DEOS-FY2009-WP-01J

JST-CREST

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
研究領域

― DEOS プロジェクト ―

DEOS: Dependable Embedded Operating System

White Paper

Version 1.0

2009/09/01

DEOS プロジェクト
研究総括　所 眞理雄

目次

1. 背景

モバイル情報端末、オフィス情報機器、情報家電、車載情報機器等の組込みシステムはもはや単独で用いられるものでなく、ネットワークにつながれて大規模システムの一要素としてそれを支える存在となっている。システムサービスがネットワークを介してこれらの組込みシステムに提供され、ユビキタス社会における生活の利便性や快適性をもたらし、世界中の人々の暮らしや社会に役立っている。これらの組込みシステムの多くは、利用者の多種多様なニーズに対応するために高度化、複雑化、巨大化して来ている。また、他の開発者が作ったソフトウェアをブラックボックス（部品）として使うことがしばしば行われ、更に、システム運用中の仕様変更を含む保守・管理も行われるため、あるシステムの開発者・運用者がシステムの隅々まで理解することが不可能になって来ており、システムの信頼性、可用性を担保することが困難になってきている。加えて、ネットワーク上で通信しあう他のシステムの変更などの外部要因の変化や、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問題も発生し、データの保全性や利用者の安全・安心を脅かす諸問題が世界的規模で急激に顕在化・増大化している。これらの脅威に適切に対応し、利用者が安全かつ安心してサービスや製品を継続して利用できるようにすることはサービスや製品を提供する側の重要な責任となって来ている。こう言った中で、システムのディペンダビリティを維持し、さらに向上させるには、これまで主にクローズド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となって来ており、新たな考え方に基づいたディペンダブルシステムの構築が求められている［1、4、5、6、12］。

2. ディペンダビリティ

2.1. 考え方の変遷

1960 年代後半以降、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ー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、故障しにくい特性（信頼性 Reliability）、高い稼働率を維持する特性（可用性 Availability）、障害が発生した場合に迅速に復旧できる特性（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特性（保全性 Integrity）、機密性が高く不正にアクセスされにくい特性（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出来る限り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］。

信頼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。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。このようにして設計・開発・製造されたシステムに対し、運転モードによって低需要運転モードまたは高需要/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 が審議されている。

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 と合同で「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］。その論文ではディペンダビリティとセキュリティを図 1 のように整理している。しかしながら、現代の複雑なシステムの問題を解決するためにはこのような要素に分解してそれぞれに対処するだけでは不十分であると考えられる。

図 1 ディペンダビリティとセキュリティ

2.2. オープンシステムディペンダビリティ

組込み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。組込みシステムのソフトウェアは要求仕様に基づいてアーキテクチャが決定され、設計、実装がなされる。近年、開発期間を短縮し、開発コストを下げるため、既存のあるいは他社から供給されるありものの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、システム運用中に機能向上や機能変更のための仕様変更が行われ、ネットワークを介して、修正がダウンロードされたり、新しい機能が追加される。このような理由から、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 2）。

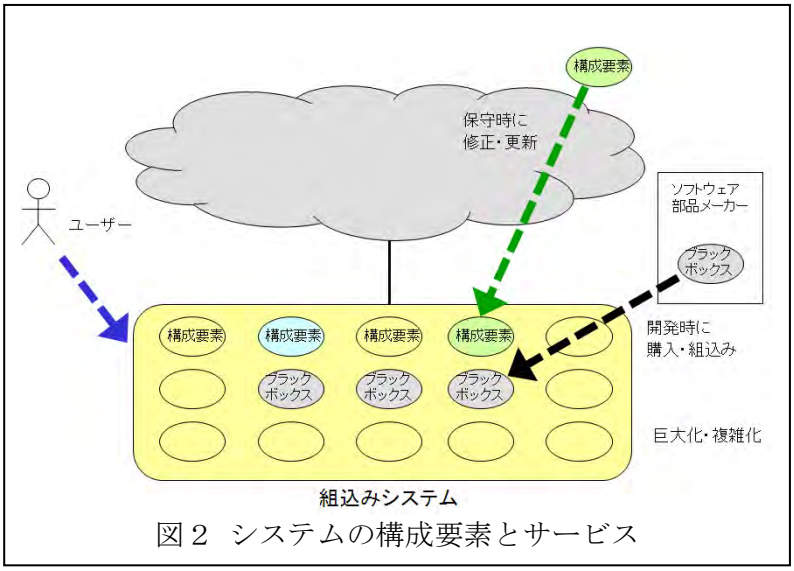

図 2 システムの構成要素とサービス

現代の組込みシステムの多くは、システム同士がネットワークでつながれて使われる。その場合、ユーザーはネットワークを介して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。ひとつのサービスドメインは様々なレベルで他のサービスドメインともつながり相互作用を行う。他のサービスドメインにおけるサービスやインタフェース仕様なども変更される可能性があり、時には終了される場合もある。このように、システムあるいはサービスドメインの境界も不明確となる。加えて、サービスを提供する設計者や開発者、サービスを運用するオペレーター、そしてサービスを享受するユーザーが意図せぬ間違いを冒す可能性を排除できない。また、ネ

