アクター概念に基づく
アシュアランスケース設計法に関する研究

猿渡　卓也

目次

1. 序論

1.1. 本研究の目的

社会における様々な分野で IT 技術が利用されている．金融システムや交通システム等の社会的に重要なインフラや，医療分野等の人の生命に直接関わる分野においても，IT 技術は不可欠な存在となってきている．このような重要な分野におけるシステムの障害は大きな損失をもたらし，社会的な問題となる．また，近年のシステムの多くには，システムが独立に存在しているのではなく，システム同士が相互に連携しているという特徴がある．複数のシステムが相互に連携しているという状況が，この問題の解決をさらに難しくしている．このような状況の中で，システムのディペンダビリティや安全性等を保証する技術として，アシュアランスケースが注目されている[1]．アシュアランスケースとは，システムのディペンダビリティや安全性に関する議論を構造的に記述した文書である．アシュアランスケースは，要件定義から運用に至るシステム開発全般で使用することができる．アシュアランスケースの使用により，システムのディペンダビリティや安全性に関する議論を整理し，ステークホルダ間で共有することが可能となる．ステークホルダによるシステムのディペンダビリティや安全性に関する議論の共有は，システムやシステムを利用したサービスの保証につながる．しかしながら，従来の研究において提案されているアシュアランスケースの記法には，Table 1 に示す 3 つの欠陥が存在する．

Table 1 従来の研究におけるアシュアランスケースの記法の欠陥

ID	欠陥内容
欠陥①	複数のシステムや複数のコンポーネントから構成されるシステムの保証が考慮されていない.
欠陥②	アシュアランスケースで実施される議論を保証する責任主体の概念が考慮されていない.
欠陥③	アシュアランスケースと既存の設計手法との関係が不明確である.

そこで，本研究では，これらの欠陥に対応するため，以下の 3 つの課題に取り組む.

A)　複数のシステムやコンポーネントから構成されるシステムの保証

近年，単独で存在するシステムは少なくなってきており，相互に連携するシステムが増加してきている．このようなシステムは，System of systems（SoS）[2]の概念でも知られている．また，1 つのシステムに着目すると，システム自体が複数のコンポーネントから構成されている場合が多くなってきている．すなわち，現状において構築されているシステムの多くは，複数のシステムやコンポーネントが相互に依存する構成となっている．このような状況であるにも関わらず，従来のアシュアランスケースには，複数のシステムやコン

ポーネントから構成されるシステムの保証が考慮されていないという欠陥が存在した．従来のアシュアランスケースでは，対象のディペンダビリティや安全性に関する議論を，基本的に全て 1 つのアシュアランスケースに記述する．これは，システムが複数の要素から構成されている場合でも同様である．従来のアシュアランスケースでは，アシュアランスケースの中で，システムの複数の構成要素を表現する手段が無い．これでは複数のシステムやコンポーネントが相互に依存するシステムの保証に関する議論を記述することは難しい．すなわち，従来のアシュアランスケースには，複数のシステムやコンポーネントが相互に依存するシステムの議論を，効率的に実施できるようにするという課題がある．本研究では，この課題に対する取り組みとして Actor の概念を導入したアシュアランスケースである d* framework を提案する．d* framework において，Actor は対象システムの構成要素を示す要素として定義されている．すなわち，対象とするサービスを構成するシステム，システムに含まれるサブシステムやコンポーネント，ユーザ，サービス提供者，システム開発者等が，Actor として定義される．

B)　アシュアランスケースに対する責任属性の導入

アシュアランスケースには，システムを保証するための議論が記述される．アシュアランスケースに記述される議論により，対象のシステムが保証されるためには，アシュアランスケースに記述される議論そのものが保証される必要がある．そのためには，アシュアランスケースに記述される議論そのものに責任を持つ責任主体が，明確になっている必要がある．しかし，従来のアシュアランスケースでは，必ずしもこの責任主体を明示的に記述することができなかった．つまり，従来のアシュアランスケースには，アシュアランスケースで実施される議論を保証する責任主体の概念が考慮されていないという欠陥が存在する．この欠陥に対する課題として，アシュアランスケースに対する責任属性の導入がある．本研究では，この課題に対する取り組みとして，d* framework に対して，責任主体を明示的に記述することができる Agent の要素を導入することを提案する．

C)　既存の設計手法とアシュアランスケースの関係の明確化

アシュアランスケースは，ソフトウェア工学の分野における手法の 1 つである．ソフトウェア工学の分野では，システム開発の上流工程から下流工程に至る様々な工程を対象として，システム開発を行うための研究が実施されている[3]．それらの研究において，システム開発で使用できる様々なモデルが提案されている．これらのモデルとアシュアランスケースの間には関係があると考えられる．また，この関係を明らかにすることで，アシュアランスケースをより効果的に使用することができるようになると考えられる．しかし，アシュアランスケースとそれら既存の設計手法との関連に関する研究は，ほとんど存在しない．すなわち，既存のアシュアランスケースに関する研究には，アシュアランスケースと既存の設計手法との関係が不明確であるという欠陥が存在する．本研究では，この問題に

対する取り組みとして，既存の設計手法のモデルであるコラボレーション図[4]と d* framework（アシュアランスケース）の関係を明確にし，コラボレーション図を使用した d* framework の作成手法を提案する．

1.2. 本研究の内容

本研究は，d* framework に関する研究を中心としている．d* framework とは，Actor の概念が導入されたアシュアランスケースの作成手法である．また，d* framework で作成されたアシュアランスケース自体も d* framework と呼ばれる．本研究では，d* framework の提案及び拡張により，上記目的に挙げた 3 つの課題に対する取り組みを実施する．d* framework の詳細については，3 章 d* framework の提案で述べる．

1.3. 本研究のアプローチ

システムの保証に対する関心が高まるのに伴い，システムを保証するための技術であるアシュアランスケースに関する研究がすすめられてきた．しかし，前節で示した通り，従来のアシュアランスケースの記法には，Table 1 に示す欠陥が存在する．そのため，本研究では，これら欠陥に対応するための課題を上記のように設定し，研究を実施する．

欠陥①に対しては，複数のシステムやコンポーネントから構成されるシステムの保証方法を提案することで対処する．具体的には，システムの構成要素であるサブシステムやコンポーネント等を示すことができる Actor の概念を，アシュアランスケースに導入する（“Actor の導入”）．

欠陥②に対しては，アシュアランスケースに記述される議論に対して，責任主体を明示的に記述することができる Agent の要素を導入することで対処する（“責任属性の導入”）．

欠陥③に対しては，既存の設計手法とアシュアランスケースの関係の明確化の一環として，既存の設計手法であるコラボレーション図を使用したアシュアランスケースの設計法について検討し提案する（“設計法の検討”）．

本研究において実施する研究を，アシュアランスケースへの Actor および責任属性の導入に対する“記法の提案”，“記法の評価”，“設計法の検討”として整理したものを，本論文における各章の位置付けとして Figure 1 に示す．Figure 1 では，本研究において実施する研究の実施範囲についても示している．

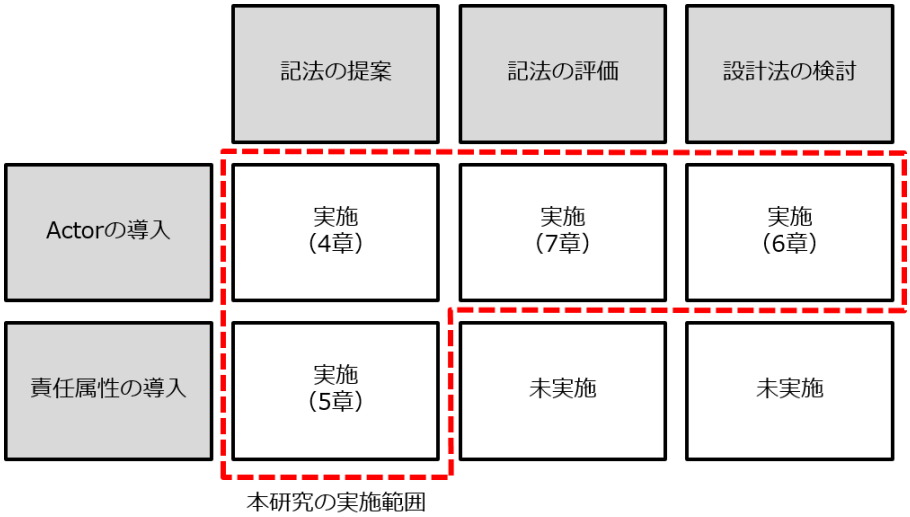

Figure 1 本論文における各章の位置付け

責任属性の導入について，本研究では “記法の評価”および“設計法の検討”が，実施されていない．しかし，これらについては，実施済みの部分と相互に依存が少なく，分けて考えても問題がない．これらについては，今後の課題となる．

2. 関連研究

2.1. アシュアランスケースの概要

アシュアランスケースとは，対象とするシステムのディペンダビリティや安全性を保証するために作成される文書である[10]．ここで，ディペンダビリティとは，システムの保証に関する概念である．一般的に，ディペンダビリティは，システムの保証に関する複数の属性の複合概念として定義されている．[13][14]では，ディペンダビリティは “可用性（Availability）”，“信頼性（Reliability）”，“安全性（Safety）”，“統合性（Integrity）”，“保守性（Maintainability）”からなる複合概念として定義されている．また，[13][14]では，ディペンダビリティに関連する概念であるセキュリティは，“可用性（Availability）”，“機密性（Confidentiality）”，“統合性（Integrity）”の複合概念として定義されている．つまり，[13][14]では，ディペンダビリティと同様にセキュリティも複数の概念の複合概念として定義され，ディペンダビリティとセキュリティは，対象とする属性に重なりのある概念として定義されている．一方，[15]では，ディペンダビリティは“安全性（Safety）”，“保守性（Maintainability）”，“セキュリティ（Security）”，“信頼性（Reliability）”，“可用性（Availability）”，“統合性（Integrity）”の複合概念として定義されている．いずれにしても，ディペンダビリティとは，システムが安全で信頼できるものであるために満たすべき属性を定義した複合概念であるといえる．本研究におけるディペンダビリティの定義は，[13][14]に従うものとする．

アシュアランスケースには，保証の対象となるシステムのディペンダビリティや安全性の保証に関する議論が記述される．歴史的には，最初に，システムの安全性を保証するための議論を記述するセーフティケースが提案された．セーフティケースがイギリスで普及した背景には，1988 年におきた死者 167 名を出した北海油田事故などの深刻な事故がある．そのような深刻な事故を背景として，なぜシステムが安全になるのか，エビデンスをもとに議論する必要性が広く認識された[5]．セーフティケースの一般的な定義は，次のようなものである．

“A documented body of evidence that provides a convincing and valid argument that a system is adequately safe for a given application in a given environment” [6][7][8]

現状においても，アシュアランスケースが必要となる背景として，システムの障害が社会的な問題になってきているという点が挙げられる[9]．特に，社会的に重要なシステム（クリティカルシステム）の障害は，重大な損失をもたらす可能性がある．社会的に重要なシステムとは，エネルギー等の様々なライフラインに関するシステム，公共の交通機関に関するシステム，金融系のシステム，医療系のシステム等の，システムの停止が与える影響が極めて大きなシステムを指す．これらのシステムの停止は，即座に重大な損失をもたら

す．場合によっては人の生命にかかわることもある．従って，システムの障害が起こらないように，システムを保証する必要がある．また，障害が起きた場合，障害が起きた原因が明確にされる必要がある．アシュアランスケースは，システムのセーフティ（安全性），セキュリティ，ディペンダビリティ等を保証する目的で使用される技術である[10]．アシュアランスケースを使ってシステムを保証することで，Table 2 に示す効果があると考えられる[12].

Table 2 アシュアランスケースによる効果

ID	効果
1	主張するサービス水準を保証するための証跡の提供
2	システム異常の検出と修正の早期化
3	開発・運用プロセスと生産物のリスクに対する客観的な管理の推進
4	システムのディペンダビリティに対する影響評価の早期化
5	システムの要求適合性に対する，客観的な証跡に基づく確認・保証プロセスの提供
6	システム開発・運用プロセスを統合的に確認できるモデルを提供することによるプロセス改善

また，アシュアランスケースを使用することにより，“システムの保証に関する議論に対するステークホルダ間の合意形成”，“システムの保証についての説明責任の明確化”等の効果も期待できる.

アシュアランスケースは，主にイギリスなど欧米において導入されてきた．欧米では，高い保証が求められるシステムを開発・運用する際に，アシュアランスケースを認証機関に提出することが義務付けられるほどに普及している．日本においても，サービスやシステムの保証に対するアシュアランスケースの使用が，徐々に広まりつつある．[103]では，アシュアランスケースの記法の 1 つである D-Case を使った実証研究が実施されている. ここには，日本における D-Case を使ったシステム保証の事例も多数掲載されている.

アシュアランスケースの保証対象は，システムのセーフティ（安全性），ディペンダビリティ，セキュリティ等である．保証する対象により，アシュアランスケースはアシュアランスケースという名称とは異なる名称で呼ばれることがある．例えば，安全性を保証するためのアシュアランスケースは，セーフティケースと呼ばれることがある．また，ディペンダビリティを保証するためのアシュアランスケースは，ディペンダビリティケースと呼ばれることがある．アシュアランスケースは，これらの名称に付けられた，一般的な名称であると考えることができる[10]. 本稿では, これらをまとめてアシュアランスケースと呼ぶ.

アシュアランスケースの記述方法としては，ゴールグラフ（ゴールモデル）が使用されることが多い[11]．ゴールグラフとは，対象の目的・目標をゴールとして定義し，上位のゴールから下位のゴールに向けて段階的にゴールを分解して記述することで，対象の目的・目

標を構造的に記述するための記法である．ゴール指向要求工学の分野では，システム開発の上流工程において，システムの要求分析を実施するためにゴールグラフが使用される[82]．アシュアランスケースでは，サービスやシステムのディペンダビリティや安全性を保証することを目的としてゴールが記述される．一般的なゴールグラフにはないアシュアランスケースの特徴として，ゴールを分解する際の議論を，戦略としてアシュアランスケースに記述する点が挙げられる．すなわち，アシュアランスケースには，サービスやシステムの保証に関して実施される議論が，ゴールと共に記述される．また，アシュアランスケースには，アシュアランスケース内で示されたゴールの保証を具体的に示す証拠が記述されるという特徴もある．

アシュアランスケースの適用事例としては，次のようなものが存在する．[83]では，医療機器に関するアシュアランスケースの適用事例が示されている．[84]では，無人飛行機の制御に関するアシュアランスケースの適用事例が示されている．[85]では，教育のための知識管理システムに関するアシュアランスケースの適用事例が示されている．[86]では，ヘリコプターの飛行管理システムに関するアシュアランスケースの適用事例が示されている．[87]では，UK におけるアシュアランスケースの適用について述べられている．また，[88]では，アシュアランスケースの有効性についてシステムダイナミクスを使って評価する試みが実施されている．このように，様々な分野においてアシュアランスケースの適用が進んでおり，それらに関する報告が行われている．また，アシュアランスケースを適用するための研修コースも提供されている[89][90]．さらに，アシュアランスケースに関する研究について，[91]ではアシュアランスケースをベースとしたシステムディペンダビリティの保証のための研究フレームワークが提案されている．

2.2. Goal Structuring Notation (GSN)

Goal Structuring Notation（GSN）[17][32][33]は，Tim Kelly によって提案されたアシュアランスケースの作成手法である．現在では，GSN COMMUNITY STANDARD VERSION 1[16]として広く知られている．もともと GSN は，システムの安全性を保証するためのアシュアランスケースであるセーフティケースの記法として提案された．その後，システムが保証するべきとされる対象属性の範囲が広がるのに伴い，GSN が対象とする属性も拡張された．現在では，ディペンダビリティも保証対象としている[15]．GSN では，システムのディペンダビリティや安全性に関する議論が，議論の要素とそれら要素間の関係で示される構造的なグラフとして記述される．議論の要素として，“Goal”，“Strategy”，“Solution”，“Context”，“Assumption”，“Justification” の 6 つが，定義されている[16]．GSN では，システムのディペンダビリティや安全性に関する Goal を定義し，定義された Goal を下位の Goal に分解することにより議論を進める．下位の Goal に分解する際，Goal を分解するための議論分解の戦略を Strategy として定義する．また，最終的に議論は Solution により保証される必要がある．すなわち，GSN の最下層の Goal は Solution により保証される．

2.2.1. GSN の構成要素

GSN では，システムの保証に関する議論を，6 つの種類の要素を使用して記述する．下記に GSN において使用される 6 つの種類の要素について述べる．

- Goal

  Goal は, システムを保証するために達成する必要がある目的･目標を示す要素である．システムのディペンダビリティや安全性を保証するための目的や目標が，Goal として記述される．

- Strategy

  Strategy は，Goal を下位の Goal に分解する際，実施する議論（議論分解の戦略）を示す要素である．Strategy により，Goal がどのような議論を経て下位の Goal に分解されたのかを，明示的に示すことができる．

- Solution

  Solution は，上位の Goal が達成されることを保証する証拠を示す要素である．通常 GSN の最下層の Goal を保証するために Solution が使用される. Goal を保証するための証拠は，具体的に示される必要がある．すなわち，Solution は現実世界に存在する証拠と関連付けられるかたちで記述される必要がある．尚，Solution は Evidence とも呼ばれる．

- Context

  Context は，Goal あるいは Strategy の背景情報を示す要素である．Goal や Strategy に記述される事項に対する詳細な説明等が，Context として記述される．

- Justification

  Justification は，Goal あるいは Strategy に関して，根拠のある背景情報を示す要素である．

- Assumption

  Assumption は，Goal あるいは Strategy に関して，確証はないが背景として存在する情報を示す要素である．

  尚，Justification および Assumption は，Context のサブクラスであると考えることができる．

GSN のアシュアランスケースは, グラフィカルな記法を使って記述される. GSN において, 要素を記述するための記法を Figure 2 に示す. GSN において, これらの記法が使用される際は, それぞれの図形の中にそれら要素の具体的な内容が記述される. 例えば, GSN の Goal には, Goal を示す記法である四角形の中に, Goal の具体的な内容が記述される.

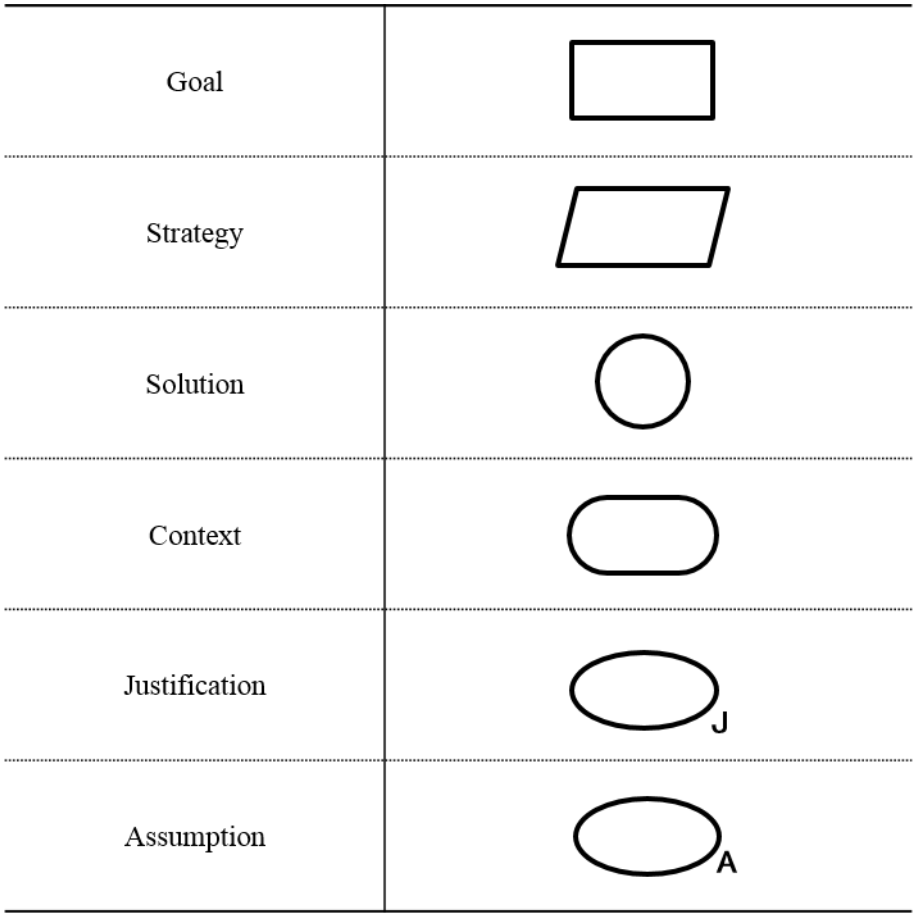

Figure 2 GSN において要素を記述するための記法

2.2.2. 議論の未完了箇所

GSN を使用したシステムのディペンダビリティの保証に関する議論は, 上位の Goal を, 下位の Goal に分解することで実施される. さらに, 全ての最下層の Goal が, Solution によって保証されることで, GSN における議論が完了する. GSN では, システムの保証に関する議論を実施する際, Goal や Strategy が下位の Goal や Strategy に分解されていない, もしくは Solution によって保証されていないことを明示的にすることで, 議論が未完了であることを示すことができる. Figure 3 に議論が未完了であることを表す記法を示す. この記法では, Goal あるいは Strategy に菱型の図形を付与することで, それらが下位の Goal や Strategy に分解されていない, あるいは Solution により保証されていないことが示され

る．Figure 3 では，議論が完了していない Goal の例を示している．この記号を含む GSN は，システムの保証に関する議論が完了していない．従って，引き続き対象システムの保証に関する議論を実施する必要がある．

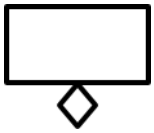

Figure 3 GSN において議論が完了していない Goal の記法

2.2.3. GSN における要素間の関係

GSN には，要素間の関係として，“supported by”，“in context of”の 2 つの種類の関係が定義されている．これら 2 つの種類の関係について以下に述べる．

- supported by
  supported by 関係は，上位の Goal，Strategy が下位の Goal，Strategy，Solution に保証される関係を示す．

- in context of
  in context of 関係は，Goal 及び Strategy と Context，Justification，Assumption の関係を示す．Goal 及び Strategy の背景情報が Context 等により示される場合，この関係が使用される．

GSN におけるアシュアランスケースは，グラフィカルな記法を使って記述される．これらの関係を記述するための記法を Figure 4 に示す．図の中で，点線で囲んだ要素は複数の種類の要素が対応することを示している．また，in context of 関係については，Context と Goal あるいは Strategy の関係として示している．

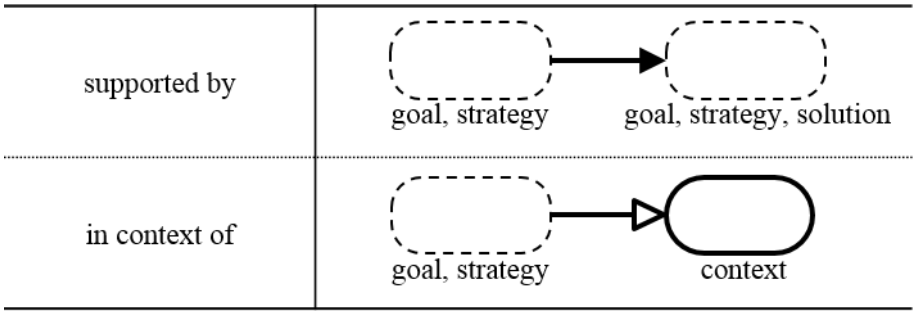

Figure 4 GSN において関係を記述するための記法

2.3. Claims, Arguments and Evidence (CAE)

Claims, Arguments and Evidence（CAE）は，UK に拠点を置く Adelard 社により提案されたアシュアランスケースの作成手法である[19]．CAE では，システムのディペンダビリティや安全性に関する議論が，議論の要素とそれら要素間の関係で示される構造的なグラフとして記述される．議論の要素として，“Claim”，“Argument”，“Evidence”の 3 つの種類の要素が使用される．CAE では，ディペンダビリティや安全性の保証に関して記述された Claim（要求）に対して，その Claim をどのように保証するかを議論する．その議論の内容については，Argument で示される．また，議論の結果生じる Claim（新規に生じた達成すべき要求）が，下位の Claim として記述される．最下層に記述された Claim は Evidence によって保証される．

2.3.1. CAE の構成要素

CAE では，システムの保証に関する議論が，3 つの種類の要素を使って記述される．下記に CAE で定義されている 3 つの種類の要素について述べる．

- Claim
  Claim は，CAE が対象とするサービスやシステムが達成すべきディペンダビリティや安全性の保証に関する目的・目標を，示す要素である．Claim は，True あるいは False で評価できるものでなければならない．

- Argument
  Argument は，上位の Claim を保証するために実施した議論の内容について，示す要素である．

- Evidence
  Evidence は，最下層の Claim を保証するための証拠を示す要素である．Evidence は，具体的に示す必要がある．従って，Evidence は現実世界に存在する証拠と紐付くかたちで記述される．

CAE において，アシュアランスケースはグラフィカルな記法を使って記述される．CAE の要素を記述するための記法を Figure 5 に示す．CAE の中でこれらの記法が使用される際には，それぞれの図形の中にそれら要素の具体的な内容が記述される．例えば，CAE の中の Claim には，Claim を示す記法である楕円形の中に Claim の具体的な内容が記述される．

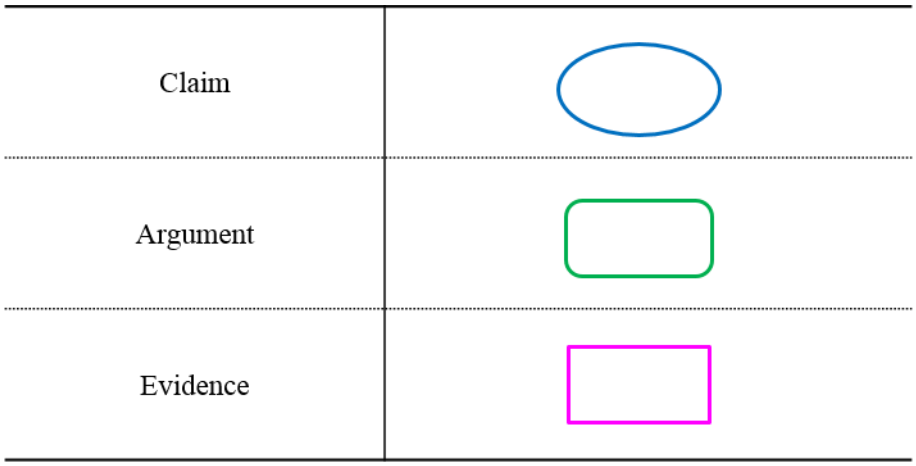

Figure 5 CAE において要素を記述するための記法

2.3.2. CAE における要素間の関係

CAE には，要素間の関係として，“supports”，“is a subclaim of”，“is evidence for”の 3 つの種類の関係が定義されている．これら 3 つの種類の関係について以下に述べる．

- supports
  supports 関係は，Argument が Claim を保証する関係である．この関係の下位にある Argument には，上位の Claim を達成するために実施する議論の内容が記述される．すなわち，上位の Claim は，下位の Argument に記述されている議論により保証される．

