平成 21 年度卒業論文

論文題目

Flash を用いた Covert Channel の視覚化

神奈川大学 工学部 電気電子情報工学科
学籍番号　200502753
新目　拓海

指導担当者　木下宏揚　教授

目 次

図 目 次

1 序論

現在、Social Network Service（SNS）はPC，モバイル環境ともにインターネット上において急速な広がりをみせている．これは今後も一層の普及と発展が予想されるが，それに伴い，様々な問題が浮き上がってくる．その中の技術的な問題の一つとして，Covert Channel(隠れた経路)[1] が挙げられる．Covert Channel とはCommunity[2] において，間接的に情報漏洩，改竄が起こりうる経路のことである．これを本研究では主要OS，ブラウザに対応し，一般に広く普及しているFlash[5] を用いて視覚化を行う．これによりCovert Channel情報に対し，容易にアクセスができる環境が整う．また，Covert Channelを視覚化することにより，情報漏洩，改竄の経路を明確化させることが可能となり，情報漏洩，改竄の発見や防止，解析等を早急かつ容易行うことができる．本研究室では，SNSで起こりうる問題に対処すべく，Covert Channelに関連する研究を年々行ってきた．その例として小松充史氏の2006年度修士論文 Covert Channel分析制御のための推論を導入した情報フィルタに関する研究[3] や西田 学氏の2007年度卒業論文 Covert Channel分析と情報フィルタ選択のための推論機能の提案[16]，また，本学講師であり，本研究室所属である森住 哲也氏の研究[4] ほか[1][2] などがあり，多くのCovert Channelに関する研究の積み重ねがなされてきた．今回，その研究の一環として，Covert Channelの未開の分野であるCovert Channelの視覚化に着手し，本研究を行った次第である．

2 基礎知識

2.1 Covert Channel

Covert Channel[1] はアクセス行列において，Subject，Object，permission をアクセストリプルと定義したとき，その 3 点で起きる不正な情報経路である，この場合，Covert Channel はアクセス禁止のパミッションに矛盾する情報フロー（アクセス禁止のものもこのフローを使えばアクセス禁止の内容を閲覧したり，書き換えることができてしまう）ともいう.（図 2.1 参照 各 S=Subject，各 O=Object，R=読みこみ可能権限，W=書き込み可能権限）図 2.1 の場合矢印の流れで Subject2 が本来読み込むことのできないの Object1 を読めてしまう．Covert Channel 流出の流れは以下のようになっている．これは間接情報フローとも呼ばれる．

2.1.1 間接情報フロー

- 始点（Subject1・Object1）Subject1 が Object1 を読み込む
- 中間点 1（Subject1・Object2）Subject1 が Object2 に Object1 で読んだ内容を書き込む
- 中間点 2（Subject2・Object2）Subject2 が Object2 を読む
- 終点（Object1・Subject2）Covert Channel により間接的に Object1 の内容を読めてしまう

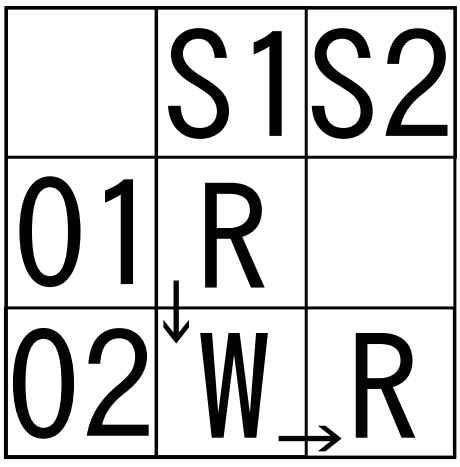

図 2.1: Covert Channel(間接情報フロー)

2.1.2 実際に発生する Covert Channel

不正な情報経路である Covert Channel を全て塞いでしまえば安全なシステムを構築することができるように見えるが, 単独では Covert Channel（隠れた経路）が存在しないようなコンピュータでもネットワークに接続されたコンピュータ群が協調することによって, Covert Channel を構成できてしまう。つまり, 単独では安全なコンピュータでも, それがネットワークを構成すると安全ではなくなるような状況が簡単に存在し得るのである. このようなネットワーク構成機能の問題点が Covert Channel で利用される. 例えば以下のような例が挙げられる.

- 会社の機密データを社外へ持ち出したり、社外の人間（社外の PC）でも見られるようにする.
- SNS の個人データが掲示板やブログ等不特定多数へ流出スパイウェア等, 個人 PC から情報を持ち出すためにこれを用いて通信を行い, 検知を困難とする.

WWW 等、不特定大多数が利用するネットワークでは意図しなくてもカバートチャンネルが発生してしまう恐れがあるのでそういった情報網では比較的安易に情報漏洩が起こりうる. このように Covert Channel は今のネットワーク社会にとって情報を安易に流出させてしまう存在なのである.

2.2 Community

Community(コミュニティ)[2] にも様々な種類があるが, インターネット上の Community を考えてみると主体の役割や利害関係, 或いはプライベートな情報 (個人情報) が複雑に絡み合っている. ここで Community の定義を次に示す.

Community とは，Community の属性，Community に属する Subject，および，Community が管理する Object とその属性の集まりから成る社会システムである.

[**定義**]Community **の属性の定義**
(1) 競合：利害関係のある `Community` 間の関係
(2) 連携：協調して情報を産出する `Community` 間の関係
(3) 独立：競合，連携に属さない独立した無関係の `Community` の関係
(4) 名前：`Community` の名前
本稿では (1) (2) (3) を総じて競合と呼ぶこととする.

[**定義**]Community **内部の属性の定義**
Community 内で Subject に割り当てる属性として，役割，階層（セキュリティレベル）と言う属性を持つ．役割とは，Community 内で組織的に決められるものであり職位等が相当する．

2.3 Subject

Subject[3] はデータベースに格納されている Object にアクセスする行為者である．Subject はアクセス行列では Subject の名前や，その Subject の Community に於ける役割 (次長，課長等) やセキュリティレベルとして表現される．Subject の属性の定義を以下に示す.

[**定義**]Subject **の属性**
名前属性：`Subject` 名
競合属性：`Subject` が属する `Community` から刻印される競合属性
役割属性：`Subject` の役割
階層属性：`Subject` のセキュリティレベル
プライベート属性：プライベートな情報に関する属性

2.4 Object

Object[3] はデータベースに格納する情報である．Object の属性の定義を以下に示す.

