

オープンシステムディペンダビリティ

新しいディペンダビリティへの挑戦

菅谷みどり[†] 山本修一郎^{††} 倉光 君郎[†]

[†] 横浜国立大学

〒240-8501 横浜市保土ヶ谷区常盤台 79-7

^{††} 名古屋大学

〒464-8601 名古屋市千種区不老町

E-mail: †{doly,kimio}@ynu.ac.jp, ††yamamotosui@icts.nagoya-u.ac.jp

あらまし 現在のソフトウェアシステムは、システムのオープン性やフォルト予想の困難さにより、十分にフォルトを除去する前に運用を開始しなくてはならない。オープンシステムディペンダビリティは、そのような状況に対し、障害による被害を減らしながらシステムを改善する概念として提唱されている。本論文では、従来のディペンダビリティの古典的分類学を対比させながら、オープンシステムでは、情報の非対称性から、事前に防ぎにくい障害が生じることを示す。また、オープンシステムディペンダビリティ実現のため、発生した障害をマネジメントする手法として説明責任に着目し、新たにフォルト評価を含めたフォルト対策の必要性を論じる。

キーワード オープンシステム, ディペンダビリティ, 説明責任, 製品のライフサイクル

Open Systems Dependability

A Challenge to a new dependability

Midori SUGAYA[†], Shuichiro YAMAMOTO^{††}, and Kimio KURAMITSU[†]

[†] Graduate School of Electronic and Computer Engineering, Yokohama National University

^{††} Nagoya University

E-mail: †{doly,kimio}@ynu.ac.jp, ††yamamotosui@icts.nagoya-u.ac.jp

Abstract Current system should be worked before the sufficient removal of fault, since it is difficult to forecast the embedded fault and its openness of the system. Open Systems Dependability (OSD) concept has been proposed as a concept to improve the system by gradually reducing the damage caused by faults. In this paper, we clarify the concepts and features of OSD, while contrasting the classical taxonomy of traditional dependability and open systems dependability. In open systems, we will show that it is hard to prevent a fault in advance that is chained to error and failure against the threat model. We also focus on accountability as a method for management failures that will occur, and discuss the necessity for fault evaluation measurements.

Key words Dependability, Open Systems, Accountability, Resilience

1. はじめに

ディペンダビリティ [1] は、信頼性、可用性、安全性など複合的な非機能要求をみたす性質である。これらの性質を備えたシステムをディペンダブルシステムと呼ぶ。伝統的なディペンダブルシステムの構築法では、あらかじめフォルト（障害要因、過失）を列挙し、そのエラー伝播を分析した上で、障害の対策を実施する手法がとられてきた。この方法は、フォルトやエラー

伝播がよくモデル化できる閉鎖系において成功してきた。しかしながら近年では、ネットワークやソフトウェアの複雑化が進んだため、情報システムを閉鎖系として構成することが難しくなっている。とくに、インターネットやオープンソースソフトウェアの利用は、この傾向をより顕著なものとしている。情報システムを構成するサブシステムは、常に更新され変化を続け、障害発生をより予測しにくいものになっている。そのため、閉鎖系を前提としない方法論が必要となっている。