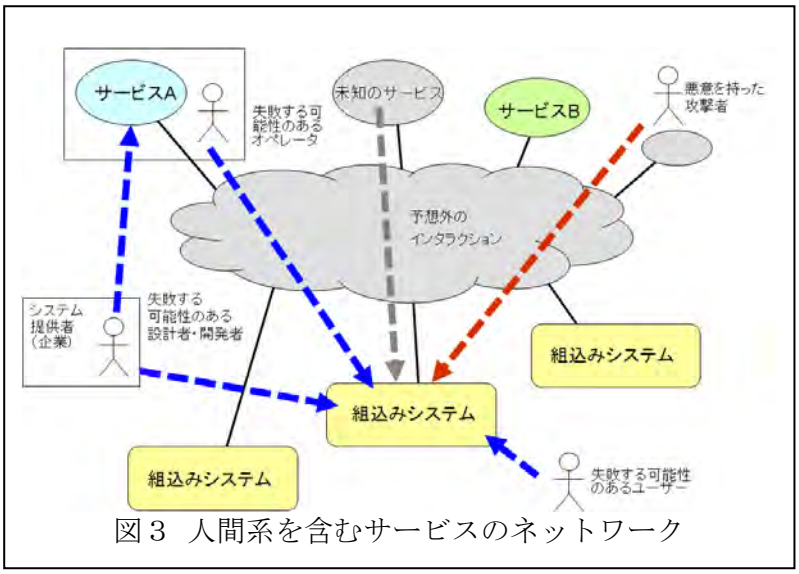

図 3 人間系を含むサービスのネットワーク

ットワークが広がるにつれて、未知のサービスと予想外のインタラクションを行ったり、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３）。

これからのシステムは以上のような問題に対応できなければならない。これらの問題に対応することは、時間的な経過ならびに空間的な変化をもシステムの要件として考えるということであり、これは開放系（すなわちオープンシステム）を対象としたディペンダビリティに他ならない。これまでのディペンダビリティは閉鎖系（クローズドシステム）すなわちシステムの性能を時間的・空間的に固定してとらえたディペンダビリティであった。オープンシステムに対応するディペンダビリティの論点を整理すると以下のようになる。

（１） 要求に対して仕様が完全でなく、出荷時のシステムの振る舞いの完全把握、完全保証が難しい（図４）
＜仕様/実装の不完全さ、システムの完全理解の困難さ＞

- 設計・開発フェーズにおける要求者の期待値、要求事項・レベルの時間的変容
- 構成要素の論理的不透明さ － 複雑化、巨大化、ブラックボックス化、レイヤー化、レガシーコードの聖域化 、ネットワークを介して利用する構成要素の仕様やサービス仕様の不完全さ
- 要求の把握の不完全さ、仕様の不完全さ、設計の不完全さ、実装の不完全さ、テスト仕様の不完全さ、テスト実施の不完全さ

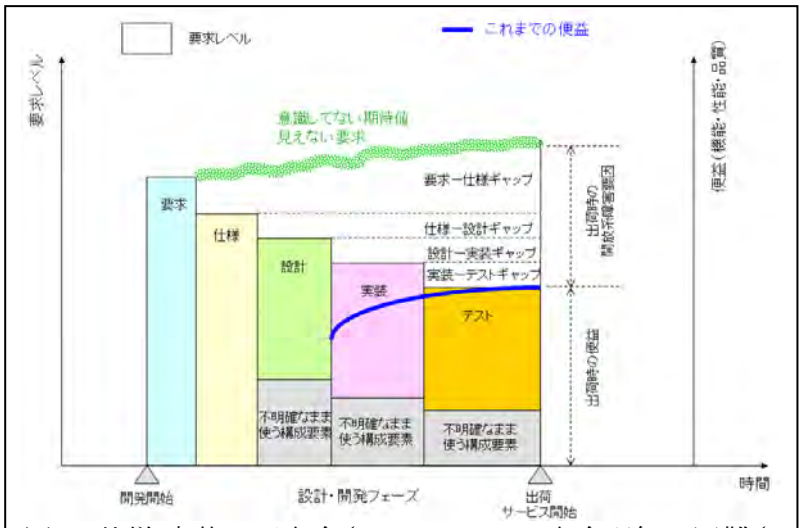

図４ 仕様/実装の不完全さ、システムの完全理解の困難さ

（２） 使用環境や構成がライフサイクルを通して変わり、設計時に機能や挙動を完全に予測できない（図５）
＜使用状況の変化に伴う不確実さ、システムの挙動予測の困難さ＞

- 保守・運用フェーズにおける使用者の期待値・能力の時間的変容 － 要求事項・レベル、操作能力、習熟度・慣れ・不注意
- 出荷数・使用者数の量的拡大による想定外の使われ方・構成要素の構成管理の複雑化
- ネットワークを介しての構成要素の機能修正や機能変容、システムの再構成
- ネットワークを介して利用する構成要素の仕様やサービス仕様の修正・変更、想定外の接続・インタラクション、外部からの意図的な攻撃

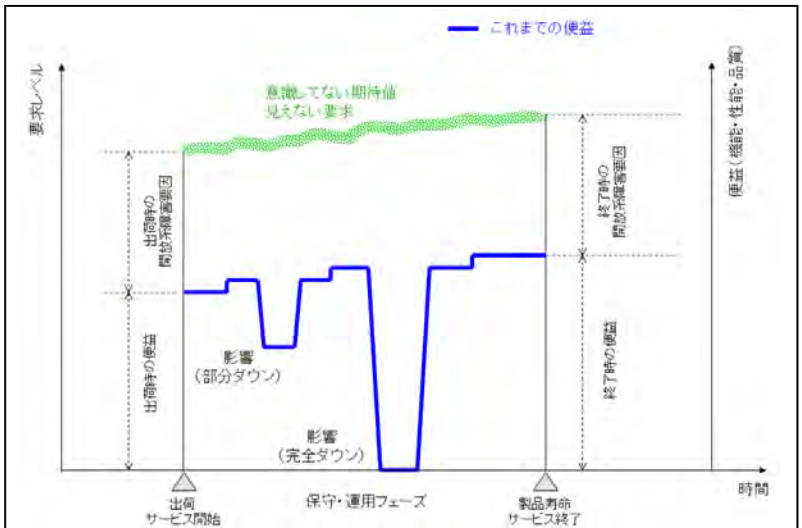

図５ 使用状況の不確実さ、システムの挙動予測の困難さ

すなわち、これからの組込みシステムはオープンシステムの特徴である**不完全さ**と**不確実さ**に対応する必要がある。しかしながら、正に対象がオープンシステムであるがゆえに、それらに完全に対応することはできない。すなわち、ユーザーの要求の変化によるシステムの変更の必要性は避けられない。外部からの攻撃に対して完全に対応できるシステムを最初から想定して作ることはできない。我々は起こ

