モデル指向形式仕様記述における ハザード解析法STAMP/STPAの活用

畑　彰拓[1,a)]　日下部　茂[1,b)]　林　信宏[1]　大森　洋一[1]　荒木　啓二郎[1]

概要：
ソフトウェアシステムの開発において形式手法は効果的に品質を高めるのに有効とされるが，形式手法を用いたとしても，どのようなシステム要求や制約事項を記述すべきかを考え，またその妥当性の判断を正しく下すことは必ずしも容易でない．ソフトウェア中心のシステム開発が様々な機能の実現を可能にする一方で，従来手法での安全性解析を困難にしている問題に対し，システム理論 STAMP とその理論に基づく解析法 STPA は提唱されているものである．我々はこのような問題解決法の一つとして STAMP/STPA の活用を検討している．本稿では，クラウドサービスの機能についてモデル指向の形式的仕様記述言語 VDM++の陰仕様記述を行うことを事例に，我々が提案する活用法を説明する．

キーワード：モデル指向形式仕様記述; STAMP/STPA; VDM++; ソフトウェア開発;

1. はじめに

ソフトウェアシステムは，年々大規模化や複雑化が進むとともに社会的に果たす役割も大きくなり，システム障害の社会への影響も大きくなっている．そのため機能的な正しさだけでなく，安全性や信頼性といった，システムの品質を向上させることが益々重要になっている．ソフトウェア開発において起こりうる障害の原因の多くが上流工程にあると言われている．開発の上流工程において仕様を厳密に決定せず，曖昧な仕様のまま開発の下流工程へ進むことが品質の欠損につながっている．

形式手法は品質向上のためのソフトウェア開発手法のひとつで，論理学や離散数学などの数学的な理論を基礎としている．本稿では，システム開発の上流工程での要件の記述を重視した，モデル指向の形式手法を用いるアプローチを考える．このアプローチでは上流工程で記述すべき要件の分析が重要であるが，どのようなシステム要求や制約事項を記述すべきかを考え，またその妥当性の判断を正しく下すことは必ずしも容易でない．本稿ではクリティカルな要件を持つシステムの品質向上を目的として，モデル指向の形式仕様記述を用いた開発で，システム理論 STAMP（System-Theoretic Accident Model and Processes）に基づくハザード解析技法 STPA(System-Theoretic Process Analysis)[2][3] を活用する手法を提案する．

ソフトウェア比重の高いシステムはシステムの状態，各コンポーネントの内部状態や授受されるデータなどによって，その振る舞いや相互作用が変遷する．ソフトウェアコンポーネントが仕様通りの振る舞いをしたとしてもシステムレベルでは問題が生じてしまうことがあるため，コンポーネント単体に焦点を当てた手法によるハザードの解析が困難になってきている．システム理論 STAMP はこの点に着目し，コンポーネント単体の故障や欠損だけでなく，コンポーネント間の相互作用やそれに起因するシステムの振る舞いと変遷によってもハザードが発生すると考える．STPA は STAMP に基づく解析技法で，システム内で実行されるコントロールの発行プロセスと対象プロセス間で構成される，フィードバックのループでの相互作用に焦点を当てることで，動的側面に着目したハザード解析を実現する．本稿では，クリティカルな要件を持つシステムとしてクラウドシステムを取り上げる．モデル指向の形式手法 VDM[4] とその仕様記述言語 VDM(Vienna Development Method)++を用いてクラウドサービスの機能を記述する際に陰仕様レベルで，STAMP/STPA 解析を行い，提案手法の有用性を評価する．

[1] 九州大学
Kyushu University
[a)] kogechaitai@gmail.com
[b)] kusakabe@ait.kyushu-u.ac.jp

2. VDM++記述と STAMP/STPA 解析の活用

提案する手法では VDM++を用いた仕様記述の際に，陰仕様レベルで，STAMP/STPA 解析を行うことにより，

- システムの妥当性 (validation) を満足する，厳密な仕様とすること
- システムに起こりうる障害を厳密な仕様として反映させること

以上 2 点を目標とする．

2.1 VDM++記述概要

2.1.1 要件仕様記述

開発対象となるシステムには求められる機能や環境的制約といった，要件事項が存在する．仕様記述言語を用いることでこのようなシステムの要件の記述を行うことができる．VDM++を用いる場合，クラスを定義し，各クラスに関して，型，インスタンス変数，値，及び機能名の定義を行う．