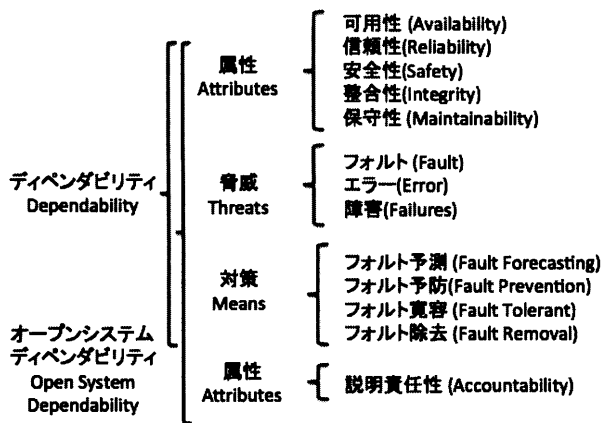


図 1 古典的ディペンダビリティとオープンシステムディペンダビリティの概念と分類学

JST/DEOS プロジェクトでは、ディペンダブル組み込み OS の研究開発に端を発し、単一の組み込みシステムから、より複合的な社会システムのインフラストラクチャを対象として、ディペンダブルシステムの構築を行う基盤技術の研究開発を進めてきた。その中核的なアイデアは、閉鎖系からオープンシステム（開放系）へのパラダイムシフトである。オープンシステムディペンダビリティの概念は、DEOS プロジェクト技術白書[2]に定義されている^(注1)。古典的なディペンダビリティの概念と分類学は、過去 40 年にわたる FTCS (Fault Tolerant Computing System) 研究や実践の集大成として、IEEE Transaction on Dependable and Secure Computing 誌の基調論文としてまとめられている。本論文で提案された分類学は、TDSC (Taxonomy of Dependable and Secure Computing) と呼ばれ、学術から産業界まで広く参照されている。本論文の目標は、TDSC と比較参照しながら、オープンシステムディペンダビリティの再定義を試みることである。

本論文の貢献は次の点にある。

- オープンシステムの概念を導入し、現在のソフトウェアシステムを管理ドメインや開発プロセスによる変化を含めたモデル化を行い、
 - その上で、従来からの障害対策の不完全性から生じる障害に加え、情報の非対称性から生じる予測困難な障害が生じることを示し、
 - TDSC のフォルト対策に対し、フォルト評価と公開を加えることで、システム改善が進むことを明らかにした
- ソフトウェアの複雑化が進んだ今日、ソフトウェアバグが主

(注1) : *Functions, structures, and boundaries of modern software systems change over time. Hence incompleteness and uncertainty may result in failures in the future (factors in open systems failure). Open Systems Dependability is a property of a system such that it has the ability to continuously prevent these problem factors from causing failure, to take appropriate and quick action when failures occur to minimize damage, to safely and continuously provide the services expected by users as much as possible, and to maintain accountability for the system operations and processes.*

要な障害の原因となり、逆にソフトウェアが障害管理の中心的な役割を担う機会が増えている。ソフトウェアの信頼性に関する研究も数多くなされてきたが、主に要求や仕様に対する機能や性能の実現性、検証やテストなどの評価方法を中心とした側面に焦点があてられており、ソフトウェアが動作する環境や要求そのものの変化を中心とした問題領域のあり方を本質的に見直すといったことは十分に行われてきていなかった。オープンシステムは、ソフトウェアに由来する変化を含めた、新しい問題領域の考え方である。より多くのソフトウェア研究者や開発者にとって、本論文が新しいディペンダビリティ問題領域に対する理解の一助となることを願っている。

本論文の構成は次のとおりである。第 2 節では、今日のソフトウェアシステムの観点からオープンシステムのモデル化を行う。第 3 節では、TDCS 脅威モデルとオープンシステムモデルから障害予測の困難さを論じる。第 4 節では、発生した障害をマネジメントする手段として説明責任性を導入し、貢献を論じる。第 5 節では、関連研究を概観する。第 6 節では、本論文を総括する。

2. オープンシステム

オープンシステムは、システムの境界が明確に定義されず、変化していくシステムである。今日のソフトウェアが複雑化した情報システムを考えると、有効なパラダイムとなる。本節では、オープンシステムディペンダビリティの議論の土台として、オープンシステムのモデル化を試みる。

2.1 システムとサブシステム

まず、TDSC にしたがって基本用語の導入を行う。あるサービスをユーザに提供するのがシステムである。システムが、期待どおりのサービス出力 (outcome) を提供できない状況をシステム障害 (system failures) と呼ぶ。一般に、システムは境界があり、その外側を環境 (environment) と呼ぶ。

本論文では、情報システムの構成法に焦点を当てた議論を展開するため、システムは何かしらの開発プロセスを通して開発される人工物とする。人間系は、システムの外側、つまり環境要素と考える。

