D-Case実証実験

D-CaseとSysML/UMLの連携

株式会社 チェンジビジョン　山本光洋

本日の内容

１．D-CaseとSysML/UML連携の
実証実験について

２．実証実験におけるD-Caseの作成について

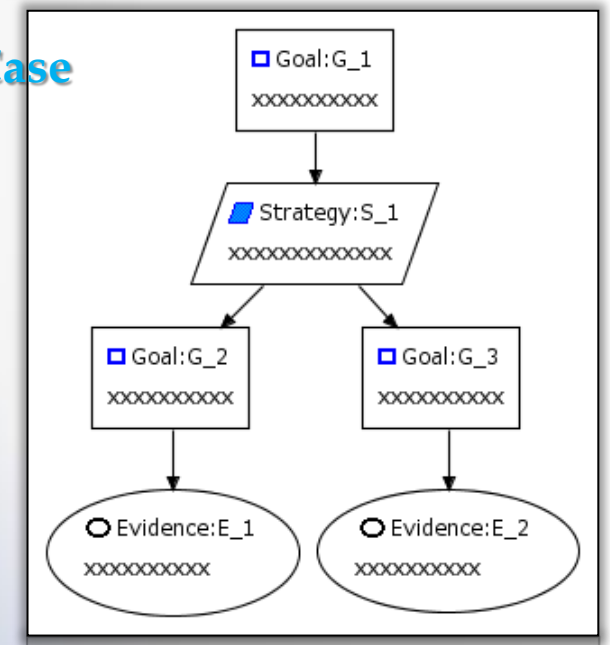
1．D-CaseとSysML/UMLの連携の実証実験

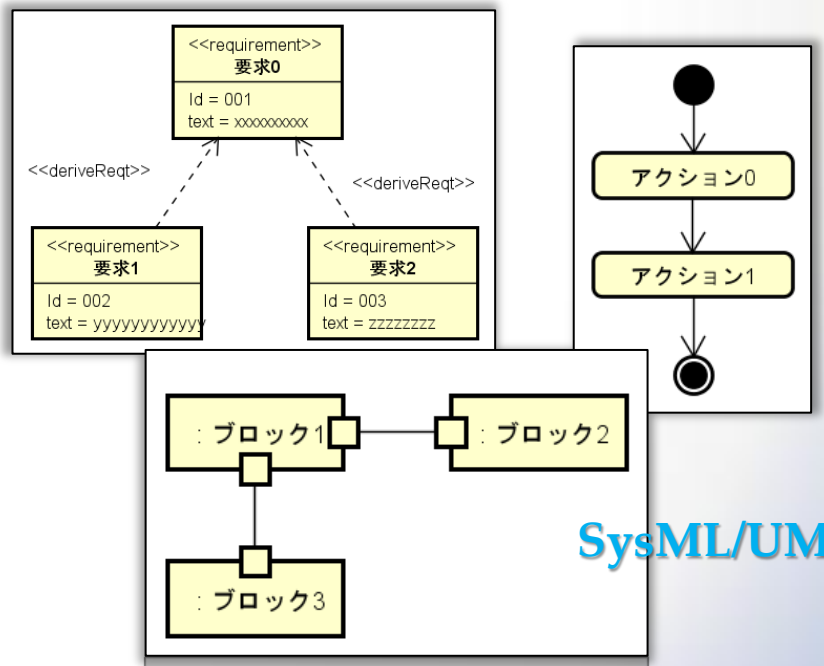
■ 実証実験の目的

D-CaseをSysML/UMLによるモデルベース開発の現場に適用し、連携することによって得られる効果を検証すること。

オブジェクト指向の概念に基づくモデリング言語であり、システムの構造や振舞いを表現する。

D-Case



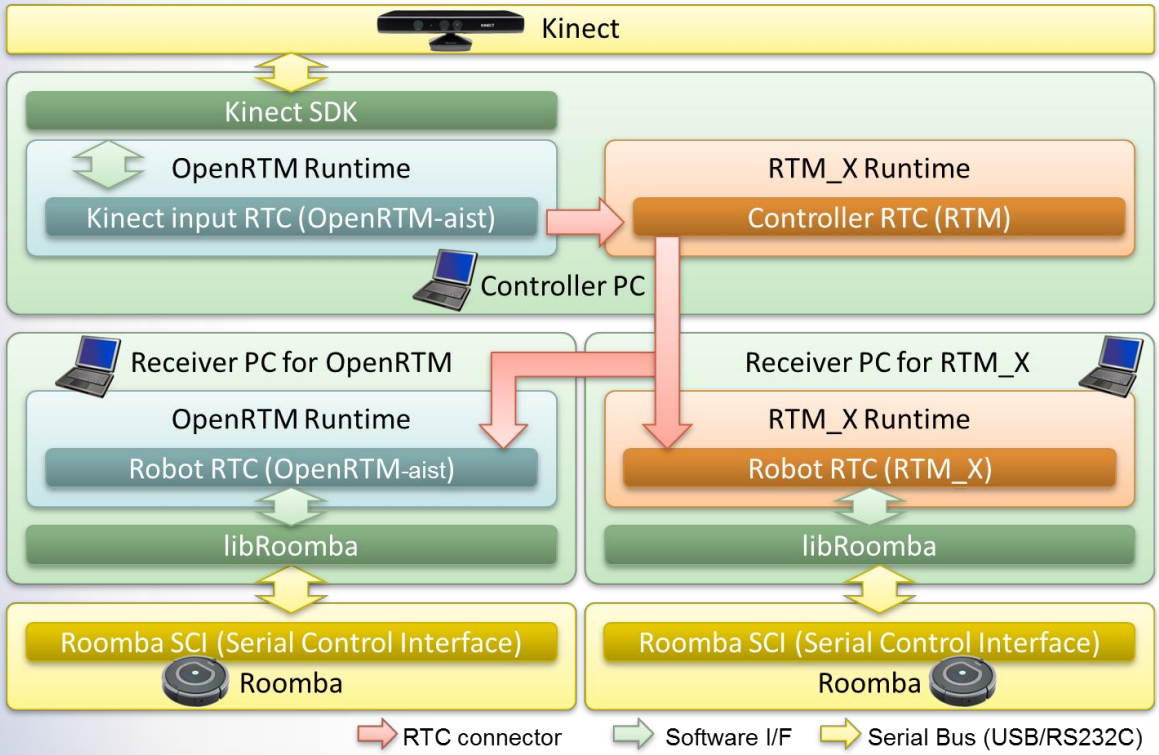


SysML/UML

ChangeVision

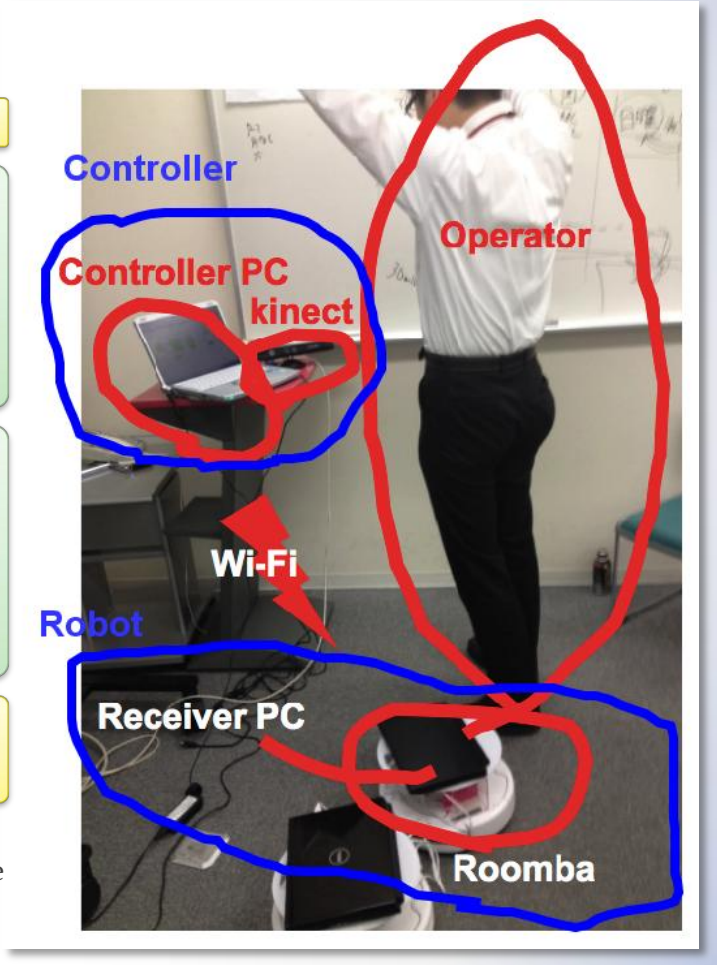
実証実験の適用対象としたシステムの概要

SysMLを用いたシステム開発の事例紹介のためのデモシステム



RTM : Robotics Technology Middleware

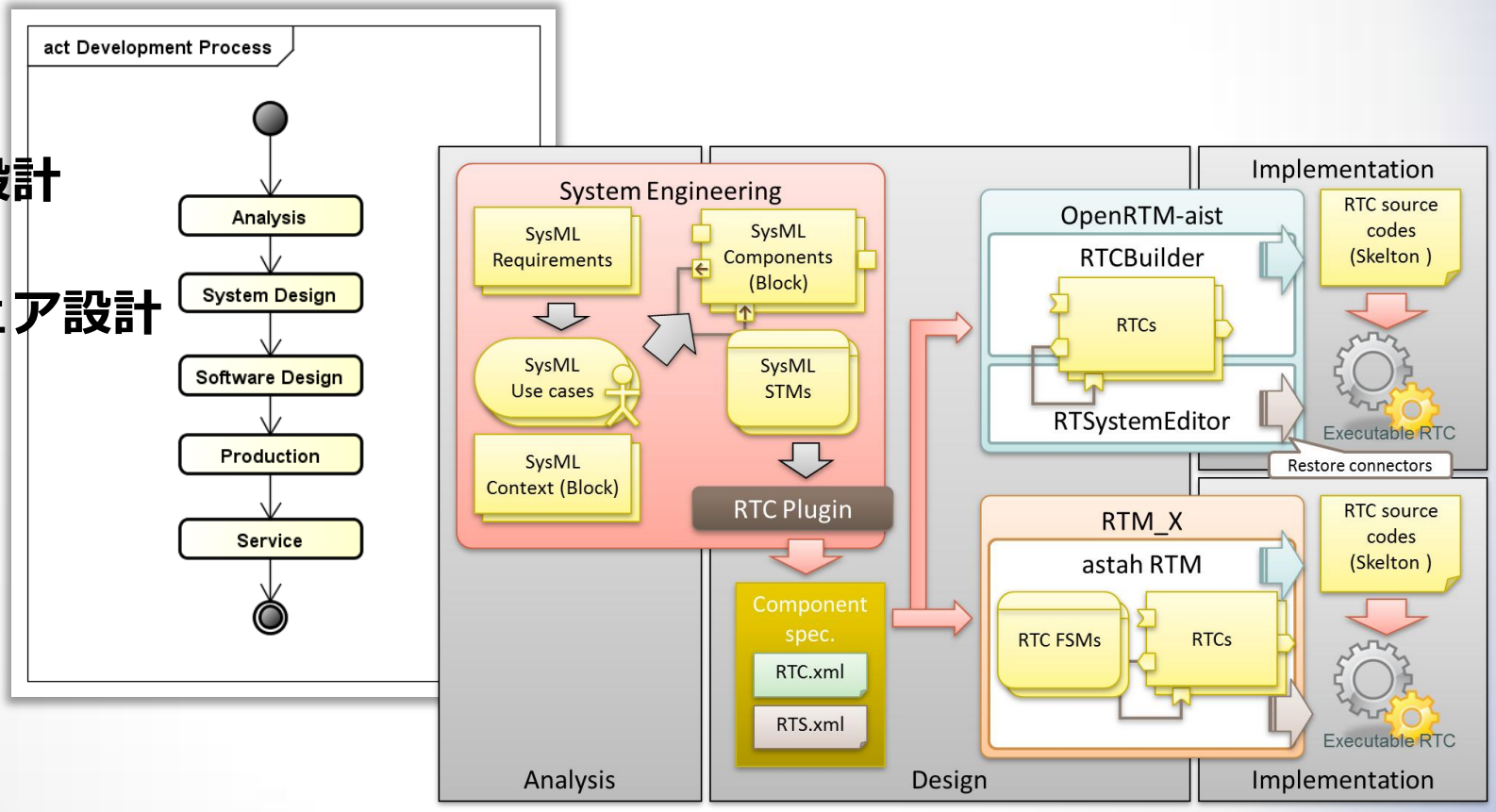
RTC : Robotics Technology Component



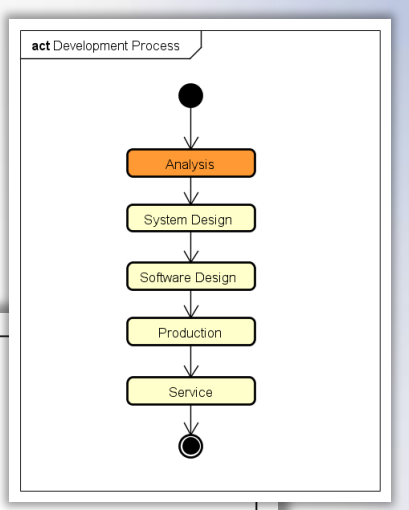
開発プロセスと成果物

各開発フェーズごとに成果物を示しながら、
その成果物とD-Caseとの連携について考察する。

分析
⇩
システム設計
⇩
ソフトウェア設計
⇩
製造
⇩
運用



開発プロセスと成果物　～分析フェーズ～



プロジェクトコンテキストの定義

⇩

ステークホルダの特定

⇩

