構成管理ツールおよびその運用手順に関するリスク分析とアシュアランスケースの記述事例*

奈良先端科学技術大学院大学†/ 産業技術総合研究所‡
高井 利憲

2012 年 7 月 17 日

概要

ソフトウェアを含むシステムに対してアシュアランスケースを記述した事例実験について報告する。実験の目的は、1) リスク分析作業の妥当性確認手段としてのアシュアランスケースの有効性の確認、および 2) アシュアランスケース記述プロセスの確立である。実験を通じて 1 に関しては、従来のリスク分析手法では十分ではなかった網羅性の確認手段としてアシュアランスケースが有効であるとの観察を得ることができ、また、2 に対しては、関係者の役割分担を含む構築プロセス案を得ることができた。

1 目的と背景

代表的なリスク分析手法には、FMEA や FTA、HAZOP などが知られており、これらはソフトウェアを含むシステムに対しても適用されている。特に、組込み機器向けソフトウェアの開発においては、FMEA ベースのリスク分析手法がしばしば使われている。FTA が、システムが引き起こす事故やシステムの障害からトップダウンにその原因を分析する手法であるのに対し、FMEA は、システムの各部品にたいして、ありうる故障のパターンを考えることから始めるボトムアップの分析手法である。ここで、部品の故障のパターンを故障モードという。ハードウェアの部品に関しては、「摩耗」「酸化」「破損」「付着」など、どのような故障モードを考慮すれば漏れがないのかが確立している場合が多く、その網羅性が議論になることは多くないが、ソフトウェアが対象の場合は事情が異なる。まず、ソフトウェアの部品としてなにを採用するかの可能性が複数あり得る。要求レベルから実装レベルまで、また、オブジェクト指向的視点での分解か、データ処理的視点か、など、全てを考慮の対象にすることは不可能であり、また、どれを選択しても恣意的にならざるを得ない。さらに、ソフトウェアの部品のレベルが決まったとしても、その故障モードの分解の粒度も任意になりうる。例えば、あるデータ処理の故障を考えた場合、そのアウトプットの間違え方として、常に間違えている場合やたまに間違える場合、アウトプットが遅れる場合、大幅に遅れる場合など様々考えられるが、それらを分析の初めから全てを考慮することはできないため、どこかで粒度を決める必要があるが、それが妥当であるか否かは実際に事故が起きて初めて妥当ではなかったことが判明することも多い。以上のような事情は、HAZOP でも同

* 本研究は、科学技術振興機構の CREST 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 (DEOS)」研究領域における産総研木下チーム「利用者指向ディペンダビリティの研究」の一環として、株式会社東芝と共同で実施された。

† 著者の現在の所属は、奈良先端科学技術大学院大学である。

‡ 本研究は、著者が産業技術総合研究所に所属していた 2011 年度に実施された。

様である。HAZOP では、ガイドワードと呼ばれる語彙集をもちいて、システムの望ましい振る舞いからの逸脱を列挙し、その影響や原因を分析する手法であるが、ガイドワードの網羅性につての保証はなく、分野ごとに標準的なものが用意されており、それを利用しているのが実情である。本事例では、リスク分析作業工程の妥当性確認手段としてアシュアランスケースを採用し、そのフィージビリティを確かめた。

アシュアランスケース (assurance case)[1] は、システムの安全性などリスクに関する性質について、システムの関係者間で合意形成を図るための文書のことをいい、安全関連分野では、セーフティケース (safety case) とも呼ばれ、既に国際標準規格などではその提出が義務づけられているものもある [2]。アシュアランスケースに含まれる主な情報は、システムについて、主にリスクに関係する性質を記述する主張 (claim)、主張が成立することを示す証拠 (evidence)、どのようにそれら証拠が主張を成り立たせているかを表す議論 (argument) の三つであり、その表記法として、GSN(goal structring notation)[8] などがよく使われており、議論についての典型的なパターンについては、標準化も試みられている [11]。GSN は木構造でアシュアランスケースを記述するもので、そのノードには、ゴール (goal)、戦略 (strategy)、証拠 (evidence) などがある。さらに、その構築プロセスについてもいくつか提案されている [9, 10]。しかし、これらは汎用的なガイドラインであるため、アシュアランスケースの記述を開発プロセスにおいて「誰が」「どのタイミングで」「アシュアランスケースのどの部分を」書くのかまでは指定しておらず、実際アシュアランスケースを採用しようとしているソフトウェアの開発現場にとっては、必ずしも十分な情報を提供していない。

本事例を始めるにあたって、アシュアランスケースを中心とした、システム開発に関わる関係者のロールを含むリスク分析工程のプロセスを仮定した。また、事例を通じて仮定したプロセスを詳細化した。また、アシュアランスケースを記述するにあたり、アシュアランスケース記述支援ツールである D-Case/Agda[4] を利用した。D-Case[5] は、GSN をサポートしていくことに加え、通常のアシュアランスケースのように、開発時にシステムのリスクに関して記述するだけでなく、運用時にも記述した内容が保たれているかを監視したり、システムの運用時の特定の振る舞いについて、アシュアランスケースの記述をそのまま参照し、実行することができるなど、運用時にアシュアランスケースの記述と実際のシステムとの乖離を極力少なくする機構を含むツール群の総称である。また、Agda[7] はもともと、定理証明支援系および仕様記述言語であるが、D-Case ツール群の中においては、アシュアランスケースの記述言語およびその整合性検査機能を提供する。本事例でも、アシュアランスケースは Agda を通じて記述した。

