DEOS-FY2012-SV-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**

D-Case/Agda による

アシュランス・ケース記述

独立行政法人**産業技術総合研究所**

木下佳樹　武山誠　平井誠　湯浅能史　木藤浩之

「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。**

1. はじめに

安心・安全には「絶対」はありません。牛肉の BSE 問題でも、原子力発電でも、「決して悪いことが起こらない」とは、全知全能の神でもない限り、いえることではありません。想定外の事象が常に起こりうることを盾にして安全対策の怠慢を正当化することは許されませんが、何にでも悪いこと（リスク）が起こる可能性があるのだという真実から目をそらしてしまうのも、賢明な態度ではないでしょう。

- 安全だと言い切って安心して忘れてしまう、
- 安全でないと言い切って不安になり、安全なものに取り替えるように求める、

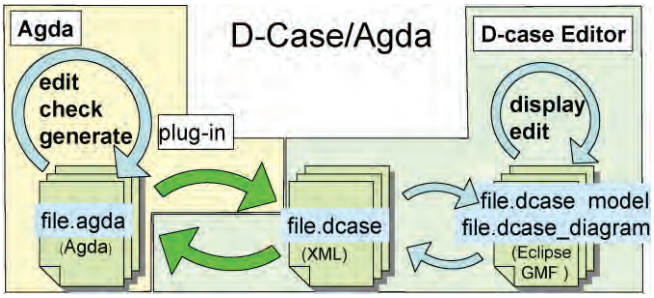
という両極端のいずれでもなく、その中道を得ることが必要です。大抵のシステムでは、安全か安全でないかの二者択一はできません。現実はそのような単純な状況にはないのです。安心・安全がどの程度担保されているのか、つまり

- どんな悪いことが起こりうるのか、
- どれくらいの可能性で悪いことが起こるのか、
- 悪いことが起こった場合の対策をどうするのか

などについて、込み入った議論をする必要があります。このような安心・安全についての議論ややりとりはアシュランス・コミュニケーション(assurance communication)あるいはリスク・コミュニケーション(risk communication)と呼ばれています。

本稿の前半では、アシュランス・コミュニケーションにおいて、安心を申立てるために使われるアシュランス・ケースとよばれる文書について、それがどんなものかを説明します。また、アシュランス・ケースの一つで、我々DEOS プロジェクトが提唱している D-Case[1] についても説明します。

本稿の後半では、D-Case を書くためのツール D-Case/Agda を紹介します。D-Case を書くためには、拡張 GSN 記法によるグラフィカルな図式を直接描く D-Case Editor というツールもありますが、D-Case/Agda は、Agda 言語というプログラミング言語によって D-Case を記述するためのツールです。プログラミング言語によって D-Case を記述することによって、プログラムと D-Case のつながりがスムーズになるばかりでなく、記述した D-Case のチェックを、型検査という形で、かなり細かく行うことができます。グラフィッ



[1] アシュランスの対象となる性質が安全であるかディペンダビリティであるかによって、アシュランス・ケースは安全ケース (safety case) と呼ばれたりディペンダビリティ・ケースと呼ばれたりします。開放系ディペンダビリティ (Open Systems Dependability) のアシュランスを記す文書が D-Case と呼ばれるものです。

クな表現が欲しいときには、Agda 言語で書かれた D-Case を D-Case Editor の入力形式に変換することができます。逆に、D-Case Editor で書いた図式に対して詳細のチェックをしたい場合には、図式のファイルを Agda 言語に変換することもできます。

2. アシュランス・ケース ― 安全・安心の申立て

2.1. アシュランス・ケースとは？

アシュランス・コミュニケーションでは、

- 客観的なデータ（証憑）に基づいて
- 正しいばかりでなく、
- みんなが納得できる議論を行って、
- 一般論ではなく、特定の環境下での特定の目的の下では、
- システムの安心・安全が十分に達成されるという主張の合意すること

が目標になります。このような主張を証憑から導く議論を記して、安全・安心を申し立てる文書がアシュランス・ケースと呼ばれる文書です。日本語にするのなら、安心・安全申し立てというところでしょう。

2.1.1. 主張

アシュランス・ケースは、安心・安全に関する何らかの主張を導く文書です。その主張を明確に記しておくことが、当然ながら求められます。

安心・安全に関する主張は大なり小なり主観的で漠然とし、曖昧なものになりがちで、技術の対象になりにくいので困ります。アシュランス・ケースが導く主張は、そのような曖昧さがあっても構いません。その代わり、曖昧な主張を、客観的なデータに基づいた厳密な主張によって、どのように解釈したのか、という解釈の過程を記すことが求められます。このような記述を与えることが、アシュランス・ケースの目的の一つです。

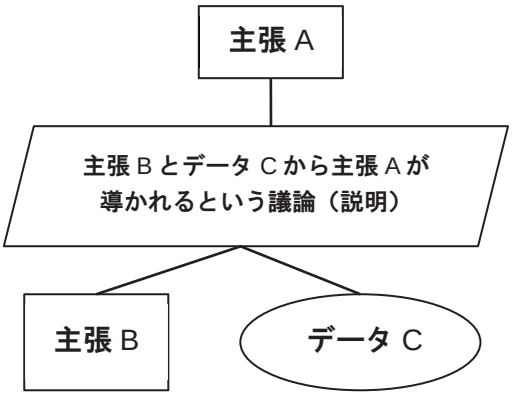
2.1.2. データ（証憑）

アシュランス・ケースが根拠とするデータ（証憑）は、いろいろあり得ます。重要なことはそれが客観的で、解釈が一義的に決まるようなデータであるということです。証憑の例には次のようなものがあり得ます。

- FTA や FMEA などの故障解析の結果
- ソフトウェアのテスト結果や形式検証の結果
- 法令や規格等の規則
- 部品などの動作測定結果

2.1.3. 議論

主張をデータによって裏付けるのがアシュランス・ケースを書く目的です。普通は、データが主張をどのように裏付けているのかについて、込み入った説明あるいは細かな説明が必要です。データからなぜ主張が導きだされるのかについての説明を「議論」と呼びます。



主張が直接データによって裏付けられる場合もあれば、主張をいくつかのデータや別の主張に分解して裏付ける場合もあります。後者の場合は分解してできた別の主張を、さらにそれらを裏付けるデータと結びつける議論を記す必要があります。

