Specifying and Verifying Organizational Security Properties in First-Order Logic

Christoph Brandt[1], Jens Otten[2], Christoph Kreitz[2], and Wolfgang Bibel[3]

[1] Université du Luxembourg
christoph.brandt@uni.lu
[2] University of Potsdam
{jeotten,kreitz}@cs.uni-potsdam.de
[3] Darmstadt University of Technology
bibel@gmx.net

Abstract. In certain critical cases the data flow between business departments in banking organizations has to respect security policies known as Chinese Wall or Bell–La Padula. We show that these policies can be represented by formal requirements and constraints in first-order logic. By additionally providing a formal model for the flow of data between business departments we demonstrate how security policies can be applied to a concrete organizational setting and checked with a first-order theorem prover. Our approach can be applied without requiring a deep formal expertise and it therefore promises a high potential of usability in the business.

Keywords: theorem proving, first-order-logic, Chinese Wall, Bell–La Padula, organizational data-flow, security, leanCoP.

1 Introduction

One of the primary interests of a banking organization is maintaining its reputation. Among other issues this involves avoiding any kind of violation of its mandatory security policies. Due to a lack of precise security models that can be applied in practice, security is primarily implemented as a risk management activity, which is informal in nature and uses rather fuzzy guidelines. Although these practices seem to imply a sound measurement of risks from a methodological point of view, only rough qualitative statements can be made, which do not lead to truly reliable security guarantees.

In this paper we will demonstrate that the shortcomings of the "best-practices" can be overcome in a way that is sound, efficient and usable. We will describe how formal models for security policies and IT landscapes can be built in a way that makes it possible to integrate them in a way transparent to people in the business. We will also show how formal methods can be used to verify security properties within specific IT landscapes. We use first-order logic as the language for representing the formal models and the first-order theorem prover leanCoP [26,24] as the formal verification tool.

Specifically, we will discuss two security policies, the Chinese Wall and the Bell–La Padula policy, in the context of a formally modeled IT landscape that needs to be secured. We will focus on a small example of an IT landscape derived from a concrete scenario at Credit Suisse. It consists of two business departments and the data-flow between them. It is assumed that the data-flow represents the email traffic between the departments, which fits the organizational reality at major banking organizations.

The novelty presented in this paper is an engineering-like approach to addressing a real-world problem with the help of formal methods, their implementations and corresponding formal models. Our approach makes it possible to add further security policies and different IT landscapes as separate modules and to combine them in a plug-and-play manner whenever this makes sense. This compositional nature fits well the expectations of people in the business, because it enables them to use the full power of implemented formal methods in an encapsulated way without requiring them to have a deep formal expertise.

The paper is organized as follows. Section 2 describes our approach to assuring security policies in a given IT landscape that enables us to check such policies against different organizational settings. In Sections 3 and 4 we will explain and formally specify the Chinese Wall and Bell–La Padula security policies. Section 5 presents a formal model of a simplified IT landscape, which provides a basis for automated security checks. Section 6 introduces the formal methods and tools that are used in a case study in Section 7, where we check that a concrete set of security constraints implements the Chinese Wall policy in the scenario of Section 4. Section 8 discusses related work and in Section 9 we will draw final conclusions and discuss possibilities for future work.

2 A Generic Approach to Assuring Security

The purpose of this paper is to show how security policies and IT landscapes can be integrated in a formal model that enables practitioners to verify organizational security properties. In the real world, banking organizations run different service landscapes that need to respect a multiplicity of security policies. As a consequence, there is a need for a set of small, well focused, and composable models that can be handled by existing formal tools. As service landscapes and security policies are usually specified in a complementary fashion, the respective modeling processes must be independent and lead to orthogonal models that can be soundly integrated. Obviously, the same formal methods should be used in both activities in order to make a combination of the results possible.

According to Hartel, van Eck, Etalle, and Wieringa [15], security policies like the Chinese Wall are rules that constrain the behavior of a system in order to accomplish a certain security principle, such as “information about a customer must not leak to other customers”. Thus a well-defined security principle can be formally checked if a formal specification of the system and the organizational security policy is given. In [15], this verification task is described by the following statement.

$$(\mathit{system} \parallel \mathit{security\ policy}) \models \mathit{security\ principle}\ . \tag{1}$$

Statement (1) can be verified using a formal tool, using an appropriate representation of the system, the security policy, and the security principle in the formal language of that tool. If the verification fails, Hartel, van Eck, Etalle, and Wieringa propose to adjust either the system or the security policy to prove the given security principle.

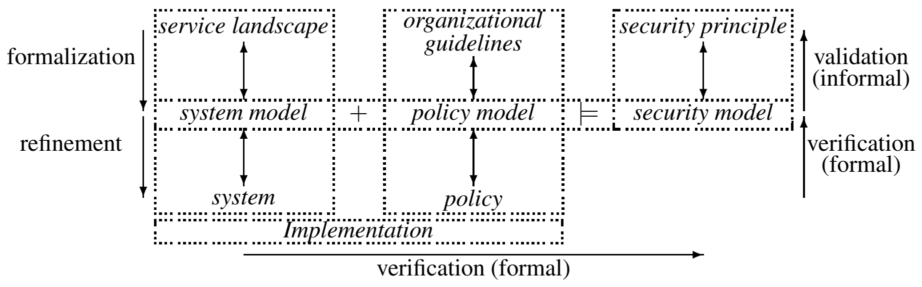

Fig. 1. Generic approach to verifying security properties

In the real world, however, security policies often come as textual documents describing organizational guidelines that have to be refined into a formal policy model. Since organizational policy documents cannot be formalized automatically, generating a formal policy model requires interpreting the meaning of informal organizational guidelines in a way that can easily be validated by security experts. This view is inspired by the work of Freeman, Neely, and Heckard, who have developed a validated security policy modeling approach [12] that can be applied to many types of systems, including networks and distributed systems. Their approach is driven by security requirements and by system architecture and supports working with orthogonal models that can be combined easily and checked automatically. As a consequence, two statements have to be validated in addition to statement (1).

$$\textit{organizational guidelines} \leftrightarrow \textit{policy model} \tag{2}$$

$$\textit{security principle} \leftrightarrow \textit{security model} \tag{3}$$

Our approach will go even one step further. Since banking organizations may use different service landscapes, different policies, and different implementations of the same policy (or landscape), we will use high-level abstractions of system models, policy models, and security models in a verification and link these to the low level representations of the actual system and the policy enforcement code, as illustrated in Figure 1.