りうるすべてのことを事前に把握して完全なシステムを作ることは出来ない。したがって事故は必ずしも避けられない。

我々がおかれたこのような状況を理解して、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］
- 「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］

等がある。

我々は、現代の多くの組込み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**として次のように定義する。

「組込みシステムは不完全さと不確実さに起因し、未来に障害となりうる要因（開放系障害要因）を宿命的に抱えている。それらの要因を顕在化する前に出来る限り取り除き、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出来る限り安全にかつ継続的に提供できること」

オープンシステムディペンダビリティを達成するために我々が行う事は、

- 継続的な努力により、サービスを安全且つ継続的に利用者に提供する事。即ち、事故の発生を最小限にし、被害を最小にし、短時間で復旧させ、同様な原因による事故が２度と起こらないようにする事
- システムの開発・設計、保守・運用、変更・復旧などのライフサイクルそのものや、そのために取った最大の努力をいつでも見えるようにし、説明可能とする事（システム提供者は、ニーズの変化に対応し、外部接続機器の変化に対応し、内部部品・モジュール変更への対応、利用者の視点での説明責任を果たすことが重要である）

である。

「オープンシステムディペンダビリティ」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を否定するものではない。これまでは、主に偶発的フォールトや意図的フォールトに焦点を当て、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ている。われわれは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（主に出荷までに開発現場で対応）と、開放系障害による影響の最小化（主に出荷後に稼働現場で対応）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「オープンシステムディペンダビリティ」は「これまでのディペンダビリティ」をさらに補完し、強化するものである。

2.3. オープンシステムディペンダビリティ実現の仕組み

オープンシステムディペンダビリティ実現のためには、要素技術、マネジメントプロセス、システムアーキテクチャの３要素を有機的に結び付ける必要がある［13］。要素技術はシステム安全構築技術、設計開発支援技術、保守運用支援技術などからなり、具体的にはカーネル拡張基盤技術、仮想モニター技術、マルチコア技術、実時間・実行時間予測技術、仕様記述技術、プログラムの正当性検証技術などを含む。マネジメントプロセスは対象システムの開発から運用に亘るすべてのプロセスにおいて、継続的な改善・改良を行うための技術であり、また、説明責任の確保のために必要な規格化、標準化の作業を含む。システムアーキテクチャは要素技術を有効に利用し、マネジメントプロセスを支援し、継続的な進化を可能とするシステムを具体的に実現するための設計仕様書である。

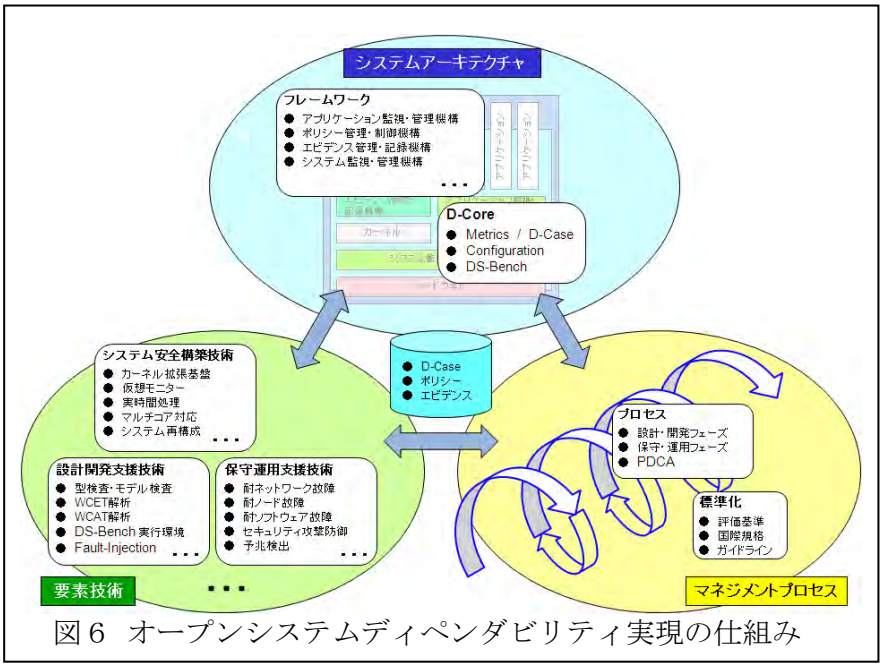

図６ オープンシステムディペンダビリティ実現の仕組み

当プロジェクトではこの有機的な仕組みを考え、研究し、開発し、広くその成果を世の中の組込みシステム提供者に提供し、利用されることを目指す（図６）。

3. プロジェクトの方針

3.1. プロジェクトの目的

本プロジェクト「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」では組込みシステムにおける信頼性、安全性、使いやすさなどの要件を「オープンシステムディペンダビリティ」と言う観点から捉えなおし、実利用を目指した組込みシステムのためのディペンダブル OS およびそのシステムの実現に必要な概念、アーキテクチャ、仕様書・実装ガイドライン、マネジメントプロセス、フレームワーク・開発環境・ツールなどの関連基盤技術も含めて開発し、評価基準を定め、見える化し、標準化する。ここで OS を狭義のオペレーティングシステムと限定するのではなく、「オープンシステムディペンダビリティ」の定義からも推察されるように、もっと広いものと考え、システムのアプリケーションを支えるすべてのシステムソフトウェア層と捉える。また、組込みシステムも狭義には捉えず、ネットワークシステムに接続され、例えば交通情報制御システムや鉄道改札システムなどの社会インフラとして使われるシステムも対象に含める。

当面我々が対象とするアプリケーション領域として、監視システム、生産制御システム、通信制御システム、オフィス機器、車載情報機器、ロボット、情報家電、携帯電話、モバイル情報端末などを考えるが、実際のユーザーの状況に応じて分野を絞り込んでいくことも考える。

