脅威トレースのためのシステムモデル記述言語の実装

山口 凌[1]　村上 隆俊[2]　熊野 修平[3]　小出 洋[4]

概要：本研究では，脅威トレースへ対象となる情報システムのモデル記述を容易にするシステムモデル記述言語を実装した．脅威トレースは，国家や企業を対象とした高度標的型攻撃の対策として，情報システムを設計する作業を支援することを目的としている．抽象度の高い情報システムのモデルと攻撃シナリオを用いて，脅威の振る舞いをシミュレートし，その結果を情報システムの設計に役立てる．現在の脅威トレースには，情報システムのモデルを容易に記述し扱うための仕組みが用意されていない．そこで，本研究で実装したシステムモデル記述言語では，脅威トレースで扱うモデル間の関係性を容易に把握でき，省略や補完によって記述者の負担を軽減することを目標とした．本論文では，このシステムモデル記述言語の機能と実装について紹介する．

キーワード：高度標的型攻撃，脅威トレース，ドメイン固有言語，ネットワーク記述言語

An Implementation of System Model Description Language for Attacks Tracer

Ryo Yamaguchi[1]　Takatoshi Murakami[2]　Shuhei Kumano[3]　Hiroshi Koide[4]

Abstract: In this study, we implement a system model description language for attacks tracer in order to describe model of information systems easily. The attacks tracer is intended to support work to design information systems as a measures for advanced persistent threats. The results of simulation of behavior of threats by using the model of information system in higher level of abstraction is used to help design the information system. In our current attacks tracer, there is no mechanisms for describe and handle easily the model of information system. Therefore, a system model description language that implement in this study aims to reduce complexities of descriptions by abbreviations and autocompletions. In this paper, we introduce functions and an implementation of this system model description language.

Keywords: Advanced persistent threats, Attacks tracer, Domain specific language, Network description language

1. はじめに

国家や企業を対象として，機密情報の窃取などを目的とした高度標的型攻撃による被害が後を絶たない．高度標的型攻撃では，対象組織の情報システム内部へ侵入するための攻撃手段が巧妙化しており，完全に侵入を防ぐことは困難である．そこで，攻撃者が情報システム内部へ侵入することを前提として，被害を出さないための情報システム設計を行うことが必要とされている [1]．現在，その情報システム設計を支援するために，脅威トレースが提案されている．脅威トレースは，情報システムの抽象度の高いモデルと攻撃シナリオを用いて，コンピュータ上で脅威の振る舞いをシミュレートし，高度標的型攻撃の対策に役立てるシステムである．

脅威トレースを使用するには，トレース対象となる情報システムをモデル化し記述したものを入力する必要がある．また，モデルを調整しながら繰り返しシミュレーションを行うことが想定されるため，記述したモデルは容易に把握・変更が行えることが望ましい．しかし，現在の脅威トレースには，情報システムのモデルを容易に記述し扱うことができる仕組みは用意されていない．そこで本研究では，そのような仕組みを脅威トレースに実現することを目的として，システムモデル記述言語の提案，実装を行った．

本稿の構成は以下の通りである．2 章では，高度標的型攻撃について具体的にどのような攻撃であるか，またその対策はどのように行うべきかについて説明する．3 章では，脅威トレースについて説明し，問題点やシステムモデル記述言語で扱う必要のあるモデルについて明らかにする．4 章では，モデルの容易な記述を実現するためには，どのような構文や機能が必要であるかの検討を行う．5 章では，システムモデル記述言語の構文や機能について，実装を紹介する．6 章では，既存のネットワーク記述言語との比較を行う．最後に 7 章では，まとめと今後の課題を述べる．

[1] 九州工業大学情報工学部知能情報工学科
Department of Artificial Intelligence Kyushu Institute of Technology
[2] 九州工業大学大学院情報工学府情報科学専攻
Department of Artificial Intelligence Graduate School of Computer Science and Systems Engineering Kyushu Institute of Technology
[3] 九州工業大学大学院情報工学府情報創成工学専攻
Department of Creative Informatics Graduate School of Computer Science and Systems Engineering Kyushu Institute of Technology
[4] 九州工業大学大学院情報工学研究院情報創成工学研究系
Department of Creative Informatics Faculty of Computer Science and Systems Engineering Kyushu Institute of Technology