オープンシステムは、システムの境界が明確に定義されず、変化していくシステムである。このような性質を再現するため、System of Systems の視点を取り入れ、サービスはシステムの集合体から提供されると解釈する。全体からみたとき、その構成要素をサブシステムと呼ぶ。サブシステムは、ハードウェア、ソフトウェア (OS, ミドルウェア, アプリケーション, ライブラリ) が相当する。

2.2 管理ドメインと開発プロセス

近年のシステムが複雑さを増している要因の一つに、独立した異なる管理ドメイン (different administrative domains) が存在し、しかもお互いに利害関係が一致しない状況で運用されている点にある。異なる管理ドメイン問題は、オープンシステムにおいてはサブシステムの緩い分散システムとみなせるため、より顕著に現われる。

管理ドメインは、ある統一的な方針にもとづいて管理を行

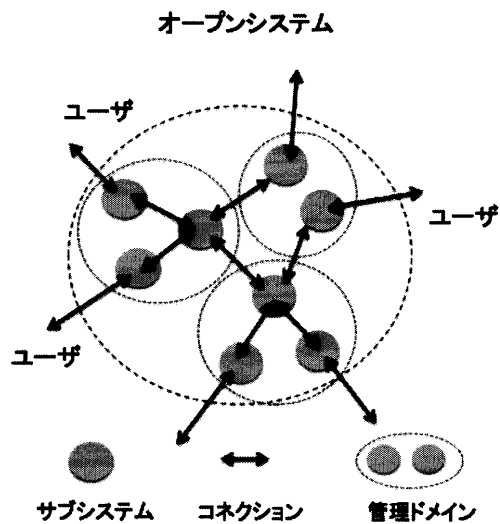


図2 オープンシステム概念図

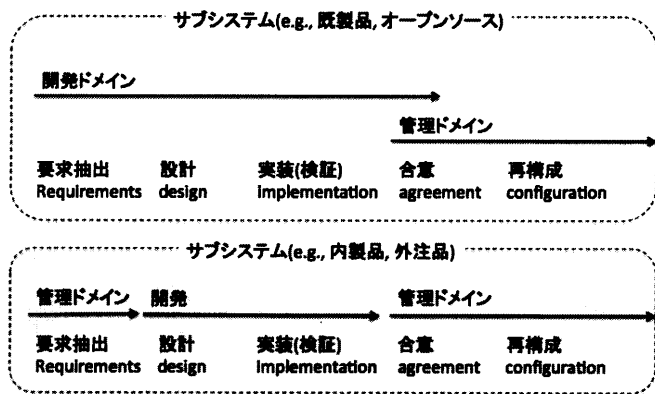


図3 管理ドメインと開発プロセスの関係

うサブシステムの集合と定義する。管理ドメインは、通常、運用保守プロセスが対象となる。我々は、オープンシステムの変化はサブシステムの更新により生じるものとして扱う。サブシステムは、開発プロセスを経て開発される。我々は、サブシステムが管理ドメインの要求により開発を管理されていないとき、管理ドメインと開発プロセスは独立していると呼ぶ。これは、サブシステムとして、COTS (commercial off-the-shelf) 製品、オープンソース製品を採用する場合に相当する。逆に、管理ドメインからの要求にもとづいてサブシステムが開発される場合、一体化されていると呼ぶ。図3は、管理ドメインと開発プロセスの関係を図示したものである。

従来ディペンダブルシステムは、管理ドメインによる開発管理により一体化した開発が行われ、ディペンダビリティ性能の達成や改善が行われてきた。しかしながら、COTS 製品やオープンソース製品の活用が進んだ今日では、管理ドメインと開発プロセスが完全に独立することが増えている。そのため、新しい手続きとして、運用者と管理者の間で合意形成が明示的に必要となる。

2.3 コネクション

コネクションは、システム運用中にサブシステムの入れ替えを可能にするため、サブシステム間の接続（通信や依存性）を

抽象化した概念である。サブシステムは、新しいコネクションが発生したとき、システムに追加されるし、逆にコネクションが消えたとき、システムから消える。オープンシステムにおけるコネクションの特性を以下に述べる。