3.2. プロジェクトの目標

目標は下記を考える。

1　２１世紀にふさわしい明確なディペンダビリティについての概念を確立する。
　1.a　第２章で考察したオープンシステムディペンダビリティを検証し、精緻化する。
　1.b　システムのライフサイクルにわたってオープンシステムディペンダビリティを向上させる要素技術、マネジメントプロセス、システムアーキテクチャを開発する。
2　実用化を推進する。実用に耐えうるディペンダブル組込みシステムの仕様書・実装ガイドラインを作成し、フレームワーク・開発環境・ツールを開発し、マネジメントプロセスを確立し、リファレンスシステムのデモを行うことにより実証する。
3　上記１、２を基本に、ディペンダビリティの評価基準を定め、見える化、標準化を推進する。（期待としては、国際標準にする。）
4　上記１～３を維持・保守・改良するためのコンソーシアム/ユーザー団体を立ち上げる。

ディペンダブルなシステム構築に必要な要素技術、マネジメントプロセス、システムアーキテクチャなどは、実用的なシステムの開発や向上サイクルを実践することにより評価・改善されていくと考えている。そのためにはプロジェクトの成果を企業に使ってもらい、その結果をフィードバックしていく必要がある。将来的には障害情報の共有、保障や説明責任などを含む社会的な責務の遂行、それらを支えるためのオープンな仕組みなどが必要になる。

3.3.プロジェクトの成果

本プロジェクトの成果としては以下のものを考える。

- オープンシステムディペンダビリティコンセプト
- 要素技術（各要素技術仕様書、API 定義書、コード、実装ガイドライン等）
- マネジメントプロセス（プロセスガイドライン等）
- フレームワーク・開発環境・ツール（システムアーキテクチャ仕様書、API 定義書、コード、実装ガイドライン等）
- リファレンスシステム（仕様書、コード）
- メトリクス、規格・標準・ガイドライン、記録フォーマット
- オープンシステムディペンダビリティコミュニティ
- オープンシステムディペンダビリティコンソーシアム

4. 研究・開発すべき項目

4.1.求められる能力・技術

「オープンシステムディペンダビリティ」を実現・向上させるには開放系障害をマネージする能力あるいは技術が求められる。ひとつは要因を最小化する能力・技術であり、もう一つは影響を最小化する能力・技術である。

◆　開放系障害マネージ能力

- 開放系障害を起こす要因の最小化　（主に出荷までに開発現場で対応する）　（図７）

1） 要求、仕様、設計、実装、テストの各フェーズ毎の不完全さ(ギャップ)を見える化し、最小化する
2） 出来る限り構成要素の動作を解析し、振舞いを検証する
3） 出来る限り動作状況を記録し、製品・システム提供者の説明責任マネジメントを支援する
4） 国際標準・規格に適合していることを検証する

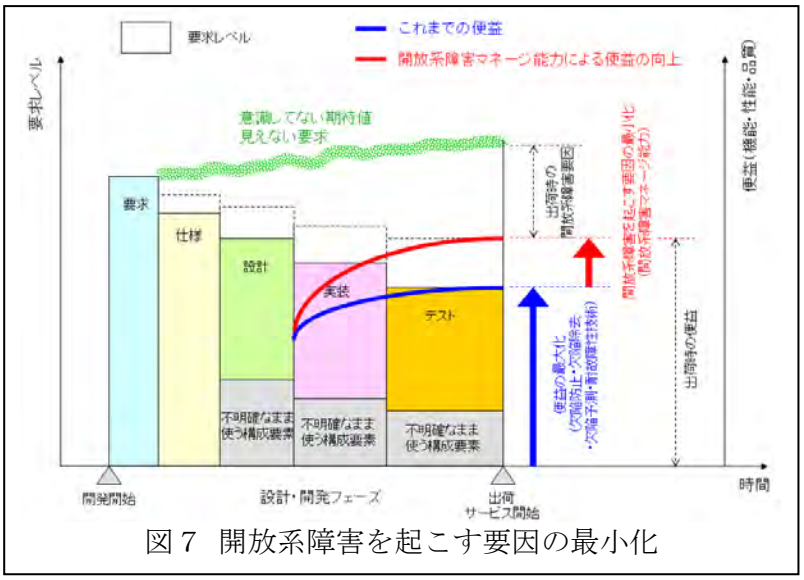

図７ 開放系障害を起こす要因の最小化

- 開放系障害による影響の最小化 （主に出荷後に稼働現場で対応する） （図８）

1） 実環境・実時間で仮稼働させて影響の有無を確認する
2） 稼働中に障害の予兆を検出する
3） 障害が起きてしまった後、その影響を最小とする、また、迅速な復旧を支援する
4） 出来る限り動作状況を記録し、製品・システム提供者の説明責任マネジメントを支援する