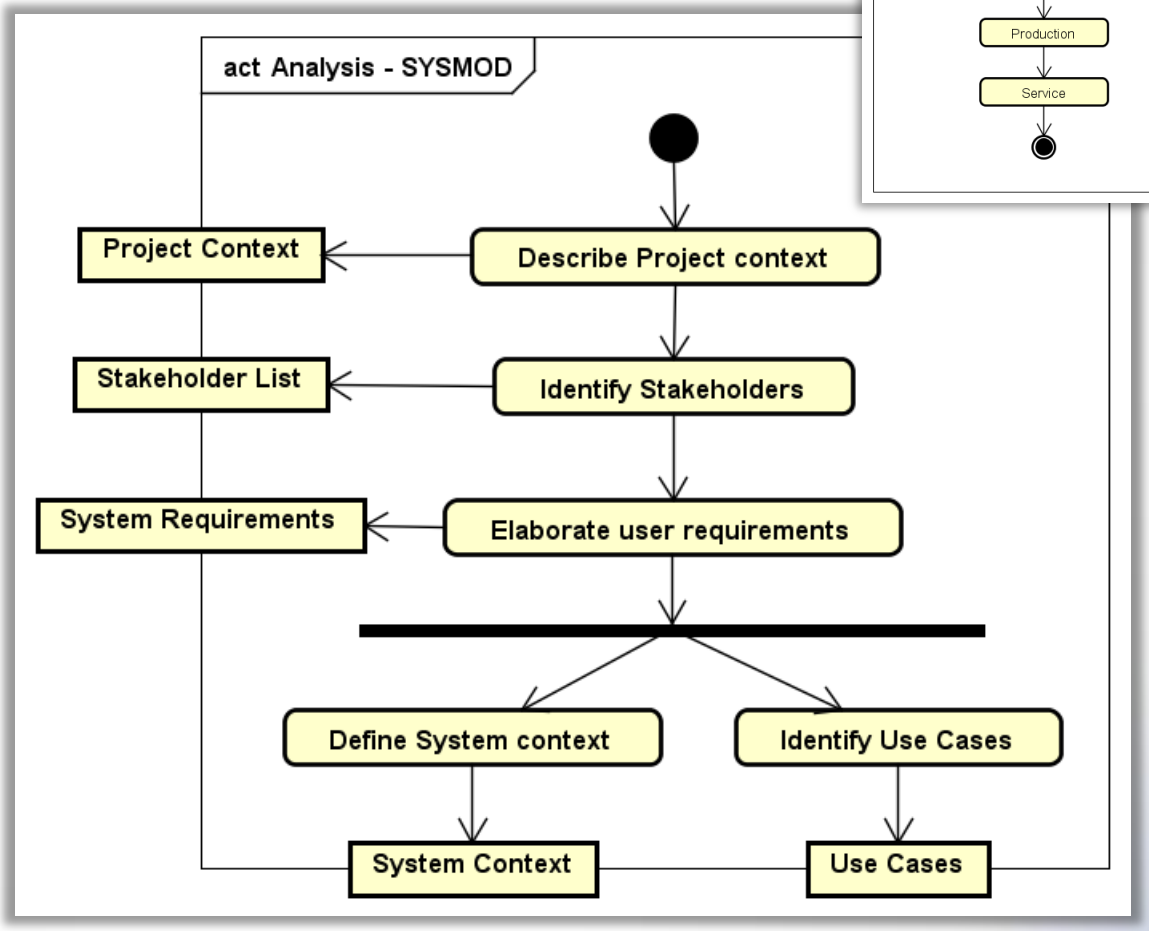
要求分析

⇩

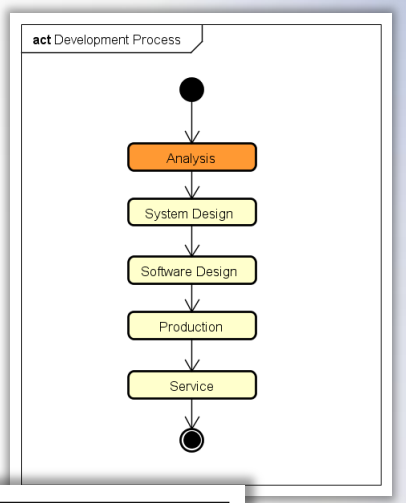
システムコンテキストの定義

⇩

ユースケースの特定



開発プロセスと成果物　～分析フェーズ～

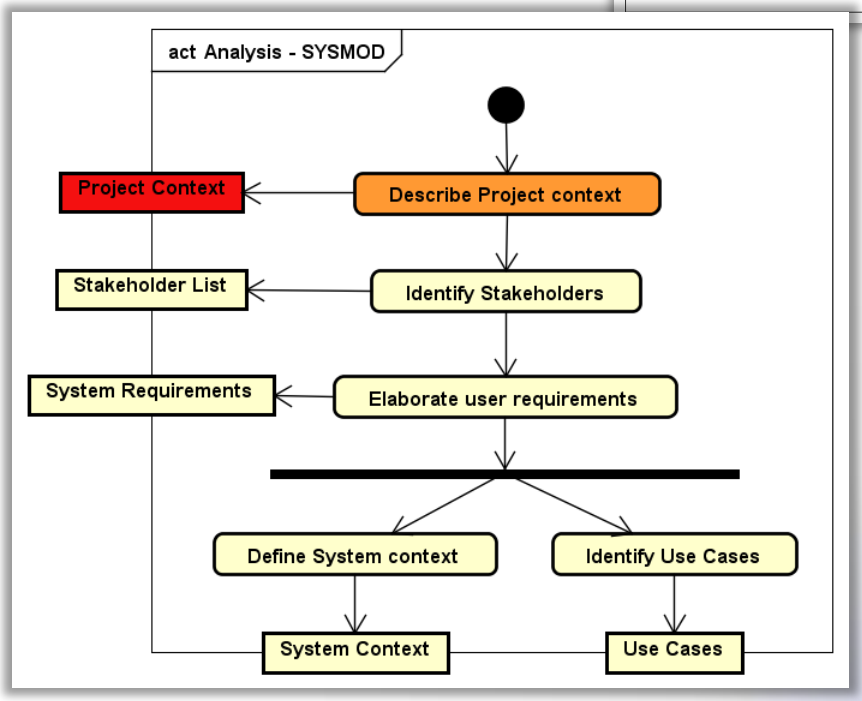


□プロジェクトコンテキストの定義

プロジェクトの目的及び構想を明確にし、
プロジェクトメンバー並びにプロジェクトに関与する者が、共通の目標に向かう動機付けを行うために定める。

■ 成果物

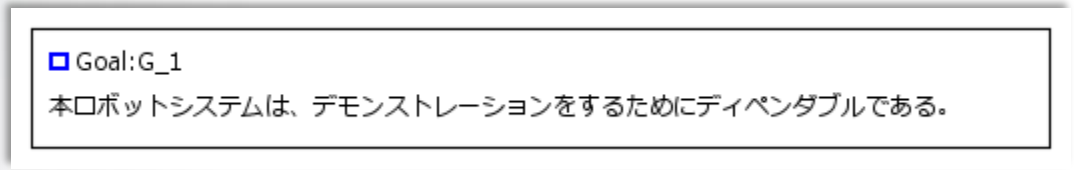
プロジェクトコンテキスト



プロジェクトコンテキストの定義とD-Caseの連携

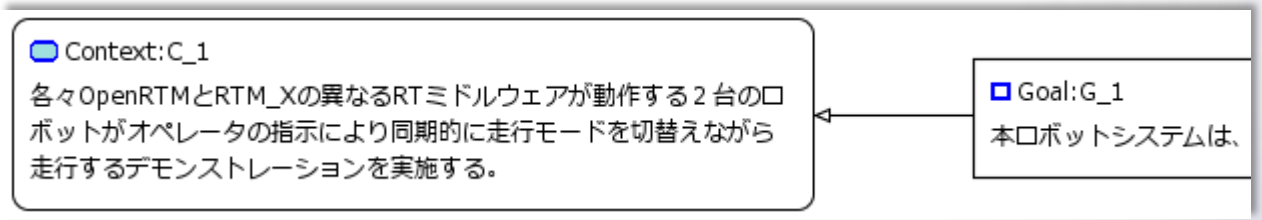
■ プロジェクトコンテキスト

『本システム開発は、SysMLを用いたモデルベースシステムエンジニアリングの実例、並びに、異なるRTミドルウェアにおいて同一のモデルから作成されたRTコンポーネントが動作することをデモンストレーションにより示すためのシステムを開発することを目的とする。』



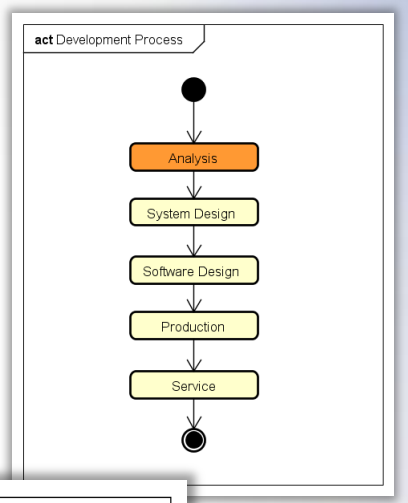
デモを実施するために十分であればよいことを示している。

システムの概要を示すために引用できる。



プロジェクトコンテキストには、システムの目的やトップゴールに示すべきエッセンスが含まれている。

開発プロセスと成果物　～分析フェーズ～

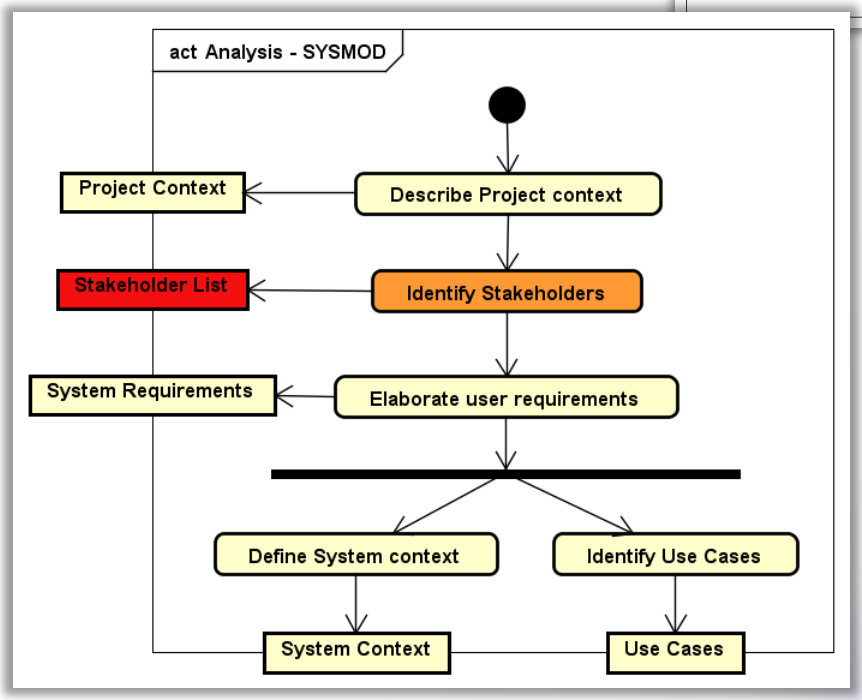


□ステークホルダの特定

システムに関心があり、システム要件の源となる要求や情報を持っている個人または組織を特定する。

■ 成果物

ステークホルダリスト
(ユースケース図)