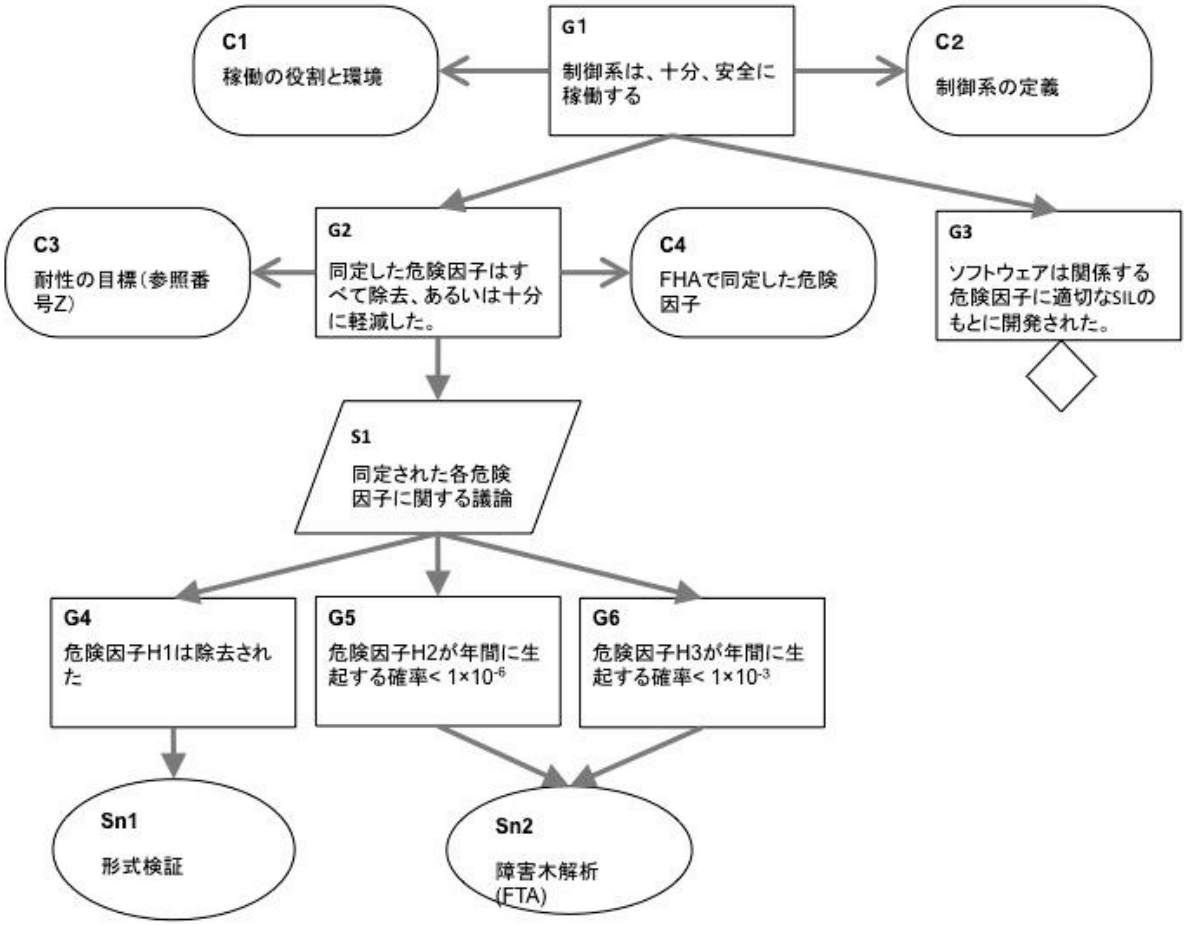


図 1　GSN によるアシュランス・ケース記述例

2.1.4. 語彙と基本の前提

アシュランス・ケースには、主張がデータによってどのように支えられるかを示す議論が記されますが、それだけでなく、主張、データ、議論の各部分で用いられる語彙を明確に記すことが求められます。また、そのアシュランス・ケース全体をとおして前提とすることがら

も、基本的前提として記されることが求められます。基本的前提のなかには、システムが稼働する環境や、稼働の目的なども含まれます。

2.1.5. 推論の図式表現

以上で説明したように、アシュランス・ケースは主張、データ、議論からなる推論部と、語彙と環境や目的などの基本的前提を記した語彙・前提部からなります。このうち、推論部の構造を図的に表現する方法がいくつか知られています。図 1 は、Goal Structuring Notation (GSN)と呼ばれる表現法です（[2]参照。）

アシュランス・ケースを記すもう一つの方法は MACL (Machine-checkable Assurance Case Language) による方法です。たとえば後で紹介する Agda 言語はそのような言語のひとつです。Agda 言語では推論部だけではなく語彙・前提部も表現することができ、記述の整合性を検査することができます。

2.1.6. 正しさと納得

アシュランス・ケースの中で、どの程度に込み入ったやりとりが必要なのかは、そのシステムにどれくらい深く関わっているかに応じて、人によって異なります。例えば、菜食主義者にとって BSE は他人事です。また、わが国の原発の近くの住民や電力会社の技術者と、大陸の真ん中の草原に暮らして、電力を殆ど使わない遊牧民では、原発問題について、どの程度つっこんだ議論が必要なのかは異なります。言うまでもありませんが、遠い草原に暮らす遊牧民でも、わが国で原発事故が起こったら何らかの影響を受けざるを得ません。直接の放射線被爆があり得るだけでなく、経済的、社会的中長期的な影響もあるでしょう。しかし、万一事故が起こったら数時間以内に避難しなければならない近隣住民やすぐに事故現場に駆けつけなければならない技術者と、何千キロも離れたところに暮らす遊牧民では、わが国の原発事故防止への関わり方の深さに大きな違いがあるのは明らかです。したがって、アシュランス・ケースに求める議論の精密さへの両者の要求も、大きく異なるでしょう。

ここで一旦まとめましょう。アシュランス・ケースとは、

- システムのアシュランスが、十分達成されていることを記した文書です。
- 客観的な証憑を基にして、正しいだけでなく読者に納得できる議論を積み重ねる文書です。
- 特定のシステムが、特定の環境下で、特定された使われ方をしているときに、どのように安心・安全が達成されるのかを記した文書です。

2.2. どんな役に立つの？

では、アシュランス・ケースはどのような役に立つのでしょうか。

- 小さく叩けば小さく響き、大きく叩けば大きく響くシステム・ドキュメント

アシュランス・ケースは、大規模で複雑なシステムの全体像を伝えます。一方、システムのある部分について、知りたければいくらでも詳しい情報を記す文書にもなります。

システムの大部分の関係者は大抵、技術の専門家ではないので、システムの技術的な詳細を理解するのは難しく、また技術情報は不要なのが普通です。例えば、原発近隣住民にとっては、原発の発電メカニズムなどは一義的には必要なく、また専門家でもない限り理解が困難です。したがって、非専門家は通常、システムについての全体像を知っていればよく、技術的な詳細は専門家に任せたいものです。

しかし、システムへの関わり方によっては技術的な情報も、一定の詳しさで知ることが必要になる場合があります。例えば近隣住民は原発事故が起こったときには避難しなければなりませんが、何故避難しなければならないのか、被爆とはどういう風に悪いことなのか、どの程度の事故ならどれくらい早急に避難しなければならないのか、といった近隣住民には最も重要なことを深く知ろうとすると、原子炉についてもっと詳しいことを知る必要が出てくるでしょう。

読者にとっての必要に応じて、詳細は専門家に任せることにして大雑把な情報を得て判断することもできるし、自分で元のデータを参照してそれを基にいろいろな判断をすることもできる、というような文書がアシュランス・ケースです。