This makes it possible to verify abstract properties of a variety of service landscapes and security policies with the same formal tool without being burdened by details of a specific implementation. Verifying that the models are fulfilled by their respective implementations will have to be done separately and, if successfully carried out, will then automatically prove that the concrete systems and policy enforcement mechanisms satisfy the desired security principle. As a consequence, one can combine multiple service landscapes, security policies, and realizations of these in a plug-and-play fashion and provide security guarantees with the help of a formal tool. This makes our approach flexible enough to cope with the dynamic nature of business while keeping the implementation of the formal methods fixed. It is cost effective because modules can be exchanged and reused in other contexts and verified automatically. It is mathematically sound and promises a high potential for being used by people in the business. The reuse potential of the formal models is an important reason to justify possible upfront investments for building them.

In their work on reasoning about security policies Halpern and Weissman [14] show that a formalism used for this purpose must not only be expressive and usable by non-logicians, but also be sufficiently tractable, since otherwise verification tools may not scale well and thus become useless in practice. According to these authors another important issue is identifying and resolving conflicts that may occur when different policy sets are merged, which is often the case in business scenarios.

In consequence of all these conceptual deliberations, our approach will use first-order logic to specify the system and policy models in an orthogonal fashion that allows for a later integration. Relevant aspects of a service landscape as well as the organizational guidelines of the security policy will be represented in the form of first-order axioms. The security principle to be verified must be formulated as a theorem in first-order logic. Then using a first-order theorem prover to derive the theorem from the set of axioms will verify that the security principle holds in the given service landscape and for the given security policy. Finally, formalizing the system and the policy enforcement mechanisms as first-order axioms as well will enable the theorem prover to verify their correctness with respect to the models and thus prove that the security principle is satisfied by the implementation of the system and of the security policy.

3 The Chinese Wall Model

A Chinese Wall, also known as Brewer and Nash Model [8], is most commonly employed in investment banks between the corporate-advisory area and the brokering department in order to separate those giving corporate advice on takeovers from those advising clients about buying shares. The " wall" shall prevent leaks of corporate inside information, which could influence the advice given to clients making investments, but allow staff to take advantage of facts that are not yet known to the general public.

Maintaining client confidentiality is crucial to any firm, but particularly to large multi-service businesses. If a firm provides a wide range of services, clients must be able to trust that information about themselves will not be exploited for the benefit of other clients. Therefore clients must be able to trust in Chinese Walls. In recent years, however, some Wall Street scandals have made people doubt the effectiveness of Chinese Walls, as executives of respectable firms have traded illegally on inside information for their own benefit.

The term Chinese Wall was popularized in the United States following the stock market crash of 1929, when the U.S. government legislated information separation between investment bankers and brokerage firms, in order to limit the conflict of interest between objective analysis of companies and the desire for successful initial public offerings. Rather than forbidding one firm from engaging in both businesses, the regulation permitted the implementation of Chinese Wall procedures.

Our formalization of the Chinese Wall policy is built upon the legal specifications, which are expressed by three security requirements. These requirements are mapped onto formal constraints that have to be proved in order to verify an implementation of the Chinese Wall policy. We use the general notion of a log file (l) to record actions (get, new, $drop$, put) of subjects (s) who send data objects (ob) to each other at certain points in time (t).

Requirement CW 1. *Once a subject has accessed an object the only other objects accessible by that subject lie within the same company data set or within a different conflict of interest class.*

The formalization of this requirement in first-order logic states that if an entry $\mathit{element}(s, \mathtt{get}, \mathit{ob}, t)$ of the log file l shows that a subject s gets an object ob at some time t (or creates a new one) and if s does not drop the object again at some time t', then the subject can get another object ob' at some time t'' afterward if and only if ob and ob' are not in a conflict of interest class. No restrictions apply if the new object carries sanitized information.

Requirement CW 2. *A subject can access at most one company data set in each conflict of interest class.*

More precisely, if a subject s gets an object ob and doesn't drop it again, it cannot get an object ob' from a different owner in the same class of conflict. This restriction does not apply to objects carrying sanitized information.

The following two first-order formulas reflect the intentions expressed by the first and the second requirement. Since the two requirements partly overlap, the formulas do not formalize them in a one-to-one fashion, but reorganize the underlying conditions in a way that helps keep the formalizations as simple as possible.

$$\begin{gathered}\forall s\ \forall \mathit{ob}\ \forall \mathit{ob}'\ \forall i\ \forall t\ \forall t'\ \forall t''\ \forall l\\ (((\mathit{member}(\mathit{element}(s, \mathtt{get}, \mathit{ob}, t), l) \vee \mathit{member}(\mathit{element}(s, \mathtt{new}, \mathit{ob}, t), l))\\ \wedge \neg \mathit{member}(\mathit{element}(s, \mathtt{drop}, \mathit{ob}, t'), l) \wedge t{<}t' \wedge\ t'{\leq}t'' \wedge\\ (\mathit{info}(i, \mathit{ob}') \wedge \mathit{sanitized}(i)) \vee \neg \mathit{conflict}(\mathit{ob}, \mathit{ob}'))\\ \Rightarrow\ \mathit{permitted}(s, \mathit{get}(\mathit{ob}', t'', l)))\end{gathered} \tag{4}$$

$$\begin{gathered}\forall s\ \forall \mathit{ob}\ \forall \mathit{ob}'\ \forall i\ \forall t\ \forall t'\ \forall t''\ \forall l\\ (((\mathit{member}(\mathit{element}(s, \mathtt{get}, \mathit{ob}, t), l) \vee \mathit{member}(\mathit{element}(s, \mathtt{new}, \mathit{ob}, t), l))\\ \wedge \neg \mathit{member}(\mathit{element}(s, \mathtt{drop}, \mathit{ob}, t'), l) \wedge t{<}t' \wedge\ t'{\leq}t'' \wedge\\ \mathit{info}(i, \mathit{ob}') \wedge \neg \mathit{sanitized}(i) \wedge \mathit{conflict}(\mathit{ob}, \mathit{ob}'))\\ \Rightarrow\ \neg \mathit{permitted}(s, \mathit{get}(\mathit{ob}', t'', l)))\end{gathered} \tag{5}$$

Requirement CW 3. *The flow of unsanitized information is confined to its own company data set; sanitized information may however flow freely throughout the system.*

If a subject s gets an object ob (and doesn't drop it again) and creates a new object ob' afterwords then the new object is allowed to be put back if and only it has the same ownership as the first one or any information shared by the two objects is sanitized.

