DEOS-FY2012-PU-01J

JST-CREST

研究領域

「実用化を目指した組込みシステム用ディペンダブル・オペレーティングシステム」

DEOS プロジェクト

Project Update 2012

2012/11/15

DEOS 研究開発センター編

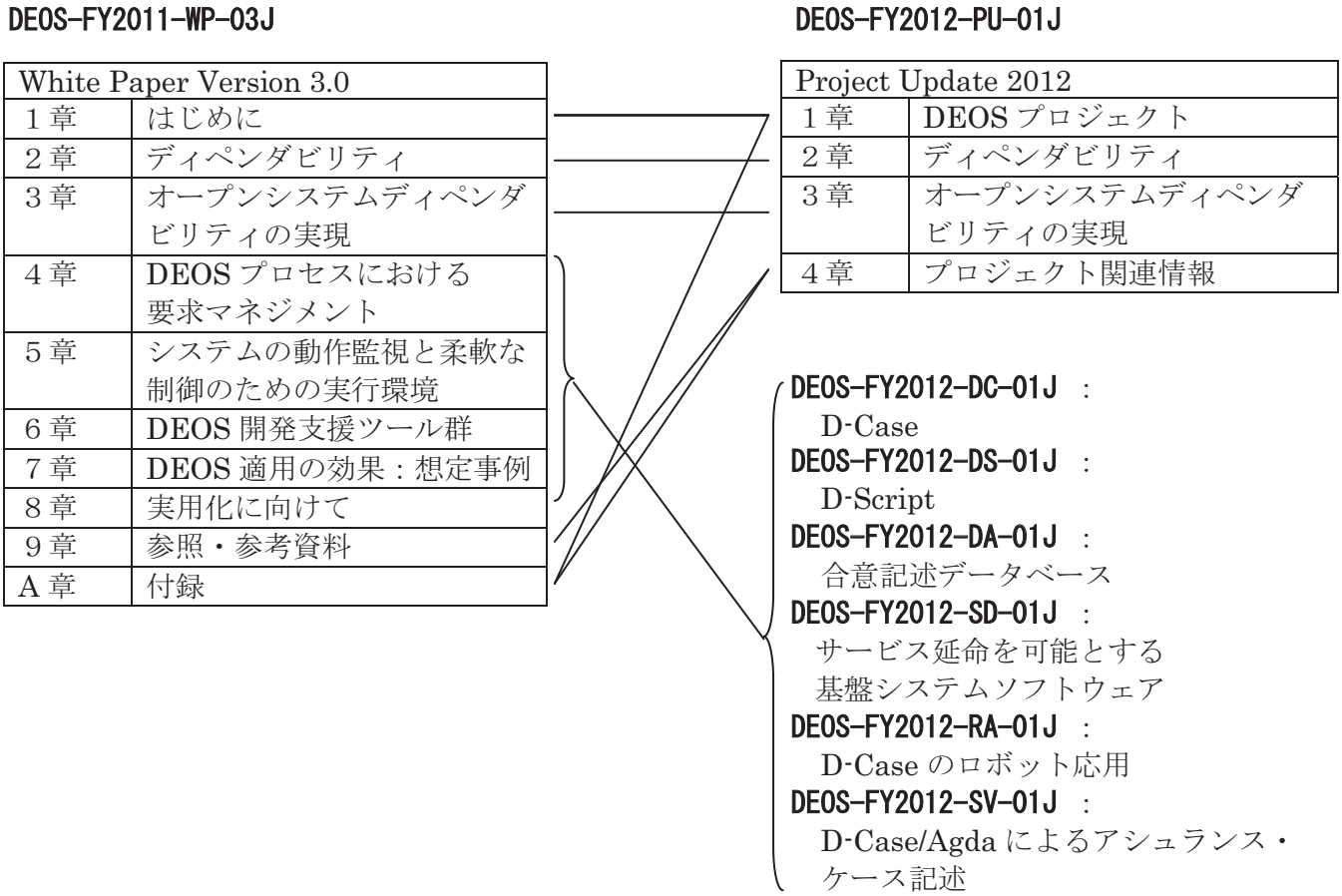
科学技術振興機構
Japan Science and Technology Agency

文書履歴

2012 年 11 月 15 日

2011 年 11 月に DEOS プロジェクト White Paper Version 3.0 を発行して一年が経過した。プロジェクトは残り 1 年半となり、より具体的な技術およびソフトウェアの開発も進んでいる。本 Project Update 2012 では、プロジェクトの目的と概要を述べ、現状を報告する。フォーカスエリアについてはより詳細に解説した技術報告書を個別に発行する。

White Paper Version 3.0 と Project Update 2012 の章建ておよびフォーカスエリア技術報告書の関係（**DEOS-**で始まる番号は DEOS プロジェクト文書番号）：



「実用化を目指した組込みシステム用ディペンダブル・オペレーティングシステム」（DEOS プロジェクト）は科学技術振興機構（JST）の戦略的創造研究推進事業 CREST の研究領域のひとつです。

目次

1. DEOS プロジェクト

1.1. 背景と目標

近年、情報システムは我々の日々の生活に深くかかわるようになってきている。携帯電話、ウェブ上の数多くのサービスはもとより、天気予報、交通制御などの行政サービス、列車や航空機の自動改札システム、座席予約システム、運行管理システムや、物流システム、金融システムなどの業務用システムなど、ありとあらゆる場面で我々は情報システムの計り知れない恩恵を受けている。さらに、それらの情報システム同士が直接、間接に接続されて巨大な情報システム群を形成し、我々の生活の基本的なインフラストラクチャーを構成するに至っている。我々の生活における情報システムへの依存度が大きくなればなるほど、情報システムの信頼性や強靭性がますます重要になってきている。本稿では以下、信頼性や安全性など、システムの提供するサービスを安心して継続的に利用できる性質を総合してディペンダビリティ（Dependability）と呼ぶ事にする。

現代の典型的な情報システムは、情報端末機器がネットワークを介してサーバ群に接続され、情報機器端末が、サーバ上のサービスを利用するように構成されている。これまで、多くの情報システムの開発では、事前に綿密な開発計画を立て、対象製品・システムの仕様を詳細に書き上げ、十分な設計・実装・テストを行った後にユーザによる利用を開始する、と言う方法がとられてきた。この方法は、仕様が開発開始時に十分見極められるような製品やサービスの開発には有効であった。しかしながら上に述べたような大規模システムの開発においては、開発開始時に将来に起こるであろうすべてを見通したシステムの仕様を記述することはほとんど不可能である。加えて、開発においては、自社内の既存ソフトウェアの再利用、他社ソフトウェア製品の利用、ネットワーク上のサービスの利用など、仕様が完全であることを仮定できないソフトウェア部品の使用も行われるようになってきた。

通常このような大規模システムは長期にわたって利用される。したがって、そのようなシステムは時とともに変わる利用者のニーズに対応し、サービスを継続しつつ、システムを変更して行く必要がある。また、技術の進歩や経営上の理由、利用者数の増加・減少などに対し、サービスを継続して提供しつつ、ハードウェアやネットワークの変更に適切に対応しなければならない。このような理由によりシステムのディペンダビリティの確保はますます困難な状況になっている。さらには、ウイルスによるシステム破壊、不正アクセスによる情報漏洩などの脅威に対しても適切に対応し、利用者が安心してサービスを利用できるようにする必要がある。しかしながら、残念なことに、世界のあちらこちらで重要な情報システム障害が発生している。情報システムの障害は個々の利用者に重大な不利益を与えるだけでなく、サービス提供者にとっては本来得られる収益を無にし、時には莫大な補償金の支払いを要求され、企業のブランド価値の毀損につながり、事業の継続が困難になる可能性も出てくる。したがって、サービス提供者には最大限事故を起こさないようにする事に加え、事故が起こった際の説明責任が当初より課されていると考えるべきであろう［1、4、6］。

システム障害の原因を見ると、システム構築の際にすべての構成要素を理解することが不可能であったために発生したもの、利用の量や利用の仕方が当初の想定範囲を超えたことによるもの、利用者の要求変化に応えるためにシステムを変更した際に発生したシステム動作の不整合によるもの、システム運用方法を事前に適切に設計できなかった事などが顕著である。これらの障害原因を見ると、現代のシステムは作られ方の観点でも、使われ方の観点でも、もはやシステムの機能や構造、境界が定義可能な固定的なシステムとして取り扱うことは困難であり、当初より時間の経過とともにそれらが変化しつづけるシステムとして取り扱うことが妥当であると思われる［5］。

本プロジェクトは、変化しつづける目的や環境の中でシステムを適切に対応させ、継続的にユーザが求めるサービスを提供することができるシステムの構築法を開発することを目標としている。そのためには、これまで機能、構造、境界が固定的なシステムを対象に考えられてきたディペンダビリティ技術だけでは不十分であることを示し、新たな概念として「オープンシステムディペンダビリティ」を提案し、これをもとにした開発と運用を一元的に扱う新たなシステム構築法として、DEOS プロセスと呼ばれる反復的プロセス、これを実現するアーキテクチャ、そしてそれらに必要な要素技術群を研究開発している。

1.2. DEOS プロジェクトの経緯、目的、予定される成果

「実用化を目指した組込みシステム用ディペンダブル・オペレーティングシステム」（DEOS プロジェクト）は、JST/CREST の研究領域のひとつとして、2006 年 10 月に開始された。今日の組込みシステムはネットワークを介してサーバ群と接続され、全体としてサービスを提供する形をとっていることから、本プロジェクトではシステムをディペンダブルに構築し運用するためのシステムソフトウェア、これを構築するために必要なツール群、システム構築や運用のプロセスを研究対象とする。

近年の情報システムは、巨大で複雑、かつ変化しつづける目的や環境に対応しなければならず、閉鎖型システム（Closed Systems）として捉えることは難しく、開放型システム（Open Systems）として捉えることが適切である。本プロジェクトでは、オープンシステムに対するディペンダビリティは、反復的なアプローチとして達成すべきものと考え、基本概念として後述する「オープンシステムディペンダビリティ（Open Systems Dependability）」を提案し、プロジェクトの目的を以下のように定義した。

変化しつづける目的や環境の中で、システムを適切に対応させ、ユーザが求めるサービスを継続的に提供することができるディペンダブルなシステムソフトウェアの方法論ならびに構築法を開発すること。 この方法論はオープンシステムディペンダビリティ（OSD）を基本概念とし、ディペンダブルシステムの構築法は、反復的なプロセス（DEOS プロセス）、これを実現するためのアーキテクチャ（DEOS アーキテクチャ）、そしてアーキテクチャを構成する要素技術などからなる。

実際のところ、オープンシステムディペンダビリティ（OSD）の概念や DEOS プロセスはディペンダブルなシステムソフトウェアの構築に応用できるだけでなく、大規模かつ複雑で変化を許容しなければならない多くのシステムをディペンダブルに構築するための方法論になる。すなわち DEOS プロセスは広範なオープンシステムに適用できる。本報告書では「DEOS」を「オープンシステムのためのディペンダビリティ工学（Dependability Engineering for Open Systems）」の略記とする。

本プロジェクトの成果としては以下のものを計画している。

- オープンシステムディペンダビリティ概念（OSD Concept）
  - DEOS プロセス（DEOS Process）
  - DEOS アーキテクチャ（DEOS Architecture）
- 要求マネジメントプロセス
  - プロセス仕様
    - 要求抽出・リスク分析
    - ステークホルダ合意
  - プロセス支援ツール
    - 合意形成支援ツール（D-Case Editor / Weaver、D-Case Viewer、D-Case Verifier）
    - 合意記述言語（D-Case、D-Case Pattern）
    - 実行手続き記述言語（D-Script）
- DEOS 実行環境（DEOS Runtime Environment：D-RE）
  - D-RE 仕様
  - D-RE リファレンス実装
- DEOS 開発支援ツール（DEOS Development Support Tools：D-DST）
  - DS-Bench/Test-Env
  - Model/Type Checker
- 各種要素技術（各要素技術仕様書、API 定義書、ソフトウェア、実装ガイドライン等）
- 規格・標準・ガイドライン
- コンソーシアム

