社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2007－DPS－130 (75)
2007－CSEC－ 36 (75)
2007／3／2

モデル検査ツールによるポリシー整合性検証

鴨田浩明[†,††] 河野和宏[††]
伊藤義道[††] 馬場口登[††]

近年，外部からの不正アクセスに加え組織内部者による情報の持ち出しや，操作ミスによる情報漏洩が問題となっている．これらの原因として組織内で利用されるシステムの分散化と設定の複雑化があげられる．個別のシステムの設定が例え正しく行われていても，他のシステムと適切に連携していないために，不正な運用を許可してしまったり，逆に適切な運用を妨げてしまうといった問題が発生する．そこで本稿では，ドキュメント管理システムを対象に各システムの設定が適切に行われ，異なるシステム間との整合性が確保されていることをモデル検査技術を用いて検証する方式について提案する．提案方式により，情報の機密性や可用性に問題がある場合には，具体的にどのようなケースで問題が発生するかを把握することが可能となる．

Access Control Policy Inconsistency Check Using Model Checker

Hiroaki Kamoda,[†,††] Kazuhiro Kono,[††] Yoshimichi Ito[††]
and Noboru Babaguchi[††]

In recent years, it becomes a serious problem that a user carries out confidential information to the outside of the office or that the security incident occurs by miss-operation in addition to unknown attacks from the outside. One of the causes of these problems could be decentralization of the system used in an organization, and complication of the configuration. Even if each system configuration is designed correctly, they may be not appropriately in cooperation with other systems. Therefore it could be a problem such that a user can perform operation beyond the user's authority or a user cannot perform operation which should be possible. In this paper, we propose a method using model checking that can verify the configurations which are defined in different document management systems. The advantage of the method is that it can detect undesirable situations about confidentiality or availability of the system.

1. はじめに

従来より情報漏洩等のセキュリティインシデントの発生が問題となっている．第三者による外部からの不正アクセスに加え，特に近年では内部犯行やシステムの設定ミス等による組織内情報の意図しない漏洩が大きな問題となっている．この原因の一つはシステム設定の複雑さにある．例えば政府や企業などの組織内の一般的なドキュメントライフサイクルを考えると，生成・保管・利用・廃棄といった具合に様々なステータスがある．しかしながら多くの組織では利便性やコストの問題からすべてのステータスを一括して管理可能なシステムを導入しているケースは少なく，ステータス毎に別々のシステムを用いて管理していたり，システムで制御できないことを運用で対処していたりする場合が多い．

例えば，LDAP (Lightweight Directory Access Protocol) サーバのようなユーザIDを管理するユーザ情報管理サーバ，共有ディレクトリへのアクセス権を制御するファイルサーバ，Microsoft Windows®Rights Management Services[3] や Adobe®LiveCycle™ Policy Server[7] の様にファイル単位で編集や印刷の可否を制御できる DRM (Digital Rights Management) 技術，印刷を制限するプリンタシステムなどが個別の技術として存在している．これらの技術を適切に組み合わせてドキュメントの一貫したライフサイクル管理システムを構築することにより，情報漏洩のリスクを低減することが可能である．しかしながら，実際にはすべてのシステムを適切に設定することは難しく，権限の無いユーザが権限外の操作を出来てしまうという機密性の問題，逆に権限のあるユーザが本来出来るべき操作を妨げられてしまうという可用性の問題が発生する可能性がある．特に，管理対象とするユーザの数

† 株式会社 NTT データ
†† 大阪大学

が多く，システムの分散化・複雑化が進むと，各システム間の整合性が保たれるように設定を実施することは困難である．また，熟練したシステム管理者が仮に正しく設定することが出来たとしても，いわゆる日本版 SOX 法と呼ばれる金融商品取引法 (証券取引法等の一部を改正する法律」およびその整備法) ではその設定が正しいことを何らかの形式で証明した評価報告書の提出が求められており，システム設定の整合性を形式的に検証する技術が今後必要不可欠となる．