2. 高度標的型攻撃

高度標的型攻撃 [2] は，国家や企業などを対象として機密情報の窃取や情報システムの破壊を目的とするサイバー攻撃の一種である．攻撃者は対象となる組織の情報を念入りに収集し，目的を遂行するために巧みに手法を変えながら攻撃を行う．

攻撃の初期段階では，対象組織の情報システム内部へ侵入するために組織の端末をマルウェアに感染させる．その際用いられる攻撃手法は，メールを使用したやり取り型攻撃 [3] やウェブサイトの改ざんによる水飲み場型攻撃 [4] など多岐にわたり，感染させるマルウェアも組織に合わせて細部がカスタマイズされたものとなる．そのため，ウイルス対策ソフトウェアや IDS(侵入検知システム) などを用いたこれまでの「入口対策」では十分に攻撃者の侵入を検知・防御することができない．また，高度標的型攻撃では継続して長期に渡る機密情報の窃取を目的としている場合も多く，情報システム内部へ侵入された後も攻撃の痕跡を検知することは困難である．

このような特徴を持つ高度標的型攻撃への対策として，情報処理推進機構 (IPA) は「『高度標的型攻撃』対策に向けたシステム設計ガイド」を公開し [1]，「内部対策」の重要性について訴えている．内部対策とは，攻撃者の情報システム内部での活動を困難にし，侵入を早期に発見できるようなシステムを設計することである．これは，組織の情報システムの設計手法を，攻撃者が入口対策を突破し情報システム内部へ侵入することを前提とした，侵入拡大の防止および監視の強化を目的とする設計手法へ変更することで実現する．

3. 脅威トレース

3.1 模擬攻撃トレースの自動化

IPA は，システム設計策を検討する際に，攻撃手法を分析することで得られた攻撃シナリオと情報システムのモデルを用いて，机上で模擬攻撃トレースを行い，攻撃の成否を判定し，その結果をシステム設計策の検討に役立てることを提案している [5]．しかし，様々な攻撃シナリオについて正確かつ網羅的にトレースを行うことは，トレース対象となる情報システムの規模が大きくなるほど人手では困難となる．そこで現在は，コンピュータ上でこの模擬攻撃トレースを自動化するシミュレータである，脅威トレース [6] の提案，開発が進められている．コンピュータ上で実行できるため，ネットワーク構成の変更といったモデルの調整をしながら繰り返しシミュレーションを行うことが容易になる．これにより，規模が大きく複雑な情報システムであっても様々な攻撃シナリオについて正確にトレースを行うことができ，情報システムの設計策の検討とその効果の確認に役立てることができる．

3.2 現在のモデル記述と問題点

脅威トレースを使用するためには，トレース対象となる情報システムをモデル化し記述したものを脅威トレースに読み込ませる必要がある．しかし，現在の脅威トレースには，情報システムのモデルを記述するための特別な方法は用意されていない．したがって，トレース対象となる情報システムのモデルはソースコード上で手続き的に定義を行い，それぞれのモデルが持つメソッドを呼び出すことでモデル間の関係性やサーバ接続先などのパラメータの指定を行っている．図 1 に現在の脅威トレースにおいてモデルを記述するコードの例を示す．

この図 1 の例では，ネットワーク”private”にノード”PC”が接続されていて，ノード”PC”はプロキシを介したウェブブラウザアプリケーションを持つことを記述している．この現在の脅威トレースにおけるモデル記述の方法の問題点については以下の通りである．

```
val private = Network("private",
  InetAddress("192.168.0.0", 24))
val pc = Node("PC",
  InetAddress("192.168.0.1", 24))
val browser = new WebBrowserProxyApp()
  .setProxy(InetAddress("192.168.0.2", 24))
pc.setApplication(browser)
private.addNode(pc)
pc.addRoute(InetAddress.Default,
    InetAddress("192.168.0.3", 24), private
    )
```

図 1 現在の脅威トレースにおけるモデル記述例．
Fig.1: An example of a model description in current attacks tracer.