- is a subclaim of
  is a subclaim of 関係は，下位の Claim（subclaim）が上位の Argument に記述されている議論の結果生じた Claim であることを示す関係である．

- is evidence for
  is evidence for 関係は，下位の Evidence が，上位の Claim を保証する関係である．下位の Evidence として，上位の Claim を保証するための具体的な証拠が記述される．

CAE における各要素間の関係は，関係の名称がラベルとして付与された矢印によって示される．

2.4. Modular GSN

アシュアランスケースが対象とするシステムが複雑になると，システムのディペンダビリティの保証に必要な議論の量が増加する．そのため，アシュアランスケースに含まれる要

素の数が増加する．その結果，アシュアランスケース自体が大きくなり，1 つのアシュアランスケースとして作成・管理することが難しくなる．この問題への対策として，アシュアランスケースを分割して作成・管理することが考えられる．アシュアランスケースの分割に関する研究として，アシュアランスケースの Module 化に関する研究が進められている．アシュアランスケースを Module 化することにより，大規模なアシュアランスケースを分割して扱うことができるという利点がある．また，その他の利点として，アシュアランスケースの参照性の向上，アシュアランスケースの再利用性の向上が期待できる．
[20][21][22][115]では，GSN を拡張して Module を使用することができるようにした，Modular GSN に関する研究が実施されている．

2.4.1. Moduler GSN の構成要素

Modular GSN は，GSN を拡張して定義されたアシュアランスケースの記法である．Modular GSN では，GSN において定義されている要素に加え，Module を記述するために次の 3 種類の新規要素が定義されている．

- Module
  Module は，GSN の Module を示す要素である．この要素を使用して，複数の要素から構成される GSN の任意の部分を Module として示すことができる．

- Away goal，Away context，Away solution
  Away goal，Away context，Away solution は，他の Module 内に記述されている要素に対するリファレンスを示す要素である．この要素を使用して，異なる Module に存在する要素間の参照関係を示すことができる．例えば Away goal は，他の Module に記述されている Goal を参照するための要素である．

- Contract Module
  Contract Module は，Module 間の関係を示す要素である．Contract Module は，Module の 1 種である．

Modular GSN において新規に定義された要素も，グラフィカルな記法を使って記述される．Figure 6 に，Modular GSN において新規に定義された要素の記法を示す．ここでは，他の要素を参照する要素の例として，他の Goal を参照する Away goal を示している．Away goal は，Goal を示す長方形を 2 つの部分に区切った図で示される．2 つに区切られた図の中で下部には，参照先の Goal が存在する Module 名が記載される．

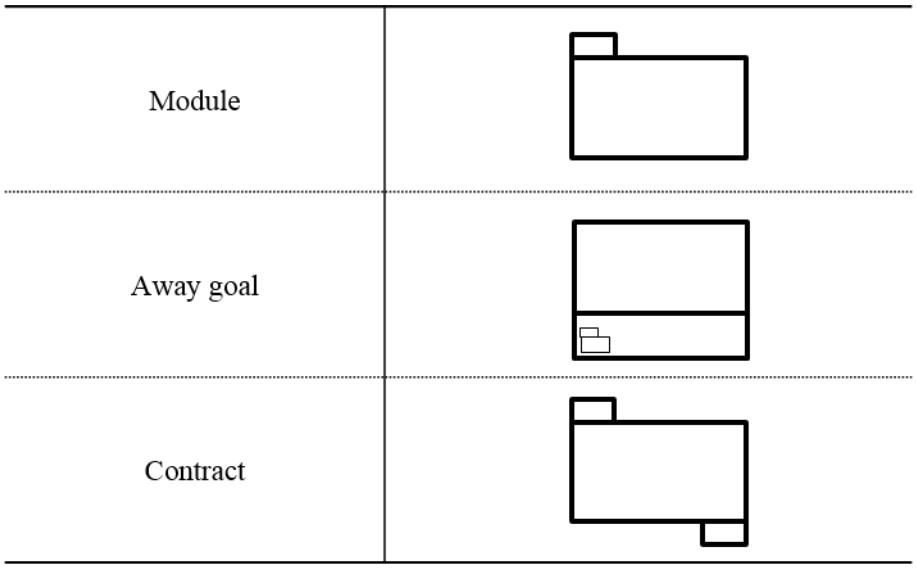

Figure 6 Modular GSN において追加定義された要素の記法

2.5. D-Case

D-Case[23][24]は，GSN を拡張したアシュアランスケースを作成するための手法である．D-Case では，GSN で定義されている既存の要素に加え，新規の要素としてモニタリングノードが定義されている．モニタリングノードとは，サービスやシステムに関するデータをリアルタイムに取得し，D-Case における議論を保証するための証拠とすることを示す要素である．従って，モニタリングノードは，Evidence あるいは Solution のサブクラスであると考えることができる．モニタリングノードを使用することで，D-Case における議論が保証されていることをリアルタイムに示すことができる．

2.5.1. D-Case Editor

D-Case Editor[25][26][27]は，DEOS プロジェクト[28]において，D-Case[23]を作成するために開発されたツールである． DEOS プロジェクトとは，科学技術振興機構（JST）の戦略的基礎研究事業(CREST)として文部科学省が設定した研究領域の一つである. D-Case Editor は，統合開発環境である Eclipse[29]のプラグインとして開発されている．Figure 7 に D-Case Editor のスクリーンショットを示す．D-Case Editor は，D-Case の作成用に開発されたツールであるが，GSN に代表される典型的なアシュアランスケースの作成に使用することが可能である．D-Case Editor を使用することで，アシュアランスケースを直感的な操作で記述することができる．D-Case Editor には，アシュアランスケースの Module を記述するための機能も実装されている[30].

本研究では後の章で，アシュアランスケースの 1 種である d* framework の記述に D-Case Editor を使用する．d* framework では，対象システムの構成要素を示す要素として Actor が定義されている．本研究では，Actor の記述に，D-Case Editor の Module を記述するた

めの機能を使用する．

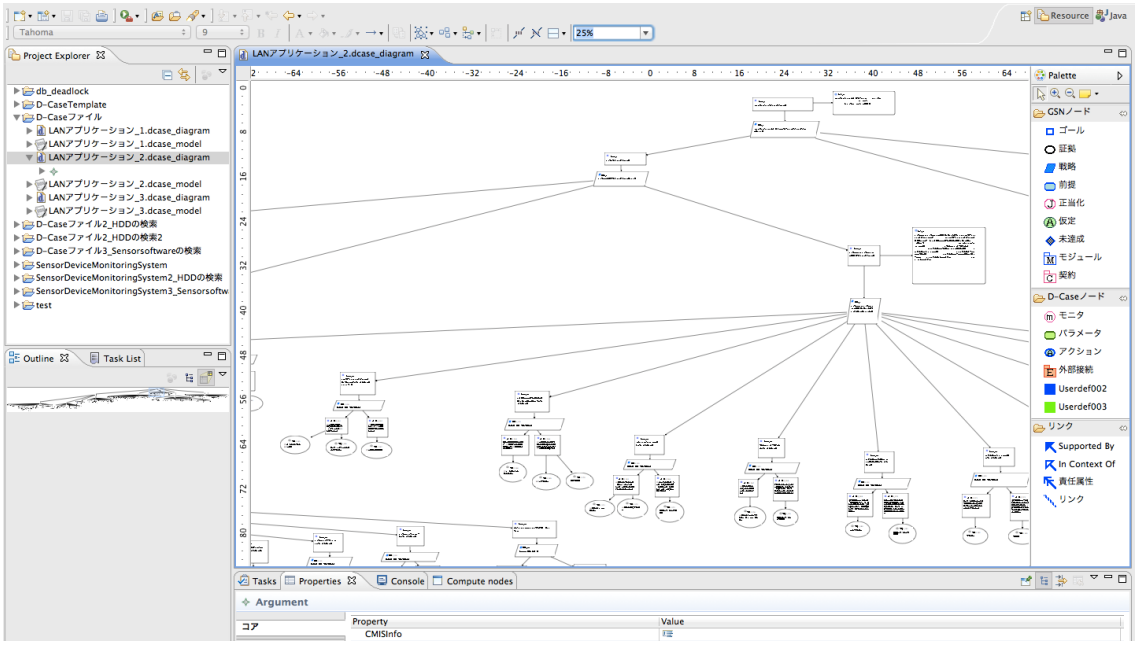

Figure 7 D-Case Editor のスクリーンショット

2.6. アシュアランスケースのパターン

アシュアランスケースを用いてシステムのディペンダビリティの保証に関する議論について記述する際，アシュアランスケース内の複数個所に，似たような議論構造が現れることがある．このような構造を，アシュアランスケースのパターンとして定義する研究が実施されている[31]．[17][18]では，アシュアランスケースのパターンを記述することを目的として，GSN にいくつかの記法が追加されている．追加された記法を使用して，典型的な議論のパターンを定義することができる．[17]には，上記記法を使った複数のパターンの例が示されている．[34][35]では，実際に幾つかのパターンが定義されている．これらのパターンは，アシュアランスケースを作成する際に使用することができる．[113]では，アシュアランスケースの 1 種である d* framework を対象として，d* framework の責任関係パターンに関する研究が実施されている．

アシュアランスケースを使った議論では，対象システムのディペンダビリティを保証するための要求を示す Goal を，下位の Goal に分解することで議論を進める．この際に使用できる議論分解のパターンも提案されている．これらのパターンを使用することで，アシュアランスケースを使った議論を実施する際, パターンを使用した Goal の分解が可能となる．[36]では，議論分解のパターンとして，Table 3 に示す 7 つのパターンが提案されている．

Table 3 [36]で提案されている議論分解のパターン

パターン名	説明
architecture	コンポーネントに対する議論を，幾つかのサブコンポーネントに対する議論に分解するパターン.
functional	コンポーネントに対する議論を，幾つかのサブ機能に対する議論に分解するパターン.
set of attributes	議論の対象が持つ属性を使い，それら属性毎の議論に分解するパターン.
infinite set	典型的な事例を使って帰納的に分類を作り，その分類に従って議論を分解するパターン.
complete	考えられるリスクや要求等の全ての集合を定義し，その集合に含まれる要素に従って議論を分解するパターン.
monotonic	古いシステムから新しいシステムへの改良点に着目し，古い部分と新しい部分に議論を分解するパターン.
concretion	形式的ではないが曖昧性の少ない記述を使って分解するパターン.

[37][38]では，これらのパターンを使用したアシュアランスケースの作成が実施されている．また，これら議論分解のパターンが，アシュアランスケースの作成に有用であることが示されている．

定義されたパターンを適切に使用するためには，パターンの使用者が，予め定義されたパターンに関する知識を有している必要がある．パターンの数が増えてくると，全てのパターンを把握することが難しくなる．そのため，用途に適合するパターンを検索し，使用することが困難になってくることが予想される．今後，パターンを効果的に使用できるような分類方法，検索方法の研究が必要となる．また，パターンを定義する際のパターンの粒度を適切にするための研究も必要となる．

2.7. アシュアランスケースの作成手法

アシュアランスケースを適切に作成し管理するためには，アシュアランスケースの記法が定義されているだけでは不十分である．アシュアランスケースを作成し管理するためのプロセスが，定義されている必要がある．このようなアシュアランスケースを，作成・管理するためのプロセスの研究が，実施されている．[17][39]では，GSN の作成プロセスとして，Table 4 で示される 6 つのステップが定義されている．

Table 4 GSN の作成手順

Step	内容
Step1	Goal の識別
Step2	Goal の定義
Step3	Strategy の識別
Step4	Strategy の定義
Step5	Strategy による検討（新規 Goal の識別）
Step6	Solution の識別

また, これらのステップの実施順序も定義されている. 定義されている実施順序を Figure 8 に示す．この作成プロセスでは，上位の Goal から Strategy を識別し，その Strategy を検討することで，下位の Goal を識別するプロセスが示されている．検討の対象としている Goal が，Strategy による検討の継続より，直接 Solution により保証される方が良いと判断された場合は，Step 6 に進み Soution が識別される．一般的なアシュアランスケースの作成プロセスは、Figure 8 に示されるような手順で実施されるものであると考えることができる．一般的に，アシュアランスケースは，上位の Goal を記述し、記述された上位の Goal を分解して下位の Goal を記述していくことで作成される.

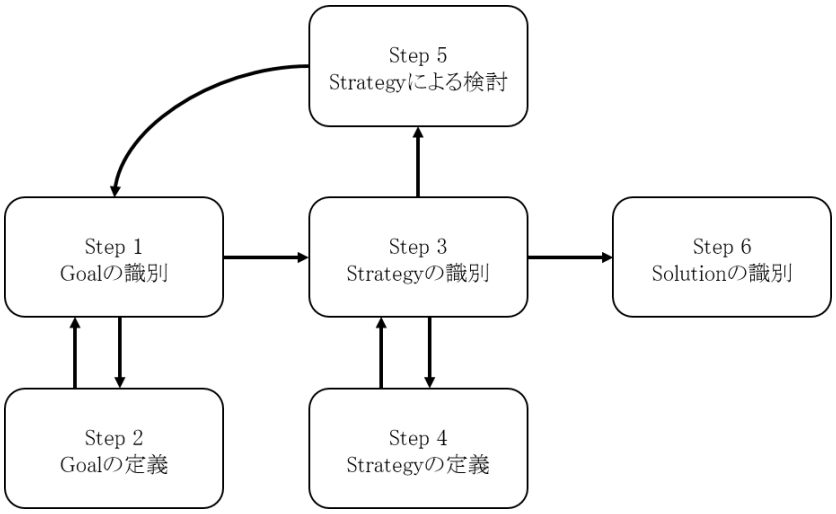

Figure 8 GSN の作成手順

アシュアランスケースの作成・管理にまたがるプロセスに関する研究も実施されている. [28][40]では，アシュアランスケースの一種である D-Case を使ったシステムのライフサイクルが, DEOS プロセスとして定義されている. DEOS プロセスを簡略化したものを Figure 9 に示す．DEOS プロセスには，“変化対応サイクル”と“障害対応サイクル”の 2 つのサ

イクルが定義されている．“変化対応サイクル”とは，システムの目的変化・環境変化に対して実施されるシステム変更のサイクルである．“障害対応サイクル”とは，システムの（障害）予兆検知・障害発生に対して実施される障害対応のサイクルである．障害対応の結果，システム変更が必要となる場合がある．そのような場合は，“障害対応サイクル”から“変化対応サイクル”に処理が引き継がれる．DEOS プロセスの中で，D-Case はシステムの保証に関する合意記述データベースとして作成される．さらに，DEOS プロセスの中で定義されている各タスクから参照・変更される．すなわち，DEOS プロセスでは，システムの保証に関する議論が D-Case によって管理される．The Open Group では，DEOS プロセスを組み込んだ O-DA フレームワークが定義されている[41].

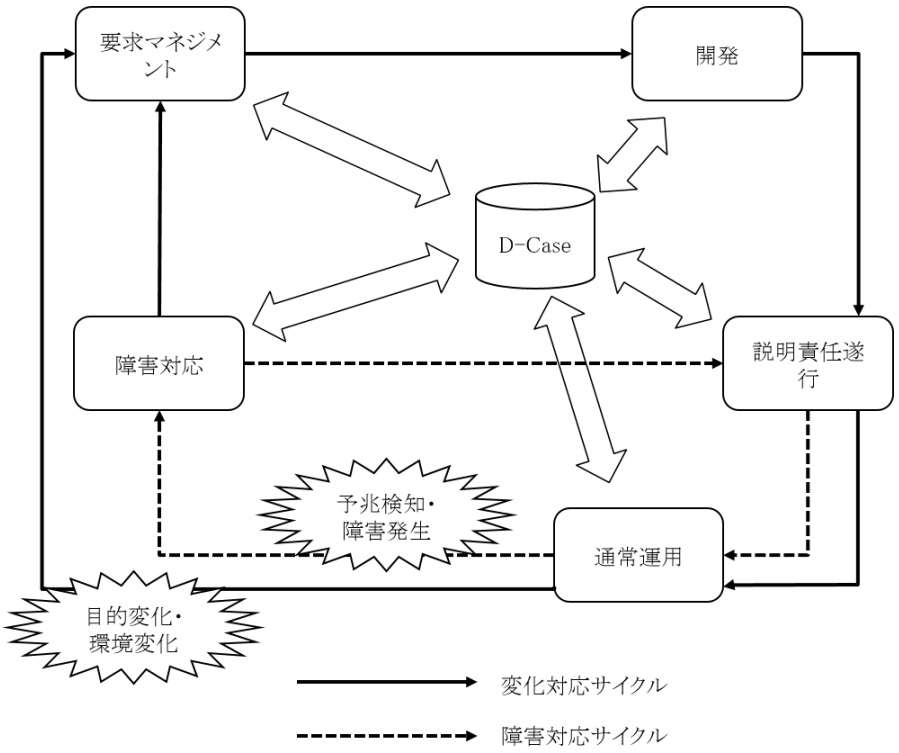

Figure 9 DEOS プロセス

その他のアシュアランスケースの作成に関する研究として，次のようなものがある．[42]では, D-Case の構築法が提案され, 提案された構築法の評価が実施されている. [43][44][45]では, アシュアランスケースの中で使用する用語に関する研究が実施されている. [46]では，アシュアランスケースの作成とシステム開発を同時に進める手法である Assurance Based Development（ABD）が提案されている．[47]では，論理式からアシュアランスケースを作成する手法が提案されている．[48]では，アシュアランスケースの記述内容について，整合性検査を行う研究が実施されている．[49]では，アシュアランスケースに変数を導入し，その変数の型チェックを行うことにより，アシュアランスケースの確認を行う手法が提案

されている．[50]では，アシュアランスケースの整合性検査を実施するためのツールが開発されている．[51]では，アシュアランスケースを使用した，情報システムの運用手順の確認手法が提案されている．[52]では，アシュアランスケースに記述される Context の推定手法が提案されている．[53][54]では，エンタープライズ・アーキテクチャの開発プロセスに対するアシュアランスケースの作成手法が提案されている．[55]では，大きなサイズのアシュアランスケースを作成するための記述方法が提案されている．[56]では，シナリオを用いて，ソフトウェアの設計時に，ディペンダビリティ要求を満たすような代替案を選択する手法が提案されている．[57]では，コンテクストモデル，フィーチャーモデル，構造モデル，ユースケースモデル，振舞いモデル，故障管理振舞いモデルに基づいて，アシュアランスケースを作成する手法が提案されている．[58]では，医療分野におけるアシュアランスケースのレビュー手法が提案されている．[59]では，アシュアランスケースを使用したサービスの提供判断方法が提案されている．[60]では，表形式にしたアシュアランスケースのレビュー手法が提案されている．

ソフトウェア工学における他の分析手法と組み合わせたアシュアランスケースの作成手法についても研究されている．[61]では，ユースケース分析に基づくアシュアランスケースの作成手法が提案されている．[62]では，形式手法による検証範囲を，アーキテクチャに基づいて明確化するアシュアランスケースが提案されている．[63][64]では，ディペンダビリティに関する要求を抽出するのに，偏差分析を利用する手法が提案されている．

このように，アシュアランスケースの作成手法に関する研究が，数多く実施されるようになってきている．

2.8. アシュアランスケースにおける議論の保証

保証に関する具体的な証拠をベースとして，システムを保証することができる [65]．アシュアランスケースにおける議論は，この考え方に基づいて保証される．アシュアランスケースを使った議論では，システムのディペンダビリティや安全性がどのように保証されているのかを，明示的に示す必要がある．すなわち，アシュアランスケースを使って実施した議論を保証するための証拠が，アシュアランスケースの中で具体的に示される必要がある．アシュアランスケースの作成手法には，実施される議論が保証されていることを具体的に示すために，保証のための証拠を示す要素が定義されている．すなわち，GSN では Solution，CAE では Evidence が定義されている．アシュアランスケースで使用される，議論を保証するための具体的な証拠の例を Table 5 に示す．

Table 5 アシュアランスケースで使用される具体的な証拠の例

項番	証拠
1	契約書，約款等
2	サービスやシステムの設計書
3	サービスやシステムの試験結果

ソフトウェア工学の手法を使用した分析結果を，アシュアランスケースの証拠として使用することも考えられる．代表的なものでは，システムの故障や障害を分析する手法であるFTA，FMEA，HAZOP[66][67]の結果が，アシュアランスケースの証拠として使用できる．FTA とアシュアランスケースは，似たようなグラフ構造を持っている．しかし，2 つの手法には違いがある．アシュアランスケースが，システムの保証に関する議論を構造化したものであるのに対し，FTA はシステムの故障につながる要因を構造化し，故障発生率を分析したものである．アシュアランスケースにおいて，FTA による分析結果をシステムの故障発生率を分析した証拠として使用することができる．アシュアランスケースの証拠に関する研究は他にも存在する．[68]では，欠陥モデルの結果をアシュアランスケースの証拠として活用することが提案されている．[69]では，ソースコードの解析結果をアシュアランスケースの証拠として活用することが提案されている．また，形式手法[70][71][72][73][74][75][76][77][78][79][80]を使ってシステムを検証した結果も，アシュアランスケースの証拠として活用することができる．

上記の技術は，システムの保証を目的として開発された技術である．システムを保証する手段として上記のような様々な技術を使用する際，アシュアランスケースは，システム全体の保証に関する議論をどのように進め，どのような技術によりその保証を行うのかを構造的に記述する手法であるということができる．

アシュアランスケースにおいて実施される議論を保証する証拠として，静的な証拠だけではなく動的な証拠を活用する技術についても提案されている．D-Case では，Solution のサブクラスとしてモニタリングノードが提案されている．モニタリングノードは，システムの状態を逐次モニタリングし，その状態を示す要素である．モニタリングノードを使用することで，システムの状態を逐次モニタリングし，その状態が正常であることをアシュアランスケースの議論を保証する証拠として活用することができる．すなわち，モニタリングノードを使用することで，議論を動的に保証することが可能となる．

2.9. i* framework

本研究において提案する d* framework は，要求工学における要求分析の手法として提案されている i* framework[101][102]を参考にして開発された．そこで，この節では，i* framework と d* framework の差異について述べる．

i* framework は，システム開発の要求定義工程において，Actor 間の Intention（意図）に

着目してサービスやシステムの要求分析を実施する手法である．i* framework には，Strategic Dependency（SD）モデルと Strategic Rationale（SR）モデルの 2 つのタイプのモデルが存在する．SD モデルは，Actor 間に存在する Intention にフォーカスを当てたモデルである．SR モデルは，より詳細な情報を示すために Actor 内の要求分析結果まで含めたモデルである．

一方，d* framework は，システム開発の全工程において，Actor 間の dependency（依存）に着目して，サービスやシステムのディペンダビリティの保証に関する議論を記述するための手法である．d* framework にも“Actor 間の責任関係に関する議論を記述するモデル”と，“Actor がディペンダブルであることを示すために記述するモデル”の 2 つのタイプのモデルが存在する．

2 つの手法の差異を，目的，使用工程，記述内容，モデルの要素の観点で整理した表を Table 6 に示す．i* framework と d* framework では，使用できるモデルの要素に違いがある．i* framework のみに存在するモデルの要素に Task と Resource がある．Task を使用することにより，2 つの Actor 間における，一方の Actor が他方の Actor のために実施するタスクと，Actor 内に記述された Goal を達成するための手段を記述することができる．また，Resource を使用することにより，2 つの Actor 間でやりとりされる情報等を記述することができる．i* framework では，Goal と Soft Goal が区別されている．このように i* framework で使用される要素は，要求分析を詳細に実施するのに必要となる要素である．一方，d* framework のみに存在するモデルの要素に，Strategy，Context，Evidence がある．3 章で示すように，これらの要素は，システムの保証に関する議論を記述するために使用される要素である．このように i* framework と d* framework は，目的，使用工程，記述内容，モデルの要素のいずれの観点からも異なる手法として位置付けられる．

Table 6 i* framework と d* framework の比較

	i* framework	d* framework
目的	システムの要求分析	システムのディペンダビリティの保証
使用工程	要件定義工程	全工程
記述内容（Actor 間）	Actor 間の Intention（意図）に着目したシステムの要求分析結果	Actor 間の dependency（依存）に着目したシステムのディペンダビリティの保証に関する議論結果
記述内容（Actor 内）	各 Actor に着目したシステムの要求分析結果	各 Actor に着目したシステムのディペンダビリティの保証に関する議論結果
モデルの要素	Actor，Goal，Task，Resource，Soft-Goal	Actor ， Goal ， Strategy ，Context，Evidence

3. d* framework の提案

本章では，本研究において実施する d* framework の提案について述べる[106][109][114]. この研究は，“複数のシステムや複数のコンポーネントから構成されるシステムの保証が考慮されていない”という既存のアシュアランスケースの欠陥に対応する研究である.

3.1. はじめに

近年，社会的に重要なインフラに，IT 技術を使用したサービスやシステムが増加してきている. 現状では, IT 技術を全く使用していないサービスやシステムの方が少ないと言える. このような状況においては，IT 技術を使用したシステムの障害が，多大な損失を引き起こす．そのため，どのようにしてシステムを保証していくのかが，重要な課題となってきている．また，近年のサービスやシステムが持つ特徴として，複数のシステム，サブシステム，コンポーネント，組織等が，複雑に連携することで構成されている点が挙げられる．このような状況が，サービスやシステムの保証を，より難しいものにしていると考えられる．これらの状況を鑑みて，複数の構成要素からなるサービスやシステムの保証に関する議論を体系的に記述するために d* framework が開発された．d* framework とは，Actor の概念が導入されたアシュアランスケースの作成手法である．また，d* framework を用いて作成されたアシュアランスケース自体は，d* framework と呼ばれる.

d* framework には，複雑に連携するシステムの保証に関する議論を記述するため，従来のアシュアランスケースに対する新規の要素として Actor が定義されている．d* framework では，d* framework が対象とするサービスやシステムを構成する要素が Actor として記述される．すなわち，サービスを構成するシステムやシステムを構成するサブシステム，コンポーネント，組織等が Actor として記述される．Actor の導入により，複数のシステム，サブシステム，コンポーネント，組織等が複雑に連携する状況において，それらを Actor として d* framework 内に記述することができる．従って，複数の構成要素から成るサービスやシステムの保証に関する議論を体系的に実施するのに，d* framework は適していると言える．d* framework は，GSN[17][32]を拡張する形で定義されている．d* framework は，GSN に Actor の概念を導入して拡張された手法である．Actor の概念を導入により，d* framework と GSN には明確な差異が存在する[111].

本章では，本研究において提案する d* framework について述べる.

3.2. d* framework の構成要素

d* framework は，GSN を拡張するかたちで定義されている．すなわち，GSN で定義されている要素は，d* framework においても，そのまま使用される．d* framework では，GSN で定義されている要素に加えて，新規の要素として Actor が定義されている．Actor は対象システムの構成要素を示す要素として使用される. 下記に d* framework において使用される 5 つの種類の要素について述べる．尚，GSN で定義されている Justification と

Assumption は，Context のサブクラスであると考えることができるので，d* framework では Context のみを使用することとし，定義していない.

- Actor

  Actor は，d* framework が対象とするサービスやシステムの構成要素を示す要素である．すなわち，対象とするサービスを構成するシステム，システムに含まれるサブシステムやコンポーネント，ユーザ，サービス提供者，システム開発者等が，Actor として定義される．Actor は，d* framework において，新規に導入された要素である．d* framework では，Actor 間の依存関係を保証するための議論と Actor 自体がディペンダブルであることを保証するための議論を分けて記述することができる.