- 全てのコネクションは動的、システム運用中に接続/切断できると想定する。しかし、コネクションが最初から最後まで固定されているものを静的コネクションとして区別する。

- コネクションは、サブシステム間で状態の共有が発生する。共有される状態によって形式/非形式に分類する。形式コネクションは、通信メッセージなど表現可能な状態である。一方、非形式コネクションは、計算機資源の共有などが該当する。たとえば、同一 OS 上で動作するプログラムは、CPU 資源の共有という形でお互いに影響を与え合って接続されている。形式コネクションは 2 項関係のみ、非形式コネクションは、 $n(> 1)$ 項関係を想定する。

- コネクションは、合意とアドホック（合意なし）の 2 種類に分類される。合意されたコネクションは、あらかじめサブシステム間で共有する状態に対して正しい状態（正しくない状態はエラー）が定義される。コネクションの合意は、サブシステムの実装ベースに対して、図3で示した合意フェーズで行うものとする。

オープンシステムでは、サブシステムは開発プロセスを経て開発されるが、コネクションは管理ドメイン側が運用時に作成できる。我々は、このための手段として、スクリプトなどのシステム再構成技術を想定している。

3. ディペンダビリティ脅威と予測困難さ

今日のソフトウェアシステムは、予測が困難な障害に直面する。本節では、TDSC の脅威モデルをオープンシステムに適用しながら、予想困難さが生じる要因を述べる。

3.1 TDSC 脅威モデル

Laprie らの分類学では、ディペンダビリティへの脅威はフォルト、エラー、障害からなる。これらは、それぞれ因果関係があり、脅威はフォルト、エラー、障害の 3 段階をえて進行するとモデル化されている。^(注2) まず簡単に、3 段階モデルを説明する。フォルトは、障害要因となりうる事象^(注3)、もしくは行動である。エラーは、フォルトによって出現する望ましくないシステムの状態である。エラーは放置すると、システム障害につながる。（障害に至らない場合もある。）障害は、別の障害を引き起こすフォルトとなりえて、障害の連鎖が起こる（図4）。

古典定義では、全ての障害はフォルトが起点となる。障害対策は、フォルトへの対策^(注4)として実施される。

- **フォルト予測 (fault forecasting)** フォルトの発生頻度、被害の大きさを評価する

(注2)：このモデルは、学術界から産業界まで広く受け入れられ、今日のディペンダブルシステム開発の標準となっている。

(注3)：ヒューマンエラーは、慣習的にエラーと呼ばれるが、この分類ではフォルト (human made fault) である。

(注4)：プログラミングレベルでは、エラー処理と呼ばれるが、実際はフォルトがわからないとエラーからの回復は行えないことに注意

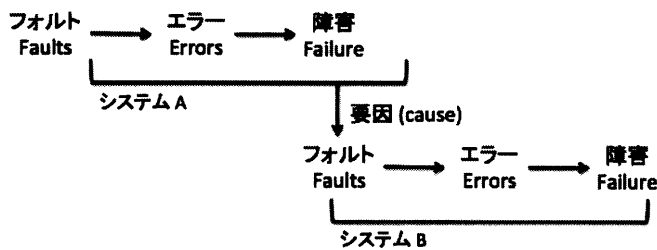


図4 TDSC 脅威伝搬モデル

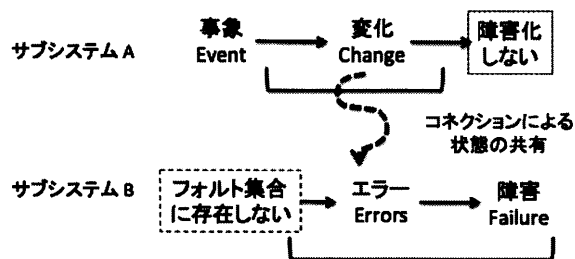


図5 オープンシステム脅威伝搬モデル

- **フォルト予防 (fault prevention)** 事前にフォルトの発生を低下させる工学的な努力. (e.g., 静的型付けプログラミング言語の使用など..)

