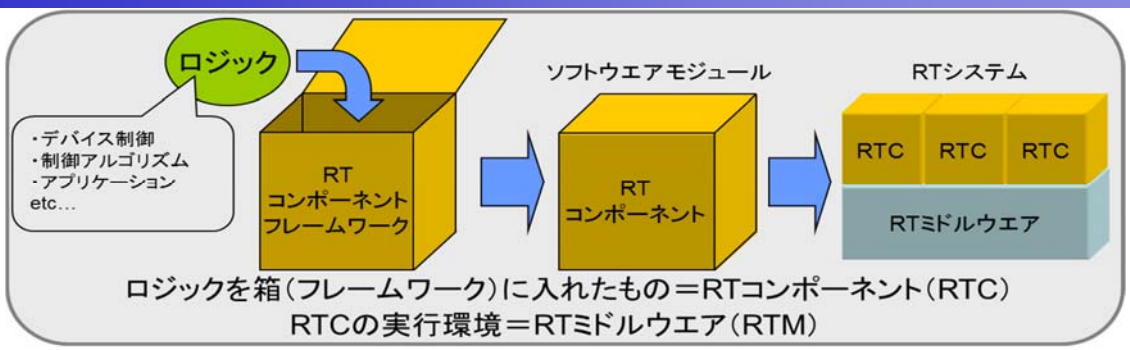
RTミドルウェア SUMMER CAMP 2016

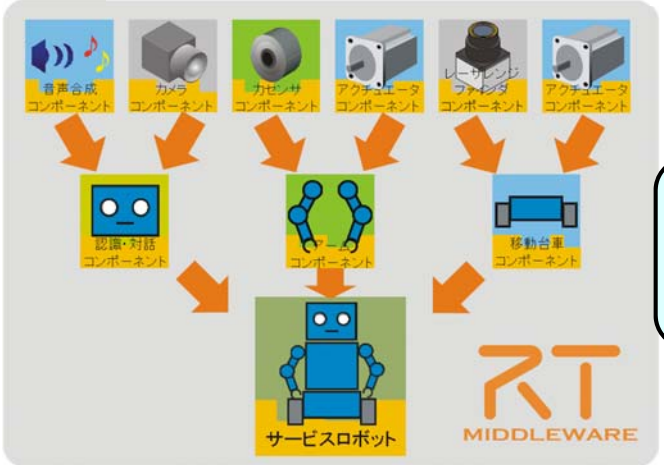
RTコンポーネント開発のための システムモデリング講習会

日時：2016年7月12日(火) 13:00～
場所：早稲田大学 西早稲田キャンパス

株式会社　グローバルアシスト
坂本 武志

RTミドルウェアとRTコンポーネント

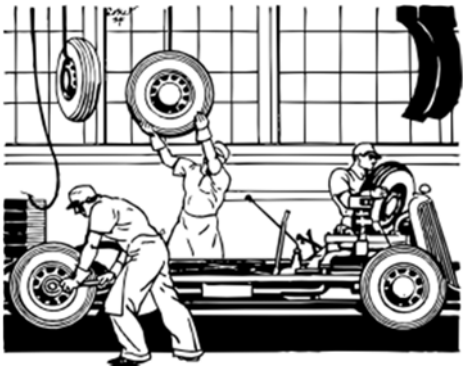




RTミドルウェアは
コンポーネントベース開発(CBD)を
行うための枠組み

コンポーネントベース開発

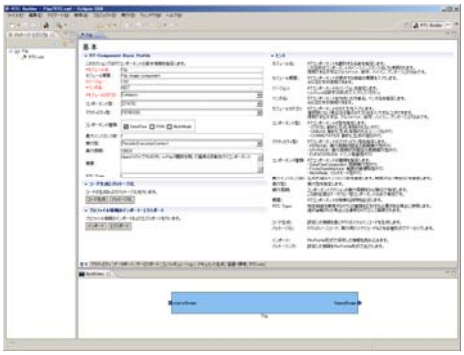
- ソフトウェアを部品(コンポーネント)単位で開発し，複数のコンポーネントを統合することによって，対象システムを開発する手法
  - メリット
    - 変更容易性，保守性の向上
    - 再利用性の向上
  - ソフトウェア以外の分野(機械系，電気系など)では，基本となっている考え方，開発手法



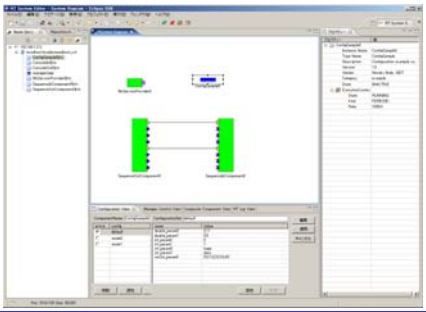
RTシステム開発時のよくある問題

OpenRTM-aistのプロジェクトページなどから利用可能な既存RTCを検索

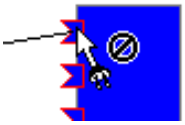
開発対象のシステムに固有で，既存には存在しないRTCを自作



揃えたRTCを使用してRTシステムを構築

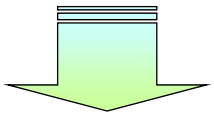


想定したポート間でコネクタの接続ができない



RTシステム開発時のよくある問題

- 同じ「コンポーネントベース開発」なのに，なぜソフトウェアの場合だけ問題が起きるのか？
  - 機械系，電気系のシステムを作成する際に，組み立て時に接続，組み付けが問題になる事はあまりない



- コンポーネントベース開発では，**標準規格(仕様)と設計図面が重要**
  - 機械系，電気系の場合，部品の仕様がJIS,ISOなどで決められている
    - ネジの径，ピッチなど部品の仕様を製作者が勝手に変更することはあまりない
  - 事前に設計図面を作成し，検討を行う
    - 図面上で問題点がないか予め検討を行っているため，実際の組み付け時に問題が発生することは少ない

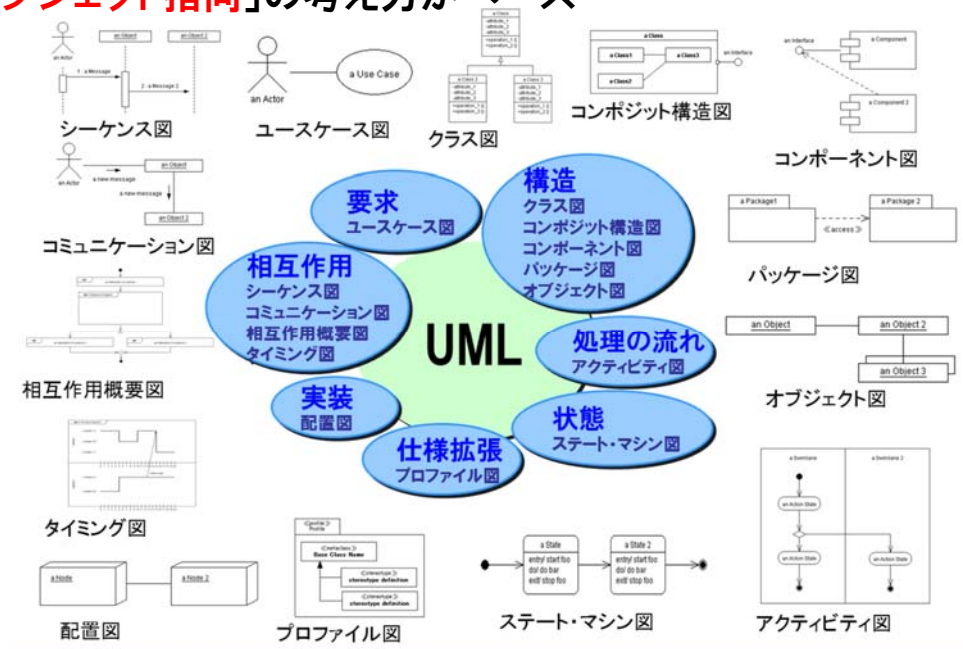
ソフトウェア開発，RTミドルウェアを用いた開発でも
標準規格(仕様)と設計図面は重要

RTコンポーネントの標準規格(仕様)

- 共通インターフェース仕様書
  - 標準的なモジュール構成と，各モジュールの規格を定義
    http://www.openrtm.org/openrtm/ja/project/Recommendation_CommonIF
  - 移動ロボット
    - 移動ロボットを構成する標準的な機能モジュール群
  - コミュニケーション機能
    - 音声によるコミュニケーションを行うための機能モジュール群
  - ロボットアーム
    - ロボットアームを構成する標準的な機能モジュール群．産業用ロボットにも対応
  - 双腕ロボット
    - 双腕ロボットを構成する標準的な機能モジュール群
  - 作業系画像認識
    - 画像認識を行うための機能モジュール群
  - カメラ機能
    - カメラ画像を入出力する機能モジュール群

RTシステムの設計図面

- Unified Modeling Language:統一モデリング言語
  - 「ソフトウェア」の「設計図」を記述するための言語(表記法＋意味)
  - 「オブジェクト指向」の考え方がベース

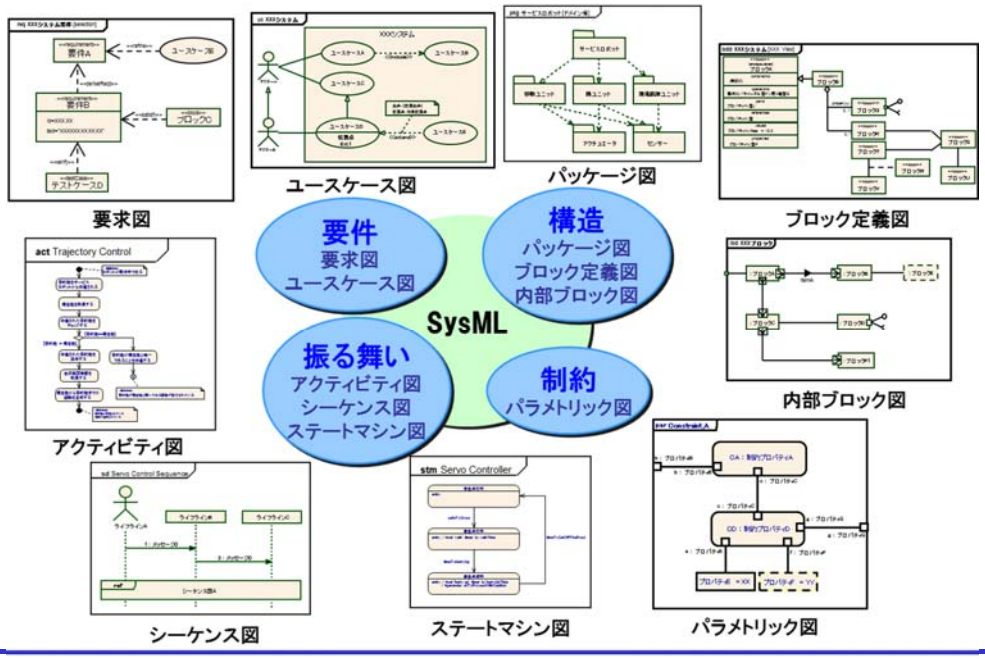


UMLの限界

- 対象が「ソフトウェア」に限定
  - 仕様策定時に想定していたのがソフトウェアのみ
    - ソフトウェアだけではシステム構築ができない
    - ハードウェア，ソフトウェア両者を考慮した設計が必要
- 仕様が複雑
  - ダイアグラム数が14種類
  - 巨大なビジネスアプリから組み込みソフトまで広範囲のカバーを意図
    - 仕様策定者の専門ドメインが多岐に渡っていたため
- 開発全体をカバーできない
  - 上位の要求(特に非機能要求)を表現することができない
    - 機能要求についてはユースケース図にて表現は可能
  - 連続系の表現に対応できない
    - 振る舞いの表現は基本的にイベントベースとなっている