2.1.2 陰仕様記述

開発対象となるシステムの機能が正確にその要件を満たす為に，各機能の制約条件を記述することができる．具体的な実現アルゴリズムに立ち入らずに事前・事後条件，不変条件を設定することができ，仕様の段階で機能の信頼性を向上させることが可能となる．

2.2 STAMP/STPA

2.2.1 システム理論 STAMP

システム理論 STAMP ではアクシデントとハザードを以下のように定義する．

アクシデント： 損失となる望まれないもしくは予期しないイベントのこと．この損失は人命や人の負傷，資産の損害，環境汚染，任務の失敗なども含む．

ハザード： ある最悪な環境条件の集合の下で，アクシデントにつながるシステムの状態や条件の集合のこと．

システム全体を通して適切な安全制約を設定し，その安全制約が守られないとき，もしくは不適切なときに「アクシデント」が発生すると考える．その安全制約を守らない状態や条件のことをここでは「ハザード」としている．

- アクシデントを故障の結果ではなく，動的なシステム全体のコントロール，要素間の相互作用の問題として考える．
- システムを技術的な観点からだけでなく，社会的要因による影響も含めた socio-technical な全体システムとして考える．
- ヒューマンエラーを，単なる誤操作ではなく，全体システムと運用者の相互作用によって発生する問題としてとらえる．

STAMP は，このようなシステム理論にもとづいて，システムを安全にするための制約が機能していることを確認し，システム全体としての振舞を安全にコントロールするためのモデルである．

2.3 ハザード解析法 STPA

STPA は，STAMP に基づき，システム内で実行されるコントロールである CA(Control Action：コントロールアクション) とその CA の結果のループで構成される相互作用に焦点を当てることで，システムの動的側面に着目したハザード解析を実現する．その際，システムの制御構造及び，CA のループを明確に図式化するコントロールストラクチャ図を解析実行の準備段階として要求する．STPA の手順はステップ 1 として，非安全な CA である UCA(Unsafe Control Action) の分析，ステップ 2 として，UCA から想定されるハザードの原因要素の分析が行われる．

UCA の分析： ハザード状態につながり得るシステムの不適切な CA を分析する為の補助ツールとして，コントロールテーブルを用意する．コントロールテーブルには以下の四つのハザードとなりうるパターンが存在し，これを表形式で各 CA に適用しハザードに至るかを分析する．

(1) 損失を回避するために必要な CA が与えられないかフォローされない．(Not providing causes hazard)
(2) 不適切な CA が与えられる． (Providing causes hazard)
(3) 潜在的に安全な CA が，正しくないタイミングで与えられる (早すぎる，遅すぎる，違う順番など)．(Too early/late wrong order causes hazard)
(4) 安全性のために必要な CA が止まるのが早すぎる，または適用が長すぎる．(Stoping too soon/applying too long causes hazard)

四つのパターンに当てはめた結果，ハザードに繋がると結論付けられたコントロールのパターンを UCA として識別する．

UCA から想定されるハザードの原因要素分析： 上記で識別された，潜在的にハザードにつながる CA がどのような原因シナリオで発生するかを分析する．具体的なシナリオを考えることで，その原因や生成すべき安全制約も具体的に分析を行うことが出来る．

2.4 提案手法の手順

提案手法は次の手順で行う．

(1) STAMP に基づいたシステム分析を行う．
  (a) システム内及びシステムを取り巻く種々の環境間

で実行される CA とその CA の結果のループを図式化したコントロールストラクチャ図を作成する．
(b) 分析対象に関して適当な抽象度でコントロールストラクチャ図をトップダウン式に詳細化する．

(2) VDM++を用いて要件仕様記述を行う．
(3) STPA 解析を次の手順で行う．
(a) 障害事例等を参考に，システムに起こりうるアクシデントを推測し，その全てに対してどのようなハザードが存在するか考察する．
(b) アクシデントごとに関連する CA を抜粋して，コントロールテーブルを用いて，UCA の分析を行う．
(c) 分析した UCA から想定されるハザードになりうるシナリオを具体的に考察し，そのシナリオを防止するための安全制約を分析する．
(4) 得られた安全制約を基に VDM++を用いて陰仕様記述を行う．

3. クラウドサービスと障害

クラウドコンピューティングによるサービスが一般的なものとなり，クラウドサービスの利用は増え続けるものと考えられる．しかしながら，障害の発生により，サービスが一時的に利用できなくなったり，データを失ったりといった事例がこれまでにも起こっている．利用形態によっては，クラウドシステムは一種のミッションクリティカルと考えることもでき，障害の発生により致命的な損害が生じかねない．こういった観点から，クラウドサービスの可用性や信頼性を高める取り組みが行われており，クラウドシステム運用の脆弱性に対する形式手法適用の研究も行われている [5].