[**定義**]Object **の属性**
名前属性：データベースに格納するための名前
所有属性：`Object` を所有する `Subject`
競合属性：`Object` から刻印される競合属性
役割属性：`Object` にアクセスする `Subject` の役割
階層属性：`Object` のセキュリティレベル
プライベート属性：`Subject` の個人情報と言う属性
記号属性：メタ言語，レトリックの区分

2.5 permission

本研究において Subject は Object にアクセスする行為者,Object は格納されているデータであるが, Subject,Object に定義される属性を以下に示す. 所有・競合・役割・階層・プライベートの５つが permission[3] に深く関連してくる. Community

内では Subject・Object に属性が定義されこの属性により Permission（権限）が決定する．

[**定義**]Permission
`READ` 権限：`Subject` が `Object` の情報を参照する（読む）ことができる権限
`WRITE` 権限：`Subject` が `Object` に情報を書き込むことができる権限
`READWRITE` 権限：２つの権限のどちらもある場合の権限

2.6 ActionScript

ActionScript[5] は Adobe の Flash プラットフォームにおける公式のプログラミング言語である．元々はアニメーションを成業するための簡単なツールとして着想されたが，Web やモバイルデバイス，デスクトップコンピュータ用のコンテンツやアプリケーションを作成するための洗練されたプログラミング言語へと進化していった．最新版は ActionScript3.0 であり，コア言語は ECMAScript4 として策定が進められていた ECMAScript 言語仕様第 4 版に基づいている．ActionScript3.0 の主要なコア言語機能には次のものがある．

- クラスやオブジェクト，インターフェイスといった通常使用されるオブジェクト指向的な構成概念のためのファーストクラスサポート
- シングルスレッド実行モデル
- ランタイム型チェック
- コンパイル時型チェック
- 実行時に新しいコンストラクタ関数や変数などを作成するダイナミックな機能
- ランタイム例外
- XML をビルドインデータ型として直接サポートする機能
- コードライブラリを編成するためのパッケージ
- 識別子を修飾する名前空間
- 正規表現

2.6.1 Flash ランタイムのクライアント

ActionScript プログラムは Adobe AIR[5]，Flash Player[5]，Flash Lite[5] という３つの異なるクライアントランタイム環境で実行することができる．Adobe AIR

は，デスクトップ，FLash Player は Web，Flash Lite はモバイルデバイスというように各環境で Flash プラットフォームアプリケーションを実行できる．本研究においては Flash Player での実行，すなわち Web 環境での実行を目的とするため，Flash Player を利用する．

2.6.2 Flash ファイル形式 (SWF)

ActionScript コードを Adobe の Flash クライアントランタイムで再生するには，コードを.swf ファイルにコンパイルする必要がある．一般的には Flash の呼称で SWF 形式の実行ファイルを指し，本論文においてもその意味として Flash という言葉を用いる．

2.6.3 ActionScript 開発ツール

Adobe は ActionScript コードの作成するためには次の 3 つのツールを提供している．Adobe Flash[5]，Adobe Flex Builder[5]，Adobe Flex 3 SDK[5]．本研究では Adobe Flex Builder を使用する．Adobe Flex Builder は純粋な ActionScript またはユーザーインターフェイスを表すための XML ベース言語である MXML を使ってコンテンツを作成するための統合開発環境 (IDE:Integrated Development Environment) である．IDE とは，コードを記述し管理するためのアプリケーションのことを言う．開発者は Flash オーサリングツールを使って，ActionScript コードと，手作業で描画したグラフィックやアニメーション，マルチメディアのセットを結びつけることで，ソフトウェアアプリケーションやマルチメディアコンテンツを作成する．Flex Builder には Flex フレームワークと呼ばれる．プログラミングユーティリティとスキンやスタイルの変更可能なユーザーインターフェイスコントロールのさまざまなセットを提供する開発フレームワークが含まれ、Flex Builder は有名なオープンソースプログラミングツールである Eclipse の上に構築されており、手動によるコード記述モードと Microsoft の Visual Basic のような視覚的な開発モードの両方で使用できる．

2.6.4 文法

ActionScript1.0 は文法が JavaScript に似ているが、ActionScript2.0 からはクラスベースのオブジェクト指向言語になり Java に似通っている。 ActionScript ではすべてのデータをオブジェクトと見なしている。ここでは ActionScript3.0 の文法を説明していく。

- 変数の宣言

  変数の宣言には var 変数名:変数の型 と書く。

var num:int;

- 関数の宣言

  関数の宣言には function 関数名 (引数 1:引数 1 の型, 引数 2:引数 2 の型,...):戻り値の型 実行するコード と書く。

```
function sum(a:Number,b:Number):Number
{
  return a + b;
}
```

  戻り値がない場合は戻り値の型を void とする。void と宣言した場合、戻り値は undefined になる。

- クラスの宣言

  クラスの構文は Java に似通っている。クラスとコンストラクタは必ず public にしなければならない。コンストラクタは省略可能である。 ファイル名はクラス名.as とし、パッケージと同じディレクトリに置かなければならない。

```
package パッケージ名
{
  public class クラス名
  {
     //コンストラクタ
     public function クラス名()
     {
     }
  }
}
```

2.7 XML

XML(Extensible Markup Language，拡張可能なマーク付け言語)[6] は，個別の目的に応じたマークアップ言語群を創るために汎用的に使うことができる仕様であり，World Wide Web Consortium (W3C) により勧告 （策定）されている国際標準の構造化文書の技術である． 1998 年 2 月に XML 1.0 が勧告された．XML は現在，広く普及している技術である．

3 Covert Channel の視覚化ついての提案

1 序論で既に触れたが Covert Channel を視覚化することにより，情報漏洩，改竄の経路を明確化させることが可能となり，情報漏洩，改竄の発見や防止，解析等を早急かつ容易にさせることができる．また，視覚化により専門のシステム管理者以外だけではなく，一般的な情報リテラシーを持つ管理者であれば充分に運用できることを目的としている．よって Covert Channel の視覚化を提案する．なお，この視覚化にあたっては Covert Channel 分析により Covert Channel を検出し，そのデータを出力するプログラムがあるものと仮定して行う．

3.1 XML の利用と形式定義

XML 形式にて．Community 内における設定されたアクセス権限，そしてその設定されたアクセス権限によって発生する Covert Channel，この 2 つのデータが XML 形式によって出力されると仮定し，前者のデータが出力されるファイル名を am.xml(am は access matrix の略) に，後者のデータが出力されるファイル名を cc.xml(cc は Covert Channel の略) として用意する．この 2 つのファイルを比較することにより，アクセス権限内に covert channel が存在するか否かを振り分ける．これら 2 つの XML の記述形式定義は同一であり，以下にその定義すべき最小単位を示す.（何故，XML を用いるのかについては 3.2 Flash を用いた視覚化に記述する.）

```
<accessmatrix>
 <node>
  <name>subject1</name>
 </node>
</accessmatrix>
```

上記の各要素の内容定義は以下の通りである．

- accessmatrix

  ルート要素．Community 内における設定されたアクセス行列.