RTシステムの設計図面

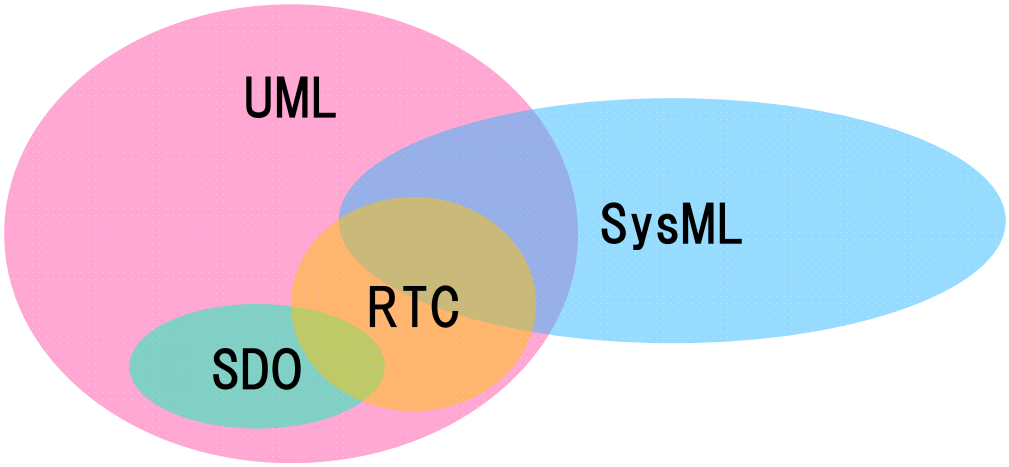
- Systems Modeling Language:システムモデリング言語
  - 「システム全体」の「モデル」を表現する言語(表記法＋意味)である。
  - 「UML」「オブジェクト指向」がベースとなっている。



SysML

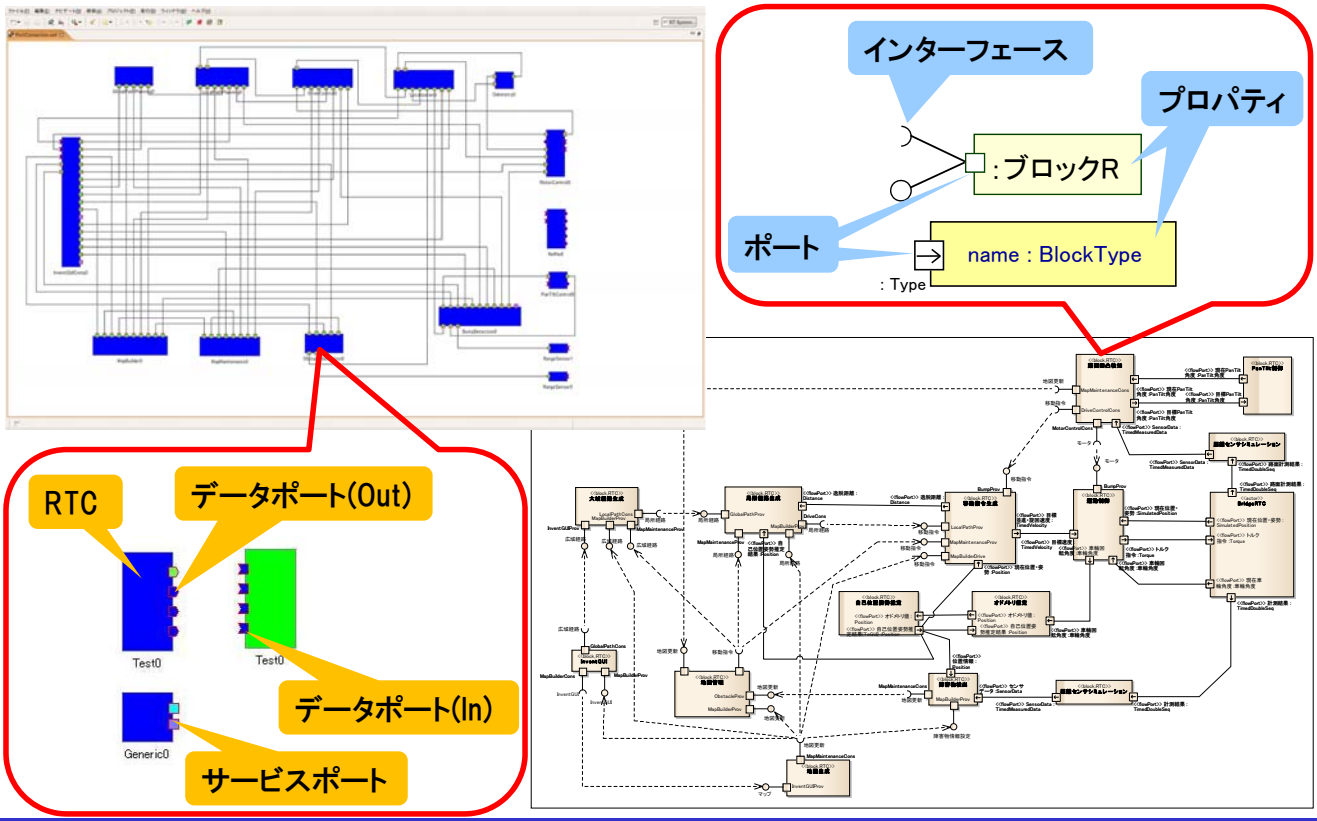
RTミドルウェアとの関係

- RTミドルウェアの基本仕様：OMGにて国際標準化
  - RTC:Robotic Technology Component Specification
  - UML:Unified Modeling Language
  - SDO:Platform Independent Model and Platform Specific Model for Super Distributed Object
  - SysML:OMG Systems Modeling Language



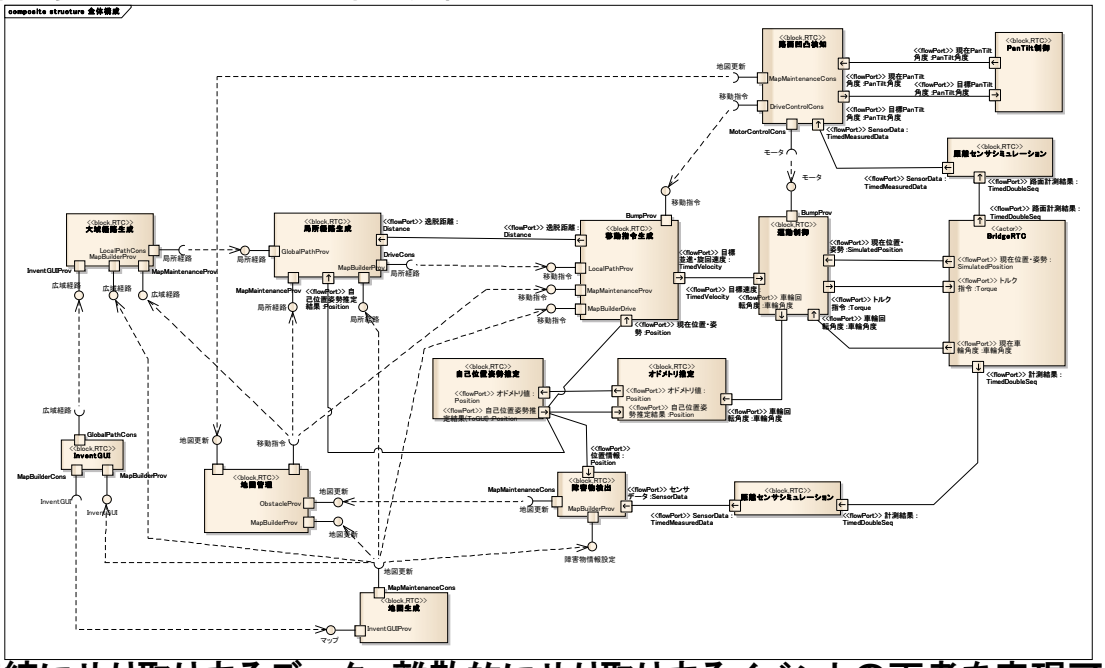
- 仕様として共通な部分が多い

RTシステムとSysMLの関係



内部ブロック図

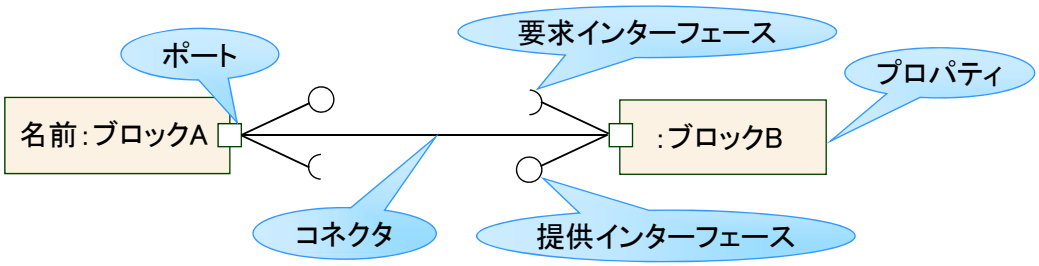
- システムを構成する要素間でやり取りする内容を表現
  - システム構成要素間の接続関係を明確にする
  - 構成要素間でやりとりする内容を明確にする



◆連続にやり取りするデータ，離散的にやり取りするイベントの両者を表現可能

内部ブロック図の構成要素①

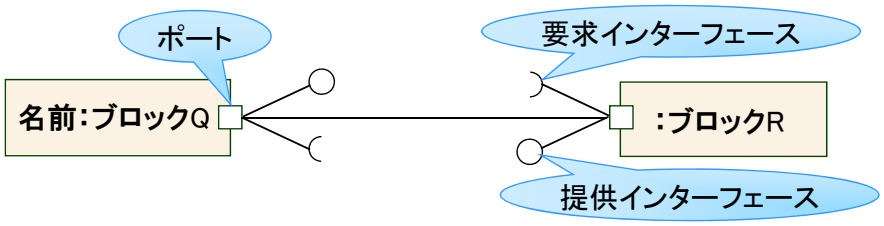
- パートプロパティ (Property)
  - ブロックに含まれる内部要素を構造的な役割で表現した要素
- ポート (Port)
  - プロパティの内部と外部の境界を表現



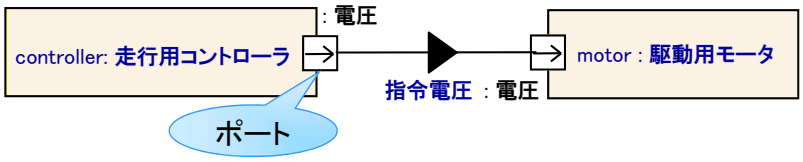
  - 実際に提供/要求している仕様は，インターフェースを用いて表現する
  - 離散的にやりとりする内容(イベントなど)を表現
- コネクタ (Connector)
  - プロパティ間に何らかの関係や相互作用が存在することを表現

ポートの種類

■ 離散的にやりとりする内容(イベントなど)を表現 → サービスポートに該当



■ 連続的にやりとりする内容を表現 → データポートに該当



- In(入力)

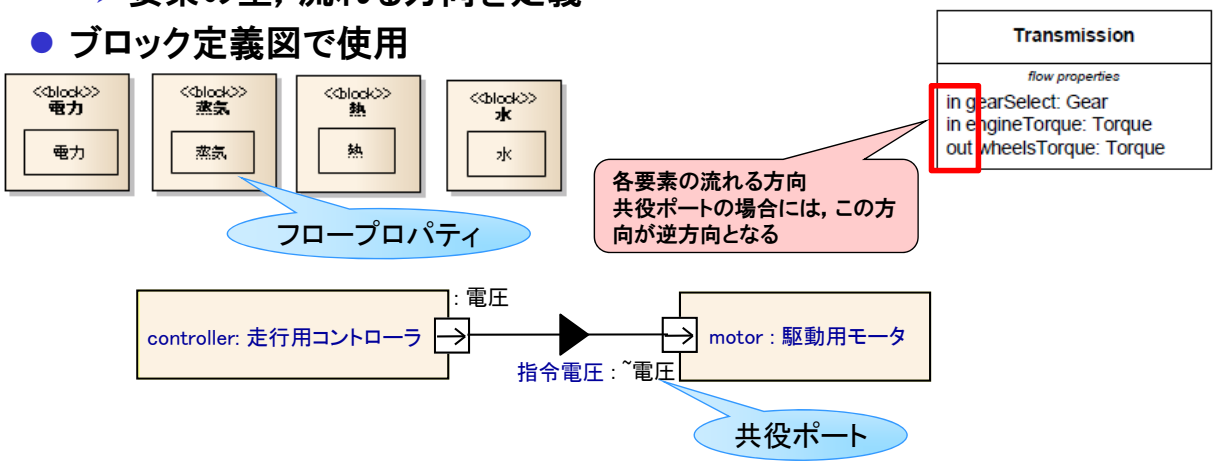
- Out(出力)