2 アシュアランスケース記述プロセス

本事例で得られた知見をもとに提案する、リスク分析工程を含むアシュアランスケース記述プロセスは以下の通りである。事例を始める際は、これより大まかで、細部が異なったものを仮定していたが、事例を通じて詳細化および改良を加えた。図 1 に、構築プロセスの概要を表すデータフロー図を示す。以下、アシュアランスケースの記述対象とするシステムを移行ではアイテム (item) と呼び、利害関係者によって設定されたアイテムが満たすべき性質を目標 (goal) とよぶ。また、ハザードは、目標の破綻に至りうるシステムの状態を表す。

Step 1　アイテム定義：設計担当者が実施。

1. 記述対象となるアイテムの確認
2. アイテムに関する既存の目標確認
3. 語彙体系の抽出

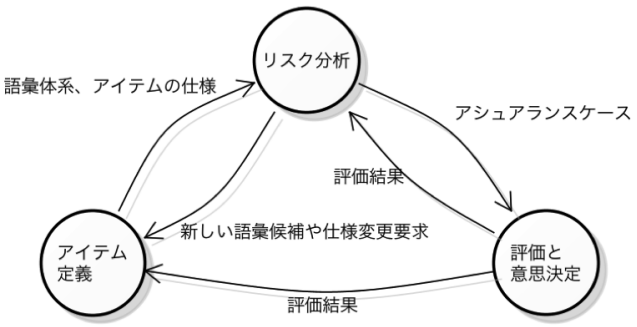

図 1　アシュアランスケース構築プロセスの概要

4. 目標を達成しようとしていることを示すことができる程度のアイテムの仕様記述。必要に応じて、語彙体系を改訂。

Step 2　リスク分析：リスク分析担当者が実施

1. 定義したアイテムが仕様通りに振る舞った場合の分析
   (a) Step 1 で定義した仕様が目標を達成することの確認
   (b) 確認できなければ、アイテムに必要な修正を加えるため、アイテム定義に戻る
   (c) どのように確認したかをアシュアランスケースに記述
2. 仕様（1.(d) 参照）から逸脱した振る舞いをした場合の分析
   (a) Step 1 で定義した仕様からの逸脱としてのハザードの抽出
   (b) 抽出の網羅性に関する根拠をアシュアランスケースに記述
   (c) 抽出したハザードの原因と影響の分析
   (d) 語彙が足りなければアイテム定義にもどり語彙を追加
   (e) ハザードに対する目標およびその重要度を設定
   (f) ハザードに対して、目標を実現するための対策案の策定
   (g) 対策案の効果と影響の分析
   (h) 語彙が足りなければアイテム定義にもどり語彙を追加
   (i) 対策案が対策になっていることをアシュアランスケースに記述
3. 目標を達成できなかった場合の分析（事後対応の評価）

Step 3　評価と意思決定：意思決定者が実施

1. アシュアランスケースが前提としているアイテムの仕様を評価
2. アシュアランスケースの論証の進め方の評価
   (a) 議論の構造について評価
   (b) 網羅性の根拠について評価
3. 対策案の評価
   (a) アシュアランスケースによる評価。以上の評価において、納得がいかない場合には、それを指摘し、必要なところへ戻る。
4. 対策案の中から採用する対策を決定

ここで、「設計担当者」「リスク分析担当者」「意思決定者」は、システム開発に関わる関係者のロールであり、これらが同じ組織に所属しているか、同一人物が担当しているかはここでは問わない。以下のような想定をしている。

設計担当者　実際にシステムの開発を担当する技術者であり、システムの設計や実装についての情報をもつものを想定する。システム開発が、一組織内で行われている場合には、開発部門が相当する。

リスク分析担当者　システムの安全性やセキュリティ、信頼性、品質などを担当する技術者であり、システムのリスクについての情報をもつもの想定する。システム開発が、一組織内で行われている場合には、組織の品質管理部門などが相当する。

意思決定者　システムに対して予算権限をもつような意思決定が可能なものを想定する。システム開発が、一組織内で行われている場合には、経営層が相当し、システム開発に受発注が伴う場合には、発注者が相当する。

上述三者はそれぞれ、システムそのものに関する情報を持つもの、システムのリスクに関する情報を持つもの、コストやマーケット、顧客の事情など、システムが従うべき制約についての情報を持つものと抽象的に考えることができるため、今回の事例に限らずある程度普遍的に適用することができると考える。

また、上記手続き中には明示していないが、任意のステップにおいて手戻りを許す。ここで、以下のように手戻りを分類する。

サイクル　アシュアランスケースの評価による手戻り

ラウンド　語彙の変更を伴う手戻り

語彙の変更を伴わない手戻りももちろん発生するが、ここでは特に名前はつけない。

2.1　記述プロセスの原理

ここで提案している記述プロセスの背景となっている原理は以下の通りである。

語彙体系ベース　アシュアランスケースの最大の目的は、利害関係者間でのリスクコミュニケーションである。ここで、コミュニケーションの第一の障害となるのが、利害関係者間で、語彙や概念を共有していることを期待できないことである。そこで、用語やその概念の共有を目的として、記述プロセスの初めに語彙体系を作成し、また、記述を通じて、語彙体系を維持管理する。基本的に、システムの記述やリスク分析は、管理している語彙体系に含まれる語彙のみを使用することとし、リスク分析で詳細な原因の記述をしようとしたり、抽出したハザードに対する対策で新たな安全機能が必要な場合など、それを記述するのに新たな語彙が必要な場合にはに、語彙体系の管理も同時に行う。また、今回採用したアシュアランスケース記述言語である D-Case/Agda は、仕様記述言語でもあるため、語彙体系の記述も可能である。従って、語彙体系からアシュアランスケースの主張や議論の記述まで、一貫してかつ整合性を保った記述が可能である。