本稿では，高い可用性や信頼性が求められるシステムとしてクラウドサービスシステムを対象に，設計や運用での観点を用いた STAMP/STPA とのモデル指向の形式仕様記述の系統的な活用について議論する．

4. 提案手法のクラウドサービスへの適用

4.1 適用対象：HA クラスタ

ここでは，クラウドサービスに提案手法である「STAMP/STPA を用いた VDM++によるモデル指向の形式仕様記述」を適用することを試みる．適用事例として，クライアントとサーバを分離したソフトウェアモデルである，クライアントサーバモデルにおける HA(High Availability:高可用性) クラスタを用いる．HA クラスタはシステムの可用性の向上を目的とした冗長構成である．クラウドサービスのプロバイダと仮想マシンを使った，Web サーバ層，アプリケーションサーバ層，データベースサーバ層からなる三層システムを構築するための契約をしている状況を想定し，データベースサーバ層においてはプライマリの仮想マシン DB1 とバックアップの仮想マシン DB2 の 2 台のサーバで HA クラスタを構成している．本稿では簡単のためデータベースサーバ層における HA クラスタを用いたシステムに視点を絞って適用を試み，クリティカルな要件として，「データの一貫性の喪失」に着目して議論を進める．また，適用手法を用いることにより，開発の上流工程において障害事例を厳密な仕様書として反映させることを目指す．その為，システム外の要素を含めたシステム全体を解析可能な STAMP/STPA に対して，関連のある，アプリケーションサーバ層からデータベースサーバ層にかけてのシステム内部にまで抽象度を下げて解析を行う．

まずシステムの基本的な処理を次に示す．

- アプリケーションサーバ層からデータベース操作のリクエストを受信する．
- リクエスト内容に応じたデータベース操作を行う．
- データベース操作の結果をアプリケーションサーバ層に返す．

以上の処理に追加して，HA クラスタを実現するための処理を考える必要がある．HA クラスタとは，複数台のサーバを用いたシステムの冗長構成であり，primary（稼働系）サーバに障害が生じた場合に，buckup（待機系）として配置していたサーバに処理が引き継がれる．結果，システムダウンを最小限に抑えられる．この primary サーバから buckup サーバへの処理の引き継ぎはフェイルオーバーと呼ばれる．HA クラスタは各サーバからアクセス可能な共有ディスクを与えられ，処理の引き継ぎはセッションを共有ディスクに格納し，参照することで行われる．. ただし，セッションがシステム内の状態やデータに干渉するステートフルな内容である場合，セッションは最初から生成をやり直す必要があり，アプリケーションサーバ層にその旨の命令を返す．また，各サーバ間において互いの動作状態を監視し合う，ハートビート通信が行われており，ハートビート通信が決められた時間内に受信できない場合は監視対象サーバに障害が生じたと判断される．その為，フェイルオーバーもこのタイミングで実行される．なお，primary サーバのみに可動性の仮想 IP アドレスを与え，ネットワークアクセスが可能となっている．フェイルオーバー時には

- primary サーバの変更
- 仮想 IP アドレスの移動
- 異常サーバの強制停止
- セッションの引き継ぎ，またはセッションの再生成命令

以上の処理が行われる．

4.2 STAMP に基づくシステム分析（ステップ 1）

STAMP に基づくシステム分析では，コントロールストラクチャ図を作成することによりシステム構造の理解を深め，コンポーネントや CA 同士の相互作用にも解析の焦点が当てられる．本事例では要件仕様記述レベルまで詳細