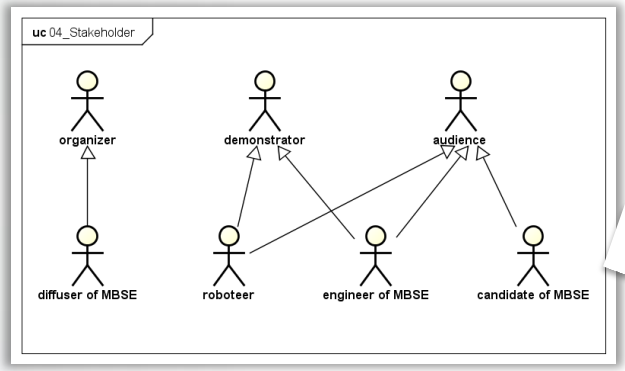
ステークホルダの特定とD-Caseの連携

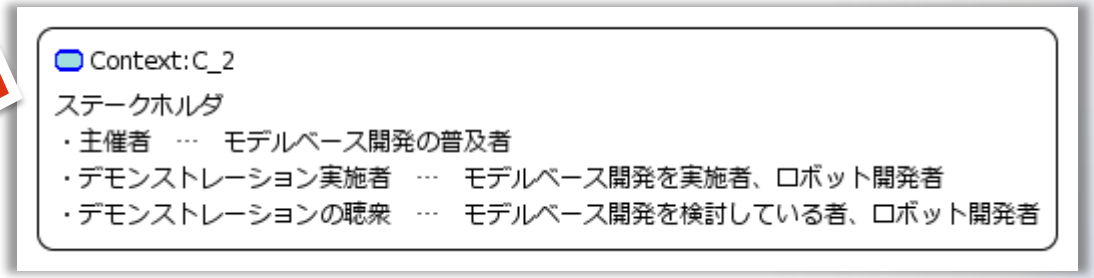
■ ステークホルダ

主催者　　：SysMLによるMBSEの普及者(OMG)

デモ実施者：SysMLによるMBSEの実施者、ロボット開発者

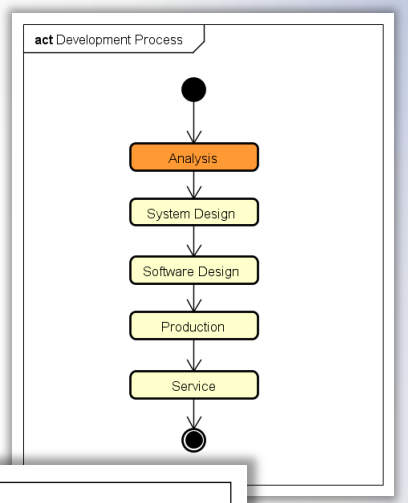
デモ聴衆　：SysMLによるMBSEの予備軍、ロボット開発者





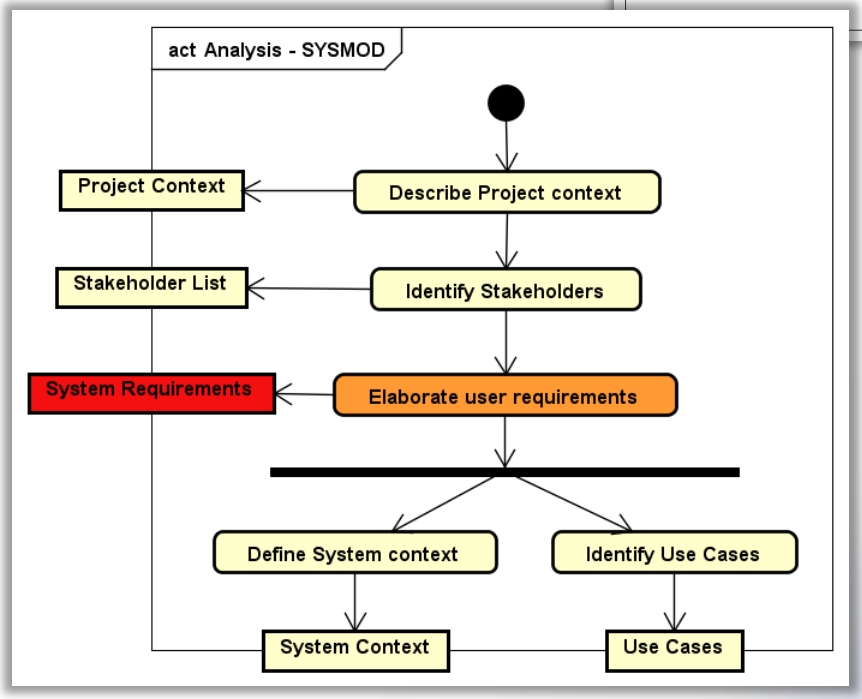
ステークホルダを明確にすることは、D-Caseにとっても不可欠であり、トップゴールのコンテキストへそのまま引用できる。

開発プロセスと成果物　～分析フェーズ～



□要求分析

ステークホルダから、機能要求、非機能要求を収集するとともに、各要求について分析し、システムが満たすべき要件を抽出する。



■ 成果物

SysML：要求図、
　　要求テーブル

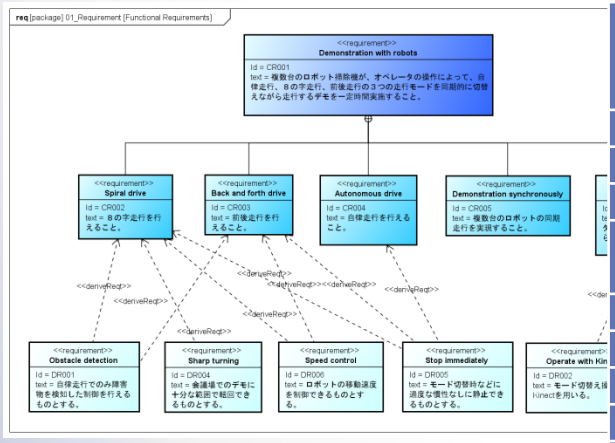
要求分析の成果物

■ 機能要求

デモを実施するために必要となる主要な機能について要求を収集。

- デモを行うロボットとして、制御用APIが公開されているロボット掃除機Roomba®を利用すること。
- オペレータからの指示により、自律走行、８の字走行、前後走行の３つの走行モードを同期的に切替えながら走行するデモを一定時間実施すること。

さらに、分析を進め、速度を調整できることなどの派生する要求を抽出。



No.	ID	名前	テキスト
1	CR001	Demonstration with robots	複数台のロボット掃除機が、オペレータの操作によって、自律走行、８の字走行、前後走行の３つの走行モードを同期的に切替えながら走行するデモを一定時間実施すること。
2	CR002	Spiral drive	８の字走行を行えること。
3	CR003	Back and forth drive	前後走行を行えること。
4	CR004	Autonomous drive	自律走行を行えること。
5	CR005	Demonstration synchronously	複数台のロボットの同期走行を実現すること。
6	CR006	Operator control	走行モードはオペレータの操作によって切り替えられること。
7	DR001	Obstacle detection	自律走行でのみ障害物を検知した制御を行えるものとする。
8	DR002	Operate with Kinect	モード切替え操作にはKinectを用いる。
9	DR003	Wireless control	ロボットは無線で制御する。
10	DR004	Sharp turning	会議場でのデモに十分な範囲で転回できるものとする。
11	DR005	Stop immediately	モード切替時などに過度な慣性なしに静止できるものとする。
12	DR006	Speed control	ロボットの移動速度を制御できるものとする。

要求分析の成果物

■ 非機能要求

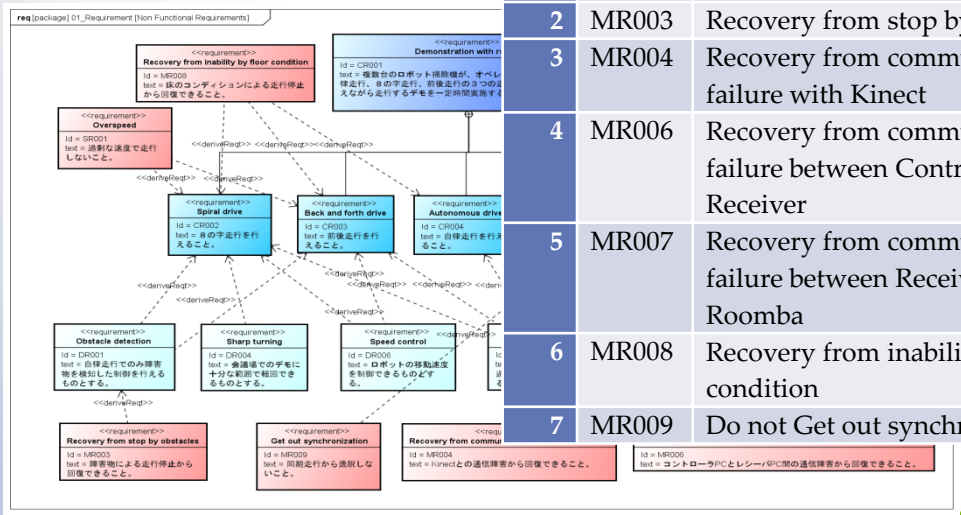
◆ 安全性要求

『システムがユーザと環境に破滅的な事態を生じさせないこと。』と定義。

No.	ID	名前	テキスト
1	SR001	Overspeed	過剰な速度で走行しないこと。

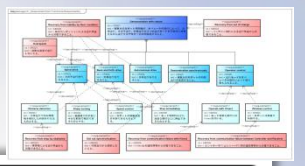
◆ 保守性要求

『システムの状態を監視し、故障を未然に防止するために行うことができること、及び、故障発生から回復することができること。』と定義。

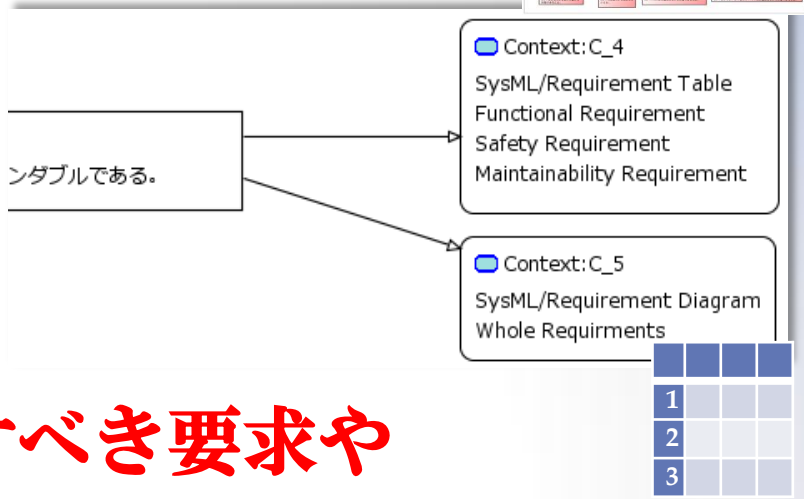


No	ID	名前	テキスト
1	MR002	Recovery from out of charge	バッテリー切れによる走行停止から回復できること。
2	MR003	Recovery from stop by obstacles	障害物による走行停止から回復できること。
3	MR004	Recovery from communication failure with Kinect	Kinectとの通信障害から回復できること。
4	MR006	Recovery from communication failure between Controller and Receiver	コントローラPCとレシーバPC間の通信障害から回復できること。
5	MR007	Recovery from communication failure between Receiver and Roomba	レシーバPCとRoomba間の通信障害から回復できること。
6	MR008	Recovery from inability by floor condition	床のコンディションによる走行停止から回復できること。
7	MR009	Do not Get out synchronization	同期走行から逸脱しないこと。

要求分析とD-Caseの連携

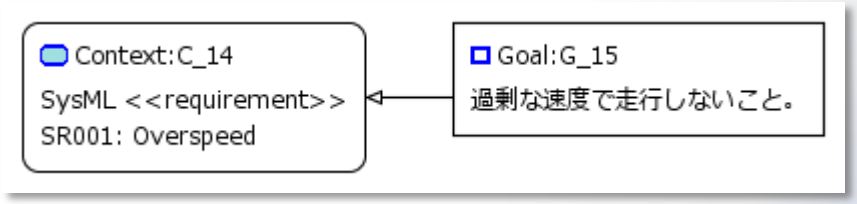


トップゴールのコンテキストに、
システム全体に関するSysMLの要求図及び
要求テーブルを関連付け、これを参照する
ことにより、システムが満たすべき要求を
知ることができることを示した。