1.3. DEOS プロジェクト研究開発体制

本研究領域は研究総括・副研究総括のもとに領域アドバイザーを置き、研究領域の方向性や内容に関するアドバイス及び研究チームの進捗評価を行っている。領域運営アドバイザーは実用化に向けての指針などをアドバイスしている。また、研究推進委員として企業の技術者等がプロジェクトに協力して、本研究成果を実利用する立場からアドバイスを行うなど研究チームとの交流を行っている。

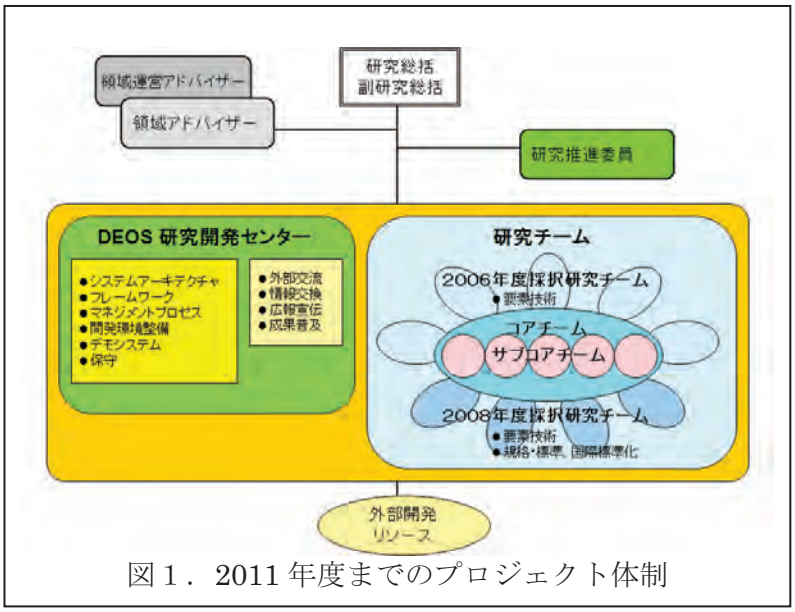


図１．2011 年度までのプロジェクト体制

本研究領域は 2006 年に採択された５研究チームによって開始され、2008 年に新たに４研究チームがこれに加わった。2006 年度採択チームはバーチャルマシン、サーバ群の仮想化、プログラム検証、ベンチマーキング、フォルトシミュレーション、リアルタイム・低消費電力化など、主に要素技術からの研究開発を開始するとともに、対象とするべきシステムの検討やディペンダビリティの概念の議論を行った。2008 年採択チームは 2006 年度採択チームに加わって、DEOS プロセス、DEOS アーキテクチャを構築し、また、要求分析手法、合意形成とその記述方法、セキュリティ、などの研究開発に加え、国際標準化活動を行っている。2006 年度採択研究チームは 2012 年３月、2008 年度採択チームは 2014 年３月まで研究開発を継続する。

採択された研究チームの活動と並行して、2008 年より各研究チームからメンバーを集めてコアチームを形成し、研究全体を俯瞰し、その方向ならびに具体的な研究開発テーマを再定義するプロジェクト活動を行ってきた。そして 2010 年度より具体的な研究テーマを担当するサブコアチームを構成し、クロスチームでの研究開発を行って来た。DEOS プロセス及びアーキテクチャの主な構成要素の研究・開発を担当し、これまでに、D-Case & Metrics、D-Script & Monitor、VM & Multi-OS、System Software Verification、DS-Bench & Test-Env などのサブコアチームが構成され、成果を上げた（図１参照）。

2006 年度採択チームが 2011 年度で研究活動を完了した。2012 年４月からはプロセス・アーキテクチャ会議を発足させた。現在は以下の６つのエリアにフォーカスして研究を推進している（図２参照）。

- D-Case
- D-Script
- ADD
- Security
- DEOS 応用
- 検証と標準化

プロジェクトおよび個々の研究チームの成果はディペンダブル組込み OS 研究開発センター（DEOS 研究開発センター）において、実用のための統合、知的資産や保守を考慮した再構成、テスト、実用のための評価やパッケージングなどを行い、企業との共同の評価や実際の製品での活用などにつなげていく。現在 DEOS プロジェクト Home Page（http://www.dependable-os.net/）で研究成果を公開している。

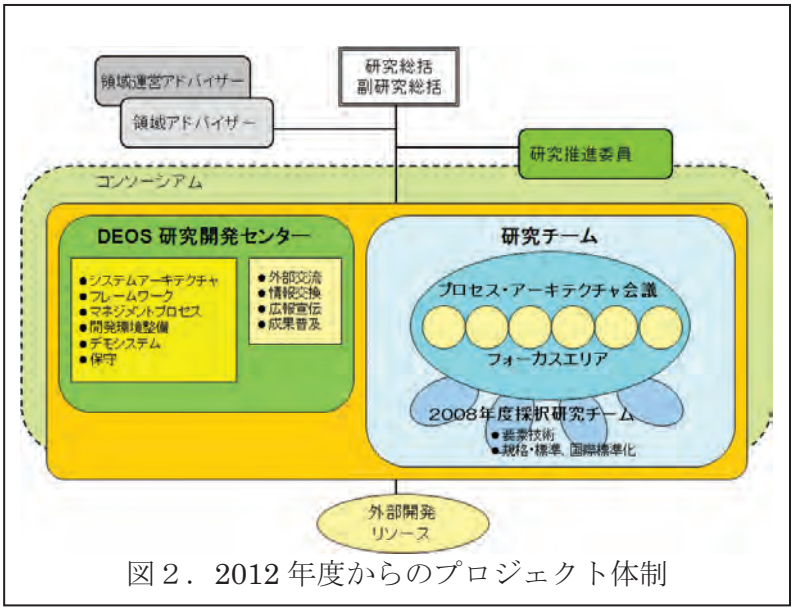


図２．2012 年度からのプロジェクト体制

1.4. DEOS プロジェクト・ロードマップ

以下のフェーズを主なマイルストーンとして全体の研究を進めて行く（図３参照）。

- フェーズ１（2006/10-2009/9）： ディペンダビリティのコンセプトの確立、それを支える開発・運用プロセスや重要な評価指標を含むシステムアーキテクチャの提示、

および 2006 年度採択研究チームの要素技術のいくつかをインテグレーションしたデモシステムによるデモ。（以上を 2009 年 9 月に 2006 年度採択究チーム中間成果報告会として公開した）

- フェーズ 2（2009/10-2011/9）：　システムアーキテクチャと 2006 年度採択研究チームの要素技術を取り込んだ D-RE とツール類の実装。企業や研究機関などを含む実証団体募集活動の開始。必要な事項の国際標準化活動。2008 年度採択研究チームの要素技術のいくつかを D-RE とツール群にインテグレーションしてデモ。（以上を 2011 年に 2006 年度採択研究チーム最終成果報告会および 2008 年度採択研究チーム中間成果報告会として公開した）
- フェーズ 3（2011/10-2014/3）：実証団体による成果物等の試用、およびその評価のフィードバックの継続と実際の開発や実用化への移行。コンソーシアムの設立、DEOS コンセプトの業界標準化活動。必要な事項の国際規格標準化活動の推進。
- フェーズ 4（2014/4- ）：国際規格団体、業界標準団体もしくはコンソーシアムによる成果の活用と維持・発展の推進。

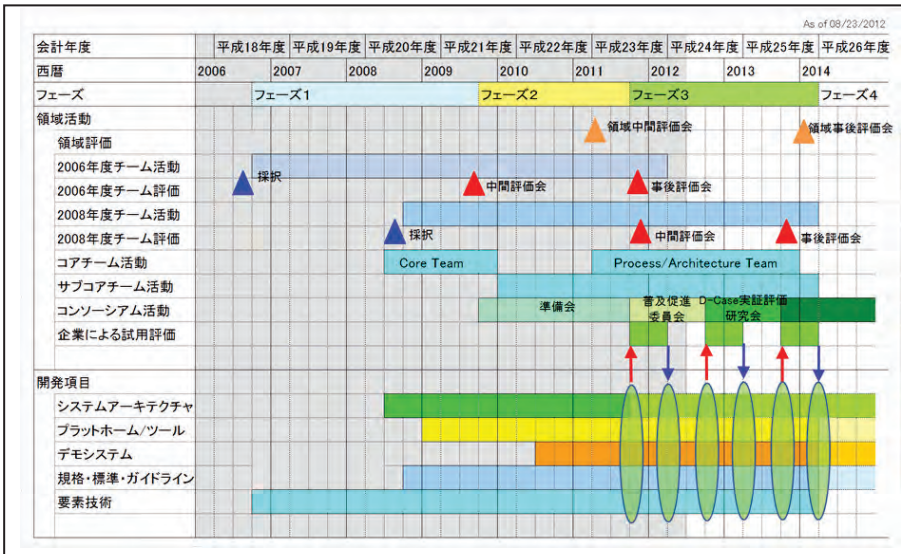


図３．ロードマップ

2. ディペンダビリティ

2.1. 考え方の変遷

1960 年代になると、コンピュータの実時間かつミッションクリティカルな利用に対応するためにフォルトトレラント計算機（Fault Tolerant Computer）が提唱され、活発な議論がなされるようになった［15、19］。その後、ハードウェアならびにソフトウェア規模の増大やオンラインサービスの普及に伴い、故障しにくい性質（信頼性 Reliability）、高い稼働率を維持する性質（可用性 Availability）、障害が発生した場合に迅速に復旧できる性質（保守性 Serviceability あるいは Maintainability）と言った 3 つの性質を一体化した RAS（ラス）という概念が出され、システムのエラー検出と回復に重点を置いて発展していった［8、14］。1970 年代後半にはこれにデータが矛盾を起こさずに一貫性を保つ性質（保全性 Integrity）、機密性が高く不正にアクセスされにくい性質（安全性 Security） を加え、RAS を拡張した RASIS（レイシス）という概念でシステムを評価するようになってきた。2000 年代に入ると、ネットワークで結合された複雑なシステムを想定し、自律神経系を模したシステム構成によりできる限り自律的にディペンダビリティを確保しようとする自律型コンピューティング （Autonomic Computing）の考え方が提案された［9、10、11、16、38］。

情報システムのディペンダビリティの実現に欠かすことができないソフトウェア開発手法についても変遷が見られる。構造化プログラミング（Dijkstra [29]）やオブジェクト指向プログラミング（SIMULA [30]、Smalltalk [41]）のようなプログラミング手法から始まったソフトウェア開発手法は、その後、ソフトウェア開発プロジェクトのマネジメント手法へと発展し、さらにはソフトウェアの開発プロセス（CMM [31,32]、CMMI [33]）へと視点が移った。また、複雑な大規模システムの開発手法に関するプロジェクト（System of Systems、Ultra-Large-Scale Systems [34]）もスタートしている。また、CoBIT や ITIL では、企業・自治体といった組織における IT ガバナンスや IT サービスマネジメントのベストプラクティスをまとめている。