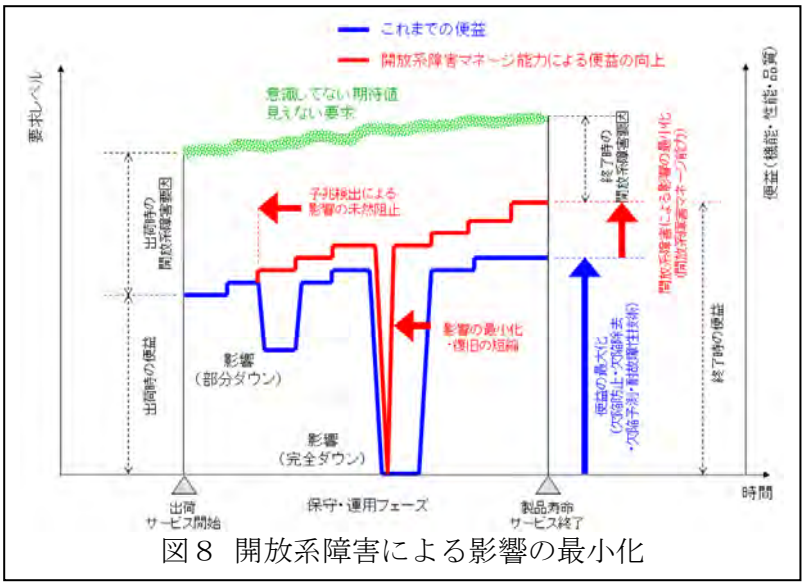

図８ 開放系障害による影響の最小化

4.2. 要素技術

要求機能を実現するための要素技術としては、次のようなものが必要である。

① 要求を仕様として確実に記述する技術、検証する技術
② 仕様を設計に確実に反映する技術、検証する技術
③ 設計を忠実に実装する技術、検証する技術
④ 実装されたものを仕様に照らし合わせて正確にテストする技術
⑤ シミュレーター上で検証する技術
⑥ わざと欠陥を挿入して異常事態を起こし、システムの限界値を測定する技術
⑦ システムの正常な振る舞いや、世の中の標準値と比較検討する技術
⑧ ディペンダビリティ評価指標を測定する技術、評価基準の妥当性を検証する技術
⑨ システムに起きるさまざまな事象を監視・記録する技術
⑩ 起きた事象を分析し、検証する技術
⑪ ポリシーの記述・管理技術
⑫ 予兆検出の技術
⑬ 原因を特定・抽出する技術
⑭ 構成を変える技術、障害部位を切り離す技術、構成を元に戻す技術

⑮ システムの資源を制限したり、動きを制御する技術
⑯ ネットワークにつながった時に他のノードからの攻撃を防ぐ技術
⑰ システムが間違いなく動作することを保証する技術
⑱ 整合の取れた時間体系を有し、改竄されない堅固な記録機構
⑲ システム時間を変えて、未来時間で試験できる技術
⑳ 時間経過を加速してシステムライフサイクルを試験する技術

実用化のためには、それぞれの要素技術の「仕様書」、「API 定義書」、「コード」、「実装ガイドライン」、「性能試験結果」が求められる。

4.3. マネジメントプロセス

これまで、多くの組込みシステムの開発では、事前にしっかりとした開発計画を立て、対象製品・システムの仕様を詳細に書きあげて長期間の設計・実装・テストフェーズを回すと言ったプロセスが多かった。そして製品・システムの便益（機能・性能・品質）を改良・改善するために PDCA サイクルを回す事が実践されて来た。このプロセスは、ネットワークを介することなく個別に利用され、仕様が開発開始時に相当程度見極められるような製品・システム（クローズドなシステム）の開発には有効であった。

しかし、今後の組込みシステムは前述のように、事前に完璧な仕様を書くことができず、開発計画も完全に見通せず、しかもネットワークへの接続が想定される。このようなシステムに対しては、見極められた範囲の仕様で製品・システムを開発して、ライフサイクルを通じてフィードバックを受け、仕様に反映させるサイクルを何度も回して行くマネジメントプロセスが重要であり、それを可能にする要素技術やシステムアーキテクチャが必要になる［18］。こう言う状況では通常の PDCA サイクルより、小さく、柔軟で、回転の速いサイクルを用いる方がよりふさわしい。正しく対応するためには、動作状況のより注意深い観察・監視と、起きる事象のより深い分析が重要と考えられる。

実用化のためには、プロセスを実行するための「プロセスガイドライン」、さらには「実施事例」などが求められる。また、万が一事故が起こったときのための説明責任の観点から、後述するように、マネジメントプロセスの規格化、標準化も重要である。

4.4. システムアーキテクチャ

要素技術とマネジメントプロセスを有機的に結び付けるためにはシステムアーキテクチャが重要な役目を担っている。システムアーキテクチャには以下のような仕組みを提供することが求められている。

- 要素技術を実行させる仕組み・構造
- マネジメントプロセス（企画・実施・監視・分析サイクル）を支援する仕組み・構造
- システムの進化を支援し、生きたままの変化を可能とする仕組み・構造

当プロジェクトではこの仕組みを「**フレームワーク**」および「**リファレンスシステム**」として実装し提供する。 その他、多くのシステム設計者・提供者に使ってもらうために、「仕様書」、「API定義書」、「実装ガイドライン」も併せて提供する。またディペンダビリティの要求項目を、システム提供者とシステム開発者とが設計・開発フェーズのみならず保守・運用フェーズを通して合意するための枠組み（**D-Core**）を提案する。これは要求の木構造記述方式（D-Case）とメトリクス、コンフィギュレーション、測定・評価・検証ツールから構成される。

どういったレベルで、どのような機能・サービスを提供させるかは、システム使用者とシステム提供者の間のビジネス合意で決められる。例えば、システムがエラーを起こした時、警告をオペレーターに

知らせて停止するのか、あるいはエラーを分析して自己修復を試みるのかなどを前もって決めておく必要がある。それをシステム動作のポリシーとして、正しく記述し、正しくシステムに与え、正しく実行させる仕組みが必要である。

対象製品・システムの構成（コンフィギュレーション）は、システムがどのようなサービスや機能を提供し、どの程度の規模で使われ、どの程度のディペンダビリティを維持しなくてはいけないのか等の要求により、大いに変わってしまう。また、強力なハードウェアアシスト機能がある CPU を使うのか使わないのか、あるいは VM（Virtual Machine）を実装するのかしないのか、などによって維持できるディペンダビリティのレベルが変わる。必要とされるレベルのディペンダビリティを確保するためのメトリクス（後述）ならびに設計・実装する場合に参考になるガイドラインを提供する。

4.5. メトリクス、測定・評価・検証ツール

対象システムが、「オープンシステムディペンダビリティ」をいかに実現しているのか、あるいはどの程度のレベルで実装されているのか、などを評価するためには、万人が認めるわかりやすい評価指標（メトリクス）が必要である。　その上でこのメトリクスがどの程度実践されているかを測定し、評価し、検証する仕組みとツールが必要となる。