- 実際のシステム・業態に即したシステム分析を得るための文書です。

一般論ではなく、特定のシステムが、特定の環境下で、特定された使われ方をしているときの安全・安心を示すのがアシュランス・ケースです。

- ライフサイクルの観点から、システムの全体をみた安全・安心を検討するための基本材料を提供します。

システムの運用・障害対応・環境や要求の変化対応・廃棄といったプロセスは、開発から保守、場合によっては契約担当者までが広く関係するため、安全・安心の検討のために必要な事項をまとめた文書が得られにくいのが現状です。アシュランス・ケースはそのような必要事項を提供します。

- 工数削減を検討するための基本データを提供します。

製品ライン展開や、類似システムへの応用展開での工数削減検討では、過去の開発プロジェクトや製造プロセスが関係するため、工数削減検討のために必要な事項をまとめた文書が必要になります。アシュランス・ケースにはこれらのデータも盛り込むことになります。

更に、アシュランス・ケースには以下のような利点もあります。

- 起こりうる事故・障害を予見し、深刻度・頻度・重要性に応じた適切な予防や復帰の手段や体制を順次構築し、継続的に改善していくために、アシュランス・ケースを書くことによって関係者の間で議論を深め、合意を広げていくことができます。
- アシュランス・ケースを書くことによって、対象とするシステムの範囲、要求・実

装・移行・運用・廃棄といったライフサイクルの範囲、想定する事故・障害・リスク・環境・規格・規制の範囲、それらに応じた適切な関係者の範囲などを、明確にして合意することができます。

- 重要度に応じて、いつまでに、誰が、どの範囲まで、どのレベル（詳細さ、精度、効果、費用）まで、検討・議論・合意・改善を拡張していくのかなどを明確にアシュランス・ケースに記すことによって、プロジェクトの全体像をつかむことができます。

2.3. 例えば？

アシュランス・ケースはシステムの安心・安全の根幹に触れる文書ですから、企業の信用保持のためにも、また、システム運用上のセキュリティ確保のためにも、なかなか公表されません。しかし、ヨーロッパの航空情報システムデータベース EAD (European Aeronautical Information System Database) についての安全ケース「EAD 安全ケース」が公表されています（[1][3][4]参照）。安全ケースは、アシュランス・ケースの一種で、安全性に関するアシュランスに絞り、これが達成されていることを記す文書です。本節では、これを概観します。

世界各国は、それぞれ自国のための航空管制システムを持ちますが、国際的な飛行を可能にするため、互いに連携して、国際航空管制システムをもつくっています。欧州各国の航空管制システムが連携を進める過程で形成してきた合意事項について、おのおのの重要度・優先度に応じた安全ケースが「EAD 安全ケース」です。

この文書が作られるまでに、過去の事故・障害の記録、その分析と類似した事故・障害のリスク分析、それらの対策とその効果の評価、EAD へのシステム内部・外部からの要求内容、内部と外部の要求の整合性、そこから派生する要求内容、要求内容が実装される過程の追跡、各国の既存データベースと並存して運用するためのデータ引継ぎ・運用・訓練の体制・手順、これらの遵守状況をモニタリング・監査するための成熟モデルと評価メジャー、アンケートなどが準備されました。

EAD 安全ケースでは、これらの EAD の安全性に関連する文書を参照し、システムとそのライフサイクル全体を俯瞰し、関係者で共有し、重要度に応じて議論を深めた結果、システムの安全性を合意し、今後の改善アクションを明確化しています。

2.3.1. GSN による表現

例えば「このシステムがこういう環境でこういう使われ方をしているときには十分な安全性が達成されている」などといった、主張したい事柄を「EAD がライフサイクル全体の観点から適切に安全性を確保できている」というゴールとして設定し、それを論証する戦略によって、いくつかのサブゴールに分割することを繰り返して、重要度に応じて議論を深め、最終的に証憑となる文書を参照する構成になっています。

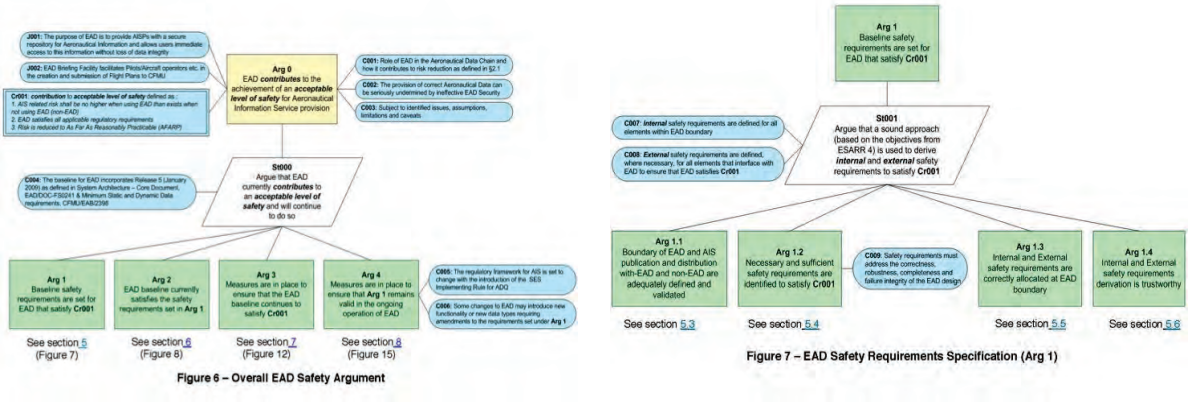


図 2　EAS 安全ケース中の GSN 表現

図２のように GSN を用いた表現が与えられ、システムとそのライフサイクル全体を俯瞰し、重要度に応じた議論の深さのバランスを、直感的に認識しやすくなっています。GSN の下位階層にあたる深い議論の部分では、表に列挙するなどの表現も使い分けられています。

また、証憑となる関連文書を参照する際には、その文書がその議論を証明する証拠として妥当なものであることが理解できる程度まで、概略をアシュランス・ケースの文書中に記述しています。

2.3.2. ライフサイクル・プロセスの中のアシュランス・ケース

ライフサイクルの各プロセスが適切に計画・実施・監視・監査されることを確かめています。特に、重要とされる以下のプロセスについては、サブゴールに分解し、議論を深めて、証憑の内容を概略説明しています。

- 内部要求実装の安全性：内部要求の実装過程が追跡できているか、機器調達プロセス、データ実装プロセス、訓練プロセス、従来システムからの引継ぎプロセス、操作・運用プロセスが適切に計画・実施・監査されているか
- 外部要求実装の安全性：データベースに情報登録・使用するユーザの責任、承諾事項を定め、契約し、遵守状況を監視、遵守させるプロセス・体制が適切か
- 移行計測の安全性：従来システムからの移行でのモニタリング対象、プロセス、体制が適切か、事故・障害の再発防止、モニタリング基準の見直し、機密管理状況のモニタリングは適切か
- 移行変化受け入れの安全性：従来システムからの移行に伴う変化に対する受け入れプロセス、体制は適切か、互換性、安全遵守、変化に応じた実装、変化に対する安全レベルの見直し、実装の見直し