- InOut(入出力)

内部ブロック図の構成要素②

■ フロープロパティ (Flow Property)

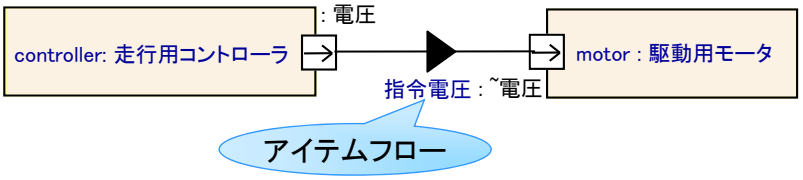
- ポートが流すことのできる要素を定義
  - 要素の型，流れる方向を定義
- ブロック定義図で使用



データポートの型定義に該当
各ポートはどのような型なのか？
各型にはどのような要素が含まれるべきなのか？を検討

内部ブロック図の構成要素③

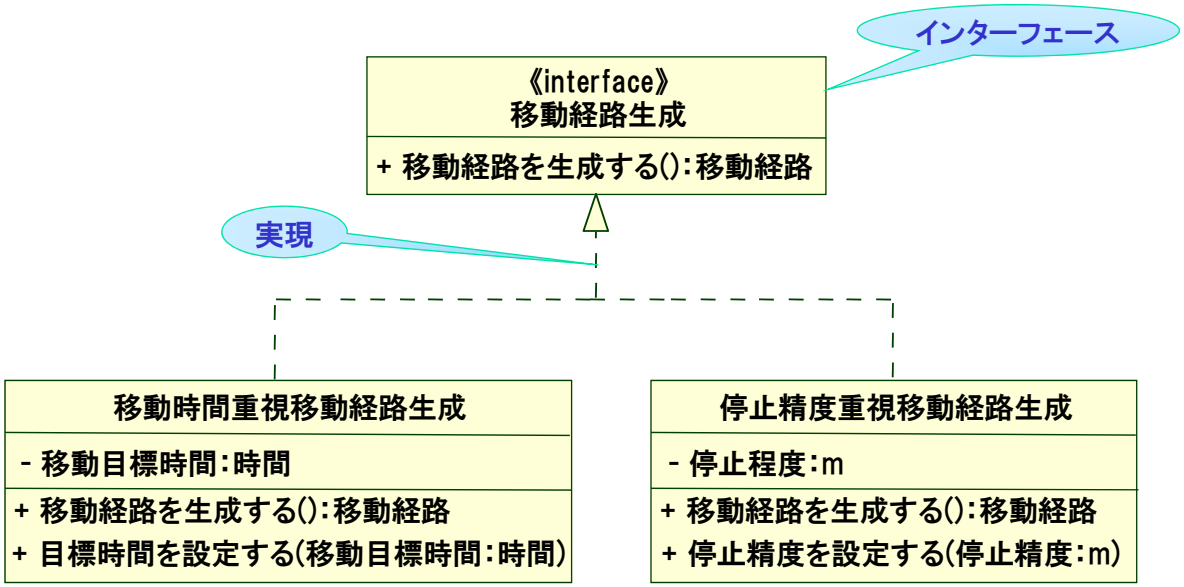
- アイテムフロー (ItemFlow)
  - プロパティ間を実際に流れている要素を表現



- ポートの型とアイテムフローの関係
  - ポートの型 → 何を流すことができるか？を表現
    - Ex)液体，位置情報，電圧など
  - アイテムフロー → 実際には何が流れているか？を表現
    - Ex)ブレーキオイル，現在位置情報，モータ速度指令電圧など

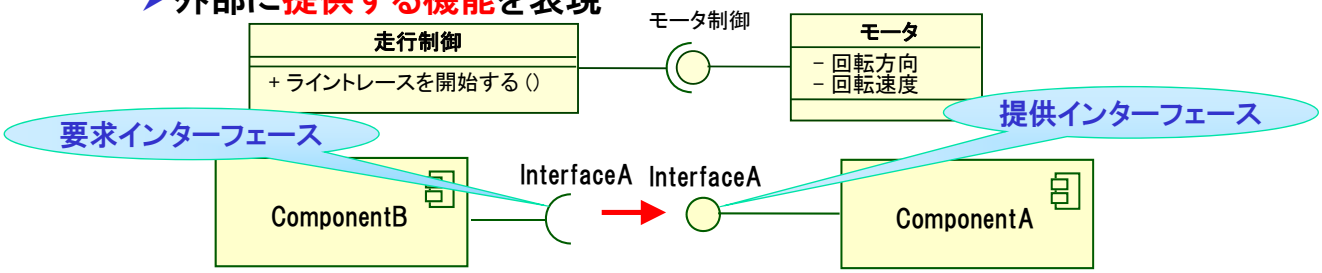
内部ブロック図の構成要素④

- インターフェース (Interface)
  - ブロックが外部に公開する操作/要求する操作の集合を表現



内部ブロック図の構成要素③

- 要求インターフェース (Required Interface)
  - 外部に**要求する機能**を表現
- 提供インターフェース (Provided Interface)
  - 外部に**提供する機能**を表現



◆機能を要求する側(使う側)　◆機能を提供する側(使われる側)

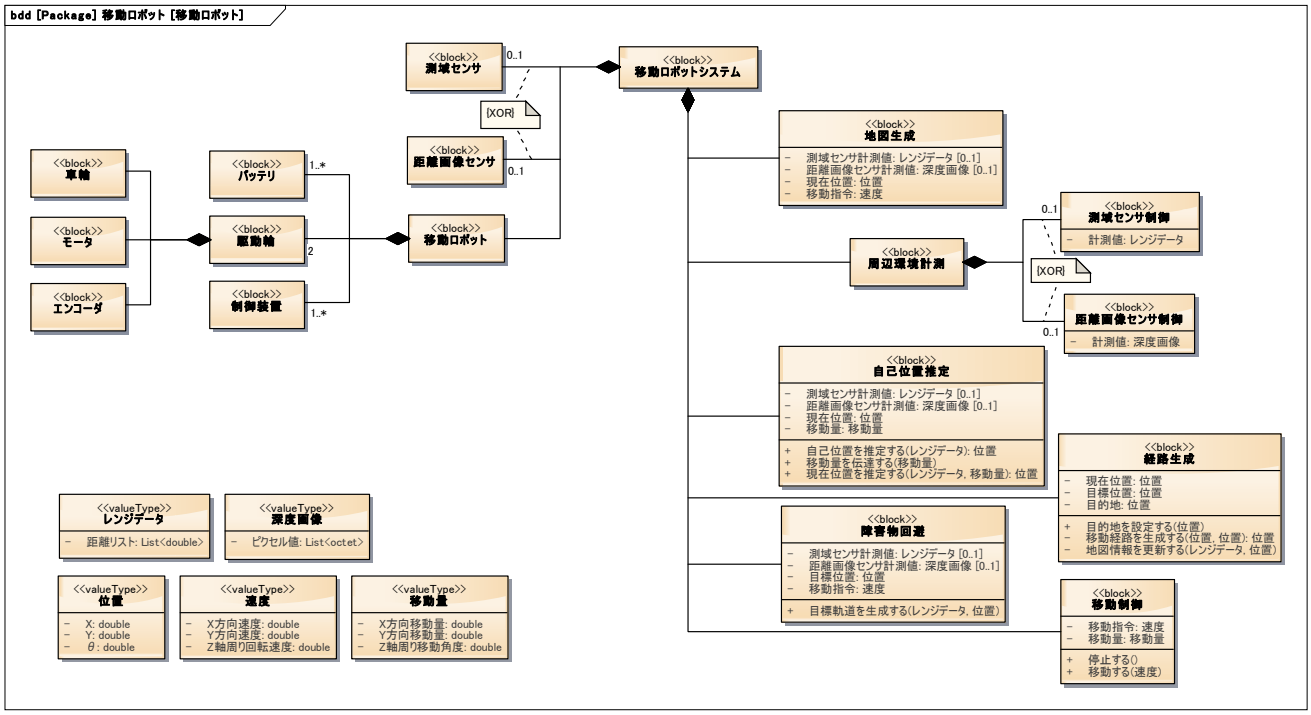
☑「要求」IF「提供」IFが表現するのは，あくまでも呼び出す方向の関係

☑ 必ずしも**データの流れとは一致しない**

サービスポートのインターフェース定義に該当
どのような操作(メソッド)が含まれるべきなのか？を検討

ブロック定義図

■ システムの静的な構造を表現



◆静的な要素，構造，関係，数的関係を表現

ブロック定義図の構成要素①

- ブロック (Block)
  - システムを構成する要素・部品を表現する
    - ソフトウェア部品だけではなく，ハードウェア部品，人，組織など明確な責務を持つ要素は全てブロックで表現することができる
    - 内部ブロック図の「プロパティ」の基となる要素

<<block>>
移動ロボットシステム

≪ block ≫
ブロック名
parts
パート名：型名 [多重度]
references
参照名：型名 [多重度]
values
値名：型名 [多重度] = デフォルト値
constraints
制約名：型名
operations
操作名(パラメータ名:型名,…):戻値型 [多重度]
receptions
<<signal>>操作名(パラメータ名:型名,…)

◆ 値(values)
ブロックが持つ属性.
ブロックの特徴を表すデータ要素
<名称>:<型名>[<多重度>][= <初期値>] [{プロパティ文字列}]

◆ 操作(operations)
ブロックが持つデータに対する処理
<名称>([パラメータ]) : <戻り値の型>
<パラメータ名称> : <型名> [<多重度>]

RTコンポーネントそのものに該当
対象システムをどのようなRTCで構成するか？を検討

ConfigrationSetの表現例

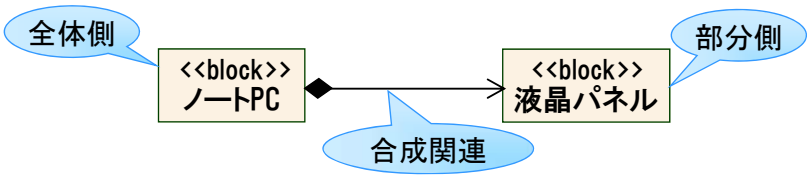
- ブロックの「値」とステレオタイプを使用
  - 値(values)
    - ブロックが持つ属性. ブロックの特徴を表すデータ要素
    - <名称>:<型名>[= <初期値>]

<<block>>
Flip
values
<<ConfigurationSet>> flipmode : int = 0

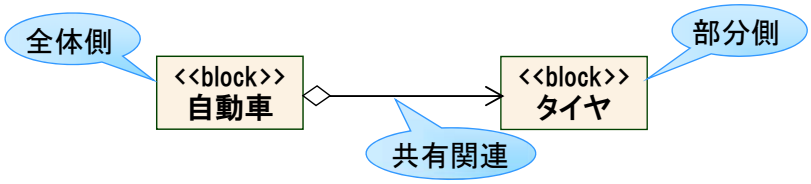
  - ステレオタイプ
    - SysML要素の意味をユーザが拡張するための仕組み
    - 本来の意味をユーザが追加拡張する場合に使用

ブロック定義図の構成要素②

- 合成関連 (PartAssociation)
  - ブロック間に「全体」-「部分」の関係があることを表現
  - 「部分」は「全体」と同じライフサイクルであり，全体側の構成要素となる
    - 「全体」が削除されると「部分」も削除される

