RTミドルウェア SUMMER CAMP 2014

モデリング講習会

日時：2013年7月25日(金)
場所：早稲田大学 理工キャンパス55号館
S棟2階第3会議室

株式会社　グローバルアシスト
坂本 武志

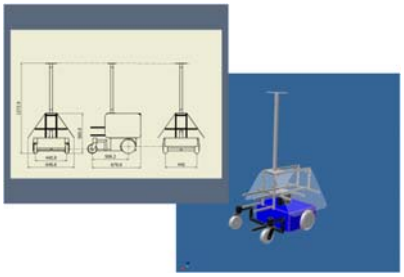
コンポーネント開発のためのモデリング手法

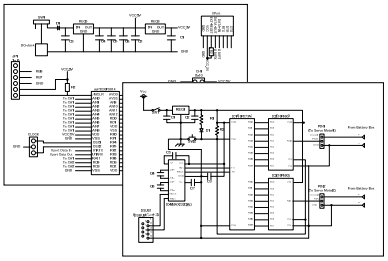
ソフトウェア/システム開発について

- ソフトウェア規模の表現方法
  - 人月：1人の開発者が1ヶ月かけて開発可能な量
    - 1人1ヶ月 → 1人月
    - 2人1ヶ月 → 2人月
    - 4人1週(0.25ヶ月) → 1人月
- ちなみに・・・
  - 一般的な2階建ての木造住宅の建築工数
    - 工期：3～4ヶ月程度
    - 人数：3人(棟梁＋大工2人)程度
  - 6～8人月
- 研究などで，担当者が1年間ソフトウェア開発を行った場合
  - 12人月
    - 2階建て木造住宅を建築するために必要な工数より多い
- 設計作業を行わずに建てた家に住みたいか？

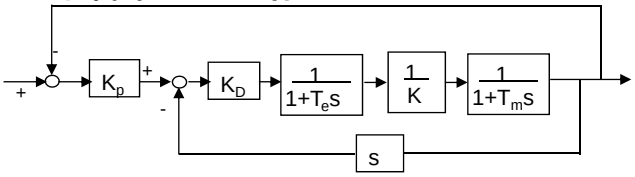
ソフトウェア/システム開発について

- 科学/工学の定義
  - 客観性：誰が作業を行っても同じ結果を導出可能
  - 再現性：何回作業を行っても同じ結果を導出可能
- 図面(モデル)の利用
  - 各ドメイン固有の内容を整理して表現するための共通した記法
    - 機械系図面
    - 電気回路図





- 制御系ブロック線図



☑ 工学分野では古くから利用されてきている

モデル

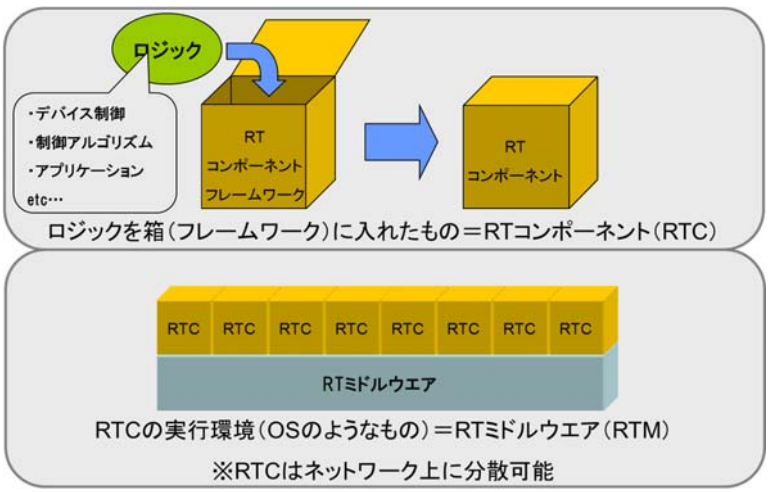
対象を「関心事」に注目し、「整理」して「表現」したもの

- 着目点/関心事の明確化
  - 検討すべき内容の明確化
  - 着目点の絞り込み
- 共通イメージの形成
  - 関係者間での認識のズレをなくす
- 7±2の法則
  - 人間が一度に記憶できる要素の数
  - 対象物の粒度/抽象度とは無関係
  - 「分割して統治せよ」(Devide and conquer)