なお、アシュランス・ケース更新の過程の中で、明確化された改善アクションのリストが公表され、改善が継続されています。

EAD では、最新のアシュランス・ケースが公表された時点では、5 年半にわたり、運用中のデータベースの障害・エラーがモニタリング・記録され、毎月のレポートとして、監視委員会へ報告され、安全性の改善活動が継続されています。

2.4. 普及しているの？

アシュランス・ケース（安全ケース、セキュリティ・ケース、ディペンダビリティ・ケース）は原子力発電や軍事システムなど、高度の安全性が要求されるシステムでは 20 **年以上前から使われてきました。数年前から情報システムにも導入され始めている他、最近では米国** FDA **による医療システム認証への導入、**ISO26262 **による車載システムへの導入の示唆などが注目されています。**

既にアシュランス・ケースの体裁や内容に関する国際規格も ISO、IEC **をはじめとする標準化団体から発行されており、現在開発中の規格もあります。**

- ISO/IEC 15026-2 Assurance case : ISO/IEC 15026 Systems and software assurance **はアシュランスに関する要件の規格で、４部からなります。現在改訂作業が進んでいます。**
  - Part 1: Concepts and vocabulary **は、用語と概念の解説をした** technical report (TR)**で、**2012 **年** 10 **月現在、**international standard (IS)**への改訂作業が進行中です。**
  - Part 2: Assurance case **がアシュランス・ケースに関する** IS **で、**2011 **年に発行されました。アシュランス・ケースの様式（構文）と内容の概略を規定しています。**
  - Part 3: System integrity levels **は、安全性のみならずセキュリティや可用性など、システムのいろいろな属性についての完全性水準を定める方法を規定した** IS **で** 2012 **年に発行されました。**IEC 61508 Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems (E/E/PE, or E/E/PES)**では安全性について、**ISO/IEC 15408 Evaluation criteria for IT security **ではセキュリティについて、それぞれ完全性水準を定めていますが、安全性やセキュリティ以外の属性の完全性水準を決める、一般的な手順を定めたのが** ISO/IEC 15026-3 **です。**
  - Part 4: Assurance in the life cycle **は、システムのアシュランスを達成するために** ISO/IEC 12207 Software life cycle processes **および** ISO/IEC 15288 System life cycle processes **に規定されているライフサイクルの各プロセスにおいて何がなされなければならないかを規定した** IS **です。**
- OMG SACM 1.0 : OMG (Object Management Group) **は** UML **をはじめ、ソフトウェアに関するいろいろな規格を発行、管理している団体です。**SACM 1.0 **は** SACM **の正式名称で、**2012 **年に制定されました。アシュランス・ケースの推論部とデータ（証憑）の形式を規定するものです。**ISO/IEC 15026-2 **よりも細部にわたって規定されています。**2012 **年** 10 **月現在、**1.1 **への改訂作業が進んでいます。**
- IEC 62741 The Dependability Case.: IEC 62741 **は、ディペンダビリティ・ケースについて詳細に規定する規格で、**2012 **年** 10 **月現在、**Committee Draft (CD) **の状態にある、制定作業が進行中の規格です。**SACM **と同様に、**ISO/IEC 15026-2 **に比べ詳細にわたった規定がなされつつあります。**
- OMG MACL : OMG MACL (Machine-checkable Assurance Case Language) **は、アシュランス・ケースの記述言語を規定してアシュランス・ケースの自動検査（整合性検査）を可能にしようとするもので、**2012 **年** 10 **月現在、**OMG **で必要性の調査が始まった** (Call For Information **提出) 段階です。**

2.5. アシュランス・ケースから D-Case へ —— どう違うの？

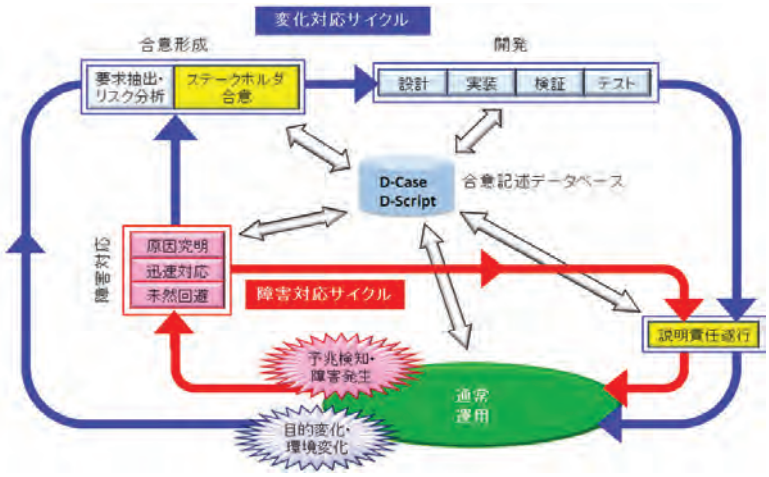
安全性に関するアシュランス・ケースを「安全ケース」と呼んだように、Open Systems Dependability に関するアシュランス・ケースのことを「D-Case」と呼びます。ところで Open Systems Dependability とは何でしょうか？

2.5.1. Open Systems Dependability

現代のシステムは、ますます大規模化・複雑化への道を歩んでいます。このようなシステムでは、どこからどこまでをシステムと認識して管理すべきなのか、その境界は曖昧で、仕様を正確に定めることは大変困難です。仮に何か仕様を定めたとしても、依って立つ動作環境のちょっとした変化などにより、変更を余儀なくされます。システムのこのような特徴に注目する立場を「開放系の立場 (Open Systems viewpoint)」といいます。一方、比較的小規模で明確な境界をもち、環境を特定し易かった従来のシステム観を「閉じた系の立場」といいます。開放系の立場で着目する問題は、大きく分けて、変化と曖昧さの二つに帰着します。

- システムは変化する
  - システムの構成要素が変化しまする。ソフトウェアの更新、機械の老化など、理由は様々です。
  - システムの環境が変化します。繋がっているネットワークが変わる、システム稼働の目的であるビジネスが変わるなど。
  - システムへの要求が変化します。利害関係者の都合が変わる、法令や社会の規律が変わるなど。
- システムは曖昧である
  - 関係者たちが用いる語彙が異なっているときなど、ひどい場合には同じ単語で異なる意味を持つこともあります。このため、システムに関する共通理解を形成しにくく、理解に齟齬のある場合も少なくありません。
  - システムの境界はあいまいになりがち。どこからどこまでがこのシステム、という設定をすることが困難なことがよく起こります。
  - システムの稼働条件や環境に関する前もっての想定には、必ず破れが生じます。予想外のトラブルは避けることができず、それにより生じる被害への対処が必要となります。

これらの問題にもかかわらず、システムを長く安定的に運用するために必要な能力のことを、“Open Systems Dependability”と呼びます。DEOS プロジェクトでは、そのようなシステ