ゴールを達成するために満たすべき要求や
どの要求とリンクしているかを明示することができる。

下位のゴールについても、関連する要求図や個別の要求をコンテキス
トに関連付けた。これを参照することにより、ゴールが
満たすべき要求を知ることができる
ことを示した。



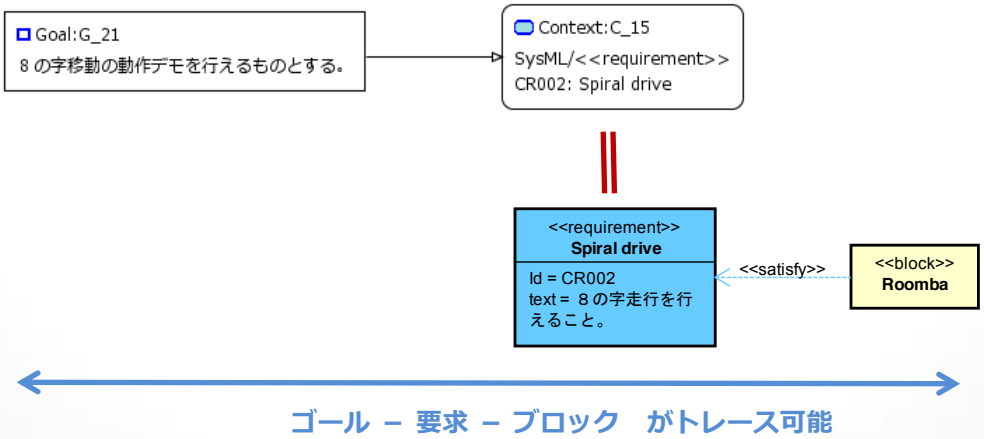
要求として分析されていないゴールはないはず。
最終的にはゴールにはいくつかの要求が関連付けられる。

要求分析とD-Caseの連携

SysMLにおいて、要求とブロック等の要素はトレース可能な関係を表現できる。つまり、例えば、どの要求をどのブロックが満たすかがトレース可能である。

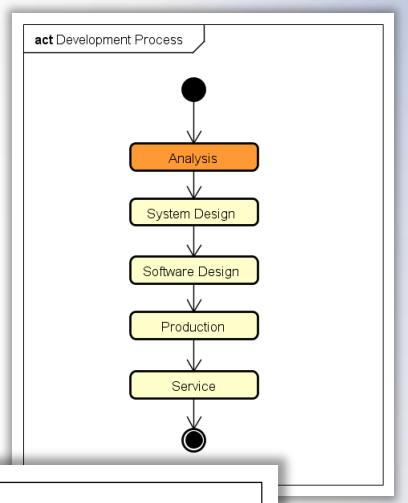
すなわち、ゴールと要求が関連付けられていれば、ゴール、要求、ブロックがトレース可能であることを意味する。

ゴール～要求～ブロックの
トレーサビリティが確保できる。



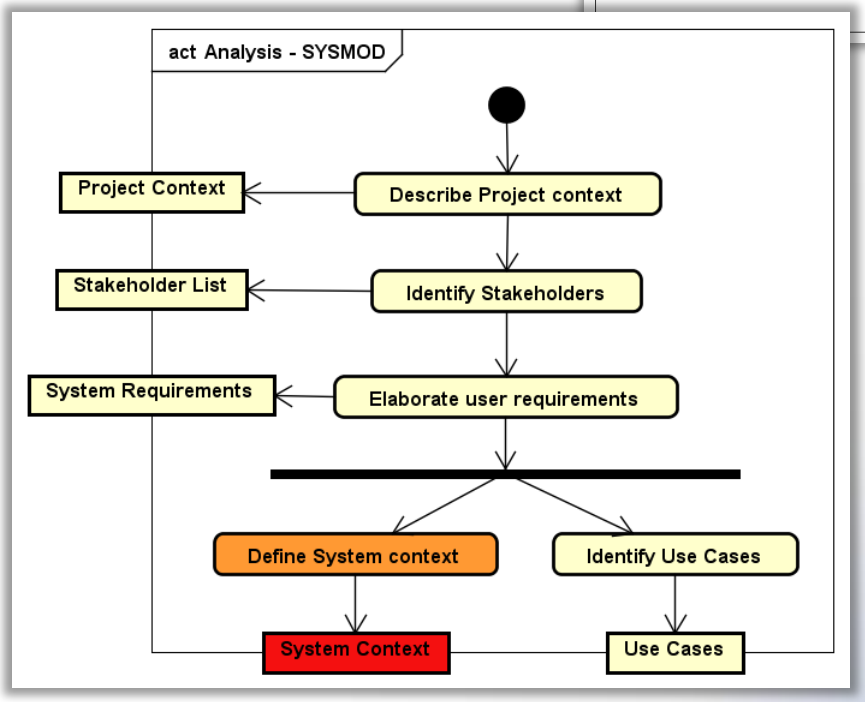
ChangeVision

開発プロセスと成果物　～分析フェーズ～



□システムコンテキストの定義

システムと相互作用するアクターや環境について
明らかにする。

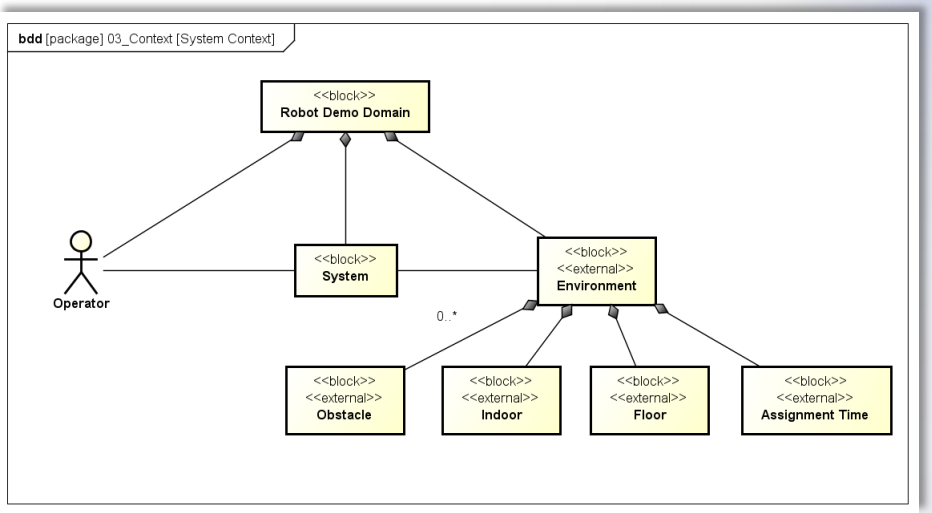


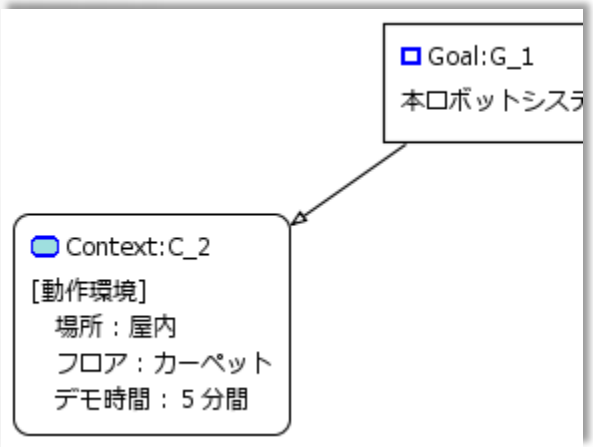
■ 成果物

システムコンテキスト
(SysML：ブロック定義図)

システムコンテキストの定義とD-Caseの連携

アクターとしてオペレータのみが存在し、
動作環境として場所や時間、障害物
といった要素が関わることが
明らかになった。





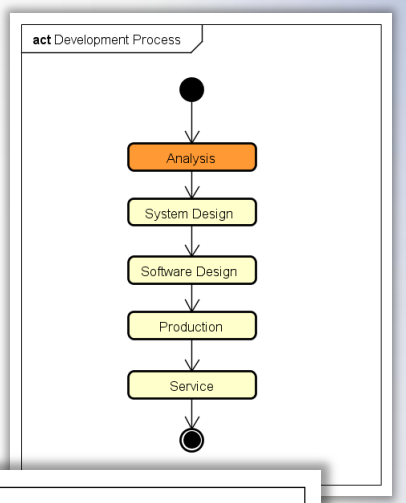
トップゴールのコンテキストへ引用

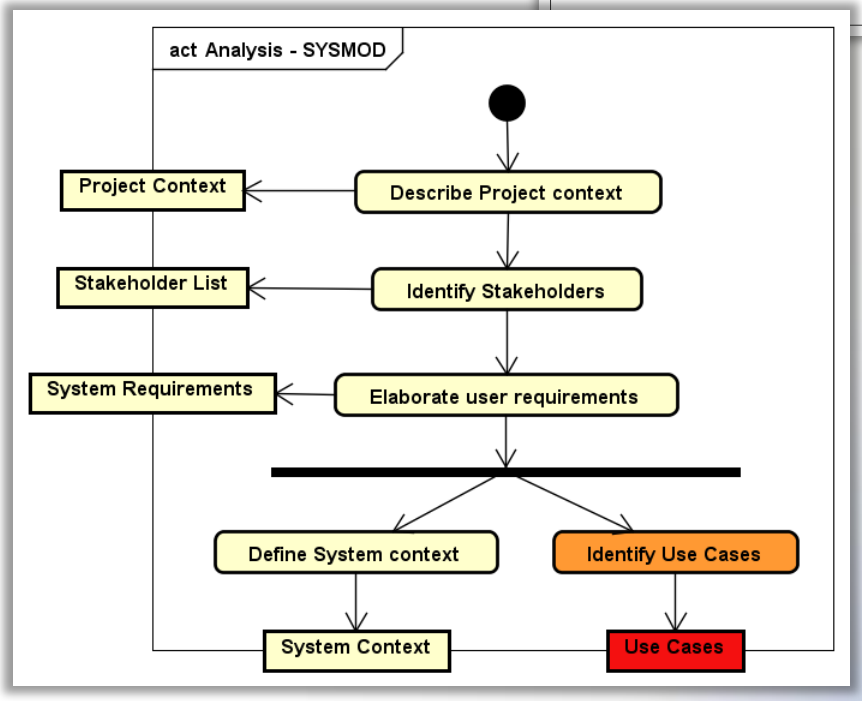
システムコンテキストでは、相互作用するアクターやシステムが置かれる環境が明らかになる。特に環境に関する情報は、ディペンダビリティを立証する上で重要なコンテキストである。

開発プロセスと成果物　～分析フェーズ～

□ユースケースの特定

ユースケースとは、システムによって提供されるサービスを表現するものである。分析した要求の中でも重要なものを実現するものであり、システムの意義と目的を決定づけるものである。
さらに、ユースケース記述を作成することで、アクターとシステムの相互作用や処理の流れについて明らかにする。





■ 成果物

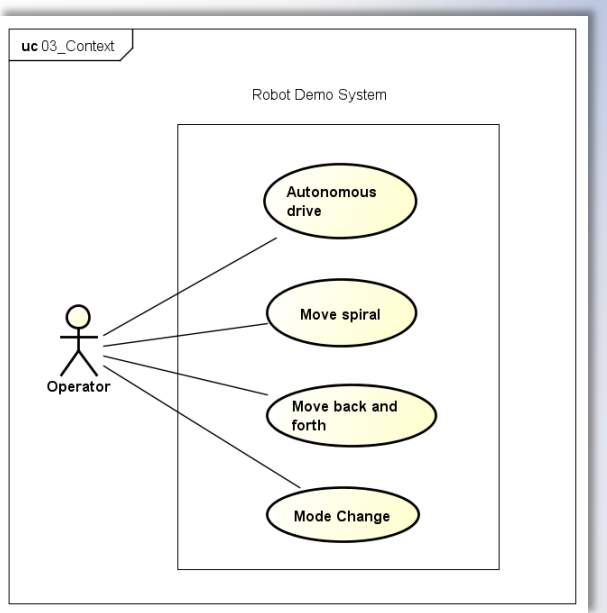
SysML:

ユースケース図、

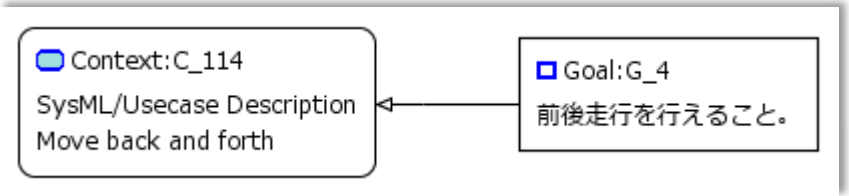
ユースケース記述

ユースケースの特定とD-Caseの連携

自律走行、８の字走行、前後走行といった走行機能と、走行モードの切り替えを司るユースケースがあることを特定した。



「前後走行を行えること」というゴールに対して、前後走行のユースケース記述をコンテキストとして関連付けた。
これを参照することで、システムが実現する前後走行のシーケンスを具体的に知ることができる。



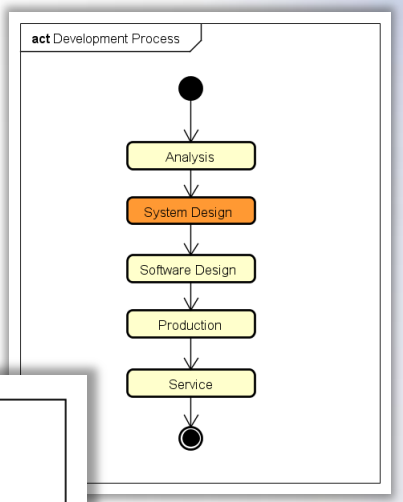
ユースケースは、システムが提供するサービスであるため、機能について立証する場面において、ユースケースやユースケース記述をコンテキストとして関連付けることで、それを参照すれば、対象となる機能について把握することが可能となる。

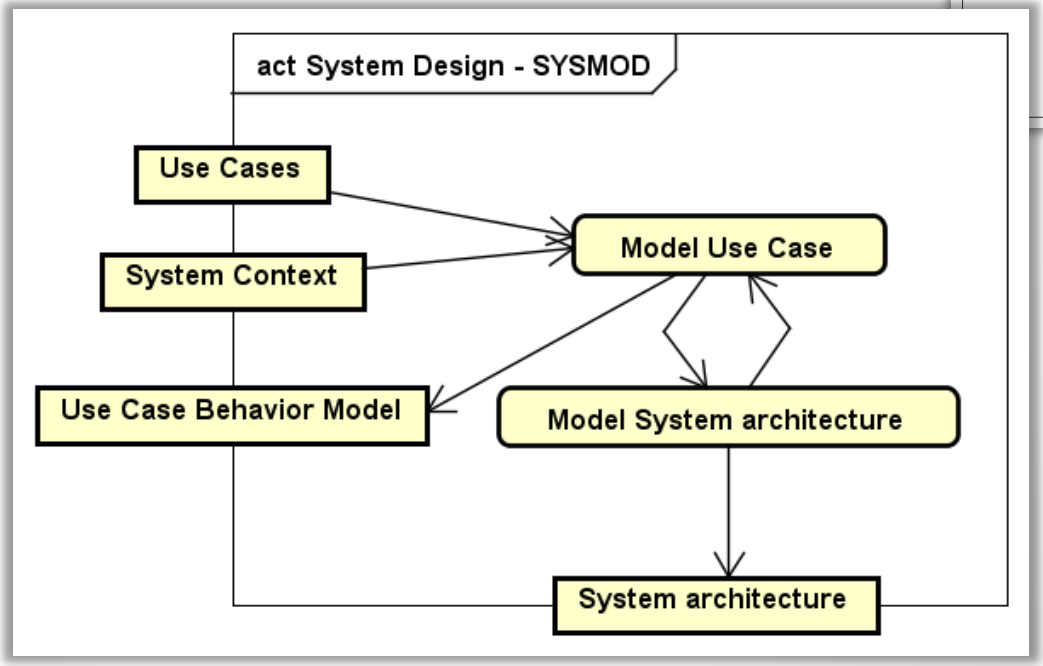
開発プロセスと成果物　～システム設計フェーズ～

ユースケース分析

⇩

システムアーキテクチャ設計

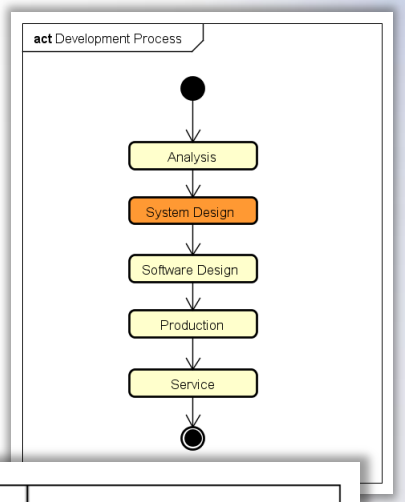




開発プロセスと成果物　～システム設計フェーズ～

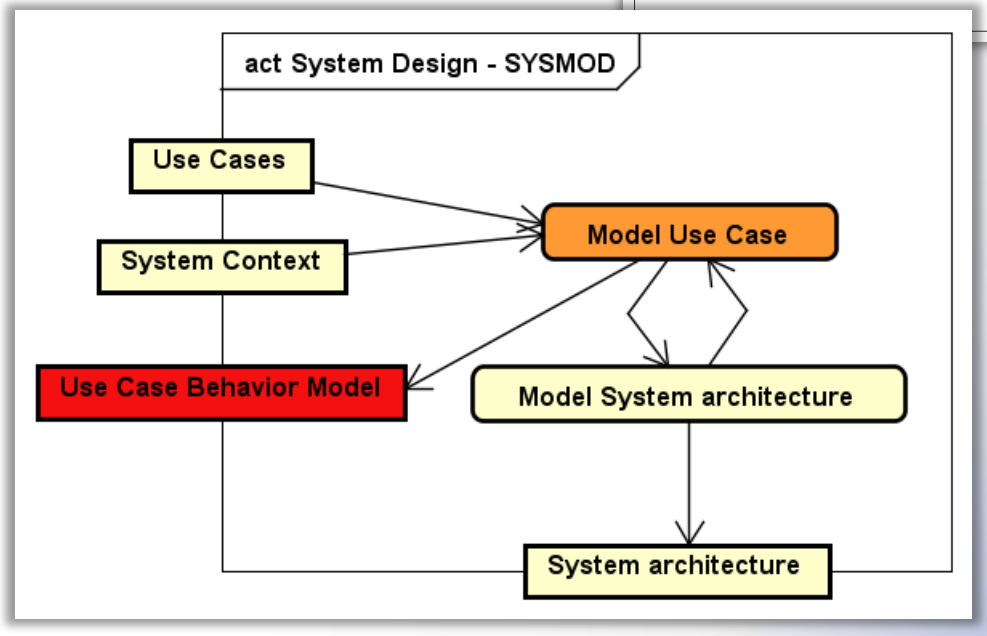
ロユースケース分析

分析フェーズで特定されたユースケースを実現するために、アクティビティ図やシーケンス図を用いて、ユースケースフローやオブジェクトフローをモデリングしながら、システムが内部で実行する処理を詳細化し、処理の入出力や処理の順序を明らかにする。



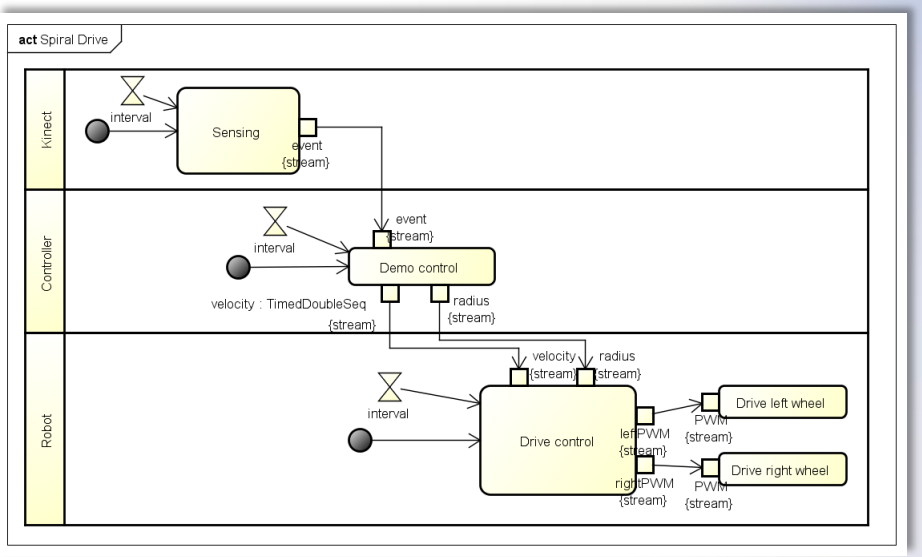
■ 成果物

SysML:
ステートマシン図、
アクティビティ図、
シーケンス図などの振舞い図

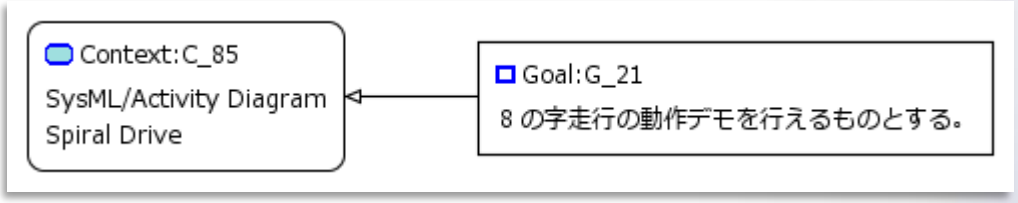


ユースケース分析とD-Caseの連携

８の字走行の振舞いを表現したアクティビティ図。
KinectからイベントがControllerに伝えられ、速度と回転半径をRobotに与えることで、8の字走行を実現することを表現。



「８の字走行の動作を行えること」というゴールに対して、アクティビティ図をコンテキストとして関連付けることで、その振舞いについて知ることができることを示した。



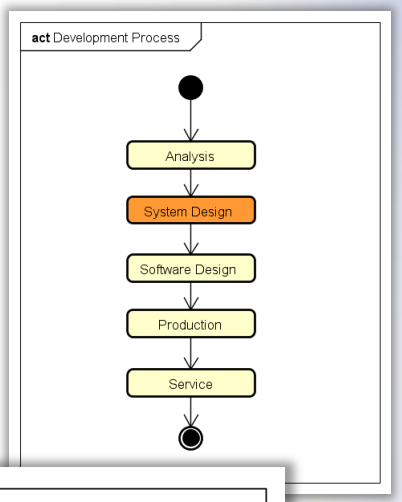
SysMLの振舞い図をコンテキストとして関連付けることで、参照すれば振舞いについて捕捉できることを示すことができる。

ChangeVision

開発プロセスと成果物　～システム設計フェーズ～

ロシステムアーキテクチャ設計

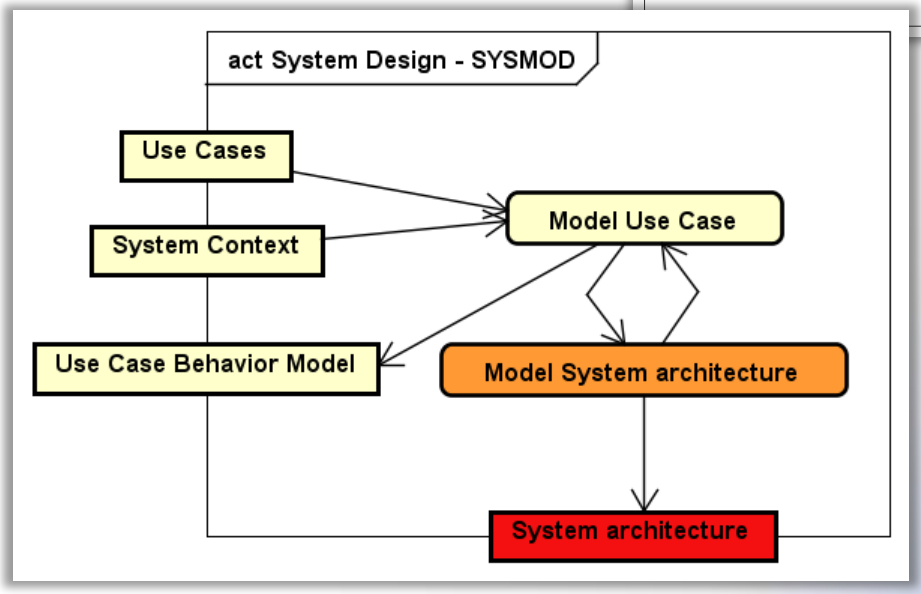
システム設計フェーズで行ったユースケースの分析並びに分析フェーズで分析した要求及びシステムコンテキストの定義を踏まえて、システムがそれらをハードウェアで実現するのか、ソフトウェアで実現するのか及び各々どのような構造で実現すのかを決定する。