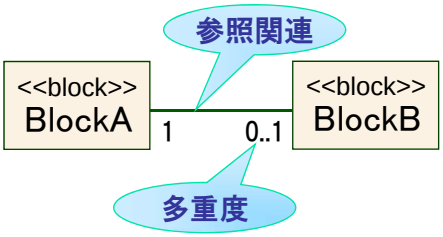


- 共有関連(SharedAssociation)
  - 合成関連と同様に，ブロック間に「全体」-「部分」の関係があることを表現
  - 「部分」と「全体」のライフサイクルは異なるため，「部分」単独でも存在できる



ブロック定義図の構成要素③

- 多重度
  - ブロック間の数量的な関係を表現
    - 固定値・・・1 や 10 など
    - 複数　・・・*(アスタリスク)
    - 範囲　・・・0..1 や 5..* など
      - なるべく「0」(なし)，「1」(あり)，「*」(複数あり)の組み合わせのみ使用する
      - 多重度が省略されている場合，多重度「1」とみなす

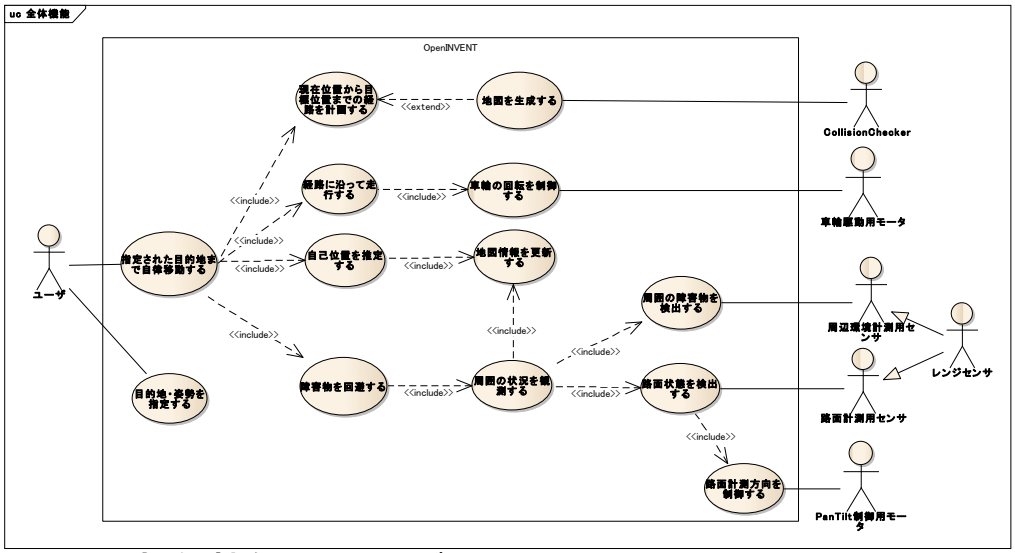


ブロック定義図作成時の注意点

- まずは構成要素の洗い出し
  - 上位の要素がどのような要素で構成されているか？どのように分割できるか？を検討
    - ハードウェア，ソフトウェア，人など，関係する要素を全て列挙
  - 「値」は，内部ブロック図を検討することで明確になる場合が多い
  - 「制約」は，パラメトリック図を検討することで明確になる場合が多い
  - 「操作」は，シーケンス図を検討することで明確になる場合が多い
    - 「インターフェース」は「操作」を検討した後に検討
- 構成要素間の関係の明確化
  - 「全体」－「部分」の関係の明確化
    - 各要素がどのような要素から構成されているかを検討
    - 単一要素の下に列挙される部分要素の数が多くなった場合には，中間的なまとまりを導入できないか検討
  - 数的関係の明確化
  - ハードウェアの情報とソフトウェアの情報を同じツリーに混在させない

ユースケース図

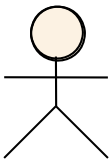
- 対象範囲(システム)が提供する機能の明確化
  - 開発対象の範囲(システム化範囲)を明確にする。
  - 機能とそのユーザの関係を明確にする。
  - 対象範囲(システム)が必要とする外部要素を明確にする。



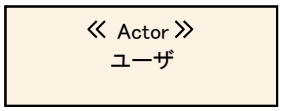
- 提供機能のカタログ化
- 開発を行う優先順位の決定にも利用される

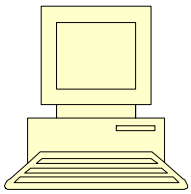
ユースケース図の構成要素①

- アクタ (Actor)
  - 対象(システム)と相互作用を行う外部要素。ユーザや外部機器など。
  - 抽象化した名前(役割等)で定義する。
  - 複数の表記方法がある。



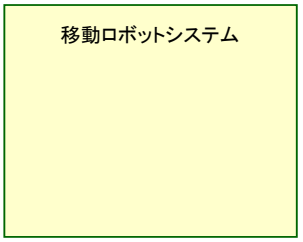
ユーザ





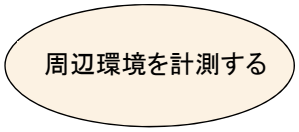
デバッグ用PC

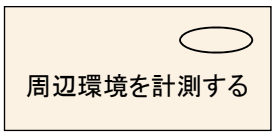
- サブジェクト(システム境界) (Subject)
  - 対象範囲(システム)の境界を表す。
  - ユースケースやアクタを囲んだ枠。
  - システムの名称やモデルの名称を記述する。



ユースケース図の構成要素②

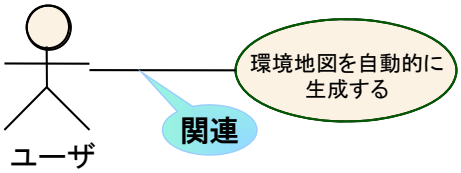
- ユースケース (UseCase)
  - 対象範囲(システム)の外部から見た機能(振る舞い)を表現する。
  - システムの内部処理(粒度の細かい処理)はユースケースではない。





RTコンポーネントを導出する際の手掛かり
どのような機能を実現するRTCが必要なのか？を検討

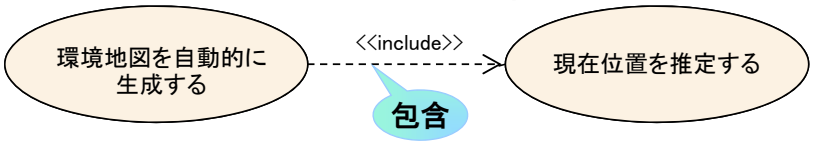
- 関連 (Association)
  - アクタとユースケース間の関係を表現する。
    - 機能とそのユーザの関係



ユースケース図の構成要素③

■ 包含（include）

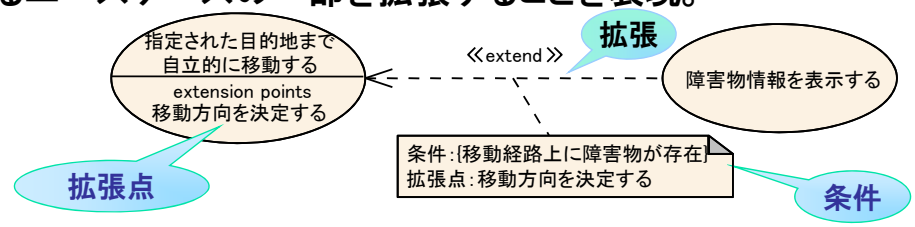
あるユースケースが、別のユースケースを機能の一部として含むことを表現。



- 矢印の方向は，機能の呼び出し方法と同じ方向
- メインルーチンからサブルーチンを呼ぶような関係
- 包含する側は，包含される側がないと機能を実現できない

■ 拡張（extend）

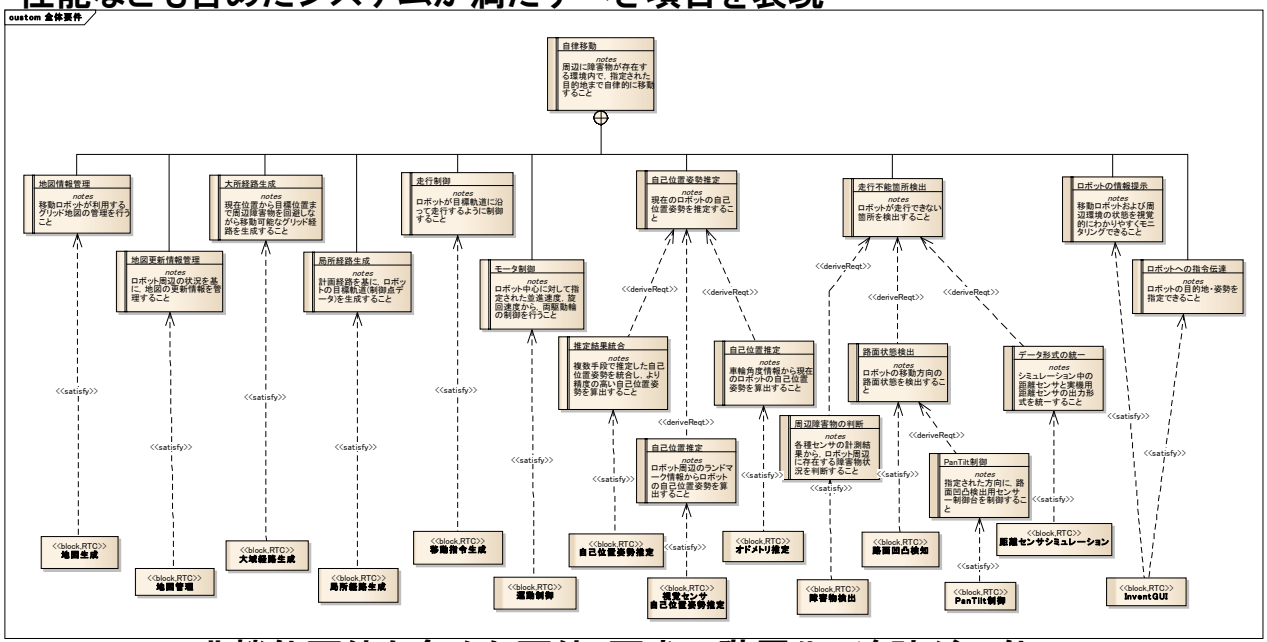
あるユースケースの一部を拡張することを表現。



- 矢印の方向は，拡張する側から，拡張される側に向かう
- 元々ある機能に，後から機能を付加するような関係
- 拡張される側は，拡張する側がなくても機能を実現できる

要求図

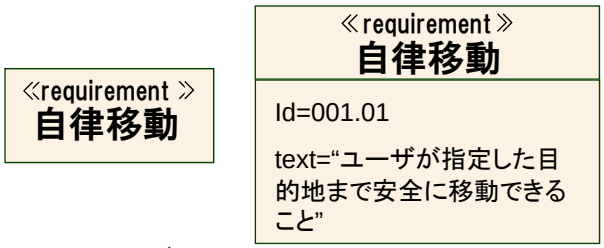
■ 性能なども含めたシステムが満たすべき項目を表現



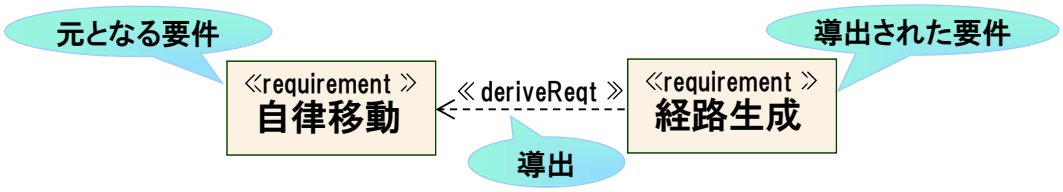
- 非機能要件も含めた要件・要求の階層化，追跡が可能
- 要件・要求の内容については，文章として記述
- 表形式での表現も可能

要求図の構成要素①

- 要件 (Requirement)
  - システム全体もしくはシステムの構成要素が満足すべき条件，性能を表現
    - ユーザからの要求，技術的な制約などを表現