(1) 脅威トレース内部の実装に依存した記述
モデル間の関係性の指定 (図 1 の例では `addNode` メソッドなど) やパラメータの設定 (図 1 の例では `setProxy` メソッド) において各モデルが持つメソッドを直接モデルの記述に使用している．脅威トレースはまだ開発途上であるため，内部のモデルの実装が変更される可能性は少なくない．そのため，内部のモデルの実装が変更された際は，情報システムのモデルの定義そのものには変更がなかったとしても記述を変更する必要が生じてしまう．

(2) 各モデルの関係性が不明確
図 1 の例程度では問題となりにくいが，より規模が大きく複雑な情報システムのモデルを扱う場合に，どのネットワークにどのノードが接続されているのか，などの各モデル間の関係性を正確に把握することは困難である．そのため，シミュレーション実行時にモデルの調整やパラメータの変更を行う場合に，どこを変更すればよいのかを容易に見つけることができない可能性がある．記述者が関係のある定義を近い場所にまとめて記述する，といった工夫を行うことで関係性を表現することは可能である．しかし，逆に言えば，記述の際に工夫をしなければならないため，モデルを追加・変更する際に記述者の負担となり，保守性が低い．

(3) 冗長な記述
前述したように，モデルが持つメソッドを直接使用するため，記述が冗長である．また，規模が大きな情報システムでは，ユーザ操作端末に相当する”PC”というノードが複数存在し，様々なネットワークに接続されていることが考えられる．モデルのインスタンスを生成後にそれらを変数に格納する場合は，変数の名前で区別をする必要が有るため，やはり冗長となる．さらに，ノードにおけるIPアドレスのサブネットマスクやルーティング情報などについて省略ができないため，同じ内容であってもノードごとに記述を行う必要がある．モデルを記述する際には，統合開発環境(IDE)などによる補完が働くため，冗長であっても入力の手間はそこまで増加しない．しかし，冗長な記述の中に本質的なモデルの定義が埋もれてしまい可読性が低いため，モデルの読解や変更は容易ではない．

3.3 脅威トレースが扱うモデル

次章では，システムモデル記述言語に必要な構文や機能などについて具体的な検討を行っていくが，その前に脅威トレースが扱うモデルについての説明をここで行う．

脅威トレースは，ネットワークシステムを正確にシミュレートするのではなく，より抽象化されたモデルによるデータグラムのやり取りをシミュレートする．ここで，データグラムは通信内容をモデル化したものであり，各アプリケーションが利用するプロトコルに応じたデータと送信先アドレス・ポート番号，送信元アドレス・ポート番号により構成される．現在の脅威トレースが扱う抽象化されたモデルは以下の通りである．

(1) Network
情報システム内のネットワークや攻撃者のネットワーク，インターネットをモデル化したものである．ネットワークには`Node`や`Firewall`が接続される．また，ネットワーク同士の接続は`Node`や`Firewall`を介して表現する．

(2) Node, Firewall
ユーザの操作端末であるPCや各種サーバなどのノードとファイアウォールをモデル化したものである．ノードは，ノード上で動作する`Application`を持つことができる．ファイアウォールは，ファイアウォールルールを持ち，データグラムをルールに応じて転送したり破棄したりする．どちらのモデルも`File`とルーティング情報を持つことができる．ルーティング情報はデータグラムのやり取りに使用される．

(3) Application
ノード上で動作するアプリケーションをモデル化したものである．クライアントアプリケーションとしては，ウェブブラウザ(`WebBrowserApp`)やメーラ(`MailerApp`)などがある．サーバアプリケーションとしては，ウェブサーバ(`WebServerApp`)や攻撃者がマルウェアのリモート操作で利用するCommand and Control(C&C)サーバ(`CCServerApp`)などがある．

(4) Malware
高度標的型攻撃でよく用いられるバックドア通信を行うマルウェアをモデル化したものである．マルウェアはノードに感染することができる．感染後は，攻撃者がもつC&Cサーバと通信を行い，自己の振る舞いを変更しながら情報システム内部を探索し，機密情報などのファイルを送信する．

(5) File
ファイルをモデル化したものである．ファイル名と内容であるデータのみを持つシンプルなモデルである．

これらのモデル間の関係性を図2に示す．脅威トレースでは，これらのモデルを利用して，トレース対象となる情報システムのモデルを構成する．

また，脅威トレースはアクターモデルを用いており，上述したモデルのうち`File`以外はすべてアクターである．アクター同士がデータグラムをやり取りする中で，マルウェアなどの脅威の振る舞いを確認し，情報システムの設計に役立てる．

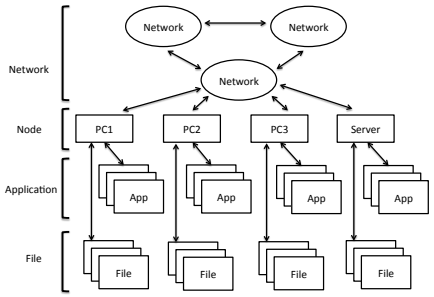

図 2 脅威トレースにおけるモデル間の関係性.
Fig.2: Relationships between models in attacks tracer.