■ **成果物**

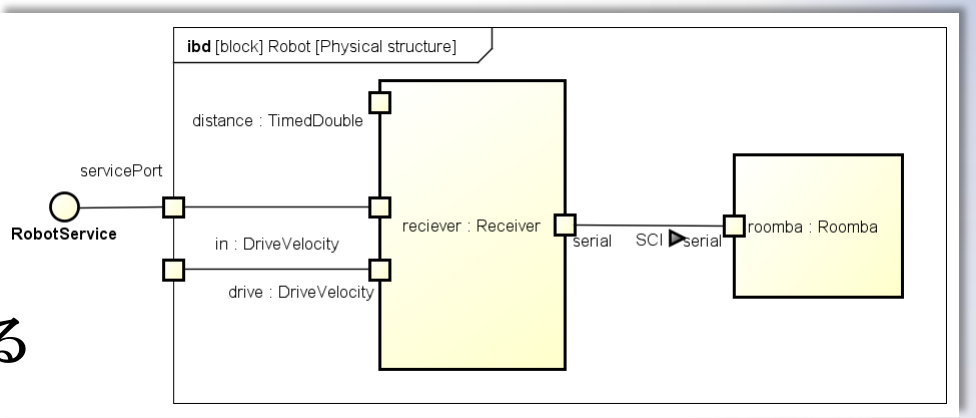
SysML:

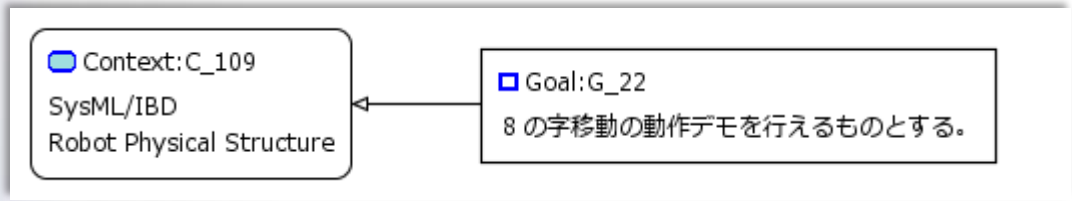
ブロック定義図、
内部ブロック図
などの構造図



システムアーキテクチャ設計とD-Caseの連携

Controllerからの走行指示を受け取るReceiverブロックとそれをRoombaに伝えるRoombaブロックが存在し、Receiverブロックが、RobotServiceというサービスを要求するインタフェースとDriveVelocityという速度を受けるインタフェースを規定していることを表現。





「8の字走行を行えること」というゴールに対して、Robot Physical Structureという内部ブロック図を関連付けることで、これを参照すれば、当該ゴールに関わるブロックの内部構造について理解できることを示した。

SysMLの構造図またはその要素を関連付けることで、参照すれば構造について捕捉できることを示すことができる。

ChangeVision

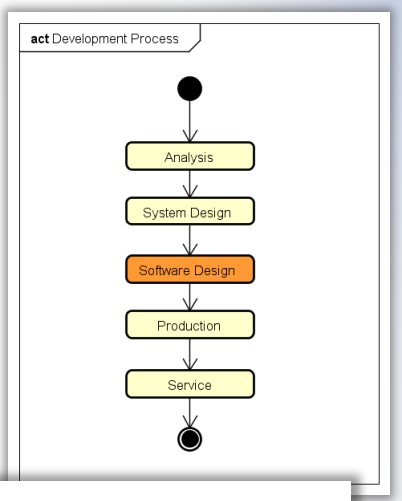
開発プロセスと成果物　～ソフトウェア設計フェーズ～

□構造設計

システムの静的な構造を分析し、モデルで表現する。

■ **成果物**

UML:クラス図、オブジェクト図などの構造図



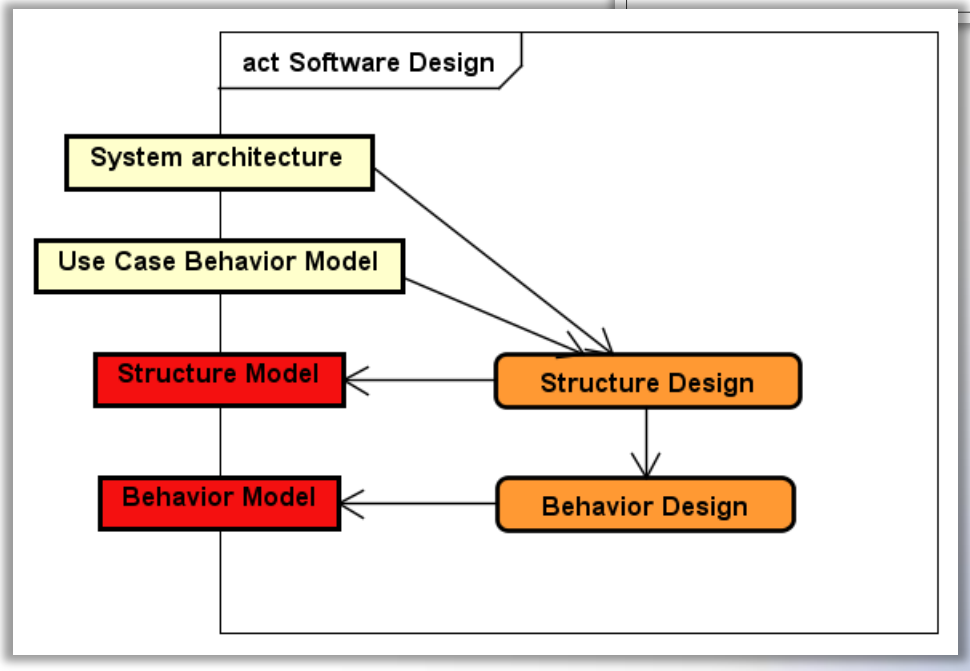
□振舞い設計

システムの振舞いを分析し、
モデルで表現する。

■ **成果物**

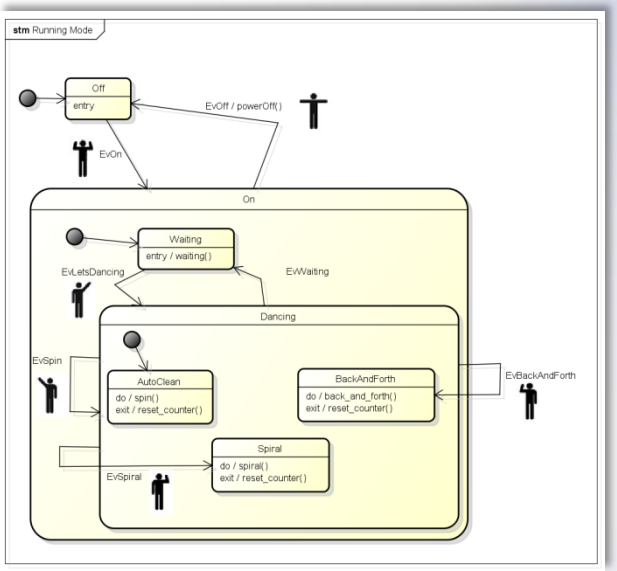
UML:

シーケンス図、
ステートマシン図、
アクティビティ図
などの振舞い図

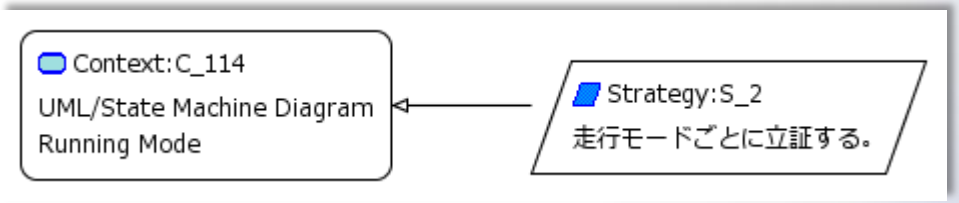


ソフトウェア設計とD-Caseの連携

ソフトウェアの構造や振舞いをモデリングするための言語であるUMLの図や図要素を関連付けることで、参照すれば、ソフトウェアの構造や振舞いを知ることができ、関連する要素の理解を助ける。



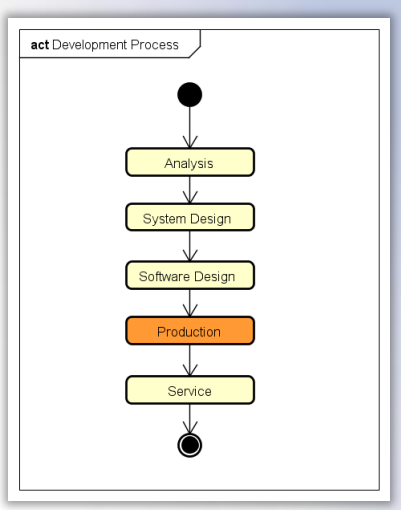
「走行モードごとに立証する」というストラテジに対して、走行モードの状態遷移を表現したステートマシン図を関連付けることで、当該図を参照すれば走行モードについて把握できることを示した。



ソフトウェア設計においても、D-CaseとUMLの連携により、立証対象となるゴールの構造や振舞いの捕捉を促す効果を得られる。

ChangeVision

製造とD-Caseの連携



■ 成果物

ハードウェア　…　部品やパーツ、筐体など
ソフトウェア　…　ソースコードやオブジェクトモジュール
　　ロードモジュール

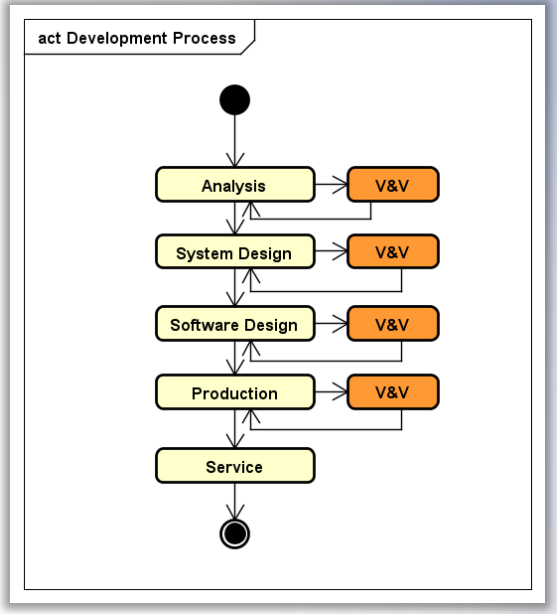
本システムでは、ハードウェアについては、ロボット掃除機をそのまま利用したため、ソフトウェアの実装が製造フェーズで行った作業。

　設計したソフトウェアコンポーネントのモデル
→インタフェースの情報などを抽出したスケルトンコードを生成
→ロジック部分についてコーディング

製造フェーズの成果物もD-Caseと関連付けることが考えられる。

例えばソースコードに関心のあるステークホルダにとっては、ソースコードを示すことに意味がある。

検証・妥当性確認とD-Caseの連携



本システムはデモ用であり、必ずしも実施していないが、本来であれば、各開発フェーズに対応する検証及び妥当性確認の作業が実施されるはずである。

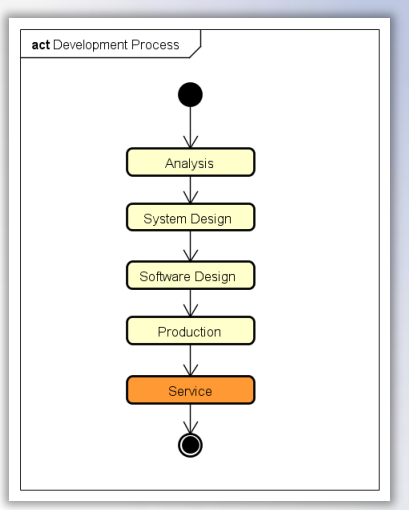
ディペンダビリティの証明には
開発プロセスが妥当に実施されたことを
立証することも必要

検証・妥当性確認の成果物がエビデンスとなる。

- 分析、設計フェーズ
  - ・レビュー結果報告書
  - ・規格に準拠したことを証するドキュメントなど
- 製造フェーズ
  - ・テスト結果報告書、テストコードなど

また、今回の開発には当てはまらなかったが、モデルがシミュレーション可能である場合には、モデル自体が、エビデンスと成り得る。

運用とD-Caseの連携



ディペンダビリティケースは、

「与えられた適用先と、環境において、
システムがディペンダブルであることの確かで、
正しい議論を提供する、エビデンスを元にした
ドキュメント」

と定義されている。Bishop, P. & Bloomfield, R. (1998). A Methodology for Safety Case Development, in Proc. of the 6th Safety-critical Systems Symposium, Birmingham, UK. Feb 1998