正常時のリスク分析　FMEA や HAZOP を使ったリスク分析では、対象システムが期待しない振る舞いをした場合の影響や原因を分析する。ここではしばしば、システムが期待したとおりの振る舞いをした場合、つまり正常時に、事故や障害を起こさないことは自明のこととしてその分析が省略されることがある。しかし、ソフトウェアを含むシステムでは、機能安全を実現する機能が、仕様通りに振る舞った

場合に、期待する安全性を実現できるか否かは、自明ではない。そこで、提案プロセスにおいては、まず、目標とする安全性を、システムの中でどのように実現するのかを記述できる程度のシステムの仕様を記述し、それが実際目標を達成していることを確かめる作業を含めている。このモデルのことを、リスク分析やアシュアランスケースでの議論の土台となるという意味をこめて、以降では、ベースモデル (base model) と呼ぶことにする。ここで、具体的な確かめ方については、プロセスでは指定しないが、その確認作業の論証の進め方をアシュアランスケースに記述し、評価可能な情報を残しておくこととする。

モデルベースリスク分析　今回の実験は、リスク分析作業の主に網羅性についてその根拠を示したいという動機があった。そこで、記述プロセスにおいては、リスク分析のために、まず対象となるシステムが、安全目標などの満たすべき性質をどのように満たそうとしているのかを確かめられる程度の仕様を書き (これは、上のベースモデルのこと) その仕様からの逸脱の仕方を網羅性を保ちながら分類することとした。ここで、どのように網羅性を考えたかをアシュアランスケースで記述しおき、評価可能な情報を残しておくこととする。以上のように、アシュアランスケースを記述する際には、まず議論の対象を、議論に耐えうる程度に明らかにしておくことが必要である。

各ステップの詳しい説明は、次節において事例に沿って説明する。

3　記述事例

はじめに、報告する事例での対象システムの概要を示す。次に、前節で紹介した記述プロセスの各ステップのうち、重要なものを事例を用いて説明する。

3.1　対象システム

今回の事例でアシュアランスケースを記述したの対象システムを簡単に説明する。主要な対象は、構成管理ツールである Subversion(SVN)[3] であるが、より正確には、SVN の利用を含むある組織の開発プロセスである。具体的には、Subversion の他には、各開発者が遵守すべき SVN の運用ルールなどのほかに、ここでは開発者自身も対象システムに含まれると考える。保証すべき性質は、そのような開発プロセスによって、誤ったコードがリリースされないことという安全性となる。対象となる開発プロセスは、ある組織で実際に運用されているものを参考にしているが、特殊な用語などを一般的な用語に置き換えるなどしていることを含め、抽象化している。ここでは、運用手順に関して以降に必要な用語だけ導入しておく。

各開発者のローカルの作業フォルダをワーク環境とよぶ。また、プロジェクト一つに、一つ特別なローカルの作業フォルダとしてメイン環境と呼ばれるものを用意しておき、そこは常に最新のリビジョンのフォルダ構成に維持しておく。開発の作業単位は「対応」とよばれ、プロジェクトにおいて、プロジェクト管理者が必要な対応を各開発者に割り当てる。各開発者は、対応が割り当てられると、まずはワーク環境でそれを実現し、テストなどを実施した後、リポジトリへのコミットを行い、さらにメイン環境をアップデートすることにより、開発対象システム全体のテストを実施する。テストが通過すれば、対応が終了した旨プロジェクト管理者に報告し、一つの対応が終了する。運用ルールは二つあり、開発者がプロジェクトに参加する際のものと、上記で述べた、対応が割り当てられた際のものである。

3.2 アシュアランスケース記述言語およびツール

本記述実験事例では、アシュアランスケースを記述するにあたって D-Case/Agda を使用した。D-Case は、GSN をサポートするアシュアランスケース記述言語およびその記述支援ツール群の総称である。GSN は、アシュアランスケースの構成要素である主張、議論、証拠を木構造に構造化して表現する記法を提供するものであり、図 2 に例を示す。

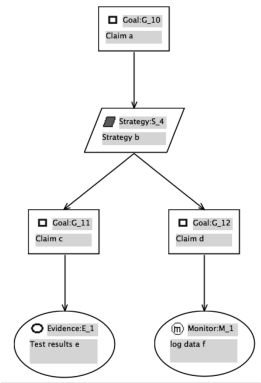

図 2　D-Case の記述例

ここで、長方形のノードは、ゴールとよばれ、主張を表し、菱形のノードは戦略と呼ばれ、議論の展開を記述する。また円の葉は証拠を表す。ここで、ゴールノード G_10 は、戦略ノード S_4 によって、ゴールノード G_11 およびゴールノード G_12 に分割されたと見なされるため、G_11 および G_12 を G_10 の部分ゴールと呼ぶ。

3.3 Step 1: アイテム定義

まず、分析対象となるアイテムの確認においては、記述対象のシステム境界を定める。今回の場合は、構成管理ツールを中心とする開発プロセスが対象であり、構成管理ツールやその運用ルールは対象とするが、開発に用いるコンパイラなど他のツールは対象外であることなどを確認した。また、最初のアイテムに関する既存の目標は、特に厳密性を求めず、次のようにした。最初のサイクルでは、以上のステップは、既存の開発関連資料などに基づき実施する。