4.6. 規格・標準・ガイドライン

機能安全や品質確保のためにはすでに IEC 61508、ISO 9000、CMMI など様々な規格・標準やガイドラインが設定されているが、これらは主に「これまでのディペンダビリティ」に対するものと考えられる。すでに述べたように、当プロジェクトでは「オープンシステムディペンダビリティ」を打ち立てるのが目標であるので、その考え方を規定する規格・標準や記録のフォーマットを定義し、その実践のためのガイドラインを提供し、国際標準化などを通じて全世界の関係者に広く共通に使ってもらうことを目指す。

企業活動において製品やシステムに関連して事故・故障（Incidence）が起こった場合、誰がユーザーや社会に対して説明し、どのように責任を取るかが、ますます重要になってきている。企業が説明責任を果たすにあたって、どう作ったかよりも、ユーザーからどう見えるかが重要になる。ユーザーの視点での説明責任を全うできるかどうかが重要なディペンダビリティの評価基準のひとつになる。

5. 研究開発体制

当プロジェクトは 2006 年に 5 研究チームを採択し、2008 年に新たに 4 研究チームを採択した。研究チームはそれぞれ 2012 年 3 月、2014 年 3 月まで研究開発を継続する。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、知的財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（図９）。現在９つの研究チームがプロジェクトに参画しているが、研究チームの研究テーマだけでは必要な要素技術やツール技術が満たされない可能性がある。たとえばファイルシステムのディペンダビリティの研究なども必要だが、今はカバーされてない。今後システムアーキテクチャの提案およびリファレンスシステムの基本設計、詳細設計を通じて、必要な要素技術やツール技術が洗い出される過程で、オープンソースとして存在していて利用できるか、あるいはプロジェクトで追加的に研究・開発をしなくてはいけないかなどを明らかにして行く。

本プロジェクトを進めるにあたり研究推進委員がユーザーの立場として企業から参加している。研究推進委員およびユーザーと継続的に要求や実用化に向けての方向性や実用化に向けての障害の確認と解決をして行く。プロジェクトの進捗は逐次公開しプロジェクトチーム外からの意見を取り入れプロジェクト全体にフィードバックをかけていく。このようにしてディペンダビリティの概念とディペンダブルなシステムの構築・運用の方法を社会的な共有資産として育てていくことを目指す。インターネットの発展や経済のグローバル化に伴い、IT システムや企業活動に国境がなくなりつつある現状を考えこのプロジェクトは日本国内に留まらず国際的に概念や手法を共有していくことが必須であると考えている。