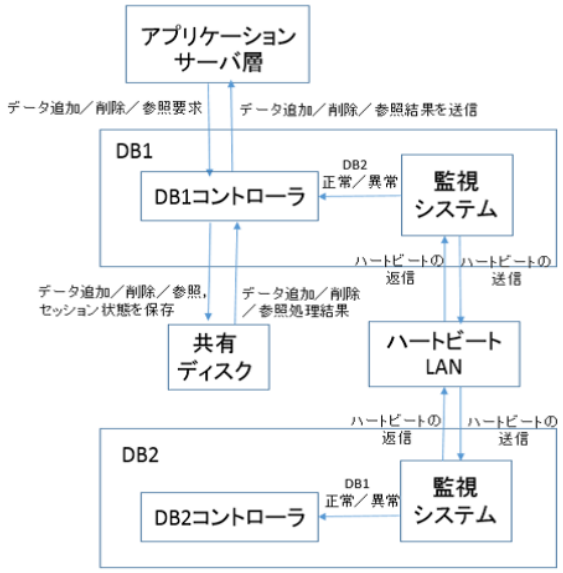

図 1　データベースサーバ HA クラスタのコントロールストラクチャ図

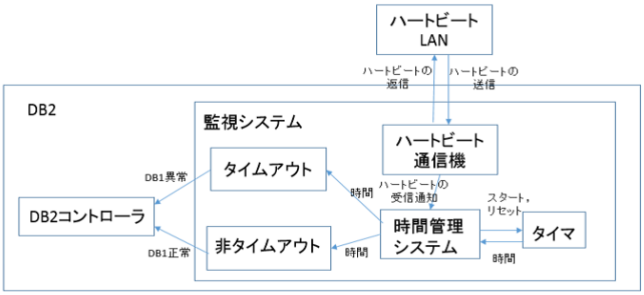

図 2　障害監視システムに関する詳細なコントロールストラクチャ図

化を行ったコントロールストラクチャ図のみ紹介する．初めに，適用事例システムの正常動作時のコントロールストラクチャを図式化した (図 1)．この図 1 は監視システムやフェイルオーバーの具体的な CA が表現されていないので，要件仕様記述を行うには抽象度が不十分である．その為，障害検知を行う監視システムのコントロールストラクチャを図 2，フェイルオーバー時のコントロールストラクチャを図 3 として図式化した．なお，コントロールのループが存在しない箇所は，自明である CA が省略されているものとする（e.g.「DB1 の動作状態を要求」,「時間データを参照」)．

4.3　VDM++要件仕様記述（ステップ 2）

4.2 節の成果物を基に VDM++を用いてシステムの要件仕様記述を行う．クラスは「App サーバ層」,「DB サーバ層」「フェイルオーバー」の 3 つのクラスでシステムを構築する．「App サーバ層」は「DB サーバ層」の，「DB サーバ層」は「フェイルオーバー」のそれぞれ親クラスとした．ここでは要件仕様のうち，次節で紹介する「システムの split-brain 現象」に起因する「データの一貫性の喪失」

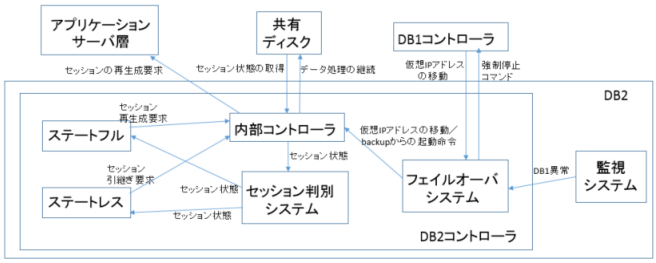

図 3　フェイルオーバーシステムに関する詳細なコントロールストラクチャ図

に関連する機能のみを抽出し，紹介する．

(1)「DB サーバ層」クラス
- データベースを操作する
- タイマを 0 秒からスタートする
- ハートビート通信を行う

(2)「フェイルオーバー」クラス
- プライマリサーバを変更する
- 異常サーバを強制停止する
- 仮想 IP アドレスを移動する
- 中断されたセッションを引き継ぐ
- 中断されたセッションの再生成命令を出す

4.4　STPA 解析（ステップ 3）

表 1　アクシデントとハザードの識別

Accident	Hazard
データの一貫性の喪失	同時に 2 つのサーバ（primary と buckup）が稼働することで split-brain な状態となり，データベース操作の重複が生じる．
	primary サーバの障害発生時に，適切に buckup サーバへのフェイルオーバーが行われない．

まず，本稿で着目しているアクシデント「データの一貫性の喪失」を引き起こす可能性のあるハザードを具体的に認識する．表 1 より,「データの一貫性の喪失」を引き起こすハザードは「システムの split-brain 現象」と「不適切なフェイルオーバー」の 2 つ存在することがわかる．本稿では，簡単のため「システムの split-brain 現象」に起因する「データの一貫性の喪失」についてのみ提案手法の適用を行う．このハザードに関連するコントロールアクションを抜粋する．ここでは前節の VDM++要件仕様記述にて抽出した機能を用いて，各機能に対してコントロールテーブル上の四つのパターンに従って UCA の分析を行い，必要に応じて表を埋めた (表 2)．UCA 分析の結果，split-brain となりうる UCA が 10 個存在した．そこで，split-brain につながりうる具体的なハザードシナリオ（ハザードになり