- node

  Subject，あるいは Object のノード情報.

- name

  Subject，あるいは Object の名前.

次にノードにアクセス権限が設定されている場合，node 要素に arc 要素が挿入される．その arc 要素が挿入された例を以下に示す.

```
<accessmatrix>
 <node>
  <name>subject1</name>
  <arc>
   <type>out</type>
   <label>read</label>
   <neighbornode>object1</neighbornode>
  </arc>
 </node>
</accessmatrix>
```

上記の挿入された各要素の内容定義は以下の通りである.

- arc

  arc 要素が設定されたノードのアクセス権限情報.

- type

  out，あるいは in が設定され，out であればアクセス権限を使役する側，in であれば使役される側となる.

- label

  read，あるいは write が設定され，read であれば，read 権限（情報の読み込み）許可，write であれば write 権限 (情報の書き込み) 許可となる.

- neighbornode

  type 要素，label 要素の設定に従い，name 要素の内容名の Subject，あるいは Object に対してアクセス権限が設定された Subject，あるいは Object の名前.

上記の例では subject1 は object1 に対して read 権限があることを示している．本節の最後に図 2.1 Covert Channel(間接情報フロー) で示した Covert Channel が存在するアクセス行列を以上で定義した XML による記述定義形式により書き直したものを以下に示す.

```
<accessmatrix>
 <node>
  <name>subject1</name>
  <arc>
   <type>out</type>
   <label>read</label>
```

```
   <neighbornode>object1</neighbornode>
  </arc>
  <arc>
   <type>out</type>
   <label>write</label>
   <neighbornode>object2</neighbornode>
  </arc>
 </node>
 <node>
  <name>subject2</name>
  <arc>
   <type>out</type>
   <label>read</label>
   <neighbornode>object2</neighbornode>
  </arc>
 </node>
 <node>
  <name>object1</name>
  <arc>
   <type>in</type>
   <label>read</label>
   <neighbornode>subject1</neighbornode>
  </arc>
 </node>
 <node>
  <name>object2</name>
  <arc>
   <type>in</type>
   <label>write</label>
   <neighbornode>subject1</neighbornode>
  </arc>
  <arc>
   <type>in</type>
   <label>read</label>
   <neighbornode>subject2</neighbornode>
  </arc>
 </node>
</accessmatrix>
```

- 1 番目の node 要素では，subject1 が object1 に対して read 権限，object2 に対して write 権限を持つことを示している．
- 2 番目の node 要素では，subject2 が object2 に対して read 権限を持つことを示している．
- 3 番目の node 要素では，object1 が subject1 の持つ read 権限によって使役されることを示している．
- 4 番目の node 要素では，object2 が subject1 の持つ write 権限によって使役されること，また subject2 の持つ read 権限によって使役されること，を示している．

3.2 Flash を用いた視覚化

3 Covert Channel の視覚化ついての提案で記述したように Covert Channel の視覚化にあたって ActionScript でプログラムを制作し，Flash にて実行する．視覚化の手段として Flash を用いる理由として以下のことが挙げられる．

- プレイヤーの高い普及率と汎用性

  Flash 6 以上の普及率は 100%に近く，最新ヴァージョンである Flash 10 の普及率も高い [14][15]．また，Microsoft Windows，Mac OS X，Linux OS などのオペレーティングシステム上で動作し，Internet Explorer や Firefox などの代表的な Web ブラウザの中でプラグインとして動作させることもできるため汎用性が高い．

- 視覚化にあたって，充実した ActionScript のクラス

  元々，Flash で使用するために拡張された言語であるため，標準で本研究の視覚化に即したクラスが充実している．

- インターネットとの親和性

  プレイヤーの高い普及率と汎用性　で述べたように Web ブラウザのプラグインが用意されているため，インターネット環境が整っていることでアクセスが可能であるため，インターネットとの親和性が高く，場所にとらわれない容易なアクセスが可能である．

- XML との親和性

  Flash では標準で XML のデータ型が用意されているため，XML 形式のデータを扱いやすく親和性が高い．また，XML 自体が広く普及しているため，汎用性がある．

以上のことより，視覚化に Flash は最適であると判断し，Flash を採択した．

3.3 視覚化の表現方法

3.1 XML の利用と形式定義に記述した XML 形式にて，アクセス権限が出力された am.xml の形式定義に基づき name 要素のノードを生成する．次に 3.1 に記述した am.xml と Covert Channel が存在するアクセス権限が出力された cc.xml との比較を行い，Covert Channel が存在する node 要素とそうではない要素を仕分けし，各 node 要素に含まれる arc 要素に含まれる type 要素，label 要素の内容，また仕分けされた Covert Channel の有無に基づきを既に生成された name 要素と同名のノードから neighbornode 要素と同名のノードへ矢印を引く．その際，惹かれる矢印の色を label 要素が read の場合は青，write の場合は赤で色分け，Covert Channel の有無によって色の濃度を変更する．以上によって Community 内に設定されたアクセス権限に存在する Covert Channel の視覚化を行う．これをもって 1 つの視覚化表現方法として提案する．

4 プログラム制作

2.6 ActionScript に記述した ActionScript を用いてプログラミングを行い，Covert Channel 視覚化プログラムを制作した．本章ではその制作過程と試行について述べ，各コードの解説を行う．

4.1 開発環境

プログラミングにあたり，Flex Builder を用いたが，MXML は使用せず，ActionScript のみでコードを記述したため，デザイン設計としての機能は使用せず，ActionScript のプログラミングエディタとして使用した．

4.2 Flex と ActionScript の違い

Flex は 2.6 ActionScript に記述したように MXML によるデザインを含めた ActionScript によるプログラミングとなり，独自のコンポーネントパッケージを持っている．それにより，Flex と ActionScript のみでは同様のプログラミングを行う場合であっても，一部異なるプログラミングを行うことがある．