- Goal

  Goal は, システムを保証するために達成する必要がある目的･目標を示す要素である．システムのディペンダビリティや安全性を保証するための目的や目標が，Goal として記述される.

  この要素は，GSN でも定義されている.

- Strategy

  Strategy は，Goal を下位の Goal に分解する際，実施する議論（議論分解の戦略）を示す要素である．Strategy により，Goal がどのような議論を経て下位の Goal に分解されたのかを，明示的に示すことができる.

  この要素は，GSN でも定義されている.

- Evidence

  Evidence は，上位の Goal が達成されることを保証する証拠を示す要素である．通常 d* framework の最下層の Goal を保証するために Evidence が使用される．Goal を保証するための証拠は，具体的に示される必要がある．すなわち，Evidence は現実世界に存在する証拠と関連付けられるかたちで記述される必要がある.

  この要素は，GSN でも定義されている．GSN では，Solution という名称で定義されているが，d* framework では Evidence という名称で定義する.

- Context

  Context は，Goal あるいは Strategy の背景情報を示す要素である．Goal や Strategy に記述される事項に対する詳細な説明等が，Context として記述される.

  この要素は，GSN でも定義されている.

d* framework は，グラフィカルな記法を使用して記述される．d* framework の要素を記述するための記法を Figure 10 に示す．d* framework において，これらの記法が使用される際は，それぞれの図形の中にそれら要素の具体的な内容が記述される．例えば，d* framework の中の Goal には，Goal を示す記法である四角形の中に，Goal の具体的な内容が記述される．尚，ここで示している記法は，D-Case Editor を使用して d* framework を記述する際のものである．D-Case Editor とは，アシュアランスケースの一種である D-Case を作成するために開発されたツールである．しかし，d* framework の作成にも使用することが可能である．D-Case Editor を使用して d* framework を作成する場合，Actor の記述に，D-Case Editor のモジュールを記述するための記法を使用する．

Figure 10 d* framework の要素を記述するための記法（D-Case Editor を使用する場合）

3.3. d* framework における要素間の関係

d* framework には，要素間の関係として，“supported by”，“in context of”，“depend on”，“belong to”の 4 つの種類の関係が定義されている．これら 4 つの種類の関係について，以下に述べる．

- supported by
  supported by 関係は，上位の Goal，Strategy が下位の Goal，Strategy，Solution に保証される関係を示す．

この関係は，GSN でも定義されている．

- in context of
  in context of 関係は, Goal 及び Strategy と Context の関係を示す. Goal 及び Strategy の背景情報が Context により示される場合，この関係が使用される．
  この関係は，GSN でも定義されている．

- depend on
  depend on 関係は，Actor が他の Actor に依存する関係を示す．depend on 関係には，依存元の Actor から依存先の Actor に対する要求が，Goal として記述される．記述される Goal は，サービスやシステムのディペンダビリティを保証する上で達成する必要があり，d* framework における議論対象となる．
  この関係は，d* framework により新規に導入された関係である．

- belong to
  belong to 関係は，d* framework 内に記述されている要素（Goal, Strategy, Context, Evidence）が，特定の Actor または depend on 関係に関する議論の要素であることを示す．Actor に関する議論要素である場合，Actor 自体がディペンダブルであることを保証するための議論を構成する要素が，その Actor に関する議論であることを示す関係となる．また，depend on 関係に関する議論要素である場合，depend on 関係がディペンダブルであることを保証するための議論を構成する要素が，その depend on 関係に関する議論であることを示す関係となる．
  この関係は，d* framework により新規に導入された関係である．

d* framework は，グラフィカルな記法を使用して記述される．d* framework において関係を記述するための記法を Figure 11 に示す．Figure 11 に示した図の中で，角の無い点線の四角形で示された要素は，複数の種類の要素が該当することを示している．尚，任意の Actor と belong to 関係にある d* framework の要素は，該当する Actor の d* framework 内に記述される．そのため，この関係を記述するための記法は，用意されていない．

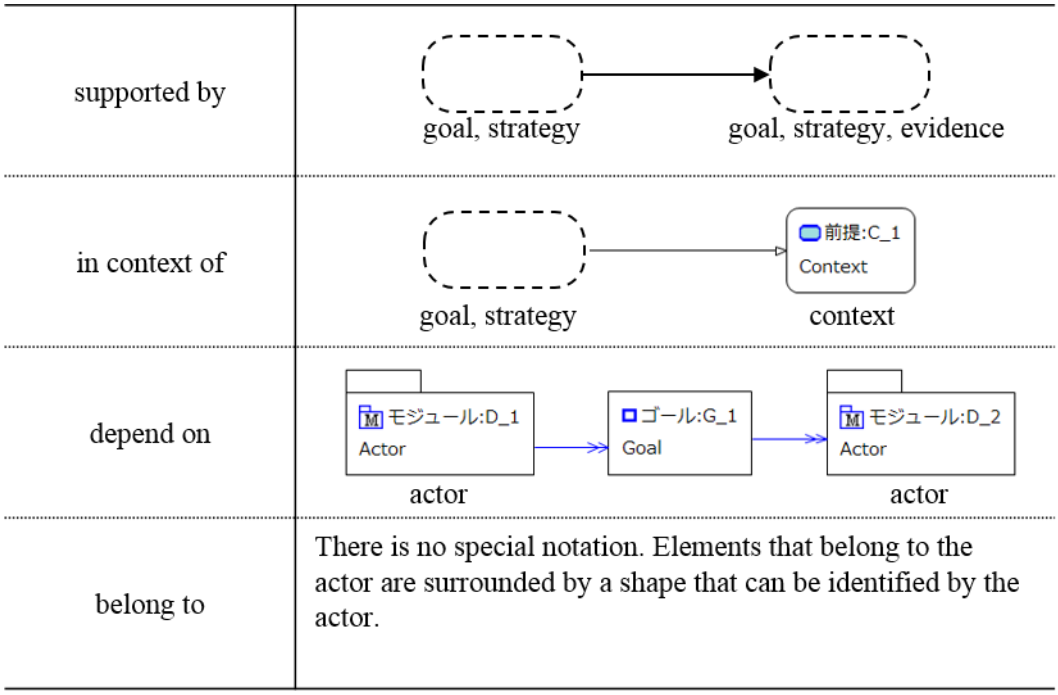

Figure 11 d* framework において要素間の関係を記述するための記法

3.4. d* framework のメタモデル

d* framework において定義されている要素と，それら要素間の関係を明確に示すために，d* framework のメタモデルが定義されている．Figure 12 に，d* framework のメタモデルを示す．このメタモデルでは，Goal，Strategy，Evidence，Context のスーパークラスとして ArgumentElement が定義されている．すなわち，ArgumentElement とは，システムのディペンダビリティを保証するために実施される議論を，記述するために使用される要素である．ArgumentElement と，Actor の間には，belong to 関係が存在する．これは，それぞれの ArgumentElement が特定の Actor に属する関係を，定義可能であることを示している．さらに，ArgumentElement と，他の ArgumentElement の間には，Supported by 関係と In context of 関係が定義されている．これは，GSN においても定義されている，システムの保証に関する議論を記述するための各要素間の関係である．depend on 関係は，2 つの Actor 間の関係として定義されている．depend on 関係は，Actor 間にディペンダビリティの保証に関する依存関係があることを示す関係である．Actor は，ここで定義される依存関係の依存元にも依存先にもなり得る．depend on 関係における依存の目的は Goal として定義され, その Goal もサービスやシステムを保証する上での議論対象となる. 従って，ArgumentElement には，depend on 関係に属する関係が存在する．すなわち ArgumentElement と，depend on 関係の間にも belong to 関係が存在する．尚，メタモデルにおいて depend on 関係は，他の要素と関係を持つ特別な関係として定義されている．

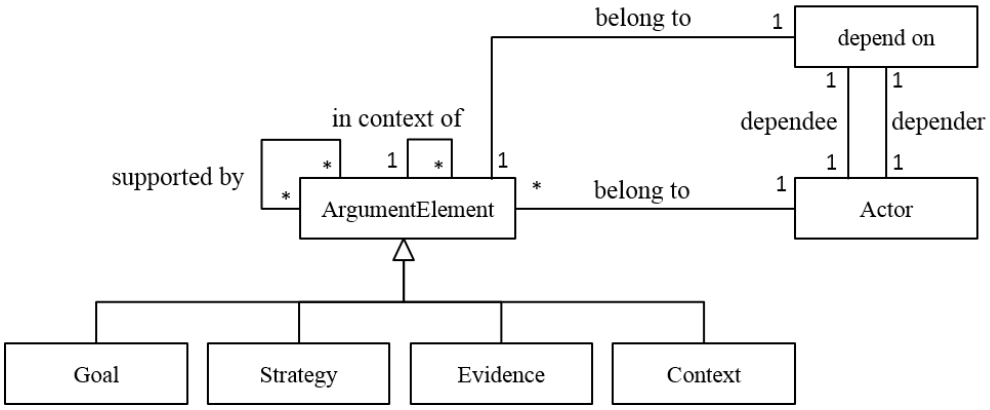

Figure 12 d* framework のメタモデル

3.5. d* framework の種類

d* framework を使用して実施される，システムのディペンダビリティの保証に関する議論には，Actor 間の依存関係の保証に関する議論と，Actor 自体がディペンダブルであることを示す議論の 2 つの種類の議論が存在する．これら 2 つの種類の議論は，異なる種類の d* framework を使用して記述される．すなわち，d* framework には，下記の 2 種類が存在する．

1. Actor が他の Actor に対して責任を遂行することを示す d* framework. すなわち, Actor 間の依存関係を示す d* framework.

2. Actor 自体がディペンダブルであることを示す d* framework

“2. Actor 自体がディペンダブルであることを示す d* framework”は，“1. Actor が他の Actor に対して責任を遂行することを示す d* framework”において記述されている，個々の Actor に対して作成される．そのため，対象とするサービスやシステムに対して，基本的に“1. Actor が他の Actor に対して責任を遂行することを示す d* framework”が 1 つ作成されるのに対して, “2. Actor 自体がディペンダブルであることを示す d* framework”は複数作成される．

3.6. d* framework の作成プロセス

d* framework を作成するために，4 つのタスクからなる作成プロセスが定義されている．Figure 13 に 4 つのタスクからなる d* framework の作成プロセスを示す．この作成プロセスは，4 つのタスクを繰り返し実行するプロセスとなっている．d* framework において議論を構成する要素は互いに関係しているので，d* framework に対する 1 つの変更が，d* framework 全体に波及していく可能性がある．そのため，このような繰り返し実行するプ

ロセスが必要となる．例えば，d* framework に Actor を追加する場合，新しい Actor と既存の Actor の間のディペンダビリティに関する議論を実施する必要がある．その結果，他の新規の Actor が必要になる可能性がある．

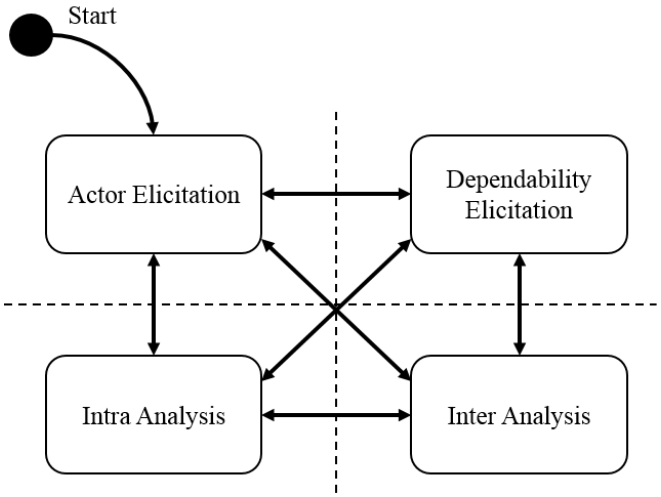

Figure 13 d* framework の作成プロセス

下記に，ここで定義する 4 つのタスクについて述べる．

- Actor Elicitation
  このタスクでは, 対象システムの構成要素を, d* framework の Actor として記述する．d* framework 作成の初期段階では，システム構成定義書などを利用して Actor が記述される．d* framework 作成の初期段階以外では，Inter Analysis もしくは Intra Analysis を実施した結果として，Actor が抽出される可能性がある．

- Dependability Elicitation
  このタスクでは, Actor 間のディペンダビリティの保証に関する依存関係が識別される．

- Inter Analysis
  このタスクでは，Actor 間のディペンダビリティの保証に関する議論を実施する．さらに，実施された議論の内容を，GSN を使って記述する．実施された議論を保証するため，他の Actor が必要となる可能性がある．既に定義されている Actor に，その議論を保証することのできる Actor が存在しない場合，必要に応じて Actor Elicitation を実施し，適切な Actor を追加する必要がある．

- Intra Analysis

  このタスクでは，Actor 内のディペンダビリティの保証に関する議論を実施する．さらに，実施された議論の内容を，GSN を使って実施する．議論の対象となっている Actor が，他の Actor から依存されている場合，依存されているディペンダビリティに関する Goal の達成を考慮して，議論を実施する必要がある．実施された議論を保証するため，他の Actor が必要となる可能性がある．既に定義されている Actor に，その議論を保証することのできる Actor が存在しない場合，必要に応じて Actor Elicitation を実施し，適切な Actor を追加する必要がある．

3.7. d* framework の例

3.5 節に示した通り，d* framework には 2 つの種類が存在する．ここでは，2 つの種類の d* framework について，それぞれ 1 つずつ例を示す．尚，これら 2 つの例は，d* framework を説明するために示したものであり，現実のシステムに対応した例ではない．

まず，“Actor が他の Actor に対して責任を遂行することを示す d* framework”の例を Figure 14 に示す．この例では，“ユーザ（Actor）”が，“システム（Actor）”に依存している状況が示されている．また，Actor 間の依存関係，すなわち依存の目的は“信頼のおけるサービスが提供される（Goal）”として記述されている．さらに，その Goal を保証するための議論が GSN を使用して記述されている．

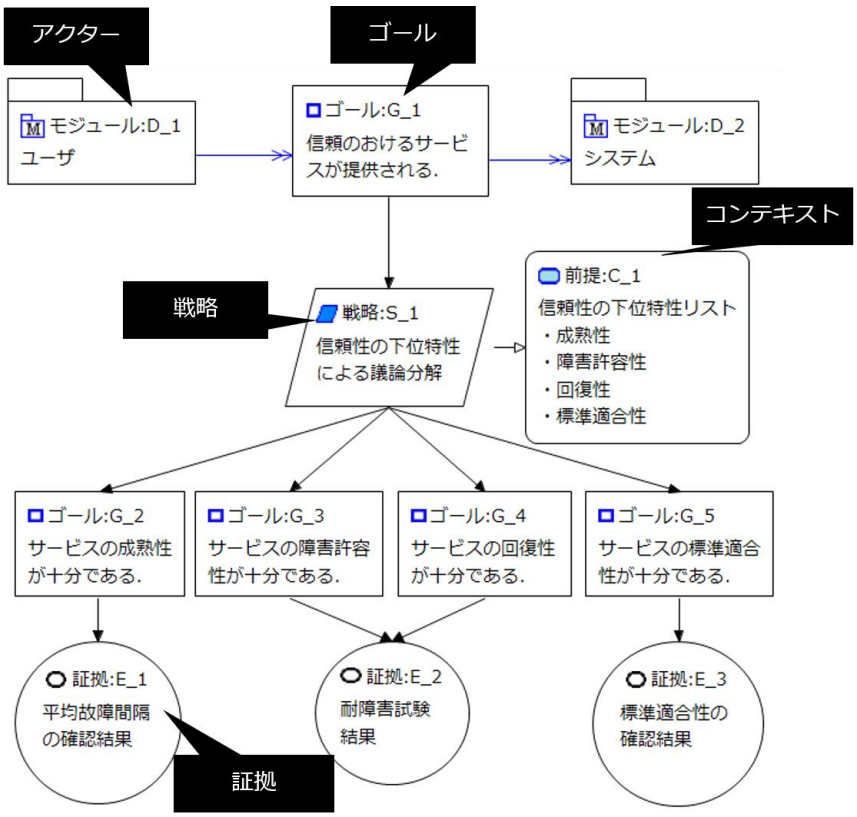

Figure 14 “Actor が他 Actor に対して責任を遂行することを示す d* framework”の例

次に“Actor 自体がディペンダブルであることを示す d* framework”の例を Figure 15 に示す.“Actor 自体がディペンダブルであることを示す d* framework”は,“Actor が他の Actor に対して責任を遂行することを示す d* framework”の中に記述されている Actor に対して作成される.この例では,Figure 14 に記述されている Actor の 1 つである“システム”を対象として,“システム自体がディペンダブルであることを示す d* framework”が示されている.“Actor 自体がディペンダブルであることを示す d* framework”の構造は,GSN と同様なものとなる.“Actor 自体がディペンダブルであることを示す d* framework”には,対象としている Actor が,達成すべき Goal に関する議論が記述される.そのため,d* framework の中に Actor が記述されることはない.

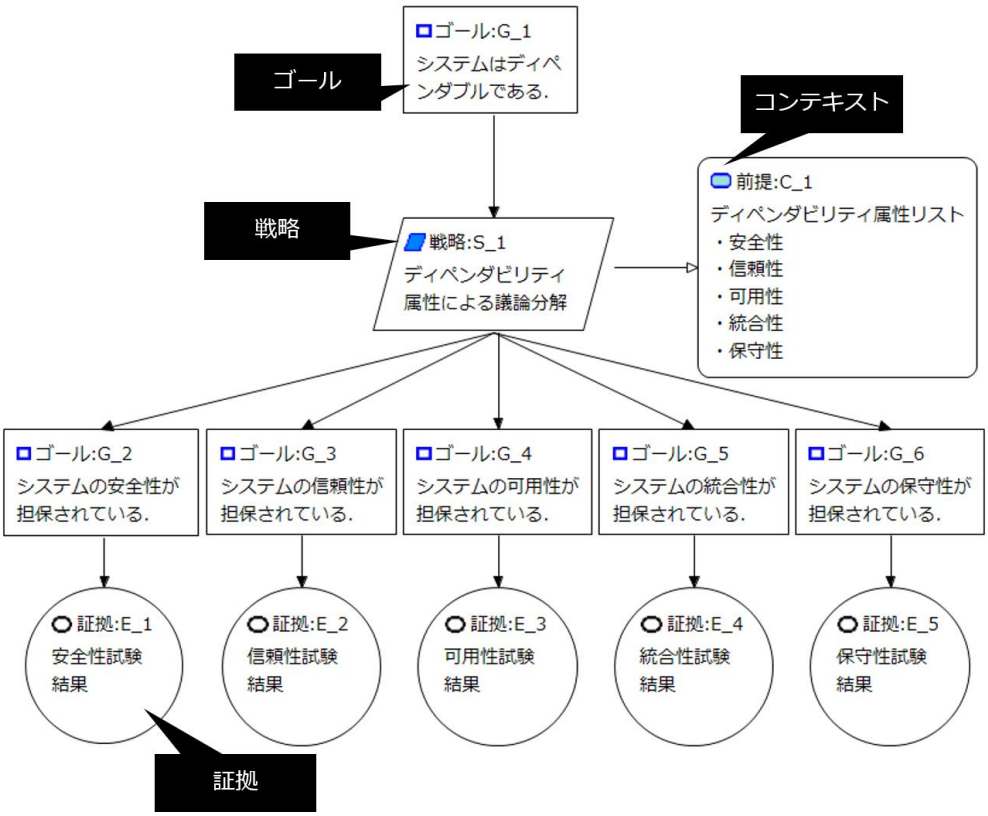

Figure 15 “Actor 自体がディペンダブルであることを示す d* framework”の例

3.8. まとめ

本研究では，アシュアランスケースの作成手法である d* framework を提案した．d* framework とは，Actor の概念を導入したアシュアランスケースの作成手法である．Actor の概念を導入するため，d* framework には，システムの保証に関する議論を構成する新規の要素として，Actor が導入されている．Actor を使用することで，d* framework が対象とするサービスやシステムの構成要素を，d* framework 内で示すことができる．従って，d* framework を使用することで，複数のシステムや複数のコンポーネントから構成されるサービスやシステムの保証に関する議論を，容易に記述できるようになることが期待される．

本章で提案した d* framework の有効性の検証は 6 章において実施する．

4. d* framework に対する責任属性の導入

本章では，d* framework に対する責任属性の導入に関する研究について述べる[110]．この研究は，“従来のアシュアランスケースでは，実施される議論を保証する責任主体の概念が考慮されていない”という既存のアシュアランスケースの欠陥に対応する研究である．

4.1. はじめに

GSN や CAE で記述された従来のアシュアランスケースでは，サービスやシステムのディペンダビリティを保証するための議論に対する責任の所在を，明示的に示すことができなかった．これでは，実際に障害が起きた際，アシュアランスケースを使って責任の所在を明確にすることができない．すなわち，アシュアランスケースを使用した障害の原因究明を十分に実施することができないという問題があった．この問題に対処するためには，アシュアランスケースに記述されている，サービスやシステムのディペンダビリティの保証に関する議論に対して，責任の所在をアシュアランスケースの中で明確にする必要がある．この節では，そのための方法として，d* framework に対する責任属性の導入を提案する．ここでの責任属性とは，対象の d* framework において実施されているサービスやシステムのディペンダビリティを保証するための議論に対して，誰が責任を持つのかを示す属性である．責任属性を導入することにより，実際に障害が発生した際，アシュアランスケースを使った説明責任の明確化が可能となる．

アシュアランスケースに対する責任属性の導入に関する研究として，[81]がある．[81]では，D-Case[23]に対する責任属性の導入が検討されている．本章で実施する研究は，[81]で示される責任属性の導入方法を参考にしている．

4.2. 責任属性の導入の概要

作成された d* framework が実際に役に立つためには，d* framework に記述された議論が有効なものになっている必要がある．すなわち，d* framework に記述されている議論の内容に対する責任の所在が明確になっており，議論の内容が責任を持って遂行される必要がある．責任属性の導入とは，d* framework に記述されている議論の内容について，責任の所在を明確にすることである．本研究では，d* framework において議論の内容を示す要素（Actor，Goal，Strategy，Context，Evidence）に対して，責任を持つことが可能な任意の人や組織を明示的に定義することにより，d* framework 内の議論に対する責任の所在を明確化する．

4.3. ソフトウェア工学の各種手法における責任属性

従来，ソフトウェア工学の分野において，いくつかの責任属性に関係する研究が実施されてきた．これらの研究において責任属性は，ソフトウェア工学において提案されたモデルの中で定義されている．責任属性が定義されたモデルには，次のようなものがある．

Lamsweerde は，ソフトウェア工学の一分野である要求工学の手法において，要求分析を実施するためのモデルの 1 つとして，エージェントモデルを提案している[82][92]．このエージェントモデルの中で，責任属性の概念が用いられている． Sommerville 等は，ソシオテクニカルシステムにおける要求とリスクを抽出するためのモデルとして，責任モデルを提案している[93][94][95]．Feltus 等は，組織構造の確認と組織のポリシーに関係した問題の発見のために使用するモデルとして，責任モデルを提案している[96][97][98]．Boness 等は, システム開発の意図を評価するためのモデルとして, 責任モデルを提案している[99]. Strens 等は，システムの要求を抽出し特定するために，責任モデルを提案している[100]. これらのモデルについて，提案者，モデルの作成目的，モデルの主な構成要素を整理したものを Table 7 に示す.

Table 7 従来の研究において提案されている責任属性が導入されたモデル

提案者	作成目的	主な構成要素
Lamsweerde	システムに対する責任の所在の可視化，およびシステムのスコープの定義.	Goal, Agent, Operation, Object
Sommerville	ソシオテクニカルシステムの構造とディペンダビリティに関する調査及び診断.	Responsibility, Resource, Agent
Feltus	組織の方針に対する違反（ポリシー違反）の発見.	Organization, Responsibility, User (Agent), Task
Boness	システム開発の目的の評価.	Goal, Actor (Agent)
Strens	システム開発の要求の特定と仕様化.	Responsibility, Resource, Agent

Table 7 に示されているモデルは, それぞれ異なる目的で作成されるモデルである. しかし，責任属性を導入しているという点では，5 つのモデルは共通している．責任属性が導入されたこれらのモデルには，責任属性を記述するための基本的なパターンが存在する．基本的なパターンとは，“責任を持つ主体”が“責任の対象”に対して責任を持つというパターンである．これらのモデルでは，このパターンを記述するために，“責任を持つ主体”と“責任の対象”の 2 つの概念が使用されている. “責任を持つ主体”の概念を示す要素としては，Agent，User，Actor が定義されている．また，“責任の対象”の概念を示す要素としては，Object，Resource，Responsibility，Goal 等の様々な要素が定義されている．“責任を持つ主体”の概念は，Agent，User，Actor というように限定された要素として定義されているが，“責任の対象”の概念は，“責任を持つ主体”が責任を持つ対象として広く定義されているという特徴がある．本研究では，従来の研究における責任属性が導入されたモデルが持つ特徴を参考にして，d* framework に責任属性を導入する.