表 2　非安全なコントロールアクション (UCA) の分析

No.	Control Action	Not providing causes hasard	Providing causes hazard	Too early/too late, wrong order causes hazard	Stopping too soon / applying too long causes hazard
1	異常サーバを強制停止する	(UCA1) 異常サーバの動作継続による，2 台のサーバの同時稼働	(UCA2) 異常サーバ以外のシステムの強制停止	(UCA3) (early) 正常サーバの強制停止	(UCA4) (soon) 強制停止の未完了による，2 台のサーバの同時稼働
				(UCA5) (late) 異常サーバの動作継続による，2 台のサーバの同時稼働	
2	仮想 IP アドレスを移動する	(UCA6) IP の異常サーバ指定による，2 台のサーバの同時稼働	(UCA7) IP の指定間違い	(UCA8) (early) サーバ正常時の IP 移動による，2 台のサーバの同時稼働	
				(UCA9) (late) primary なサーバが長時間存在しない	
3	プライマリサーバを変更する	(UCA10) サーバ異常時の buckup サーバへの未変更		(UCA11) (late) サーバ異常時の遅すぎる buckup への変更	(UCA12) (soon) プライマリサーバ変更途中での停止による，2 台のサーバの同時稼働
4	中断されたセッションを引き継ぐ (再生成命令を出す)	(UCA13) クライアントへの非出力	(UCA14) 再生成すべきセッションの引き継ぎ(セッションの再生成)	(UCA15) (early) 中断前の現行セッションとの重複による，2 台のサーバの同時稼働	(UCA16) (soon) 引き継ぎ（再生成命令）途中の停止による，クライアントへの非出力
5	データベースを操作する	(UCA17) クライアントへの非出力	(UCA18) クライアントへの誤出力		
6	ハートビート通信を行う			(UCA19) (late) 正常サーバを異常と判断されることによる，2 台のサーバの同時稼働	(UCA20) (soon) 正常サーバを異常と判断されることによる，2 台のサーバの同時稼働
7	タイマを 0 秒からスタートする	(UCA21) 異常サーバを正常であると誤判断		(UCA22) (early) 正常サーバを異常であると誤判断，2 台のサーバの同時稼働	
				(UCA23) (late) 異常サーバを正常であると誤判断	

うるシナリオ例）(表 3) を挙げる．具体的なハザードシナリオを用いることで，そのシナリオを防止するための安全制約 (Safety Constraint) を具体的に分析できる．実際に次の安全制約が得られた．

- プライマリサーバの変更に当たって，決して primary サーバが同時に 2 つ存在してはいけない．(from H2)
- フェイルオーバーは必ず「プライマリサーバを変更する」,「仮想 IP アドレスを移動する」,「異常サーバを強制停止する」,「中断されたセッションを引き継ぐ（セッションの再生成命令を出す)」の順に処理を行わなければならない．(from H1〜H6)
- ハートビート通信を行うネットワーク環境も冗長構成にする必要がある．(H5〜H6)

4.5　VDM++陰仕様記述（ステップ 4）

STPA 解析の成果物である安全制約を基に，システムの陰仕様を形成する．4.3 節の VDM++を用いた要件仕様記述に対して以下の制約条件を追加記述する．

表 3　UCA 分析より推測される split-brain を引き起こすシナリオ例

No.	Hazardous Scenario	UCA
H1	障害の一種により，buckup とされた異常サーバがデータベース操作を行う．	1,4,5
H2	primary サーバ変更の際，異常な primary サーバを buckup サーバへ変更することに先んじて，正常な buckup サーバを primary サーバへ変更する．このことにより，一時的に primary サーバが 2 つ存在する．	12
H3	仮想 IP アドレスが誤って異常サーバを指定し，アプリケーションサーバ層からのリクエストを異常サーバが受理する．	6,8
H4	primary サーバにおいて，現行のセッションが中断される前に，buckup サーバがセッションを引き継ぎ（再生成命令を出し），セッションの重複が生じる．	15
H5	ハートビート通信を行う，ネットワークの障害（遅延，瞬断等）により，buckup サーバが primary サーバの異常とみなして primary サーバへ変更し，稼働するサーバが 2 つ存在する．	19,20
H6	タイマをハートビート送信を行うよりも極端に早くスタートすることにより，buckup サーバが primary サーバの異常とみなして primary サーバへ変更し，稼働するサーバが 2 つ存在する．	22