4.3 ActionScript ライブラリの利用

ノードを生成し，接続することによりグラフを生成する ActionScript ライブラリ [13] が存在するため，それを本研究用に改造を施した．

4.4 XML の読み込み

ActionScript ライブラリの改造にあたり，まず，XML の読み込みに着手した．この時点では Flex(MXML) の利用を考えていたため，Flex 独自のコンポーネントを用いての XML 読み込みと ActionScript での読み込みのコードを考慮し，試行したが，ActionScript ライブラリ自体が ActionScript のみの AS 形式のファイルで構成されていたため，Flex 用に変更する際の不具合を回避するために ActionScript のみでのプログラミングで研究制作とすることを決定し，ActionScript での XML 読み込みを試行した．ActionScript によるコードを以下に示す．なお各コードの説明はコード上部に「//コードの説明」という形で記載し，以後もその形式でコードの説明を行うものとする．

```
//XML 型変数の宣言
public var amxml:XML = new XML();

//String 型変数を宣言し，am.xml という文字列を代入する．
public var amxmlurl:String = "am.xml";
```

```
//URLRequest クラスを使用し，文字列 am.xml を代入．ファイル am.xml をロードするための情報をセット．
public var amrequest:URLRequest = new URLRequest(amxmlurl);

//URLLoader クラスを使用し，ファイル am.xml のデータをダウンロード．
public var amloader:URLLoader = new URLLoader(amrequest);
```

そして，実行する関数側にて

```
//data プロパティにてダウンロードしたデータを XML 型変数 amxml に代入．
amxml = XML(amloader.data);
```

これにより，am.xml に出力されたデータを ActionScript にて操作することが可能となる．

4.5 XML の扱い方

アクセス権限が出力される am.xml，CovertChannel が出力される cc.xml を基に今後必要となる XML 要素の内容の抽出を示す．抽出には

> [メソッド]
> toString()：XML オブジェクトのストリング表現を返す．

を使用する．例として 3.1 XML の利用と形式定義に記述した XML を使用する．

```
<accessmatrix>
 <node>
  <name>subject1</name>
  <arc>
   <type>out</type>
   <label>read</label>
   <neighbornode>object1</neighbornode>
  </arc>
 </node>
</accessmatrix>
```

データが格納されている XML 型変数名を amxml とすると

- name 要素の内容を抽出する場合
  amxml.node[0].name.toString()

- type 要素の内容を抽出する場合

  amxml.node[0].arc[0].type.toString()

- label 要素の内容を抽出する場合

  amxml.node[0].arc[0].label.toString()

- neighbornode 要素の内容を抽出する場合

  amxml.node[0].arc[0].neighbornode.toString()

といったように記述することにより各要素の内容を抽出することができる．[ ] で囲まれている数字は要素内における番号を示し，0 から始まる．

4.6 ノード生成

ノード生成にはファイル Graph.as に定義された独自関数 createNode を使用し，以下のように記述する．

`graph.createNode(生成する X 座標, 生成する Y 座標,{ color: 生成するノードの色，text: ノード名 })`

これに対し，ノード名の部分に am.xml に出力されたデータの name 要素の内容を入力することにより，Subject あるいは Object の名前を入力することができる．実際のノード生成については for を利用し，以下のように記述し，am.xml に出力された全ての name 要素をノード名としてノードの生成を行う．

`//length()` メソッドにより `node` 要素の数を返す．これにより `node` 要素の数だけノード生成を行う．

```
for (i1 = 0; i1 < amxml.node.length(); i1++) {
```

`//allnode` はオブジェクト型変数であり，`name` 要素を動的プロパティとして作成する．

```
allnode[amxml.node[i1].name.toString()] =
graph.createNode(生成する X 座標, 生成する Y 座標,{ color: 生成するノードの色,text: amxml.node[i1].name.toString() });
}
```

Object 型変数である allnode を宣言し使用する理由は，ノード生成後にノード同士の接続を行うためである．この際，ノード操作，およびノード判別のために allnode の動的プロパティを各 name 要素とし，独自関数 createNode(ノード情報) を格納する．なお，最初は Array 型を用いて XML の要素番号を用いた数値インデックスによる多次元配列に createNode を格納し，ノード操作を行う方法を模索した

が，プログラム自体が複雑となり，計算量の増加が懸念されるため，キー (String 型の文字列) を用いたハッシュまたはマップとも呼ばれる結合配列を使用することにした.（結合配列は一般的に連想配列と呼ばれるものと同じである.）結合配列自体は Array 型であっても利用は可能だが，Array 型のプロパティやクラスのメソッドが利用できず，Array 型を使用するメリットが無いため，Object 型による結合配列を行った次第である.

4.7 ノードの接続

4.6 ノード生成によって生成されたノードの接続には Node.as に定義された独自関数 connect を使用し，以下のように記述する.

```
Subject1(接続する node).connect(Object1(接続される node),Subject1 が Object1
に対する type 要素,Subject1 が Object1 に対する label 要素,Covert Channel
かそうでないか);

//実際に定義通り代入した場合 (Subject1 が Object1 を read し，これは Covert Channel
となる場合)
Subject1.connect(Object1,out,read,"cc");
```

元々，ActionScript ライブラリにおける connect の定義では引数が Object1(接続されるノードオブジェクト) のみであったが，拡張を行い，Subject1 が Object1 に対する type 要素 (in あるいは out)，Subject1 が Object1 に対する label 要素 (read あるいは write),Covert Channel かそうでないか (am か cc) という 3 つの引数を定義した．これらの引数は 4.9 矢印作成および，4.10 色設定で使用するために必要となる.

4.8 am.xml と cc.xml の比較判別

生成したノードを XML 形式定義に基づきノードを接続する前にアクセス権限が出力される am.xml，CovertChannel が出力される cc.xml の比較判別を行う必要がある．am.xml と cc.xml の形式定義は同じであり，出力される各 Subject あるいは Object アクセス権限の設定数は必然的に cc.xml ⊇ am.xml となるが，比較により重複したアクセス権限の部分を Covert Channel，am.xml にのみ存在するアクセス権限の部分を Covert Channel ではないと判別する．以下に比較判別コードを示す.

```
//ccnode という Object 型を宣言しする.
var ccnode:Object = new Object();

//ccxml の node 要素数だけ実行する
for (i1 = 0; i1 < ccxml.node.length(); i1++) {
```

```
//ccxml の node[i1] 要素の arc 要素数だけ実行する
for (i2 = 0; i2 < ccxml.node[i1].arc.length(); i2++) {

//動的プロパティに cc.xml に出力された「接続する node 名 + 接続された node
名」を作成する.
ccnode[ccxml.node[i1].name.toString() +
ccxml.node[i1].arc[i2].neighbornode.toString()];
 }
}

//amxml の node 要素数だけ実行する
for (i1 = 0; i1 < amxml.node.length(); i1++) {

//amxml の node[i1] 要素に arc 要素が存在する場合，実行する
if (0 < amxml.node[i1].arc.length()){

//amxml の node[i1] 要素の arc 要素数だけ実行する
for (i2 = 0; i2 < amxml.node[i1].arc.length(); i2++) {

//ccnode の動的プロパティに「amxml の node[i1] 要素の name 要素 + amxml の
node[i1] 要素の arc[i2] 要素の neighbornode 要素」が存在する場合
if (ccnode[amxml.node[i1].name.toString() +
amxml.node[i1].arc[i2].neighbornode.toString()]){

//「allnode の amxml の node[i1] 要素の name 要素と同名の動的プロパティに含
まれる node」と「node[i1] 要素の arc[i2] 要素の neighbornode 要素と同名の動
的プロパティに含まれる node」を接続する.
allnode[amxml.node[i1].name.toString()].connect(
allnode[amxml.node[i1].arc[i2].neighbornode.toString()],

//引数として allnode の amxml の node[i1] 要素の type 要素 (in あるいは out),
label 要素 (read あるいは write), 文字列 cc(条件式に当てはまるものは Covert Channel
であるため)
amxml.node[i1].arc[i2].type.toString(),
amxml.node[i1].arc[i2].label.toString(),"cc");
}
```

```
//Covert Channel でない場合
else {

    //ノードの接続に関しては最後の引数の文字列が cc ではなく am 以外は同じ
である.
    allnode[amxml.node[i1].name.toString()].connect(
    allnode[amxml.node[i1].arc[i2].neighbornode.toString()],
    amxml.node[i1].arc[i2].type.toString(),
    amxml.node[i1].arc[i2].label.toString(),"am");
   }
  }
 }
}
```