We use two formulas to represent the "if" and the "only if" parts separately, as this simplifies the verification process.

$$\begin{gathered}\forall s\ \forall \mathit{ob}\ \forall \mathit{ob}'\ \forall i\ \forall i'\ \forall t\ \forall t'\ \forall t''\ \forall t'''\ \forall t''''\ \forall l\\ ((\mathit{member}(\mathit{element}(s, \mathtt{get}, \mathit{ob}, t), l) \wedge \neg \mathit{member}(\mathit{element}(s, \mathtt{drop}, \mathit{ob}, t'), l)\\ \wedge\ \mathit{member}(\mathit{element}(s, \mathtt{new}, \mathit{ob}', t''), l) \wedge \neg \mathit{member}(\mathit{element}(s, \mathtt{drop}, \mathit{ob}', t'''), l)\\ \wedge\ t{<}t' \wedge t'{\leq}t'' \wedge\ t''{<}t''' \wedge\ t'''{\leq}t'''' \wedge\ \mathit{ob}{\neq}\mathit{ob}' \wedge\ \mathit{info}(i, \mathit{ob}) \wedge\ \mathit{info}(i', \mathit{ob}')\\ \wedge\ (\mathit{sameowner}(\mathit{ob}, \mathit{ob}') \vee \mathit{sanitized}(i') \vee i{\neq}i'))\\ \Rightarrow\ \mathit{permitted}(s, \mathit{put}(\mathit{ob}', t'''', l)))\end{gathered} \tag{6}$$

$$\begin{array}{c}\forall s\, \forall ob\, \forall ob'\, \forall i\, \forall i'\, \forall t\, \forall t'\, \forall t''\, \forall t'''\, \forall t''''\, \forall l \\ ((\mathit{member}(\mathit{element}(s, \texttt{get}, ob, t), l) \wedge \neg \mathit{member}(\mathit{element}(s, \texttt{drop}, ob, t'), l) \\ \wedge\, \mathit{member}(\mathit{element}(s, \texttt{new}, ob', t''), l) \wedge \neg \mathit{member}(\mathit{element}(s, \texttt{drop}, ob', t'''), l) \\ \wedge\, t<t' \wedge\, t'\leq t'' \wedge\, t''<t''' \wedge\, t'''\leq t'''' \wedge\, ob\neq ob' \wedge\, \mathit{info}(i, ob) \wedge\, \mathit{info}(i', ob') \\ \wedge\, \neg \mathit{sameowner}(ob, ob') \wedge\, \neg \mathit{sanitized}(i') \wedge\, i=i') \\ \Rightarrow\, \neg \mathit{permitted}(s, \mathit{put}(ob', t'''', l)))\end{array} \tag{7}$$

4 The Bell–La Padula Model

The Bell–La Padula security policy aims at securing the confidentiality of information. Its underlying idea is to prevent subjects with a given security level from reading data with a higher security level than their own. This is called the *no read-up principle*. It also prevents a subject from writing to any object of a security level lower than its own (*no write-down principle*). Finally, it specifies the discretionary access control through an access matrix (*need-to-know principle*). Later, lattice- or compartment-based models were introduced to realize horizontal and vertical data segments.

The Bell–La Padula security policy dates back to 1973 when David Elliot Bell and Leonard J. La Padula were working on a security model on behalf of the US Air Force with the intention to protect confidential data [4]. Formally, the model is based on the concept of a state machine representing a set of allowable states in a computer network system. The model divides entities into subjects and objects and defines the notion of a secure state. It has been shown that state transitions preserve the security properties by moving from secure states to secure states.

A system state is defined to be secure if the only permitted access modes of subjects to objects are in accordance with the Bell–La Padula policy. To determine whether a specific access mode is permitted, the clearance level of a subject is compared to the classification of the object. The clearance scheme is expressed in terms of a lattice. The no read-up and the no write-down principle define two mandatory access control (MAC) rules, and the access matrix realizes a discretionary access control (DAC) rule.

Information may be transferred from a classified document to an unclassified one by the use of trusted subjects. Trusted subjects are not restricted by the no write-down principle, but untrusted subjects are. The latter can only create content at or above their own security level and they can only view content at or below their own security level.

It is instructive to compare the Chinese Wall and the Bell–La Padula security policy using similar terms as used by Brewer and Nash, who argue that the Bell–La Padula model is not able to simulate the Chinese Wall model. Brewer and Nash point out, for example, that in the real world management may suddenly require user A to have access to a company data-set X if user B is unavailable. However, it is not possible to change the need-to-know of user A to the one of user B unless user A has had no access to another conflicting company data-set before. Otherwise, the Chinese Wall security policy would be violated. Brewer and Nash argue that the Bell–La Padula model is not able to provide the necessary information to answer this question. They also argue that the Bell–La Padula security policy only works under the assumption that subjects are not given the freedom to choose which company data-set they wish to access. In Brewer and Nash's formalization of the Bell–La Padula policy the freedom of choice can only be restored at the expense of failing to express the mandatory controls. As a consequence, the Bell–La Padula model can be used to model either the mandatory part or the free of choice part of the Chinese Wall security policy, but not both at the same time. As the Brewer and Nash model cannot simulate all aspects of the Bell–La Padula security policy, both security policies need to be treated independently but applied simultaneously in case they have to be enforced together in a given scenario.

By using one common formalization for both security policies we can represent the Bell–La Padula properties by two formal constraints on a service landscape, which again can be checked by a first-order theorem prover.