> 最初に設定した目標: 誤りを含むコードがリリースされることはなく、また、誤りを含まないコードは速やかにリリースされる。

ここで仮決めした目標は、以下、リスク分析などを通じて随時改訂していく。つぎに、語彙体系の構築である。今回の事例では、語彙は大きく名詞と動詞にわけ、それぞれ UML のクラス図で記述した。例えば、図 3 において、「ローカル保管場所は、ワーク環境やメイン環境を一般化した概念であり、その構成要素としてフォルダ構成および管理情報を持つ」ことを表している。用いた辺は、`is-a` 関係を表す`<is-a>` と `has-a` 関係を表す`<has-a>`など一般的なものだけである。より精密なオントロジー記述体系を採用しても良いかもしれないが、今回のアシュアランスケースの記述言語である D-Case/Agda において集合と関数で記述できるものを採用し

ている。属性やメソッド、多重度については基本的に使用しない。次に、アイテムが目標を達成できることを

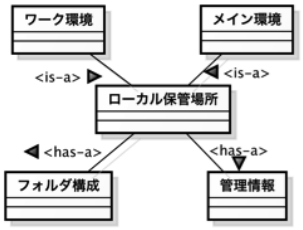

図 3　名詞の語彙の記述例

示す程度のアイテムの仕様記述である。ここでの目的は、まずリスク分析やアシュアランスケースでの議論の対象となるアイテムを明らかにすることである。対象について共通の理解がなければ、議論は成立しない。ただ、ここで、何の基準もなくシステムの仕様記述をしてしまうと、通常の開発プロセスにおける設計などと変わらなくなってしまう。そこで、ここではアイテムのなかで、目標を達成するための仕組みを記述する。リスク分析においてはもちろん、このように記述したアイテムの部分以外の部分が原因となって発生する障害や事故も存在するが、少なくとも目標が破綻するに至る過程には、ここで定義する「目標を達成するための仕組み」部分についての逸脱も発生すると考えられるためである。また、当然そうならない想定外の障害や事故も発生しうるが、その場合は「目標を達成するための仕組み」についてアシュアランスケースを通じて合意した利害関係者全てが責任を共有することとなる。事例で説明する。今回はまず、先に用意した語彙体系のクラス図を用いたオブジェクト図によってアイテムの全体像を記述した。これは、オブジェクト図に限定しているわけではなく、利害関係者間で合意可能な記述方式を採用すればよい。ここで開発者は二名しか現れないが、実際に

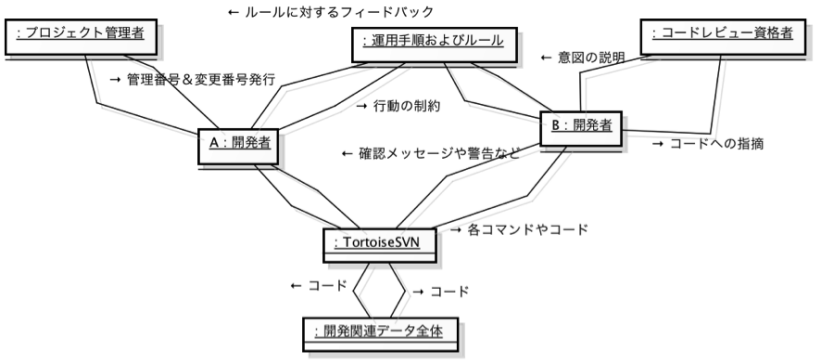

図 4　アイテムの全体像

は任意の人数の開発者が参加する可能性があることは関係者間で合意しておく。この図に現れる各オブジェクトが、個のシステムの一番大まかな部品となる。また、各部品についても、「最初の目標」が達成できる程度まで中を記述しておく。例えば、運用ルールについては次のようなアクティビティ図によって記述した。

さらに、Subversion(図 4 中では TortoiseSVN) については、運用ルールに現れる各コマンドの仕様を表によって記述した。例えば、開発者が変更を加えたローカルのファイルをリポジトリに反映させるコミットコマンドの仕様は図 6 に示した通りである。基本的には、コマンド実行に対する事前条件と事後条件を記述しておく。ここでは省略しているが、リポジトリや各開発者のもつワーク環境などの状態を表す、アイテム全体のの

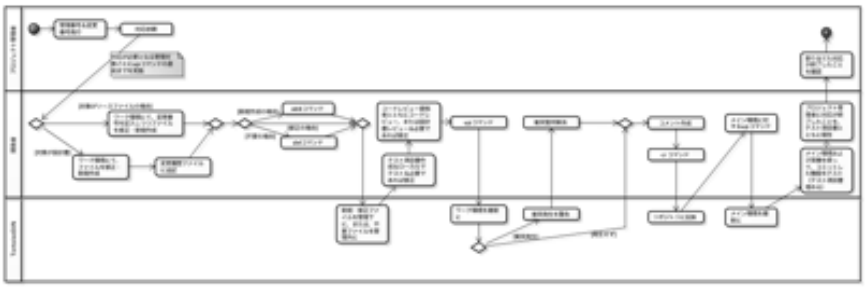

図 5　運用手順の一部

状態集合についても事前に定義している。以上のアイテムの定義は、リスク分析やアシュアランスケースでの

コマンド名：SVNコミット（CI）			コマンド実行前	コマンド実行後	
リポジトリの状態	最新のリビジョン番号		N	N'=N+1	リビジョン番号が1つ更
	フォルダ構成		fr(n) (n<=N)	fr'(n)=fr(n) (n<=N), partCO(fr'(N))=fl	1つ前のリビジョンまで 変わらない。最新のリ た部分のフォルダ構成 になる
	フォルダ中の誤りの有無 （bool）		efr(n) (n<=N)	efr'(n)=efr(n) (n<=N), efr'(N)=chkefr(efl) ‖ efr'(n)=efr(n)	1つ前のリビジョンまで 誤りの有無による。最 ダ中の誤りの有無とコ の有無による。
ローカルの状態（実行者の） ※実行者以外の人の状態に変化は無い	フォルダ構成		fl	fl' = fl	コマンド実行前のフォ
	管理状態（bool）	修正ファイルの有無	em	–	不確定
		管理外のファイルの有無	eo	eo' = eo	コマンド実行前の管理
		競合状態ファイルの有無	ec	ec' = FALSE	競合状態のファイルは が存在する場合はコミ
	管理対象ファイルのフォルダ構成		p	p' = p	コマンド実行前の管理 わらない
	フォルダ中の誤りの有無 （bool）		efl	efl' = efl	コマンド実行前のフォ

