ and $S$ only has read access to objects within the same company, both conditions are satisfied. Therefore, $S$ is allowed to write to both $O_i$ and $O_j$. □

**Corollary 4.4.**

$$\forall(S, O_i, O_j) \mid (canWrite(S, O_i) \wedge canWrite(S, O_j))$$
$$\rightarrow (O_i \in sameCompany(O_j) \vee O_i \notin \texttt{COI}(O_j))$$

**Proof:** If a subject $S$ has write access to both $O_i$ and $O_j$, then by Definition 4.3 (Write Rule), $S$ must have read access to both objects. Furthermore, for $S$ to be granted write access, $O_i$ and $O_j$ must either belong to the same company (i.e., $O_i \in sameCompany(O_j)$) or they must not belong to the same COI class (i.e., $O_i \notin \texttt{COI}(O_j)$). This ensures that there is no conflict of interest in granting $S$ write access. Therefore, $S$ can write to both $O_i$ and $O_j$ only if the objects are either from the same company or from different COI classes. □

These formalized definitions and corollaries provide a robust foundation for implementing and verifying the CWSPM in the Event-B formal method. By rigorously defining access control rules, we can ensure that potential conflicts of interest are effectively managed, and the integrity of sensitive information is maintained.

4.2. **Modeling the Chinese Wall security policy model using Event-B.** The CWSPM, as described in Section 3.1, can be translated into the Event-B formal method through the following steps. First, in Section 4.2.1, we define the abstract model, which provides the foundational structure of the CWSPM in Event-B. Next, Section 4.2.2 introduces the refinement model, specifically incorporating the concept of a time frame to add temporal dynamics and further details to the policy's implementation mechanisms.

4.2.1. *Abstract model.* In this abstraction, we begin with an abstract model of the CWSPM that focuses on the properties of the operations affecting the user. The security requirements of this level can be summarized as follows.

- Req 1: Information must be categorized into distinct conflict of interest classes (COIs). Each COI represents a group of companies or entities that are in competition or have conflicting interests.
- Req 2: Each piece of information (data object) must be associated with a specific company within a COI.
- Req 3: A user can only read $O_i$ if they have previously accessed another object within the same company as $O_i$, or if $S$ has not accessed any objects within conflicting COI classes.

- Req 4: A user can only write to a data object $O_i$ if they are already permitted to read $O_i$ and have not accessed any other unsanitized objects outside the company to which $O_i$ belongs.

To implement these requirements within the abstract model, we define a set of machine variables and invariants that ensure the properties of the CWSPM are maintained, as illustrated in Figure 2. The abstract machine defines two key variables: *read* and *write*. The invariants associated with these variables ensure that the above security requirements of the CWSPM are maintained. Specifically, *inv1* and *inv2* state that the *read* and *write* variables represent a relation between users and data, ensuring that each user can be associated with one or more data objects. Additionally, *inv3* and *inv4* ensure that a user cannot read from or write to multiple data objects belonging to conflicting companies within the same conflict of interest class. These invariants check that if a user has read access to one data object from a company, they cannot read or write any other data object from a different company in the same `COI`. In the context seen by this abstract machine, *USERS*, *DATA*, and *COMPANIES* are defined as carrier sets, while `COI` and *belong* are constants (see Figure 3). *axm1* states that `COI` is a subset of the power set of *COMPANIES*. In other words, each `COI` represents a group or collection of companies that are in competition or have conflicting interests. This grouping is essential to ensure that access controls can be applied based on the relationships between different companies, thereby preventing conflicts of interest. Additionally, *axm2* defines a relation called *belong* that links each data object in *DATA* to a specific company in *COMPANIES*. This ensures that every piece of data is associated with exactly one company, which is crucial for enforcing the CWSPM. By knowing which company a data object belongs to, the system can apply appropriate access restrictions based on the user's previous access history and the associated `COI`.

Following the definition of the machine variables and context, we now introduce the *read* and *write* events. These events are crucial for implementing the operations that users

**Variables:** *read*, *write*
**Invariants:**
*inv1:* $read \in USERS \leftrightarrow DATA$
*inv2:* $write \in USERS \leftrightarrow DATA$
*inv3:* $\forall u, d1, d2, c1, c2 \cdot (u \in USERS \land d1 \in DATA \land d2 \in DATA \land c1 \in COMPANIES \land c2 \in COMPANIES \land (u \mapsto d1 \in read) \land (d1 \mapsto c1 \in belongs) \land (d2 \mapsto c2 \in belongs)) \Rightarrow \neg(u \mapsto d2 \in read)$
*inv4:* $\forall u, d1, d2, c1, c2 \cdot (u \in USERS \land d1 \in DATA \land d2 \in DATA \land c1 \in COMPANIES \land c2 \in COMPANIES \land (u \mapsto d1 \in write) \land (d1 \mapsto c1 \in belongs) \land (d2 \mapsto c2 \in belongs)) \Rightarrow \neg(u \mapsto d2 \in write)$