ムが辿るべきライフサイクルとして “DEOS プロセス”を提唱しています（[5]参照）。

DEOS プロセスは２つのループからなっています。内側を回っている赤いループは「障害対応サイクル」と呼ばれています。日々のシステム運用の中で直面する典型的な障害への対応は、このサイクルを辿って行われます。障害対応サイクルでは処理しきれない、想定もしていなかったようなトラブルが起きたときには、外側の青いループに従って対処します。また、長期にわたる運用で、システムを取り巻く自然・社会環境が運用開始時と大きく変わってしまった時などの対処も、これに従います。青いループは「変化対応サイクル」と呼ばれています。

変化への対応は多くの場合、システムの変更を伴いますので、関係者達に「このように変更しますがよいですね」と合意をとる必要があります。システムに関する利害関係者の合意が、全ライフサイクルを通じて常に維持されていることは、Open Systems Dependability の重要な特性であり、DEOS プロセス中にもこれを確保する箇所が明示的に組み込まれています。ここで合意内容を表現するのが D-Case です。合意された D-Case は合意記述データベースに登録されて、DEOS プロセスの各所で参照されます。

2.5.2. DEOS プロセスの D-Case

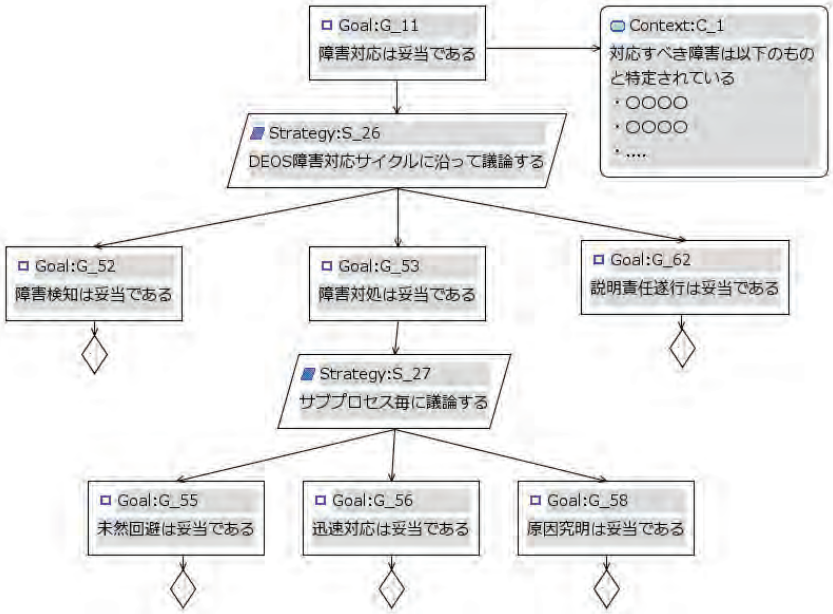
システムが開放系のディペンダビリティを持つことを示すために、DEOS プロセスの二つのサイクルに対して、それぞれの妥当性を示す D-Case を作成します。これをそれぞれ、障害対応 D-Case、変化対応 D-Case とよびます。

障害対応 D-Case

まず、障害対応サイクルの D-Case から説明しましょう。 この D-Case は、「（このシステムの）障害対応は妥当である」という主張を説明するものです。これをいくつかの主張に分解するのが定石です。どのように分解するのかは、システム毎に異なるものであり、利害関係者が合意した分解であれば、何でもかまいません。ひとつの方法は、障害対応サイクルが辿るプロセスに沿って議論を進める、というものです（[7] 参照）。ここでは、そのような例を紹介しましょう。

- 障害検知 の妥当性： 障害やその予兆を検知するプロセスが適切に定められていること。検知はシステムが自動的に行うものだけとは限りません。オペレータの判断など、システム運用全体を見渡して、全ての「予め特定された」障害が検知できる体制にあることを主張します。
- 障害対処 の妥当性： 検知された障害や予兆が適切に処理されること。これは更に三つのサブゴールに展開されます。検知されたのが予兆なら未然回避が、すでに障害が生じてしまったのなら一刻もはやく正常状態に戻すための対処が、それも無理ならば事態の分析と変化対応サイクルへの移行が、行える体制にあることを主張します。
- 説明責任遂行 の妥当性: 生じた障害とそれへの対処について、利害関係者に適切に説明できること。きちんと説明するためには障害の種類や起こった時刻、対処後の経緯などの情報が分からなくてはいけません。これらが記録され、関係者に通知できる手筈が整っていることを主張します。

これらの主張を説明する D-Case をさらに書いていき、根拠がすべて客観的なデータになるまでそれを繰り返すことになります。



ところで、障害検知の妥当性 のところに出てくる「予め特定された」障害とは、何時、誰が特定するのでしょうか。これについては、あとで触れます。

システムの運用者オペレータは、運用時にこの D-Case を参照しながらシステムを扱います。紙に書かれた D-Case のファイルを使うこともあるかもしれませんが、現代であれば電子化してブラウザ等で閲覧することが多いでしょう。そのようなときには、更に便利なことがあります。ゴールに書かれた合意事項が、今まさにその時点で成り立っているか否かを、リアルタイムで知ることができるのです。原子炉の温度が適切にたもたれているか、とか、サーバへのアクセス数が処理可能な範囲にあるか、といった情報が D-Case 上で確認できます。これは普通のアシュランス・ケースには無い D-Case 独特の考え方です。このような機能を実現するため、データ(証憑)をセンサなどに結び付ける指定ができるようになっています。これを「モニタリングノード」と呼びます。

変化対応 D-Case

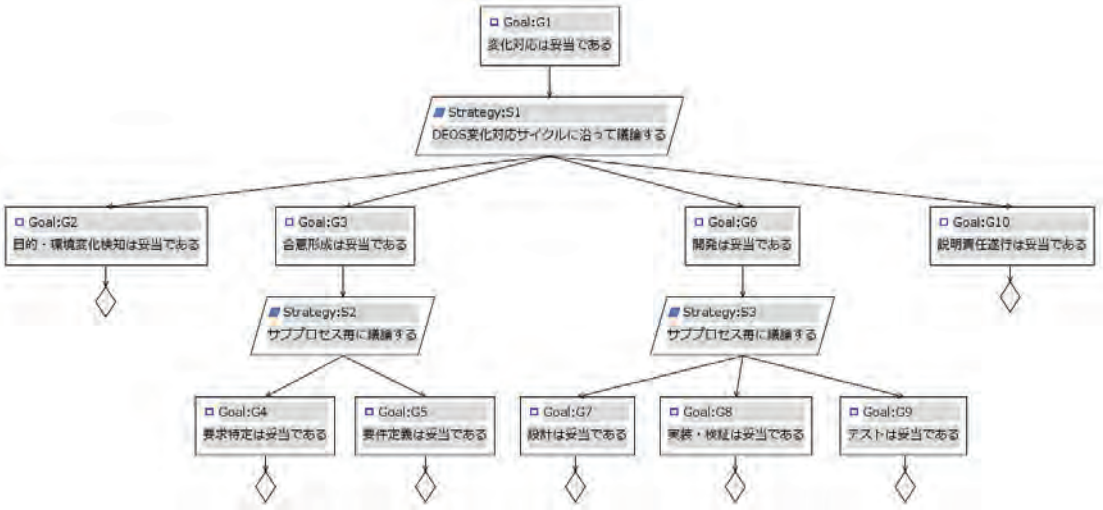
変化対応 D-Case は「（このシステムの）変化対応は妥当である」という主張を説明する D-Case です。障害対応 D-Case の場合と同じく、主張の分解のしかたは、利害関係者が合意するような方法であれば何でも構いません。ここでは、変化対応サイクルの辿るプロセスに沿って議論を進める方法を紹介します。