図 6　コミットコマンドの仕様記述

議論の土台となるもので、今回の事例のベースモデルであり、議論が進むにつれて改訂していく。例えば、リスク分析の過程で目標が達成できないことが判明すれば、目標が達成できる仕組みに変更しなければならなかったり、アシュアランスケースの評価の過程で、アイテム定義に基づく議論の展開に納得が得られなければ、議論の変更に伴いアイテム定義も変える必要が生じる。今回は、図 4 の全体像の変更はなかったが、最終的に、図 5 の運用ルールについてはは三回の改訂、Subversion の各コマンドの仕様については一回の改訂の必要があった。

3.4　Step 2: **リスク分析**

3.4.1　**定義したアイテムが仕様通りに振る舞った場合の分析**

リスク分析は、ISO26262 でいえばハザード分析（hazard analysis）に相当する工程であり、ここでは、FMEA や HAZOP によるリスク分析のように部品の振る舞いの逸脱についての議論の前に、「アイテムが仕様通りに振る舞った場合に目標を達成することの確認」の作業を行う。ハードウェアなどのリスク分析では、システムが仕様通りに振る舞えば目標が達成されることは自明のこととして議論の対象にならないこともしばしばあるが、ソフトウェアの場合にはそれは自明ではなく、何らかの方法でそれを示さなければならない。例えば、対象システムの形式的なモデルを構築し、その上で、モデルが目標を達成することを形式的な証明で示すなどである。例えば今回の事例の場合は、複数の開発者が、図 5 のような運用手順に従って独立に、かつリポジトリを通じて相互作用しながら動いている。従って、形式的に扱おうとすれば、任意個のプロセスが並行して実行し、インタラクションしているモデルで、目標が達成していることを形式的な定理証明手法によって証明しなければらない。今回アシュアランスケース記述言語に採用した Agda は、定理証明支援系としての機

能も持つため、原理的には以上のようなアプローチも可能であったが、今回は、実験の期間等のコストも勘案し、より単純な場合分けによる証明を、D-Case/Agda 上で命題論理の範囲で記述することにより、議論を展開した。以上の判断についてもアシュアランスケースに記述しておき、意思決定の際に評価できるようにしておく。

場合分けは具体的には、

- 誤りコードの修正に対するコミットまでに他の開発者のコミットがあった場合となかった場合
- 誤りコードの修正に対するコミットまでに他の開発者の更新があった場合

の二つの条件を考え、四つの場合についてそれぞれ検討した。以下、主張における連言や、議論おける、連言の分割、などの命題論の範囲における議論の展開の仕方について Agda での記述例を示しておく。

```
_かつ_ : Set → Set → Set
A かつ B = A × B

それぞれ示す : {A B : Set} → A → B → A かつ B
それぞれ示す a b = a ,′ b

論証付き仮定の導入 : {A B : Set} → (A → B) → A → B
論証付き仮定の導入 f a = f a
```

上の議論の使い方を示す例を次の図で示す。この例では、目標である「A かつ B」を示すためには、この木で

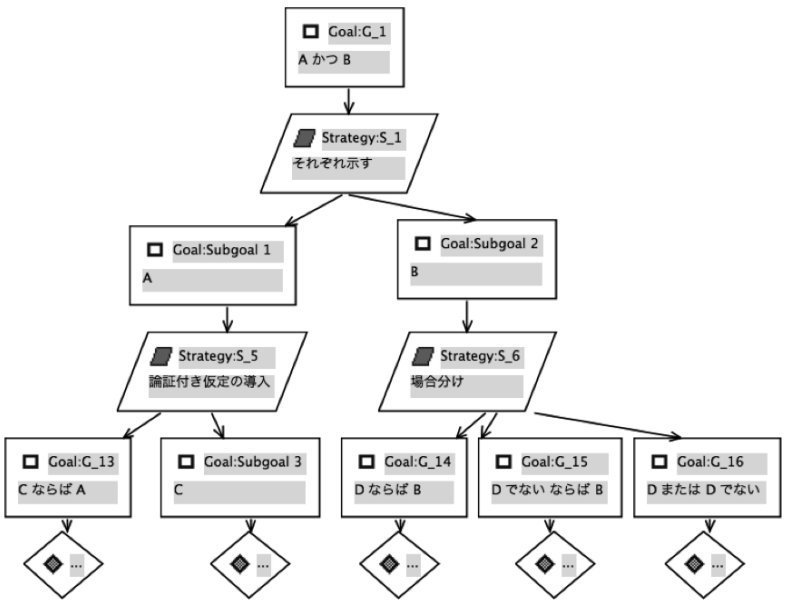

図 7　Agda による形式的な議論を用いたアシュアランスケースの例

展開された下位の目標をすべて示せば良いことがわかる。命題論理の範囲でもこのような形式的な議論を導入することにより、議論の展開についての健全性がある程度保証される。