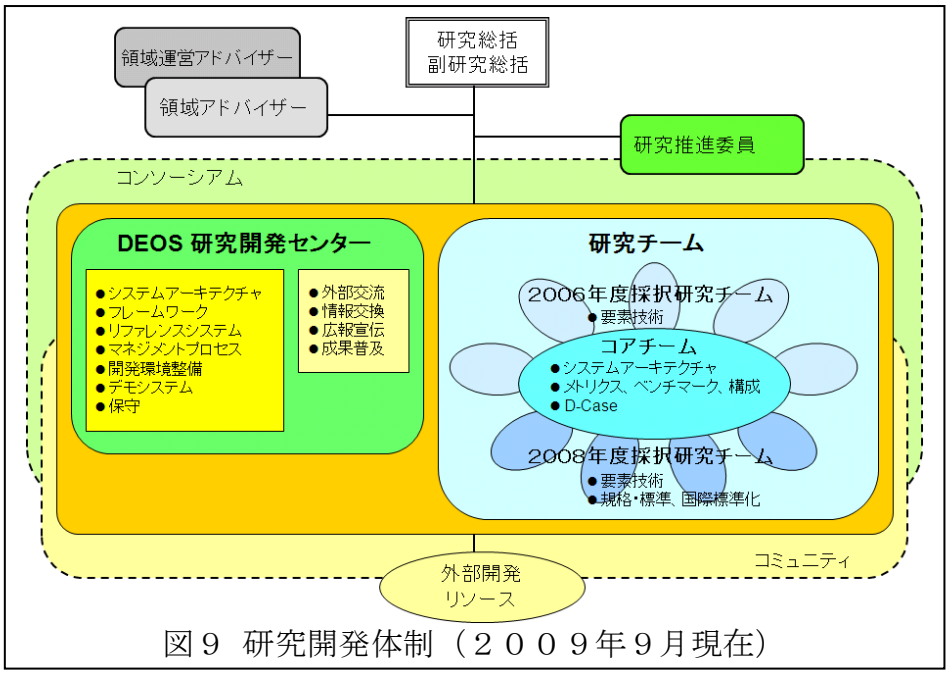

図９ 研究開発体制（２００９年９月現在）

6. ロードマップ

上記を進めるにあたって以下のフェーズを主なマイルストーンとして全体のプロジェクトを進めて行く（図１０）。

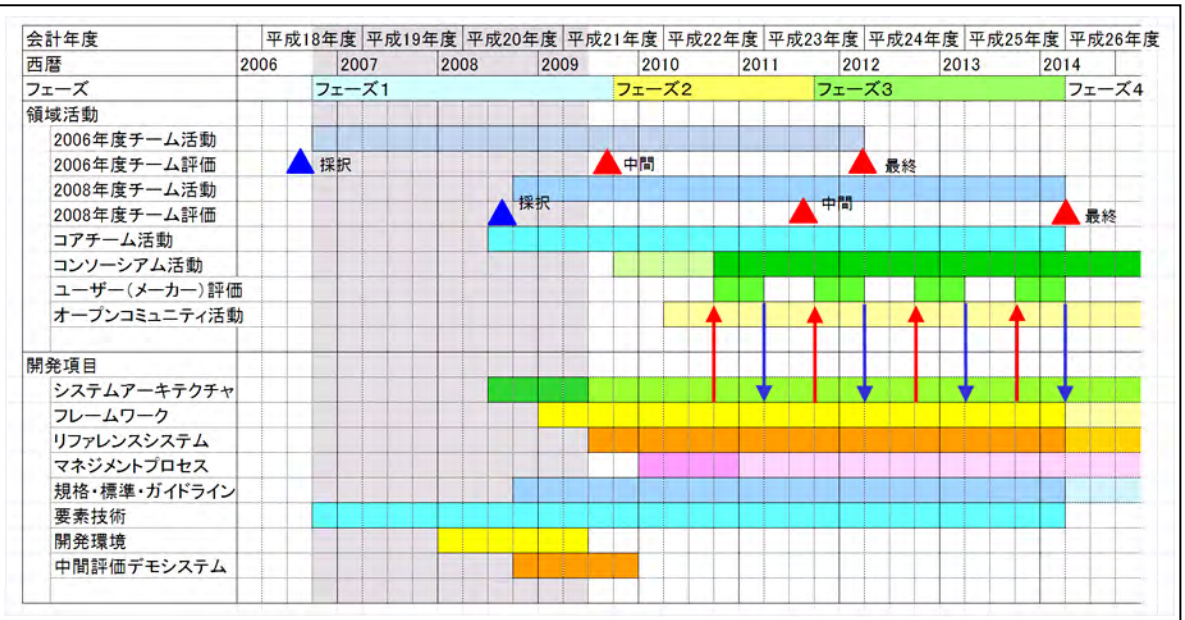

図１０ ロードマップ（２００９年９月現在）

- 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発表会として公開する）

- 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「**フレームワーク**」と「**リファレンスシステム**」の実装。企業など実際のポテンシャルユーザーを募ったコンソーシアム（ユーザー団体）の立ち上げ。システムアーキテクチャや要素技術の研究・開発に向けてのオープンコミュニティ結成。必要な事項の標準化。コンソーシアムメンバーによる「**フレームワーク**」と「**リファレンスシステム**」の試用、およびその評価のフィードバック。2008 年度採択研究チームの要素技術のいくつかを「**フレームワーク**」と「**リファレンスシステム**」にインテグレーションしてデモ。（以上を 2011 年 9 月に 2008 年度採択研究チーム中間発表会として公開する）
- フェーズ３（20011/10-2014/3）：コンソーシアムメンバーによる「**フレームワーク**」と「**リファレンスシステム**」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商品化への移行。必要な事項の標準化。
- フェーズ４（20014/4- ）：オープンコミュニティによる機能拡充、及びコンソーシアムによる活用と維持・発展。

7. 考慮すべき項目、課題

7.1. 知的財産・著作権の扱い

本プロジェクトでは研究開発された成果をできるだけ多くの企業やユーザーの実用に供し、ディペンダブルな組込みシステムによる社会インフラ構築に貢献するために成果物を可能な限り OSS（オープンソースソフトウェア）の形で提供することを目指している。OSS ライセンスには GNU GPL（GNU General Public License）、GNU LGPL（Lesser General Public License）、New BSD License、MPL（Mozilla Public License） 等様々なものがある。本プロジェクトの成果物にどのようなライセンスを適用するかは、本プロジェクトの成果の普及のために最も良い方法という観点で検討していく。企業との共同開発が行われた場合は企業の要求により制限をつける必要が出てくる可能性がでてくる。この場合は将来的に本プロジェクトの成果を普及するためにはどのような方法が良いかという観点で検討して方針を決めて行く。外部の研究・開発組織や標準化組織との連携を可能な限り行いユーザーが使いやすいような提供の形を目指して行く。具体的な連携については今後の課題として考えて行く。

7.2. オープンシステムディペンダビリティ・コミュニティ

本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に関する世界中の英知を結集するために、ある程度コンセプトが固まり、システムアーキテクチャの土台ができ、マネジメントプロセスのたたき台ができた時点で、国際コミュニティを立ち上げ、広く世界の研究者、技術者の参加を募っていきたい。その中では、コンセプトに関する議論や、アーキテクチャをさらに堅固にする議論や、プロセスを実用に供するための議論や、オープンシステムディペンダビリティを実現・向上させるための要素技術の研究・開発が活発に行われることを期待している。

7.3. オープンシステムディペンダビリティ・コンソーシアム

ユーザーとなる可能性のある企業や団体に当プロジェクトの成果物を評価してもらい、評価結果を受けて大きなマネジメントプロセスを回すことを通じてより実用的な成果物を目指し、賛同者を増やし、ポテンシャルユーザーと DEOS 研究開発センターとでコンソーシアムを作り、当プロジェクト完了後はそのコンソーシアムが母体となって新機能の追加や改善などの保守と利用のためのサービスを継続することを目指す。

8. 付録

8.1. ディペンダビリティの阻害要因

ディペンダビリティの阻害要因としてソフトウェアの動作する環境を中心に見てみると、運用や開発を含む環境から来るもの、ハードウェアの要因によるもの、人的ミスによって引き起こされるもの、悪意的な攻撃によるもの、という４つの阻害カテゴリーに分けて考えることができる。それぞれの阻害カテゴリーに関して、ライフサイクル毎にディペンダビリティを阻害する要因を考えることにより、阻害カテゴリーの軸と開発から廃棄までの時間軸を通じて総合的に把握することができる。このようにしてまとめた表を下に示す（表１）。