- 目的・環境変化検知の妥当性: システムに対する要求や、置かれた環境の変化が、的確に認識されること。環境の変化は大なり小なり常に起こることですが、ここでいうのは、現在の D-Case で合意されているシステム動作の前提が、もはや成り立たなくな

ってしまった、という程の大きな変化です。

- **合意形成の妥当性**: 新しいシステムへの要求やシステム要件についての、利害関係者たちの議論や合意が適切に行われること。利害関係者たちは、環境の変化や起きてしまった想定外の障害についての分析を十分に踏まえて議論し、要求や要件を改定できなくてはいけません。また、これらの改定により D-Case の変更も必要になります。これらが適切に行える体制が整っていることを主張します。
- **開発の妥当性** : システムの変更方法が妥当であること。合意されたシステム要件に従って、設計が行われ、更に設計に従って実装が行われなくてはいけません。これを保証する手段が開発の各フェーズで行われていることを主張します。また、完成した新システムが、関係者の要求を満たしているかどうかの検証・妥当性確認の方法自体が適切であることも主張します。
- **説明責任遂行の妥当性**: 開発の全てのプロセスが正常に行われたことが、関係者に適切に説明されること。説明のためには、各フェーズが正常に終了したことを示す何某かの証拠が必要です。そのようなものがちゃんと確保できて、関係者たちに提示する適切な場も用意されていることを確認します。

変化対応 D-Case では、想定外のトラブルや環境の大きな変化に晒されたときシステムをどうやって変更するのか、そのプロセスの妥当性について合意事項をまとめます。事前にこのような形で合意をとっておけば、いざというときにあわてずに行動できるので安心です。



障害対応 D-Case のところで「予め特定された 障害とは、いつ、誰が特定するのか」という疑問を棚上げしたままでした。答えは「変化対応サイクルの合意形成プロセスにおいて、利害関係者たちが決める」です。彼らは新しいシステムへの要求やシステム要件に合意したあと、これを脅かすリスクについても考えます。想定される障害を特定して、それらへの対応を議論し、合意結果を障害対応 D-Case にまとめるのです。

変化対応 D-Case では、このような D-Case 改変手順についても議論します。D-Case の変化を、更にまた D-Case で記述することになるというのは、面白い現象だと思います。

3. D-Case/Agda — 形式言語による D-Case の記述

3.1. 形式言語で記述するメリットについて

D-Case をはじめとするアシュランス・ケースは、一般に巨大な文書になりがちです。文書の細部にまでわたって整合性をとるのは大変ですが、一方で、ほんの少しの不整合も、システムの大きなトラブルに直結し得るものです。そこで、機械的に整合性をチェックできる部分はできるだけ機械にチェックさせたいものです。いうまでもなく、整合性検査を 100%機械に任せることは不可能です。D-Case は現実を反映する文書ですから、そこに現れる法則を全て書き下ろすことは人間にはできません。それでも、機械的検査が可能な部分はあります。それを機械に任せることによって、D-Case の、より重要な部分の検査に人間の注意を集中することができます。

以下では、我々が開発している D-Case/Agda システムを例にとって、機械的検査を行いながら D-Case を開発していく様子を紹介しましょう。

3.2. D-Case/Agda とは

汎用の証明支援系 Agda システムを用いて、D-Case の記述を支援することができます。Agda 言語による D-Case の記法、表現法を D-Case in Agda と呼びます。Agda システムを使えば、インクリメンタルに構文や型の検査をしながら、対話的に D-Case を構築していくことができます。標準エディタとして Emacs がついていますが、別のエディタで作成した D-Case in Agda の構文や型を自動検査することもできます。検査に通った記述は、かなりの程度、整合的な D-Case だということができます。

DEOS プロジェクトでは D-Case Editor という、D-Case のグラフィカルな表現を構築するシステムも開発しています。D-Case Editor で構築したグラフィカルな表現と、Agda システムで構築した D-Case in Agda による表現は、互いに変換可能です。ふたつのシステムを行き来して、ビジュアルな取扱いが可能な図式表現と、より豊富な自動検査が可能な Agda 表現を使い分けながら D-Case を構築することができます。

3.3. くるっと、一巡り

Agda システムを用いた D-Case の作成、編集、検証の流れをざっと見てみましょう。

右の図式に相当する D-Case in Agda 表現を作成する様子を、Agda システムの六つのスナップショットによって示します。

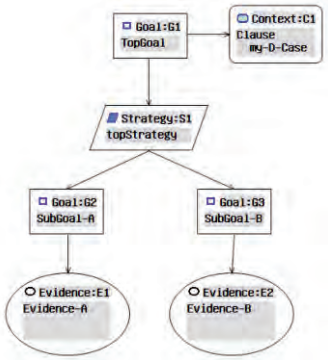
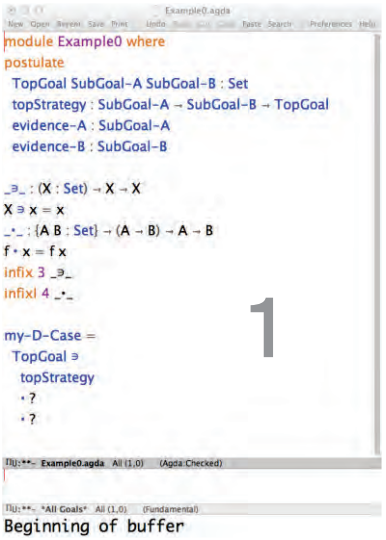
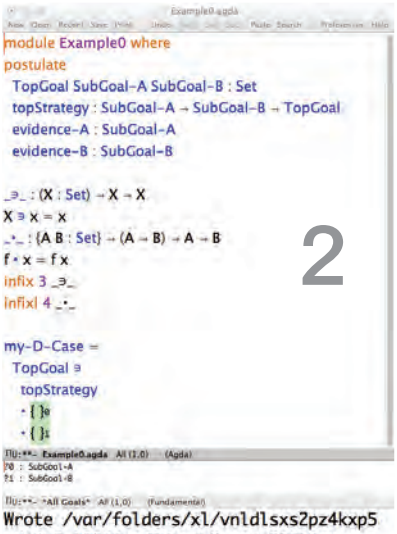