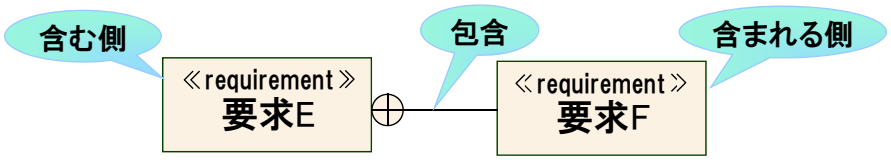
- 導出 (Derive Dependency)
  - ある要件から別の要件が導き出される関係



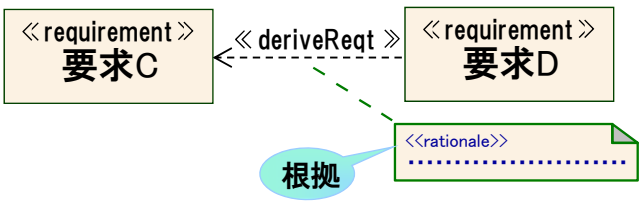
    - ユーザ視点の要件を技術的な要件に分解，整理
    - 抽象的な要件の具体化

要求図の構成要素②

- 包含 (Containment)
  - ある要件が他の要件に含まれていることを表現
    - 要件を構成する内容の明確化

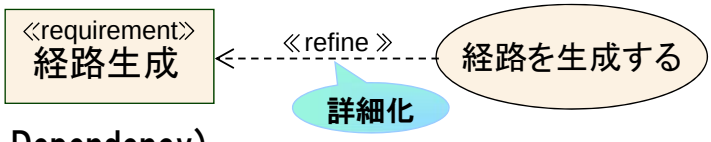


- 根拠 (Rationale)
  - 分析/設計中に行った決定の根拠を記述
    - 各要素間の関係などに付加



要求図の構成要素③

- 詳細化 (Refine Dependency)
  - 異なるフェーズ間の関係を表現
    - 要件定義フェーズと設計フェーズ間の要素間の関係など
      - 「要件」と「機能」(ユースケース)の関係



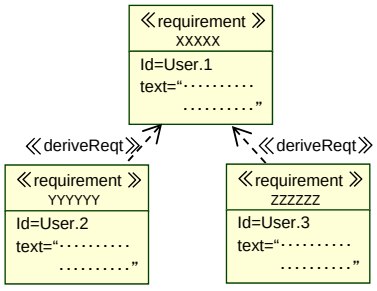
- 充足 (Satisfy Dependency)
  - 要件とその要件を実現する要素の関係

- 現状のシステム分析，システム設計にてカバーしている範囲の明確化
- 将来的に対応が必要な項目の明確化

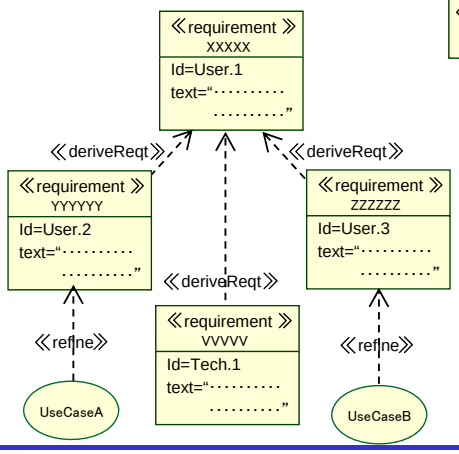
要求に関係しないユースケース，ブロックは存在しえない
もしも存在する場合，開発者が勝手に追加した機能，要素となる

要求図

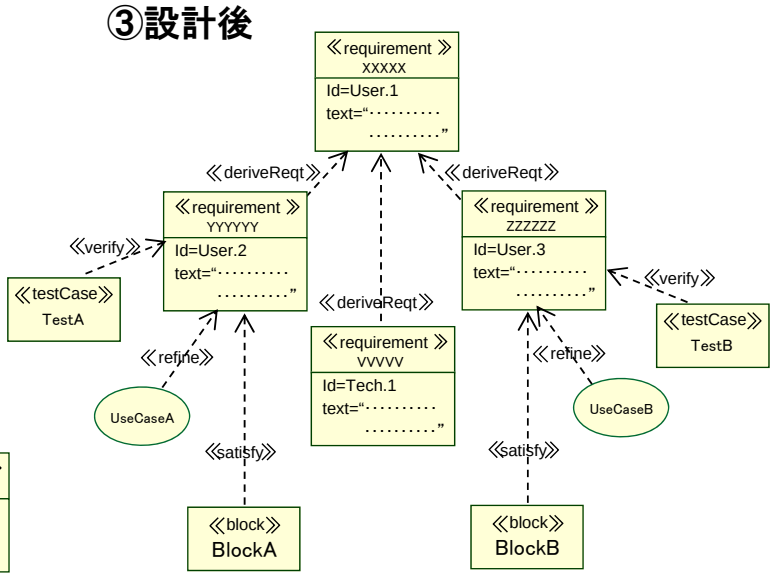
①要求分析時



②機能分析後

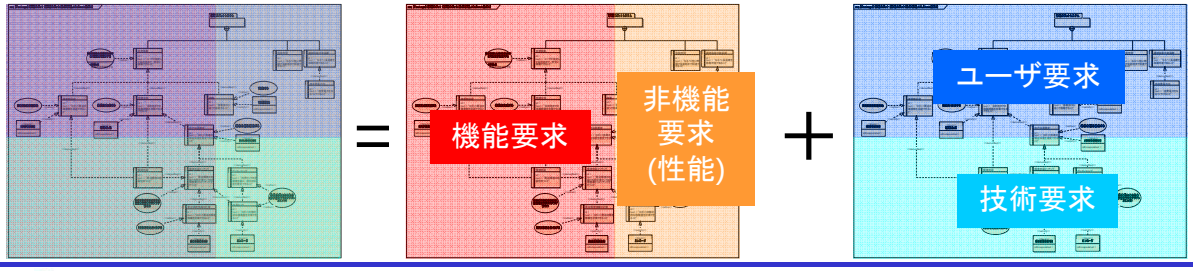


③設計後



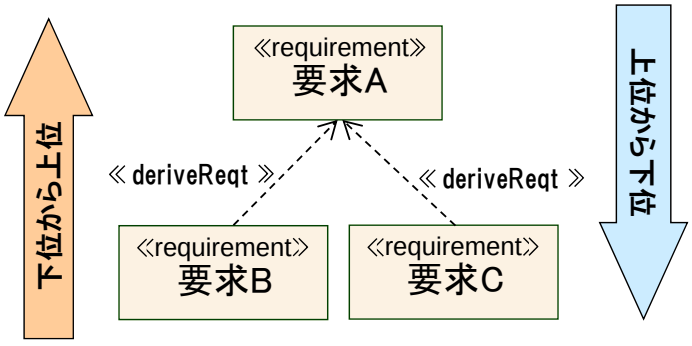
要求図作成時の注意点

- 機能要件と非機能要件
  - 機能要件：システムに対して要求されている機能
  - 非機能要件：システムに対する品質特性や制約事項
- 非機能要件よりも機能要件の方が特定を行いやすい
  - 非機能要件は，ユーザの使い勝手に直結していることが多いため重要
- 要件のレベル分け
  - ユーザ要件：ユーザ自身が希望していること
    - ユーザは必ずしも技術には詳しくない
  - 技術要件：ユーザが望む事を実現するために必要な内容
- ユーザ要件と技術要件は明確に区別
  - ユーザ要件の明確化がより重要



要求図作成時の注意点

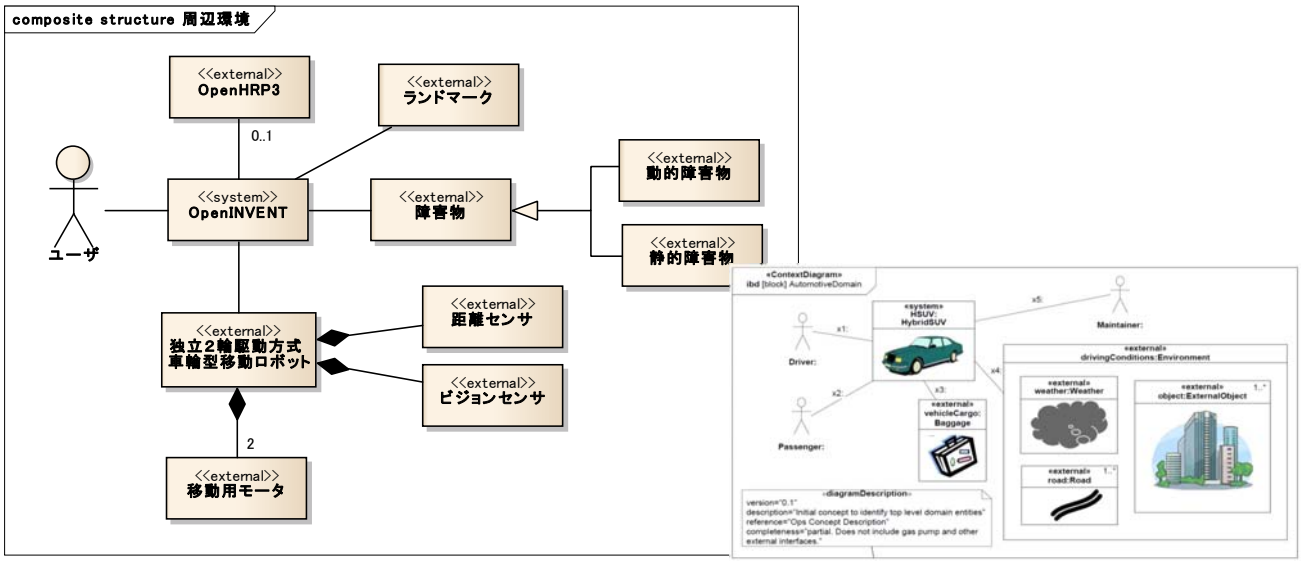
- 導出関係の意味について



- 下位から上位
  - 【下位要求】は，【上位要求】を満たす(実現する)ために必要
    - 【要求B】【要求C】は，【要求A】を満たすために必要
- 上位から下位
  - 【上位要求】を満足させる(実現する)ためには，【下位要求】が必要
    - 【要求A】を満たすためには，【要求B】【要求C】が必要

コンテキスト図

- システムが動作する環境を表現



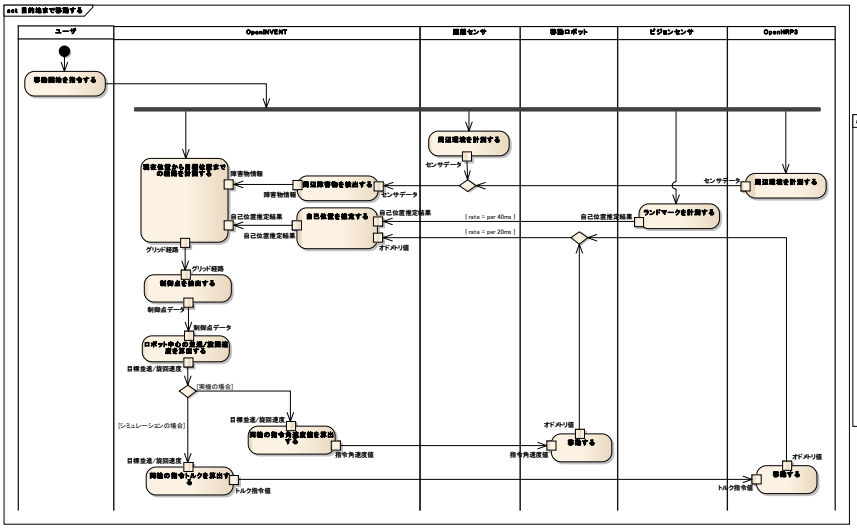
- 対象システムと相互作用する要素，影響を与える要素を抽出する
- 以降の分析/設計作業にて，検討する必要のある要素を洗い出す
  - 記述していない要素については，検討対象外
    - ☑ SysML仕様内で明示的に定義されたダイアグラムではない