- **フォルト寛容 (fault tolerant)** 発生したフォルトに対して耐性がある設計と実装を行う

- **フォルト除去 (fault removal)** フォルトの発生を除去する古典的なディペンダビリティでは、これらのフォルト対策を組み合わせることで、受容可能なレベルで信頼性や可用性などのディペンダビリティ属性が達成できるとされてきた。しかしながら、オープンシステムにおいては次の2点の理由からディペンダビリティの達成が困難である。

3.2 不完全性

閉鎖系では、(外部からの影響や変化を限定するため) 有限集合とみなしたフォルトに対して対策がとられてきた。実際、過去、半世紀にわたる FTCS の開発やリスク分析の経験により、フォルトはほとんど識別されている。TDSC 脅威モデルにおいてエラーと障害は、常に同一システム上で起こる。システムは有限状態数と仮定できるため、エラー状態も有限個に抑えられる。しかし、エラー状態が定義されても、そのエラーを引き起こしたフォルトを特定しなければ障害回避や障害回復はできない。そのため、エラーを引き起こしたフォルトを特定する必要がある。このための操作をことをフォルト追跡と呼ぶ。閉鎖系では、フォルト追跡は有限個のフォルトから探すことができる。しかし、自システムの外で発生したフォルトに対して、エラー処理という形で自ら対策を実施できない。こうした、外部へのコネクションが多数存在するオープンシステムにおいては、フォルト追跡、フォルト特定は困難であるといえる。

3.3 情報の非対称性

オープンシステムでは、フォルトは無限集合とみなすことが

できる。概念的には、まったく未知のフォルトが障害をおこすことがありえる。しかし、本論文ではこの見解をストレートに採用しない。なぜなら、未知のフォルトに対して何かしら対策を立てることは不可能であるためである。我々は、サブシステム間のコネクションによって発生する情報の非対称性に注目する。ここで情報の非対称性とは、各サブシステムで保有する情報に偏りがあることをさす。現実のソフトウェアシステムでは、このような情報の非対称性は数多く発生する。仕様の齟齬などもこの範疇に含まれるが、我々は、障害に焦点をあてて情報の非対称性を論じる。

TDSC 脅威伝搬モデルでは、情報の非対称性は問題とされていない(図4)。これに対して、オープンシステム脅威伝搬モデルにおいては、サブシステム A で発生した状態変化は A 上では変化は障害を起こさないため、フォルトと認識されないが、B 上ではコネクションを通じて伝搬した変化が障害を引き起こすエラーと認識される(図5)。ここでは、サブシステム A とサブシステム B の間では想定するフォルト集合が異なり、情報の非対称性が生じている。このような例としては、サブシステム B から呼び出しに偶然応じた、サブシステム A の RPC のレスポンスが遅く、全体の処理がデッドラインミスする、サブシステム A が大量にメモリを消費し、メモリを共有しているサブシステム B に十分なリソース割り当てができなくなるといった例が当てはまる。ここでは、サブシステム B の方では、必ずしも事前にサブシステム A との接続を想定していないこともあり、こうした場合、フォルトを特定することが困難となる。

さらに、サブシステム A とサブシステム B は、お互いに独立しているため、閉鎖系のように事前にコネクションを想定し、フォルト集合を共有した上で、設計、開発、テストすることができない。第2.2で述べたとおり、コネクションに関して合意の手続きが必要となるが、現在の技術実践ではフォルト情報の共有まではあまり行われぬ。こうしたフォルトは、そもそも本質的に予想しにくく、また、フォルト追跡もサブシステム B の予想されたフォルト集合にないため困難である。従来のディペンダブルシステムでは、サブシステムの更新自体が、常に予期できないフォルトとして除去し、オープン性を排除してきた。しかし、今日、オープン性を排除したシステム構築はますます難しくなる。逆に、こうした情報の非対称性をマネジメントする手法が必要となる。

4. 説明責任

現在のソフトウェアシステムは、第3節で述べたとおり、不完全性やオープン性により、フォルトの予測やフォルトの追跡が困難なまま運用しなければならない。そのため、障害発生に備え、発生した障害をマネジメントする必要性が生じる。従来のディペンダビリティ属性(信頼性、可用性、安全性など)では対応力が求められる。オープンシステムディペンダビリティでは、新しい属性として説明責任性(accountability)を重視し、説明責任性を高めることでシステムの改善を促進させる。

4.1 説明責任とは?