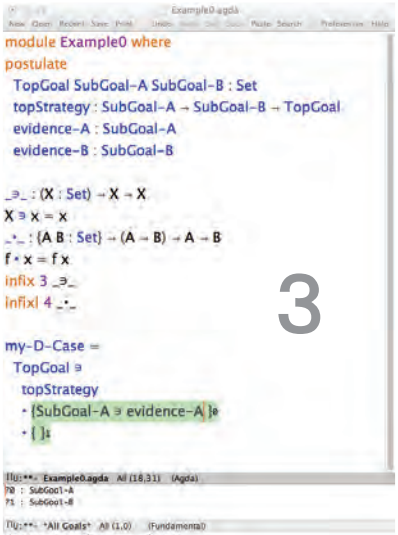
図 3　簡単な D-Case の図式

スナップショット１は、図 3 の図式に相当する D-Case をだいたい打ち終わったところです。ただし、topStrategy という議論の戦略に必要な二つのサブゴール（主張）をまだ記入しておらず、そこに ？ を記しています。

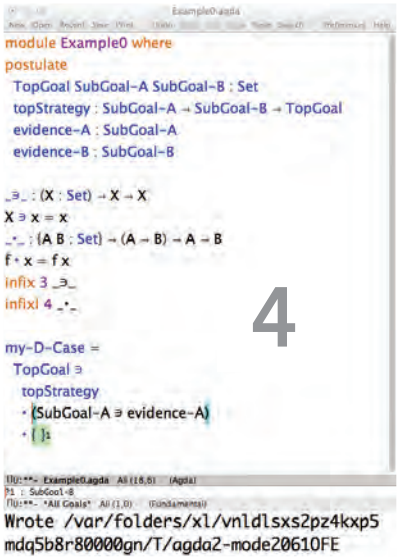




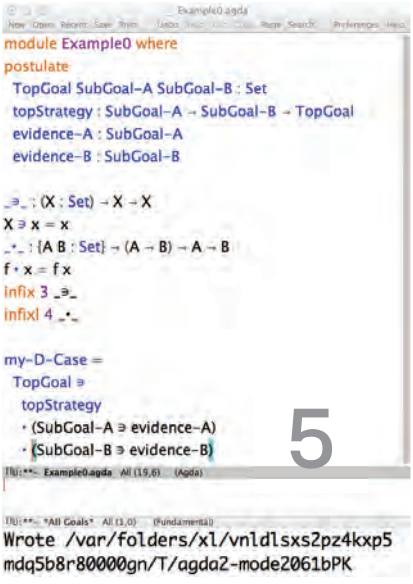
このような不完全な入力しかない状態で検査（構文検査と型検査）を行うと、スナップショット２のようになります。ここでは、 ？ を交えた不完全な記述が、検査を無事通過しています。 ？ はプレースホルダ { } に変わりました。カーソルをプレースホルダの中に持っていき、そこに、このプレースホルダの位置にくるべき項（式）を入力します。



右のスナップショット３は、０番のプレースホルダに項を入力し終わったところです。プレースホルダ０をこの項で置き換えても、構文や型の検査を通過するかどうかを確認するコマンド refine があります。

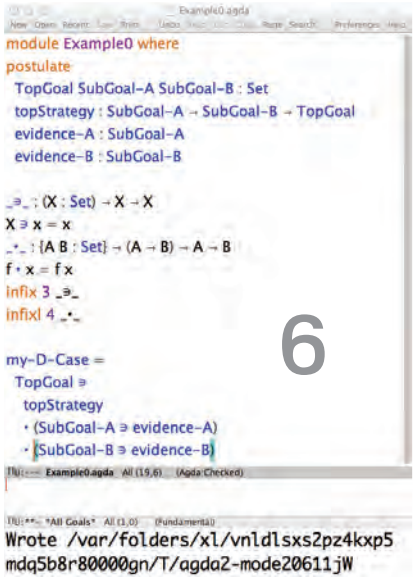


コマンド refine を適用した結果がスナップショット４です。構文および型の検査に通過しました。プレースホルダ０も消えてしまいます。さらにプレースホルダ１にも、項を入力して refine してみましょう。



スナップショット５では、Emacs 下部の小さな窓が空白になり、もう埋めるべきプレースホルダがないことがわ

かります。このようになれば、全体の検査が済んでいます。



念のため最後に全体の検査を行ったのが、スナップショット６です。全体の色が変わります。

Agda システムを用いた、Agda 言語による D-Case の作成は、以上のようにして進められます。

基本的には、テキストを入力しては、検査（構文検査および型検査）を行う、ということの繰り返しですが、一部をプレースホルダとしておいて、詳細の記述を保留し、あとでそこを詳述する、という方法を Agda が支援してくれます。

3.4. Agda 言語による D-Case の記述

ここでは D-Case in Agda の仕組みについて簡単に説明します。詳しい解説は D-Case/Agda のホームページを見てください[6]。

3.4.1. 基本的な原理と構文

はじめにも述べましたが Agda 言語はプログラミング言語です。もうすこし詳しく言うと、型付関数型言語の一つです。この種のプログラミング言語としては、ML や Haskell といった言語が有名ですが、Agda 言語はこれらよりさらに強力な型理論(Martin-Löf 型理論)に基づいてつくられています。Agda 言語は、プログラムだけでなく、高階述語論理という豊かな表現力をもつ論理体系をコーディングできるため、その処理系である Agda システムは、汎用の証明支援系としても使われています。実は D-Case は、論理学でいう「証明木」とよく似た構造をしており、論理をコーディングするのと同じような要領で Agda 言語上に表現して、その作成を Agda システムで支援することができるのです。

Agda による D-Case の表現には「浅いコーディング」と呼ばれる方法が使われています。この方法では、D-Case 中の“主張”を Agda 言語の型(type)として定義します。そして、主張を分解する“戦略”は、その型を返す函数として定義します。分解後の主張達は、この函数の引数達の型になります。また、“証憑”はその型のデータと定義します。本稿の最初の方で、主張を支える議論には、それをいくつかの別の主張に分解するものと、直接証憑で裏付けるものがあると述べましたが、函数やデータは、議論のこの二つの在り様に対応するものです。

議論とは、主張を戦略に基づいて分解していって、最後に証憑に結び付けるものでした。D-Case in Agda は、函数やデータを組み立てた項(term)を与えることで、これを行います。勿論、項は型整合に作らなくてはなりません。これは Agda システムが支援してくれます。

D-Case in Agda の構文を拡張 BNF 記法で定義すると以下のようになります。ここで {…} は、０回以上の繰り返しを表しています。

```
<D-Case>      ::=  <主張> ∋ <議論>
                 |  Context[ <前提部> / <D-Case> ]
<議論>        ::=  <戦略> { ・<D-Case> }
                 |  <証憑>
                 |  Context[ <前提部> / <議論> ]
```

上記文法で使われている３つの二項演算子_∋_, _・_, Context[_ /_]は、D-Case を階層的な項目リストのように記述することを可能にすると同時に、グラフィカルな表現に変換する際の目印にもなります。これらは DCaseSpecShallow.agda というファイルの中で、函数定義されています。