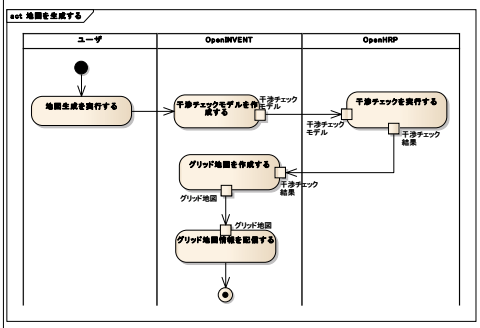
アクティビティ図

- 処理の流れを表現

  現実世界の処理流れを整理する(ビジネスプロセスも含む)

  ある機能を実現するための処理の流れ(アルゴリズムを含む)を表現

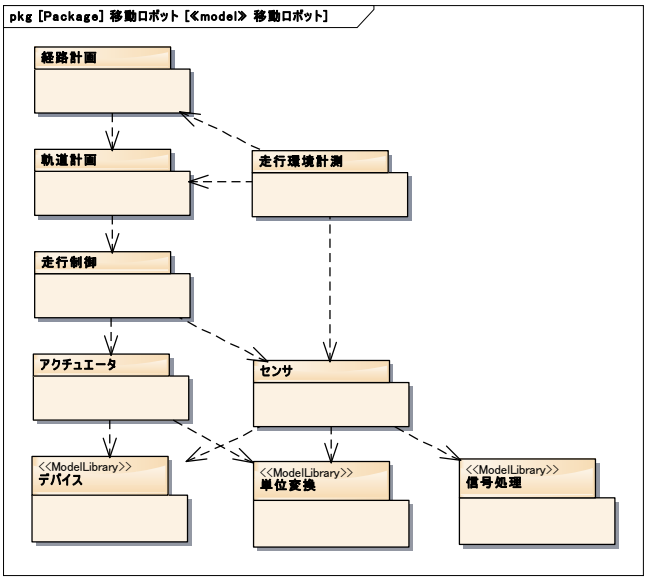




- 処理の実行順序，各処理の実行主体を表現
- 各処理間でやり取りするデータを表現
- 粒度に応じて処理を階層化して表現することも可能

パッケージ図

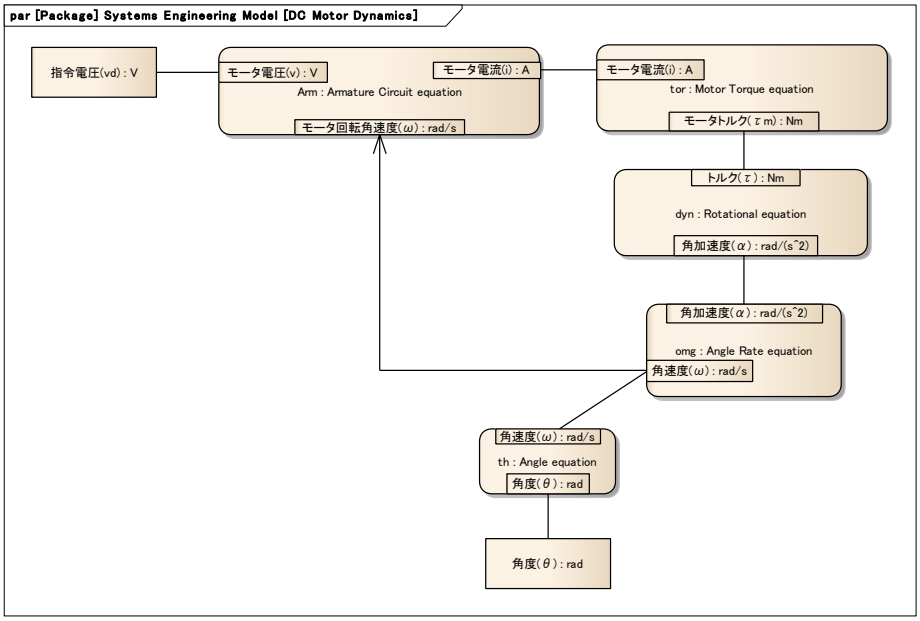
- 静的なおおまかな構造および構造間の関係を表現
  - パッケージは再利用を行う際の一番大きな単位であり，モデル要素をグループ化する単位となる



☑ 専門分野(ドメイン)の分割，依存関係を表現する「ドメイン・チャート」としても利用可能

パラメトリック図

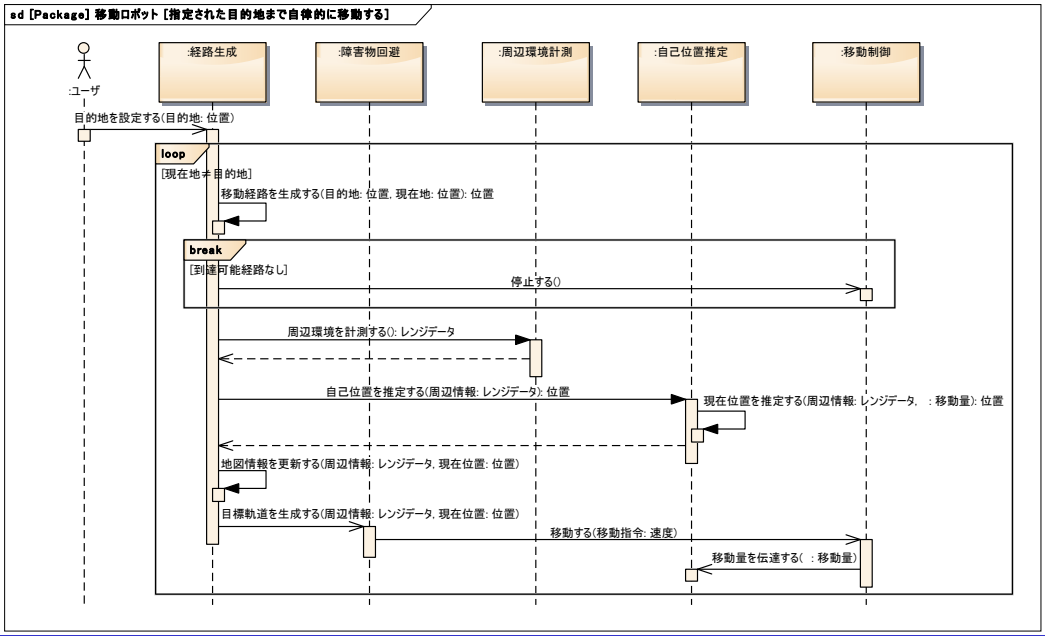
- システムに付随する制約の関係を表現
  - 各ブロックで定義された制約間の関係を表現
  - ブロック線図と似た内容を表現することが可能



☑ 内部ブロック図の1種

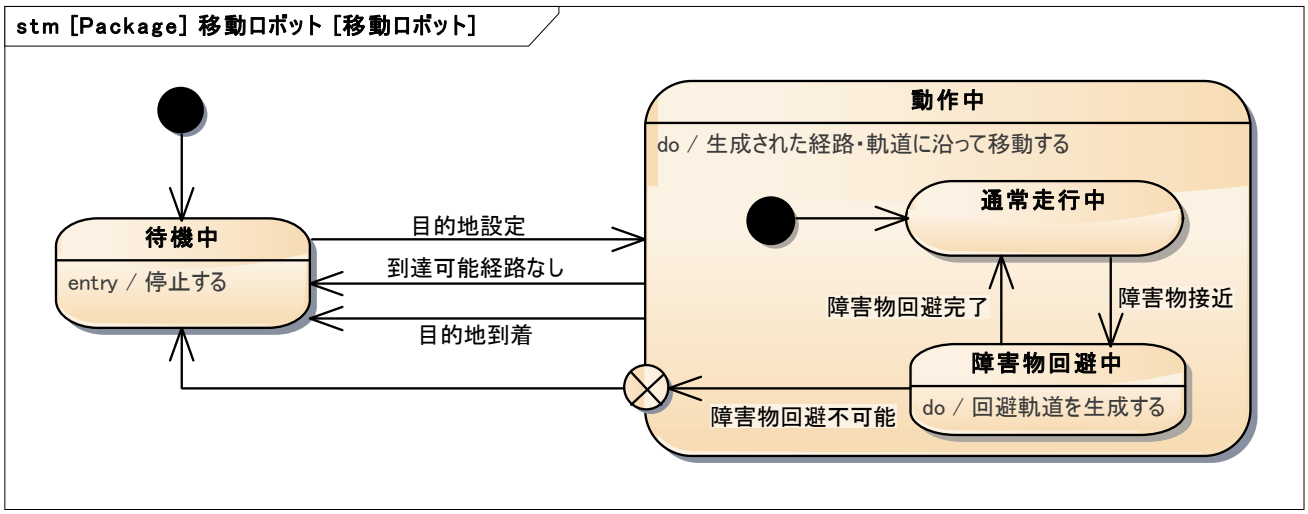
シーケンス図

- システムの相互作用を表現
  - 要素間でやり取りする情報とその手順を表現
- ある“機能”を実現するための処理の流れを時系列に沿って表現する
  - 上から下に向かって時間が流れる



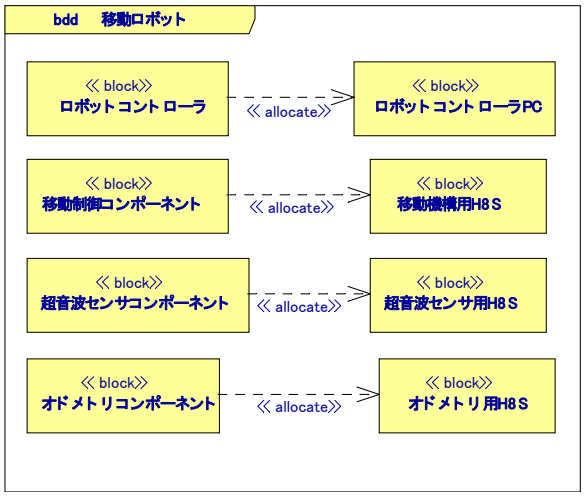
ステートマシン図

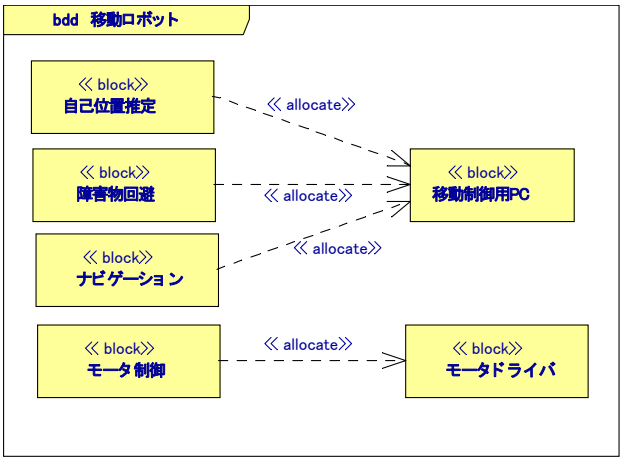
- ある要素が生成されてから破棄されるまでの状態の移り変わりを表現
  - 状態が変化する流れと，そのきっかけを表現可能
  - 各状態で実行する振る舞い(アクティビティ)も表現可能