複数の制御装置にまたがる設定間に不整合が無いことを検証する研究としては，例えば VLAN の設定検証技術[4] やホームネットワークの連携サービス検証技術[8] がある．4) では複数のスイッチに設定された内容が適切であり，届くべきではない相手にパケットが配信されてしまったり，逆に届くべき相手にパケットが配信されないなどの不整合が発生しないことをモデル検査ツールである SPIN[2] を用いて検証する方法について提案している．また，8) ではエアコンや温度計，煙探知機などの家庭で使用されるネットワーク機器を協調させた連携サービスの設定情報を同様に SPIN を用いて検証する手法に関して提案している．しかしながら情報のライフサイクルに着目し，システムの設定が機密性や可用性を確保していることを検証する技術に関してはこれまで提案されていない．

本稿では，ドキュメント管理のために組織内で利用される一般的なシステムの制御情報をモデル化し，モデル検査技術[1] を用いて検証することにより，制御システム間の機密性と可用性に関する不整合を検出する方式を提案する．

本論文の構成は以下の通りである．2 章では，本稿で対象とするドキュメント管理システムのモデルについて説明する．3 章では，設定したドキュメント管理システムにおいて発生する不整合について説明する．4 章では，本稿で整合性を検証するために用いるモデル検査技術の概要について説明する．5 章では，対象とするドキュメント管理システムをどのようにモデル検査ツールで検証するかについて説明する．6 章では，検証結果について説明する．最後に 7 章でまとめを述べる．

2. ドキュメント管理システム

本章では，管理対象ドキュメント及びドキュメント管理システムの概要について説明する．本稿ではモデル検査技術を用いた汎用的な不整合検出方式について提案するため，具体的な製品に特化したシステムではなく，一般的な特徴を兼ね備えたいくつかのシステムから構成される仮想ドキュメント管理システムを扱うこととする．

2.1 管理対象ドキュメント

管理対象となるドキュメントは一般的な電子文書であり，読み込み・書き込み・印刷等の文書に関する操作が可能であるものとする．ただし，ドキュメントに対する各操作は以下で説明する種々のシステムによりユーザ単位またはグループ単位で制限されるものとする．例えば，同一のドキュメントであってもユーザによっては読み込みは出来るが書き込みは出来ないという場合が考えられる．

2.2 ドキュメント管理システム

本稿で対象とするドキュメント管理システムは，ユーザ情報管理システム，共有ディレクトリシステム，ファイル保護システムの 3 つのシステムから構成される．各システムの概要を以下に説明する．

2.2.1 ユーザ情報管理システム

ユーザ情報管理システムは，ユーザ ID 及びユーザが所属するグループを管理する機能を提供する．具体的な製品の例としては LDAP サーバや Microsoft 社の提供する Active Directory 等が該当する．

ユーザ情報管理システム上にはドキュメント管理システムを利用する組織内のすべてのユーザに 1 対 1 に紐尽くユーザ ID が登録されている．簡単のため全ユーザの集合を U，個別のユーザ ID を $\{u_1, u_2, \cdots, u_n\}$ と表現することとする．さらに，ユーザ情報管理システムは複数の任意のユーザ ID をまとめたグループを複数定義することが出来るものとする．すべてのユーザ ID は複数のグループに所属することが可能である．簡単のためにグループを記号 G を用いて $G_1, G_2, \cdots$ 等と表現することとする．

2.2.2 共有ディレクトリシステム

共有ディレクトリシステムは，管理対象ドキュメントを保存し，組織内ユーザがネットワーク経由で共有することの出来る機能を提供する．ユーザ情報管理システムが管理するユーザ ID 及びグループ単位でディレクトリ内のドキュメントへの操作を制御することが可能である．

共有ディレクトリシステムはディレクトリ毎に共有ディレクトリポリシーを定義することが可能である．共有ディレクトリポリシーはディレクトリ名，ユーザ情報及び許可する操作の組から構成され (D, U, A) と表現される．ここで，D はディレクトリ名，U はユーザ ID またはグループ名，A は許可する操作名をそれぞれ表現するものとする．共有ディレクトリポリシーで定義される操作の種類はディレクトリ内に保存され