以上により，am.xml と cc.xml の比較判別を行う．この比較判別の処理速度が大量のノード接続処理に影響を与えるため，重要なコードの一つと言える．

4.9 矢印作成

4.8 am.xml と cc.xml の比較判別ではノードの接続自体は行ったが，このままではアクセス権限の流れが視覚的に判別不能であるためノードからノードに対して矢印を引く必要がある．ActionScript には矢印を作成するクラスが存在しないため，接続されるノード側の終端に三角形を作成し，矢印を作成する．そのコードと作成過程図を以下に示す.

4.9.1 変数定義

```
//Math クラスの atan2 メソッドにより座標 (dest.y,dest.x) の円の X 軸からの
角度 (ラジアン単位) を Number 型変数 angle に代入する.
var angle:Number = Math.atan2(dest.y,dest.x);

//矢印の長さを設定する.(任意の数値)
var r:Number = 50;

//矢印の鋭さ (角度) を設定する.(任意の数値)
var diffAngle:Number = Math.PI/6;

//Point クラスの polar メソッドにより極座標を直交座標に変換する. polar(極
座標の長さ, 極座標の角度).
var polar:Point = Point.polar(r,angle+diffAngle+Math.PI);
```

```
polar = new Point(dest.x+polar.x,dest.y+polar.y);
var polar2:Point = Point.polar(r,angle-diffAngle+Math.PI);
polar2 = new Point(dest.x+polar2.x,dest.y+polar2.y);
```

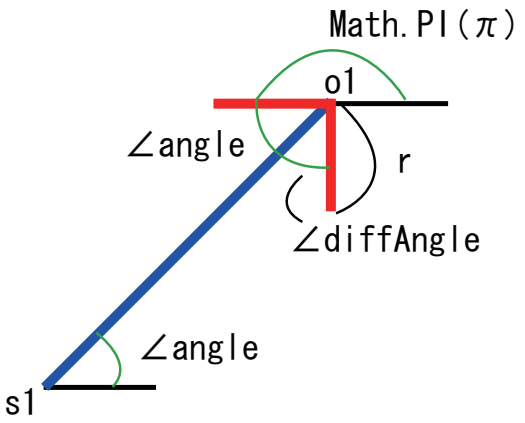

図 4.1: 矢印作成定義

4.9.2 矢印描画

//4.7 ノードの接続で示した独自関数 connect の引数である type 要素が out の場合

```
if(type == "out") {

//lineTo(x 座標,Y 座標)…線の終点を定義. 3.1 XML の利用と形式定義で示した定義に従い，アクセス権限を使役する側から使役される側へ矢印を引く.
graphics.lineTo(dest.x,dest.y);

//
graphics.moveTo(polar.x,polar.y);
graphics.lineTo(dest.x,dest.y);
graphics.lineTo(polar2.x,polar2.y);
}
```

type 要素が out の場合のみ矢印を引く理由は，type 要素が in(使役される側) を除外しなければ，接続する両端に矢印が描画され，視覚的にアクセス権限を理解できなくなるためであり，使役する側から使役される側へのみ矢印を引くことで直感的にアクセス権限の内容を知ることができる.