説明責任は、不都合な事実や障害を含めた情報開示を積極的

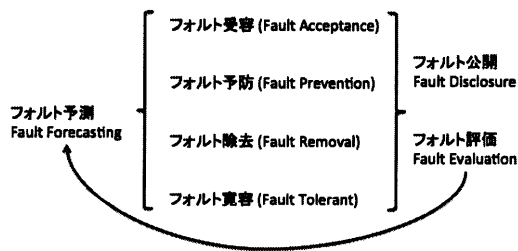


図6 新しいフォルト対策とサイクル運用

に行うことで、逆に情報開示を正しく行わないものを社会的に排除し、最終的に全体としてのシステムの信頼度を向上させる方法として知られている。説明責任性の導入は、主に企業経営や公的機関などの社会システムにおいて実績が認められているが、近年、情報システム構築においても有効な属性とみなされている。^{(注5) (注6)}

オープンシステムにおいては、第3.節で述べたとおり、予測困難な障害に直面するとき、説明責任の遂行は重要なフォルト対策となる。我々は、近年の情報システム分野の研究事例をふまえて、より具体的に以下の行動が説明責任遂行に必要な行動と考えている。

- **コンプライアンス** - (事前に) 安全基準の達成を第三者に確信させること
- **情報開示** - 障害の発生時に、速やかに要因や影響範囲を開示すること
- **改善計画の立案** - 将来への改善計画を合理的に説明できること

オープンシステムディペンダビリティの実現に向けて、これらの活動を支援する技術が必要となる。以下、新たな技術的な挑戦をふまえながら、議論を進める。

4.2 コンプライアンス

今日、多くの産業分野において、過去の事故の経験にもとづいて、安全基準が設定されている。コンプライアンスは、既存の安全基準を満たすことである。とくに、説明責任という観点からは、単に安全基準の達成を自己目標とせず、第三者に確信を持たせることが重要となる。

なお、安全基準や第三者認証という概念は、機械工学や電子工学などハードウェア分野から始まっている。ソフトウェアに関しては比較的新しい試みであるが、独立検証確認 [6] [7] の整備も始まっている。

4.3 フォルト評価と情報開示

障害に対する説明責任は、発生したフォルトの特定が必要である。障害を引き起こしたエラーは、システムの状態であるため、障害中のモニタリングでも観測可能である。しかし、その

(注5)：TDSCにおいても、新しい属性のひとつとして、説明責任性は *Accountability is availability and integrity of the identity of the person who performed an operation.* のように定義されている。

(注6)：ただしこの定義は、accountability (説明責任) と responsibility (対応責任) の分離がやや不明瞭である。これは、著者のひとりである Laprie との議論でもほぼ同一視していたことで確認された。本論文では、対応責任は別の属性としている。

エラーを引き起こしたフォルトは、既に過去の事象となっている。そのため、ログシステムなどに、フォルトが事象として記録されていることが前提となる。フォルトを特定する手法は、フォルト診断 (fault diagnosis, root cause analysis) と呼ばれる。オープンシステムにおいても、原理的にサブシステム間のコネクションに対して全てログに記録すれば、どのサブシステムから事象が発生したか特定できる。現実的には、ログ可能な情報量は限られており、フォルト診断は大きな課題となっている。

説明責任においては、フォルトの特定は最終的に個人^(注7)と結びつける。第2.2節で述べたとおり、全てのサブシステムは管理ドメインに属しているため、管理者の特定は行える。現時点で、管理ドメインをこえて、ログを追跡することは困難であるため、新しくログ追跡の技術のサポートが必要となる。