たファイルに対して閲覧を許可するreadと，書き込みを許可するwriteの2種類が定義可能であるとする．

例えば，$(pass1, u_1, read)$は，pass1と表現されるディレクトリに対して，ユーザu_1はreadが許可されるというポリシーになる．また，$(pass2, G_1, write)$は，pass2と表現されるディレクトリに対して，ユーザグループG_1に所属するユーザはwriteが許可されるというポリシーになる．

2.2.3 ファイル保護システム

ファイル保護システムは，Windows Rights Management ServicesやAdobe LiveCycle Policy Serverのようなファイル保護技術を提供するシステムであり，ドキュメント毎に読み込み・書き込み・印刷等の可否を制御することが出来る機能を提供する．実現方式にはいくつかの種類があるが，本稿ではファイル保護情報をドキュメントに埋め込むことによりファイル保護を実現する方式を想定する．具体的には，ドキュメントを作成した時点でドキュメント制作者が，そのドキュメントに付与するファイル保護ポリシーを決定しドキュメントにポリシーを埋め込んで保存する．ファイル保護ポリシーはユーザ情報と操作の組から構成され(U, A)と表現される．ここで，UはユーザIDまたはグループ名，Aは許可する操作名をそれぞれ表す．ファイル保護ポリシーで定義される操作の種類は当該ファイルの閲覧を許可するread，書き込みを許可するwrite，ファイルの内容を他のドキュメントへコピー&ペーストすることを許可するcopy，印刷を許可するprintの4種類が定義可能であるとする．

3. ポリシー不整合

前章で定義したドキュメント管理システムにおいて，ユーザ情報の登録やポリシーの設定を不適切に行うと，意図しない情報漏洩が発生したり，本来出来るべき操作が出来なくなると言った問題が発生する可能性がある．これらの問題をポリシー不整合と呼ぶこととする．前章で定義したドキュメント管理システムにおいて発生する可能性のある2種類の不整合について以下に説明する．

3.1 機密性に関する不整合

ユーザがドキュメント作成時に付与したファイル保護ポリシー及び最初に保存したディレクトリに設定されている共有ディレクトリポリシーにより許可されてない事象が発生してしまうことを，機密性に関する不整合と呼ぶ．

機密性に関する不整合は，厳密には次のように定義される．ドキュメントが最初に保存されたディレクトリのポリシーが(D, U_1, A_1)，ドキュメントに付与されたファイル保護ポリシーが(U_2, A_2)であるとする．この時，何らかの過程を経てあるユーザ$u \notin U_1 \cup U_2$がある操作$a \in A_1 \cup A_2$を実行出来てしまう場合に，機密性に関する不整合が発生すると定義する．

3.2 可用性に関する不整合

ユーザがドキュメント作成時に付与したファイル保護ポリシー及び最初に保存したディレクトリに設定されている共有ディレクトリポリシーにより許可されている操作が出来ない状態になってしまうことを，可用性に関する不整合と呼ぶ．

可用性に関する不整合は，厳密には次のように定義される．ドキュメントが最初に保存されたディレクトリのポリシーが(D, U_1, A_1)，ドキュメントに付与されたファイル保護ポリシーが(U_2, A_2)であるとする．この時，何らかの過程を経てあるユーザ$u_1 \in U_1, u_2 \in U_2$がある操作$a_1 \in A_1, a_2 \in A_2$をそれぞれ実行することが不可能な状態になる場合に，可用性に関する不整合が発生すると定義する．

4. モデル検査技術

ドキュメント管理システムに設定されたポリシーにおいて，3章で定義した機密性及び可用性に関する不整合が発生するか否かをモデル検査技術を用いて検証する方式について5章で説明する．本章では，具体的な検証方式を述べる前にモデル検査の仕組みと，モデル検査の実装であるモデル検査ツールについて説明する．