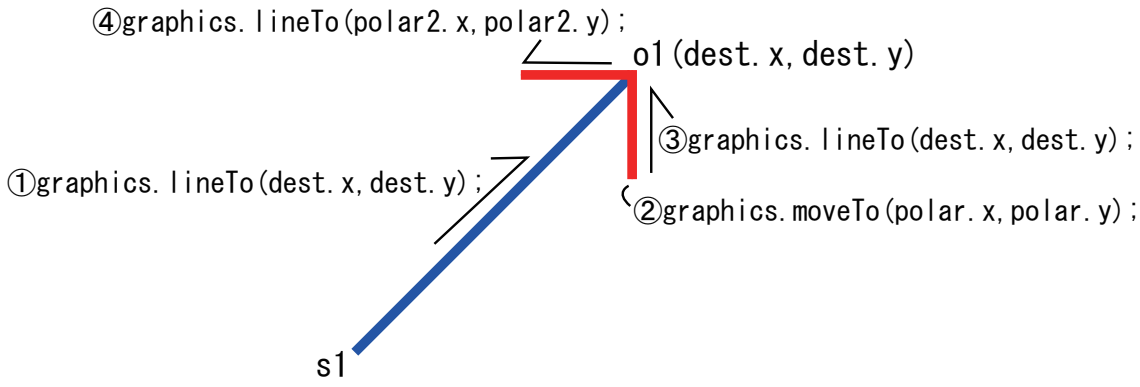

図 4.2: 矢印作成過程

4.10 色設定

矢印の色を Covert Channel の場合の read と write，Covert Channel ではない場合の read と write の 4 色に分ける．Covert Channel の場合の read は赤，write は青，Covert Channel ではない場合の read は薄い青，write は薄い赤とした．これにより，明瞭な視覚化が可能となる．以下にその 4 色に分けるためのコードを示す．

```
//label 要素が write の場合に設定する unit 型変数を定義
var Wcolor:uint;

//label 要素が read の場合に設定する unit 型変数を定義
var Rcolor:uint;

//4.8 am.xml と cc.xml の比較判別にて仕分けする際に Covert Channel で無ければ引数に文字列 am，Covert Channel であれば文字列 cc を設定したが，その引数を用いる．

//Covert Channel でない場合
if(mode == "am") {
//薄い赤
Wcolor = 0xFF7F7F;
//薄い青
Rcolor = 0x7F7FFF;
}

//Covert Channel の場合
else if(mode == "cc") {
```

```
//赤
Wcolor = 0xFF0000;
//青
Rcolor = 0x0000FF;
 }

//Covert Channel かつ，label 要素が read の場合
if(label == "read" && mode == "cc") {

//lineStyle(太さ , カラー)…単色の線の定義
graphics.lineStyle(2,Rcolor);

//beginFill(カラー)…単色の塗りを開始．これにより，図 4.2 矢印作成過程で
示した矢印が塗り潰され，三角となる．
graphics.beginFill(Rcolor);
}

//Covert Channel では無く，label 要素が read の場合
else if(label == "read" && mode == "am") {
graphics.lineStyle(2,Rcolor);
graphics.beginFill(Rcolor);
}

////Covert Channel かつ，label 要素が write の場合
else if(label == "write" && mode == "cc") {
graphics.lineStyle(2,Wcolor);
graphics.beginFill(Wcolor);
}
//Covert Channel では無く，label 要素が write の場合
else if(label == "write" && mode == "am") {
graphics.lineStyle(2,Wcolor);
graphics.beginFill(Wcolor);
}

//4.9.2 矢印描画で示したコード．便宜上，先に説明を行った．
if(type == "out") {
graphics.lineTo(dest.x,dest.y);
graphics.moveTo(polar.x,polar.y);
```

```
graphics.lineTo(dest.x,dest.y);
graphics.lineTo(polar2.x,polar2.y);
}
```

//endFill()…単色の塗りを終了．biginFill と対となり，ここまでに設定した色で矢印が表示される．

```
graphics.endFill();
```

4.11 視覚化プログラム完成

これまで 4 プログラム制作で記述した方法および，プログラムにより，3 Covert Channel の視覚化ついての提案で示した提案を満たし，視覚化プログラムを完成させることができた．以下に Flash Player9 で本プログラムを実行したプログラム動作図を示す．

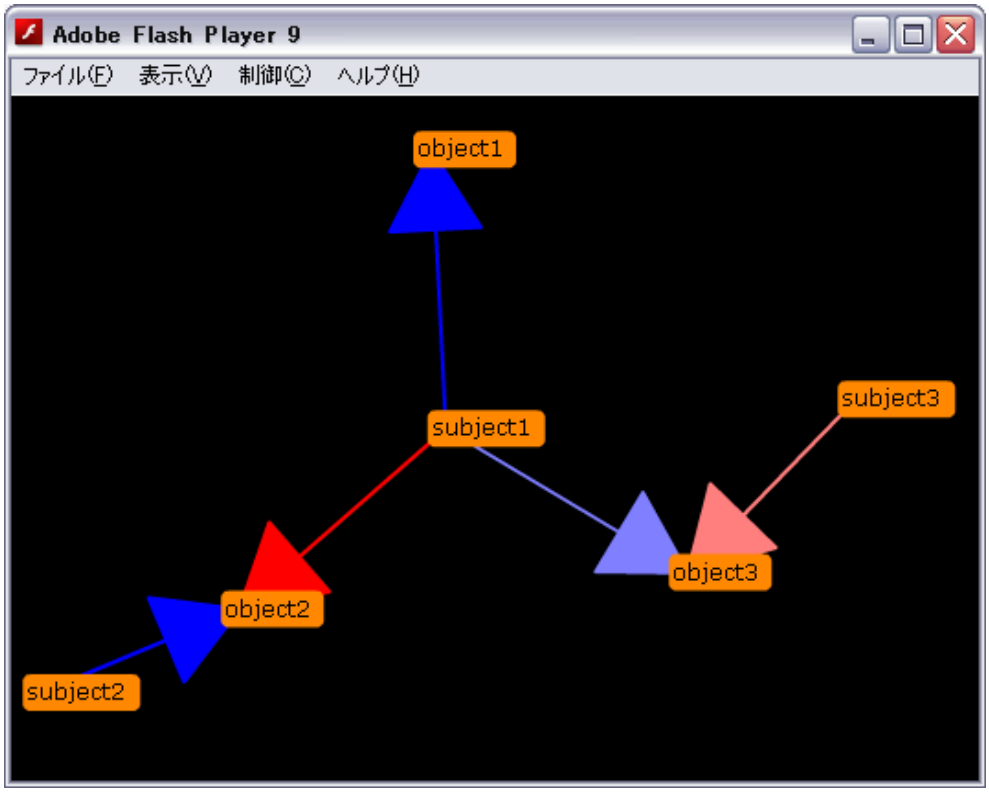

図 4.3: 視覚化プログラム実行画面

図 4.3 視覚化プログラム実行画面では subject1 が青い矢印で object1 を read し，赤い矢印で object2 に write している．そして，その object2 を subject2 が read している．これは，図 2.1: Covert Channel(間接情報フロー) で示したアクセス行列

並びに，3.1 XML の利用と形式定義で示した Covert Channel が存在するアクセス行列を定義した XML による記述をプログラム実行により Covert Channel 色（濃い赤と青）で描画され色分けされている．よって，subject1 から object3 に引かれた薄い青の矢印（read），subject3 から object3 に引かれた薄い赤の矢印（write）は正当なるアクセス権限に基づいた，Covert Channel では無い部分であるということを示している．

4.12 am.xml データ自動生成プログラムの作成

本研究は Covert Channel 分析により Covert Channel を検出し，そのデータを出力するプログラムがあるものと仮定して行った．よって，実装は不可能であるため，制作した視覚化プログラムで大量のノード生成，接続を行った場合を想定し，実行する場合，am.xml に XML 形式定義を手動で入力するには限界があるため，XML 形式定義に基づきアクセス権限を自動で任意の数だけランダムに生成するプログラムを作成した．なお，Covert Channel の自動生成については，そのプログラムの構想に長時間を費やすことが予想されたため，行わなかったが，am.xml データのみを自動生成するプログラムであっても，大量のノード生成，接続を行った場合，制作した視覚化プログラムがどの程度の描画処理に耐えうるかという実現性を測る目的は達成することができた．また，単純な大量描画処理実験を目的としているため，作成時間を短縮するために本プログラムは視覚化プログラムには組み込まず，am.xml としては出力せず，データのみを出力し，出力したデータを手動にて am.xml に保存するものとした．以下に本プログラムの一部とプログラム動作図，出力されたデータ使用した視覚化プログラム図を示す．

```
//設定した数値の回数だけ node 要素（Subject と Object）を出力
for ( iNum = 0; iNum < nodeNum; iNum++ ) {

//Subject を subject1,subject2…と出力
subject = "subject" + String(iNum+1);
name = "<node><name>" + subject + "</name></node>" ;
xml.appendChild(name);

//Object を object1,object2…と出力
object = "object" + String(iNum+1);
name = "<node><name>" + object + "</name></node>" ;
xml.appendChild(name);

//label 要素をランダムに決定
```