FIGURE 2. Machine variable and invariants of an abstract model

**Sets:**
USERS, DATA, COMPANIES
**Constants:**
`COI`, belong
**Axioms:**
*axm1:* $\texttt{COI} \subseteq \mathcal{P}(COMPANIES)$
*axm2:* $belongs \in DATA \leftrightarrow COMPANIES$

FIGURE 3. A context of an abstract model

can perform on data objects while ensuring that the CWSPM is consistently maintained. Event *evt_Read*: This event is designed to grant a user read access to a specific data object. In the event, the parameters $u$ and $d$ represent the user and the data object, respectively. The guards (*grd1*, *grd2*, and *grd3*) ensure that the user is allowed to read the data object. Specifically, *grd1* checks that the user is part of the set *USERS*, and *grd2* ensures the data object is within the *DATA* set. *grd3* enforces that the user can only read the data object if it does not conflict with any previously accessed objects within the same `COI` class. This guard is critical in maintaining the integrity of the CWSPM by preventing access to conflicting data. Once the conditions are satisfied, the action *act1* updates the read relation to include the new user-data pairing.

| **Event** *evt_Read* | **Event** *evt_Write* |
|---|---|
| **Any** | **Any** |
| $u, d$ | $u, d$ |
| **Where** | **Where** |
| *grd1:* $u \in USERS$ | *grd1:* $u \in USERS$ |
| *grd2:* $d \in DATA$ | *grd2:* $d \in DATA$ |
| *grd3:* $\exists d1, c1, c2 \cdot ((u \mapsto d1) \in read \land (d1 \mapsto c1) \in belongs \land (d \mapsto c2) \in belongs \land c1 \neq c2)$ | *grd3:* $\neg(\exists d1, c1, c2 \cdot ((u \mapsto d1) \in write \land (d1 \mapsto c1) \in belongs \land (d \mapsto c2) \in belongs \land c1 \neq c2))$ |
| | *grd4:* $(u \mapsto (d \mapsto \text{read})) \in permissions$ |
| **Then** | **Then** |
| *act1:* $read := read \cup \{(u \mapsto d)\}$ | *act1:* $write := write \cup \{(u \mapsto d)\}$ |
| **End** | **End** |

FIGURE 4. Event read and write of an abstract model

Event *evt_Write*: This event is designed to grant a user write access to a data object. Similar to the *evt_Read* event, the parameters $u$ and $d$ represent the user and the data object, respectively. *grd1* ensures that the user is a valid member of *USERS*, and *grd2* verifies that the data object is a valid member of *DATA*. Additionally, *grd3* ensures that the user is not attempting to write to a data object from a company that conflicts with another company for which the user already has write access, while *grd4* confirms that the user has read access to the data object before being granted write access. These guards are essential to ensure that the user does not violate the conflict of interest constraints established by the policy. If all guards are satisfied, the action *act1* updates the write relation to reflect the new write access.

4.2.2. *Refinement model: Time frame, permission.* In this refinement, the concept of a time frame is introduced to the CWSPM. This refinement adds temporal dynamics to the model, allowing for more complex and realistic scenarios where user access rights can change over time. Three additional machine variables are introduced at this level: $timeFrame$ (representing the duration for which a user's access is restricted within a `COI`), $currentTime$ (tracking the last time a user accessed a data object within a `COI`), and permission. These variables are crucial for implementing time-based restrictions and ensuring that the policy adapts to changes over time. Additionally, the *evt_Read* and $evt\_Write$ events from the abstract model are refined to incorporate these new temporal constraints. The additional requirements for this level are outlined below.

- Req 1: Each data object must have an associated time frame, specifying the start and end times during which the object can be accessed.

- Req 2: The system must continuously track the current time, to determine whether it falls within the predefined time frame of a data object.
- Req 3: A subject can only perform operations (read, write) on a data object if the current time is within the object's time frame.