信頼性や安全性に対する考え方の変化は国際標準においても表れている。信頼性に関する国際標準としては、IEC60300 シリーズがディペンダビリティマネジメントの規格として知られている。これらの規格を策定している IEC TC56 はもともと電子部品の信頼性に関するテクニカルコミッティであったことから、IEC 60300 シリーズの核となる IEC 60300-1（2003 年版）の規格はソフトウェアを含む形で十分議論されていない。そのため現在の改訂作業ではディペンダビリティマネジメントの対象を製品・システム・サービス・プロセスに拡大して展開している。国際安全規格 ISO 13849-1 （EN954-1） や電気安全規格 IEC 60204-1 は単純な部品や機器などに関するもので、ソフトウェアを含むシステムに対応していなかった。ソフトウェアを含むシステムの安全規格の必要性から 2000 年に機能安全規格 IEC 61508 が制定された。IEC 61508 では機器の障害を「不規則なハードウェア故障」と「系統的障害」に分ける。前者は部品の劣化による故障から故障確率を算出し、後者はシステムの設計・開発・製造、保守・運用に起因する障害を安全ライフサイクルに基づいた手順と文書化および V 字モデルなどによるソフトウェア検証により許容目標値以下にする。また、このようにして設計・開発・製造されたシステムに対し、運転モードを低需要運転モードと高需要/連続運転モードに分け、それぞれのモード毎に目標故障限度を定め、安全完全性レベル Safety Integrity Level （SIL）として管理する。SIL1 から SIL4 までの 4 段階で要求レベルが規定されている（SIL4 が最も高い安全完全性を要求する）。IEC 61508 をもとに、機械類関連の IEC 62061、プロセス関連 IEC 61511、原子力関連 IEC 61513、鉄道関連 IEC 62278、などが規定され、自動車関連では ISO 26262 が 2011 年に制定された。

また、多くの考え方をまとめてディペンダビリティに関する定義を統一化しようとする試みも続けられ、1980 年に IFIP WG10.4 on "Dependable Computing and Fault Tolerance” と IEEE TC on Fault Tolerant Computing は合同で「ディペンダビリティの基本概念と用語法」に関する検討が開始された。その検討の経緯と結果をまとめた論文が 2004 年に出版された［2、3］。

しかしながら、ディペンダビリティの研究やそれに基づいた技術の開発にもかかわらず、近年においても大規模ソフトウェアシステムの障害が発生している。それらのいくつかの例を 4. 4. 節に挙げた。障害原因を分析すると、システム構築の際に全ての構成要素を理解しないまま開発を

進めたことによるもの、利用者数やトランザクション数、さらにはデータ量や処理範囲が当初の設計値を越えたことによるもの、利用者の要求変化に応えるために機能を追加・変更した際に発生したシステムの動作の不整合によるもの、などが顕著である。加えて、プログラマーやオペレータによる不用意なミスが連鎖的にシステム全体のダウンを引き起こした例もある。

ディペンダビリティに関する考え方は時代の要求にこたえるようにこれまでも大きく変化してきている。しかしながら、これまでの考え方は、今我々が対象とするようなシステム、すなわち、変化しつづける目的や環境の中でシステムを適切に対応させ、継続的にユーザが求めるサービスを提供することを可能とする大規模システムを対象としたとき、必ずしも十分ではない。なぜなら、現代のシステムは、作られ方の観点でも、使われ方の観点でも、もはやシステムの機能や構造、システムの境界が定義可能な固定的なシステムとして取り扱う事は困難であり、当初より時間の経過とともにそれらが変化するシステムとして取り扱う事が妥当であると思われる。以下に現代のシステムの特徴と障害要因を整理したあと、現代のシステムのための新しいディペンダビリティの考え方を提案する。

2.2. 現代のシステムの特徴と障害要因

現代の大規模ソフトウェアシステムは利用者の多種多様なニーズに対応するために高度化、巨大化、複雑化の一途をたどっている。このようなシステムの開発においては、開発期間を短縮し、開発コストを下げるため、既存のソフトウェアを再利用し、あるいは他社から供給されるソフトウェア部品をブラックボックスとして使うケースが増えて来ている。また、システム運用中にサービスの向上や変更のための仕様変更が行われ、システムのサービスを中断することなく、ネットワークを介して修正がダウンロードされることも一般的である。そのため、開発からサービス終了に至るすべての時点に対して、設計者・開発者や運用者がシステムの隅々まで完全に理解することが極めて難しくなって来ている（図４参照）。

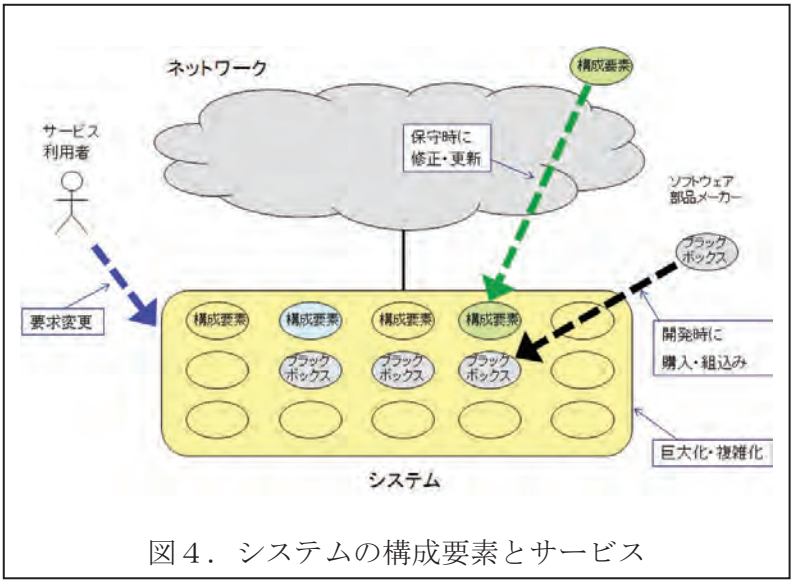


図４．システムの構成要素とサービス

現代のソフトウェアシステムの多くは、ネットワークを介して他のシステムと接続された形でサービスを提供している。利用者は直接的には一つのサービスドメインが提供するサービスを利用するが、間接的に他のサービスドメインが提供するサービスを利用していることがある。そしてそれらのサービスドメインは異なる所有者によって所有され、運用されていることが多い。その場合、サービスの項目や内容、処理性能、インタフェース仕様などが十分に告知されないまま変更される可能性があり、未知のサービスが適用されたり、時にはサービスが終了される場合もある。ネットワーク自体のサービス項目や内容、処理性能、インタフェース仕様も変更されたり、一時的にサービスが停止することもある。このように、利用者から見たとき、あるいはシステム開発者から見たときに、システムあるいはサービスドメイ

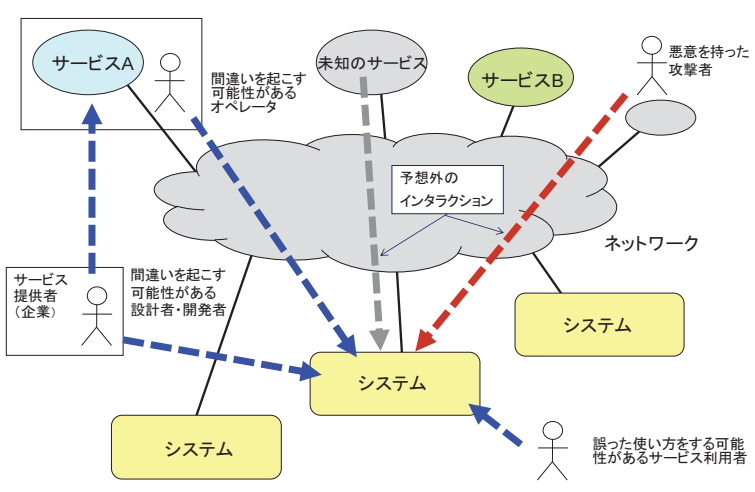


図５．人間系を含むサービスのネットワーク

ンの境界も不明確となる。これに加えて、あるいは悪意を持った攻撃者が意図的に攻撃してくる恐れもある。このように、ネットワーク化に伴う予測不能性が増えている（図５参照）。

これらをシステムの開発、運用の面からとらえ、システムに対する障害要因を検討し、分類すると、以下のようになる。

（1） 不完全さ

要求に対して仕様が完全でなく、また、仕様に対して実装が完全でなく、出荷時や運用時のシステムの振る舞いを完全に把握することが困難であること。この障害要因の例として以下のものがあげられる：

- システムが多くのソフトウェアの組合せから作られており、巨大化、複雑化に伴い網羅的な仕様記述やテストが不可能
- 要求・仕様・設計・実装・テストなどの各開発フェーズにおける理解の違い、文書の誤りなどによる仕様ミスや漏れ、設計ミスや漏れ、実装ミスや漏れ、テストミスや漏れ
- 管理、運用、保守における変更や修正の失敗やライセンスの期限切れ
- ブラックボックスソフトウェアやレガシーコードの動作と仕様の不一致

（2） 不確実さ

事業者や利用者の要求やシステム環境がライフサイクルを通して変化し、設計時や運用時に機能や挙動を完全に予測できない事。この障害要因の例として以下が挙げられる：

- 事業者の事業目的の変化によるシステムへの要求の変化
- 利用者の要求の変化、システムへの期待値の変化、操作能力や習熟度の変化、など
- 出荷数・使用者数の増加、稼働経済性の変化による使われ方の変容
- 上記変容に対応するため現場で（人手を介して、ネットワークを介して）構成要素の機能修正やサービス変更、システムの再構成を行う事（複雑性の増加）
- ネットワークを介した環境による想定外の接続・インタラクションの増加、外部からの意図的な攻撃を受ける事

すなわち、現代の大規模ソフトウェアシステムは不完全さと不確実さを完全に排除することが極めて困難である。このため、全ての障害を完全に回避することは極めて難しい状況にある。

現代の大規模ソフトウェアシステムの状況から、これまでにもディペンダビリティを定義するためにいろいろな表現が試みられてきた。たとえば、「故障や障害がまったく起こらない状態が望ましいが、異常が発生した時には直ちに状況が把握でき、先の状況が予測可能であり、社会的なパニックやカタストロフィックな破綻を引き起こさない事が保障できる状態を、適正なコストで維持し続けること」［7］や「様々なアクシデントがあったとしても、システムが提供するサービスを、利用者が許容できるレベルで維持すること」［17］がある。

われわれは全ての障害を完全に回避することが困難であるとしても、致命的な障害の発生をできる限り減らし、万が一障害が発生した場合には被害を最小にし、同様な障害の再発を防止し、説明責任を果たし、事業を継続可能とするための方法や技術を開発することはできると考えている。我々はこのことを目標とし、以下にディペンダビリティを再定義し、そのための方法ならびに技術を開発する。

2.3. オープンシステムディペンダビリティ

これまでに述べてきたように、我々が対象とするシステムは、仕様や実装の不完全性と利用者の要求や使用環境の変化に起因する不確実性を完全に排除できない。そのような性質を持つシステムはオープンシステム（開放型システム、Open Systems）であるということができる。オープンシステムを説明するために、その対極をなすクローズドシステム（閉鎖型システム、Closed Systems）と対比させてそれぞれの特徴を列挙する。

クローズドシステムの一般的な特徴は以下のとおりである（図６参照）。

- システムの境界が定義できる
- 外部との相互作用が限定的で、システムの機能が一定である
- システムの構造（構成要素）が固定的で、構成要素間の関係が一定である
- 外部観測者視点がとれる
- 要素還元主義が成り立つ