- X ∋ x は x に評価されます。ただし、x の型は X である必要があります。
- _・_は函数の適用です。つまり f・x は ｆ x に評価されます。x の型は f の引数の型である必要があります。
- Context[C / y] は y に評価されます。C は D-Case や議論の中で使う定義や、前提となる事柄などの宣言です。自然言語の文章として、文字列型で書きます。

Context[_/_]の<前提部>には、本来なら、文字列ではなく、Agda 言語での宣言達が書けるのが理想なのですが、技術的に困難であるため、現状ではこのような形になっています。Agda 言語による宣言は、語彙宣言などと一緒に、Agda ファイル中の D-Case 記述に先立った部分に書かれます。

3.4.2. 拡張構文 ― 自然言語表示

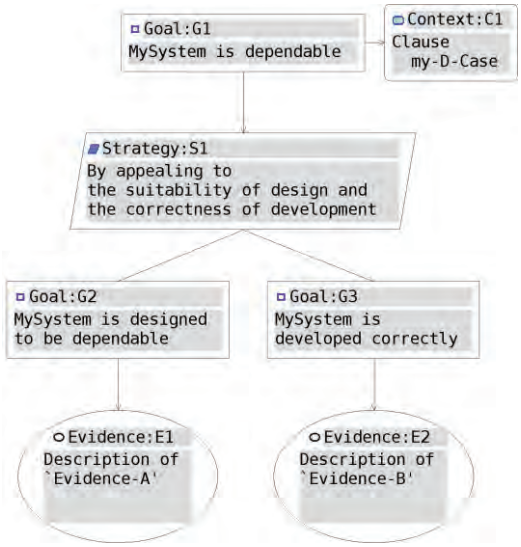
D-Case はシステムに関する合意を助けるツールです。技術者だけでなく、経営者や営業部員、顧客、場合によっては地域住民など、全ての利害関係者の目に触れるものになります。記述の方は訓練を受けた専門家が行うにしても、読むのは一般の人たちですから、図式中の表現には、自然言語での記述を望むでしょう。これは、以下の拡張構文を用いて可能です。

```
<D-Case>      ::=  <主張> ∋ <議論>
                 |  Context[ <前提部> / <D-Case> ]
<議論>        ::=  <戦略> { ・<D-Case> }
                 |  <証憑>
                 |  Context[ <前提部> / <議論> ]
<主張>        ::=  <主張(Agda 言語)>
                 |  《 <主張(Agda 言語)> / <主張(自然言語)> 》
<戦略>        ::=  <戦略(Agda 言語)>
                 |  〈 <戦略(Agda 言語)> / <戦略(自然言語)> 〉
<証憑>        ::=  <証憑(Agda 言語)>
                 |  〈 <証憑(Agda 言語)> / <証憑(自然言語)> 〉
```

つまり、Agda 表現と自然言語表現を〈_/_〉や《_/_》で対にしておけばよいのです。自然言語表現は、文字列として与えます。

下記は前節で用いた例に自然言語表現を加えた D-Case in Agda コードと、それを変換した D-Case Editor 上の図式表現です。ここには自然言語表現のみ示しましたが、変換時には Agda 言語表現のものも同時に作られます。Eclipse のメイン画面で右クリックして、メニューから “flip description”を選ぶことで、二つの表示を切り替えることができます。

```
my-D-Case =
  《 TopGoal / "MySystem is dependable" 》∋
  ⟨ topStrategy / "By appealing to\n\
                  \the suitability of design and\n\
                  \the correctness of development" ⟩
  · (《 SubGoal-A / "MySystem is designed\n\
                    \to be dependable" 》∋
     ⟨ Evidence-A / "Description of\n\
                     \`Evidence-A'" ⟩)
  · (《 SubGoal-B / "MySystem is\n\
                    \developed correctly" 》∋
     ⟨ Evidence-B / "Description of\n\
                     \`Evidence-B'" ⟩)
{-# DCASE my-D-Case root #-}
```



4. おわりに

本稿ではアシュランス・ケースや、その一つである D-Case、およびこれらを 形式言語により記述する D-Case/Agda について解説しました。我々の生活を支えるシステム達は、今後もますます巨大化・複雑化していくと思われます。そのようなものを手懐けて、安全で柔軟な運用を行うために、これらの技術は重要度を増していくでしょう。

現在 DEOS プロジェクトでは、今までの研究成果の実用化を図るべく、DEOS プロセスや D-Case の実証実験に力を入れています。例題としての仮想的なシステムではなく、社会の中で実際に稼働するシステムを対象にすることで、 より実践的な方法論の確立を目指しています。展示会場の案内ロボットや、ミニ人工衛星などの D-Case 作成から、多くの有用な知見が得られつつあります。

我々産総研 RISEC チームでも、この夏、ファイル共有サーバを対象とした D-Case 記述実験に着手しました。このサーバは、我々自身が日々の業務の中で使うものです。可用性や経済性など様々な観点から、ユーザとして D-Case はどのように書かれていて欲しいか、を身をもって体験し、明らかにしたいと考えています。この実験における、我々のもう一つの目標は、D-Case/Agda を用いて、全ての D-Case を機械検査/処理可能な形で作成することです。ファイル共有サーバ自体は巨大なシステムとはいえませんが、D-Case の大きさや複雑

さは、システムのそれらにより決まるわけではありません。システムや、それへの要求に関する情報を沢山盛り込めば 、D-Case は大きく複雑になります。そして、良い情報を盛り込めば、その分 D-Case の利用価値も上がります。そのような高品質の D-Case を精緻に構築しようとするときにも D-Case/Agda の力が発揮されるはず、と期待しています。

また、産総研 RISEC チームでは、本文中で紹介したような国際標準化の活動にも力を入れています。実証実験から得られた知見は、いち早くそれに反映したいと考えています。

参照文献

[1] EAD Safety Case, European Aeronautical Information System, edition 2.3, September 2009

[2] Origin Consulting Limited, GSN Community Standard, Version 1, 2011.

[3] Safety Case Development Manual, European Organisation for the Safety of Air Navigation, edition 2.2, 13 Nov. 2006, http://www.eurocontrol.int/safety/public/site_preferences/display_library_list_public.html

[4] Safety Assessment Made Easier, European Organisation for the Safety of Air Navigation, edition 1.0, 15 Jan. 2010, http://www.eurocontrol.int/safety/public/site_preferences/display_library_list_public.htm

[5] Tokoro (ed), Open Systems Dependability - Dependability Engineering for Ever-Changing Systems, CRC Press, 2012.

[6] "D-Case in Agda" Verification Tool (D-Case/Agda), http://agda.cvs.gr.jp/agda/D-Case-Agda/

[7] **松野裕、高井利憲、 山本修一郎**. D-Case **入門** ~**ディペンダビリティ・ケースを書いてみよう**!. **株式会社ダイテックホールディング**, 2012.