4.1 モデル検査の仕組み

モデル検査技術とは，形式的手法の一つであり，有限状態遷移系でモデル化されたシステムの状態空間を網羅的に探索することにより，デッドロックや不活性などの望まない状態に到達する可能性があるか無いかを検証する技術である．具体的には，システムの有限状態遷移系を記述したものと，システムに要求される性質を時相論理式で記述したものを入力として与え，状態遷移系が時相論理式で与えられた性質を満足するかどうかを網羅的に探索する技術である．モデル検査の詳細については文献1)に詳しく解説されている．

4.2 モデル検査ツール

モデル検査ツールとは，モデル検査技術に基づいたアルゴリズムが実装されたツールであり，SPINやLTSA[5]，SMV[6]等いくつかのツールが存在している．本稿では状態遷移系の記述の簡便性，及び記述可能な時相論理式の自由度が高いという観点でSPINを用いることとする．

5. ドキュメント管理システムのモデル化

2章及び3章で定義したドキュメント管理システムのポリシーとその不整合をSPINを用いて検証するためには，各システムのポリシーをSPINの仕様記述言語であるPromela[2]に変換する必要がある．本章では各ポリシーをどのようにPromelaに変換するかについて説明する．なお，本章で説明するPromelaの完全なソースコードを付録A.1に掲載する．

5.1 ポリシー

本節では，SPINによる検証の対象とする具体的なポリシーについて説明する．

最初に，ユーザ情報管理システムに登録されるユーザは$\{u_1, u_2, u_3, u_4\}$の4ユーザであるとする．そして，2つのグループG_1, G_2があり，$G_1 = \{u_1, u_2\}$，$G_2 = \{u_2, u_3, u_4\}$という所属関係にあるとする．

次に，共有ディレクトリシステムには2つのディレクトリD_A, D_Bがあり，以下のようなポリシーが定義されているものとする．

policy 1 ： $(D_A, G_1, \{read, write\})$

policy 2 ： $(D_B, G_2, \{read, write\})$

即ち，ディレクトリD_Aには，G_1に所属するユーザのみがreadまたはwrite可能であり，ディレクトリD_Bには，G_2に所属するユーザのみがreadまたはwrite可能であるというポリシーが設定されているものとする．

また，今回はユーザu_1がドキュメントの作成者であり，作成したドキュメントをディレクトリD_Aに保存する際に，下記のファイル保護ポリシーを当該ドキュメントに設定したと仮定する．

policy 3 ： $(G_1, print)$

即ち，u_1が作成したドキュメントはG_1に所属するユーザのみが印刷可能であるとする．

これらのユーザ情報及びポリシーをどのようにPromelaで記述するかについて次節で説明する．

5.2 Promelaによる記述

ユーザ情報管理システムで管理されるユーザIDはmtype型のデータとして下記のように最初に宣言する．

```
mtype={u1, u2, u3, u4}
```

そして各ユーザがどのグループに属しているかは，Promelaのマクロ記述であるinline文を用いて必要な場合に判定するようにする．具体的には下記に示すように，ユーザがu_1またはu_2である場合には，グループG_1に所属していることを示す変数u_in_g1をtrueにすることにより，ユーザの所属を判定する．

```
inline g1(u){
  if
    ::(u==u1||u==u2)->u_in_g1 = true
    ::else->skip
  fi}
```

次に共有ディレクトリポリシーとファイル保護ポリシーについて説明する．共有ディレクトリシステムでは，各ディレクトリ毎にポリシーが定義されており，ユーザ情報に応じて実行可能な操作が異なる．そこで2重のif文を用いて下記のような記述でポリシーを表現することとする．

```
if
 :: (file==directory_a) ->
 if
  ::u_in_g1->event=read; file=directory_a
  ::u_in_g1->event=write;file=directory_a
  ::u_in_g1->event=print;file=directory_a
  ::(u_in_g1 && u_in_g2)
          ->event=write;file=directory_b
  :: else -> skip
 fi;
 ::else -> skip
fi
```