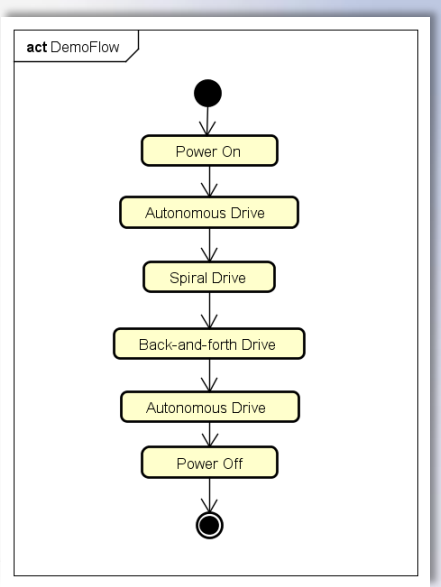
つまり、

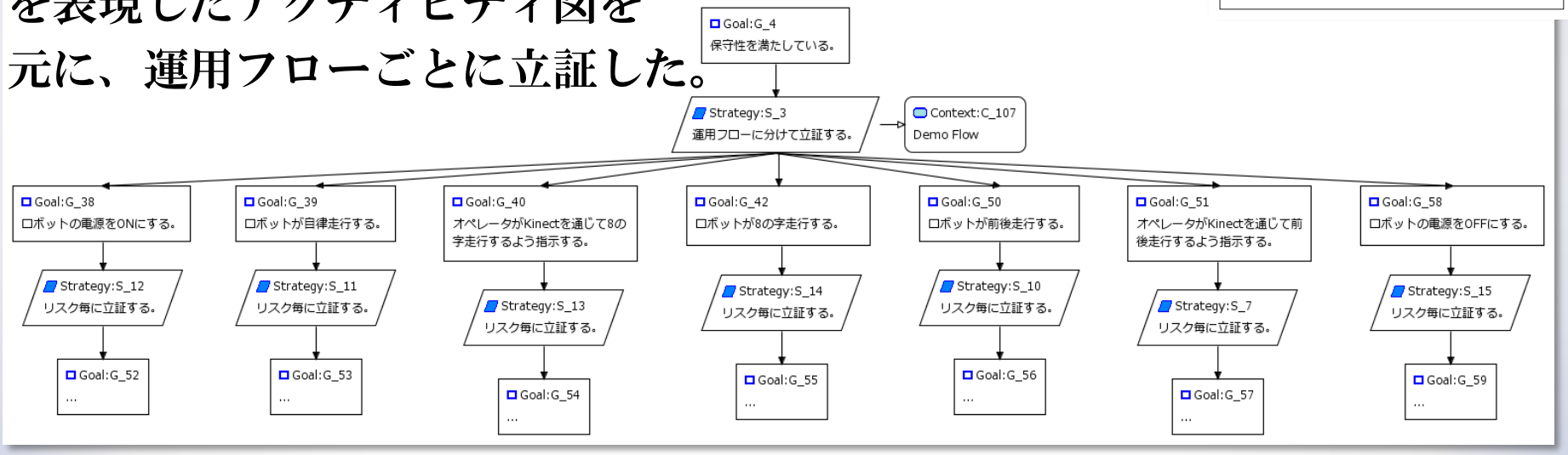
システムが、運用フェーズにおいてディペンダブルであることを立証することが求められている。

運用フェーズとD-Caseの連携

本システムにおける運用フェーズは、デモの実施を指す。
通信異常や走行モードの異常によるデモの実行不能の事態は回避しなければならず、通信内容や走行モードの状態を監視すること、並びに、走行不能からの回復が必要。

運用フェーズのディペンダビリティについて、デモフローを表現したアクティビティ図を
元に、運用フローごとに立証した。





運用フェーズのディペンダビリティは、運用フローごとに立証すると、理解しやすい議論構成となる。

考察

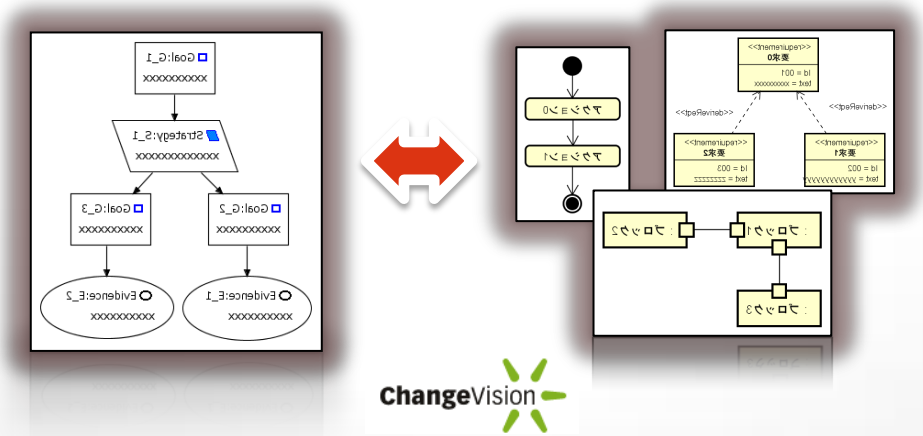
■ モデルの視認性を利用した効果

SysML/UMLのモデルは、システムの構造や振舞いの視認性に優れているため、D-Caseにそれらを関連付け、参照することで、ゴール等が証明しようとしている対象について、理解し、把握することを促すことができる。今回はデモ用でありシステムの規模は小さかったが、規模が大きくなってもこの有益性は変わらない。むしろ規模が大きくなるほど、効果は大きくなるであろう。

■ モデリングの成果物を体系的に管理できる相乗効果

SysML/UMLから見ても、モデリングの成果物を別のドキュメントに貼り付けるなどしなくても、 D-Caseと関連付けられることにより、体系的に管理できるようになるという相乗効果がある。

D-CaseとSysML/UMLによるモデルベース開発の親和性は高い。



D-CaseとSysML/UML連携の構想

今回適用対象としたシステムはデモ用であり、規模が小さく、また、信頼性や秘匿性については求められないものであったが、検証できた有意性については、規模が大きくなっても変わるものではない。

- ただし、**規模が大きくなることによって、作成負荷やメンテナンス負荷が増大することはもちろん、立証する内容や手順が煩雑になることも考えられる。**

- D-Caseの要素とモデルを関連付けることや、モデルの内容を引用することで、ゴールやストラテジを捕捉することができ、有意義であることわかったが、**連携による有益性は単なる捕捉に留まるものではない。**

➡ より効果的な連携をツール利用の観点で構想してみた。

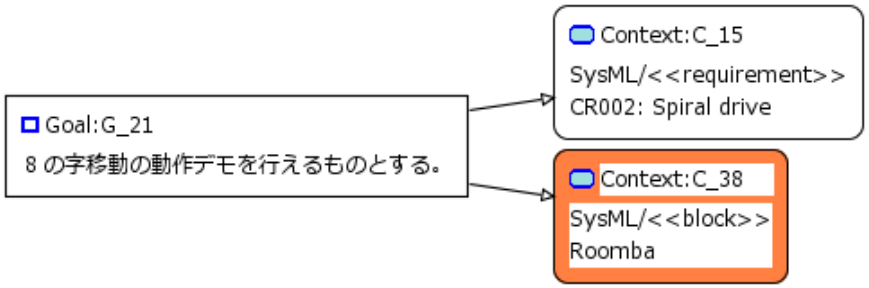
影響分析

D-Caseとモデルを関連付けることは、それらに何らかの関係があることを意味し、すなわち、モデルを変更すれば、そのゴールやストラテジにも影響が及ぶ可能性があることを意味する。

例えば、要求を変更した場合、当該要求が関連付けられたゴールについて影響が及ぶ可能性がある。

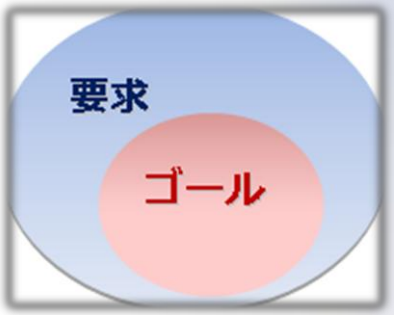
そこで、モデルに変更があった場合に、影響を受けるD-Caseの要素を示す機能が有用であると考えられる。

以下は、Roombaブロックが変更された場合に、それをD-Case上で明示する例である。



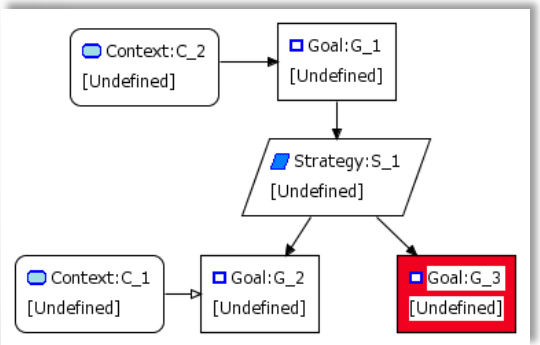
要求の立証カバレッジ分析

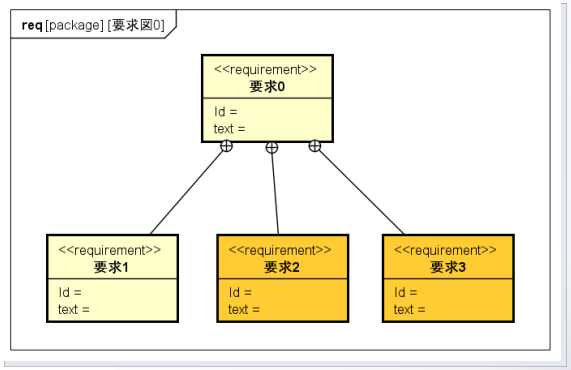
D-Caseにおいて、すべての要求が立証されるものではないが、ゴールとして挙げられている事項については何らかの要求として分析されていなければならないはずである。ゴールに対して要求が関連付けられていない状態は不正な状態である。
また、立証されるべき要求が立証されないことも防がなければならない。



そこで、要求が関連付けられていないゴールや、ゴールに関連付けらていない要求を検出する機能が考えられる。

以下は、要求が関連付けられていないゴールやゴールに関連付けられていない要求をハイライトして明示した例である。





参照容易性

D-Caseと関連付けられたモデルを参照することで、ゴールやストラテジを捕捉することができることは先述の通りであり、立証局面に限らず、作成時の確認作業などでも、モデルを参照する頻度は高いと考えられる。そこで、参照手順を簡略化できると、使用頻度に比例して効果が高くなる。

コンテキストメニューからの選択による方法が一般的？
もっと効率的な方法はないか？

表形式による表示

D-Caseは木構造であるため、トップダウンで見ていくには都合がよいが、一覧性となると決して優れているとは言えない。そこで、表形式で表示、出力できるなど異なるビューで見られることは有用である。

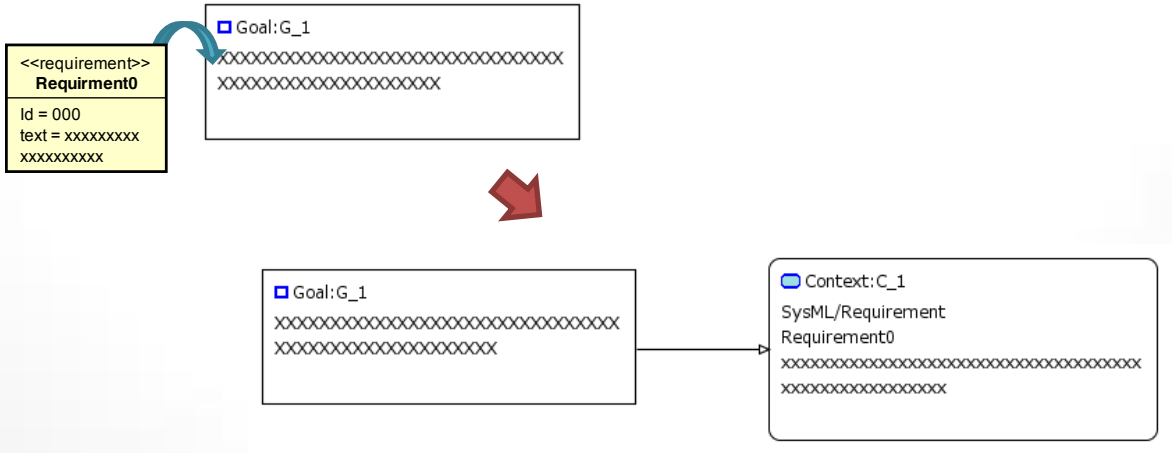
トップゴール	ゴール0	ストラテジ00	ゴール000	エビデンス0001		
			ゴール001	エビデンス0010		
	ゴール1	ストラテジ10	ゴール100	ストラテジ1000	ゴール10000	エビデンス100000
		・・・				
		・・・				