以下に、事例実験で記述した対応するアシュアランスケースの一部を示す。

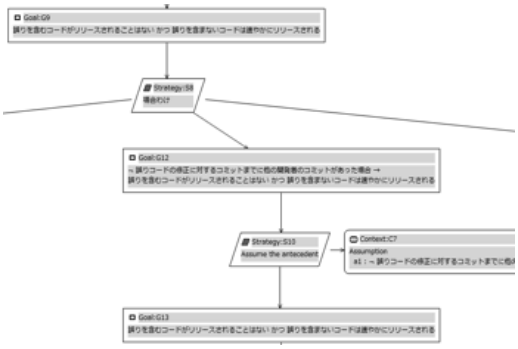

図 8　アシュアランスケースの一部

対応する Agda での記述を図 9 に示す。

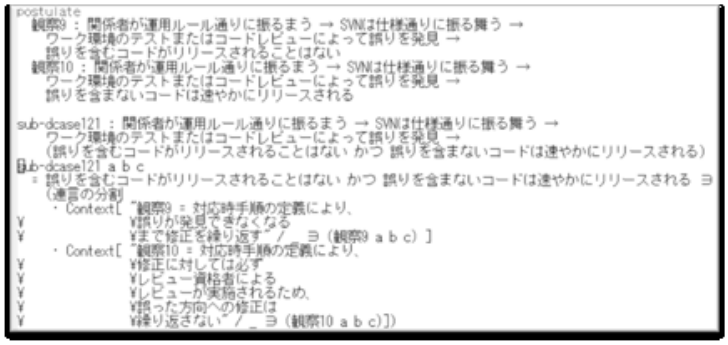

図 9　Agda によるアシュアランスケースの記述例

このようなアシュアランスケースの記述の過程で、誤りコードの修正中に他の開発者からコミットや更新があった場合に、誤ったコードが気づかれずにリリースされる可能性があった。具体的には、開発者は、自らの修正をコミットした後に、開発プロジェクトに関係するすべてのフォルダ構成のリファレンスとしての役割を持つメイン環境を更新し、その上でテストを行い、自らの修正と他との整合性を確認する。しかし、アシュアランスケースを記述する過程で、メイン環境でのテストで誤りが発見した場合の場合分けとして、その誤りを修正するまでに、他の開発者が、別のコミットをしたり、その誤りを含むリビジョンによってアップデートを実施したりする場合が考えられた。これらの場合は、誤ったコードを前提に他の開発者が開発を進めてしまう恐れがある。改めてインタビューすると、メイン環境に対するテストで誤りが発見され、かつその部分が、他の開発者の担当部分と重なる恐れがある場合には、修正を戻したものを改めてコミットしておくという運用をしていることが判明した。そこで、この運用ルールを反映したアクティビティ図を改めて作成し、それに対するアシュアランスケースの記述を進めた。実験では、改訂版のアクティビティ図もまだ目標を達成できないことが確認されたため、もう一度改訂作業を実施した。最終的に、記述した範囲の運用手順によっては目標が達成されることを確認した。ただし、これはある意味理想的な環境での振る舞いであり、また、議論した仕様は様々な面で抽象化されていることに注意しておく。

また、リスク分析の過程を通じて、アイテムの目標についての厳密な記述が定まってくる。最初の目標は次

のように改訂された。

> 改訂版の目標: (関係者が運用ルール通りに振るまう ならば誤りを含む対応関連ファイル群がリリースされることはない かつ 誤りを含まない対応関連ファイル群は速やかにリリースされる) かつ すべての逸脱に対して十分対策がとられている かつ (各開発者が運用ルール通りに振る舞っていれば リリース後、判明した誤りに対して 誤りの原因に対する変更番号と変更者名が特定できる)

ここで、三つの命題の連言になっており、それぞれを部分目標 1 から部分目標 3 とよぶ。部分目標 1 は、定義したアイテムが仕様通りに振る舞った場合の分析の部分に対応する。

3.4.2 仕様から逸脱した振る舞いをした場合の分析

ここでのリスク分析はまず、FMEA のように、評価対象の構成要素ごとにその故障モードを想定し、それら故障モードの影響を分析するものであるが、ここでは構成要素として部品ではなく、HAZOP のように操作の方とし、操作のクラスに含まれる操作の集合の中で、期待されない部分集合を逸脱 (deviation) と呼び、逸脱に関して分析を行った。逸脱は、ISO26262 におけるハザードに相当する。また、逸脱を考える操作のクラスは、FMEA のように細かい部品から始めるのではなく、大まかな単位から必要に応じて小さなクラスへと分析を進めた。作業は基本的に、定義したベースモデルに沿って逸脱を列挙する。例えば、図 6 で定義したコミットコマンドに対する逸脱は次のように列挙する。

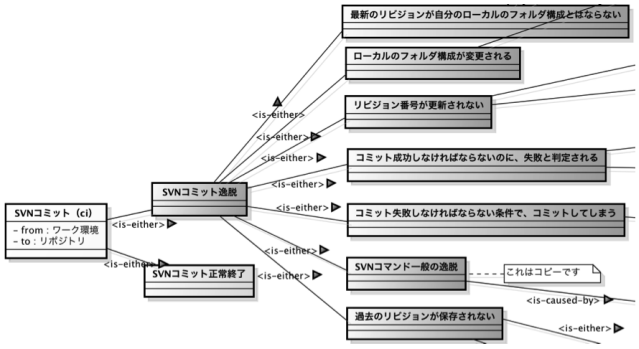

図 10　コミットコマンドのリスク分析例

ここでのクラス図にはその網羅性についての根拠について明示していないが、これを構造的に明示するのがアシュアランスケースである。仕様から逸脱した振る舞いをした場合の分析は、改訂版の部分目標 2 に対応するものであり、その部分のアシュアランスケースの上位構造を次の図 11 で示す。