This refinement enhances the model by allowing the system to manage access not just based on conflicts of interest but also within temporal boundaries, making it more adaptable to real-world use cases where access permissions may need to expire or change over time. Figure 5 presents a list of machine variables and invariants formulated to satisfy the above requirements. In this refinement, new variables such as $timeFrames$, $currentTime$, and permissions are introduced to incorporate the concept of time-based access control. *inv5* ensures that each data object has an associated time frame, defined as a pair of start and end times, while *inv6* tracks the current time within the system. *inv7* defines the permissions relation, which keeps track of the users and their permitted operations (read or write) on data objects. Refinement of Event *evt_Read*: In this refinement, as illustrated in the model (Figure 6), additional details are introduced to enhance the original *evt_Read*. Specifically, *grd4* is added to ensure that the data object $d$ is accessible within a valid time frame. This guard checks that $d$ is part of the domain of time frames and verifies that the current time falls within the start and end times associated with $d$. These conditions refine the original event by incorporating temporal constraints, ensuring that the read operation can only be performed when the time frame is valid. Furthermore,

**Variables:** $timeFrames$, $currentTime$, $permissions$
**Invariants:**
*inv5:* $timeFrames \in DATA \rightarrow (\mathbb{N} \times \mathbb{N})$
*inv6:* $currentTime \in \mathbb{N}$
*inv7:* $permissions \subseteq USERS \times (DATA \times \{read, write\})$

FIGURE 5. Machine variables and invariants of the refinement model

| **Event** *evt_Read* | **Event** *evt_Write* |
|---|---|
| **Any** | **Any** |
| $u, d, start, end$ | $u, d, start, end$ |
| **Where** | **Where** |
| *grd1:* $u \in USERS$ | *grd1:* $u \in USERS$ |
| *grd2:* $d \in DATA$ | *grd2:* $d \in DATA$ |
| *grd3:* $(\exists d1, c1, c2 \cdot ((u \mapsto d1) \in read \land (d1 \mapsto c1) \in belongs \land (d \mapsto c2) \in belongs \land c1 \neq c2))$ | *grd3:* $\neg(\exists d1, c1, c2 \cdot ((u \mapsto d1) \in write \land (d1 \mapsto c1) \in belongs \land (d \mapsto c2) \in belongs \land c1 \neq c2))$ |
| *grd4:* $d \in dom(timeFrames) \land currentTime \geq start \land currentTime \leq end$ | *grd4:* $(u \mapsto (d \mapsto read)) \in permissions$ |
| | *grd5:* $d \in dom(timeFrames) \land currentTime \geq start \land currentTime \leq end$ |
| **Then** | **Then** |
| *act1:* $read := read \cup \{(u \mapsto d)\}$ | *act1:* $write := write \cup \{(u \mapsto d)\}$ |
| *act2:* $permissions := permissions \cup \{(u \mapsto (d \mapsto read))\}$ | *act2:* $permissions := permissions \cup \{(u \mapsto (d \mapsto write))\}$ |
| **End** | **End** |

FIGURE 6. Event read and write of the refinement model

*act2* updates the permissions relation to reflect the new read access, ensuring that the permissions are consistently managed within the system.

Refinement of Event *evt_Write*: Similarly, the *evt_Write* event is refined to include additional conditions, as shown in the model. *grd5* is introduced to ensure that the data object $d$ is within its designated time frame during the write operation. This guard verifies that $d$ is part of the time frames' domain and checks that the current time is within the specified start and end times. This refinement adds a temporal constraint to the write operation, ensuring that users can only modify data objects when their time frames permit it. Additionally, *act2* in the *evt_Write* event updates the permissions relation to reflect the new write access, thus maintaining the integrity and security of the system by incorporating time-based restrictions. This ensures that even after a write operation, access control is aligned with the temporal boundaries set for each data object. At the same time, the combination of conflict-of-interest rules and time constraints enhances the security and flexibility of the system.

## 5. **Results and Discussion.**

5.1. **Experimental results.** Figure 7 illustrates the implementation of CWSPM in Event-B. In this figure, the `CM_Machine` is associated with a comprehensive set of invariants (*inv1* through *inv9*) and events, including `INITIALISATION`, `evt_Read`, `evt_Write`, `evt_Delete`, and `evt_UpdateTime`. These events and invariants are meticulously designed to ensure the correctness and strict enforcement of the CWSPM, crucial for preventing conflicts of interest within the system.