従来、障害対策は、第3.1節で述べたとおり、フォルト予測、フォルト予防、フォルト寛容、フォルト除去の総合的な対策によって行われる。フォルト発生源に関与した人物だけでなく、フォルト予測、そしてそれに基づいてそれぞれの対策を実施した人物も特定することが重要である。これにより、総合的に障害の発生を分析可能になる。我々は、このステップをフォルト評価 (さらに、評価結果を外部に報告することをフォルト公開) と呼ぶ。

フォルト評価やフォルト公開の結果、フォルト予測やフォルト予防が改められることで、システムの改善が進むと期待される。従来のディペンダビリティ概念では、このようなサイクル的な改善性は欠如していた。

4.4 改善計画の立案

TDSCでは、フォルトの分類として、単純に開発フェーズと運用フェーズのみ分類していた。しかし、障害発生時の改善計画をより詳しく分析するためには、より詳細な分類が必要となる。我々は、図3に示したオープンシステムと開発プロセスをベースに、次のフェーズごとにフォルトの分類を行う。

- (1) 要求フォルト (requirement bugs)
- (2) 設計フォルト (design bugs)
- (3) 実装フォルト (implementation bugs)
- (4) 合意フォルト (agreement bugs)
- (5) 構成フォルト (configuration bugs)

あるフェーズで発生したフォルトを修正すると、それ以降の全てのフェーズの開発の修正が必要となる。明らかに、要求フォルトの修正は、実装フォルトの修正よりコストが高い。要求レベルのフォルトを管理する手法として、合意形成のための手法が提案されている [12]。こうした手法を利用することにより、説明責任を果たしつつ、要求レベルのフォルトの修正と、それに合わせてより下位のフェーズの修正を行う必要がある。

また、管理ドメインと開発プロセスが独立している場合は、異なる管理ドメインで開発されている製品のフォルトを、自管

(注7)：説明責任性は、個人を犯人として特定し、糾弾することではない。単に技術的な追跡性の目標として、個人まで追跡できるかどうかである。その個人を評価するかは、本論文の範囲外である。

理ドメインでの開発プロセスフェーズで修正することは難しい。この場合は、合意フェーズによって、他の製品に切り替えるか、もしくは再構成によってフォルトを回避することになる。2.3節で述べたように、オープンシステムでは、サブシステムは管理プロセスを経て開発されるが、コネクションは管理ドメイン側が運用時に作成できる。このため、フォルトが生じた際には、よりサブシステム間のコネクションの柔軟性を高めることで、管理ドメイン側が運用時に再構成によるフォルトの回避を行うことができる。運用時にこうした再構成を行うためには、柔軟な記述力を持つスクリプト技術 [8] が必須となる。今後は、こうした柔軟な手段で、サブシステムの構成を変化させることで、オープンシステムの変化や障害に対応するための技術が必要である。

5. 関連研究

ディペンダビリティは、伝統的には、信頼性、可用性、安全性、整合性、秘匿性など5つの非機能要求をみだす性質と考えられてきた。以来、コンピュータシステムの複雑化が進むにつれ、さまざまな新しい属性の追加^(注8)が含まれてきた。これらの概念との比較と考察を加えたい。

オープンシステムディペンダビリティと同じく、変化に着目したディペンダビリティの新概念のひとつに復興性 (resilience) がある。復興性は、ネットワークやコンピュータシステムなど分野ごとに解釈が異なるが、一般に変化 [9] に直面したときの機能回復を尺度として考えられている。オープンシステムディペンダビリティは、外部変化への対応だけでなく、サブシステムの更新によって自分自身も変化しながら、サービス改善を行うことを目指している。

同じく、変化に対応する類似の概念として適応性 (adaptability) や生存性 (survivability) などがある。適応性は、オートノミックコンピューティング [10] などで具現化が進み、システムアーキテクチャによって実現される。開発プロセスなどは含まれていない。生存性は、主にセキュリティ脅威など、外部からの攻撃に対する耐性を評価 [11] するものである。