4.6　考察

モデル指向形式仕様記述において STAMP/STPA 解析を陰仕様レベルで行うことにより，アクシデントである障害事例に対し，局所的ではあるが網羅的に VDM++陰仕様記述の制約条件として安全制約を設定することが出来た．すなわち，STAMP/STPA を行うことにより，クライアント要求（本事例ではアプリケーションサーバ層によるデータベース操作要求）を満足する陰仕様記述が行えた．このことより，モデル指向形式仕様記述の問題点である，どのような要求や制約事項を記述すべきかの考慮及び，その妥当性の正しい判断の困難性に対して，STAMP/STPA 解析は有用と思われる．

以上の理由によりクラウドサービスのような複雑なソフトウェアインテンシブシステムにおいても，モデル指向形式仕様記述におけるハザード解析法 STAMP/STPA の活用は有効であると考えられる．

5.　おわりに

クラウドサービスのシステムを対象に，高い可用性や信頼性を求められるシステム開発の上流工程において，モデル指向形式仕様記述と STAMP/STPA を系統的に併用する提案手法を適用した．モデル指向形式仕様記述におけるハザード解析法 STAMP/STPA の活用により，特定の品質特性に焦点を当てた仕様記述が効率良く円滑に行えようになった．さらに，STAMP/STPA 解析によって，仕様の過度な冗長性の防止にもなるという点も挙げられる．この解析から得られる仕様の制約条件はアクシデントを防ぐための安全制約のみであるので，アクシデントとは関係のない冗長な制約条件の記述を防ぐことができる．

引き続き，形式手法と STAMP/STPA の併用について研究を行う予定である．

表 4　安全制約から変換された VDM ++陰仕様指向の制約条件

No.	Constraint Conditions for VDM++
1	・操作「異常サーバを強制停止する」 事前条件 : (プライマリサーバを変更する) and (仮想 IP アドレスを移動する) and (buckup サーバの動作状態 =< 異常 >) 事後条件 : (buckup サーバの稼働態系 =<off>)
2	・操作「プライマリサーバを変更する」: 事前条件 : (primary サーバの動作状態 =< 異常 >) and (buckup サーバの動作状態 =< 正常 >) 事後条件 : (primary サーバの動作状態 =< 正常 >) and (buckup サーバの動作状態 =< 異常 >) 不変条件 : not ((DB1 =<primary>) and (DB2 =<primary>))
3	・関数「仮想 IP アドレスを移動する」 事前条件 : (プライマリサーバを変更する) and (primary サーバの仮想 IP アドレス = nil) and (buckup サーバの仮想 IP アドレス = primary 用仮想 IP アドレス) 事後条件 : (primary サーバの仮想 IP アドレス = primary 用仮想 IP アドレス) and (buckup サーバの仮想 IP アドレス = nil)
4	・関数「中断されたセッションを引き継ぐ」,「中断されたセッションの再生成命令を出す」 事前条件 : (プライマリサーバを変更する) and (仮想 IP アドレスを移動する) and (異常サーバを強制停止する)
5	型定義 :「ハートビート LAN」=<primary> | <buckup> 操作「プライマリサーバを変更する」: primaryLAN と buckupLAN の変更も同時に行う．

参考文献

[1] 荒木啓二郎 : アーキテクチャ指向形式手法に基づく高品質ソフトウェア開発法の提案と実用化，http://hyoka.ofc.kyushu-u.ac.jp/search/details/K000218/index.html
[2] Nancy G. Leveson, Engineering a Safer World -Systems Thinking Applied to Safety-, MIT press, 2012.
[3] MIT Partnership for a Systems Approach to Safety (PSAS), http://psas.scripts.mit.edu/home/
[4] SCSK CORP.: VDM information web site, http://www.vdmtools.jp/modules/tinyd1/index.php?id=1
[5] Shinji Kikuchi, and Toshiaki Aoki, Evaluation of Operational Vulnerability in Cloud Service Management using Model Checking, Proc. of IEEE SOSE, pp.37-48, 2013.