4.4. d* framework に対する責任属性の導入方法

d* framework とは，Actor が導入されたアシュアランスケースである．d* framework に責任属性を導入するにあたって，ソフトウェア工学における責任属性が導入されたモデルが持つパターンを利用する．すなわち，“責任を持つ主体”が，“責任の対象”に対して責任を持つ関係を，d* framework の中で記述できるようにすることで，d* framework に責任属性を導入する．ここで，“責任の対象”は，d* framework で実施されている議論そのものとなる．すなわち，“責任の対象”は，従来の d* framework において定義されている要素であると考えることができる．そのため，“責任の対象”を示すために新規に要素を定義する必要はない．一方，“責任を持つ主体”は，従来の d* framework では，定義されていない概念である．そこで，d* framework において“責任を持つ主体”を記述する新規の要素として Agent を定義する．d* framework では，Agent として定義できる対象を，人や組織に限定する．これは，システムやコンポーネント等，人により構成されていない対象は，一般的な社会の中で責任を取ることができないという点を考慮したためである．すなわち，d* framework における Agent は，主体的に責任をとることのできる要素として定義される．また，Agent が“責任の対象”に対して責任を持つ関係として，新規に responsible for 関係を定義する．d* framework に対する責任属性の導入において，新規に定義する Agent 及び responsible for 関係の定義を以下に示す．

- Agent
  Agent は，d* framework の他の要素に対して，責任を持つ人あるいは組織を示す要素である．すなわち，d* framework が対象とするサービスやシステムのディペンダビリティを保証するための議論に対して，責任を持つ人あるいは組織が Agent として定義される．

- Responsible for
  Responsible for 関係は，Agent が d* framework 内の Agent 以外の要素に対して責任を持つ関係を示す．

従来の d* framework においても，Agent に近い概念を持つ要素が，Actor として定義されている．Actor とは，d* framework が対象とするサービスやシステムの構成要素を示す要素である．人，組織，機械的なシステム，IT システム，サービス等が，Actor として定義される．d* framework に対して責任属性を導入する際，人や組織等の人が直接関係する構成要素以外は，d* framework に記述される議論に対して責任を取ることができないという点を考慮する必要がある．機械的なシステム，IT システム等は，それら自身では責任を取ることができない．そのため，d* framework で既に定義されている Actor を，そのまま“責

任を持つ主体”として定義することは出来ない．それゆえ，責任属性を導入するにあたり，d* framework で既に定義されている Actor とは別に，“責任を持つ主体”を示す要素として Agent を定義する必要がある．

d* framework では，現実世界における同一の対象が，Agent 及び Actor として同時に記述される可能性がある．例えば，システム開発者は，サービスの保証に関する議論に対して責任を持つ必要があると同時にサービスの構成要素でもあると考えられるので，Agent と Actor の両方に定義される可能性がある．

4.5. 責任属性が導入された d* framework のメタモデル

責任属性が導入された d* framework のメタモデルを Figure 16 に示す. このメタモデルは, Figure 12 に示されている d* framework のメタモデルを，責任属性の導入に伴い拡張したものである．メタモデルには，d* framework の要素と，それら要素間の関係が示されている．ArgumentElement は，Goal，Strategy，Evidence，Context のスーパークラスとして定義されている．すなわち，ArgumentElement とは，d* framework においてシステムのディペンダビリティの保証に関する議論を記述するために使用される要素である．ArgumentElement と，Actor の間には，belong to 関係が定義されている．この関係は，ArgumentElement が特定の Actor に属することを示している. さらに, ArgumentElement と，他の ArgumentElement の間には supported by 関係と in context of 関係が定義されている．これらの関係は，GSN でも定義されているシステムのディペンダビリティの保証に関する議論を記述するための各要素間の関係である．depend on 関係は，2 つの Actor の間の関係として定義されている．depend on 関係は，Actor 間にディペンダビリティの保証に関する依存関係があることを示す関係である．Actor は，ここで定義される依存関係の依存元にも依存先にもなり得る．depend on 関係における依存の目的は Goal として定義され，その Goal もサービスやシステムを保証する上での議論対象となる．従って，ArgumentElement には，depend on 関係に属する関係も存在する．すなわち ArgumentElement と，depend on 関係の間にも belong to 関係が存在する．責任属性を導入するために追加される要素である Agent は, Actor のサブクラスとして定義されている．また, Agent と，Agent 以外の要素（Actor 及び ArgumentElement）の間には，responsible for 関係が定義されている．これは，Agent が他の要素に対して，責任を持つ関係を示している．

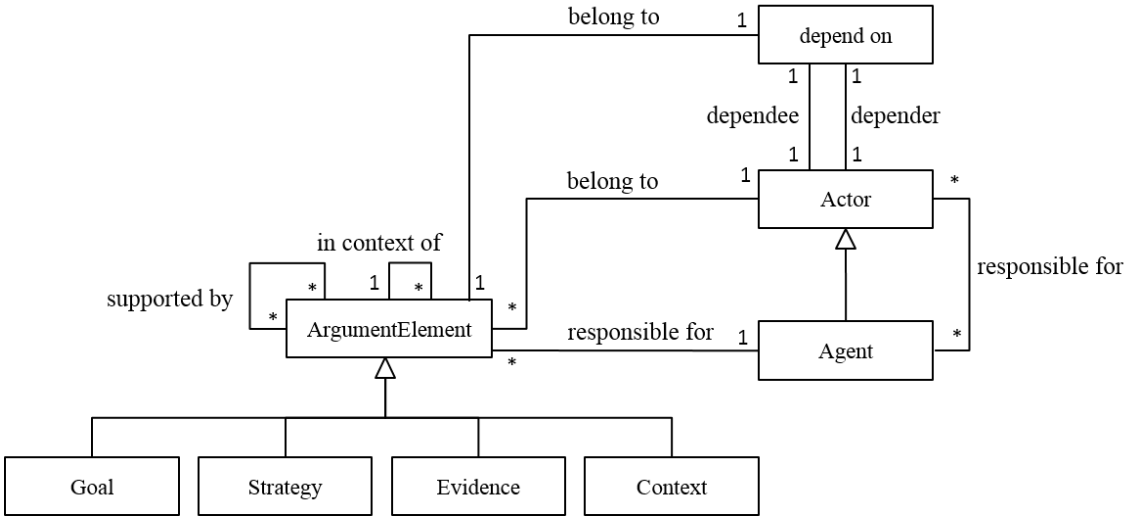

Figure 16 責任属性が導入された d* framework のメタモデル

4.6. 責任属性を導入するための記法

この節では，d* framework に責任属性を導入するにあたり，新規に追加した要素（Agent）と関係（responsible for 関係）の記法について述べる．本研究では，d* framework における責任属性の記述に，グラフィカルな記法を使用する．責任属性を導入するために使用する記法として，グラフィカルな記法以外の記法も考えられる．例えば，d* framework 内の要素に記載される文章の書式を，その要素に対する責任の所在を明示的に示すことができるように定義することなどが考えられる．しかし，従来の d* framework では，議論の要素や要素間の関係を記述するための記法として，グラフィカルな記法が使用されている．そのため，新規に定義される要素や関係についても，グラフィカルな記法を使用する方が容易に理解できることが考えられる．そこで，本研究では，責任属性の導入に伴う要素の記法として，グラフィカルな記法を提案する．

新規に追加する記法を Figure 17 に示す．Agent の記法には，左右が孤を描く 2 重線の長方形を使用する．また，Agent が Agent 以外の要素に対して責任を持つ関係を示すために導入される responsible for 関係には，Agent が責任を持つ要素を点線で囲む記法を使用する．

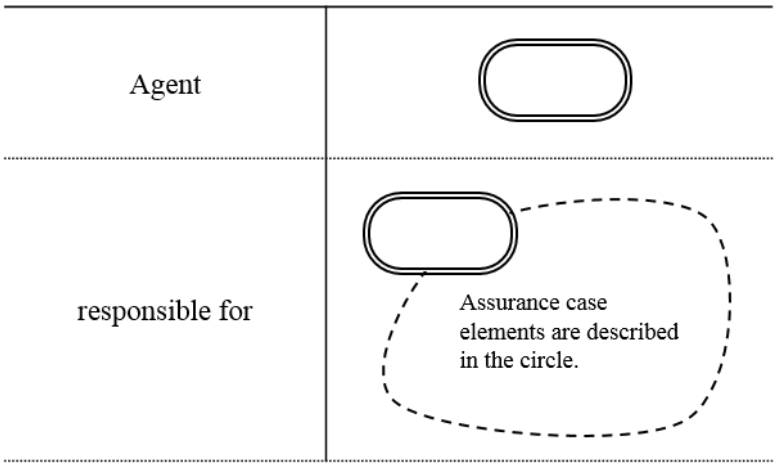

Figure 17 責任属性を導入するために d* framework に導入される記法.

4.7. 責任属性が導入された d* framework の例

この節では，責任属性が導入された d* framework の例について述べる．これらの例は，本研究において作成されたものである．また，作成した例によって，責任属性が導入された d* framework が広範囲な分野に適用できるか確かめる．

本研究では，責任属性が導入された d* framework の例として，以下の 3 つのシステムを対象として d* framework を作成した．

- 例 1 : エレベーターシステム
- 例 2 : アプリケーション（AP）ダウンロードシステム
- 例 3 : LAN デバイス管理システム

例 1:エレベーターシステムの例では，“Actor 間の依存関係を示す d* framework”と“Actor 自体がディペンダブルであることを示す d* framework”の 2 つの d* framework の例が示されている．また，他の 2 つの例では，“Actor 間の依存関係を示す d* framework”の例のみが示されている．まず，本節における例で使用している記法について説明し，その後，それぞれの例について説明する．

4.7.1. 例において使用する d* framework の要素の記法

ここでは，本節の例において使用する d* framework の要素の記法について述べる．Figure 18 に記法を示す．

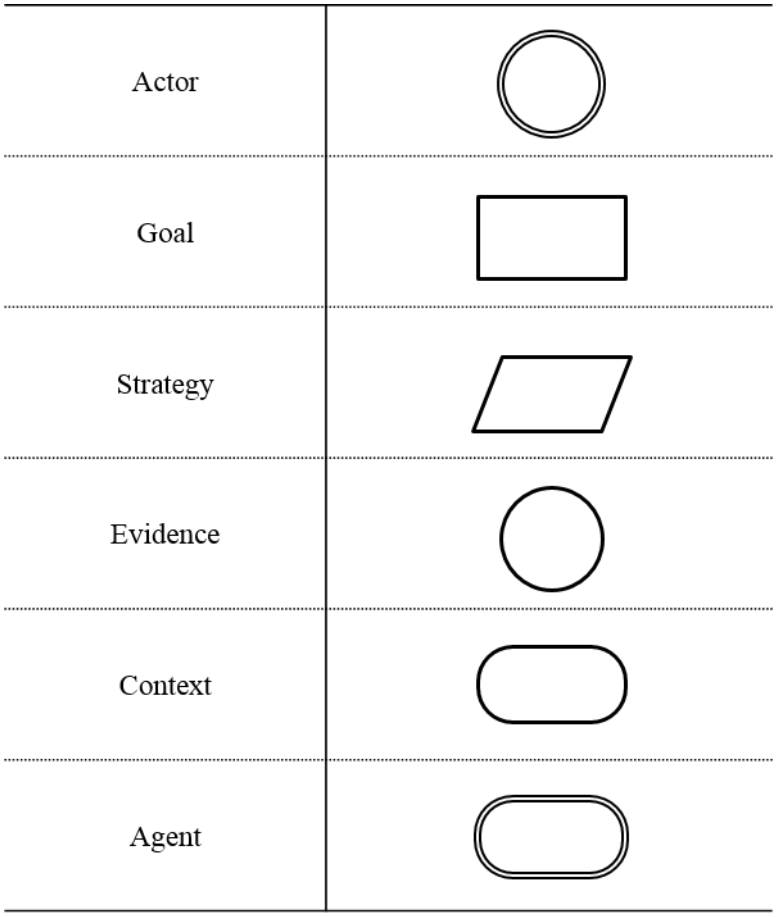

Figure 18 4 章で使用する d* framework の要素を記述するための記法

Figure 19 に，d* framework における要素間の関係を記述するための記法を示す.

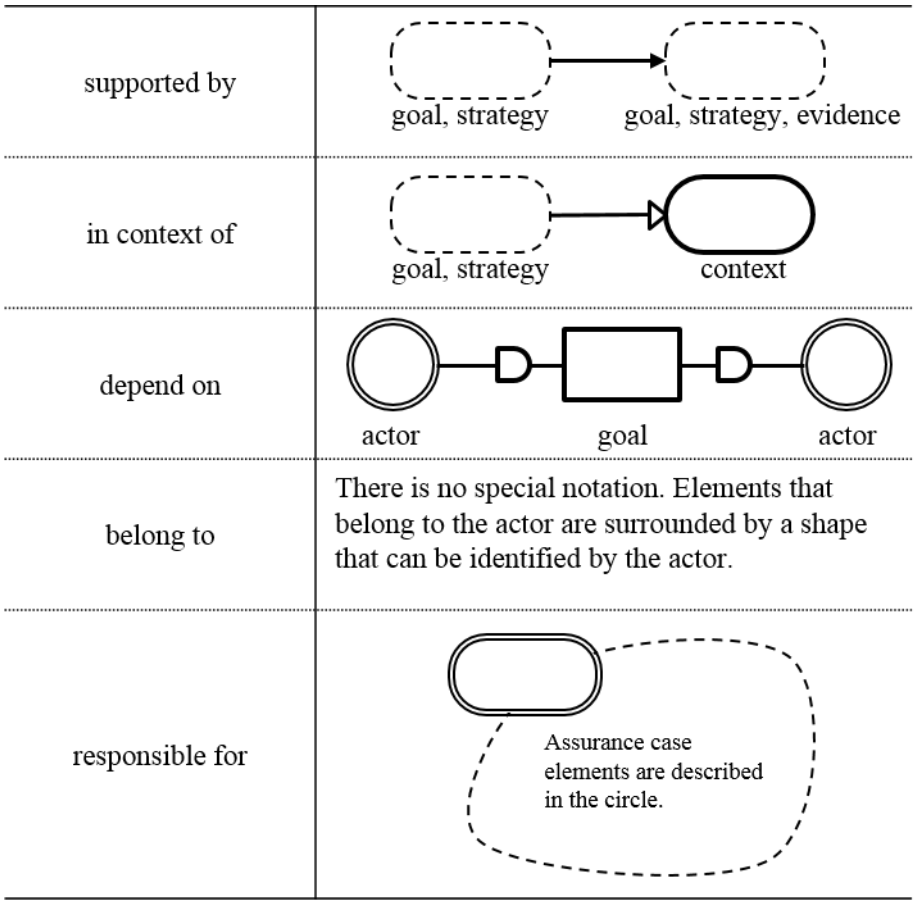

Figure 19 4 章で使用する d* framework の要素間の関係を記述するための記法

4.7.2. 例 1：エレベーターシステム

エレベーターシステムとは，エレベーターを運行するためのシステムである．このシステムは，Rope，Door，User，Cage，Control panel，Balance weight，Rupture detection equipment，Shock mitigation equipment，Speed regulator，Lifting equipment の 10 個の要素から構成されている．このエレベーターシステムに対して，以下に示す 2 つの種類の d* framework を作成した．

1. Actor が他の Actor に対して責任を遂行することを示す d* framework. すなわち, Actor 間の依存関係を示す d* framework.

2. Actor 自体がディペンダブルであることを示す d* framework.

まず，“Actor 間の依存関係を示す d* framework”について述べる．Figure 20 に作成した d* framework を示す．この d* framework では，エレベーターシステムの構成要素である Rope, Door, User, Cage, Control panel, Balance weight, Rupture detection equipment,

Shock mitigation equipment，Speed regulator，Lifting equipment が，Actor として記述されている．さらに，それぞれの Actor 間に，Actor 間のディペンダビリティの保証に関する議論が記述されている．また，これらの議論に対して責任を持つ Agent として，development section，IT system development section，maintenance section，wire company，user が記述されている．これら Agent と d* framework に記述されている他の要素間の関係は，responsible for 関係として記述されている．尚，Agent として記述されている要素は，全て人か人が所属する組織となっている．

この例で記述されている Actor は，2 つの種類に分けることができる．1 つめの種類は，ただ 1 つの Agent と responsible for の関係にある Actor である．例えば，Cage は，development section という 1 つの Agent が責任を持つ Actor として記述されている．2 つめの種類は，複数の Agent と responsible for の関係にある Actor である．例えば，Rope は，maintenance section と wire company の 2 つの Agent が責任を持つ Actor として記述されている．2 つの Agent が責任を持つ Actor は，Agent の責任範囲を示す点線の枠（responsible for 関係）に，2 重に囲まれている．Rope は，maintenance section の責任範囲を示す枠と wire company の責任範囲を示す枠の両方の枠に囲まれている．d* framework 内の Agent 以外の全ての要素は，任意の Agent と responsible for 関係を持つ必要がある．d* framework 内に、任意の Agent と responsible for 関係を持たない要素が残っている場合，その d* framework における議論は，不十分であると考えられる．そのような d* framework では，責任が明確になっていない要素について，どの Agent が責任を持つのかを議論する必要がある．

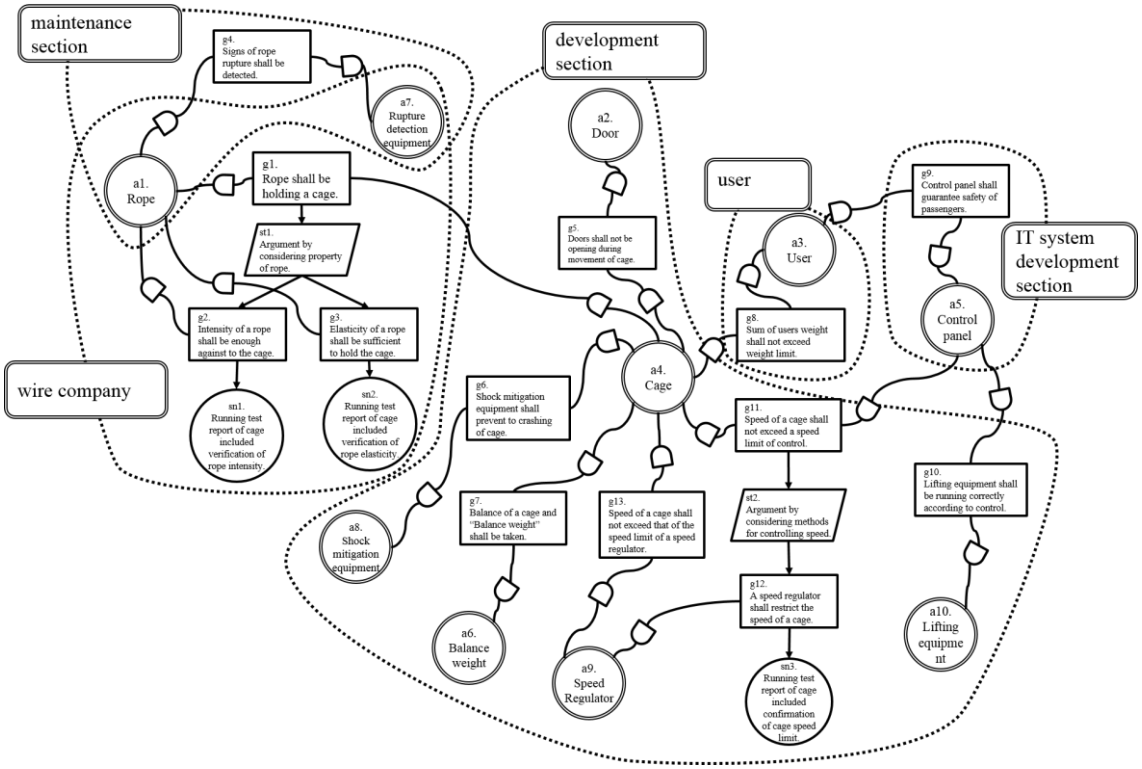

Figure 20 Actor 間の依存関係を示す d* framework の例（エレベーターシステム）

次に，“Actor 自体がディペンダブルであることを示す d* framework”について述べる．Figure 21 に作成した d* framework を示す. この d* framework は, Figure 20 の中で Actor として記述されている Rope について，Rope がディペンダブルであることを示したものである. Figure 20 に示された d* framework の中で, Rope は 2 つの Agent が責任を持つ Actor として記述されている．そのため，Actor 自体がディペンダブルであることを示す d* framework では，Actor 内の議論に使われている要素が，責任関係にもとづいて 2 つのグループに分割されている．分割された要素は，それぞれ適切な Agent の責任範囲を示す点線の枠内に記述されている. “Actor 自体がディペンダブルであることを示す d* framework”でも，Actor 内の全ての要素は，任意の Agent と responsible for 関係を持つ必要がある．任意の Agent と responsible for 関係にない要素が d* framework 内に残っている場合，その d* framework における議論は，不十分であると考えられる．そのような d* framework では，責任が明確になっていない要素について，どの Agent が責任を持つのかを議論する必要がある.

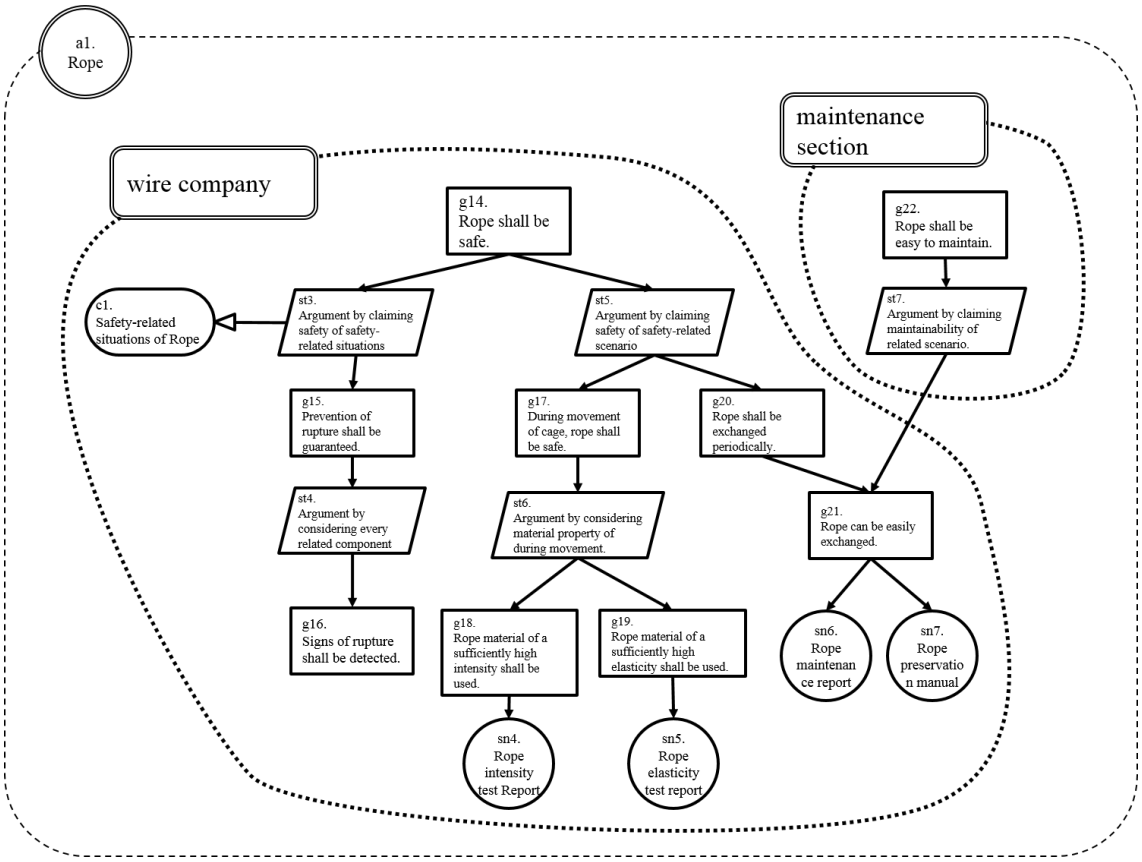

Figure 21 Actor 自体がディペンダブルであることを示す d* framework の例（エレベーターシステム）

4.7.3. 例 2：AP ダウンロードシステム

AP ダウンロードシステムは，ユーザがアプリケーション（AP）をダウンロードするためのシステムである．このシステムを使用することで，ユーザはシステムからユーザの IC カードにアプリケーションをダウンロードすることができる．このシステムは，Issuer of cards，Service provider，AP download system，Memory card，User から構成されている．ここでは，AP ダウンロードシステムについて，“Actor 間の依存関係を示す d* framework”を示す．Figure 22 に，作成した d* framework を示す．この d* framework では，AP ダウンロードシステムの構成要素である Issuer of cards，Service provider，AP download system，Memory card，User が Actor として記述されている．さらに，それぞれの Actor 間に，Actor 間のディペンダビリティの保証に関する議論が記述されている．また，これらの議論に対して責任を持つ Agent として，Issuer of cards，Service provider，System developer，User が記述されている．これら Agent と他の要素間の責任関係は，responsible for 関係として記述されている．この例では，Agent 以外の全ての要素は，ただ 1 つの Agent との間に responsible for 関係を持っている．また，Issuer of card，Service provider，User は，Actor と Agent の両方の種類の要素として記述されている．これは，

同一の対象が Actor と Agent の双方の要素として記述できるという例を示している．

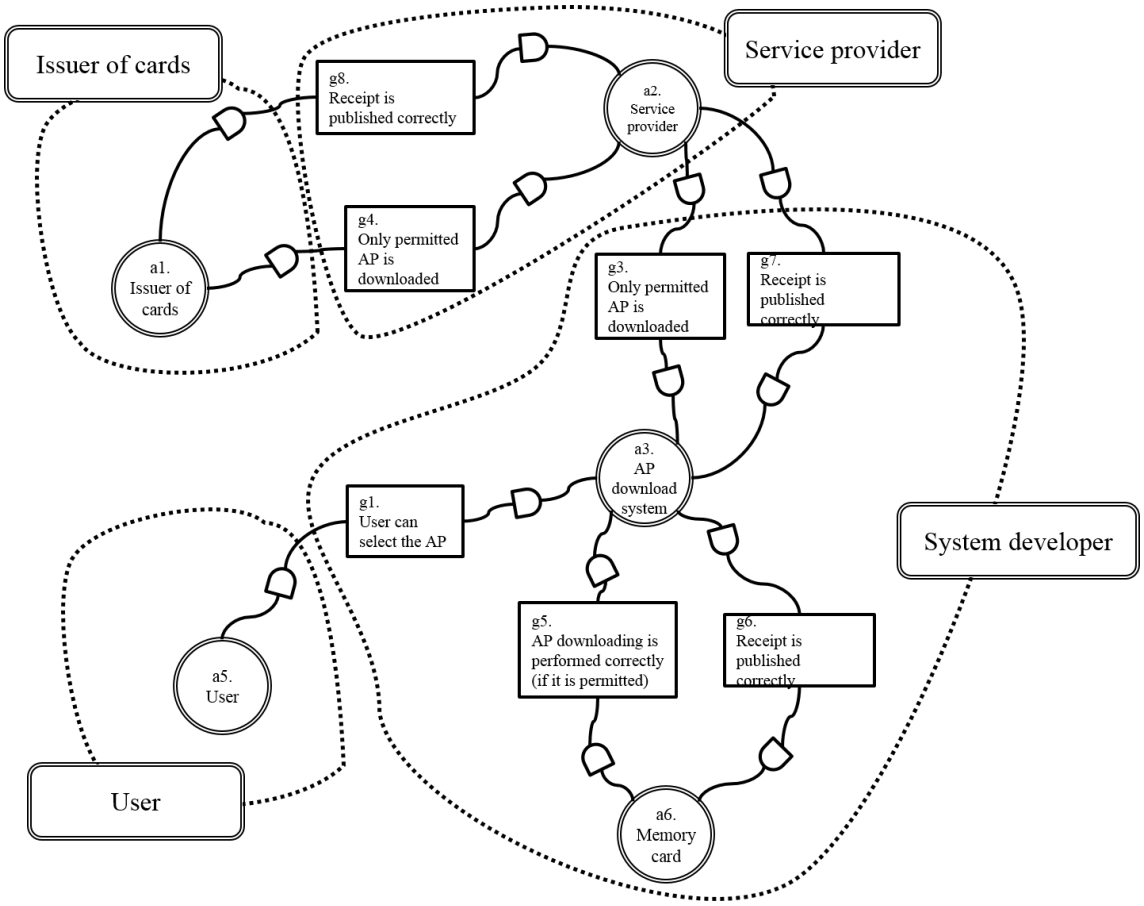

Figure 22 Actor 間の依存関係を示す d* framework の例（AP ダウンロードシステム）

4.7.4. 例 3：LAN デバイス管理システム

LAN デバイス管理システムは，LAN 内の不正なデバイスを検出し，遮断するためのシステムである．このシステムは，Surveillance server，Manager，Sensor，LAN device から構成されている．ここでは，LAN デバイス管理システムについて“Actor 間の依存関係を示す d* framework”を示す. Figure 23 に, 作成した d* framework を示す. この d* framework には, LAN デバイス管理システムの構成要素である Surveillance server, Manager, Sensor, LAN device が Actor として記述されている．さらに，それぞれの Actor 間に，Actor 間のディペンダビリティの保証に関する議論が記述されている．また，これらの議論に対して責任を持つ Agent として，Service provider，Sensor vender，User が記述されている．これら Agent と他の要素間の責任関係は，responsible for 関係として記述されている．この例では，記述されている全ての Actor が，人または組織ではない．そのため，システムのディペンダビリティの保証に関する議論に責任を持つことのできる人あるいは組織が，別途 Agent として記述されている．ここでは，Service provider，Sensor vendor，User の 3