4. システムモデル記述言語の検討

脅威トレースにおける情報システムのモデルを容易に記述するためのシステムモデル記述言語をどのように実現すべきかについて検討を行う.

4.1 ドメイン固有言語

まず，システムモデル記述言語はドメイン固有言語 (DSL) として実装される．ドメイン固有言語とは，特定の問題領域を表現することに特化した言語である．ドメイン固有言語は大別して外部 DSL と内部 DSL に分けられる．外部 DSL では，構文解析器などを独自に用意する必要がある代わりに，自由に文法を定義することができる．内部 DSL では，既存の汎用プログラミング言語 (ホスト言語) 上に，特定の問題領域を表現するための見かけ上新しい文法を実現する．システムモデル記述言語では，脅威トレースの開発に利用しているプログラミング言語 Scala にあわせて，Scala をホスト言語とする内部 DSL として実装することを選択した．この選択による利点を以下に示す.

(1) 構文解析器などの実装が不要
前述したように，内部 DSL では既存の汎用プログラミング言語であるホスト言語上に実装されるため，構文解析などはホスト言語のコンパイラに任せることができ，独自に実装を行う必要がない．そのため，外部 DSL と比較すると実装コストが低い.

(2) 簡潔な記述と将来における拡張性
例えば，開始タグと終了タグが必要な XML と比較すると，より簡潔な記述を実現することが可能である．また，内部 DSL では，将来において拡張が必要になった際に，XML や JSON などと比較して柔軟に文法を追加することが可能である.

(3) Scala が持つ言語機能の活用
Scala のプログラムとして有効であるため，変数や条件式，反復などの制御構文を自由に埋め込み利用することが可能である．また，変数の型によるメソッドの多重定義などを利用した構文を用意することも可能である．図 3 に for 式を用いて 10 個のノードを生成するコード例を示す.
第 5 章で詳細については説明するが，モデルの宣言はオブジェクトのインスタンス化の構文を利用するため，モデル宣言時にトレイトのミックスインが可能である．これにより，マルウェアが持つ機能をトレイトとして用意し，モデルを記述する際に，必要な機能をミックスインしたマルウェアのモデルを用意することが可能になる．図 4 にミックスインを用いて任意の機能を持たせたマルウェアを定義するコード例を示す．この例では，`FuncA` と `FuncB` という機能を持ったマルウェアのモデルを定義している.

(4) IDE やエディタのサポートを利用可能
Scala をサポートする統合開発環境 (IDE) やエディタでは，シンタックスハイライトや構文チェック，入力時の自動補完，ドキュメント表示などの機能を追加の実装なしで利用することが可能である.

4.2 パラメータや各モデル間の関係性の表現

続いて，パラメータや各モデル間の関係性を容易に記述・把握できるようにするために必要な構文について検討を行う．現在の記述方法では，記述の自由度が高く，それらの関係性を表現するには記述者の工夫を必要としている．そこで，ブロックの入れ子を利用して図 2 に示したモデル間の関

```
new Network("private") {
  for (i <- 1 to 10) {
    new Node("PC" + i) {
      new WebBrowserApp
    }
  }
}
```

図 3 for 式を用いて 10 個のノードを生成するコード例.
Fig.3: An code example that generates 10 nodes by using for-expression.

```
new Malware("malware") with FuncA with
    FuncB
```

図 4 ミックスインを用いてマルウェアのモデルを定義するコード例.
Fig.4: An code example that defines a malware model by using mix-in.