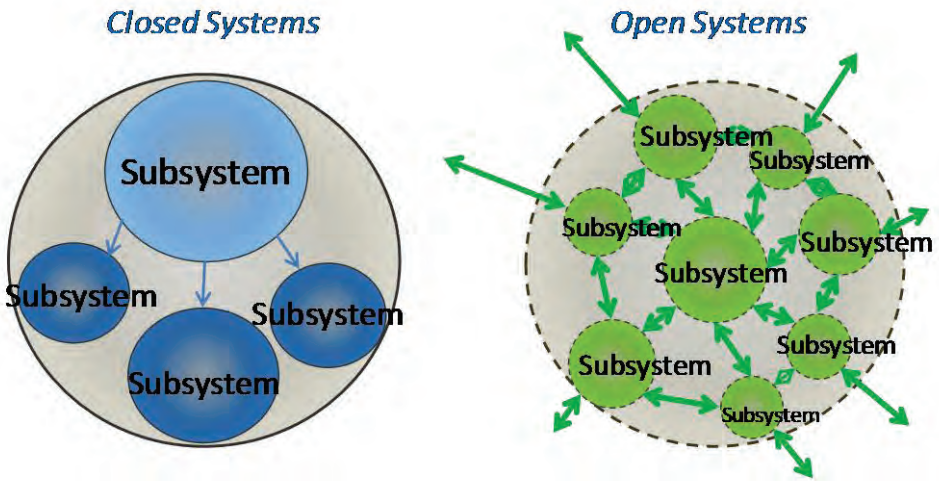


図６．クローズドシステムとオープンシステム

一方、オープンシステムの一般的な特徴は以下のとおりである（図６参照）。

- システムの境界が時間とともに変化する
- 外部との相互作用がある。システムの機能が時間とともに変化する
- システムの構造（構成要素）ならびに構成要素間の関係が時間とともに変化する
- 観測者自身がシステムに含まれるため、内部観測者視点しか取りえない
- 要素還元主義が成り立たない

我々が対象とするシステムは機能、構造、境界が時間の経過とともに変化する、と言うオープンシステムの一般的な特徴を備えている。そして常に変化するために要素還元主義が成り立たないことがあることや、システムの所有者、設計者、開発者、運用者、利用者など、システムの開発や運用にかかわるすべての人がシステムに影響を与えるという意味で内部観測者であると言う事もできる［13］。

対象システムを時点時点でクローズドシステム、すなわち時間的な変化がないシステムとしてとらえ、その継続としてディペンダビリティを考えてゆくことも一見可能のように思われる。これまでのディペンダブルシステムの開発は、主にこの視点で行われてきたと考えられる。この場合には、それぞれの時点でシステムの境界を定義し、システムの機能を確定し、仕様を策定し、これに基づいてシステムの設計を行い、検証、テストを行い、これを繰り返すことになる。しかしながら、通常はいくつかの変更と対応が同時並行的に進む中でサービスを継続することが行われるため、実際にはシステムを固定期間と変更期間に区別することが極めて困難である。

それであれば、我々の視点を「変化するシステム」に移し、システムの継続的変化に対するサービスや事業の継続性維持を主眼としたディペンダビリティの概念を確立すべきであろう。すなわち、我々は対象システムをオープンシステムとしてとらえ、時間の流れの中でいかにディペンダビリティを高めて行くか、と言う視点を取ることにする。そして、現代の多くのソフトウェアシステムに特質的な問題点を捉えて、これからのディペンダビリティを「オープンシステムディペンダビリティ（OSD：Open Systems Dependability）」として次のように定義する。

現代の大規模ソフトウェアシステムは機能、構造、システム境界が時間的に変化し、これに起因する不完全さと不確実さを完全に排除することができず、未来に障害となりうる要因（開放系障害要因）を本質的に抱えている。オープンシステムディペンダビリティとは、それらの要因を顕在化する前にできる限り取り除き、また、顕在化した後に迅速かつ適切に対応し、影響を最小とするようにマネージし、利用者が期待する便益をできる限り安全にかつ継続的に提供し、社会への説明責任を全うし、およびそれらを継続的に行う能力を言う。

OSD はこれまで多くの研究者が研究し、議論し、整理して来たディペンダビリティの概念を否定するものではない。これまでは、システムを時点時点でとらえ、主に偶発的フォルトや意図的フォルトに焦点を当て、システムの安心安全を高めるための技術が研究され議論され開発されて来た。これに対して我々は対象を時間的に変化するシステムとしてとらえ、不完全さと不確実さに起因する開放系障害に焦点を当て、開放系障害を起こす要因の最小化と、開放系障害による影響の最小化によりシステムのディペンダビリティを向上させようと考えた。すなわち OSD はこれまでのディペンダビリティを強化するものである。

3. オープンシステムディペンダビリティの実現

前章で定義したように、オープンシステムディペンダビリティは「変化しつづけるシステム」に対するディペンダビリティである。オープンシステムディペンダビリティの実現のためには、反復的プロセスとしてのアプローチが必須であり、そのようなプロセスは、目的や環境の変化に対してシステムを継続的に変更して行くためのサイクルと、障害に対して迅速に対応するためのサイクル、を備えていなければならないと考える。そして、それらのサイクルからなるプロセスは、構成要素として要求マネジメントプロセス、開発プロセス、通常運用プロセス、障害対応プロセス、説明責任遂行プロセスなどを含む「プロセスのプロセス（Process of Processes）」であり、それらの構成要素プロセスは相互に有機的に結びつけられていなければならないと考える。我々はそのような統合的反復プロセスとして DEOS プロセスを提案する。

DEOS プロセスの実施にはそれを支援するための仕組みが必須である。このような仕組みは要求マネジメントプロセスを支援するためのツール群、合意内容を収納し維持するための合意記述データベース、ディペンダブルなソフトウェアを開発するためのプログラム検証やベンチマーキング、フォールトインジェクションテストなどのツール群、システムの状態を常にモニターし、記録・報告し、障害発生時に動的に対応して障害の影響を最小限にとどめるためのプログラム実行環境、などをそなえていなければならないと考える。我々はそのような仕組みを DEOS アーキテクチャとして提案する。

3.1. DEOS プロセス

対象システムのディペンダビリティに関する利害関係者を DEOS プロセスにおいては「ステークホルダ」と呼ぶ。ステークホルダとして我々は以下を想定している。

- サービス・製品の利用者（顧客、社会的インフラの場合は社会全体）
- サービス・製品の提供者（事業主）
- システム提供者
  - 設計開発者
  - 保守運用者
  - ハードウェア供給者
- サービス・製品認可者（規制監督官庁）

ステークホルダは時間の経過や環境の変化によってそれぞれの目的を変化させ、機能やサービスに対する要求を変化させる可能性がある。これらの変化をここでは「目的・環境変化」と呼ぶ事とする。これらの変化に対し、ステークホルダは熟慮し、相互に合意したうえで、適切な時期にシステムの変更を要求する。DEOS プロセスはこのような要求に対するサイクルとして、「変化対応サイクル」を備える。

対象システムは不完全さと不確実さに起因する障害を完全に回避することがきわめて困難である。障害の予兆を検出した場合には障害を未然に回避し、不幸にも障害が発生してしまった場合には迅速に対応して被害を最小化し、原因を究明し、説明責任を遂行する必要がある。DEOS プロセスではそのような状況に対応するために「障害対応サイクル」を備える。

新規にシステムを開発したり、目的・環境の変化に対応してシステムの変更を行う場合、その理由やステークホルダ間で行われた議論の過程、合意内容などを詳細に記録するための合意記述データベースを備えていることが効果的な反復的プロセスを実現し、説明責任を遂行するために必須である。このデータベースにはディペンダビリティを達成するための議論や根拠を記述した D-Case、障害に対してサービスを継続するためのシナリオを基に障害予兆の検出や障害発生に迅速に対応するための実行手続きを記述した D-Script が含まれていなければならない。そして、これらの合意記述を基に開発プロセス、障害対応プロセス、通常運用プロセスが実行され、また、説明責任遂行プロセスを支援することができる。合意記述データベースは構成要素プロセスを有機的に繋ぐ重要な役割を果たす。

DEOS プロセスの特徴をまとめると、以下のとおりである（図７参照）。

①「通常運用」から開始される「変化対応サイクル（青い外側ループ）」と「障害対応サイクル（赤い内側ループ）」の２つのサイクルから成り立っていること

②システム変更要求のための「ステークホルダ合意」と、システム変更や障害対応の「説明責任遂行」の２つのフェーズ（黄色いボックス）が組み入れられていること

③ステークホルダ間で利害関係を調整し、ディペンダビリティを達成するために議論した過程、論拠、結果等を記述した「D-Case」と障害に迅速に対応するための実行手続きを記述した「D-Script」を含む合意記述データベースを備え、構成要素プロセスを有機的に結合すること。

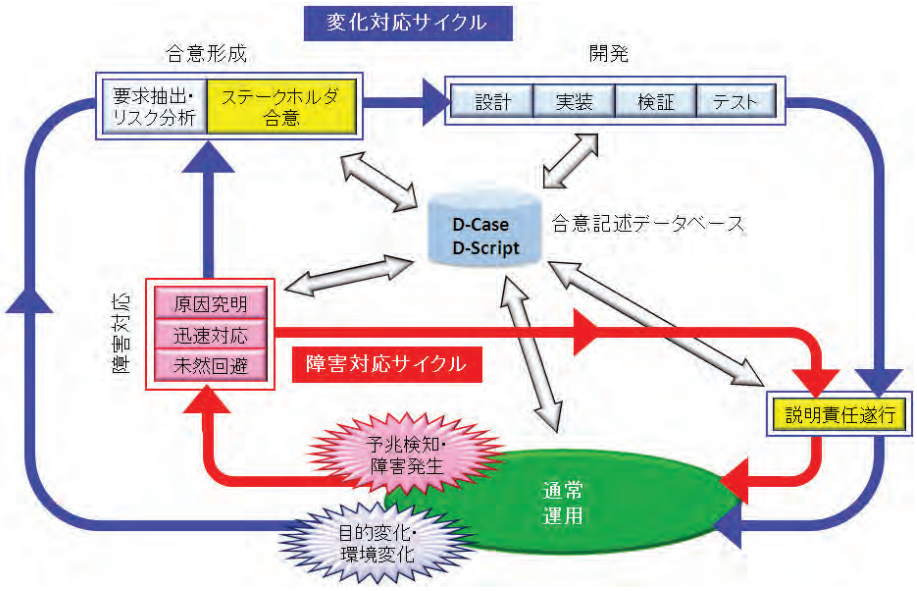


図７．DEOS プロセス

以下に DEOS プロセスを詳しく説明する。

3.1.1. 通常運用プロセス

通常運用はシステムがステークホルダ間で合意されたサービスレベル変動許容範囲（In-Operation Range）から大幅な逸脱がなく、ユーザに対してサービス提供を継続している状態である。変化対応サイクルは通常運用と並行して実行され、サービスの提供を継続しつつシステムの変更が行われることが望ましい。同様に、障害対応サイクルも通常運用を継続しながら実行されることが望ましい。実際、システムが異常の予兆を検知しても、D-Script に記されたサービス・機能レベル変動許容範囲内で自動的に回避処理が働いてサービスが継続される場合がある。あるいは一部の機能を縮退してサービスを継続している場合もある。しかしながらサービスの提供が完全に停止されてしまう場合もある。

運用状態において実行されるプロセスとしては、日常的な動作記録の点検、プロセスの定期的な見直し・改善、要員の訓練・しつけ・教育など、継続的なディペンダビリティ向上活動が行われる。システムの稼働状況を記録し、日々点検する事により保守担当者や運用担当者が、何かの兆候をそこから見出す事ができる可能性がある。また、システムのメモリー資源を常にクリーンな状態にすることも、非常に有効な日常保守・改善活動である。あるいは、積極的に予行を行う事も有効である。障害はある時間が経過してある状態に達した時発生する。であれば、時間を先に経過させると障害の発生を事前に知る事ができる。いわゆるリハーサルである。情報システムの提供するサービスの運用時において、どの程度適切なリハーサルができるのかは場合による。