Requirement BLP 1 (Simple security). *Access is granted only if the subject's clearance is greater than the object's classification and the subject's need-to-know includes the object's category(ies).*

This requirement can be translated immediately into two first-order formulas.

$$\begin{gathered}\forall s\,\forall ob\,\forall c\,\forall l\\ ((seclevel(ob)\leq seclevel(s)\wedge needtoknow(ob,s)\\ \wedge\,(category(c,ob)\Rightarrow category(c,s)))\\ \Rightarrow\; permitted(s,get(ob,t,l)))\end{gathered}\tag{8}$$

$$\begin{gathered}\forall s\,\forall ob\,\forall c\,\forall l\\ ((seclevel(ob)>seclevel(s)\vee\neg needtoknow(ob,s)\\ \vee\,\neg(category(c,ob)\Rightarrow category(c,s)))\\ \Rightarrow\;\neg permitted(s,get(ob,t,l)))\end{gathered}\tag{9}$$

Requirement BLP 2 ($*$-property). *Write access is granted only if the output object's classification is greater or equal to the classification of all input objects, and its category includes the category(ies) of all input objects.*

If a subject s gets or creates an object ob (and doesn't drop it again) and later creates a new object ob' that shares information with ob then the new object is allowed to be put back if and only if its security level is greater or equal to the one of ob and if its category includes the category of ob. Again, we use two formulas to represent this requirement.

$$\begin{gathered}\forall s\,\forall ob\,\forall ob'\,\forall c\,\forall i\,\forall i'\,\forall t\,\forall t'\,\forall t''\,\forall t'''\,\forall t''''\,\forall l\\ (((member(element(s,\mathtt{get},ob,t),l)\vee member(element(s,\mathtt{new},ob,t),l))\\ \wedge\,\neg member(element(s,\mathtt{drop},ob,t'),l)\wedge member(element(s,\mathtt{new},ob',t''),l)\\ \wedge\,\neg member(element(s,\mathtt{drop},ob',t'''),l)\\ \wedge\,t<t'\wedge\; t'\leq t''\wedge\; t''<t'''\wedge\; t'''\leq t''''\;\wedge\; info(i,ob)\wedge info(i',ob')\wedge\;(i\neq i'\\ \vee\,(i=i'\wedge\; seclevel(ob)\leq seclevel(ob')\wedge(category(c,ob)\Rightarrow category(c,ob'))))\\ \Rightarrow\; permitted(s,put(ob',t'''',l)))\end{gathered}\tag{10}$$

$$\begin{gathered}\forall s\,\forall ob\,\forall ob'\,\forall c\,\forall i\,\forall i'\,\forall t\,\forall t'\,\forall t''\,\forall t'''\,\forall t''''\,\forall l\\ (((member(element(s,\mathtt{get},ob,t),l)\vee member(element(s,\mathtt{new},ob,t),l))\\ \wedge\,\neg member(element(s,\mathtt{drop},ob,t'),l)\wedge member(element(s,\mathtt{new},ob',t''),l)\\ \wedge\,\neg member(element(s,\mathtt{drop},ob',t'''),l)\\ \wedge\,t<t'\wedge t'\leq t''\wedge\; t''<t'''\wedge\; t'''\leq t''''\wedge\; info(i,ob)\wedge info(i',ob')\;\wedge\;(i=i')\\ \wedge\,(seclevel(ob)>seclevel(ob')\vee\;\neg(category(c,ob)\Rightarrow category(c,s)))\\ \Rightarrow\;\neg permitted(s,put(ob',t'''',l)))\end{gathered}\tag{11}$$

5 The Service Landscape Model

In [2], F. Arbab introduces the notion of abstract behavior types (ABT) and Reo connectors as a model for system components and their composition. ABTs provide a well-defined semantics based on constraint automata and are therefore suited to automatic model checking. On the other hand they can be used in a very intuitive way by the people in the field. The model supports exogenous coordination and synchronous communication by abstracting away the internal implementation of components, focusing only on their observable outside behavior. Hence, formally pre-defined connectors and components can be simply clicked together as graphical objects conforming a formal graph grammar for this domain-specific modeling language.

ABTs can be seen as a high-level alternative to abstract data types. They define an abstract behavior as a relation among a set of timed-data-streams without specifying the operations or data types that may be used to implement such a behavior. Therefore, ABT models allow for a much looser coupling of components than ADTs and support formal models of a message driven service landscape that fit the Chinese Wall policy's conceptual understanding of messages being sent between different subjects.

The example IT landscape that we will formalize here is a simplified version of a concrete scenario at Credit Suisse. There are two departments, A and B, which are able to send data objects to each other by separate connectors. For instance, department A is the corporate-advisory area and department B is the brokering department. The data-flow between the two departments is realized by two separate Reo connectors as shown in Figure 2.

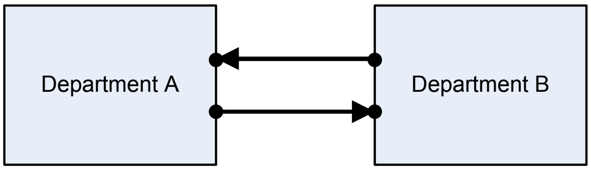

Fig. 2. Data-flow between department A and B

In our model, the different departments are represented by their separate abstract behavior types shown as blocks in Figure 2. Formally, the Reo channels conform to $Sync$ channels, realized as a $Sync$ ABT, and the abstract behavior types representing the two departments are realized as *department-A* ABT and *department-B* ABT. The operational semantics of a $Sync$ ABT is defined as $\langle \alpha, a\rangle Sync\langle \beta, b\rangle \equiv \langle \alpha, a\rangle = \langle \beta, b\rangle$, where, as shown in [2], the $Sync$ ABT represents the behavior of any entity that (1) produces an output data stream identical to its input data stream ($\alpha = \beta$), and (2) produces every element in its output at the same time as its respective input element is consumed ($a = b$). The operational semantics of a *department-A* ABT is defined as

department-A $ABT(\langle \alpha_i, a_i\rangle; \langle \alpha_o, a_o\rangle) \equiv (a_i < a'_i) \wedge (a_o < a'_o) \wedge (a_i \neq a_o)$,

which means that there is an input and an output timed-data stream representing put and get operations at a certain point in time. Operations are atomic. They happen sequentially and do not overlap. As we focus on the behavior, the concrete data

(α_i, α_o) is not specified further. The operational semantics of *department-B* is defined analogously.