6. むすびに

現在のソフトウェアシステムは、フォルト対策の不完全性やシステムのオープン性により、完全にフォルトを除去する前に運用することになる。オープンシステムディペンダビリティは、そのような状況に対し、障害による被害を減らしながらシステムを改善する概念として提唱されている。本論文では、従来のディペンダビリティの古典的分類学を対比させながら、オープンシステムディペンダビリティの概念と特色を述べた。とくに、情報の非対称性により、新たに「想定外の」障害が発生する可能性を論じた。また、発生した障害をマネジメントする新しい手法として、説明責任遂行を導入し、古典的なフォルト対策に対し、新しくフォルト評価の手法を確立する必要性をえた。JST/DEOS プロジェクトでは、オープンシステムディペンダビリティを実現する具体的な手法の確立を進めている。それら

は、運用情報に基づく要求管理手法、開発プロセスとの双方向な合意形成法、フォルトの追跡が可能なログシステム、システム再構成を高めるディペンダブルなスクリプト技術が含まれる。

謝辞 本研究は、科学技術振興機構 JST/CREST 「実用化を目指したディペンダブル組込みオペレーティングシステム領域」の研究助成の一部として行われてきた。オープンシステムディペンダビリティの概念形成に対し、ご助言を頂いた Karama Kanoun (LAAS-CNRS), Jean-Charles Fabre (LAAS-CNRS) らに感謝します。

文 献

- [1] Avizienis, A., Laprie, J.-C., Randell, B., and Landwehr, C. E.: Basic Concepts and Taxonomy of Dependable and Secure Computing., *IEEE Trans. Dependable Sec. Comput.*, Vol. 1, No. 1(2004), pp. 11–33.
- [2] Mario Tokoro: Dependable Embedded Operating System Project White Paper version 3.0, October 2011. <http://www.dependable-os.net/en/topics/file/WhitePaperV3.0E.pdf>.
- [3] Ralf K sters, Tomasz Truderung, Andreas Vogt: Accountability: definition and relationship to verifiability. In *Proc. of CCS '10: ACM conference on Computer and communications security*. October 2010.
- [4] Andreas Haeberlen, Petr Kouznetsov, Peter Druschel: Peer-Review: practical accountability for distributed systems. In *Proc. of SOSP '07: ACM SIGOPS symposium on Operating systems principles* October 2007
- [5] Lorcan Coyle, et.al, Guest Editors Introduction: Evolving Critical Systems, *IEEE Computer*, vol.43, no.5, pp.28-33, 2010
- [6] 山本修一郎, 独立検証確認と形式手法がもたらすソフトウェア開発プロセスの改革, *IPA SEC Journal*, 2010
- [7] IEEE Std. 1012-2004 Software Verification and Validation
- [8] K. Kuramitsu, “Konohascript: static scripting for practical use,” in *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, ser., Abstract, SPLASH '11. New York, NY, USA: ACM, 2011, pp. 27–28.
- [9] Laprie, J.-C. From Dependability to Resilience, In *Proc. of DSN'98 IEEE International Conference on Dependable Systems and Networks*, Fast Abstract, Anchorage, Alaska, 2008, G8-G9.
- [10] A. G. Ganek, T. A. Corbi, The dawning of the autonomic computing era, *IBM Systems Journal* , Volume 42 Issue 1, 2003 .
- [11] James P. G.Sterbenz, D.Hutchison, E.Ketinkaya, A. Jabbar, J.P. Rohrer, M.Schiller, P.Smith, Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines, *Computer Networks: The International Journal of Computer and Telecommunications Networking* , Volume 54 Issue 8,,2010.
- [12] Y. Matsuno, J. Nakazawa, M.Takeyama, M.Sugaya, Y.Ishikawa, Toward a Language for Communication among Stakeholders, *Proceedings of the 16th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC'10)*, 2010, pp.93-100.

(注8) : taxomomy