3.1.2. 変化対応サイクル

変化対応サイクルはステークホルダの目的の変化や、各種外部環境の変化に対応するためのサイクルである。このサイクルにおける主要なフェーズは、システム変更のための「要求抽出・リスク分析」、「ステークホルダ合意」、「設計・実装・検証・テスト」である。大きな変化に対応する場合は「説明責任遂行」が必須となる。障害対応サイクルにおける原因究明フェーズの実行の結果、システムの根本的な改良の要求が発生した場合も、変化対応サイクルが開始される。

要求抽出・リスク分析フェーズは目的や環境の変化によりステークホルダからの要求が変化（新規の要求も含む）した場合、あるいは障害発生に迅速に対応した後、原因究明を行った結果、システムの変更が必要である場合に始まる。いずれの場合も、事業主のサービス目的をベースにユーザ

の要求、システム環境、関連する法律や国際標準を勘案し、システムの機能要件を抽出する。また同時に、サービス目的からシステムのサービス継続シナリオを作成してリスク分析を行い、ディペンダビリティ要件を含む非機能要件を抽出する。

ステークホルダ合意フェーズでは、何をどのように変更するのかを、すべてのステークホルダに分かりやすく、誤解のないように記述し、ステークホルダ間の議論を経て、合意を D-Case として記述する。またサービス継続シナリオを作成し、その実行手続きである D-Script を作成する。要求抽出・リスク分析フェーズとステークホルダ合意フェーズが「要求マネジメントプロセス」を構成する。

設計・実証・検証・テストフェーズは、いわゆる設計開発のプロセスである。ここでは、これまで多くの研究がなされ、多くの手法やツールが出されている。優れた手法やツールは積極的に活用すべきだと考える。我々のプロジェクトでは DEOS プロセスの強化のために必要なソフトウェア検証やベンチマーキング、フォールトインジェクションテストなどのツール群を開発している。

説明責任遂行フェーズでは、目的や環境変化によるステークホルダの要求変化を満たすためにシステムを変更した場合、その経緯と、いつからどのようにサービスや機能がよくなるのか（変化するのか）を説明する。また、日常のサービス遂行状況や設計開発・保守運用プロセスに関する説明が必要なときもこれに対応する。これは利用者や社会からの信頼を維持し、インフラサービス提供上のコンセンサスを醸成し、ひいてはサービス提供者のビジネス遂行上の便益を守るという大変重要な役目を持つ。合意記述データベース特に D-Case 記述が説明責任遂行に役立つ。

3.1.3. 障害対応サイクル

障害対応サイクルは障害に対して迅速に対応して障害による被害を最小化するためのサイクルである。DEOS プロセスでは「障害」をステークホルダ間で合意されたサービス・機能レベル変動許容範囲から逸脱する事象と定義する。

障害対応サイクルにおける主要なフェーズは、「未然回避」、「迅速対応」、「原因究明」であり、障害が発生した場合は「説明責任遂行」が必須である。「未然回避」、「迅速対応」、「原因究明」はそれぞれ別個に、かつ順番に行われるとは限らない。多くの場合、これらはお互いが関連しあい、渾然一体となった事象・活動となる。

未然回避フェーズは、システムのオペレーション中に障害が発生する前に障害発生を予知したり、あるいは障害が起きる可能性の増大を検出すると、障害を回避するように対応・動作するフェーズである。障害の予知が障害の発生予想時刻の充分に前であれば効果的な対策が打てる。例えばシステムの資源を制限してスループットを下げてシステムダウンを回避したりシステムダウンまでの時間を稼いだりすることが行われる。直前に予知した場合には障害の影響の最小化に努力することになる。また、原因解析に有効な、障害に至るまでのシステムの内部情報を記録することができる。予知のための具体的な方法としては、過去の障害パターンから類似の障害を判別する事などがある。未然回避シナリオは D-Script に事前に記述され、オペレータやシステム管理者と協調して未然回避動作が実行される。

迅速対応フェーズは、障害が起きた時にその影響を最小化するためのフェーズである。障害に対する迅速対応のシナリオは D-Script に事前に記述されおり、自動的に行われるのが望ましい。しかしながら、想定しない障害にも対応しなければならない場面もある。対応分野や領域ごとの目的に応じたサービス継続のための緊急対応計画（責任者や対応組織、手順、エスカレーションパスなどが記されている）を事前に立てて、ステークホルダ間で合意しておく事が求められる。その計画の指示に基づきオペレータやシステム管理者と協調して迅速に障害による影響を最小化することになる。すなわち、障害を分離して影響の局所化を行い、サービス全体のダウンを回避する。そのために障害が発生したアプリケーションやシステムの一部分のオペレーションを中断し、リセットし、その後にオペレータやシステム管理者による復旧活動が行われる。

原因究明フェーズは、障害対応サイクルと変化対応サイクルに関連したフェーズである。サイクルにより深さの違う判断がなされる。障害対応サイクルでは、どのような迅速対応が可能であるかを見極めることを目的とした原因究明がおこなわれる。その結果によっては変化対応サイクルが開始される。

説明責任遂行フェーズでは、サービス提供者、特に社会インフラサービス提供者や社会に広く使われる製品提供者が、障害発生時にサービス利用者、製品使用者、社会に対し、障害状況、迅速対応、今後の見通しなど説明する。これは利用者や社会からの信頼を維持し、インフラサービス提供上のコンセンサスを醸成し、ひいてはサービス提供者のビジネス遂行上の便益を守るという大変重要な役目を持つ。合意記述データベース特にD-Case 記述と、システム監視記録が説明責任遂行に大いに役立つ。

3.2. DEOS アーキテクチャ

DEOS プロセスはオープンシステムに対するディペンダビリティを実現するための反復的プロセスを提供している。このプロセスをより具体的に対象とするシステムに適用した場合、対象のカテゴリー毎にプロセス実行のためのアーキテクチャを考える必要がある。ここでは、組込みシステムを含む現代の大規模かつ複雑なソフトウェアシステムへの適応を念頭に考案された DEOS アーキテクチャについて述べる。DEOS プロセスと DEOS アーキテクチャを並べて眺めると DEOS プロセスが実際のシステムでどのように実行されるかが理解しやすい。

DEOS アーキテクチャは以下の構成要素からなる（図８参照）。

- 要求抽出・リスク分析フェーズを支援するためのツール群
- ステークホルダ合意フェーズを支援する合意形成支援ツール D-Case Editor / Weaver、D-Case Viewer、D-Case Verifier
- 合意の記述である D-Case とサービス継続シナリオの実行手続きである D-Script を含む合意記述データベース（Agreement Description Database：ADD）
- DEOS 実行環境（DEOS Runtime Environment：D-RE）
- プログラム検証ツールとベンチマーキングならびにフォールトインジェクションテストのためのツール群を含む DEOS 開発支援ツール（DEOS Development Support Tools：D-DST）

要求抽出・リスク分析フェーズは事業主のサービス目的を基にユーザの要求、システム環境、関連する法律や国際標準を勘案し、システムの機能要件を抽出し、想定される障害に対するサービス継続シナリオを作成してリスク分析を行い、ディペンダビリティ要件を含む非機能要件を抽出する。現在そのためのツール群を開発中である。

ステークホルダ合意フェーズは合意を形成するための方法と合意記述の記法に基づいて合意内容を D-Case として記述する［43］。そのためのツールが Eclipse をベースに作られた D-Case Editor [37]、あるいは Web Browser ベースに Java Script で作られた D-Case Weaver である。

従来のシステムは目的が変化することなく、したがって大きなシステム変更がないため障害マネジメントも予め想定できる障害に対する対処法を作り込むことで対応できた。しかし、近代システムは変化し続ける（システムへの要求変化・外部環境の変化）ため、あらかじめ障害をすべて予見することは困難であり、現実的には不可能となっている。そのため柔軟な障害マネジメントの仕組みを考える必要がある。

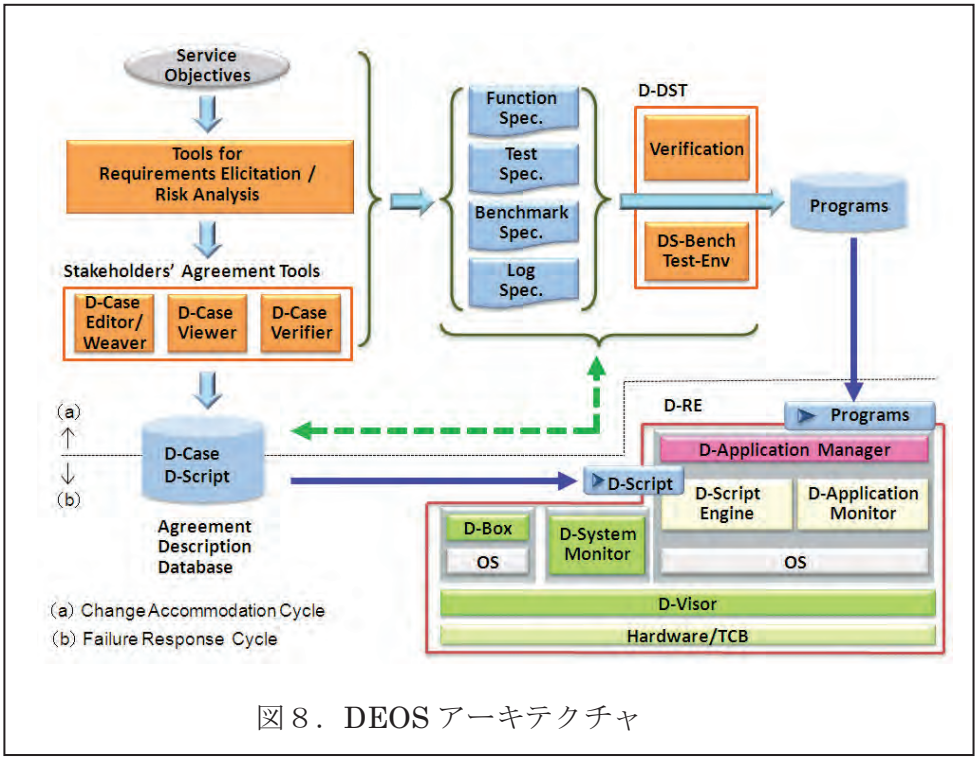


図８．DEOS アーキテクチャ

D-Case で合意された事項に基づいた運用時の柔軟な障害マネジメントの仕組みとしてはスクリプトによりシステムを監視し、記録し、分析し、再構成する。これはコンポーネントなどを再構成（Reconfiguration）したり、障害箇所の隔離（Isolation）や障害箇所の再起動(Reboot)を行うことで障害からの回復を行う。

何を監視するか、何をログとして収集するかは D-Case によりステークホルダ間で合意されており、収集されたログは変化対応サイクルを回す際にステークホルダの意思決定に重要なエビデンスになる。

D-Script は DEOS アーキテクチャにおいて D-Case 記述とアプリケーションプログラムの実行を動的に結合する重要な役割を果たしている。D-Script には D-Script Engine が実行するシナリオが書かれている。そのシナリオは D-RE に対して、1）いつ、どのようなログ情報を収集するかを指示し、また、2）故障発生時においては故障に対してどのように対処するかを指示している。この時、エスカレーションルールに従ってオペレータの介入を指示する場合もある。このように、D-Script は動的かつ双方向に情報を交換することによりアプリケーションプログラムの実行を柔軟に制御し、オープンシステムディペンダビリティの達成に寄与している。

ADD は D-Case や要求マネジメントで取り扱われるドキュメントなどディペンダビリティ情報を格納して、必要な時に適切な情報を抽出することが出来るデータベースである。この中には