This specification fits well with our formal specification of the Chinese Wall policy. To describe which agent is able to get data-objects from another one based on the given ABT/Reo model, we introduce two first-order axioms, which specify that it is possible for data-objects to be sent between both departments.

Axiom 1. *If department A has created a new object that still has not been dropped or put elsewhere, department B can get it, assuming that department A can send objects to department B as shown in Figure 2.*

$$\begin{array}{c}\forall ob\ \forall t\ \forall t'_1\ \forall t'_2\ \forall t'' \\ ((member(element(A, \texttt{new}, ob, t), l) \wedge \neg member(element(A, \texttt{drop}, ob, t'_1), l) \\ \wedge\ \neg member(element(A, \texttt{put}, ob, t'_2), l)\ \wedge\ t{<}t'_1\ \wedge\ t'_1{\le}t''\ \wedge\ t{<}t'_2\ \wedge\ t'_2{\le}t'') \\ \Rightarrow\ possible(B, get(ob, t'', l)))\end{array} \tag{12}$$

Axiom 2. *If department B has created a new object that still has not been dropped or put elsewhere, department A can get it, assuming that department B can send objects to department A as shown in Figure 2.*

$$\begin{array}{c}\forall ob\ \forall t\ \forall t'_1\ \forall t'_2\ \forall t'' \\ ((member(element(B, \texttt{new}, ob, t), l) \wedge \neg member(element(B, \texttt{drop}, ob, t'_1), l) \\ \wedge\ \neg member(element(B, \texttt{put}, ob, t'_2), l)\ \wedge\ t{<}t'_1\ \wedge\ t'_1{\le}t''\ \wedge\ t{<}t'_2\ \wedge\ t'_2{\le}t'') \\ \Rightarrow\ possible(A, get(ob, t'', l)))\end{array} \tag{13}$$

6 Methods and Tools

In order to prove first-order formulas that occur in our verification process we will use the automated theorem prover leanCoP. leanCoP is a very compact implementation of the connection calculus, a popular proof search method for first-order logic.

6.1 The Connection Calculus

There are several proof search calculi to automate formal reasoning in first-order logic; see [11] for an introduction. Connection calculi [5,6,20] are particularly successful due to the goal-oriented proof search. Their main inference step connects an atomic formula of the conjecture, or an atomic formula of the proof derivation, to a new atomic formula with the same predicate symbol but different polarity. This pair of atomic formulas $\{A, \neg A\}$ is called a connection and corresponds to a closed branch in the tableau framework [11] or an axiom in the sequent calculus [13].

The standard version of the connection calculus requires the input formula to be in disjunctive normal form or *clausal form*, i.e. of the form $C_1 \vee \ldots \vee C_n$ where C_i is a clause of the form $L_1 \wedge \ldots \wedge L_n$, L_i is a literal of the form A or $\neg A$ where A is an atomic formula. A formula in clausal form can be written as a set of clauses $\{C_1, \ldots, C_n\}$ and is called a *matrix*. In the graphical representation of a matrix, its clauses are arranged horizontally, while the literals of each clause are arranged vertically.

Example 1. Let $(((\exists x\, Q(x) \vee \neg Q(c)) \Rightarrow P) \wedge (P \Rightarrow (\exists y\, Q(y) \wedge R))) \Rightarrow (P \wedge R)$ be a (first-order) formula. Its equivalent clausal form, in which y is replaced by the Skolem term b, is $(P \wedge R) \vee (\neg P \wedge Q(x)) \vee (\neg Q(b) \wedge P) \vee (\neg Q(c) \wedge \neg P) \vee (P \wedge \neg R)$. The matrix of this formula is

$$M_1 = \{\{P, R\}, \{\neg P, Qx\}, \{\neg Qb, P\}, \{\neg Qc, \neg P\}, \{P, \neg R\}\}$$

where some parentheses are omitted for simplicity. It consists of five clauses and has the following two-dimensional graphical representation:

$$\begin{bmatrix} P & \neg P & \neg Qb & \neg Qc & P \\ R & Qx & P & \neg P & \neg R \end{bmatrix} .$$

A *connection* contains two literals of the form $\{A, \neg A\}$. A *path* through a matrix $M = \{C_1, \ldots, C_n\}$ is a set that contains one literal from each clause, i.e. $\cup_{i=1}^{n} \{L_i'\}$ with $L_i' \in C_i$. A *first-order substitution* σ is a mapping from the set of first-order variables to the set of terms. In $\sigma(L)$ all variables of the literal L are substituted according to the substitution σ.

Example 2. In the matrix M_1 of Example 1 the sets $\{P, \neg P\}$, $\{R, \neg R\}$, $\{Qx, \neg Qb\}$ and $\{Qx, \neg Qc\}$ are connections. $\{P, \neg P, \neg Qb, \neg Qc, \neg R\}$ and $\{R, Qx, \neg Qb, \neg Qc, P\}$ are, e.g., paths through the matrix M_1. $\sigma(x) = c$ is, e.g., a first-order substitution.

According to the *deduction theorem*, a (conjecture) formula C is a logical consequence of a set of (axiom) formulas $\{A_1, \ldots, A_n\}$ if and only if the formula $(A_1 \wedge \ldots \wedge A_n) \Rightarrow C$ is logically valid. Hence, we can use logical calculi, such as the connection calculus, in order to determine if a given first-order formula is logically valid.

The *matrix characterization* [6] of classical validity can be seen as the underlying basis of the connection calculus. This characterization can be extended to some non-classical logics as well [18,34]. Let M^μ be the matrix M, in which copies of clauses have been added according to the *multiplicity* $\mu : M \rightarrow \mathbb{N}$.

Matrix characterization. *A matrix M is valid iff there exists a multiplicity μ, a first-order substitution σ and a set of connections $\mathcal{C}$, such that every path through M^μ contains a complementary connection of $\mathcal{C}$, i.e. a connection $\{A_1, \neg A_2\} \in \mathcal{C}$ with $\sigma(A_1) = \sigma(A_2)$.*

The connection calculus uses a *connection-driven* search strategy in order to calculate an appropriate set of connections $\mathcal{C}$. In each inference step a connection is identified along an active (sub-)path and only paths not containing the active path and this connection will be considered afterward. See [5,6,25] for more details. Proof search in the connection calculus is carried out by first applying the *start rule* and then repeatedly applying the *reduction* or the *extension rule*. The latter rules identify a connection $\{A_1, \neg A_2\}$ with $\sigma(A_1) = \sigma(A_2)$. An unification algorithm for calculating the first-order substitution σ is given in, e.g., [21].