つの Agent が記述されている．

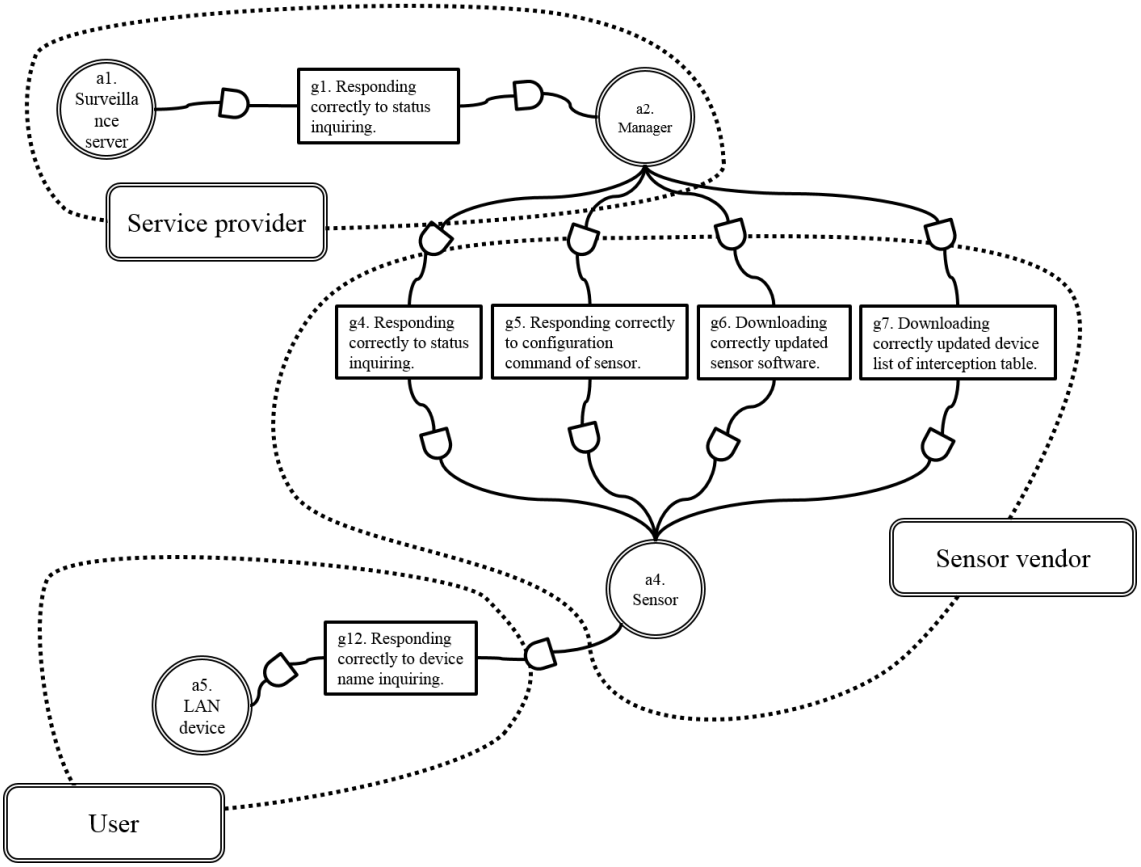

Figure 23 Actor 間の依存関係を示す d* framework の例（LAN デバイス管理システム）

4.8. 考察

4.8.1. d* framework 内の議論の構造と組織構造

責任属性が導入された d* framework おいて Agent として記述される人や組織は，一般的に，階層化された組織構造に属している．このような組織構造では，責任と権限が上位から下位へ委譲される．一方，d* framework には，システムのディペンダビリティの保証に関する議論が，階層構造を使って記述されている．すなわち，ディペンダビリティの保証に関する上位の Goal を，1 つまたは複数の下位の Goal が保証するという関係を記述し構造化することで，システムのディペンダビリティの保証に関する議論全体が記述されている．

d* framework に記述されている Agent が，階層化された組織構造に属している場合，それら Agent が責任関係を持つ d* framework の要素も，d* framework の議論の構造において，同様の階層構造を持つことが望ましいと考えられる．例えば，Agent A が Goal A に対して責任を持つ関係があり，Agent B が Goal B に対して責任を持つ関係がある場合を考える．

d* framework の中で Goal A が Goal B によって保証されている関係がある場合，Agent A と Agent B は同一の Agent であるか，組織構造において，Agent B に対して Agent A が上位に位置するような階層関係があることが望ましい．Agent B に対して Agent A が下位であるような階層関係になっている場合，Agent 間の権限委譲がうまくいっていないことが考えられる．なぜなら，Goal B に対する責任と Goal B を達成することにより達成される Goal A に対する責任では，一般的に Goal A に対する責任の方が大きいと考えられるからである．また，Goal B に対する責任が，Goal A に対する責任に包含される場合もあると考えられる．組織構造において下位である Agent A が，上位である Agent B より大きな責任を持つのは不都合が生じる可能性が高い．したがって，このような箇所は，議論の見直しが必要な可能性がある．d* framework に責任属性を導入することにより，このようなチェックが可能となる．

4.8.2. Actor と Agent

本研究では，責任属性を d* framework に導入するため，新規の要素として Agent を定義した．Agent は，d* framework において，従来から定義されている Actor と類似した要素である．また，AP ダウンロードシステムの例において示した通り，d* framework 内に Agent と Actor の両方の要素に対して，同じ対象が記述される可能性がある．しかし，本研究では，Agent と Actor を明確に区別して定義した．なぜなら，責任属性という概念を考える際，Agent の概念と Actor の概念は異なっていると考えられるからである．
また，d* framework に責任属性を導入するにあたっては，Agent を Actor のサブクラスとして定義した．これは，Agent と Actor の下記の 2 つの関係を示している．

- Actor として記述されている対象が人や組織であった場合，d* framework 内の議論に対して責任を持つことができるので，Agent としても記述可能である．逆に，Actor として記述されている対象が人や組織以外であった場合，例えばサブシステムやシステムのコンポーネントであった場合，それ自体では d* framework 内の議論に対して責任を持つことができないので，Agent としては記述できない．

- Agent として記述できる対象は人か組織なので，全て Actor としても記述可能である．

d* framework に対する責任属性の導入を検討することにより，責任属性の導入に Agent が要素として必要であることがわかった．Agent は，システムの保証を議論する上で，責任の所在を明確にするという観点から必要な要素である．d* framework において従来から使用されていた Actor とは別に Agent を導入することで，システムのディペンダビリティの保証に関する議論を，議論に対する責任の所在という観点から，より明確にすることができると考えられる．

4.8.3. 責任属性の導入に伴う記法の拡張

本研究では，責任属性の導入に伴う d* framework の記法の拡張方法として，グラフィカルな記法を提案した．責任属性の導入にあたっては，グラフィカルな記法を導入する以外の他の方法も考えられる．例えば，下記の方法が考えられる．

- d* framework 内の要素に記載される文章の書式を，その要素に対する責任の所在が明示的に示されるように定義する方法.

- d* framework 内のそれぞれの要素に対する責任の所在を示す表を，d* framework とは別に作成する方法.

今後，上記の方法についても記法を定義し，どの方法が d* framework の作成目的と照らし合わせて適切であるかを評価する必要がある．

4.8.4. メタモデル

責任属性を導入した d* framework のメタモデルを Figure 16 に示した．このメタモデルには，責任属性を導入するために定義した新規の要素が含まれている．Figure 24 では，責任属性の導入に伴い，従来の d* framework のメタモデルに対して，新規に追加した部分を網掛けで示している．新規に追加された Agent は，Actor のサブクラスとして定義されている．また，Agent と，Actor 及び ArgumentElement（Goal，Strategy，Evidence，Context）の間には，responsible for 関係が定義されている．

本研究では，従来から定義されている d* framework の構造に対しての変更は実施していない．本研究における d* framework に対する責任属性の導入では，従来の d* framework が持つ構造を，そのまま利用しているという特徴がある．

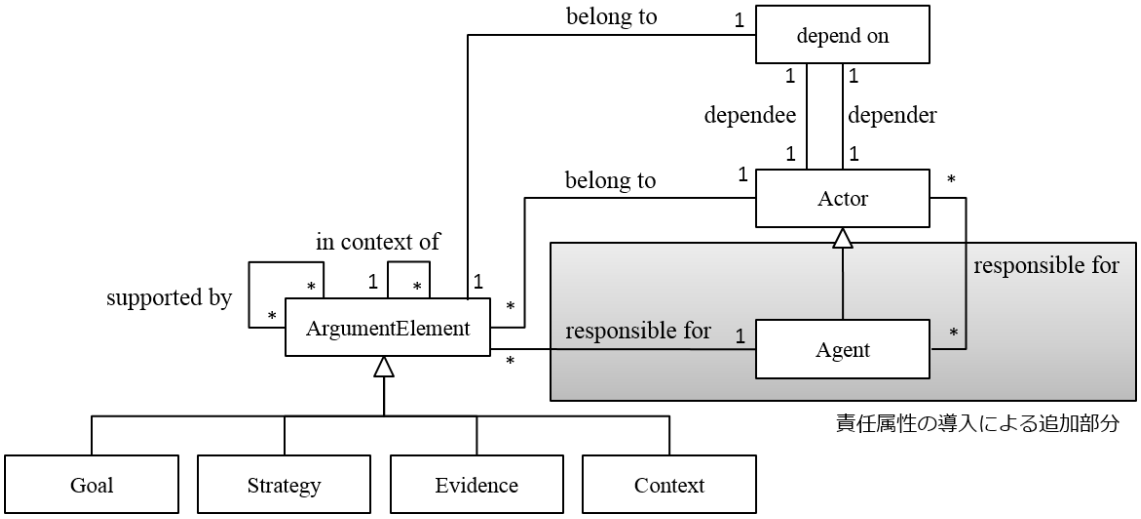

Figure 24 責任属性の導入による追加部分

4.8.5. 責任属性を導入した d* framework の適用可能なドメイン

本研究では，責任属性を導入した d* framework の例を 3 つ示した．これら 3 つの例は，責任属性を導入した d* framework が，広範囲のドメインに適用できることを示している．例 1 : エレベーターシステムは, 組み込み系システムに対する例である. 従って, この例は, 組込み系システムのドメインに対する適用例として考えることができる．例 2 : AP ダウンロードシステムは，ビジネスドメインに対する適用例として考えることができる．この例では，ビジネスの実施における役割が Agent として記述され，それぞれの役割（Agent）の責任範囲が示されている．また，LAN デバイス管理システムは，LAN のセキュリティを向上させることを目的としたシステムである．従って，例 3 : LAN デバイス管理システムは，セキュリティドメインに対して適用した例として考えることができる．このように，本研究で示した 3 つの例により, 責任属性を導入した d* framework を適用できるドメインが，十分に広いということが示されている．

4.8.6. 本研究の限界

本研究では, 責任属性を導入した d* framework の作成プロセスについては, 示していない. 従って，どのように議論を進めて，責任属性を導入した d* framework を作成するのかを，明らかにしていないという限界がある．今後，責任属性を導入した d* framework の作成プロセスについて検討し，定義する必要がある．また，本研究で提案した責任属性を導入した d* framework の有効性の検証が実施できていない点も，本研究の限界である．

4.9. まとめ

本研究では，d* framework に対する責任属性の導入について提案した．また，責任属性を d* framework 内で記述するための記法についても提案した．従来のアシュアランスケース

では，システムのディペンダビリティの保証に関する議論の記述に，Goal，Strategy，Context，Evidence 等の要素が使用されていた．これらの要素は，システムの保証に関する議論そのものを記述するために使用される要素である．本研究では，そのようなアシュアランスケースの 1 つである d* framework に対して，責任属性を導入した．システムの保証に対する説明責任を明確にするためには，システムの保証に関する議論に対する責任が明確になっている必要がある．責任属性を導入した d* framework を使用することにより，システムのディペンダビリティの保証に関する議論に対する責任の所在を明確にすることができる．すなわち，説明責任が誰にあるのかを，明確にすることができる．また，責任属性を導入した d* framework を使用することにより，システムのディペンダビリティの保証に対する責任に関する議論をステークホルダ間で共有することが可能となる．

本研究では，従来の d* framework に対する新規の要素として Agent を定義した．また，新規の要素と従来から d* framework において定義されていた要素の間の関係として，responsible for 関係を定義した．d* framework の中で，Agent は，責任を持つ主体として，人や組織を示すために記述される．また，d* framework の中に記述されている他の要素と responsible for 関係を持つ．すなわち，d* framework の中で実施されている議論に対して誰が責任を持つのかを示すために，Agent と responsible for 関係が使用される．さらに，本研究では，責任属性を導入した 3 つの d* framework の例を示した．3 つの例は，それぞれ異なるドメインにおける d* framework の例である．この 3 つの例により，本研究において提案した責任属性を導入した d* framework が，広範囲のドメインに使用可能なことが示されている．

今後，現実のシステムに対して，本研究の提案手法である責任属性を導入した d* framework を作成することにより，現実的な状況における手法の有効性の検証を実施する必要がある．また，責任属性を導入した d* framework の作成プロセスの検討や，責任属性を導入した d* framework を使った議論の確認方法についての検討も今後の課題となる．

5. コラボレーション図を使った d* framework の作成

本研究では，コラボレーション図[4]を使った d* framewrok の作成に関する研究を実施した[107][112]．コラボレーション図は，ソフトウェア工学における既存の設計手法で作成されるモデルの 1 種であり，UML[104]において定義されている．本研究を実施することで，従来のアシュアランスケースと既存の設計手法との関係が不明確であるという問題にも対処する．

5.1. はじめに

一般的なアシュアランスケースは，ステークホルダが対象システムのディペンダビリティの保証に関する議論を実施し，その議論の内容を記述することで作成される．しかし，この方法は時間がかかる．すなわち，コストの面からみて効率が悪いと考えられる．そこで，d* framework を作成する際，対象システムに関する既存の情報を使用することで，d* framework の作成にかかるコストを削減することが考えられる．本研究では，対象システムに関する既存の情報として，コラボレーション図を使用した d* framework の作成手法について述べる．また，本研究を実施することで，既存の設計手法の 1 つであるコラボレーション図と d* framework の関係を明らかにする．

まず，本研究において提案するコラボレーション図を使った d* framework の作成手順について述べる．その後，提案した作成手順を使用した d* framework の作成に関するケーススタディについて述べる．

5.2. コラボレーション図を使用した d* framework の作成手順

本節では，本研究において提案するコラボレーション図を使用した d* framework の作成手順について述べる．提案する d* framework の作成手順は，“ステップ 1：Actor の記述”，“ステップ 2：Actor 間のディペンダビリティに関する議論の記述”，“ステップ 3：Actor 内のディペンダビリティに関する議論の記述”の，3 つのステップから構成されている．下記に 3 つのステップについて述べる．

- ステップ 1：Actor の記述
  このステップでは，d* framework の中に Actor を記述する．Actor の記述には，コラボレーション図に記述されているオブジェクトが使用される．すなわち，コラボレーション図に記述されているオブジェクトと同じ名前を持つ Actor を，d* framework の中に記述する．

- ステップ 2：Actor 間のディペンダビリティの保証に関する議論の記述
  このステップでは，Actor 間のディペンダビリティの保証に関する議論を記述する．Actor 間のディペンダビリティの保証に関する議論の記述には，コラボレーション図に

記述されているオブジェクト間のメッセージ情報が使用される．2 つの Actor 間のディペンダビリティの保証に関する議論を記述する場合，コラボレーション図において，それら 2 つの Actor に対応するオブジェクト間のメッセージ情報が，議論の記述に使用される．ここでは，下記の 2 つのタイプの議論が記述される．

- 議論タイプ A
  コラボレーション図に記述されているオブジェクト間のメッセージ情報は，正確に転送される必要がある．そのため，d* framework の Actor 間に，メッセージが正確に転送されることを保証するための議論が必要となる．すなわち，コラボレーション図のオブジェクト間にメッセージ情報の転送がある場合，そのメッセージ情報が正確に転送されることを保証する議論が，d* framework の該当する Actor 間に記述される．

- 議論タイプ B
  コラボレーション図に定義されているオブジェクト間のメッセージ情報が転送される順序は，正確である必要がある．そのため，d* framework の Actor 間のメッセージ情報が，正確な順序で転送されることを保証するための議論が必要となる．すなわち，コラボレーション図のオブジェクト間にメッセージ情報の転送が複数ある場合，それらのメッセージ情報の転送順序を保証する議論が，d* framework の該当する Actor 間に記述される．

➢ ステップ 3：Actor 内のディペンダビリティの保証に関する議論の記述

このステップでは，Actor 内のディペンダビリティの保証に関する議論を記述する．Actor 内のディペンダビリティの保証に関する議論の記述には，コラボレーション図のオブジェクト間のメッセージ情報が使用される．Actor 内のディペンダビリティの保証に関する議論を記述する場合，コラボレーション図においてその Actor に対応するオブジェクトが送信するメッセージ情報が議論の記述に使用される．ここでは，下記の 2 つのタイプの議論が記述される．

- 議論タイプ C
  コラボレーション図に記述されているオブジェクトが転送するメッセージの順序は，正確に管理されている必要がある．そのため，d* framework のメッセージを転送する Actor 内で，転送するメッセージの順序が正確であることを保証するための議論が必要となる．すなわち，コラボレーション図のオブジェクトに，送信するメッセージ情報を複数持つものがある場合，それらのメッセージの順序を保証する議論が d* framework の該当する Actor 内に記述される．

- 議論タイプ D
  コラボレーション図に記述されているオブジェクト間のメッセージに付与されているパラメータは，正確に管理されている必要がある．そのため，d* framework のメッセージを転送する Actor 内で，転送するメッセージに付与されているパラメータが正確に管理されていることを保証するための議論が必要となる．すなわち，コラボレーション図のオブジェクトが送信するメッセージにパラメータが付与されている場合，そのパラメータを保証する議論が d* framework の該当する Actor 内に記述される．

5.3. ケーススタディ

ケーススタディでは，提案した d* framework の作成手順を使用して，コラボレーション図から d* framework が作成できることを確認する．

5.3.1. 対象のコラボレーション図

ケーススタディで使用するコラボレーション図が対象とするシステムは，AP ダウンロードシステムである．AP ダウンロードシステムとは，ユーザが，サーバからアプリケーションを，ユーザのメモリカードにダウンロードするためのシステムである．このシステムを使用することで，ユーザはダウンロードしたいアプリケーションを選択し，そのアプリケーションをシステムからユーザのメモリカードにダウンロードすることができる．ケーススタディでは，このシステムを対象として作成されたコラボレーション図を使用し，d* framework を作成する．ケーススタディで使用する AP ダウンロードシステムのコラボレーション図を Figure 25 に示す．このコラボレーション図では，Issuer of cards，Service provider，AP download window，AP download management，User，Memory of Card の 6 個のオブジェクトが定義されている．また，それらオブジェクト間に 13 個のメッセージ情報が定義されている．メッセージ情報に付けられた番号は，メッセージ情報の送信順序を示している．また，メッセージ情報には，メッセージ情報を送信する際のパラメータが定義されているものがある．そのようなメッセージ情報には，メッセージ情報名の後の括弧内にパラメータ名が記述されている．

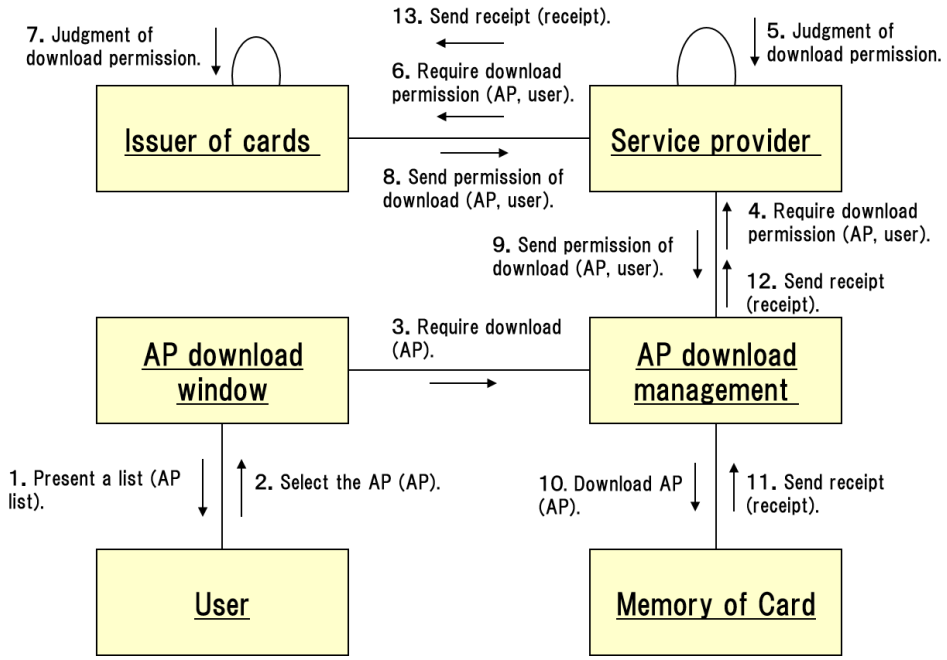

Figure 25 AP ダウンロードシステムのコラボレーション図

5.3.2. d* framework の作成

ケーススタディでは，提案した作成手順を用いて d* framework を作成する．ここでは，提案した作成手順のステップ毎に，d* framework の作成結果について述べる．尚，本研究では，d* framework の記述に D-Case Editor を使用している．

- ステップ 1

  このステップでは，コラボレーション図を使って，Actor を記述した．d* framework の Actor は，コラボレーション図のオブジェクトの名前を使って記述される．ここでは，Issuer of cards，Service provider，AP download window，AP download management，User，Memory of Card の 6 個の Actor が記述された．

- ステップ 2

  このステップでは，“Actor 間の依存関係を示す d* framework”が作成された．作成された d* framework を Figure 26 に示す．作成された d* framework には，ディペンダビリティの保証に関する 15 個の Goal が，Actor 間に記述された．コラボレーション図におけるオブジェクトから送信されるメッセージ情報が 1 つの場合は，メッセージ情報が正確に送信されることを保証するための Goal（議論タイプ A）のみが該当する Actor 間に記述されている．オブジェクトからオブジェクトへのメッセージ情報が複数ある場合は，メッセージ情報が正確に送信されることを保証するための Goal（議論タイプ A）と，メッセージ情報の送信順序が正確であることを保証するための Goal（議

論タイプ B）が該当する Actor 間に記述されている．

ステップ 2 において記述された具体的な Goal の例を次に示す．コラボレーション図では，Issuer of Card は Service provider に対して“Send permission of download (AP, user)”というメッセージ情報を送信している．そのため，d* framework には，Service provider から Issuer of card への依存関係として，メッセージが正確に送信されることを保証するための Goal（議論タイプ A）が記述されている．実際に記述されている Goal は“Permission of download is sent correctly”である．

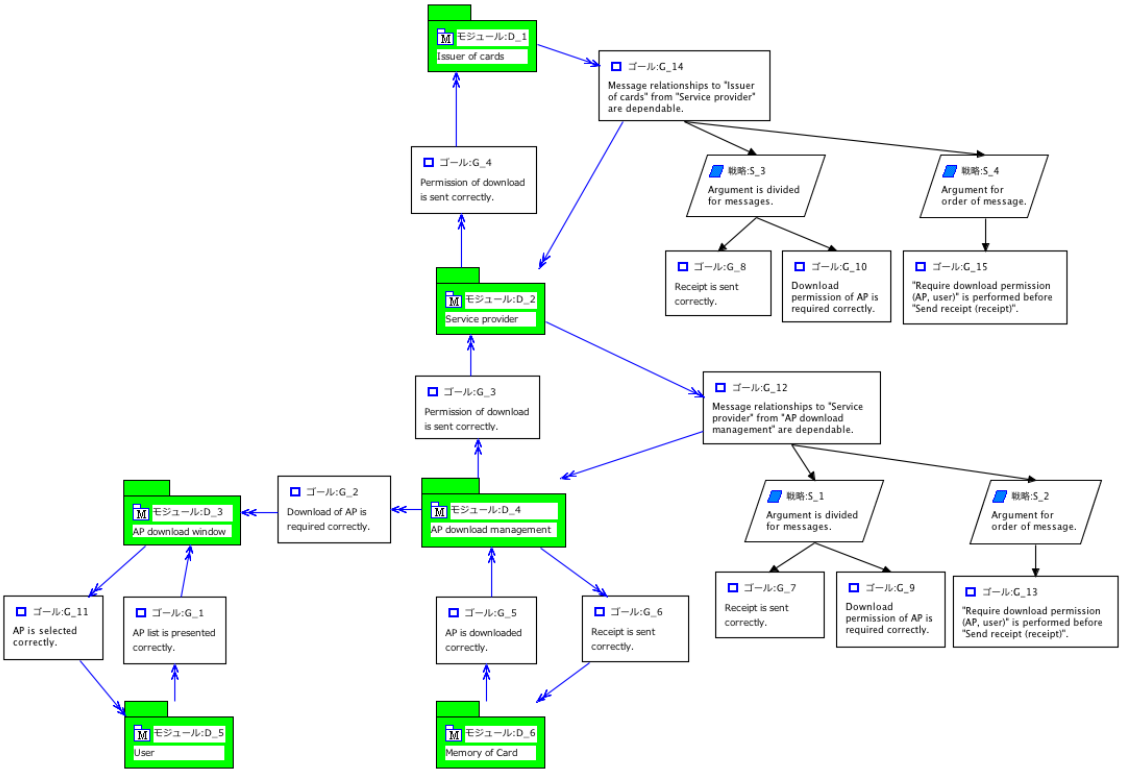

Figure 26 ステップ 2 で作成された Actor 間の依存関係を示す d* framework

- ステップ 3

このステップでは，“Actor 自体がディペンダブルであることを示す d* framework”が作成された．このステップで作成された d* framework は，“Actor 間の依存関係を示す d* framework（Figure 26）”において記述されている個々の Actor に対して，それぞれ個別に作成されている．すなわち，Actor 間の依存関係を示す d* framework において記述されている Issuer of cards，Service provider，AP download window，AP download management，User，Memory of Card の 6 個の Actor に対して，6 個の d* framework が作成された．それぞれの d* framework では，Actor 自体がディペンダブルであることを保証するための議論が記述されている．Figure 27 から Figure 32 に，作成された d* framework を示す．これら d* framework には，Actor が送信するメッ