- 対象システムの構造（構成）、コンポーネント・システムを基礎とした記述
- ステークホルダ間のディペンダビリティ要求に関する議論
  －過去・現在の議論された事項、議論が中止された事項
  －現在の議論の状況・合意事項、合意されていない事項
  －タイムスタンプ、誰が責任を取るのか等の情報（説明責任の遂行時に有用）
- 過去の全障害に関する情報とその復旧方法、通常運用時に検出されたアノマリー等の情報
  －これらの情報は同じ原因による障害の発生を回避する上で有用
- D-Script：D-Case に対応したスクリプト

などが格納される。ADD は DEOS プロセスの障害対応サイクル、変化対応サイクルを回すうえで要となる中心的存在である。

DEOS 実行環境（D-RE）はステークホルダ合意に基づくディペンダビリティを実現するサービスを提供するための実行環境であり、以下のサブシステムを含む。D-Visor は対象システムの再構成のため、システムの構成要素の各々の独立性を担保する仕組み（System Container）を提供する。ある System Container 内における異常や障害が他に波及することを抑える働きを担っている。D-System Monitor はシステムの動作監視機能を提供する。D-Application Manager は複数のアプリケーションの独立性を担保する仕組み（Application Container）を提供し、各アプリケーションのライフサイクル（起動、更新、停止）を管理し制御する。D-Application Monitor はアプリケーションの動作監視機能を提供し、エビデンスを収集し、D-Box に蓄積する。D-Box はエビデンスを始め、OSD 実現に有益な情報を安全・確実に記録する。D-Script Engine は D-Script を安全・確実に実行する役割を担い、D-Application Manager、D-Application Monitor、D-System Monitor を制御する。

DEOS 開発支援ツール（D-DST）は事業目的や事業継続シナリオに基づいて決められた機能仕様、テスト仕様、ベンチマーキングシナリオ、さらにはログ仕様に基づいてプログラムを設計し、開発し、検証し、ベンチマーキングを行い、テストを行うための開発支援ツール群である。

3.3. DEOS の利点

DEOS プロセスを適用する事により享受できる最大の利点は、ステークホルダ間で要求の変化に対する合意議論を充分に行う事ができ、合意結果や、その結論に至った理由や、議論の経緯を D-Case に記録する事ができる点である。システム開発時に D-Case 記述を用いることにより、DEOS アーキテクチャと連携して、障害時に適切かつ迅速な対応を取ることが可能なシステムを設計することができる。また D-Case 記述がある事により、障害の原因究明や説明責任を果たす事がより容易になる。

DEOS プロセスのもう一つの利点は、要求が適切に抽出されリスクが充分検討されたのちに、システムの変更が実行される点である。それぞれのステークホルダはシステムの状態をどんな時点でも、それぞれの観点で知る事ができる。これによりシステムを簡潔かつ強力に管理運用する事ができる。

DEOS アーキテクチャの利点は、モニタリング機能を備え、解析に必要なシステムやアプリケーションの実行状態の監視と記録を実行し、監視記録と D-Script に従って障害時の迅速対応を実行し、また D-Case 記述や監視記録から得られた情報をエビデンスとして原因分析を遂行する事により、システムの自動的、あるいは必要であればオペレ―タを介した（D-Case 記述に基づく）柔軟な対応を可能にする。また、実行前にプログラムを検証するツール、パフォーマンスを測定するツール、フォルトを埋め込んで異常時の振舞いをテストする開発ツール等も提供し、ディペンダビリティを向上する。

4.4.節「近年の障害事例」にみられるように、現代の大規模システムの障害は設計ミスやプログラムの単純なバグに起因する事例もあるが、運用時のユーザの使い方やデータの量など開発時の設計からの変化や、運用時の緊急対応への検討・準備不足や、ネットワーク環境での想定外のシステム間の相互作用、等に起因する事例が多い。DEOS では、このような問題に対して上記の仕組みや機能を利用し、DEOS プロセスと DEOS アーキテクチャを用いることで継続的な障害回避のための能力を備え、障害時には適切かつ迅速な対応をして、その影響を最小限にすることが可能となる。さらに、サービスを継続し、説明責任を遂行することができる。DEOS プロセスと DEOS アーキテクチャはオープンシステムディペンダビリティを実現するための方法論ならびに構築法である。

4. プロジェクト関連情報

4.1. 関連報告書・書籍

White Paper（DEOS プロジェクト文書）

DEOS-FY2009-WP-01J ：DEOS Project White Paper Version 1.0
DEOS-FY2010-WP-02J ：DEOS Project White Paper Version 2.0
DEOS-FY2011-WP-03J ：DEOS Project White Paper Version 3.0

フォーカスエリア技術報告書（DEOS プロジェクト文書）

DEOS-FY2012-DC-01J ：D-Case ― ディペンダビリティ合意形成のための手法とツール
DEOS-FY2012-DS-01J ：D-Script：スクリプトによる障害対応の実現
DEOS-FY2012-DA-01J ：合意記述データベース ― オープンシステムディペンダビリティと D-Case を繋ぐリポジトリ
DEOS-FY2012-SD-01J ：サービス延命を可能とする基盤システムソフトウェア
DEOS-FY2012-RA-01J ：D-Case のロボット応用 ― 日本科学未来館フロア移動ロボットを題材として
DEOS-FY2012-SV-01J ：D-Case/Agda によるアシュランス・ケース記述

出版書籍

ISBN：978-1-46657-751-0 “Open Systems Dependability” （CRC Press）
ISBN：978-4-86293-079-8 「D-Case 入門」（株式会社ダイテックホールディング）

4.2. 参照・参考資料

1. 安浦 寛人、「Dependability について」、南谷 崇、「ディペンダビリティの概念と課題」、岩野 和生、「社会サービスの Dependability」、ディペンダビリティワークショップ報告書 CRDS-FY2006-WR-07, CRDS, JST, March 2007
2. http://www.dependability.org/wg10.4/
3. A. Avizienis, J.-C. Laprie, B. Randell, C. E. Landwehr, “Basic Concepts and Taxonomy of Dependable and Secure Computing”, IEEE Trans. On Dependable and Secure Computing, Vol.1, No. 1, Jan.-March 2004
4. T. Kikuno, “Requirements for Dependability in the 21th Century”, Speech at the Kickoff Symposium of JST/CREST Dependable Embedded OS Project, December 2006
5. M. Tokoro, “On Designing Dependable Operating Systems for Social Infrastructures”, Keynote Speech at MPSoC, Awaji Island, Japan, June 25, 2007.
6. 安浦 寛人、「社会システムを支えるディペンダブルコンピューティング」、電子情報通信学会誌、Vol.90, No.5, May. 2007, pages 399-405
7. 加納 敏行、菊池 芳秀、「ディペンダブル IT・ネットワークとは」、NEC 技法、Vol.59, No.3, 2006, pages 6-10
8. M. Y. Hsiao, W. C. Carter, J. W. Thomas, and W. R. Stringfellow, “Reliability, Availability, and Serviceability of IBM Computer Systems: A Quarter Century of Progress”, IBM J. Res. Develop., Vol. 25, No. 5, 1981, pages 453-465
9. A. G. Ganek and T. A. Corbi, “The dawning of the autonomic computing era”, IBM Systems Journal, Vol 42, No. 1, 2003, pages 5-18
10. “An architectural blueprint for autonomic computing, 4th edition”, IBM Autonomic Computing White Paper, June 2006
11. http://www-03.ibm.com/autonomic/

12. 所　眞理雄、「技術成熟期における研究開発」、電子情報通信学会誌、Vol.90, No.9, 2007, pages 742-744
13. 所　眞理雄、他、「オープン システム サイエンス」、NTT 出版
14. H. B. Diab, A. Y. Zomaya, “Dependable Computing Systems”, Wiley-Interscience
15. G. M. Koob, C. G. Lau, “Foundations of Dependable Computing”, Kluwer Academic Publishers
16. M. C. Huebscher, J. A. McCann, “A survey of Autonomic Computing”, ACM Computing Surveys, Vol. 40, No.3, Article 7, August 2008, pages 7:1-7:28
17. 松田　晃一、巻頭言、IPA SEC journal No.16, 第５巻第１号（通巻１６号）2009, page 1
18. T. Forbath, interview「日本企業は２０世紀型の開発プロセスから脱却すべき」NIKKEI ELECTRONICS 2009.2.23, page 29
19. A.Avizienis, ” Design of fault-tolerant computers”, In Proc. 1967 Fall Joint Computer Conf., AFIPS Conf. Proc.Vol.31, 1967, pages 733-743
20. ナシーム N タレブ、「ブラックスワン」、ダイヤモンド社
21. ナンシーG レブソン、「セーフウェア」、翔泳社
22. 「不具合連鎖 － プリウス リコールからの警鐘」、日経 BP 社
23. 大和田　尚孝、他、「システムはなぜダウンするのか」、日経 BP 社
24. H. B. Diab, A. Y. Zomaya, “DEPENDABLE COMPUTING SYSTEMS”, WILEY-INTERSCIENCE
25. G. M. Koob, C. G. Lau, “FOUNDATIONS OF DEPENDABLE COMPUTING”, KLUWER ACADEMIC PUBLISHERS
26. K.Kanoun, L.Spainhower, “Dependability Benchmarking for Computer Systems”, IEEE COMPUTER SOCIETY
27. B.Kirwan,"A Guide to PRACTICAL HUMAN RELIABILITY ASSESSMENT”,CRC PRESS
28. 「安全・安心を実現する情報社会基盤の普及に向けて」(http://www.scj.go.jp/ja/info/kohyo/pdf/kohyo-20-t58-4.pdf ）、日本学術会議　情報学委員会セキュリティ・ディペンダビリティ分科会　（委員長　今井 秀樹）、２００８年６月２６日
29. O.-J. Dahl, E. W. Dijkstra, C. A. R. Hoare, Structured Programming, Academic Press, London, 1972 ISBN 0-12-200550-3
30. Birtwistle, G.M. (Graham M.) 1973. SIMULA begin. Philadelphia, Auerbach.
31. Humphrey, Watts (March 1988). "Characterizing the software process: a maturity framework". IEEE Software 5 (2): 73–79. doi:10.1109/52.2014. http://www.sei.cmu.edu/reports/87tr011.pdf.
32. Humphrey, Watts (1989). Managing the Software Process. Addison Wesley. ISBN　0201180952.
33. http://www.sei.cmu.edu/cmmi/
34. Ultra-Large-Scale Systems: The Software Challenge of the Future, http://www.sei.cmu.edu/library/assets/ULS_Book20062.pdf
35. Railtrack. Yellow Book 3. Engineering Safety Management Issue3, Vol. 1, Vol. 2, 2000.
36. City University London　Centre for Software Reliability http://www.city.ac.uk/informatics/school-organisation/centre-for-software-reliability/research
37. D-Case Editor: http://www.dependable-os.net/tech/D-CaseEditor/
38. Daniel P. Siewiorek, Robert S. Swarz, “Reliable Computer Systems: Design and Evaluation”, Third Edition. A K Peters/CRC Press. 1998.
39. A.M. Davis, Just Enough Requirements Management: Where Software Development Meets Marketing, Dorset House. 2005.
40. Object Management Group/Business Process Management Initiative: http://www.bpmn.org/
41. Smalltalk: http://www.smalltalk.org/main/
42. Nancy Leveson: “A new accident model for engineering safer systems”, Safety Science, Volume 42, Issue 4, Pages 237-270, April 2004.
43. Yutaka Matsuno, Jin Nakazawa, Makoto Takeyama, Midori Sugaya, Yutaka Ishikawa, “Towards a Language for Communication among Stakeholders”. PRDC 2010: 93-100