最初のif文では現在の操作対象のドキュメントがどのディレクトリに保存されているかの判定を行う．上記の記述ではディレクトリD_Aに関する処理について記述している．二つめのif文ではディレクトリポリシーとファイル保護ポリシーで設定されている制約を記述する．即ち，$u \in G_1$である場合には，read, write, printのいずれの操作も可能であり，操作後も該当ドキュメントはディレクトリD_Aに保存され続ける．しかし，$u \in G_1$かつ$u \in G_2$に所属するユーザが存在する場合には，当該ファイルをディレクトリD_Bに保存する可能性があることを示している．ディレクトリD_Bに関しても同様の記述となる．

最後に，ドキュメントを操作するユーザが非決定的に選択するようにPromelaで記述することにより，モデル化が完了する．実際のシステムにおいてはどのユーザがどのような操作を行うかを事前に把握することは不可能である．そこで下記のように，提案方式ではユーザが非決定的に選ばれるようにする．

```
if
 :: true -> u=u1 ; g1(u1) ; g2(u1)
 :: true -> u=u2 ; g1(u2) ; g2(u2)
 :: true -> u=u3 ; g1(u3) ; g2(u3)
 :: true -> u=u4 ; g1(u4) ; g2(u4)
fi;
```

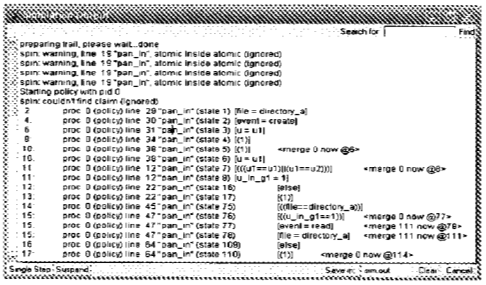

図 1 SPIN の反例トレース
Fig. 1 Counter Example Trace of SPIN

6. 検　　証

本章では，5 章で説明した Promela 記述を用いて，機密性と可用性の不整合を SPIN により検証できることを示す．

6.1 機密性の検証

機密性の不整合の例として次のような事例を考える．

- u_1 が G_1 に所属するユーザのみに閲覧・印刷させる目的で作成したドキュメントを G_2 に所属する u_1 以外のユーザが閲覧できてしまう．

このことを検証するためには，以下の式を検証式として SPIN に入力し反例の有無を検証すればよい．

$\Box\neg((u \notin G_1) \wedge (event = read) \wedge (u \in G_2))$

即ち，G_1 に所属せずグループ G_2 にのみ所属しているユーザが当該ドキュメントを read 出来るという状態が決して発生しないということを検証する．

前述のモデルに対してこの検証式を検証すると反例が出力される．従って現状のポリシーでは機密性が保証されない可能性があることがわかる．出力の一部を**図 1** に示す．SPIN の出力する反例トレースを詳細に検証すると，次の場合に機密性の不整合が発生することがわかる．つまり u_2 がグループ G_1, G_2 の両方に所属しているため，u_2 がディレクトリ D_A に保存された当該ドキュメントをディレクトリ B_2 に移動してしまった場合に，G_2 に属する u_2 以外のユーザが閲覧できてしまうということになる．

6.2 可用性の検証

可用性の不整合の例として次のような事例を考える．

- u_1 が G_1 のみに閲覧・印刷させる目的で作成したドキュメントが G_1 に所属するユーザであっても印刷できない状態が発生してしまう．

このことを検証するためには，以下の式を検証式として SPIN に入力し，反例の有無を検証すればよい．

$\Diamond((u \in G_1) \wedge (event = print))$

即ち，G_1 に所属するユーザであれば，print という操作をいつか実行することが出来るということを検証する．前述のモデルに対してこの検証式を検証すると反例が出力される．従って現状のポリシーでは可用性が保証されない可能性があることがわかる．SPIN の出力する反例トレースを分析すると，u_2 がグループ G_1, G_2 の両方に所属しているため，u_2 がディレクトリ D_A に保存された当該ドキュメントをディレクトリ D_B に移動してしまった場合に，G_1 に属するユーザが印刷することが出来なくなってしまう可能性があることがわかる．