ここで、部分目標 2 の議論は、「網羅性」と言ったときの枠組みそのものについての議論を左側の部分ゴール (図中の Goal: G_47) で行い、その枠組みで実際網羅性があるか否かは右側の部分ゴール (図中の Goal: G_56) によって議論している。ゴール G_56 の証拠には、網羅性のチェックのために、上で述べたようなクラス図によるリスク分析の結果から抽出した情報によって構成した FMEA の表も採用している。すなわち、FMEA による網羅性の確認は、確立された網羅性の枠組みにおいて、実際リスク分析作業が網羅性を満たしているかを確かめるのには有効だからである。

次に、列挙した逸脱について、その原因や影響、対策についても同じクラス図上でイベントツリーのように辺でつないで記述していくことにより、分析を進める。辺の種類は、以下のようなものであり、それぞれ上か

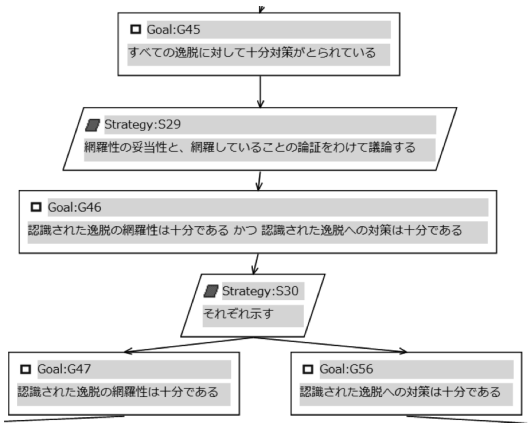

図 11　網羅性の妥当性と網羅していることをわけて議論

ら、包含関係、原因、システム全体に及ぼす影響、リスクに対する対応策であることを表す。

```
_<is-either>_ : Set → Set → Set
A <is-either> B = A → B
_<is-caused-by>_ : Set → Set → Set
A <is-caused-by> B = A → B
_<results-in>_ : Set → Set → Set
A <results-in> B = A → B
_<is-mitigated-by>_ : Set → Set → Set
A <is-mitigated-by> B = A → B
```

ただし、木構造である必要はなく、グラフとして自由に記述できる。また、原因と結果の階層は任意でよい。FMEA のように、原因や影響についての認識の統一の必要はない。例えば、開発者がプロジェクト参加時にリポジトリから必要なファイルをローカルへコピーする作業の際利用するコマンドであるチェックアウトに対するリスク分析は、図 12 のようになる。

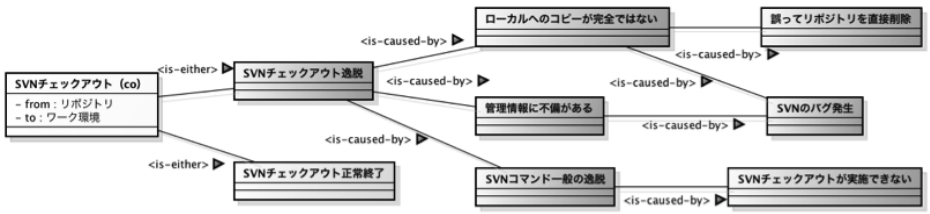

図 12　チェックアウトコマンドに対するリスク分析

ここで、木構造ではなく、任意のグラフ構造を許してしまうと、抽出した逸脱が全て網羅的に対策がとられているか否かが見えにくくなるという懸念があるが、今回この部分は Agda の定理証明支援系としての側面を利用し、任意の逸脱に対して、`<is-either>` や `<is-caused-by>`、`<results-in>`、`<is-mitigated-by>` などで結ばれた対策や、システム全体に大きな影響を与えないことを表す影響に到達可能であることを示す

こととした。ただし、今回の事例実験ではこの部分は記述するコストが非常に大きかったため、一つの逸脱「TortoiseSVN 使用ガイドに反する日常業務」に関する論証のみを Agda で記述できることを確かめて、他は省略している。ただし、この部分は単純なグラフの到達可能性問題であるので、ある程度自動化できると考えられる。

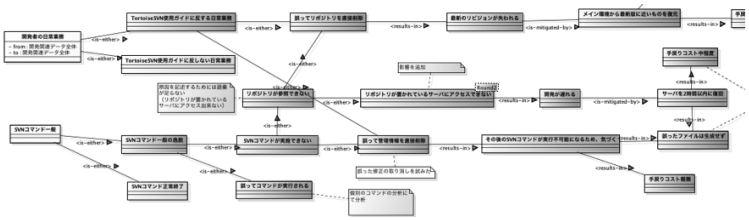

図 13　チェックアウトコマンドに対するリスク分析

リスク分析作業のなかで、原因を記述したり、対策を記述する際に当初定義した語彙では不足する場合がある。その際は、その部分の分析は一旦おいて、一通り用意した語彙で記述できる範囲を記述した後で、語彙を追加する。実験では、次のような語彙を導入した。

- 名詞（オブジェクト）：ハードディスク容量、サーバ、チェックボックス、テスト項目書、テスト結果、レビュー結果
- 動詞（操作、機能）：チェックボックスチェック、コピーによる変更、レビュー報告書添付、レビュー報告書確認

これらの語彙を用い、一巡目で逸脱や対応策が記述できなかったところを埋める。例えば、はじめ、開発者がローカルのファイル群に実施する「修正」の逸脱のサブクラスに、「SVN 変更の逸脱」があったが、その分類について語彙が足らなかったため記述できなかった。そこで、それを記述するために、「コピーによる変更」という語を二巡目の最初で導入したうえで、逸脱の分析と対応策の決定内容を記述している。