セージ情報の送信順序が正確であることを保証するための Goal（議論タイプ C）と，Actor が送信するメッセージ情報に付与されたパラメータが正確に管理されることを保証するための Goal（議論タイプ D）の，2 つのタイプの Goal が記述されている．ステップ 3 において記述された具体的な Goal の例を次に示す．コラボレーション図において，Issuer of card は，Service provider に対して“Send permission of download (AP, user)”というパラメータが付与されたメッセージ情報を送信している．従って，Issuer of card の d* framework には，“Parameters of “Issuer of card” is dependability managed”という Goal が記述されている（Figure 27）．これは，Actor が送信するメッセージのパラメータを保証するための Goal である．さらに，この Goal は，“Information of AP that is permitted to download is managed”と“Information of user who is permitted to download is managed”の 2 つの下位の Goal に分割されている．これは，元になったコラボレーション図におけるメッセージ情報が，AP と user の 2 つのパラメータを持っているためである．すなわち，2 つのパラメータに対して議論分解が実施され，それに対応して 2 つの Goal が記述されている．

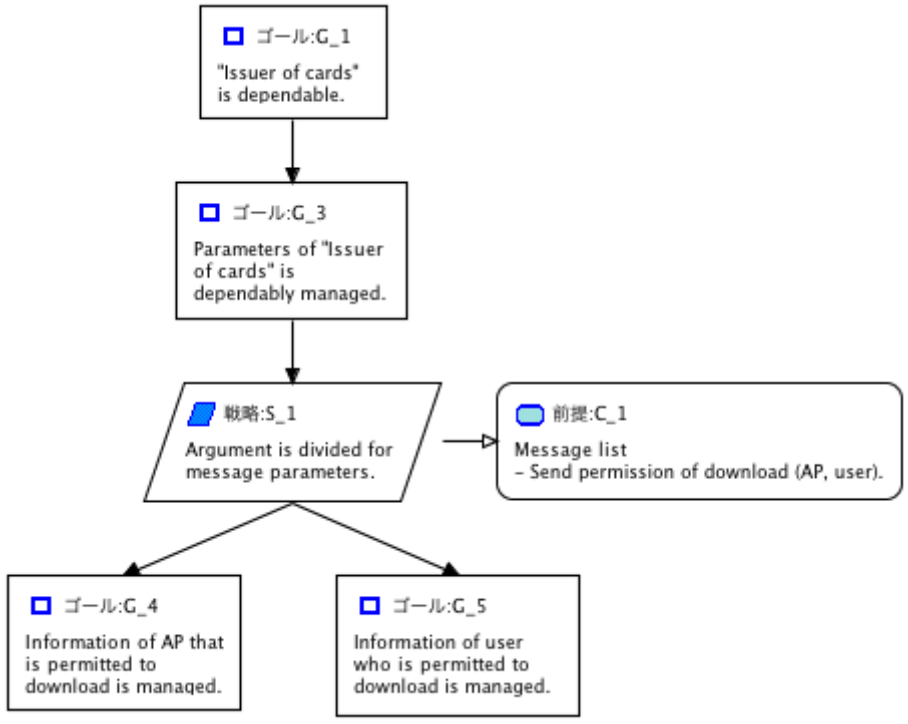

Figure 27 ステップ 3 で作成された“Actor（Issuer of cards）自体がディペンダブルであることを示す d* framework”

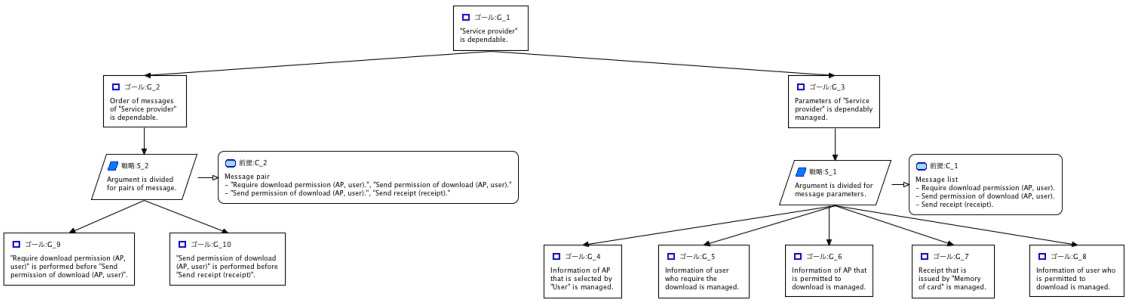

Figure 28 ステップ 3 で作成された“Actor（Service provider）自体がディペンダブルであることを示す d* framework”

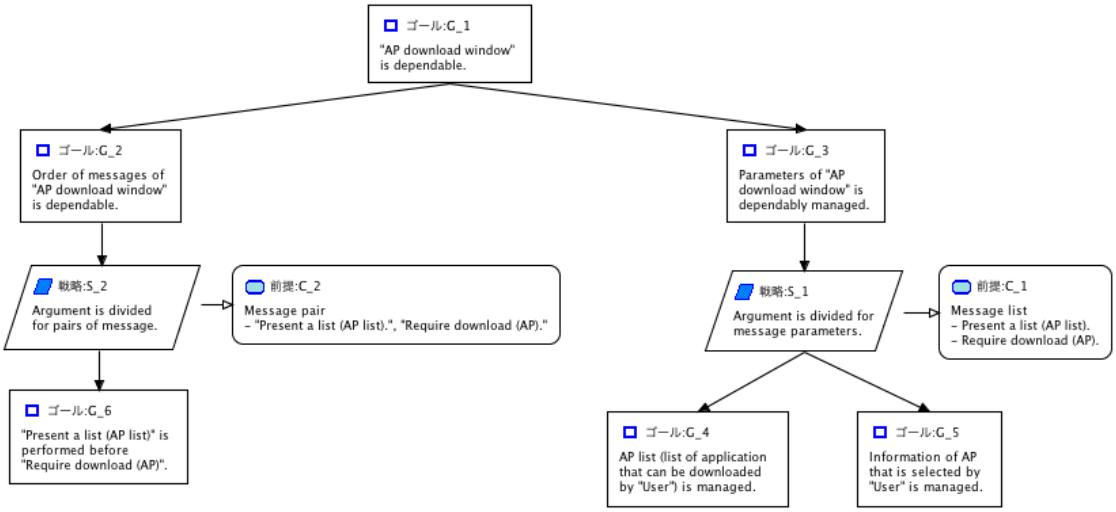

Figure 29 ステップ 3 で作成された“Actor（AP download window）自体がディペンダブルであることを示す d* framework”

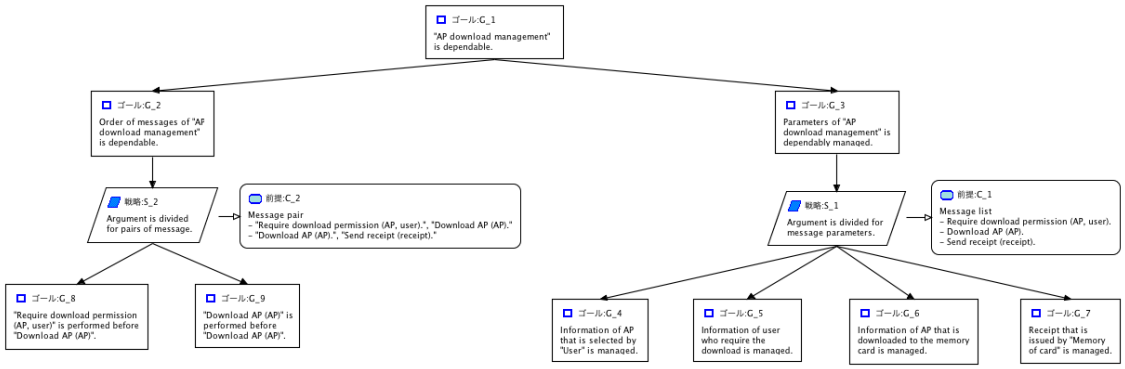

Figure 30 ステップ 3 で作成された“Actor（AP download management）自体がディペンダブルであることを示す d* framework”

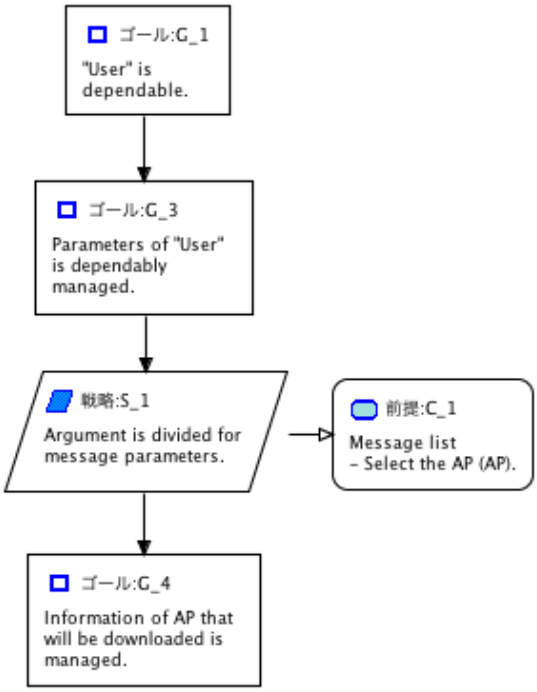

Figure 31 ステップ 3 で作成された“Actor（User）自体がディペンダブルであることを示す d* framework”

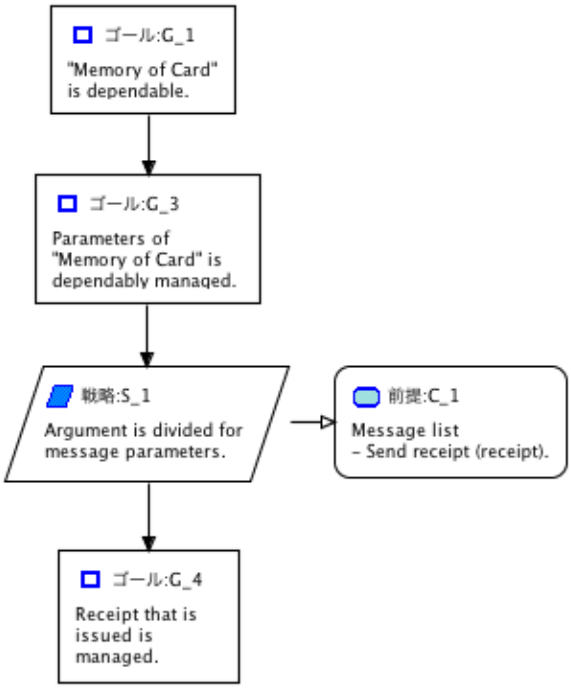

Figure 32 ステップ 3 で作成された“Actor（Memory of Card）自体がディペンダブルであることを示す d* framework”

5.3.3. ケーススタディの結果

ケーススタディの結果作成された d* framework を，Figure 26 から Figure 32 に示した．この節では，d* framework の作成に使用したコラボレーション図と，作成した d* framework に含まれる要素の数を示す．

まず，ケーススタディにおいて，入力情報として使用したコラボレーション図 (Figure 25) に含まれている要素の数を Table 8 に示す．コラボレーション図には，システムを構成するオブジェクトが 6 個と，それらオブジェクト間に 13 個のメッセージ情報が記述されている．

Table 8 コラボレーション図に含まれる要素の数

Object	Message relationship
6	13

次に，作成した d* framework に含まれている要素の数を Table 9 に示す．この表の列には，d* framework の要素の種類が示されている．行には，作成した d* framework の名前が示されている．Inter actor d* framework は，ステップ 2 で作成された“Actor 間の依存関係を示す d* framework”を示している．それ以外の d* framework は，ステップ 3 で作成された d* framework であり，それぞれ Actor の名前が d* framework の名前として使用されている．ステップ 3 で作成された d* framework は，要素として Actor を含まないので，それら d* framework の Actor 列には，“-”を記述している．

本研究におけるケーススタディでは，d* framework の中に全部で 78 個の要素が記述された．これらは，コラボレーション図の 6 個のオブジェクトと，13 個のメッセージ情報から記述されたものである．

Table 9 作成された d* framework に含まれる要素の数

D* framework	Actor	Goal	Strategy	Context	Total
Inter actor d* framework	6	15	4	0	25
Issuer of cards	-	4	1	1	6
Service provider	-	10	2	2	14
AP download window	-	6	2	2	10
AP download management	-	9	2	2	13
User	-	3	1	1	5
Memory of Card	-	3	1	1	5
Total	6	50	13	9	78

5.4. 考察

5.4.1. d* framework の作成におけるコラボレーション図の有用性

d* framework を作成する際，対象システムに関する既存の情報を使用することにより，d* framework の作成にかかる時間的なコストを，削減することが考えられる．本研究では，対象システムに関する既存の情報として，コラボレーション図を使用した．
コラボレーション図を使用した d* framework の作成には，下記に示す両者の間の 2 つの関係が利用できる．

- 関係 1 : d* framework の Actor は，コラボレーション図のオブジェクトの情報を使って記述することができる．

- 関係 2 : d* framework におけるシステムのディペンダビリティを保証するための議論は，コラボレーション図のメッセージ情報を使って記述することができる．

本研究で実施したケーススタディでは，ステップ 1 において，コラボレーション図の 6 個のオブジェクトから，d* framework に 6 個の Actor が直接記述された．この記述には，上記の関係 1 が使われている．さらに，ステップ 2 において，コラボレーション図のメッセージ情報から，システムのディペンダビリティの保証に関する 15 個の Goal が，d* framework の Actor 間に記述された．また，ステップ 3 では，コラボレーション図のメッセージ情報から，システムのディペンダビリティの保証に関する 35 個の Goal が，d* framework の Actor 内に記述された．ステップ 2 とステップ 3 の記述では，コラボレーション図に記述されている 13 個のメッセージ情報が使用されている．これらの記述には，上記の関係 2 が使われている．このように，本研究を実施したことにより，コラボレーション図を d* framework の作成に役立てられることが確かめられた. すなわち, d* framework を作成する際，コラボレーション図が有用であることが示された.

5.4.2. d* framework に記述される Goal の種類

提案した手法では，d* framework の Goal は，コラボレーション図のメッセージ情報から検討され記述される．ここで記述される Goal には，保証の対象毎に Table 10 に示される 3 つの種類（タイプ）が存在する．

Table 10 メッセージから定義される Goal の種類（タイプ）

タイプ 1	メッセージ情報の保証
タイプ 2	メッセージ情報の順序の保証
タイプ 3	メッセージ情報のパラメータの保証

これら Goal の種類について，以下に述べる．

5.4.2.1. メッセージ情報の保証（タイプ 1）

メッセージ情報は，コラボレーション図のオブジェクト間に記述されており，オブジェクト間にメッセージ情報の送受信があることを示している．オブジェクト間にメッセージ情報が定義されている場合，d* framework における議論では，メッセージが正確に送信されることが，保証される必要がある．そのため，コラボレーション図のメッセージ情報に対して，メッセージが正確に送信されることを保証する Goal が，d* framework に記述される．これは，タイプ 1 の Goal である．

ケーススタディでは, これらは, 議論タイプ A における Goal として, d* framework の Actor 間に記述されている．例えば，Actor である User と AP download window の間に “AP list is presented correctly”という Goal が, タイプ 1 の Goal として記述されている (Figure 26)．これは，コラボレーション図の“2. Select the AP”というメッセージ情報を使用して記述された Goal である．

5.4.2.2. メッセージ情報の順序の保証（タイプ 2）

コラボレーション図に記述されているメッセージ情報には，メッセージを送信する順序が定義されている．その場合，d* framework における議論では，メッセージが送信される順序が正確であることを保証する必要がある．そのため，コラボレーション図のメッセージ情報の順序に対して，その順序が正確であることを保証する Goal が d* framework に記述される．このメッセージの送信順序を保証するための Goal は，該当するメッセージ関係を持つ Actor について，Actor 間と Actor 内のどちらにも記述する必要がある．これは，タイプ 2 の Goal である．

ケーススタディでは，これらは，議論タイプ B における Goal として Actor 間に，また，議論タイプ C における Goal として Actor 内に，それぞれ定義されている．例えば，Actor である Issuer of card と Service provider の間に, “‘Require download permission (AP, user)’ is performed before ‘Send receipt (receipt)’”という Goal が定義されている (Figure 26)．これは，Actor 間に定義された Goal の例である．また，Actor である Service provider 内に，“‘Require download permission (AP, user)’ is performed before ‘Send permission of download (AP, user)’”という Goal が定義されている（Figure 28)．これは，Actor 内

に定義された Goal の例である．

5.4.2.3. メッセージ情報のパラメータの保証（タイプ 3）

コラボレーション図で定義されているメッセージ情報には，パラメータが付与されているものがある．その場合，d* framework における議論では，メッセージ情報に付与されているパラメータが，正確に管理されていることを保証する必要がある．そのため，メッセージ情報のパラメータが正確に管理されていることを保証する Goal が，d* framework に記述される．これは，タイプ 3 の Goal である．

ケーススタディでは，これらは，議論タイプ D の Goal として d* framework の Actor 内に記述されている．メッセージ関係に付与されているパラメータは，Actor の入力および出力情報である．それゆえ，これらのパラメータは，該当する Actor により正確に管理される必要がある．例えば，Actor である Issuer of card 内には，“Information of AP that is permitted to download is managed”という Goal が，記述されている（Figure 27）．ここで，AP は，Issur of card が送信するメッセージに付与されているパラメータである．

5.4.3. 本研究の限界

本研究では，ケーススタディを実施した．しかし，実施したケーススタディは規模が小さいものであった．また，ケーススタディにおける d* framework の作成は，1 人の研究者によって実施されたものである．現実的には，d* framework の作成は，システムの関係者であるステークホルダによって実施されると考えられる．また，作成される d* framework の規模も，大きなものになることが想定される．従って，本研究におけるケーススタディは，必ずしも現実的な状況を反映していないという限界がある．また，作成された d* framework 自体の正当性が，確かめられていないという限界も存在する．

5.5. まとめ

本研究では，コラボレーション図を使用した，3 つのステップから構成される d* framework の作成手順を提案した．さらに，提案した手順を使用して，コラボレーション図から d* framework を実際に作成するケーススタディを実施した．ケーススタディでは，ユーザがアプリケーションをダウンロードするためのシステムである AP ダウンロードシステムを対象として作成されたコラボレーション図を使用して，d* framework を作成した．その結果，19 個の要素を含むコラボレーション図から，78 個の要素を含む d* framework が作成された．モデルに含まれる要素数からは，提案した手順を使用することで，リッチな内容の d* framework を作成することができたと言える．

また，コラボレーション図から d* framework を作成する手順を検討することにより，ソフトウェア工学におけるモデルの 1 つであるコラボレーション図と，d* framework の関係を示すことができた．

6. d* framework の有効性の検証

本研究では，実際の現場における d* framework の有効性を検証するため，IT システムを開発する企業である SIer（System Integrator）における d* framework の作成実験を実施した[108]．本章では，本研究において実施した，適用実験に基づく d* framework の有効性の検証について述べる．

6.1. はじめに

従来，組み込み系システムに対しては，アシュアランスケースが使用されてきた．この理由として，自動車に搭載されるシステムのように，組込み系システムでは，システムの不具合が重大な損害に直結しやすいことに加え，不具合が発覚してからの修正が困難であることなどが考えられる．しかしながら，エンタープライズ系サービスに対するアシュアランスケースの適用事例は，ほとんど存在しなかった．エンタープライズ系のサービスやシステムにも社会的に重要なインフラは存在し，その障害が重大な損害を引き起こす可能性があることは，組み込み系システムと同様である．そのため，エンタープライズ系のサービスにおいても，システムの保証は重要な課題となる．組込み系システムとエンタープライズ系システムにおけるアシュアランスケースの適用に，本質的な違いは無いと考えられる．

本研究では，エンタープライズ系サービスを対象として，アシュアランスケースの 1 種である d* framework を作成し確認する．本研究では，対象サービスの直接のステークホルダではない第三者が主導して，d* framework を作成する．第三者による作成には，実際の現場に対する d* framework 適用の導入障壁を下げる狙いがある．さらに，本研究では，第三者による，d* framework を使用したサービスのディペンダビリティに対する保証の確認を実施する．

6.2. d* framework の作成

本研究では，定義した作成手順を使用して，d* framework を作成する．また，サービスに直接関係していない第三者が主導して，d* framework を作成する．本研究において定義した作成手順を使用するには，“作成を主導する第三者が，d* framework の作成に精通している”，“ディペンダビリティについて記載された対象サービスの文書が存在し，閲覧することができる”，“作成された d* framework を修正するために，ステークホルダの協力が得られる”といった前提条件が存在する．

d* framework は，要件定義から運用に至るシステム開発工程の全般で使用することができる．本研究では，運用中のサービスに対して d* framework の作成を実施する．以下では，本研究において実施する d* framework の作成について述べる．

6.2.1. 作成対象サービス

本研究では，現実のサービスに対して d* framework を作成する．この節では，本研究における d* framework の作成対象について述べる．本研究における d* framework の作成対象は，SIer において使用されている IT システムを使った社内向けサービスである．対象とした SIer では，IT システム開発プロジェクトを管理するために，このサービスが利用されている．ユーザは，このサービスを利用して，IT システム開発プロジェクトの進捗管理，品質管理等を実施できる．対象とした SIer では，このサービスにより，常時 500 前後のプロジェクトが管理されている．また，このサービスは，24 時間 365 日提供されている．

6.2.2. 作業者

本研究では，サービスに直接関わっていない第三者が主導するかたちで，d* framework の作成作業を実施する．まず，第三者が d* framework を作成する．その後，作成された d* framework を使用して，サービスに直接関わっているステークホルダとディスカッションを実施し, 得られた指摘事項に基づいて第三者が d* framework を修正する. d* framework に関する知識が乏しい状態のステークホルダが, d* framework を作成するのは困難である．本研究では，d* framework に関する知識を十分に有した第三者が，d* framework の作成を主導することで，この困難性を克服している．

6.2.3. 作成手順

本研究では，第三者が主導して d* framework を作成する．第三者は，対象となるサービスに関する情報を持っていない．そこで，d* framework を作成するにあたり，複数のステップからなる手順を定義した．以下に各ステップにおける実施内容を示す.

1. 事前準備

1.1. 資料確認

本研究における d* framework の作成では，対象となるサービスに関する資料を利用する．このステップでは，それら資料の概要を把握する．

1.2. 資料選定

“1.1. 資料確認”により概要を把握した資料から，実際に d* framework の作成に使用する資料を選定する．

1.3. 情報記載箇所の識別

“1.2. 資料選定”により選定された資料から，サービスのディペンダビリティに関係する記述と, 資料内の記載箇所を識別する. さらに, 識別された記述を, Goal, Strategy,

Context，Evidence の 4 つのカテゴリに分類する．

2.　d* framework の記述

2.1.　Actor の定義

サービスの全体像が記述されている資料等を利用して Actor を識別し，d* framework に記述する．

2.2.　Actor 間の議論記述

“1.3. 情報記載箇所の識別”により識別された d* framework に関係する記述を利用して，Actor 間で必要となるディペンダビリティの保証に関する議論を検討し，d* framework に記述する．

2.3.　Actor 内の議論記述

“1.3. 情報記載箇所の識別”により識別された d* framework に関係する記述を利用して，Actor 内で必要となるディペンダビリティの保証に関する議論を検討し，d* framework に記述する．

3.　d* framework の修正

3.1.　ディスカッション

“2. d* framework の記述”により作成された d* framework を使用して，サービスに直接関わっているステークホルダとディスカッションを実施し，記述の誤り，記述の漏れ等に関して確認する．

3.2.　記述修正

“3.1. ディスカッション”の結果に基づいて，d* framework を修正する．

6.2.4.　作成に使用した資料

本研究では，対象となるサービスに関する資料を使用して d* framework を作成する．上記 6.2.3 に示した作成手順“1.1. 資料確認”，“1.2. 資料選定”を経て，対象サービスに関する 172 個の資料から 11 個の資料が，d* framework の作成に使用する資料として選定された．選定された資料の一覧を Table 11 に示す. 表では，資料のページ数についても示している．

Table 11 d* framework の作成に使用した資料一覧

項番	資料名	ページ数
1	バックアップ管理対象一覧	3
2	バックアップ・リストア設計_詳細設計書	59
3	性能要件	12
4	信頼性要件	4
5	システム運用管理要件	11
6	運用・保守性検討書	3
7	可用性検討書	6
8	性能・拡張性検討書	3
9	運用・保守性要件一覧	2
10	可用性要件一覧	2
11	性能・拡張性要件一覧	2

6.2.5. 作成結果

本研究では，6.2.3 の作成手順に従い，対象となるサービスの，“Actor が他の Actor に対して責任を遂行することを示す d* framework”と“Actor 自体がディペンダブルであることを示す d* framework”の 2 つの d* framework を作成した．ここでは，作成した各 d* framework に対して“サービス”，“IT システム”という名称を使用する．

6.2.5.1. 作成した d* framework のノード個数

本研究において作成された 2 つの d* framework は，複数の議論の要素，すなわちノードから構成されている．作成された d* framework のノードの個数を，2 つの d* framework 及びノードの種類毎に整理したものを Table 12 に示す. 表では，上記作成手順における“2. d* framework の記述”の結果と“3. d* framework の修正”の結果を，それぞれ個数列の作成後列と修正後列に示している．また，行には作成した d* framework の名称と d* framework における要素の種類が示されている．表から，本研究では，第三者のみによる d* framework の作成において，ノード数 42（サービス）とノード数 54（IT システム）の 2 つの d* framework が作成されたことがわかる．また，ステークホルダとディスカッションを実施し d* framework の記述を修正した結果，ノード数が 1.375 倍に増加し，最終的にノード数 60（サービス）と，ノード数 72（IT システム）の 2 つの d* framework が作成されたことがわかる．

Table 12 作成された d* framework の要素の個数

d* framework	要素数	個数	
		作成後	修正後
サービス	Actor	4	4
	Goal	29	37
	Strategy	3	6
	Evidence	6	13
	合計	42	60
IT システム	Goal	44	56
	Strategy	7	10
	Context	3	3
	Evidence	0	3
	合計	54	72

6.2.5.2. Actor が他の Actor に対して責任を遂行することを示す d* framework（サービス）

この節では，本研究において作成した“Actor が他の Actor に対して責任を遂行することを示す d* framework（サービス）”について述べる．この d* framework には，利用者，IT システム，システム開発者，システム運用者の 4 つの Actor と，Actor 間の依存関係に関する議論が記述されている. Actor 間の依存関係に関する議論として記述された内容を, Table 13 に示す.