Example 3. A proof in the connection calculus for the matrix M_1 of Example 1, using the graphical matrix representation, is given in Figure 3. It illustrates the seven proof steps required for a proof of M_1. The literals of each connection are connected with a line. The literals of the active path are boxed. While the extension steps connect a literal to a new clause (step 1, 2, 4, 5, and 6), the reduction steps connect to literals of the active path (step 3 and 7). With the first-order substitution $\sigma(x') = \sigma(x'') = c$ all paths through M_1 contain a complementary connection. Hence, the matrix M_1 and the original formula given in Example 1 are valid.

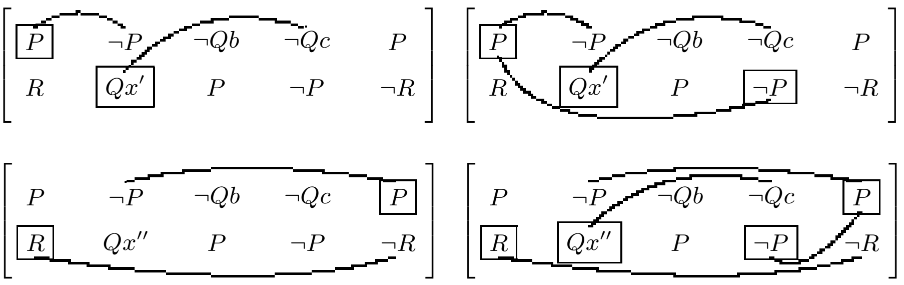
P $\neg P$ $\neg Qb$ $\neg Qc$ P
R Qx' P $\neg P$ $\neg R$
P $\neg P$ $\neg Qb$ $\neg Qc$ P
R Qx'' P $\neg P$ $\neg R$

Fig. 3. A (graphical) proof in the connection calculus

6.2 The Automated Theorem Prover leanCoP

leanCoP [26,24] is an automated theorem prover for classical first-order logic with equality[1]. It is a very compact implementation of the connection calculus. The reduction rule of the connection calculus is applied before the extension rule. Open branches are selected depth-first and iterative deepening on the proof depth is performed in order to achieve completeness. Additional inference rules include regularity and lemmata [20], as well as restricted backtracking [25]. leanCoP uses an optimized structure-preserving transformation into clausal form and a fixed strategy scheduling [25].

leanCoP is implemented in Prolog and uses Prolog's built-in indexing mechanism to find connections quickly. Although the source code of the core proof search algorithm consists of only one Prolog predicate that is a few lines long, leanCoP shows a strong performance and is currently the fastest implementation of a connection calculus [25]. leanCoP has won several prizes at the CADE System Competition (CASC), a yearly competition for fully automatic theorem provers for first-order logic. E.g., at this years CASC-J5, leanCoP-Ω, an extended version of leanCoP, won the new TFA division that contains problems in first-order logic with arithmetic [31].

leanCoP can read input formulas in leanCoP syntax as well as in TPTP syntax [30]. Equality axioms are automatically added if required. The leanCoP core prover returns a very compact connection proof, which is translated into a readable proof. Several output formats are available. Version 2.2 of leanCoP supports the output of proofs in the new TPTP syntax for representing derivations in connection calculi [27]. SETHEO [19] and KoMeT [7] are other high-performance implementations of the connection calculus. SETHEO was, e.g., successfully used to verify large software systems [28].

[1] The leanCoP theorem prover is available under the GNU general public license. It can be downloaded from the leanCoP web site at `http://www.leancop.de`

7 The Case

In the following we will use the formal specification of the Chinese Wall policy and prove that the first-order formulas defining its operational semantics are in line with the expected high level behavior of the policy as stated in Section 3. For our case we will choose formula (4) of Requirement CW 2 of the Chinese Wall specification. We will give a brief overview of the used predicates, explain the implementation of this requirement in the corresponding leanCoP format, present certain actions we took to optimize the proof search and discuss the obtained result.

The operational semantics has been specified in 97 first-order formulas. The predicates used are `admissibleL`, `eappend`, `emember`, `info`, `sanitized`, `conflict` and `permitted`. `admissibleL` plays the role of a type guard. It makes sure that, in case the second parameter is set to `yes`, every log file `L` is well-formed according to the rules of the Chinese Wall policy. The predicate `eappend` is used to append two log files and takes into account that log files contain only elements that have a certain structure. Correspondingly, the predicate `emember` is true if a log file contains a given element when the third parameter is set to `yes`. When the third parameter is set to `no`, the predicate `emember` is true if a given log file does *not* contain the specified element. Using the additional parameter `yes`/`no`, instead of using the logical negation, ensures better control when using the predicate within other definitions. The predicate `info` defines the relation between concrete *information* and *data*-objects carrying it. The predicate `sanitized` is true if the given information is sanitized; otherwise it is false. The predicate `conflict` implements the conflict behavior between any two data objects that can be created, sent and received in a Chinese Wall scenario by the different subjects. It is true if there is a conflict between two data objects; otherwise it is false. The predicate `permitted` is true if, for a given log file, it is permitted to add the given element describing a specific operation.