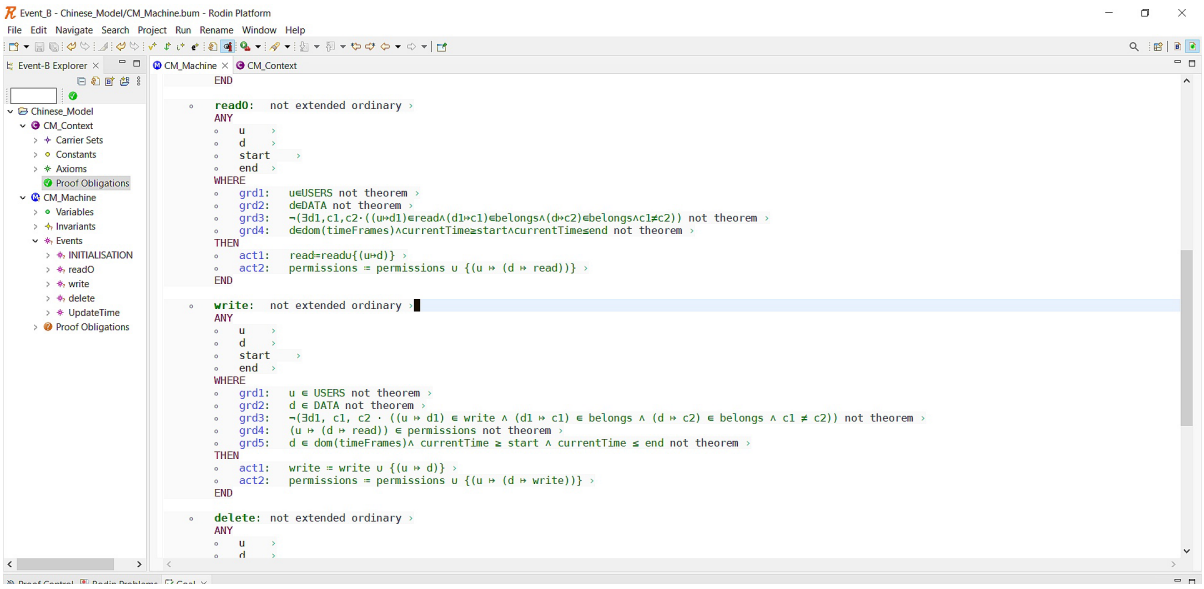


FIGURE 7. The CWSPM implemented in Event-B

The figure also displays the experimental results obtained for the `CM_Machine` model using the Rodin platform. According to the proof statistics, the Rodin platform generated 20 proof obligations. Of these, 12 (or 60%) were proved automatically, while the remainder were discharged through interactive proof[1].

These results confirm that while the Event-B formalization of CWSPM is largely effective, there remains a significant portion of the model that requires detailed manual

[1]The full source code and proof obligations of the model can be found at https://github.com/binhtt/Chinese_Model

verification. This underscores the importance of continuous refinement and the potential need for enhanced automation techniques within the Rodin platform to further reduce the reliance on interactive proof. The findings contribute to a better understanding of the strengths and limitations of the current approach and suggest directions for future improvements in the formalization and verification of complex security policies.

5.2. **Evaluation and discussion.** The formalization of CWSPM using Event-B demonstrated a high degree of accuracy, thanks to the rigorous mathematical framework provided by Event-B. Through experimental results, we confirmed that the model strictly adhered to the security principles of CWSPM, effectively detecting and handling potential conflicts of interest. This shows that using Event-B significantly enhances the reliability of the model compared to non-formal methods.

Despite the positive evaluation results, the process encountered some challenges, particularly in proving properties within large-scale models. Fully automating the proof process remains a challenge. We suggest enhancing the level of automation in supporting tools like Rodin and exploring the integration of other security models, such as RBAC or ABAC, to develop more flexible and comprehensive security solutions.

6. **Conclusion and Future Works.** In this paper, we propose an approach to model the CWSPM using the Event-B formal method. By translating CWSPM into Event-B constructs, we can clearly define and verify key aspects of the policy, particularly the read-and-write rules, which are essential for managing conflicts of interest. The formalization ensures that the security requirements are met with a high degree of precision, and the use of the Rodin platform helped automate proof obligations, improving the overall reliability of the model.

The experimental results confirmed that the model effectively ensured the management of conflicts of interest, as evidenced by the successful generation of the necessary proof obligations. However, some proof obligations still require manual intervention. This suggests that future work should aim to increase the level of automation increasing the level of automation within the Rodin platform to reduce the need for manual proof discharges, thereby streamlining the verification process. Additionally, exploring hybrid models that integrate the CWSPM with other access control frameworks such as RBAC or ABAC could offer more flexible and comprehensive solutions for managing access in complex environments.

## REFERENCES

[1] D. F. C. Brewer and M. J. Nash, The Chinese Wall security policy, *IEEE Symposium on Security and Privacy*, pp.206-214, 1989.