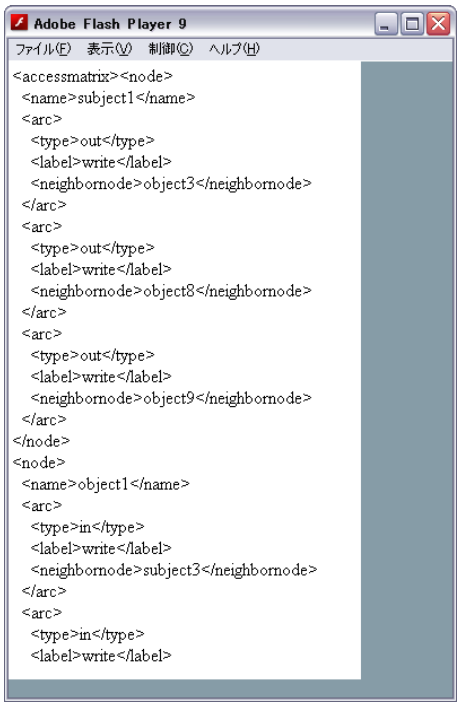

図 4.4: am.xml データ自動生成プログラム実行画面

```
random = Math.floor(Math.random() * 10) + 1;
if ( random \% 2 == 0 ) {
label = "read";
}
else {
label = "write";
}
```

//任意の値の回数だけ Subject と Object の node 要素に arc 要素を出力（以下の例では 3 回）

```
for ( jNum = 0; jNum < 3; jNum++ ) {
stNum = Math.floor(Math.random() * nodeNum) + 1;

//Subject の場合の設定
Stype = "out";
neighbornode = "object" + String(stNum);
arc = "<arc><type>" + Stype + "</type>" +
    "<label>" + label + "</label>" +
    "<neighbornode>" + neighbornode + "</neighbornode></arc>";
```

```
xml.node[iNum+syusei].appendChild(arc);

//Object の場合の設定
Otype = "in";
neighbornode = "subject" + String(stNum);
arc = "<arc><type>" + Otype + "</type>" +
    "<label>" + label + "</label>" +
    "<neighbornode>" + neighbornode + "</neighbornode></arc>";
xml.node[iNum+1+syusei].appendChild(arc);
}
syusei += 1;
```

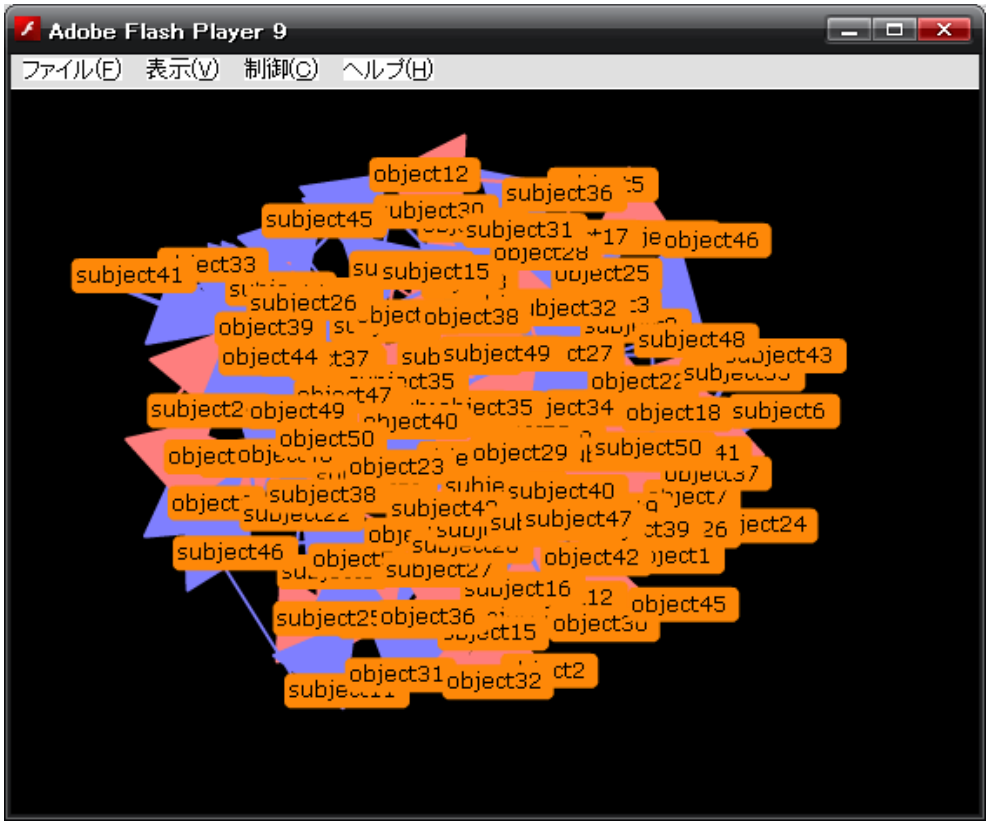

図 4.5: 出力データを使用した視覚化プログラム実行画面

4.12.1 大量描画処理実験の結果

実験により，生成するノード数，および接続数が増大することにより，処理速度が低下し，一般的な PC による運用は耐え難く，また，視覚的にも大量のノードを

モニタの一画面に表示させた場合，3 Covert Channel の視覚化ついての提案に記述した目的の一つである，情報漏洩，改竄の発見や防止，解析等を早急かつ容易にさせることができるということは困難となった.

5 考察と展開

5.0.2 大量描画処理実験の考察

4.12.1 大量描画処理実験結果より，現状のプログラムでは実際の運用には耐え難いことが判明した．これを改善する対策としては，以下のことが考えられる．

- 本研究に用いた ActionScript ライブラリには論文 [17] に基づいたノード間の反発処理プログラムが組み込まれているが，本研究においては必要以上の処理であり，この処理プログラムを削除あるいは改善することにより，大量描画処理時における処理速度低下を改善できるのではないかと考える．
- プログラム実行画面を地図と考えれば，現状ではまったく縮尺がなされていない地図同然であり，縮尺概念を設けることで am.xml に出力された node 要素数に応じて，単純化を行い，大量描画処理を必要とすること無く，また，視覚的な効率を損なわずに済むのではないかと考える．構想する縮尺概念として具体的な例を挙げると，Google マップや Yahoo!地図などがある．

5.0.3 操作性についての考察

現状ではノードの移動操作をすることはできるが，それ以外は一度プログラムを実行すると変更することができない．よって，今後検討される展開 (拡張) として以下のことが考えられる．