3.5　Step 3: 評価と意思決定

意思決定者は、アシュアランスケースによって、リスク分析の妥当性やリスクに対する対応策候補について意思決定を行う。ここで、意思決定者が評価すべきものは、アシュアランスケースそのものだけではなく、その土台となっているベースモデルも含まれる。ベースモデルを記述する語彙体系の抽象度について、意思決定者と認識が同じであるかや、故障モードなどの逸脱の列挙の根拠となったアイテムの仕様記述の妥当性についても評価する。今回の事例では、アシュアランスケースが前提としているアイテムの仕様の部分について、部分チェックアウト等は抽象化されているが、その抽象化が妥当かどうか、疑問があると指摘される。また、誤りを含まない対応関連ファイル群は、速やかにリリースされる」の部分の論証が見えにくいことを指摘された。それを受け、Subversion のコマンドの仕様定義を部分チェックアウトを含むものに変更し、サイクルを 2 巡することとなった。

4　まとめと考察

今回のアシュアランスケース記述事例によって、アシュアランスケースを通じたソフトウェアに対するリスク分析手法についていくつか知見が得られた。ソフトウェアはハードウェアと違い、確立した部品への分割方法や部品ごとの故障モードのリストのようなものはないため、そのリスク分析は恣意的になることがおおい。担当者が異なると、全く異なる視点での分解や粒度での分析となることもある。また、リスク分析で対象とする部品の粒度はソフトウェアにおいては、ソースコードのコマンド一つも部品になり得るため、任意に詳細にはできない。しかし、どこまで考慮すべきかその妥当性を確認することも困難である。今回の事例ではまず、ソフトウェアの仕様に関して、どのようにリスクに対処しているのかの構造を説明できる程度に記述した。本報告ではこれをベースモデルと呼んだ。これは、事故モデルとして提案されている Leveson の STAMP(Systems-Theoretic Accident Model and Processes)[12] とも考え方は近い。リスク分析は、そのベースモデルをベースに議論を進める。つまり、何をもって「網羅性」というのかの根拠を維持管理しながら分析を進める。ここで、アシュアランスケースは、リスク分析作業がベースとしている根拠を記述するメディアとして機能することが確認できた。リスク分析で実施した作業の根拠とその議論の過程を構造化して記述しておくことにより、リスク分析の結果を使う人間がその妥当性について根拠を伴った評価が可能になる。

また、今回のアシュアランスケースのトップレベルの構造は次の図の通りである。これは、ある程度汎用的にアシュアランスケースを記述する際のガイドとして利用できるのではないかと考える。

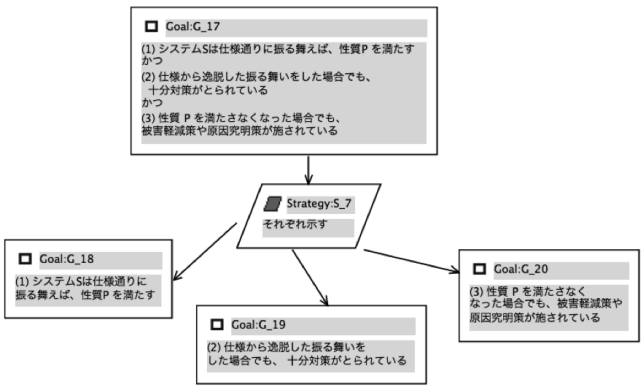

図 14　今回記述したアシュアランスケースのトップレベルの構造

謝辞

本研究は、JST のプロジェクト「実用化を目指した組込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-CREST)」において、株式会社東芝と産総研との共同で実施されたものである。関係者の方々に感謝の意を表します。

参考文献

[1] ISO/IEC 15026-2:2011, Systems and software engineering – systems and software assurance – Part 2: Assurance case, 2011.

[2] ISO26262-1: 2011, Road vehicles – Functional safety, Part 1: Vocabulary, 2011.

[3] C. Michael Pilato, Ben Collins-Sussman, Brian W. Fitzpatrick: *Version Control with Subversion*, O'Reilly Media, Second Edition, 2008.

[4] D-Case/Agda download page.
`http://wiki.portal.chalmers.se/agda/pmwiki.php?n=D-Case-Agda.D-Case-Agda`

[5] Yutaka Matsuno, Jin Nakazawa, Makoto Takeyama, Midori Sugaya and Yutaka Ishikawa: "Toward a Language for Communication among Stakeholders", *Proc. of the 16th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC'10)*, pp. 93–100, Dec 2010.

[6] D-Case Editor: A Typed Assurance Case Editor.
`http://www.il.is.s.u-tokyo.ac.jp/deos/dcase/`

[7] The Agda Wiki. `http://wiki.portal.chalmers.se/agda/pmwiki.php`

[8] R. A. Weaver and T. P. Kelly: "The Goal Structuring Notation – A Safety Argument Notation", *Proc. of Dependable Systems and Networks 2004(DSN2004), Workshop on Assurance Cases*, July 2004.

[9] G. Despotou: *Managing the Evolution of Dependability Cases for Systems of Systems*, PhD Thesis, Department of Computer Science, University of York, 2007.

[10] N. Mansourov and D. Campara: *System Assurance: Beyond Detecting Vulnerabilities*, The MK/OMG Press Series, Elsevier Science & Technology, 2010.

[11] OMG: *Argumentation Metamodel (ARM)*, Beta 1, OMG Document Number: ptc/2010-08-36, 2010.

[12] Nancy Leveson: "A new accident model for engineering safer systems", *Safety Science*, Vol. 42, No. 4. pp. 237–270, April 2004.