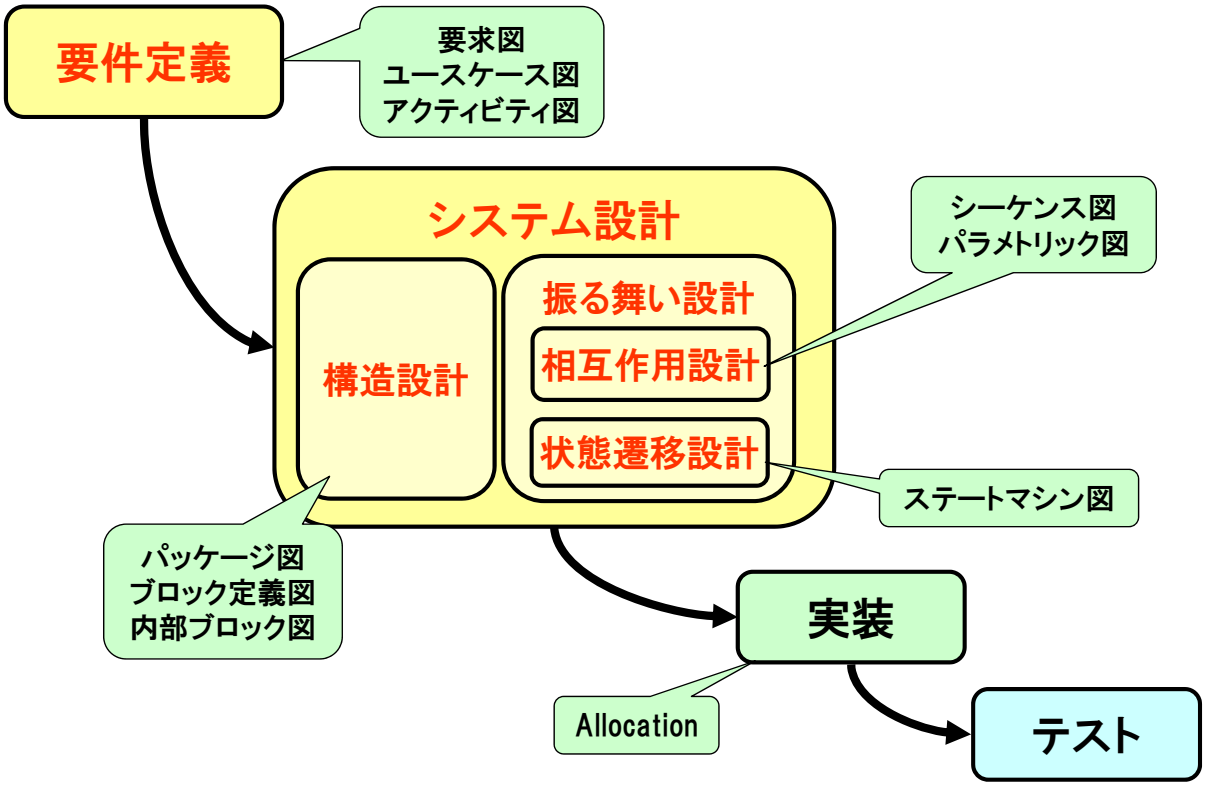
アロケーション

- 要素間の対応関係を表現
  - ソフトウェアコンポーネントのノード配置，アクティビティのブロックへの振り分けなどを表現
  - 「機能」と「構造」の分離を実現





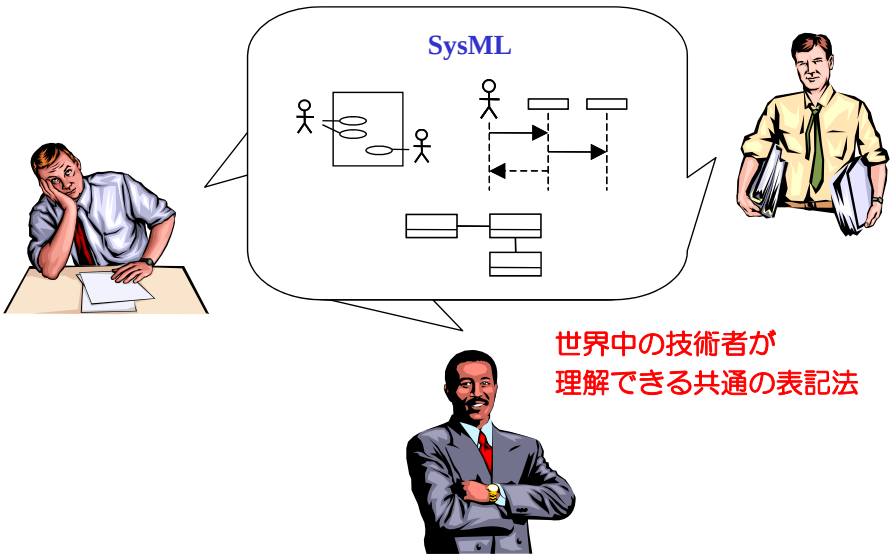
システム開発の流れ



SysMLの特徴

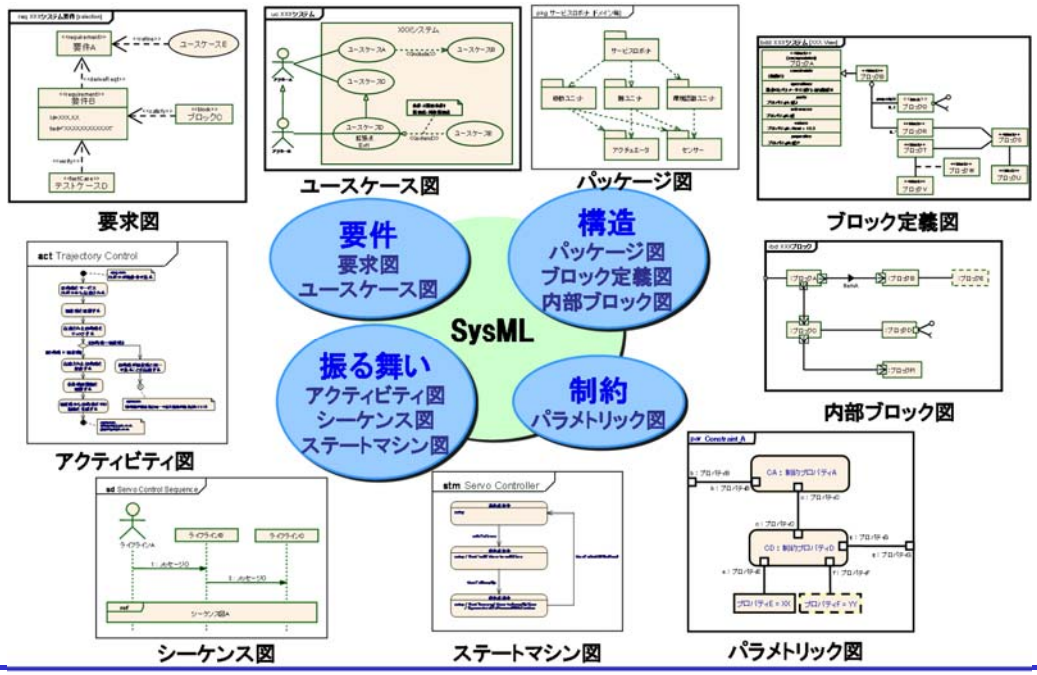
SysMLの特徴

- コミュニケーションツールとしての効果
  - 仕様書がビジュアル化されるため，読みやすく理解しやすい
  - 表記法が国際的に標準化されているため，世界中の技術者が理解できる
  - ユーザー、開発者、テスターなど開発に関与する誰もが使える



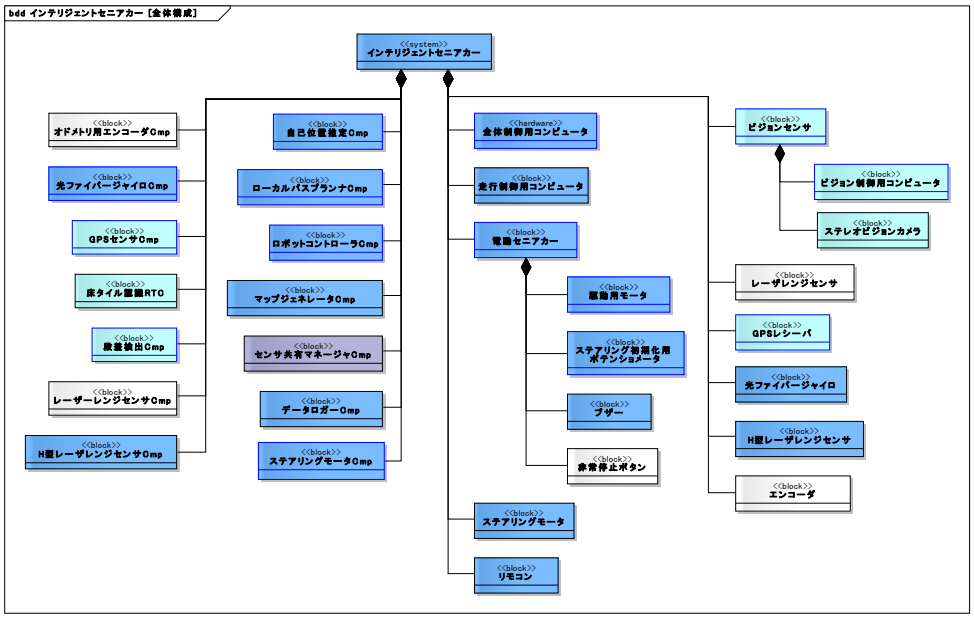
SysMLの特徴

- 視点、関心事を変え、様々な側面から表現が可能
  - 複数のダイアグラム(図面)が用意されている
  - 上流工程から下流工程まで幅広く使用できる



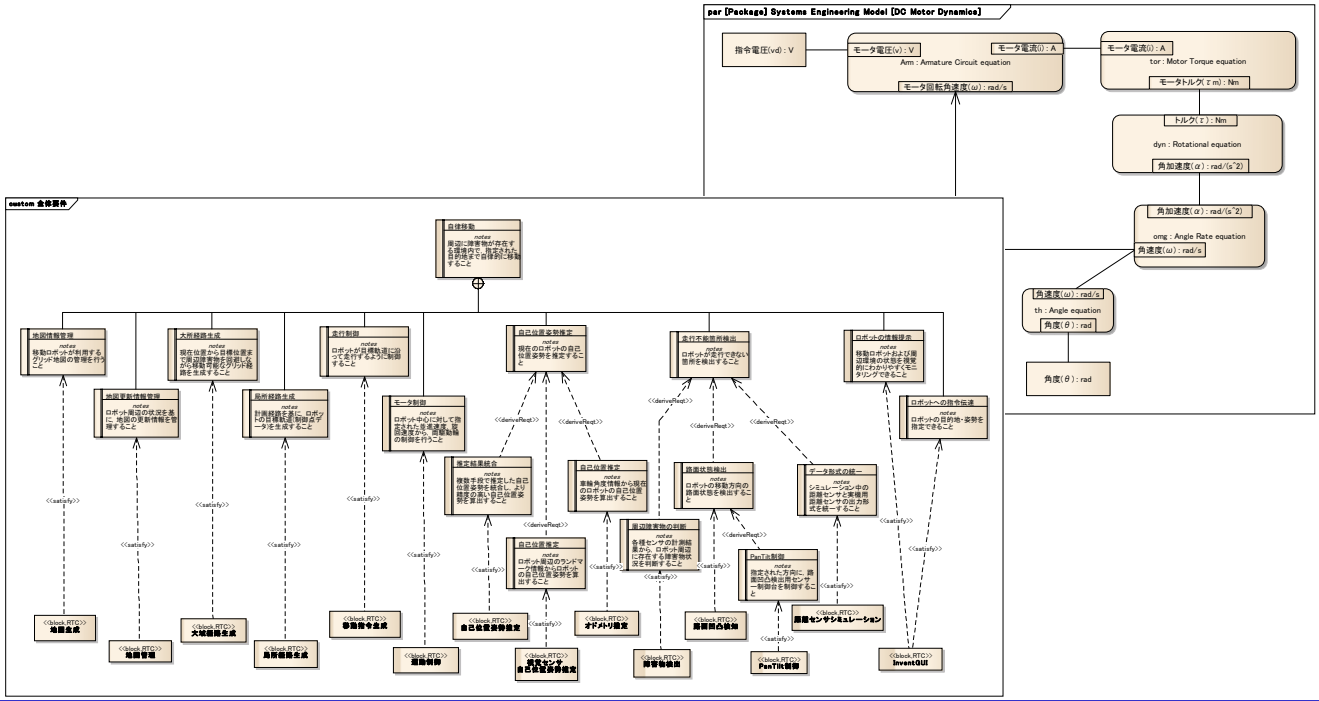
SysMLの特徴

- ハードウェアも含めたシステム設計全体をサポート
  - 既存モデルの置き換えを狙っているのではなく，あくまでも他ドメインの専門家とのコミュニケーションの円滑化が狙い
  - 必要に応じて，処理，データ，人，組織などの要素も表現可能