- よって実用においては，am.xml，cc.xml に変更があった場合，再描画操作を行う機能.
- 本プログラムから直接，アクセス権限を変更する操作できる機能.
- 本プログラムではノード接続に対する矢印のサイズや色を一意的に設定したが，設定を変更できる機能，これにより各使用者に応じたプログラムの使用しやすい設定が実現できる．
- ノードの操作，移動範囲を実行画面領域に限定せず，ノードの操作，移動範囲を画面外に拡大することで，より，大量描画処理時に視覚的に理解しやすくなるのではないかと考える．

5.0.4 am.xml と cc.xml の比較判別についての考察

4.8 am.xml と cc.xml の比較判別に記述した比較判別法には 1 つの矛盾が存在する．現状では 1 つの node 要素に対して，同じ neighbornode 要素を持つ arc 要素は存在しないものとして判別を行っている．label 要素が read あるいは write のみで

あり，write = READWRITE 権限とみなし，WRITE 権限のみのものが存在する場合を想定していなかったため，WRITE 権限のみというものは存在しないと言うことになっている．実際，ある Subject あるいは，Object に対し，WRITE 権限が存在し，READ 権限は存在しないという状況は局所的にしか存在しないが，存在自体は否定できない．よって，もし，そのような状況が存在した場合，現状では，最後に読み込んだ同 node 要素における arc 要素のアクセス権限が有効となってしまい，ノード接続矢印に対して誤った色設定を行ってしまう可能性がある．この問題は局所的ながらも改善は不可欠なものである．よって，4.8 am.xml と cc.xml の比較判別に記述したコードは改善が必要であると考える．

5.0.5 ActionScript 使用に関する考察

本研究では 4.4 XML の読み込みで記述した経緯から Flex を利用せず，ActionScript のみでのプログラム制作を行ったが，今後の展開，拡張を考慮すると Flex を利用したプログラム制作の方がデザイン面，操作性などにおいて望ましい可能性がある．

6 結論

本研究はCovert Channelの視覚化を行うということで開始した．次に実行プログラムにはFlash，実データの出力にはXMLを用いるなど仕様を決定した．そして，ActionScriptライブラリを用い，本研究用にプログラミングを行った結果，Covert Channelを視覚化するという当初の目標には達成することができた．また，視覚化においても，直感的に理解が可能となるよう，矢印を用い，色分けを行うことで，より明確なCovert Channelの視覚化を行うことができたと言えるだろう．しかし，5 考察と展開で記述したように，Covert Channelの視覚化において，大量描画処理，操作性，また，局所的問題など，まだ発展，展開を行うことができる要素は多々あり，今後より理想的なCovert Channelの視覚化を目指す上では避けては通れない道である．Flashにおける実現性については充分可能であるため，今後の展開，展望には大きな可能性を秘めている．

謝辞

本研究を行うにあたり，終止熱心にご指導して頂いた木下宏揚教授と鈴木一弘特別助教，ご多忙の折研究室に足を運び様々な面で有益なご助言をして頂いた森住哲也氏に深く感謝いたします．さらに，公私にわたり良き研究生活送らせて頂いた木下研究室の方々に感謝いたします.

参考文献

[1] 森住 哲也, 木下 宏揚:“ 社会システムの中の Covert Channel について ”, 技術と社会・倫理研究会, (2005).

[2] 森住 哲也, 木下 宏揚:“ インターネット社会の情報漏えいを防止するセキュリティモデルの提案（社会システム論と記号論からの着想）”, 情報セキュリティ学際シンポジウム, 2005.11, (2005).

[3] 小松 充史:“ Covert Channel 分析制御のための推論を導入した情報フィルタに関する研究 ”, 2006 年度神奈川大学修士論文.

[4] 森住 哲也：”直観主義論理の意味論に基づく統合セキュリティモデル ”, 2007 年度情報セキュリティ大学院大学博士論文.

[5] Colin Moock (著), 永井 勝則 (翻訳) ：”詳説 ActionScript 3.0 ”, オライリージャパン.

[6] 山田 祥寛：”10 日でおぼえる XML 入門教室 第 2 版 ”, 翔泳社.

[7] ”ActionScript 3.0 言語およびコンポーネントリファレンス ”, http://help.adobe.com/ja_JP/AS3LCR/Flash_10.0/

[8] Charles E. Brown (著), 永井 勝則 (監修) ：”Flex 3 ビギナーズガイド ”, 翔泳社.

[9] ”ActionScript 3.0 コンポーネントリファレンスガイド ”, http://livedocs.adobe.com/flash/9.0_jp/ActionScriptLangRefV3/

[10] トップスタジオ：”ActionScript3.0 ビジュアル・リファレンス ”, トップスタジオ.

[11] ”Adobe Flex 3.2 リファレンスガイド ”, http://livedocs.adobe.com/flex/3_jp/langref/

[12] ”Adobe ActionScript 3.0 * Adobe Flash 用 Adobe ActionScript ”, http://help.adobe.com/ja_JP/ActionScript/3.0_ProgrammingAS3/

[13] ”Graph.as – force-directed graph layout by ActionScript ”, http://sawamuland.com/flash/graph.html

[14] ”Vista 普及率 23%、Silverlight は 24%、Flash10 は 58% - J ストリーム調査 ”, http://www.sem-r.com/09/20090510174808.html

[15] ”Silverlight 普及率 24 %は高い？ 低い？”, http://www.atmarkit.co.jp/news/200905/07/ria.html

[16] 西田 学:“ Covert Channel 分析と情報フィルタ選択のための推論機能の提案 ”, 2007 年度神奈川大学卒業論文.

[17] THOMAS M. J. FRUCHTERMAN, EDWARD M. REINGOLD：”Graph Drawing by Force-directed Placement ”, SOFTWARE PRACTICE AND EXPERIENCE, VOL. 21(1 1), 1129-1164 (NOVEMBER 1991).

質疑応答

Q： どのような使用者を想定しているのか？（能登准教授）

A： 現状では SNS の管理者です.

Q： 一般ユーザーは想定していないのか？（能登准教授）

A： 展開としては表示する情報を制限することで一般ユーザーも使用することは可能だと考えます.

Q： Covert Channel 自体は検出はどうやっているのか？（松澤教授）

A： Covert Channel の検出は行っていません．Covert Channel を検出するプログラムが存在すると仮定して本プログラムを制作しました.

Q： Covert Channel の判別は具体的にはどのようにやっているのか？（松澤教授）

A： アクセス権限が出力される am.xml と Covert Channel 情報が出力される cc.xml の 2 つの XML ファイルを用意し，両ファイルを比較判別しています.