係性を表現することを考える．図 5 にこの記述によるコード例を示す．

この図 5 のコード例は，図 1 での例と同じモデルを表現している．この例では，モデルのブロック内に関係のあるパラメータや子要素となるモデルが記述されることにより，それらの関係性を視覚的に把握することが可能になっている．具体的には，Network ブロック内には，ネットワークアドレスを表現する Address やノードモデルである Node が記述され，それらがネットワークに関連のあるパラメータや子要素となるモデルであることが分かる．同じように，入れ子となる Node ブロック内には，ノードアドレスを表現する Address やノード上で動作するアプリケーションのモデルである WebBrowserProxyApp が記述され，それらがノードに関連があるものであることが分かる．また，この構文ではパラメータやモデルの記述が宣言的であり，脅威トレースにおける内部のモデルの実装には依存していない．そのため，内部のモデルが変更された場合でも，システムモデル記述言語を評価するプログラムの変更のみで対応が可能であり，情報システムのモデルそのものを変更する必要はない．

```
new Network("private") {
  Address := "192.168.0.0/24"
  new Node("PC") {
    Address := "192.168.0.1/24"
    new WebBrowserProxyApp {
      ProxyServer := "192.168.0.2/24"
    }
  }
}
```

図 5 ブロックによりモデル間の関係性を表現したコード例.
Fig.5: An code example that expresses relationships between models by using block.

4.3 記述の把握しやすさと誤りの軽減

次に，記述者が記述内容を把握することに対する負担や記述時の誤りを軽減することを考える．脅威トレースは各モデルのアクター同士がデータグラムをやり取りすることで通信のシミュレーションを行う．その際のデータグラムの転送先の指定には IP アドレスをモデル化したものを用いている．そのため，情報システムのモデルを記述する際にも，ネットワークアドレスやノードアドレス，サーバ接続先の指定など様々な場所で IP アドレスを記述する必要がある．しかし，この中でサーバ接続先の指定においては，”どこに接続するのか”を表現できればよいため，必ずしも IP アドレスを用いる必要はないと考えられる．そこで，ネットワークやノードに一意な名前をつけることとし，それを用いた接続先の表現を可能にする．図 6 に IP アドレスを用いた接続先指定のコード例を，図 7 に名前を用いた接続先指定のコード例を示す．

図 6 と図 7 の例において，ネットワーク”dmz”にメールサーバとなるノード”MailServer”が存在し，その IP アドレスが”192.168.1.6”の場合，この 2 つのメーラのメールサーバ接続先は同じである．

```
new MailerApp {
  MailServer := "192.168.1.6"
}
```

図 6 IP アドレスを用いた接続先の指定例.
Fig.6: An example that specifies the connection destination by the IP address.

```
new MailerApp {
  MailServer @= "dmz/MailServer"
}
```

図 7 名前を用いた接続先の指定例.
Fig.7: An example that specifies the connection destination by the name.

IP アドレスを用いた指定よりも名前を用いた指定のほうが，どのネットワークのどのノードに接続しているかを十分に説明できているため，把握が容易である．また，IP アドレスは数字の組み合わせであるため間違えやすいが，間違えてしまった場合でも誤りに気付きにくいという問題がある．名前を用いた指定は，数字の組み合わせと比較すると間違えにくいと考えられる．

4.4 記述の省略と簡略化

最後に，記述の省略や簡略化を可能にすることを考える．パラメータの冗長な指定や繰り返し同じ定義を記述することを避けるために，上位のモデルからの設定値の継承や自動補完などの機能を用意する．ここでは，ノードアドレスとデフォルトルートの記述の 2 つを例に挙げ，どのように継承や補完を行うかについて説明する．

クライアントノードの IP アドレスは，さほど重要ではない場合も多く，全てのノードについて指定を行うことは冗長である．そこで，ノードの IP アドレスについて記述が省略された場合は，ネットワーク内で未使用の IP アドレスを自動的に計算し割り当てる補完を行う．

ルーティング情報の指定は各ノードに対してそれぞれ行う必要がある．しかし，デフォルトルートの指定はネットワーク内のほとんどのノードで同じ定義になると考えられる．このような場合は，ネットワークモデルでデフォルトゲートウェイとなるノードを指定することで，ネットワーク内のノードに対して一括でその情報を継承したデフォルトルートを自動的に設定する．これに加えて，ノードのレベルでも継承された値を上書きしてパラメータを指定することができるため，一部例外があり明示的に指定したい場合でも柔軟に対応することが可能である．

5. 実装

システムモデル記述言語の実装について説明する．

5.1 ブロックの表現

パラメータや各モデル間の関係性を表現するためにブロックによる記述を可能にするが，そのためにはブロック内のコードを評価する実装が必要である．例えば，動的言語である Ruby には，引数で受け取ったブロックを受け取ったオブジェクトのコンテキストで評価して結果を返す `instance_eval` というメソッドがある．しかし，Scala には同様の処理を実現できる言語機能は用意されていない．そこで，似た構文を実現する手法として匿名クラスのインスタンス生成を利用した．モデルを宣言する際は，`Network` や `Node` などの DSL 構文として用意されているモデルのクラスを拡張した匿名クラスのインスタンスを生成する．これにより，匿名クラスとして拡張するブロック内から，そのモデルが持つ `Address` などのパラメータ指定メソッドを呼び出すことが可能になる．この手法の欠点は，インスタンス生成のキーワードである `new` が必要になってしまい，少し冗長な点である．