Table 13 Actor 間に記述された Actor 間の依存関係に関する議論内容

依存元	依存先	議論の内容
システム運用者	利用者	サービス範囲に関する議論
利用者	IT システム	サービスの信頼性責任に関する議論
利用者	IT システム	ユーザビリティ責任に関する議論
IT システム	システム運用者	運用責任に関する議論
システム運用者	IT システム	システム耐性責任
IT システム	システム開発者	開発責任に関する議論
システム運用者	システム開発者	運用設計責任
システム開発者	サービス運用者	発注責任

作成した d* framework の一部を Figure 33 に示す．ここでは，利用者から IT システムに依存する関係として実施されている“サービス信頼性責任に関する議論”が記述されてい

る．利用者から IT システムに依存する関係として“信頼のおけるサービスが提供されている”という Goal が記述されている．さらに，記述された Goal が，信頼性の下位特性により分解されることで議論が進められている．これより下位では，サービスの成熟性及びサービスの回復性についての議論が記述されている．このように，Actor が他の Actor に対して責任を遂行することを示す d* framework には，Actor 間の依存関係に関する議論が構造的に記述される．

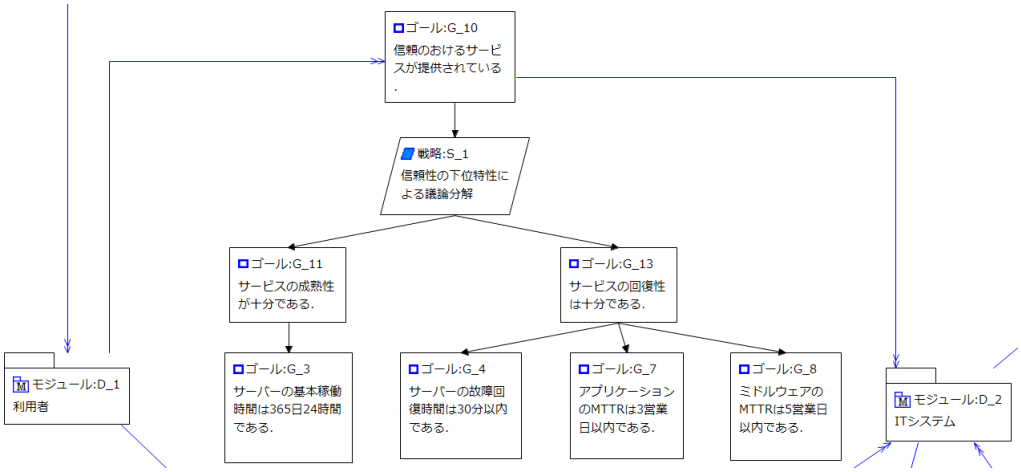

Figure 33 作成された d* framework からの抜粋（利用者と IT システム間のサービス信頼性責任に関する依存関係における議論）

6.2.5.3. Actor 自体がディペンダブルであることを示す d* framework（IT システム）

この節では，本研究において作成した，“Actor 自体がディペンダブルであることを示す d* framework（IT システム）”について述べる．この d* framework の対象は，“Actor が他の Actor に対して責任を遂行することを示す d* framework（サービス）”において，Actor として記述されている IT システムである．IT システムは，“Actor が他の Actor に対して責任を遂行することを示す d* framework（サービス）”内で記述されている 4 つの Actor のうちの 1 つである．他の Actor については，d* framework を作成するための入力情報となる資料が存在しなかったため，今回は作成していない．しかし，今回は，入力情報から作成できる範囲での評価であるので，この点について差し支えはない．作成した d* framework では，可用性，信頼性，性能に関する議論が実施されている．すなわち，この d* framework では，“システムの可用性が担保されている”，“システムの信頼性が担保されている”，“システムの性能が担保されている”の 3 つの Goal を分解することで，IT システムのディペンダビリティの保証に関する議論がすすめられている．Figure 34 に，IT システムの信頼性に関する議論について記述された d* framework の一部を示す．ここでは，信頼性に関する議論として，“システムの信頼性が担保されている”という Goal が記述されている．さらに，ハードウェアとソフトウェアに分けて議論する戦略に沿って，2 つの

Goal が記述されている．

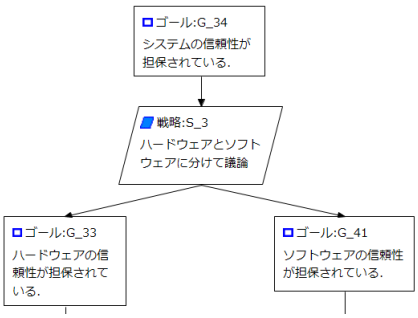

Figure 34 作成された d* framework からの抜粋（IT システムにおける信頼性の担保に関する上位の議論）

Figure 35 に，バックアップの取得に関する議論について記述された d* framework の一部を示す．ここに記述されている議論は，Figure 34 で示した，信頼性の担保に関する議論の下位で実施されている議論である．“バックアップを取得する”という Goal が記述され，対象機器毎に分解する戦略に沿って議論が進められている．また，議論を分解する戦略には対象機器のリストが Context として付与されている．

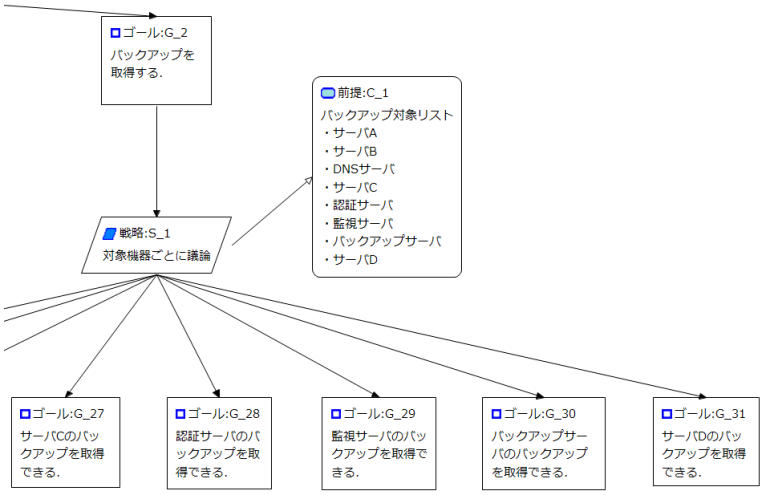

Figure 35 作成された d* framework からの抜粋 （IT システムにおけるバックアップ取得に関する議論）

6.2.6. 作業時間

本研究では，6.2.3 に示した作業のステップ毎に，作業に要した時間を計測した．作業時間を計測した結果を Table 14 に示す．本研究において，d* framework の作成作業に費やした時間は，全体で 14 時間であった．“3.1. ディスカッション”の 1 時間以外は，ステーク

ホルダ以外の第三者 1 人による作業時間である.

Table 14 d* framework 作成に要した作業時間

作業手順		作業時間
1.事前準備	1.1. 資料確認	2 時間
	1.2. 資料選定	10 分
	1.3. 情報記載箇所抽出	1 時間 50 分
2.d* framework 記述	2.1. Actor の定義	20 分
	2.2. Actor 間の議論記述	2 時間
	2.3. Actor 内の議論記述	3 時間 40 分
3.d* framework 修正	3.1. ディスカッション	1 時間
	3.2. 記述修正	3 時間

6.3. d* framework の確認

この節では，本研究において実施した d* framework の確認について述べる．本研究において作成した d* framework には，作成に使用した資料，及びステークホルダとのディスカッションでは明らかにできなかった議論の不足個所が存在することが考えられる．本研究では，d* framework の構造に着目した次の 3 つの観点を利用することにより，第三者による d* framework の議論の不足箇所を指摘する確認を実施した.

1. 「証拠の不足箇所」の確認

   証拠の不足箇所とは，d* framework において記述されている Goal を保証するための証拠が，記述されていない箇所である．そのような箇所は，証拠の記述が不十分であると考えられる．ここでは，そのような箇所を「証拠の不足箇所」として識別した．Figure 36 に「証拠の不足箇所」の例を示す．この例では，“サーバの基本稼働時間は 365 日 24 時間である”,“サーバの故障回復時間は 30 分以内である”という 2 つの Goal が, 証拠のついていない Goal として識別されている. この 2 つの Goal は, d* framework において最下層に記述されている Goal であると考えることができる．このように，証拠の不足箇所は，d* framework の最下層の Goal に対して，その Goal を保証する証拠が不足している際に識別することができる.

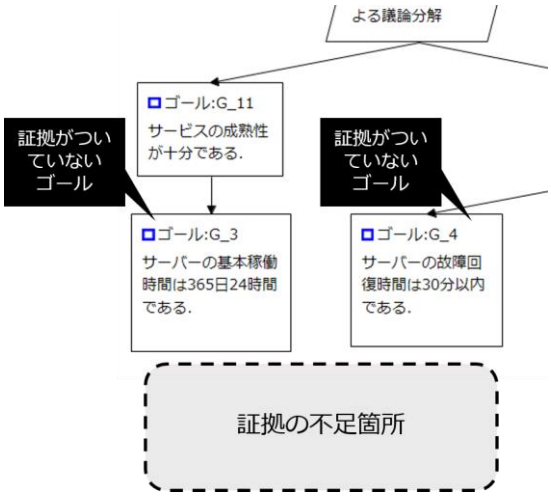

Figure 36 証拠の不足箇所の例

2. 「議論の不足箇所」の確認

議論の不足箇所とは，d* framework における議論が不十分な箇所である．すなわち，下位の Goal が検討可能であるにもかかわらず，下位の Goal に議論が展開されていない Goal が，記述されている箇所である．そのような箇所は，議論が不足していると考えられる．ここでは，そのような箇所を「議論の不足箇所」として識別した．Figure 37 に「議論の不足箇所」の例を示す．この例では，“NIC の信頼性が担保されている”が，議論展開が不十分な Goal として識別されている．この Goal は，下位の Goal に議論が展開できる Goal であると考えることができる．このように，議論の不足箇所は，d* framework 内で，下位の Goal に議論が展開できるにもかかわらず，展開されていない Goal が存在する場合に識別することができる．

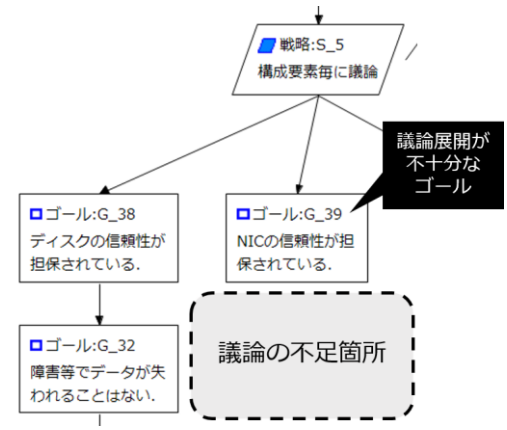

Figure 37 議論の不足箇所の例

3.　議論分解妥当性の不足箇所の確認

議論分解妥当性の不足箇所とは，d* framework を使って Goal を分解することによる議論は実施されているが，その議論分解の妥当性が不足している箇所である．d* framework における議論分解の戦略は，Strategy を使って記述される．また，Strategy で記述された議論分解の戦略に対して具体的な背景情報を与えるためには, Context が使用される. 従って, d* framework において, Strategy が記述されており具体的な背景情報が必要であるにもかかわらず，Context が付与されていない箇所を，議論分解妥当性の不足箇所として識別できる．本研究では，“対象機器毎に議論”など，d* framework が対象としているサービスについて固有の背景情報が議論分解に必要な箇所について, Context が付与されていない箇所を「議論分解妥当性の不足箇所」として識別した．Figure 38 に「議論分解妥当性の不足箇所」の例を示す．この例では，“対象機器毎に議論”が，議論分解の妥当性が不足している Strategy として識別されている．この Strategy には，“対象機器毎に議論”と記述されているので，対象機器が明確にされる必要がある．本来であれば，それが Context として記述される必要があるが，付与されていない．従って，この Strategy は，議論分解の妥当性が不足している Strategy であると識別できる．

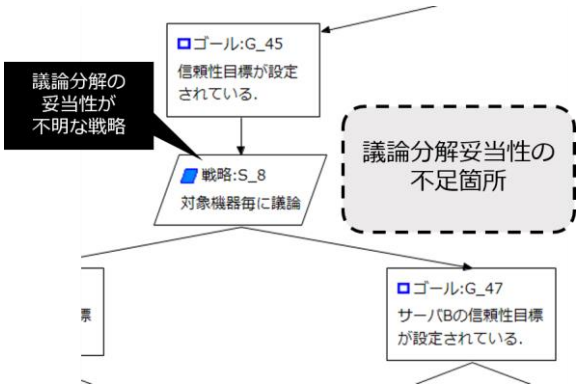

Figure 38 議論分解妥当性の不足箇所の例

6.3.1.　d* framework の確認結果

本研究では，ケーススタディで作成した d* framework における議論の不足箇所について，上記 3 つの観点を使った第三者による確認を実施した．確認による議論の不足箇所の指摘結果を，Table 15 に示す．表では，“Actor 間の責任関係を示す d* framework（サービス）”と，“Actor がディペンダブルであることを示す d* framework（IT システム）”に分けて不足箇所の個数を示している．また，d* framework に含まれる要素の個数に対する不足箇所の個数の割合も示している．表より，“Actor 間の責任関係を示す d* framework”に 13 個（22%），“Actor がディペンダブルであることを示す d* framework”に 32 個（44%）の合計 45 個（34%）の議論の不足箇所を指摘できたことがわかる．

Table 15 d* framework の確認結果

確認の観点	サービス		IT システム		合計	
	個数	割合	個数	割合	個数	割合
1.証拠の不足箇所	8	13%	25	35%	33	25%
2.議論の不足箇所	4	7%	4	6%	8	6%
3.議論分解妥当性の不足箇所	1	2%	3	4%	4	3%
合計	13	22%	32	44%	45	34%

本研究における d* framework の確認において指摘された議論の不足箇所は, 比較的単純なものである. しかし, d* framework の作成に使用された資料からは漏れていた議論であり, 今回のような, サービスの保証に関する議論の体系的な整理がなければ, 見落とされていたものである. d* framework を作成し, 体系的に議論を整理することにより, 比較的短い時間でこれらの不備が発見できたことから, 今回の d* framework の作成が有益であったと考えることができる.

6.4. 考察

6.4.1. ステークホルダによる評価

本研究では, d* framework の確認後, 実際にサービスに関わっているステークホルダから, 本研究の取り組み及び d* framework に対するコメントを収集した. 得られたコメントを Table 16 に示す. これらのコメントから, 以下の知見が得られた. d* framework に記述されている内容は, 容易に理解することができる. また, 十分な知識があれば, ステークホルダ自身で d* framework を作成することも可能である (C01, C02, C03). 社会インフラ等のミッション・クリティカルなシステムが関係するサービスを対象とした方が, そうではないサービスを対象とするよりも, d* framework を有効に活用できると考えられる. また, そのようなサービスであれば, d* framework の作成コストも許容できると考えられる. ただし, 監査等の明確な目的があれば, 一般的なサービスにおいても d* framework は有効であると考えられる (C04, C05, C08). サービス開発の上流工程の方が, d* framework の有効性を活かすことができる (C06). d* framework の導入障壁として「作成コストがかかる」,「関係者への説明が難しい」という問題がある (C08, C09). しかし, 作成コストについては, 導入サービスを適切に選択すれば問題にならない可能性がある (C05).
また, アシュアランスケースの作成に関する課題として,「関係者の増加による作成の困難さの増大の克服」,「アシュアランスケースの肥大化に伴う作成の困難さの増大の克服」が挙げられている (C10, C11). これらのうち, アシュアランスケースの肥大化については, アシュアランスケースをモジュールに分割して記述する方法が提案されている[20][21][22]. しかし, アシュアランスケースをモジュールに分割するだけでは, アシュアランスケース

を意味のあるまとまりとして適切に分割することができない可能性がある．本研究で使用した d* framework は，アシュアランスケースを Actor に分割して記述する．これは，アシュアランスケースを，まとまった意味を持つ Actor という単位に分割して記述する手法であると言える．従って，d* framework により，肥大化するアシュアランスケースを適切に分割して記述することができると考えられる．

Table 16 本研究及び d* framework に対するステークホルダのコメント

ID	コメント
C01	サービスのディペンダビリティに関する情報を，このように整理したことは無かった.
C02	作成された d* framework については, 構造化されていたので理解できた. また, Actor 同士の関係についても理解できた.
C03	整理学等の作成のための知識があれば，担当内（ステークホルダ自身）でも d* framework の作成は可能である.
C04	今回作成した d* framework は，手段指向で作成しているので，直接役に立っているとは言えない．しかし，監査等の明確な目的があれば役に立つ可能性はある.
C05	銀行のシステム，防衛分野，航空管制，社会インフラ等のクリティカルなシステムの方が，d* framework を役立てることができると考えられる.
C06	企画，要件定義工程における活用の方が，d* framework を役立てることができると考えられる.
C07	システム開発者だけで d* framework を作成すると，その枠内のみの議論になってしまうという問題がある.
C08	d* framework を作るための工数が導入障壁になると考えられる．品質レビュー等に多くの時間を割いているシステムでは，導入が可能かもしれない.
C09	マネージャー等の上位層に対する価値の説明が難しいという問題がある.
C10	関係者が多くなると，作成が困難になる可能性があると考えられる.
C11	アシュアランスケースが大きくなり過ぎると，作成が困難になる可能性があると考えられる.

6.4.2. 従来の取組みとの比較

従来，サービスあるいはシステムの保証に関する議論が明示的に示されているケースは，少なかった．d* framework は，このようなサービスやシステムの保証に関する議論を文書化することで，明示的に示すための技術である．すなわち，従来の取り組みと比較すると，d* framework を使用することにより，サービスあるいはシステムの保証に関する議論を明示的に示すことができるという点で異なる．議論を明示的に示すことにより，サービスやシステムの保証に関する議論や対策をステークホルダ間で共有でき，議論の漏れを明確に

できる．本研究では Table 15 に示すとおり，作成した d* framework において，45 個の議論の不足箇所を指摘することができた．これらの指摘箇所は，サービスのディペンダビリティを保証するために議論が必要な箇所であり，従来の取り組みでは見過ごされていたと考えることができる．つまり，d* framework を使用することにより，従来の取り組みでは見過ごされていた，サービスのディペンダビリティを保証する上で必要な議論の発見に繋がったと考えることができる．

6.4.3. ソフトウェア工学における他の手法との関係

d* framework は，要求定義手法，設計手法，試験手法のようなシステム開発における他の既存手法に置き換わる技術ではない．サービスやシステムを保証するための議論を，構造的に整理するための新しい技術である．既存手法において作成されるドキュメントは，d* framework における議論を保証するための証拠となり得る．また，既存手法で作成されるドキュメントを，d* framework を使って保証するという関係もあり得る．今後 UML[104]，SysML[105]，FTA，テスティング，形式手法のような既存手法におけるドキュメントと d* framework の関係を，体系的に整理する必要がある．

6.4.4. エンタープライズ系情報サービスを対象としたアシュアランスケース

これまで，アシュアランスケースの適用対象は，組み込み系のシステムが中心であった．この理由として，システムの不具合が直接重大な損害に直結しやすいことや，リリース後の修正が困難であるという組み込み系のシステムが持つ特性が考えられる．本研究では，エンタープライズ系サービスである，SIer における IT システムを使用した社内向けサービスを対象として，d* framework（アシュアランスケース）を作成した．その結果，エンタープライズ系サービスにおいても，アシュアランスケースの作成が有効であるというステークホルダの評価が得られた（Table 16）．

6.4.5. 第三者による d* framework の作成

サービスオーナー，顧客，サービス提供者，システム開発者等の実際のサービスに関わっているステークホルダが，d* framework に関する知識が乏しい状況で d* framework を作成するのは困難である．そこで，本研究では，サービスに直接関係していない第三者が主導して d* framework を作成した．この第三者は，対象サービスに直接関係していないが，d* framework について，十分な知識を持つ者である．本研究では，直接のステークホルダではない第三者による d* framework の作成を可能とするため，d* framework の作成手順を定義した．この作成手順では，d* framework を作成するための入力情報として，サービスに関係する資料が利用される．定義した手順を使用して d* framework を作成した結果，実際のステークホルダから見ても違和感のない d* framework を作成することができた．これは，ステークホルダによる評価からもわかる（Table 16）．すなわち，本研究で提示した

手順によって，直接サービスに関係していない第三者が，対象サービスの d* framework を構築できるという知見が得られた．

Table 16 のコメントから，本研究を実施したことで，ステークホルダの d* framework に対する理解が高まったことがわかる．また十分な学習を積めば，ステークホルダも d* framework を作成でき，対象サービスの高信頼化に役立つと考えていることがわかった．

ステークホルダが自ら d* framework を作成する利点として，自分たちの実施するサービスの保証に関する議論を，直接 d* framework として記述できる点が挙げられる．第三者が主導する d* framework の作成では，第三者が間に入るというロスが生じる．また，第三者には，d* framework に対する十分な知識に加え，“一般的なシステム開発文書の読解力”，“ステークホルダとのディスカッションを主導する能力”等も必要となる．しかし，第三者が主導することで d* framework 作成の障壁を下げ，中立的な視点で d* framework を作成できるという利点がある．今後は，これらのトレードオフを考慮した上で，状況に応じた d* framework の作成が必要になることが考えられる．

6.4.6. d* framework の作成手順

サービスに直接関係していない第三者は，サービスに関する情報を持っていない．そのため，そのままの状態では，サービスのディペンダビリティの保証に関する議論を記述する d* framework を作成することができない．そのため，本研究では，サービスに直接関係していない第三者がサービスに関連する情報を得ることを目的として，サービスに関連する資料を使用した．一般的に，サービスに関連する資料の量は膨大である．本研究で実施したケーススタディにおいても，172 個の資料が存在した．これらの膨大な量の資料を使って，d* framework を作成するのは難しい作業となる．そこで，本研究では，サービスに関連する資料から，第三者が d* framework を作成する手順を明確にした．今回，この手順を使用することで，適切な d* framework を作成することができた．このことから，他のサービスについても，この作成手順を使用することにより，適切な d* framework を作成できると考えられる．すなわち，本研究で提案した手順は，他の d* framework の作成に使用することができる．

6.4.7. d* framework の作成時間

本研究では，d* framework を作成する際の作成時間を計測した．その結果，作成に要した時間は 14 時間だったことがわかった（Table 14）．1 日の作業時間を 5 時間程度として見積もると，これは約 3 営業日に換算することができ，比較的短い時間で d* framework を作成できたことがわかる．ただし，本研究において実際作成に使用されたドキュメントは，全体で 107 ページ（Table 11）であり大規模なものではなかった．そのため，比較的短い時間で d* framework を作成することが出来たと考えられる面もある．より大規模・複雑なサービスやシステムにおいて，d* framework の作成に要する時間がどのような規模に増大し

ていくのかを，今後明らかにしていく必要がある．

6.4.8. 第三者による d* framework の確認

本研究では，サービスに直接関わっているステークホルダではなく，サービスに直接関わっていない第三者による d* framework の確認を実施した．d* framework の持つ構造的な特徴を利用した結果，議論の不足箇所が全体で 45 箇所あることがわかった（Table 15）．すなわち，d* framework を利用することで，第三者がサービスのディペンダビリティに関する議論の不足箇所を確認できるということがわかった．つまり，d* framework を使用することによるサービスのディペンダビリティの向上が，第三者においても可能であることが明らかになった．第三者が指摘した議論の不足箇所は，ステークホルダが議論を実施し，補っていく必要がある箇所である．

6.4.9. 本研究の限界

本研究には，次のような限界がある．

- 本研究は，エンタープライズ系サービスに対する d* framework の作成事例の 1 つに過ぎない．エンタープライズ系サービスに対する d* framework の作成事例，及び第三者による d* framework の作成事例の数としては，少ないという限界がある．

- 本研究では，SIer における社内サービスを対象として d* framework を作成した．このサービスは，対象の企業にとっては重要なサービスであるが，社会的に重大な影響を持つサービスではない．従って，本研究は，社会的に重要なサービスに対しての適用事例ではないという限界がある．ただし，利用者，IT システム，システム開発者，システム運用者がステークホルダとして存在するという点は，重要なサービスにおいても同様であるため，本研究で作成した d* framework の構造を適用できる可能性がある．

6.5. まとめ

本研究では，今までほとんど存在しなかった，エンタープライズ系サービスに対する d* framework（アシュアランスケース）の作成を実施した．また，本研究では，サービスに直接関係していない第三者による d* framework の作成を試みた．第三者による d* framework の作成を可能とするため，事前に作成手順を定義し，その手順を使用して d* framework を作成した．その結果，サービスに関係するステークホルダからも納得感を得られる d* framework が，第三者により作成できるという知見が得られた．さらに，d* framework の構造的な特徴を利用することにより，第三者によるサービスのディペンダビリティに関する確認が可能であるという知見も得られた．また，本研究による d* framework

作成後のステークホルダの評価より，実際のサービスに対して d* framework を適用することの有用性が示された．

今後は，今回実施したような d* framework の作成を他の事例でも実施し，d* framework の適用事例を増やしていく必要がある．様々な事例を実施することにより，新しい知見の獲得が期待できる．また，本研究において示した d* framework の作成手順についても，適用事例を増やすことで改善できることが期待される．

7. 結論

7.1. 本研究のまとめ

本研究では，アシュアランスケースに関する研究を実施した．アシュアランスケースとは，システムのディペンダビリティを保証するために実施する議論を記述した文書である．システムのディペンダビリティの保証に関する関心が高まり，その重要性が認識されるに伴い，アシュアランスケースの研究が進められてきた．しかし，従来のアシュアランスケースに関する研究には，Table 1 で示される欠陥が存在した．本研究では，これらの欠陥に対処するため，“d* framework の提案（Actor の導入)”，“d* framework への責任属性の導入（責任属性の導入)”，“コラボレーション図を使った d* framework の作成（設計法の検討)”を実施した．また，Actor の導入に関する評価を実施するため，“適用実験に基づく評価”も実施した．これらの実施事項については，Figure 1 で整理している．本研究において実施したこれらの研究について，得られた主な結果を以下に列挙する．

① d* framework の提案（3 章）

3 章では，d* framework を提案した．d* framework とは，Actor の概念を導入したアシュアランスケースの作成手法である．また，d* framework を使用して作成されたアシュアランスケース自体も d* framework と呼ばれる．d* framework は，従来のアシュアランスケースの作成手法である GSN を拡張して定義された手法である．d* framework では，GSN において使用されている要素に加えて, 新規要素の Actor が定義されている. d* framework における Actor は，d* framework が対象とするシステムの構成要素を示す要素として使用される．システム，サブシステム，コンポーネント，人，組織等を Actor として定義することができる．Actor の概念が導入された d* framework には，下記の 2 種類が存在する．実際に d* framework を作成する際には，下記の 2 種類の d* framework を複数作成して，システムの保証に関する議論を記述する．

1. Actor が他の Actor に対して責任を遂行することを示す d* framework. すなわち, Actor 間の依存関係を示す d* framework.

2. Actor 自体がディペンダブルであることを示す d* framework

本研究において d* framework を提案したことで，アシュアランスケースに対する Actor の導入を実施することができた．d* framework を使用し，システム，サブシステムやシステムを構成するコンポーネントを Actor として定義することにより，複数のサブシステムや複数のコンポーネントから構成されるシステムの保証を明確に示すことができるようになったと考えられる．

② d* framework に対する責任属性の導入（4 章）

4 章では，d* framework に対する責任属性の導入に関する研究について述べた．d* framework とは，アシュアランスケースの 1 種であり，システムの保証に関する議論について記述された文書である．従来のアシュアランスケースでは，対象とするシステムのディペンダビリティの保証に関する議論について，誰が責任を持つのかが必ずしも明確にされていなかった. このような状況は, d* framework においても同様であった. 本研究では，d* framework に責任属性を導入することで，システムの保証に関する議論について，誰が責任を持つのかを明確にした．具体的には，d* framework に対する新規要素として Agent を追加した．Agent は，d* framework において記述されているディペンダビリティを保証する議論に対して, 責任を持つ責任主体を示す要素として定義されている. すなわち, Agent として記述された要素は，d* framework 内の他の要素（Actor，Goal，Strategy，Context，Evidence）に対して責任を持つ．そのため，Agent として記述できるのは，議論に対して責任を持つことができる人や組織等であるとした．逆に，それ自体では責任を持つことができないシステムやサブシステム等は，Agent として記述することはできないとした．本研究において，d* framework に対して Agent の要素を導入したことで，d* framework を使用する際，責任主体の概念を考慮することができるようになったと考えることができる．