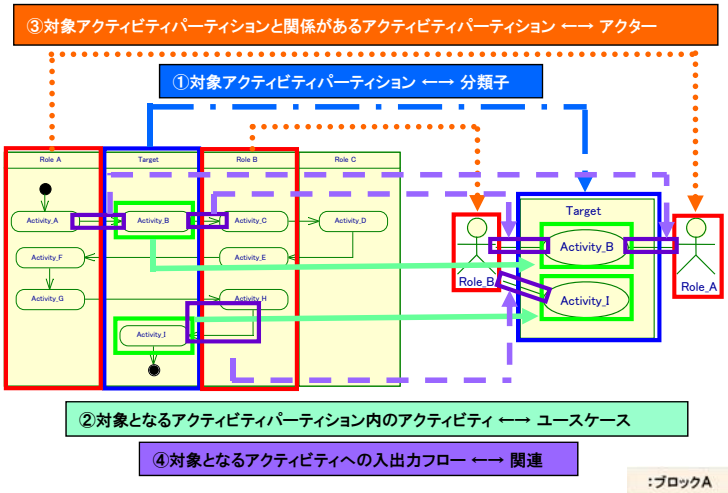
SysMLの特徴

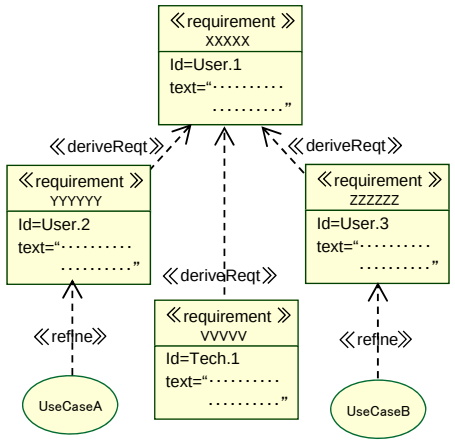
- システム開発の広範な工程をサポート
  - ユーザ要求を明確化する部分から，実際の制御系まで広範囲をサポート



SysMLの特徴

- ダイアグラム間の関係が明確
  - メタモデル定義により，仕様自体を定義
  - 粒度が異なる要素間の関係も明確化可能





仕様変更が発生した際の影響範囲の明確化が可能

SysMLの利用方法

- スケッチとしてのSysML
  - コミュニケーションやものごとを理解することを目的とした利用方法
  - 仕様に厳密に準拠しているかどうかという細かいことは重視しない
  - 主として，分析段階で使用する
- 設計図としてのSysML
  - 設計面の全ての意志決定が明確になるだけの必要十分な情報を提供する
  - モデリング・ツールを利用することがほぼ前提となる
  - 主に設計段階で使用する
- プログラミング言語としてのSysML
  - 実行可能なソフトウェアをモデル上で実現する
    - MDA(Model Driven Architecture)は，この立場で利用する一例

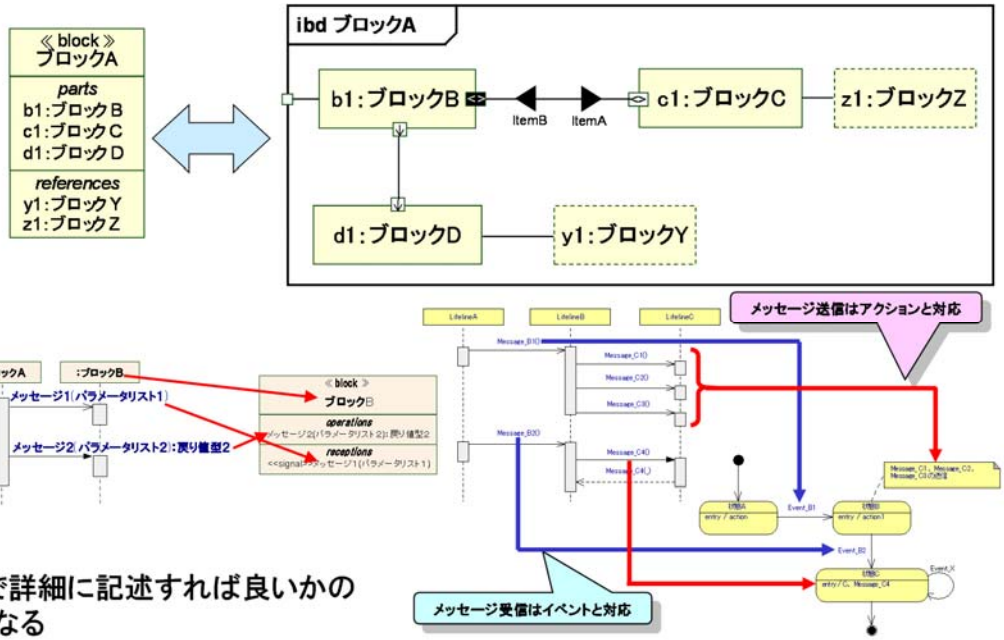
何を目的としてモデルを作成しているのか？
何の検討を行いたいのか？何を明確に整理したいのか？
を常に意識する事が重要

SysMLを活用するポイント

- 使用する用語の意味を統一する
  - 用語辞書の作成
  - 同じ概念を表現する用語は1つに統一する
  - 1つの用語で表現する概念は1つに統一する
    - 一般的な用語ほど，誤解を発生しやすい
    - 同じ用語でも，開発フェーズによって違う意味で使われていることも多い
- 対象としているモデルの抽象度に注意する
  - 工程の早い段階から，必要以上に詳細に拘らない
    - 仕様，アルゴリズムなどの詳細が最初からわかっていることも多いため
    - 最終的には，詳細まで明確にする必要はある
    - 早い段階から詳細にばかり拘ると，先に進まなくなってしまう
  - 作成しているモデルの抽象度，粒度にバラツキがあると，
    後工程での利用，再利用が難しくなる

SysMLを活用するポイント

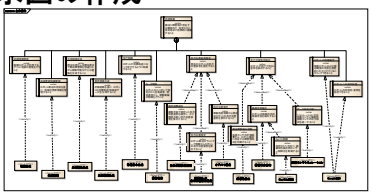
- ダイアグラム間，工程間の繋がりを意識する
  - ダイアグラム間の関係はある程度決まっている
  - 成果物，モデルを後工程でどのように利用するかを意識する



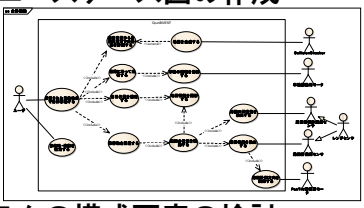
➢どのレベルまで詳細に記述すれば良いかの判断材料にもなる

RTSystem開発時のSysML利用法

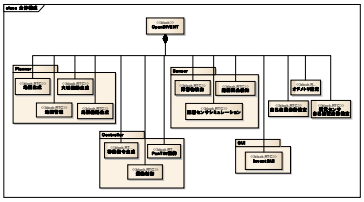
01. システムが「何を」実現すべきなのかを検討
  - 要求図の作成



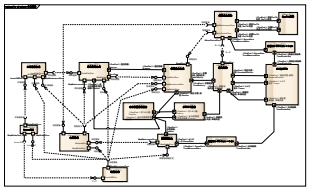
02. システムが実現すべき機能の検討
  - ユースケース図の作成



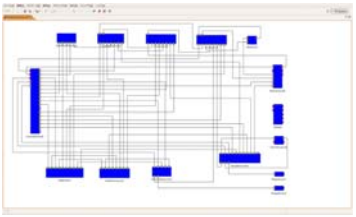
03. システムの構成要素の検討
  - ブロック定義図の作成

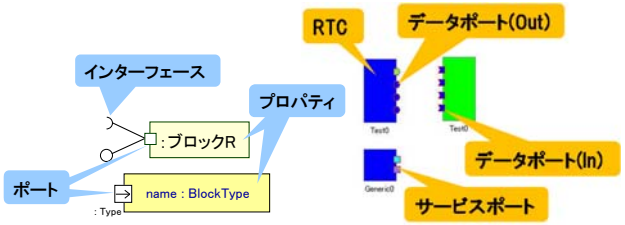


04. 構成要素間でやり取りするデータの検討
  - 内部ブロック図の作成



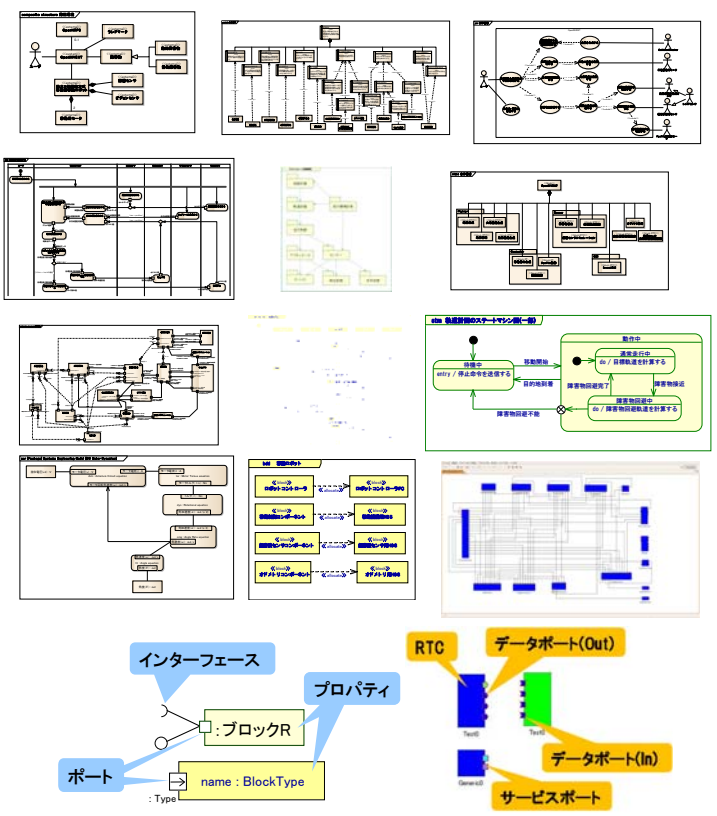
05. RTC,RTSystemへのマッピング





RTSystem開発時のSysML利用法

00. システムが動作する環境の定義
    - コンテキスト図の作成
01. システムが「何を」実現すべきなのかを検討
    - 要求図の作成
02. システムが実現すべき機能の検討
    - ユースケース図の作成
03. 機能を実現するための手順の検討
    - アクティビティ図の作成
04. システムの概略構成(名前空間)の検討
    - パッケージ図の作成
05. システムの構成要素の検討
    - ブロック定義図の作成
06. 構成要素間でやり取りするデータの検討
    - 内部ブロック図の作成
07. 構成要素間のやり取りの手順の検討
    - シーケンス図の作成
08. 各構成要素毎の状態遷移の検討
    - ステートマシン図の作成
09. 連続系(制御系)の検討
    - パラメトリック図の作成
10. 実際の動作環境の検討
    - アロケーションの作成
11. RTC,RTSystemへのマッピング



まとめ

- RTミドルウェアは，コンポーネントベース開発のための枠組み
- コンポーネントベース開発では，標準規格(仕様)と設計図面が重要
- 標準規格(仕様)としては，共通インターフェース仕様書が存在
- 設計図面としては，SysMLが利用可能
    - ハードウェア，ソフトウェア両方を含めた表現が可能であるため，ロボットシステムと相性が良い
    - RTミドルウェアと同じOMGで標準化されており，ベースとなっている仕様に共通部分が多いため，RTコンポーネントと相性が良い
    - 内部ブロック図を作成できれば，そのままRTコンポーネントに機械的にマッピングする事も可能
- 後々，他のユーザが利用する事も考慮し，開発時にどのような検討を行ったのか？を残しておく事も重要