4.3. DEOS プロジェクト主要メンバー

研究総括			
	所　眞理雄	株式会社ソニーコンピュータサイエンス研究所　会長・ファウンダー	
副研究総括			
	村岡　洋一	早稲田大学 理工学術院　教授	
領域アドバイザー			
	岩野　和生	三菱商事株式会社　ビジネスサービス部門　顧問	
	落水 浩一郎	北陸先端科学技術大学院大学　副学長　高信頼組込みシステム教育研究センター特任教授	
	菊野　亨	大阪学院大学 情報学部　教授	
	妹尾　義樹	日本電気株式会社 情報・ナレッジ研究所 技術主幹	
	田中　英彦	情報セキュリティ大学院大学 情報セキュリティ研究科　学長・研究科長・教授	
	松田　晃一	（独）情報処理推進機構 顧問	
	安浦　寛人	九州大学 理事・副学長　大学院システム情報科学研究院　教授・システム LSI 研究センター センター長	
2006 年度採択 研究代表者			研究テーマ
	石川　裕	東京大学 情報基盤センター　センター長・教授	並列・分散型組込みシステムのためのディペンダブル シングルシステムイメージ OS
	佐藤　三久	筑波大学 計算科学研究センター センター長・教授	省電力でディペンダブルな組込み並列システム向け計算プラットフォーム
	徳田　英幸	慶應義塾大学 環境情報学部　教授	マイクロユビキタスノード用ディペンダブル OS
	中島　達夫	早稲田大学 理工学術院　教授	高機能情報家電のためのディペンダブル OS
	前田　俊行	東京大学 大学院情報理工学系研究科　助教	ディペンダブルシステムソフトウェア構築技術に関する研究
2008 年度採択 研究代表者			研究テーマ
	加賀美　聡	（独）産業技術総合研究所デジタルヒューマン工学研究センター　副センター長	実時間並列ディペンダブル OS とその分散ネットワークの研究
	木下　佳樹	（独）産業技術総合研究所　セキュアシステム研究部門　主幹研究員	利用者指向ディペンダビリティの研究
	倉光　君郎	横浜国立大学 大学院工学研究院 准教授	Security Weaver と P スクリプトによる実行中の継続的な安全確保に関する研究
	河野　健二	慶應義塾大学 理工学部　准教授	耐攻撃性を強化した高度にセキュアな OS の創出
研究推進委員			
	浅井　信宏	日本アイ・ビー・エム株式会社 ソフトウェア開発研究所 ディスティングイッシュト・エンジニア	
	大野　毅	横河電機株式会社　IA 技術開発事業部　ネットワークテクノロジ部　　組込基盤課　課長	
	中川　雅通	パナソニック株式会社　システムエンジニアリングセンター　オープンシステムエンジニアリンググループ　グループマネジャー	
	森田　直	ソニー株式会社 CPDG・プロフェッショナル・ソリューション事業本部 Felica 事業部 要素技術開発部　統括部長	
	山浦　一郎	富士ゼロックス株式会社 コントローラ開発本部 コントローラプラットフォーム第 2 開発部　グループ長	
	横山　和俊	株式会社 NTT データ 技術開発本部　課長	
領域運営アドバイザー			
	梶本　一夫	パナソニック株式会社 システムエンジニアリングセンター　所長	
	田中　譲	北海道大学 大学院情報科学研究科　教授	
	鶴保　征城	学校法人 専門学校 HAL 東京　校長	
	戸井　哲也	富士ゼロックス株式会社　執行役員	
DEOS 研究開発センター			
	屋代　眞	（独）科学技術振興機構 DEOS 研究開発センター　センター長	

（各グループ内 氏名 あいうえお順、2006 年度採択研究代表者所属は 2012.3.31 現在、他所属は 2012.10.31 現在）

4.4. 近年の障害事例

	発生日時	障害内容	原因
1	2012 年 8 月 13-15 日	NTT ドコモの携帯で、最大 220 か国・地域で 8 万人の携帯電話での音声通話やインターネット通信がしにくくなった。ローミングサービスの障害。	5 月の時点で、3 月に入れ替えた装置の設定ミスでローミングに関して能力の半分しか使えない状態になっていたことを発見していたが、修正によりロンドン五輪期間中に大規模障害が発生しては問題であるとして、修正を見送ったため。
2	2012 年 8 月 7 日	東京証券取引所で、デリバティブ売買システムのネットワーク機器のハードウェア障害が発生。自動切替えを試みるも不具合により切替え処理が行われず取引が停止した。	本番系である 1 号機にハードウェア障害が生じた場合は待機系である 2 号機に自動切替えされるが、1 号機では内部の部分的ハードウェア障害を正しく検知できなかったため、1 号機、2 号機とも本番系として稼働することとなりスイッチに接続されている装置がどちらに信号を送ればよいのか特定することが不可能となったため。
3	2012 年 8 月 2-3 日	NTT ドコモの携帯で、152 万人の携帯電話が通じにくくなった。	2 台ある装置の 1 台が故障したため。信号経路を迂回すれば対応できるとして修正を見送ったため。
4	2012 年 6 月 20 日	● ファーストサーバを利用していた一部の顧客のサーバ設定情報やデータベースの情報等が消失した。 ● その復旧作業において、情報漏洩が起きた可能性もある。	● 脆弱性対策のための更新プログラムに「ファイル削除コマンドを停止させるための記述漏れ」不具合があったことに加え、検証環境下で確認による防止機能が十分に働かず、意図しないファイル削除が発生したため。 ● 復旧作業の手順やプログラムに不備があり、別顧客の情報が混入するのを防げなかったため。
5	2012 年 2 月 2 日	東京証券取引所で、株式売買システムの情報配信機能で障害が発生し、外部に情報が発信できなかったために、3 時間半にわたって一部銘柄の取引を停止した。	三重化されたサーバの 1 台に障害が発生したが、残り 2 台への自動切り替えに成功したと判断して障害対応を完了したが、実際には切り替えに失敗していたため対応が遅れ、経営陣への報告も行わなかったため。
6	2011 年 6 月 6 日から 2012 年 1 月 25 日まで（5 度に渡る）	NTT ドコモの携帯で、通話やパケット通信がつながりにくい状況が繰り返し発生。関連してメールアドレスが他人のものに置き換わるという事故も発生。	この期間に起こった、携帯端末の位置情報を管理するシステムの障害、中継ルータの故障に伴う機器の切り替えをきっかけにした認証サーバでの輻輳、新たに運用を始めたパケット交換機の設計における信号量の見積もりミスによる動作不安定などが原因と思われる。
7	2011 年 4 月 21 日	Amazon EC2 サービスなどがダウンした。これにともなって Engine Yard、Heroku、など、数多くのサイトがダウンした。	仮装マシンの外付けストレージサービス Amazon EBS でネットワーク設定を誤ったことが原因。
8	2011 年 3 月 15 日から 22 日	● みずほ銀行で夜間バッチ処理やオンラインの停止が起きた。 ● ATM が利用不可能になり、為替処理遅延や二重振り込みなどの問題が発生した。	義援金を受け付ける口座の振り込み件数の上限を大きく設定していなかったため、上限を超える振り込みがあったことを発端として、夜間バッチ処理の異常終了、それに伴う多重のオペレーションミスなどが関連していたためと思われる。
9	2010 年 8 月 10 日から 12 日	ユーザが mixi （日本最大の SNS サイト）にアクセスできなかった。	汎用の分散型メモリーキャッシュシステム memcached のバグ。 memcached デーモンが多数の接続/切断を持っているときに突然終了する事が原因。

10	2009 年 5 月 22 日	NTT ドコモから携帯電話に内蔵されている JavaScript が任意のウェブサイトへの不正アクセスを許可していた。ドコモは販売を停止した。	JavaScript の実装に欠陥があり任意のウェブサイトへの不正アクセスを生じた。Web ブラウザで使用される SOP（Same Origin ポリシー）のセキュリティポリシーの実装に問題がある可能性がありドコモが携帯電話用の仕様で書いていなかったと疑われている。
11	2009 年 2 月 24 日	Google Apps の Gmail ユーザは自分のアカウントにアクセスできなかった。	データセンターでの定期的なメンテナンス中に予想外のサービス中断が発生した。このよう場合、ユーザはメンテナンス作業の準備のために代替データセンターに振り分けられるが、ユーザデータの場所を最適化する新しいソフトウェアに予期せぬ副作用があり Gmail のコード内の潜在的なバグを誘発した。バグはユーザが送信先のデータセンターに振り分けられるとユーザのトラフィックが自動的に障害対応に移行してしまった。これにより複数のダウンストリームの過負荷状態を引き起こし、データセンターを過負荷状態にしたことが原因。
12	2008 年 9 月 14 日	複数の空港の搭乗口の端末が非稼動となりフライトのキャンセルを発生させた。	ターミナルからサーバシステムへのアクセスを承認する証明書が 9 月 14 日の早朝に期限切れをむかえたことが原因。
13	2008 年 7 月 22 日	デリバティブ取引システムから情報の一部がユーザに配信できなかった。	一銘柄あたりのワーキングメモリーを予想されたサイズよりもはるかに小さく定義していた。これにより複数の銘柄に損失をもたらした。
14	2007 年 10 月 12 日	東京近郊の鉄道 IC カード用チケットゲートが機能しなくなった。	サーバから改札側に本質的な情報を送信する間の巨大データを小さなデータの塊に分割するロジックに原始的なエラーがあった。改札側にデータ受信の無限ループを発生させた。
15	2007 年 5 月 27 日	ANA の搭乗手続きシステムが停止した。130 便がキャンセル、306 便に遅延が生じた。	ハードウェア障害によって引き起こされるネットワーク機器のトラブルがホストコンピュータと端末間の輻輳をもたらした。ネットワーク機器のトラブルのイベントと輻輳の関係が知られていなかったため見過ごされていた。
16	2006 年 9 月 19 日、2006 年 10 月 23 日、2007 年 5 月 16 日、2007 年 5 月 23 日	● IP 電話の接続が困難になった。 ● NTT 西と NTT 東の接続が故障し不通となった。 ● フレッツが 7 時間非接続となった。	● キャパシティを越えた。シグナリングサーバ内のバグが原因でシグナリング処理のオーバーフローが発生した。 ● キャパシティプランニングの推定ミスが原因で信号処理のオーバーフローが発生した。 ● 不良データが保守後に復元されてしまい、これがシグナリングサーバを停止させた。 ● ひとつのルータ障害が他のルータへの不正なルーティング情報の伝播を引き起こした。しかし、ルータを再起動することが解決だという確信が持てなかった。
17	2003 年 3 月 1 日	航空計画データ処理システムがダウンした。215 便がキャンセル、1500 便に遅延が生じた。	原因は一つのバグにより生じた。このバグは特定のメモリーをアドレスすると発生するがテストが十分に行われなかったため発見できなかった。