モデルからの展開

モデル作成時に一度入力した情報と同じ内容の文言をD-Caseに記述することは得策ではない。一方に変更が生じれば、他方も変更しなければならなくなるなどまさに負の連鎖である。

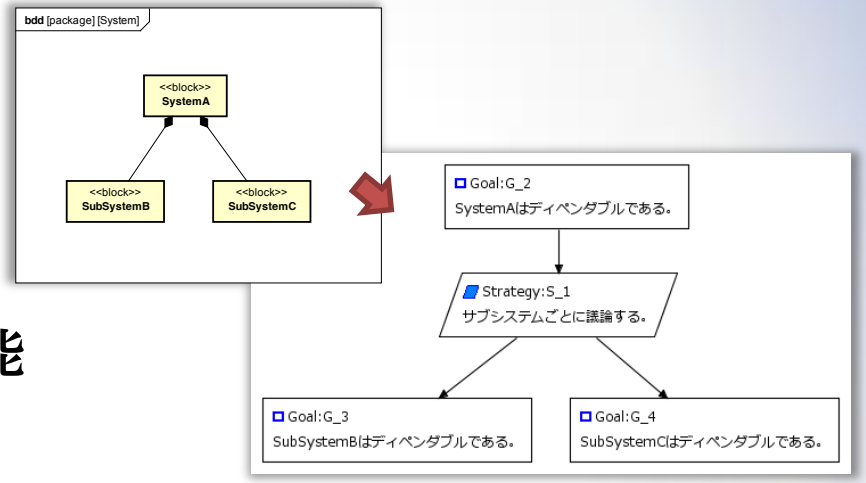
そこで、要求をD-Caseにドラッグ＆ドロップすることで、ゴールが生成されたり、コンテキストになるといった重複入力を不要にする機能は有用である。また、こうしたモデルやD-Caseは、相互に参照関係にあることがわかるため、モデルを編集すれば、D-Caseの要素も変更されるといったことも可能となり、メンテナンスコストを抑えることができる。

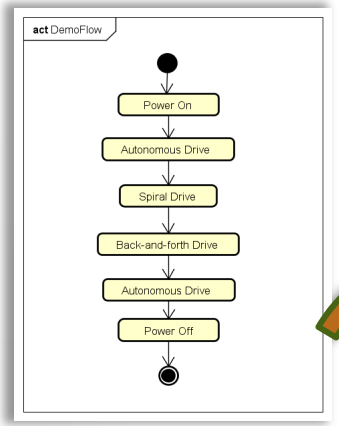
以下は、要求をゴールにD&Dすることで、コンテキストに要求が関連付けられる例である。



モデル構造の展開

構造が階層化されているモデルについて、
D-Caseにドラッグ&ドロップすることで、
階層構造を展開したD-Caseを作成する機能





アクティビティ図などのフローをD-Caseに
ドラッグ&ドロップすることで、各アクション
を展開したD-Caseを作成する機能。

ChangeVision

２．実証実験におけるD-Caseの作成

□ 作成したD-Caseの規模

総作成時間	約100時間
Goal	26
Strategy	11
Context	52
Evidence	9
Monitor	3

今回対象としたシステムは、デモ用であり、規模は小さく、信頼性もそれほど必要とされるものではなかった。

D-Caseを始めとするアシュアランスケースについての知見が無い状態からのスタートとしても、

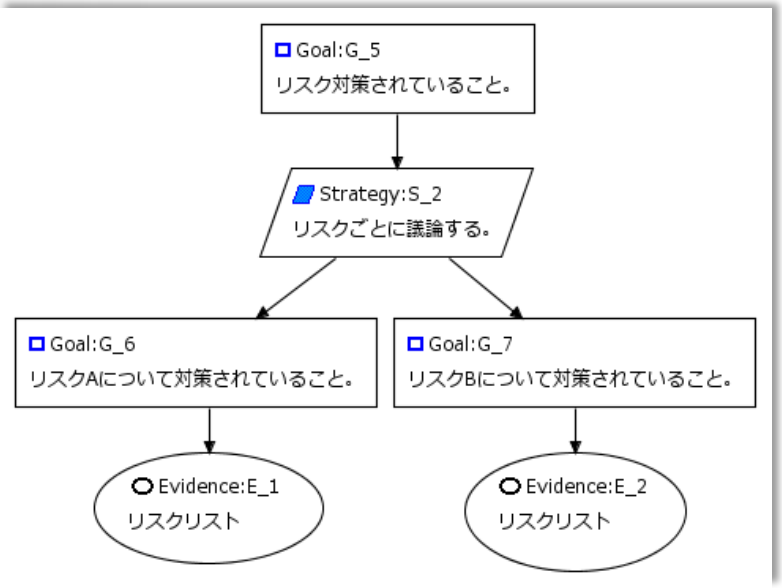
規模の割りには、作成コストを要したという印象。

∵どのように書くべきか都度試行錯誤を要したことが原因。

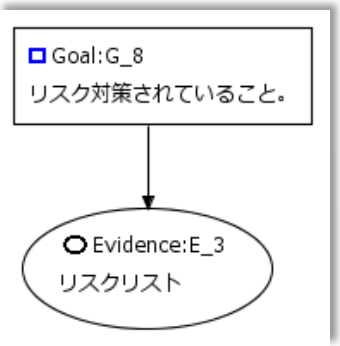
記載する粒度および重複記載について

コンテキストやエビデンスの内容は、参照先にすべて記載されている。
D-Caseに別の成果物に記載されているのと同様の内容を記載することは、作成コスト、保守コストの面で望ましくない。
立証責任を参照先のドキュメントに一切委譲することもできるはず。

リスクごとに記載する必要があるか？
記載すると重複記載の問題も生じる。



委譲できるものは委譲した方がよい？



リスクリスト
リスク　対策
リスクA　xxxxxxxxxx
リスクB　xxxxxxxxxx

機能充足性の立証

■ D-Caseは、非機能要求を議論の対象とすることが一般的？

・機能要求の充足性が関心事であることもある。

・機能要求と非機能要求を完全に切り離して考えることにも違和感。

⇨機能要求について立証した方がよい場合がある。

■ ディペンダビリティ属性ごとに立証する構造をとった場合、機能要求の立証の位置づけは？

・ディペンダブル＝ディペンダビリティ属性＋機能要求

　ディペンダブル属性を満たすだけではディペンダブルであるとはいえないという立場をとることを意味する。

・ディペンダブル＝ディペンダビリティ属性(機能要求を含む)

　今回は、可用性に機能充足性を含むというこちらの立場をとった。

⇨全体を通して立証する内容に違いはなく、概念的な問題なので拘るべきテーマではないかもしれないが、

指針を示しておけば、余計な迷いが生じることは避けられる。

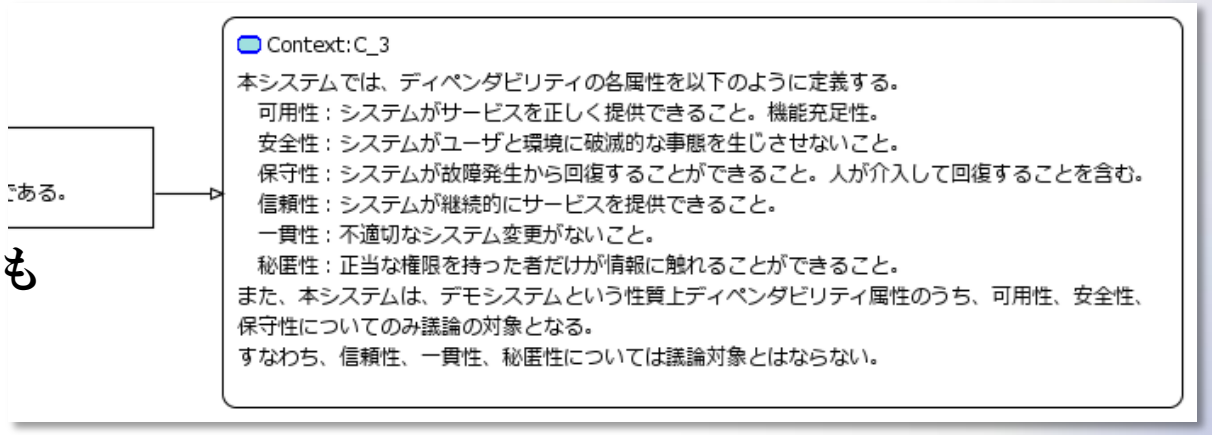
ディペンダビリティ属性の定義

ディペンダビリティ属性ごとに立証する構造をとったときに、各ゴールをどの属性に分類するかを決めることが難しかった。
何らかの規格に準拠することがよい方法だが、規格自体に曖昧さがないわけではない。

スコープ、コンテキストにおけるディペンダビリティ属性の定義を明示することが必要。

システムの性質によっては、すべての属性について立証する必要はない。

デモ用のシステムという性質上、必ずしもシステムが継続的に運用される必要はないし、一貫性や秘匿性も要求されない。



立証すべき属性を明確にすることが必要。

公開されているサンプルが少ない

■ 良いサンプルを多く公開して欲しい

　扱うテーマの性質上、そのまま公開することは難しいドキュメントであろうが、普及のためには、良いサンプルが多く公開されることが重要。今回、初めてD-Caseを作成したわけだが、参考にできるサンプルがあまり存在しないことや、サンプルがあっても、それが良い例なのか、良い例であれば、どういうところが良いのかが不明であったため、どういったレベルで記載すればよいのかがわからなかった。

テンプレート化

■ 規格準拠のテンプレートが望ましい

　議論の分割方法や対象となるシステムの種類、立証すべき内容などによりテンプレート化できれば、D-Case導入の敷居を下げることや、作成効率を高めるといった効果が期待できると考える。将来的には、規格に準拠することを求められるシステムにとっては、対応したテンプレートが用意されることによって、それに従って書くことで、必要な立証事項を満たすことができるといったチェックリストのような機能を果たせることが、成熟した望ましい状態だと思われる。

D-Caseに関する考察

■作成方針を決める際に一定の障壁がある

議論となった事項は、どれも作成方針に関わるものであった。D-Caseについて意見交換を繰り返すも、議論が収束しない状態に陥ることがあった。一方、作成方針さえ決めることができれば、以降はスムーズに書き進めることができることが、その後に議論すべき問題に遭遇しなかったことから言える。

■記載の自由度が高いがゆえに

良い意味では、立証対象となるシステムや立証の相手方の立場によって柔軟に記載することが可能であるが、言い換えれば、作成者によって粒度や観点にばらつきを生じる結果をもたらす。今回行った意見交換においても人それぞれに考え方や意見が異なった。曖昧さが生じないように、作成するD-Caseのスコープにおける定義を明確にする必要がる。

テンプレート化、サンプルの充実、指針の明示などにより解決できる問題

D-Caseの作成コストについては、今回は初めて作成したことや、どのように議論展開すべきかが定まらない状態で作成していたこともあり、システムの規模のわりには時間を要したという印象であるが、この辺りは解決できる問題である。

総まとめ

■ D-CaseとSysML/UMLの連携の親和性は高い

今回適用対象としたシステムはデモ用であり、規模が小さく、また、信頼性や秘匿性については求められないものであったが、検証できた有意性については、規模が大きくなっても変わるものではない。

■ 実運用レベルで活用できるようにすべき

ただし、規模が大きくなることによって、作成負荷やメンテナンス負荷が増大することはもちろん、立証する内容や手順も煩雑になることが考えられる。これらの課題については、先述した連携構想にあるようなサポートによって、実運用レベルで有効に活用できるようにすることが必要である。

■ D-Case作成の敷居をできるだけ下げるべき

D-Caseについては、表記法は難しいものではないが、自由度が高く、実際に書こうとすると行き詰まることがよくある。これについては、書き方の指針、テンプレート化、良いサンプルを示すことで解決できると考える。

近来、システムのディペンダビリティを証明する必要性が益々増している中、本実証実験を行ったことで、立証責任を果たすために、D-CaseとSysMLやUMLのモデルがより効果的に協調することを追及する価値は多いにあるという確信を得ることができた。

FIN