5.2 包含関係

各モデル間は包含関係があるため，上位のモデルは下位のモデルをリストで保持する必要がある．しかし，図 8 のように，宣言したモデルを明示的にモデルのリストへ追加する記述は冗長である．

```
nodes += new Node("Gateway") {...}
nodes += new Node("PC") {...}
```

図 8 モデルをリストに明示的に追加するコード例.
Fig.8: An code example that explicitly add models to the list.

そこで，モデルのコンストラクタ内で，上位モデルが持つリストへの追加を行う実装とした．そのためには，コンストラクタで上位のモデルのイ

ンスタンスを受け取る必要がある．これをモデル記述言語で明示的に指定することなく実現するために，Scala が持つ暗黙のパラメータ (implicit parameter) という機能を利用した．暗黙のパラメータは，引数リスト内で `implicit` 指定された引数として，その型に応じて `implicit` 宣言された値を暗黙的に選択する機能である．モデルのコンストラクタでは，暗黙のパラメータを使用して上位モデルのインスタンスを暗黙的に受け取り，上位モデルが持つリストへ生成したモデルを自動的に追加する．

5.3 パラメータ指定メソッド

パラメータの指定には，一見すると代入文のように見える構文を実現しているが，実際には代入記号のようなメソッド名を持つメソッドの呼び出しである．また，同じパラメータに対して複数のメソッドを定義することが可能である．これにより，異なる代入記号の使い分けや同じ代入記号の多重定義が可能であり，構文の柔軟性が高い．今回実装した記述方法では，代入記号として以下の 3 種類を用途に応じて使い分けている．

- `:=`メソッド
  ほとんど全てのパラメータに対して通常の代入のような意味を持つ記号として用意している．例えば，`Network` や `Node` などの IP アドレス指定や `File` のデータ指定などである．
- `@=`メソッド
  IP アドレスを指定できるパラメータの一部に対して，名前を用いた指定を行うことを意味する記号として用意している．IP アドレスの代わりに，名前を用いて接続先を指定したい場合は，この記号を用いてパラメータを指定する．
- `+=`メソッド
  ルーティング情報などの複数の値を持つパラメータに，値を追加することを意味する記号として用意している．

メソッド呼び出しによるパラメータの指定は次のように実現した．まず，各パラメータを引数なしのメソッドで定義する．そのメソッドは代入記号などの続くメソッドを持った匿名クラスを返す．この匿名クラスのメソッド内で実際のパラメータ指定を行う．Scala では，引数を持たないメソッドは呼び出し時に括弧 () を省略することができる．また，単一の引数を取るメソッド呼び出しの場合は，ドット . と括弧 () を省略することができる．これにより，冗長な括弧などを取り除いた構文を実現している．図 9 にこの実装のコードを，図 10 にパラメータ指定時のコード例を示す．

```
def DefaultGateway = new {
  def :=(flag: Boolean) = {...}
  def :=(address: String) = {...}
  def @=(connection: String) = {...}
}
```

図 9 パラメータ指定の実装.
Fig.9: An implementation of the parameter specification.

```
Node("Server") {
  DefaultGateway := "192.168.2.1"
}
Node("Router") {
  DefaultGateway @= "Gateway"
}
Node("Gateway") {
  DefaultGateway := false
  Route += "192.168.1.0/24" ->
       "192.168.1.1"
  Route += "192.168.2.0/24" ->
       "192.168.2.1"
}
```

図 10 パラメータ指定の例.
Fig.10: An example of the parameter specification.