	環境 （運用環境、開発環境、等）	ハードウェア	人的ミス	人的攻撃
仕様	利用環境の予測（自然環境、システム環境（ハードウェア、ソフトウェア）、人的環境（組織、運用環境）、既存APの再利用、度重なる要求変更）	ハードウェアリソース見積もりの誤り、ソフトウェア生産性への考慮不足	仕様不備 諸元（機能見積もり）、性能（負荷見積もり誤り）、標準との整合性	企画・仕様の盗難、誤情報・偽情報流布
設計	マージン設定（対自然環境、対システム環境、対運用環境）、対象操作者（エンドユーザ、管理者など）の考慮不足、設計ツールの不具合 、他システムへの影響範囲分析ミス 、既存プログラムの再利用ミス、設計ツールの選定ミス、度重なる仕様変更	テスト機能実装不足、ソフトウェア生産性への考慮不足（ソフトウェアとハードウェアの機能分担ミスマッチ）、部品間の仕様不整合	アーキテクチャの選択・設計ミス、モジュール・サブシステム間のインターフェース設計、設計レビューもれ、仕様の読み違い、ハードウェアの性能見積もりミス、ユーザーインターフェースの設計ミス、例外処理不足、前提ソフトウェアのバージョン違い、障害時動作の検討漏れ	設計の盗難
実装・UT	開発ツールのバグ、開発環境の機能不足、開発環境不具合（組み合わせによるバージョン不整合など）、教育体制不備、利用ソフトウェアの検証不足（障害、性能）、度重なる設計変更	組み込み対象ハードウェアの開発スケジュールのミスマッチ・遅れ、歩留まり（品質）、仕様のミスマッチ、バグ（期間的に修正不可能なもの、修正できるもの）	コーディングエラー、コードレビュー漏れ、アルゴリズムのエラー、ライブラリー選択ミス、インテグレーションのミス、バージョン違い、タイミングの考慮ミス、ユニットテスト漏れ、著作権侵害、特許侵害	不正な製造偽造、コードの盗難、特許訴訟、著作権訴訟、不正コード組込み
インテグレーション・テスト	テストツールのバグ、テスト環境不具合、十分なテスト期間が確保できない、度重なる実装変更	テスト機能実装不足、ハードウェア残バグ	テスト項目漏れ（テストケース不足、実行時のミス）、結果確認ミス、テスト項目の間違い、テストレビュー漏れ、テスト環境設定ミス	不良品混入、テスト結果改竄、テスト仕様・テスト結果の盗難、意図的手抜き
流通	環境変化（自然環境、流通システム、関連担当者、流通規則）	環境変化（温度、衝撃、傾き、水没、破損）	不良、偽造品混入、輸送時の事故	偽造品混入、部品抜き取り、破損、開封
運用	経年変化、温度環境、湿度環境、他システムの不具合による連鎖エラー、想定外のデータによるシステムダウン、入力の過負荷、衝撃、電源異常（瞬断、変動、停電、コンセント抜け）、ノイズ（電磁波、静電気、宇宙線）、度重なるバージョンアップ・パッチ、過度の運用費、残バグによるサービス停止・誤動作	ハードウェアの劣化（メカニカル、化学、固体）、接触不良（コネクタ、スイッチ）過度な消費電力、騒音、電磁波ノイズ、加熱	仕様の誤解、ユーザーインターフェース考慮不足によるユーザー過誤、誤操作（誤機能選択、誤データ入力・選択）、インストールミス、設置ミス、データ移行ミス	運用時の攻撃（スパイウェア、ウィルス、アタック）、不正モジュール組込み、データ抜き取り、不正侵入、情報持ち出し
保守・更新	再現性のないエラー、保守・更新ツールの不具合、頻繁（過度）なバージョンアップ、バージョンアップ履歴の欠如、過度の保守費	障害情報収集機能の欠如、部品の保守期間、新部品の互換性	障害情報の情報不足、障害情報の連絡ミス、バージョンミスマッチ、バックアップミス、リストアミス、バージョンアップ未実施（不完全実施）	保守・更新時の攻撃（ウィルス、アタック）、不正モジュール組込み、データ抜き取り
廃棄・再利用	消去情報の痕跡	環境汚染、リサイクル・リユーズ考慮不足	個人情報等の消去ミス、操作履歴等の消去ミス	情報抜取り、部品抜き取り

表１　ディペンダビリティ阻害要因

この阻害要因を組込み OS やその周辺の技術と関連付けるために、

① 企業などシステムを開発しそのシステムを使ったサービスを提供する側の体制やプロセス及びシステム全体のインテグレーションでカバーするべき領域
② OS が関連するが他の要素によるディペンダビリティの技術を必要とする領域
③ OS やその周辺技術によってディペンダビリティの大きな貢献が期待できる領域

と言う３つの領域に分けることができる。表１ではそれぞれを黄色、空色、橙色で色分けしている。

上記の阻害要因を見たときに従来の技術や CMMI、モダン PM （Project Management） 手法などに従って要求・仕様から設計・開発・テストへのプロセスをきちんと実行し要求通りのものづくりを行っていくことが必要とされ、２１世紀の組込み機器の置かれた状況によって起こる問題に対処するための新たな手法や技術が必要とされていることも読み取れる。その背景にあるものは、技術の進歩によるソ

フトウェアの巨大化とともに組込みシステムが単なる独立した“製品”ではなくネットワークに組込まれてユーザーへサービスを提供するための基盤として位置づけられるところにある。そのための技術者の育成やマネジメントは欠かせない。

8.2. 世界の関連標準、関連活動団体

標準

- IEC 61508: Functional Safety
  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: Software Life Cycle Processes
  http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=21208
- ISO/IEC 15288: System Life Cycle Processes
  http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=43564

プロセスガイド

- 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/
- 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

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 https://www.trustedcomputinggroup.org/home

- CELF: CE Linux Forum, an international open source software development community http://www.celinuxforum.org/
- 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 https://www.jaspar.jp/
- FlexRay Consortium: Consortium for the communications system for advanced automotive control applications http://www.flexray.com/
- NoTA: Network on Terminal Architecture http://www.notaworld.org/
- LIMO Foundation: Industry Consortium dedicated to Linux-based operating system for mobile devices http://www.limofoundation.org/

8.3.参照資料

1. H. Yasuura, “On Dependability”, T. Nanya, “Concept and Issues on Dependability”, K. Iwano, “Dependability in Social Services”, in Dependability Workshop Report （in Japanese）, 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, pages 399-405, May. 2007
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, pages 733-743, 1967