Formula (4) of Requirement CW 2 in leanCoP format is presented in Figure 4. It can be decomposed into three different parts. The first part contains one `admissibleL` and two `eappend` predicates and ensures that the log file is well-formed. The second part

```
all OB: all OB1: all I: all T: all T1: all T2: all L:
all TT1: all TT2: all TT3: all L1:
all OPT1: all OPT2: all OBB1: all OBB2: all HEAD: all TAIL:
( ( (T1<=T2), (TT1<TT2), (TT2<=T2), admissibleL(L,yes),
    eappend([element(OPT1,OBB1,TT1)],L1,L),
    eappend(HEAD,[element(OPT2,OBB2,TT3)|TAIL],L),
    ( emember(element(new,OB,T),L,yes), (T<T1) ;
      emember(element(get,OB,T),L,yes), (T<T1) ),
    emember(element(drop,OB,T1),L,no),
    ( (info(I,OB1), sanitized(I)) ; ~ conflict(OB,OB1) )
  ) => permitted(get,OB1,T2,L,yes)
)
```

Fig. 4. Chinese Wall requirement of formula (4) in leanCoP format

specifies that the log file contains an element with a `new` operation or an element with a `get` operation regarding an object `OB`. It further states that there was no `drop` operation of this object since then. The third part of the formula ensures that there is no conflict between the object `OB` and another object `OB1`. This is true if the other object carries sanitized information only, or if there is no direct conflict between the two objects. Compared to the original formula (4) in Section 3, the order of some predicates have been changed in order to optimize the proof search. For the same reason the current formalization considers only one subject s.

The formula shown in Figure 4 has been proved fully automatically using the automated theorem prover leanCoP. The translated input formula consists of 185 clauses. leanCoP finds a proof in 80 seconds, performing more than 10 million inference steps during the proof search. The final proof consists of 148 proof steps and is output in a readable form. The proof is non-trivial and shows that fully automated tools are necessary in order to find proofs for these kinds of problems. Table 1 contains a summary of the statistics for the proof found by leanCoP.

By default leanCoP uses a fixed strategy scheduling, where different proof search strategies are consecutively invoked in order to increase the chance to find a proof. If the given formulas have a specific form, specific strategies can be switched on or off in order to speed up the proof process. It turned out that for our formulas the strategy "`[def,conj,cut]`" is the most successful one (see [24] for details). It uses a definitional transformation into clausal form, starts the proof search with the conjecture clauses, and uses restricted backtracking, a technique that restricts the search space in a very effective way [25]. Hence, these settings were also used to find the proof for the requirement given in Figure 4.

During the development of the formal specification many adjustment were necessary, in order to ensure correctness of the given specification. This process is probably as difficult as the proof process itself. But once the formal specification is done, it can be used to prove similar requirements with only little additional effort.

Table 1. Some statistics about the proof found by leanCoP

Number of formal axioms	97
Number of input clauses	185
Number of required inference steps	10.687.197
Number of proof steps in final proof	148
Time required to find the proof	80 seconds

The obtained result is an example of a formal verification of a part of the policy model based on the given policy as introduced in Figure 1. It helps to countercheck the top-down development of the policy by a bottom-up verification.

8 Related Work

Although it has been long established in the scientific community that integrated models and tools are needed to address security issues [9], research solutions often have focused on technical issues. There is no holistic approach taking care of the interplay between the

IT and the business universe. Besides a generic interest in integrating different models, there is the claim that different concerns should be covered by specific models [12] to reduce the overall complexity for each single model. As a result, architectural models and security requirements are usually handled separately, even if there might be an integrated view at the end. In order to be useful in practice, models have usually been required to be suitable to model checking and simulation [17], which allows for an automatic evaluation of their security properties. Therefore models often have to reduce the level of details that are available to the formal analysis [33]. There are only few examples where first-order reasoning is used to deal with issues of real-world business applications [29].

Regarding the universe of models there are suggestions to look at the physical, logical and business layer in a separate way and to differentiate between a static build-time and a dynamic run-time view [10,22]. Regarding the IT landscape there are suggestions to look at the security issues of components and the ones of the overall system in a separate way [16] or to separate the handling of policies and security principles [1]. In addition to classical logic used to reason about security models, fuzzy reasoning and fuzzy measurement can be used to cover additional aspects in the security domain [35]. From an architectural point of view there are proposals to extract security functionality out of single applications making it available as general security services [3].

In today's security research literature and in the practical field, there is a strong focus on basic technical issues like authentication, authorization, confidentiality and accountability. However, there is not much regarding the semantic or pragmatic aspects of security. Semantics comes into play when, for example, customer data and deposit data are not allowed to be displayed on the same screen at a time. Encryption does not help here, nor do authentication, authorization, confidentiality or accounting. Pragmatics show up when executing single operations of IT or business services is completely legal. However, a certain combination or thread of such operations might produce a situation that is not compliant with legal regulations. For example, opening a set of accounts to transfer money between them, which is a thread of business operations to fog the history of money's origin, might produce such a violation. The pragmatic aspect can be seen as a certain property of a material business dialog happening between a bank's employee and an IT service that is used [36]. The sound interplay between the business and IT universe is likewise not sufficiently elaborated yet [32].

9 Conclusion

The global financial crisis of the past few years has many different aspects which have been analyzed in a rapidly growing literature. With a few exceptions [23] the role of the IT systems involved in the financial business practices was thereby neglected despite the fact that the lack of transparency of those systems was not helpful in monitoring business policy issues that showed up as one of many reasons leading to this crisis.

International politics is now discussing the introduction of regulatory rules to prevent a future crisis of such a dimension. However, the way IT technology is used to automate business processes in the banking industry is currently very product-oriented. A methodological view is often neglected because of high operational pressures. By addressing methodological questions in a way that could lead to solutions highly usable by people in the field we have shown that the benefits of using formal tools enforcing business policies as well as legal policies become compelling. As a consequence the presented approach will help to lead to new opportunities that support the implementation of legal regulations in the banking industry.

Our approach is a first proof of concept demonstrating how fully implemented formal methods may support the enforcement of policies in a way that is integrated with the ongoing business. As such, it will help addressing some accountability, risk and security issues that cannot be handled successfully today. Therefore, it will support the reduction of risk factors that have shown to be of some importance in the context of the current crisis. Our contribution is just a first step into this direction. There is a lot more that needs to be done, such as providing the remaining horizontal and vertical verifications from Figure 1 for the Chinese Wall policy, verifying other security policies and IT landscapes, and finally creating a verification infrastructure that could be used by skilled people in the business. Nevertheless, our contribution built on top of first-order logic might be a step towards opening the door into such a future development of the financial system, because the methodology underlying our approach does have the full potential for the realization of a future development of this kind.