図 10 の例では，パラメータ (`DefaultGateway`) に対して，様々な型や代入記号を用いて指定を行っている例を示している．`DefaultGateway` に対する，引数の型とメソッド名の組み合わせによるパラメータ指定の意味は以下の通りである．

String 型を引数に取る:=メソッド デフォルトゲートウェイとなるノードの IP アドレスを文字列で指定する．

String 型を引数に取る@=メソッド デフォルトゲートウェイとなるノードを名前を用いて文字列で指定する．

Boolean 型を引数に取る:=メソッド 上位モデルのデフォルトゲートウェイ設定を継承するか否かを真偽値で指定する．なお，この指定のデフォルト値は `true` であり，上位モデルのデフォルトゲートウェイ設定は自動的に継承される．そのため，明示的にデフォルトゲートウェイ設定の継承を行いたくない場合のみ，`false` を指定することになる．

`DefaultGateway` に `false` を指定し，上位モデルのデフォルトゲートウェイ設定を継承しなかった場合は，図 10 の例のように，パラメータ (`Route`) を用いてルーティング情報を指定することがほとんどであると考えられる．ルーティング情報は，複数の値を持つパラメータであるため，指定には `+=`メソッドを使用する．また，一見すると単一の引数ではないように見えるが，`Route` の`+=`メソッドは要素数 2 のタプルのみを引数に取る．Scala では`->`を用いると要素数が 2 であるタプルが作られるため，実際に`+=`メソッドに渡される値は一つである．

6. 既存のネットワーク記述言語との比較

最後に，実装したモデル記述言語を既存のネットワーク記述言語と比較する．

金子佳正らによる「ネットワーク記述言語を用いたネットワーク設計支援に関する一考察」[7] では，ネットワーク設計を支援するために XML を用いたネットワーク記述言語を提案している．これは，記述した抽象的な要求モデルから具体的な機器まで決定した実装モデルへの変換を行うことで，ネットワーク設計を支援することを目的としている．脅威トレースも情報システムの設計を支援するものではあるが，高度標的型攻撃への内部対策となるシステム設計策を検討する作業を支援するものであり，目的が異なる．そのため，脅威トレースでトレース対象となるシステムのモデル記述やパラメータの設定を行う記述言語としては適していない．また，XML を用いているため記述が冗長であり，構文の拡張性が高いとはいえないが，本研究で実装した記述言語では，簡潔な記述と将来における構文の柔軟性・拡張性を実現している．

知念賢一らによる「Kuroyuri: ネットワーク実験記述言語処理系」[8] では，ネットワーク実験を自動化するためのスクリプト言語を提案している．これは，実際の機器を利用したネットワーク実験において，機器の設定などのネットワーク構築や制御，実験の処理を記述できるスクリプト言語により，それらを自動化するものである．ネットワークにおけるシミュレーションを行う点では，脅威トレースも同様であるが，これは実際の機器を対象としていることに対して，脅威トレースでは抽象化されたモデルを対象とする点において異なる．そのため，抽象度が低く，脅威トレースにおけるモデル記述言語としては適していない．しかし，このスクリプト言語では，実験の手順やノードの挙動であるシナリオを扱うことができるため，その点において本研究で実現した記述言語よりも優れているといえる．脅威トレースでも攻撃シナリオを扱うため，モデル記述言語と合わせて記述できるようにすることで，情報システムを設計する際に，それらをまとめて扱えるようになり，よりモデル記述言語を便利なものにできると考えられる．

本研究で実現した記述言語は，既存のネットワーク記述言語よりも脅威トレースで扱うモデルの抽象度に合わせたものとなっており，モデル記述の負担の軽減を達成できた．また，内部 DSL として開発したことにより脅威トレースの開発自体とも親和性が高く，今後の脅威トレースの改良に合わせてモデル記述言語を拡張していくことも容易である．

7. おわりに

本研究では，脅威トレースへトレース対象となる情報システムの記述を容易にするための情報システム記述言語を実装した．脅威トレースで用いられるモデル間の関係性をブロックによる包含関係で表現し，記述の省略・簡略化を可能にすること

で記述・保守が容易なモデル記述言語を実現した．

今後の課題としては，6 章でも述べたように，攻撃シナリオの記述を行えるようにすることで，モデルの記述のみでなく，脅威トレースによるシミュレーション手順を統合して扱えるようにしたいと考えている．また，脅威トレース自体が開発途上であるため，今後も様々なモデルや機能が追加されることが想定されるが，その際に，モデル記述言語に自動的に構文を追加できるような仕組みを実現したいと考えている．