[2] A. Capozucca, M. Cristiá, R. Horne and R. Katz, Brewer-Nash scrutinised: Mechanised checking of policies featuring write revocation, *2024 IEEE 37th Computer Security Foundations Symposium (CSF)*, pp.112-126, 2024.

[3] T. Y. Lin, Chinese Wall security policy models: Information flows and confining Trojan Horses, *IFIP Advances in Information and Communication Technology*, vol.142, pp.275-287, 2003.

[4] S. Fehis, O. Nouali and T. Kechadi, A new distributed Chinese Wall security policy model, *Journal of Digital Forensics, Security and Law*, 2016.

[5] S. Fehis, O. Nouali and T. Kechadi, Secure encryption key management as a SecaaS based on Chinese wall security policy, *Journal of Information Security and Applications*, vol.63, 102975, 2021.

[6] J.-R. Abrial, A system development process with Event-B and the Rodin platform, in *Lecture Notes in Computer Science*, Springer, Berlin, Heidelberg, 4789, 2007.

[7] J.-R. Abrial, *Modeling in Event-B – System and Software Engineering*, Cambridge University Press, 2010.

[8] J.-R. Abrial, M. Butler, S. Hallerstede, T. S. Hoang, F. Mehta and L. Voisin, Rodin: An open toolset for modelling and reasoning in Event-B, *STTT*, vol.12, no.6, 2010.

[9] J. Coleman, C. Jones, I. Oliver, A. Romanovsky and E. Troubitsyna, RODIN (Rigorous open Development Environment for Complex Systems), *The 5th European Dependable Computing Conference: EDCC-5 Supplementary*, 2005.
[10] H. Arshad, R. Horne, C. Johansen, O. Owe and T. A. C. Willemse, XACML2mCRL2: Automatic transformation of XACML policies into mCRL2 specifications, *Science of Computer Programming*, vol.232, 103046, 2024.
[11] D. D. A. Nguyen, F. Autrel, A. Bouabdallah and G. Doyen, A robust approach for the detection and prevention of conflicts in I2NSF security policies, *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*, Miami, FL, USA, pp.1-7, 2023.
[12] G. Liu, W. Pei, Y. Tian, C. Liu and S. Li, A novel conflict detection method for ABAC security policies, *Journal of Industrial Information Integration*, vol.22, 100200, 2021.
[13] S. Anderer, T. Kempter, B. Scheuermann and S. Mostaghim, Dynamic optimization of role concepts for role-based access control using evolutionary algorithms, *SN Comput. Sci.*, vol.4, no.4, 2023.
[14] T. Y. Lin, Chinese Wall security policy – An aggressive model, *Proc. of the 5th Annual Computer Security Applications Conference*, pp.282-289, 1989.
[15] T. Y. Lin, Placing the Chinese Walls on the boundary of conflicts – Analysis of symmetric binary relations, *Proc. of the 26th Annual International Computer Software and Applications*, pp.966-971, 2002.
[16] T. Y. Lin, Chinese Wall security model and conflict analysis, *Proc. of the 24th Annual International Computer Software and Applications Conference (COMPSAC2000)*, pp.122-127, 2000.
[17] A. Mohamed, D. Auer, D. Hofer and J. Küng, A systematic literature review for authorization and access control: Definitions, strategies and models, *International Journal of Web Information Systems*, vol.18, no.8, 2022.
[18] T.-B. Trinh, V.-K. To, N.-T. Truong and H. A. Le, Analysing conflict of interest integrated in role-based access control model using Event-B, *Intelligence of Things: Technologies and Applications*, pp.57-72, 2024.
[19] S. C. Mouliswaran, C. A. Kumar and C. Chandrasekar, Modeling Chinese Wall access control using formal concept analysis, *2014 International Conference on Contemporary Computing and Informatics (IC3I)*, pp.811-816, 2014.
[20] J. R. Abrial, *The B-Book: Assigning Programs to Meanings*, Cambridge University Press, 1996.

## Author Biography

**Thanh-Binh Trinh** obtained his Ph.D. degree in Software Engineering from VNU University of Engineering and Technology, Vietnam, in 2012. He is currently the Program Director for Software Engineering at Phenikaa University, Hanoi, Vietnam. His research primarily focuses on software engineering, including formal methods, software verification and validation, and model-driven development.

**Ninh-Thuan Truong** obtained his Ph.D. degree in Computer Science from Nancy 2 University, France, in 2006. He is currently an Associate Professor at the VNU University of Engineering and Technology, Hanoi. His main research interests include software engineering, formal methods, and software security.