References

1. Al-Shaer, E.S., Hamed, H.H.: Management and translation of filtering security policies. In: IEEE International Conference on Communications, vol. 1, pp. 256–260 (2003)
2. Arbab, F.: Abstract Behavior Types: A Foundation. Model for Components and Their Composition. Science of Computer Programming 55(1-3), 3–52 (2005)
3. Baltatu, M., Lioy, A., Mazzicchi, D.: Security policy system: status and perspective. In: IEEE International Conference on Networks, pp. 278–284 (2000)
4. Bell, D., La Padula, L.: Secure Computer Systems: Unified Exposition and Multics Interpretation. MTR-2997, The MITRE Corporation, Bedford, MA (1975)
5. Bibel, W.: Matings in Matrices. Communications of the ACM 26, 844–852 (1983)
6. Bibel, W.: Automated Theorem Proving. Vieweg (1987)
7. Bibel, W., Brüning, S., Egly, U., Rath, T.: KoMeT. In: Bundy, A. (ed.) CADE 1994. LNCS(LNAI), vol. 814, pp. 783–787. Springer, Heidelberg (1994)
8. Brewer, D., Nash, M.: The Chinese Wall Security Policy. In: IEEE Symposium on Security and Privacy, pp. 206–214 (1989)
9. Cuppens, F., Saurel, C.: Specifying a security policy: a case study. In: 9th IEEE Computer Security Foundations Workshop, pp. 123–134 (1996)
10. Deswarte, Y., Blain, L., Fabre, J.-C.: Intrusion tolerance in distributed computing systems. In: IEEE Computer Society Symposium on Research in Security and Privacy, pp. 110–121 (1991)
11. Fitting, M.: First-Order Logic and Automated Theorem Proving. Springer, Heidelberg (1990)
12. Freeman, J.W., Neely, R.B., Heckard, M.A.: A validated security policy modeling approach. In: 10th Annual Computer Security Applications Conference, pp. 189–200 (1994)
13. Gentzen, G.: Untersuchungen über das logische Schließen. Mathematische Zeitschrift 39, 176–210, 405–431 (1935)
14. Halpern, J., Weissman, V.: Using First-Order Logic to Reason about Policies. ACM Transactions on Information and System Security 11(4), 21–41 (2008)
15. Hartel, P., van Eck, P., Etalle, S., Wieringa, R.: Modelling mobility aspects of security policies. In: Barthe, G., Burdy, L., Huisman, M., Lanet, J.-L., Muntean, T. (eds.) CASSIS 2004. LNCS, vol. 3362, pp. 172–191. Springer, Heidelberg (2005)

16. Ko, C., Fink, G., Levitt, K.: Automated detection of vulnerabilities in privileged programs by execution monitoring. In: 10th Annual Computer Security Applications Conference, pp. 134–144 (1994)
17. Kotenko, I.: Active vulnerability assessment of computer networks by simulation of complex remote attacks. In: International Conference on Computer Networks and Mobile Computing, pp. 40–47 (2003)
18. Kreitz, C., Otten, J.: Connection-based Theorem Proving in Classical and Non-classical Logics. Journal of Universal Computer Science 5, 88–112 (1999)
19. Letz, R., Schumann, J., Bayerl, S., Bibel, W.: Setheo: A High-performance Theorem Prover. Journal of Automated Reasoning 8, 183–212 (1992)
20. Letz, R., Stenz, G.: Model Elimination and Connection Tableau Procedures. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, pp. 2015–2114. Elsevier, Amsterdam (2001)
21. Martelli, A., Montanari, U.: An Efficient Unification Algorithm. ACM Transactions on Programming Languages and Systems 4, 258–282 (1982)
22. Menezes, R., Ford, R., Ondi, A.: Swarming computer security: an experiment in policy distribution. In: Swarm Intelligence Symposium, pp. 436–439 (2005)
23. Müller, G., Accorsi, R., Höhn, S., Sackmann, S.: Sichere Nutzungskontrolle für mehr Transparenz in Finanzmärkten. Informatik Spektrum 33(1), 3–13 (2010)
24. Otten, J.: leanCoP 2.0 and ileanCoP 1.2: High Performance Lean Theorem Proving in Classical and Intuitionistic Logic. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS (LNAI), vol. 5195, pp. 283–291. Springer, Heidelberg (2008)
25. Otten, J.: Restricting Backtracking in Connection Calculi. AI Communications 23(2-3), 159–182 (2010)
26. Otten, J., Bibel, W.: leanCoP: Lean Connection-Based Theorem Proving. Journal of Symbolic Computation 36(1-2), 139–161 (2003)
27. Otten, J., Sutcliffe, G.: Using the TPTP Language for Representing Derivations in Tableau and Connection Calculi. In: Konev, B., Schmidt, R., Schulz, S. (eds.) Workshop on Practical Aspects of Automated Reasoning, Edinburgh, UK (2010)
28. Schumann, J.: Automated Theorem Proving in Software Engineering. Springer, Heidelberg (2002)
29. Stolzenburg, F., Thomas, B.: Analyzing Rule Sets for the Calculation of Banking Fees by a Theorem Prover with Constraints. In: Bibel, W., Schmitt, P.H. (eds.) Automated Deduction - A Basis for Applications, vol. III, pp. 243–264. Kluwer, Dordrecht (1998)
30. Sutcliffe, G.: The TPTP Problem Library and Associated Infrastructure. The FOF and CNF Parts, v3.5.0. Journal of Automated Reasoning 43(4), 337–362 (2009)
31. Sutcliffe, G.: The 5th IJCAR Automated Theorem Proving System Competition. AI Communications (to appear 2011)
32. Tatsubori, M., Imamura, T., Nakamura, Y.: Best-practice patterns and tool support for configuring secure web services messaging. In: IEEE International Conference on Web Services, pp. 244–251 (2004)
33. Trcek, D.: Security policy management for networked information systems. In: Network Operations and Management Symposium, pp. 817–830 (2000)
34. Waaler, A.: Connections in Nonclassical Logics. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, pp. 1487–1578. Elsevier, Amsterdam (2001)
35. Xie, W., Ma, H.: A policy-based security model for web system. In: IEEE International Conference on Communication Technology, vol. 1, pp. 187–191 (2003)
36. Yu, C.-F.: Access control and authorization plan for customer control of network services. In: IEEE Global Telecommunications Conference, vol. 2, pp. 862–869 (1989)