謝辞 本研究は独立行政法人情報処理推進機構の支援および，JSPS 科研費 24500043 の助成を受けたものです．

参考文献

[1] IPA 独立行政法人 情報処理推進機構：『高度標的型攻撃』対策に向けたシステム設計ガイド，pp.19-20(オンライン)，入手先〈http://www.ipa.go.jp/files/000042039.pdf〉(参照 2014-12-01)．
[2] 同上，p.2．
[3] IPA 独立行政法人 情報処理推進機構：サイバー情報共有イニシアティブ（J-CSIP）2013 年度 活動レポート～「やり取り型」攻撃に関する分析情報の共有事例 ～，pp.20-29(オンライン)，入手先〈https://www.ipa.go.jp/files/000039231.pdf〉(参照 2014-12-01)．
[4] IPA 独立行政法人 情報処理推進機構：ウェブサイト改ざんの脅威と対策～ 企業の信頼を守るために求められること ～，p.2(オンライン)，入手先〈http://www.ipa.go.jp/files/000041364.pdf〉(参照 2014-12-01)．
[5] IPA 独立行政法人 情報処理推進機構：前掲『高度標的型攻撃』対策に向けたシステム設計ガイド，p.21．
[6] Al-Shaer, E., Ou, X. and Xie, G. (Eds.): Automated Security Management, Kato, M., Matsunami, T., Kanaoka, A., Koide, H. and Okamoto, E.: Tracing Advanced Persistent Threats in Networked Systems, pp.179-187, Springer International Publishing (2013).
[7] 金子佳正，武田利浩，平中幸雄：ネットワーク記述言語を用いたネットワーク設計支援に関する一考察，FIT2010(第 9 回情報科学技術フォーラム)，第 4 分冊，L-002，pp.167-168 (2010)．
[8] 知念賢一，宮地利幸，篠田陽一：Kuroyuri: ネットワーク実験記述言語処理系，日本ソフトウェア科学会，Vol.27，No.4，pp.43-57 (2010)．

付　　録

A.1 システムモデル記述言語で記述したモデル例

```
new Model {
  new Network("global") {
    Address := "192.168.2.0/24"
    DefaultGateway := "192.168.2.1"
    new Node("Router") {
      Address := "192.168.2.1"
      DefaultGateway @= "Gateway"
      Route += "192.168.2.0/24" -> "192.168.2.1"
    }
    new Node("Gateway") {
      Address := "192.168.2.2"
      DefaultGateway := false
      Route += "192.168.0.0/24" -> "192.168.1.2"
      Route += "192.168.1.0/24" -> "192.168.1.1"
      Route += "192.168.2.0/24" -> "192.168.2.1"
    }
    new Node("Web Server") {
      Address := "192.168.2.3"
      new WebServerApp
    }
    new Node("Taint Web Server") {
      Address := "192.168.2.4"
      new TaintWebServerApp {
        CCServer @= "Command and Control Server"
      }
    }
    new Node("Command and Control Server") {
      Address := "192.168.2.5"
      new CCServerApp
    }
  }
  new Network("dmz") {
    Address := "192.168.1.0/24"
    DefaultGateway @= "Gateway"
    new NodeConnection {
      Node @= "global/Gateway"
```

```
    Address := "192.168.1.1"
  }
  new Firewall("Firewall") {
    Address := "192.168.1.2"
    Route += "192.168.0.0/24" -> "192.168.0.1"
    Route += "192.168.1.0/24" -> "192.168.1.1"
  }
  new Node("Mail Server") {
    Address := "192.168.1.6"
    new MailServerApp
  }
}
new Network("private") {
  Address := "192.168.0.0/24"
  DefaultGateway @= "DMZ-Firewall"
  new NodeConnection("DMZ-Firewall") {
    Node @= "dmz/Firewall"
    Address := "192.168.0.1"
  }
  new Node("Proxy") {
    new ProxyServerApp
  }
  new Node("PC1") {
    new WebBrowserProxyApp {
      ProxyServer @= "Proxy"
    }
    new SmbApp
    new MailerApp {
      User := "user1"
      MailServer := "192.168.1.6"
    }
    new File("ID_Password.db") {
      Data := "ID:hoge, Pass:1234"
    }
    new File("NetworkSettings.conf") {
      Data := "Apache's password is 'webmaster8080'"
    }
  }
  new Node("PC2") {
    new WebBrowserApp
    new SmbApp
```

```
      new File("devApplication.exe") {
        Data := "-*-binary file-*-"
      }
    }
    new Node("PC3") {
      new WebBrowserApp
    }
  }
}
```