4.5. 開放系障害要因表

要因の分類	要因	要因例	システムの症状例
＜不完全さ＞ 要求に対して仕様が完全でなく、また、仕様に対して実装が完全でなく、出荷時や運用時のシステムの振る舞いを完全に把握する事が困難である事	★システムの完全把握破綻 ・システムの巨大化 ・システムの複雑化 ・システムのネットワーク化 ・オープンソフトウェア多用 ・ブラックボックス多用 ・レガシーコード依存 ★構成要素変容 ★構成変容・インテグレーション問題	◆システムが多くのソフトウェアの組合せから作られており、巨大化、複雑化に伴い網羅的な仕様記述やテストが不可能 ◆要求・仕様・設計・実装・テストなどの各開発フェーズにおける理解の違い、文書の誤りなどによる仕様ミスや漏れ、設計ミスや漏れ、実装ミスや漏れ、テストミスや漏れ ◆管理、運用、保守におけるモジュールの更新、機能変更、タイムアウト値の変更、修正ミス、ライセンスの期限切れ ◆オープンソフトウェア、ブラックボックス、レガシーコードの仕様と動作の不一致、開発者の理解不足 ◆モジュール実行の優先度、順番、タイミングなどの見極め不足 ◆設計外、テスト外の構成要素の取り込み（運用時のダウンロード等）	●要求書にない動作をする ●機能仕様書にない動作をする ●テスト仕様書にない動作をする ●モジュールごとの機能テストはパスしたのに、組み上げてみたら期待通りに動かなかった ●運用中にモジュールの更新がなされ、期待通りに動かなくなった ●ある日突然動かなくなった
＜不確実さ＞ 利用者の要求や使用環境がライフサイクルを通して変わり、設計時や運用時に機能や挙動を完全に予測できない事	★期待値・能力変容 ★動作環境変容 ★構成変容・インテグレーション問題 ★システム性能バランス問題 ・パーフォマンス設計 ・キャパシティ計画 ★ネットワーク化	◆利用者の要求の変化、システムへの期待値の変化、利用者やオペレーターの操作能力や習熟度の変化 ◆出荷数・使用者数・アクセス数の増加、稼働経済性の変化による使われ方の変容 ◆現場での（人手を介して、ネットワークを介して）構成要素の機能修正やサービス変更、システムの再構成（複雑性の増加） ◆システムリソースの変容（老化、メモリー制限、クロック制限等） ◆ネットワークの向こう側の変容（レスポンスの変容、仕様の変容） ◆ネットワークを介した環境による想定外の接続・インタラクションの増加、外部からの意図的な攻撃を受ける事	●処理スピードが遅くなった ●端末の前で順番待ちをするユーザーが増えた ●いつも通りの処理要求を出したのに、「しばらくお待ちください」のプロンプトが出た ●新しいサービスがこれまでの電子マネーでは受けられなかった ●義捐金を振り込もうとしたが受け付けられなかった ●２４時間サービスに変わったと言われたのに、深夜に使おうとしたら動かなかった ●クレジットカード番号が不正使用された ●ある日突然動かなくなった

4.6. DEOS プロジェクト用語集

【あ行】
安全性（Security）システムが外部から意図的に攻撃されても防御できる事
逸脱した要求（Deviated Requirements）ステークホルダ合意から逸脱した要求の状態
運用状態（In-Operation）ステークホルダ合意（サービスレベル変動許容範囲）内でシステムを運用しサービスを継続している状態
エビデンス（Evidence）議論において主張を支える情報
オープンシステム（Open Systems）開放系。外部との相互作用があり、機能、構造、境界およびそれらの関係が時と共に変化する系
オープンシステムディペンダビリティ（Open Systems Dependability）開放系に対するディペンダビリティ
詳しくは２章参照。

【か行】
開放系障害（Open Systems Failure）不完全さと不確実さに起因するオープンシステムの障害
仮想化技術（Virtualization Technology）計算機資源を抽象化し、論理的に分割する技術
可用性（Availability）システムが稼働し続ける事
クローズドシステム（Closed Systems）閉鎖系。外部との相互作用が限定的で、構成要素やそれらの関係が変化しない系
形式的検証（Formal Verification）数理的技法に基づいてプログラムの性質を厳密に証明すること
原因究明（Cause Analysis）エビデンスを基に障害原因、あるいは原因のありそうな範囲を特定すること
合意された要求（Agreed Requirements）ステークホルダにより合意された要求の状態
コンソーシアム（Consortium）利害関係者が集まって活動をする事により目的を達しようとする団体

【さ行】
システムアーキテクチャ（System Architecture）システムの設計思想、基本機能ならびに基本構造
実現された要求（Realized Requirements）実システムに組み入れられ実装されたシステム要求
仕様記述言語　（Specification Description Language）プログラムの満たすべき性質を記述するための言語
障害（Failure）運用においてサービスレベルに関するステークホルダ合意からの逸脱
障害対応サイクル（Failure Response Cycle）障害に対応するサイクル
迅速対応（Responsive Action）障害や異常状態に迅速かつ適切に対応する事
信頼性（Reliability）システムが期待期間、期待された機能を実行しつづける事
ステークホルダ（Stakeholders）システムやサービスに権利・義務・関心を持つ個人あるいは組織
利害関係者。詳しくは３章参照。
ステークホルダ合意（Stakeholders' Agreement）ステークホルダ間のサービスレベル変動許容範囲と障害対応方法についての利害関係合意
説明責任遂行（Accountability Achievement）障害発生時には現状、原因、回復見込みなど、サービス変更時にはサービス開始時期や条件を明らかにして利用者に説明する事

【た行】
抽出された要求(Elicited Requirements）ステークホルダのニーズから抽出され同定された要求の状態
通常運用（Ordinary Operation）適切な定期点検、予防保守、継続的な改善活動による不具合再発防止などを含む日常運用
通常運用時要求（Ordinarily Operated Requirements）運用時における要求の状態
ディペンダビリティ（Dependability）信頼性や安全性など、システムの提供するサービスを安心して継続的に利用できる性質、その能力
DEOS アーキテクチャ（DEOS Architecture）分野・アプリケーション別に DEOS プロセスを支援する。合意記述データベース、開発支援ツール群、DR-E からなる
DEOS プロセス（DEOS Process）OSD を実現するための「変化対応サイクル」と「障害対応サイクル」からなる反復的プロセス
D ケース（D-Case）DEOS における合意形成のための方法並びにツール

D スクリプト（D-Script）DEOS における動的対応スクリプト

【は行】
不確実さ（Uncertainty）システムが変容し、振る舞いが事前に予測できない事
不完全さ（Incompleteness）システムが完全に要求を実現しない状態
不具合（Incident）不都合な事象
ブラックボックス（Black Box）内部構造を見ることができず、外部仕様のみから機能が活用される製品
プロセス（Process）システムやサービスを開発し運用するためのステップ（フェーズ）の連続体
変化対応サイクル（Change Accommodation Cycle）ステークホルダの目的変化あるいは環境変化に対応するサイクル
変動許容範囲（In-Operation Range）ステークホルダ間で合意されたサービスレベルの変動許容範囲
保守性（Serviceability、Maintainability）システム保守のやりやすさ
保全性（Integrity）処理されるデータの整合性が保たれる事

【ま行】
マネージ（Manage）問題を努力して解決し、システムの状態をより好ましい方向に持って行くこと
未然回避（Failure Prevention）システム異常や障害の予兆を検知して、障害発生を回避する事
メトリクス（Metrics）定性的、定量的に対象を評価する指標

【や行】
要求状態管理モデル（Requirements States Management Model）要求状態を管理するモデル
要求抽出（Requirements Elicitation）ステークホルダ間の目的や要望を合意形成を経て要求として特定する事
要求マネジメント（Requirements Management）ステークホルダの合意に従って要求を管理する方法
要素技術（Elemental Technology）個々の要求された機能を実現する技術

【ら行】
レガシーソフト（Legacy Software）製作者・保守者がいなくなっても使われ続けているソフトウェア

4.7. DEOS プロジェクト略語集

ADD（Agreement Description Database）
DEOS（Dependable Embedded Operating Systems / Dependability Engineering for Open Systems）
D-DST（DEOS Development Support Tools）
D-RE（DEOS Runtime Environment）
OSD（Open Systems Dependability）

4.8. 世界の関連標準、関連活動団体

標準

- IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems
  http://www.iec.ch/zone/fsafety/fsafety_entry.htm
- IEC 60300-1: Dependability management
  http://www.iec.ch/cgi-bin/procgi.pl/www/iecwww.p?wwwlang=E&wwwprog=sea22.p&search=text&searchfor=IEC+60300-1&submit=OK
- IEC 60300-2: Dependability Program Elements and Tasks
  http://www.iec.ch/cgi-bin/procgi.pl/www/iecwww.p?wwwlang=E&wwwprog=sea22.p&search=text&searchfor=IEC+60300-2&submit=OK
- ISO/IEC 12207: Systems and software engineering - Software life cycle processes
  http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=21208

- ISO/IEC 15026: Systems and software engineering - Systems and software assurance http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=62526
- ISO/IEC 15288: Systems and software engineering - System life cycle processes http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=43564
- ISO 26262: Road vehicles – Functional safety http://www.iso.org/iso/catalogue_detail.htm?csnumber=43464
- IEC 61713: Software dependability through the software life-cycle processes - Application guide http://www.iec.ch/cgi-bin/procgi.pl/www/iecwww.p?wwwlang=E&wwwprog=sea22.p&search=text&searchfor=IEC+61713&submit=OK
- IEC 62347: Guidance on system dependability specifications http://www.iec.ch/cgi-bin/procgi.pl/www/iecwww.p?wwwlang=E&wwwprog=sea22.p&search=text&searchfor=IEC+62347&submit=OK

プロセスガイド・フレームワーク

- CMMI: Capability Maturity Model® Integration http://www.sei.cmu.edu/cmmi/
- DO-178B: Software Considerations in Airborne Systems and Equipment Certification http://www.rtca.org/
- MISRA-C: http://www.misra-c.com/
- TOGAF:The Open Group Architecture Framework http://www.opengroup.org/togaf/

ソフトウェア

- SELinux: Security-Enhanced Linux http://www.nsa.gov/research/selinux/index.shtml
- AppArmor®: a Linux application security framework http://www.novell.com/linux/security/apparmor//
- Xen® hypervisor: the powerful open source industry standard for virtualization http://www.xen.org/

関連団体・プロジェクト

- ISO: International Organization for Standardization http://www.iso.org/iso/home.htm
- IEC: International Electrotechnical Commission http://www.iec.ch/
- ISO/IEC JTC1: Joint ISO/IEC Technical Committee 1 http://www.iso.org/iso/standards_development/technical_committees/list_of_iso_technical_committees/iso_technical_committee.htm?commid=45020
- IEC/TC56: Technical Committee 56: IEC Technical Committee for International Standards in the field of Dependability http://tc56.iec.ch/index-tc56.html
- OpenTC Consortium: Open Trusted Computing Consortium http://www.opentc.net/
- Linux-HA Project: High Availability Linux Project http://linux-ha.org/
- Carrier Grade Linux Workgroup http://www.linuxfoundation.org/en/Carrier_Grade_Linux
- TCG: Trusted Computing Group http://www.trustedcomputinggroup.org/home
- ERTOS Group: Embedded Real-Time Operating-Systems Group http://ertos.nicta.com.au/
- ARTEMIS: Advanced Research & Technology for EMbedded Intelligence and Systems http://www.artemis.eu/
- CPS Program: Cyber-Physical Systems Program http://www.nsf.gov/pubs/2008/nsf08611/nsf08611.htm
- MISRA: Motor Industry Software Reliability Association http://www.misra.org.uk/
- AUTOSAR: AUTomotive Open System ARchitecture http://www.autosar.org/
- JasPar: Japan Automotive Software Platform and Architecture http://www.jaspar.jp/
- FlexRay Consortium: Consortium for the communications system for advanced automotive control applications http://www.flexray.com/
- The Open Group http://www3.opengroup.org/
- CoBIT: Control Objectives for Information and related Technology http://www.isaca.org/Knowledge-Center/COBIT/Pages/Overview.aspx
- ITIL: Information Technology Infrastructure Library http://www.itil.org/en/vomkennen/itil/index.php
- OMG: Object Management Group http://www.omg.org/


Dependable Embedded Operating Systems
Dependability Engineering for Open Systems
DEOS

DEOS プロジェクト