③ コラボレーション図を使った d* framework の作成（5 章）

5 章では，コラボレーション図を使用した d* framework の作成手法に関する研究について述べた．d* framework を含む一般的なアシュアランスケースは，ステークホルダによる対象のディペンダビリティの保証に関する議論の内容を記述することで作成される．しかし，この方法は時間がかかり非効率であると考えられる．そこで，d* framework の対象に関して既存の情報が存在する場合，それらの情報を使用して d* framework を作成することで，d* framework の作成効率を上げることが考えられる．例えば，既存のソフトウェア工学の手法により，対象システムについて作成されたモデルを，d* framework の作成に使用することが考えられる．本研究では，d* framework を作成するために，対象のシステムに対して作成されたコラボレーション図を使用することを試みた．コラボレーション図には，システムを構成するオブジェクトと，それらオブジェクト間のメッセージ情報が定義されている．本研究において，コラボレーション図に定義されている情報は，d* framework を作成するのに使用される．本研究では，コラボレーション図から d* framework を作成するための手順を提案した．提案した手順には，コラボレーション図に含まれる情報を使って，d* framework における Goal を定義するための方法も示されている．

現状のアシュアランスケース記法には，アシュアランスケースと既存の設計手法との関係が不明確であるという欠陥が存在する．本研究を実施することにより，d* framework（アシュアランスケース）と，既存の設計手法の 1 つであるコラボレーション図との関係を示

すことができた．今後，コラボレーション図以外の既存の設計手法についても，アシュアランスケースとの関係を明確にする必要がある．これは，今後の課題となる．

④　適用実験に基づく評価（6 章）

6 章では，適用実験に基づく d* framework の評価に関する研究について述べた．本研究では SIer で実際に使用されている IT システムを使ったサービスを対象として，d* framework を作成し，d* framework の評価を実施した．d* framework の作成に伴うステークホルダの負担を軽減するため，対象のサービスに直接関係していない第三者が中心となって d* framework を作成した．対象のサービスに直接関係していない第三者が，対象のサービスの d* framework を作成することは困難である．そこで，本研究では，“対象のサービスに直接関係していない第三者が d* framework を作成する際の作成手順”を定義し，その手順を使用して d* framework を作成した．その結果，132 個の要素を持つ d* framework が作成された．作成された d* framework は，対象のサービスの関係者であるステークホルダから見ても違和感のないものとなった．また，本研究では，作成された d* framework を使用して，ディペンダビリティに関する議論の漏れを確認する実験も実施した．議論の抜けもれの確認には，d* framework が持つ構造的な特徴を利用した．その結果，全体で 45 個の議論の漏れを識別することができた．

本研究では，d* framework の適用可能性を評価するため，対象としたサービスのステークホルダに対して，本研究の取り組み及び d* framework に関するヒアリングを実施した．その結果，d* framework を作成することで，サービスのディペンダビリティに関する議論を分かりやすく整理することができ，対象サービスを考慮すれば d* framework を実際に活用できるという評価が得られた．この結果より，d* framework は，実際の現場においても利用価値のあるものであることが示された．また，本研究を実施したことにより，サービスに直接関係していない第三者でも，d* framework の作成が可能であることが示された．

7.2. 今後の課題

上記で述べたとおり，本研究では，アシュアランスケースに対して Actor の導入（d* framework の提案）と責任属性の導入を実施した．Figure 1 に示したように，Actor の導入については，記法の提案，記法の評価，設計法の提案を実施した．また，責任属性の導入については，記法の提案のみ実施した．本研究では，責任属性の導入について，記法の評価及び設計法の提案までは実施できなかった．今回実施できなかった，責任属性の導入に関する記法の評価及び設計法の提案は，今後の課題となる．

既存の設計手法とアシュアランスケースの関係の明確化について，本研究では，コラボレーション図を使用した d* framework の設計法の提案を通して，既存の設計手法の 1 つであるコラボレーション図と d* framework との関係を明確にした．しかし，既存の設計手法は，コラボレーション図のみではない．その他の設計手法も多数存在する．それら既存の設計

手法と d* framework との関係を明確にすることも今後の課題となる．特に，サービスやシステムのアーキテクチャを記述するモデルと，d* framework の関連を明確にし，相補的に使用できるようにすることで, d* framework の有用性を高めることができると期待される．本研究では，SIer において実際に使用されているサービスに対して d* framework を作成することで，d* framework の評価を実施した．しかし，d* framework を評価するための事例としては，まだまだ少ないという限界がある．d* framework の評価という観点から考えると，適用評価事例を増やしていくことも今後の課題となる．

謝辞

本研究をまとめるにあたり，種々の御指導，御鞭撻，御支援を賜りました，名古屋大学大学院情報科学研究科情報システム学専攻教授　山本修一郎博士，同教授　坂部俊樹博士，同教授　高田広章博士，同准教授　森崎修司博士に心から感謝の意を表します．
また，本論文において実施した研究について，貴重な御討論，御助言を頂いた電気通信大学大学院情報システム学研究科助教授　松野裕博士，日本電信電話株式会社（NTT）ソフトウェアイノベーションセンタ　星野隆プロジェクトマネージャーに深く感謝致します．
6 章．d* framework の有効性の検証では，株式会社 NTT データ執行役員　木谷強博士，同社技術開発本部サービスイノベーションセンタロボティクスインテグレーション推進室室長　渡辺真太郎博士，同社技術開発本部 ALM ソリューションセンタ　丹羽隆部長，同社技術開発本部 ALM ソリューションセンタ　堀口智久氏，同社技術開発本部 ALM ソリューションセンタ　荻野広樹氏，日本電信電話株式会社（NTT）ソフトウェアイノベーションセンタ　塚本英昭主幹研究員にご協力頂きました．深く感謝致します．
また，本研究を進めるにあたり御協力，御議論して頂きました名古屋大学大学院情報科学研究科情報システム学専攻山本研究室の諸氏に心から感謝致します．
最後に，本研究を進めるにあたり，様々な面で支えになってくれた家族に心から感謝の意を表します．

参考文献

[1] 山本修一郎，松野裕，“ディペンダビリティケース作成法に関する一考察．”電子情報通信学会技術研究報告．KBSE，知能ソフトウェア工学，vol.112，no.165，pp61-66，2012.

[2] R. L. Ackoff. “Towards a system of systems concepts.” Management science 17.11, pp661-671, 1971.

[3] 玉井哲雄，“ソフトウェア工学の基礎.”岩波書店，2004.

[4] H. C. Purchase, L. Colpoys, M. McGill, and D Carrington. "UML collaboration diagram syntax: an empirical study of comprehension." Visualizing Software for Understanding and Analysis, 2002. Proceedings. First International Workshop on. IEEE, 2002.

[5] 松野裕，パトゥヴァイセ，山本修一郎，“アシュアランスケースへの構造化文書の適用に関する調査．”電子情報通信学会技術研究報告．KBSE，知能ソフトウェア工学，vol.112，no.165，pp49-54，2012.

[6] P. Bishop and R. Bloomfield. “A methodology for safety case development.” Industrial Perspectives of Safety-critical Systems. Springer London, pp194-203, 1998.

[7] T. S. Ankrum and A. H. Kromholz. “Structured assurance cases: Three common standards.” High-Assurance Systems Engineering, 2005. HASE 2005. Ninth IEEE International Symposium on. IEEE, 2005.

[8] T. S. Ankrum and A. H. Kromholz. “Structured assurance cases: Three common standards.” In Slides presentation at the Association for Software Quality (ASQ) Section 509 meeting, 2006.

[9] M. A. Cusumano, “Reflections on the Toyota debacle.” Communications of the ACM 54.1, pp33-35, 2011.

[10] C. C. Howell, et al. “Workshop on assurance cases: best practices, possible obstacles, and future opportunities 1 July 2004, Florence, Italy.” Dependable Systems and Networks, 2004 International Conference on. IEEE, 2004.

[11] P. Bishop, R. Bloomfield, and S. Guerra. “The future of goal-based assurance cases.” Proceedings of Workshop on Assurance Cases. Supplemental Volume of the 2004 International Conference on Dependable Systems and Networks. 2004.

[12] 山本修一郎，“機能安全規格対応に向けた説明力向上の課題と方向性．”第 13 回システム検証セミナー，pp185-202，2013.

[13] A. Avizienis, et al. “Fundamental Concepts of Computer Systems Dependability.” Proc. of the Workshop on Robot Dep., 2001.

[14] A. Avizienis, et al. “Basic concepts and taxonomy of dependable and secure computing.” Dependable and Secure Computing, IEEE Transactions on 1.1, pp11-33, 2004.

[15] G. Despotou and T. Kelly. “Extending the safety case concept to address dependability.” Proceedings of the 22nd international system safety conference. 2004.

[16] GSN COMMUNITY STANDARD VERSION 1, http://www.goalstructuringnotation.info/documents/GSN_Standard.pdf

[17] T. Kelly. “Arguing Safety - A Systematic Approach to Managing Safety Cases.” PhD thesis, University of York, 1998.

[18] T. Kelly and J. McDermid, “Safety case patterns-reusing successful arguments.” In IEE Colloquium on Understanding Patterns and Their Application to System Engineering, 1998.

[19] Claims, Arguments and Evidence (CAE), http://www.adelard.com/asce/choosing-asce/cae.html

[20] I. Bate and T. Kelly. “Architectural considerations in the certification of modular systems.” Reliability Engineering & System Safety 81.3, pp303-324, 2003.

[21] T. Kelly. “Using software architecture techniques to support the modular certification of safety-critical systems.” Proceedings of the eleventh Australian workshop on Safety critical systems and software-Volume 69. Australian Computer Society, Inc., 2007.

[22] J. Fenn, et al. “Safety case composition using contracts-refinements based on feedback from an industrial case study.” The Safety of Systems. Springer London, pp133-146, 2007.

[23] M. Tokoro, ed. Open systems dependability: dependability engineering for ever-changing systems. CRC Press, 2012.

[24] Y. Matsuno, J. Nakazawa, M. Takeyama, M. Sugaya and Y. Ishikawa, “Towards a language for communication among stakeholders.” Dependable Computing (PRDC), 2010 IEEE 16th Pacific Rim International Symposium on. IEEE, 2010.

[25] Y. Matsuno, H. Takamura, and Y. Ishikawa. “A Dependability Case Editor with Pattern Library.” HASE. 2010.

[26] Y. Matsuno and S. Yamamoto, “An implementation of GSN community standard.” Assurance Cases for Software-Intensive Systems (ASSURE), 2013 1st International Workshop on. IEEE, 2013.

[27] D-case editor a typed assurance case editor, http://www.dependable-os.net/tech/D-CaseEditor/.

[28] DEOS project, http://www.crest-os.jst.go.jp/.

[29] ©2013 the eclipse foundation. all rights reserved: Featured eclipse project, http://www.eclipse.org/.

[30] 松野裕，山本修一郎，“アシュアランスケースツールへのプログラミング言語技術の適用.”電子情報通信学会技術研究報告. KBSE, 知能ソフトウェア工学, vol.112, no.496, pp73-78，2013.

[31] T. Kelly and J. A. McDermid. “Safety case construction and reuse using patterns.” Safe Comp 97. Springer London, pp55-69, 1997.

[32] T. Kelly and R. Weaver. “The goal structuring notation–a safety argument notation.” Proceedings of the dependable systems and networks 2004 workshop on assurance cases, 2004.

[33] D. Jackson, M. Thomas and L. I. Millett, “Software for dependable systems: Sufficient Evidence?,” National Research Council, 2007.

[34] R. A. Weaver. “The safety of software: Constructing and assuring arguments.” University of York, Department of Computer Science, 2003.

[35] R. Alexander, et al. “Safety cases for advanced control software: Safety case patterns.” YORK UNIV (UNITED KINGDOM) DEPT OF COMPUTER SCIENCE, 2007.

[36] R. Bloomfield and P. Bishop. “Safety and assurance cases: Past, present and possible future–an Adelard perspective.” Making Systems Safer. Springer London, pp51-67, 2010.

[37] S. Yamamoto and Y. Matsuno. “An evaluation of argument patterns to reduce pitfalls of applying assurance case.” Assurance Cases for Software-Intensive Systems (ASSURE), 2013 1st International Workshop on. IEEE, 2013.

[38] 山本修一郎，松野裕，“ディペンダビリティケース分解パターンについての考察.”電子情報通信学会技術研究報告. KBSE, 知能ソフトウェア工学，vol.112，no.496，pp67-72，2013.

[39] T. Kelly, “A six-step Method for Developing Arguments in the Goal Structuring Notation (GSN).” Technical report. York Software Engineering, UK, 1999.

[40] © 2011 科学技術振興機構，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 White Paper Version3.0，2011.

[41] Copyright © 2013, The Open Group, Open Group Standard Real-Time and Embedded Systems: Dependability through Assuredness™ (O-DA) Framework, 2013.

[42] 松野裕，高井利憲，ヴァイセパトゥー，山本修一郎，“アシュアランスケース構築法の

提案.”電子情報通信学会技術研究報告. KBSE, 知能ソフトウェア工学, vol.112, no.314, pp13-17, 2012.
[43] 松村昌典, 松野裕, 山本修一郎, “ディペンダビリティ用語辞書構築方法の提案.” 電子情報通信学会技術研究報告. KBSE, 知能ソフトウェア工学, vol.112, no.314, pp115-120, 2012.
[44] 松村昌典, 松野裕, 山本修一郎, “ディペンダビリティケース用語構成規則の提案.” 電子情報通信学会技術研究報告. KBSE, 知能ソフトウェア工学, vol.112, no.419, pp29-34, 2013.
[45] 松村昌典, 松野裕, 山本修一郎, “ディペンダビリティケース用語構成規則の適用評価.” 電子情報通信学会技術研究報告. KBSE, 知能ソフトウェア工学, vol.112, no.496, pp61-66, 2013.
[46] P. J. Graydon, J. C. Knight and E. A. Strunk, “Assurance based development of critical systems.” Dependable Systems and Networks, 2007. DSN'07. 37th Annual IEEE/IFIP International Conference on. IEEE, 2007.
[47] N. Basir, E. Denney and B. Fischer, “Deriving safety cases from automatically constructed proofs.” Systems Safety 2009. Incorporating the SaRS Annual Conference, 4th IET International Conference on, 2009.
[48] J. Rushby, “Formalism in safety cases.” Making Systems Safer. Springer London, pp3-17, 2010.
[49] Y. Matsuno and K. Taguchi, “Parameterised argument structure for GSN patterns.” Quality Software (QSIC), 2011 11th International Conference on. IEEE, 2011.
[50] M. Takeyama, “Programming assurance cases in Agda.” In ICFP, pp142, 2011.
[51] 高間翔太, 松野裕, 山本修一郎, “スーパーコンピュータの運用手順に対するディペンダビリティの確認手法の提案.” 電子情報通信学会技術研究報告. KBSE, 知能ソフトウェア工学, vol.112, no.165, pp37-42, 2012.
[52] 高間翔太, 松野裕, 山本修一郎, “ディペンダビリティ・コンテクストの推定手法の提案.”電子情報通信学会技術研究報告. KBSE, 知能ソフトウェア工学, vol.112, no.314, pp25-30, 2012.
[53] 徳野達也, 松野裕, 山本修一郎, “エンタープライズ・アーキテクチャに対するディペンダビリティケース作成法の提案.” 電子情報通信学会技術研究報告. KBSE, 知能ソフトウェア工学, vol.112, no.165, pp145-150, 2012.
[54] 徳野達也, 松野裕, 山本修一郎, “TOGAF NEXT に対する ADM プロセステンプレートの提案.” 電子情報通信学会技術研究報告. KBSE, 知能ソフトウェア工学, vol.112, no.314, pp103-108, 2012.
[55] 田中康平, 松野裕, 中坊嘉宏, 白坂成功, 中須賀真一, “アシュアランスケースを用いた小型人工衛星の品質保証.” 信頼性シンポジウム発表報文集, pp63-66, 2012.

[56] G. Despotou and T Kelly, “Using Scenarios to Identify and Trade-off Dependability Objectives in Design.” Proceedings of the 23rd International System Safety Conference (ISSC), 2005.

[57] I. Habli, et al. “Model-based assurance for justifying automotive functional safety.” No. 2010-01-0209. SAE Technical Paper, 2010.

[58] O. Sokolsky, I. Lee and M. Heimdahl, “Challenges in the regulatory approval of medical cyber-physical systems.” Embedded Software (EMSOFT), 2011 Proceedings of the International Conference on. IEEE, 2011.

[59] 小林茂憲，山本修一郎，“保証ケースを用いたサービス提供判断方法の提案." 電子情報通信学会技術研究報告.” 電子情報通信学会技術研究報告. KBSE, 知能ソフトウェア工学，vol.111, no.489, pp7-12, 2012.

[60] S. Yamamoto and Y. Matsuno. “A review method based on a matrix interpretation of GSN.” JCKBSE. 2012.

[61] 松野裕，山本修一郎，“ユースケース分析に基づくディペンダビリティケース作成法の提案.”電子情報通信学会技術研究報告. KBSE, 知能ソフトウェア工学, vol.112, no.419, pp19-24，2013.

[62] 山本修一郎，“アーキテクチャに基づく検証ケースの提案.” 電子情報通信学会技術研究報告. KBSE, 知能ソフトウェア工学，vol.112，no.496，pp79-83，2013.

[63] G. Despotou and T. Kelly, “Extending safety deviation analysis techniques to elicit flexible dependability requirements.” System Safety, 2006. The 1st Institution of Engineering and Technology International Conference on. IET, 2006.

[64] G. Despotou and T. Kelly, “Design and development of dependability case architecture during system development.” 25th International System Safety Conference. System Safety Society. 2007.

[65] J. A. McDermid, “Software safety: where's the evidence?” Proceedings of the Sixth Australian workshop on Safety critical systems and software-Volume 3. Australian Computer Society, Inc., 2001.

[66] N. G. Leveson and J. D. Herrera. “Safeware: system safety and computers.” Vol. 680. Reading: Addison-Wesley, 1995.

[67] C. A. Ericson. “Hazard analysis techniques for system safety.” John Wiley & Sons, 2005.

[68] R. Lutz and A. Patterson-Hine, “Using fault modeling in safety cases.” Software Reliability Engineering, 2008. ISSRE 2008. 19th International Symposium on. IEEE, 2008.

[69] J. P. Near, et al. “A lightweight code analysis and its role in evaluation of a dependability case.” Software Engineering (ICSE), 2011 33rd International

Conference on. IEEE, 2011.
[70] SCSK, VDM information web site, http://www.vdmtools.jp/.
[71] 栗田太郎, “3. 携帯電話組込み用モバイル FeliCa IC チップ開発における形式仕様記述手法の適用 (Part II: 産業界への応用,< 特集> フォーマルメソッドの新潮流).” 情報処理 49.5, pp506-513, 2008.
[72] Verifying Multi-threaded Software with Spin, http://spinroot.com/spin/whatispin.html.
[73] B. Bérard, et al. “Systems and software verification: model-checking techniques and tools.” Springer Publishing Company, Incorporated, 2010.
[74] G. J. Holzmann, “The SPIN model checker: Primer and reference manual.” Vol. 1003. Reading: Addison-Wesley, 2004.
[75] E. M. Clarke, O. Grumberg, and D. Peled. “Model checking.” MIT press, 1999.
[76] 猿渡卓也, 塚本英昭, 神谷慎吾, 宮田俊介 “モデル検査を使ったデッドロック可能性検出.”電子情報通信学会技術研究報告. KBSE, 知能ソフトウェア工学, vol.112, no.496, pp43-48, 2013.
[77] LTSA - Labelled Transition System Analyser, http://www.doc.ic.ac.uk/ltsa/.
[78] UPPAAL Home, http://www.uppaal.org/.
[79] D. Jackson, “Alloy: a lightweight object modelling notation.” ACM Transactions on Software Engineering and Methodology (TOSEM) 11.2, pp256-290, 2002.
[80] D. Jackson, alloy: a language & tool for relational models, http://alloy.mit.edu/alloy/index.html.
[81] 山本修一郎, 松野裕, “ディペンダビリティケースへの責任属性の導入法の検討.” 電子情報通信学会技術研究報告. KBSE, 知能ソフトウェア工学, vol.112, no.314, pp85-90, 2012.
[82] A. V. Lamsweerde. “Requirements engineering: from system goals to UML models to software specifications.” 2009.
[83] M. A. Sujan, F. Koornneef and U. Voges. “Goal-based safety cases for medical devices: opportunities and challenges.” Computer Safety, Reliability, and Security. Springer Berlin Heidelberg, pp14-27, 2007.
[84] E. Denney, G. Pai and I. Habli, “Perspectives on software safety case development for unmanned aircraft.” Dependable Systems and Networks (DSN), 2012 42nd Annual IEEE/IFIP International Conference on. IEEE, 2012.
[85] V. Patu, Y. Matsuno and S. Yamamoto, “Application of D-Case to the data-upload flow diagram scenario of the Distributed E-Learning System called KISSEL.” Software Reliability Engineering Workshops (ISSREW), 2012 IEEE 23rd International Symposium on. IEEE, 2012.

[86] P. Chinneck, D. Pumfrey, and T. Kelly. “Turning up the HEAT on safety case construction.” Practical Elements of Safety. Springer London, pp223-240, 2004.

[87] J. R. Inge, “The safety case, its development and use in the United Kingdom.” PROC. 25TH INTERNATIONAL SYSTEM SAFETY CONFERENCE. 2007.

[88] 山本修一郎, “システムダイナミクスによる IT サービスプロセスの評価.” 電子情報通信学会技術研究報告. KBSE, 知能ソフトウェア工学, vol.112, no.64, pp1-6, 2012.

[89] AAMI, Safety Assurance Cases for Medical Devices, http://www.aami.org/meetings/courses/safety.html.

[90] Dependable Computing LLC, Assurance Cases Courses, http://www.dependablecomputing.com/courses.html.

[91] V. Patu, Y. Matsuno and S. Yamamoto, “A proposed research framework for “Dependability Science” based on assurance cases.” 電子情報通信学会技術研究報告. KBSE, 知能ソフトウェア工学, vol.112, no.165, pp55-59, 2012.

[92] A. V. Lamsweerde and E. Letier. “Integrating obstacles in goal-driven requirements engineering.” Proceedings of the 20th international conference on Software engineering. IEEE Computer Society, 1998.

[93] I. Sommerville, et al. “Deriving information requirements from responsibility models.” Advanced Information Systems Engineering. Springer Berlin Heidelberg, 2009.

[94] D. Greenwood and I. Sommerville. “Responsibility modeling for the sociotechnical risk analysis of coalitions of systems.” Systems, Man, and Cybernetics (SMC), 2011 IEEE International Conference on. IEEE, 2011.

[95] Socio-technical systems engineering handbook. http://archive.cs.st-andrews.ac.uk/STSEHandbook/FullHandbook.pdf.

[96] C. Feltus and M. Petit. “Building a responsibility model including accountability, capability and commitment.” Availability, Reliability and Security, 2009. ARES'09. International Conference on. IEEE, 2009.

[97] C. Feltus, M. Petit and E. Dubois, “Strengthening employee's responsibility to enhance governance of IT: COBIT RACI chart case study.” Proceedings of the first ACM workshop on Information security governance. ACM, 2009.

[98] C. Feltus and M Petit, “Building a responsibility model using modal logic-towards Accountability, Aapability and Commitment concepts.” Computer Systems and Applications, 2009. AICCSA 2009. IEEE/ACS International Conference on. IEEE, 2009.

[99] K. Boness and R. Harrison. “Goal sketching with activity diagrams.” Software Engineering Advances, 2008. ICSEA'08. The Third International Conference on.

IEEE, 2008.

[100] R. Strens and J. Dobson. “Responsibility modelling as a technique for organisational requirements definition.” Intelligent Systems Engineering 3.1, pp.20-26, 1994.

[101] E. S. Yu. “Towards modelling and reasoning support for early-phase requirements engineering.” Requirements Engineering, 1997, Proceedings of the Third IEEE International Symposium on. IEEE, 1997.

[102] E. S. Yu. “Modeling organizations for information systems requirements engineering.” Requirements Engineering, 1993, Proceedings of IEEE International Symposium on. IEEE, 1993.

[103] Copyright © 2013 The University of Electro-Communications, Nagoya University All Rights Reserved. “D-Case 安全・安心，ディペンダビリティ合意形成のための手法とルール,” http://www.dcase.jp/index.html.

[104] Object Management Group, Inc., Unified Modeling Language™ (UML®) Resource Page, http://www.uml.org/.

[105] Object Management Group, Inc., OMG Systems Modeling Language The Official OMG SysML site, http://www.omgsysml.org/.

本論文に関する原著論文

【論文】

[106] Takuya Saruwatari and Shuichiro Yamamoto. “Definition and application of an assurance case development method (d*).” SpringerPlus, Vol. 2, No. 1, pp.1-8, 2013.（3 章）

[107] Takuya Saruwatari and Shuichiro Yamamoto, “D* framework creation procedure from collaboration diagram.” IT CoNvergence PRActice (INPRA), Vol. 2, No. 2, pp43-54, 2014.（5 章）

[108] 猿渡卓也，丹羽隆，山本修一郎．“IT システムを利用するサービスのアシュアランスケース作成，” 情報処理学会デジタルプラクティス（採録決定）（6 章）

【国際会議】

[109] Takuya Saruwatari, Takashi Hoshino, and Shuichiro Yamamoto. “Evaluation of an assurance case development method (d*).” In JCKBSE, pp.72-80, 2012.（3 章）

[110] Takuya Saruwatari, Takashi Hoshino, and Shuichiro Yamamoto. “Method to share responsibility knowledge of dependability cases.” In 17th International Conference in Knowledge Based and Intelligent Information and Engineering Systems - KES2013, 2013.（4 章）

[111] Takuya Saruwatari, Yutaka Matsuno, and Shuichiro Yamamoto. “A comparative study of d* framework and GSN.” In Software Reliability Engineering Workshops (ISSREW), 2013 IEEE International Symposium on, pp.315-320. IEEE, 2013.（3 章）

[112] Takuya Saruwatari and Shuichiro Yamamoto. “Creation of assurance case using collaboration diagram.” In The 2014 Asian Conference on Availability, Reliability and Security (AsiaARES 2014), 2014.（5 章）

【国内会議（査読あり）】

[113] 猿渡卓也，山本修一郎．“アクターを導入したアシュアランスケースにおける責任関係パターンの検討.” In AsianPLoP 2014: 3rd Asian Conference on Pattern Languages of Programs, 2014.（2 章）

[114] 猿渡卓也，山本修一郎．“Assurance case 作成手法(d*) の適用評価.” ソフトウェア・シンポジウム SS2012, 2012.（3 章）

【研究会】

[115]　猿渡卓也, 松野裕, 星野隆, 山本修一郎. “ModularGSN の定式化.” 電子情報通信学会技術研究報告. KBSE, 知能ソフトウェア工学, Vol. 112, No.165, pp.151-156, 2012.（2 章）