以上のように，提案方式により機密性と可用性の不整合を検出可能であることを確認することが出来た．

7. おわりに

本稿では，ドキュメント管理システムのユーザ管理情報やポリシーを仕様記述言語の Promela で記述し，モデル検査ツールの SPIN を用いて解析することにより，機密性と可用性の不整合を検証する方式について提案した．また簡単な事例を用いて，実際に検証が可能であり，不整合が発生する場合には，どのよう手順で発生するかを確認することが出来ることを示した．

今後は，対象とするシステムを拡大し，例えばルーターやメールサーバなどのネットワーク機器，プリンタなどの出力機器を含めた整合性検証技術に関して検討を行う予定である．また，各システムの設定情報から自動的に Promela 記述への変換を行う方式についても検討を行う予定である．

謝辞　モデル検査技術に関する有益な情報を数多く提供して下さった国立情報学研究所の田原 康之氏，長野 伸一氏に，謹んで感謝の意を表する．

参 考 文 献

1) E. M. Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. The Mit Press, 2000.
2) Gerard J. Holzmann. *The SPIN MODEL CHECKER, Primer and reference manual*. Addison Wesley, 2004.
3) Microsoft. Windows rights management services, http://www.microsoft.com/japan/windowsserver2003/technologies/rightsmgmt/default.mspx.
4) Hideki Sakurada. Model checking configurations of local area networks. 第 2 回ディペンダブルソフトウェアワークショップ論文集 (2005). 日本ソフトウェア科学会, 2005.
5) Labelled Transition System. http://www.doc.ic.ac.uk/ltsa/.
6) TheSMV System. http://www.cs.cmu.edu/mod-

elcheck/smv.html.
7) Adobe Systems. Adobe livecycle policy server, http://www.adobe.com/jp/products/server/policy/.
8) 松尾尚文, パッタラリーラープルット, 土屋達弘, 菊野亨. ホームネットワークシステムにおける連携サービスのモデル検査による検証. 電子情報通信学会技術研究報告 (SS2005-84). 電子情報通信学会, 2006.

付　　録

A.1 Promela による記述

```
01: mtype = {u1, u2, u3, u4};
02: bool u_in_g1;
03: bool u_in_g2;
04: mtype = {directory_a, directory_b};
05: mtype = {create, read, write, print};
06: mtype file;
07: mtype event;
08: mtype u;

10: inline g1(u){
11:  if
12:     ::(u==u1 || u==u2) ->
13:        u_in_g1 = true
14:     ::else -> skip
15:   fi
16: }

17: inline g2(u)
18: {
19:   atomic{
20:   if
21:     ::(u==u2 || u==u3 || u==u4) ->
22:        u_in_g2 = true
23:     ::else -> skip
24:   fi
25:   }
26: }

28: active proctype policy(){
29: file = directory_a;
30: event = create;
31: u=u1;
32: do
33:  ::true ->
34:  atomic{
35:  if
36:   :: true -> u=u1 ; g1(u1) ; g2(u1)
37:   :: true -> u=u2 ; g1(u2) ; g2(u2)
38:   :: true -> u=u3 ; g1(u3) ; g2(u3)
39:   :: true -> u=u4 ; g1(u4) ; g2(u4)
40:  fi;
41:  if
42:   :: (file==directory_a) ->
43:   if
44:    :: (u_in_g1==true) -> event=read;
45:                    file=directory_a
46:    :: (u_in_g1==true) -> event=write;
47:                    file=directory_a
48:    :: (u_in_g1==true) -> event=print;
49:                    file=directory_a
50:    :: (u_in_g1==true && u_in_g2==true) ->
51:                    event=write;
52:                    file=directory_b
53:    :: else -> skip
54:   fi;
55:   ::else -> skip
56:  fi;
57:  if
58:   :: (file==directory_b) ->
59:   if
60:    :: (u_in_g2==true) -> event=read ;
61:                    file=directory_b
62:    :: (u_in_g2==true) -> event=write;
63:                    file=directory_b
64:    :: (u_in_g1==true && u_in_g2==true) ->
65:                    event=write;
66:                    file=directory_a
67     :: else -> skip
68    fi;
69    ::else -> skip
70   fi;
71  }
72  od
73: }
```