ソフトウェア/システム開発について

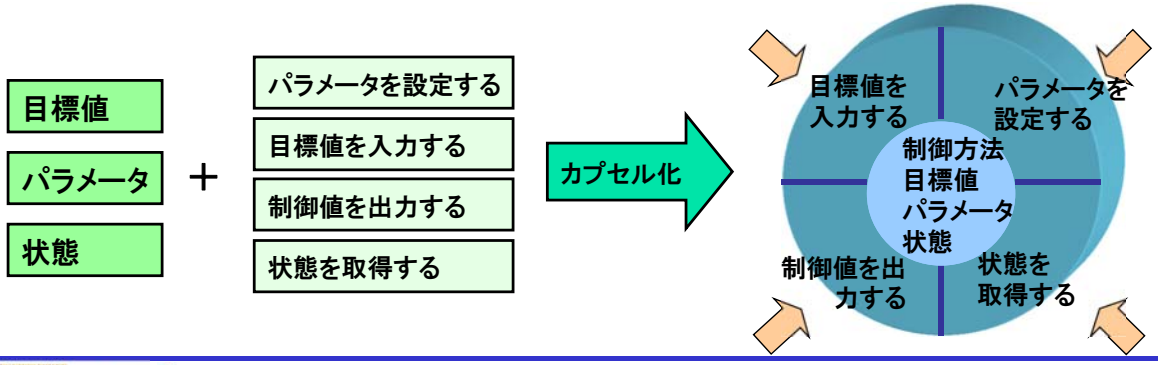
- 作業の標準化
  - 製造/開発手法の標準化
  - 製造/開発成果物の標準化
    - 住宅の場合:2×4

RTミドルウェアはソフトウェア成果物の標準化を行う技術



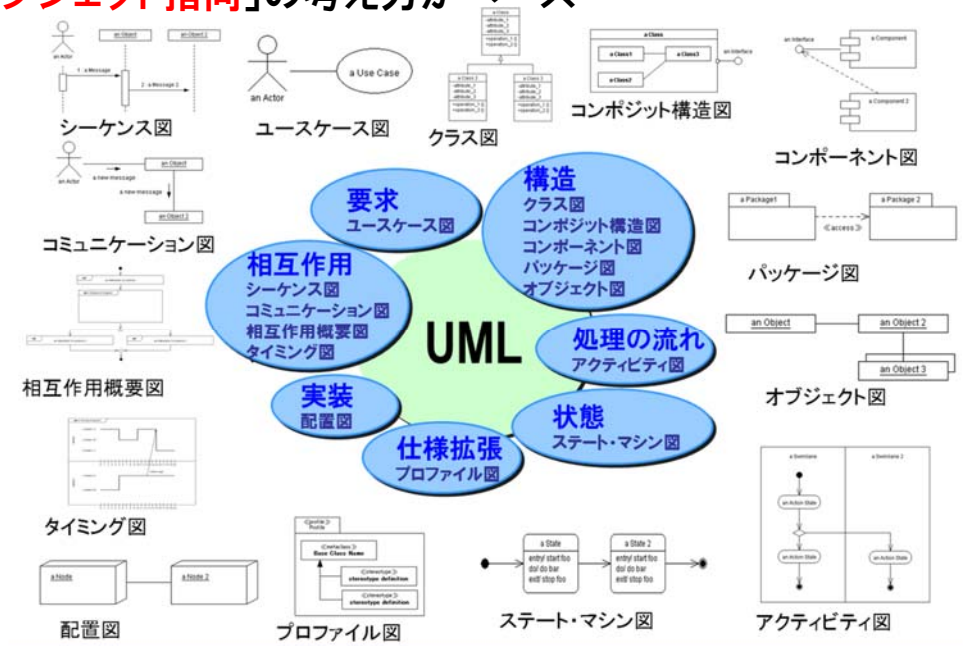
オブジェクト指向

- 元々は･･･ソフトウェアを開発するときの考え方のひとつ
  - ソフトウェア開発手法のひとつ
    - 他には構造化手法、手続き指向、アスペクト指向 などがある
    - 過去の考え方の良い部分を組み合わせて，更に発展させた考え方
- データとその操作方法をまとめてオブジェクト(クラス)を定義し、オブジェクトの組み合わせでシステムを構成する方法
- オブジェクトの情報(値)や操作内容を隠す(情報隠蔽)
- インターフェース(外部からの操作方法)と実装(内部の実装)を分離



UML

- 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

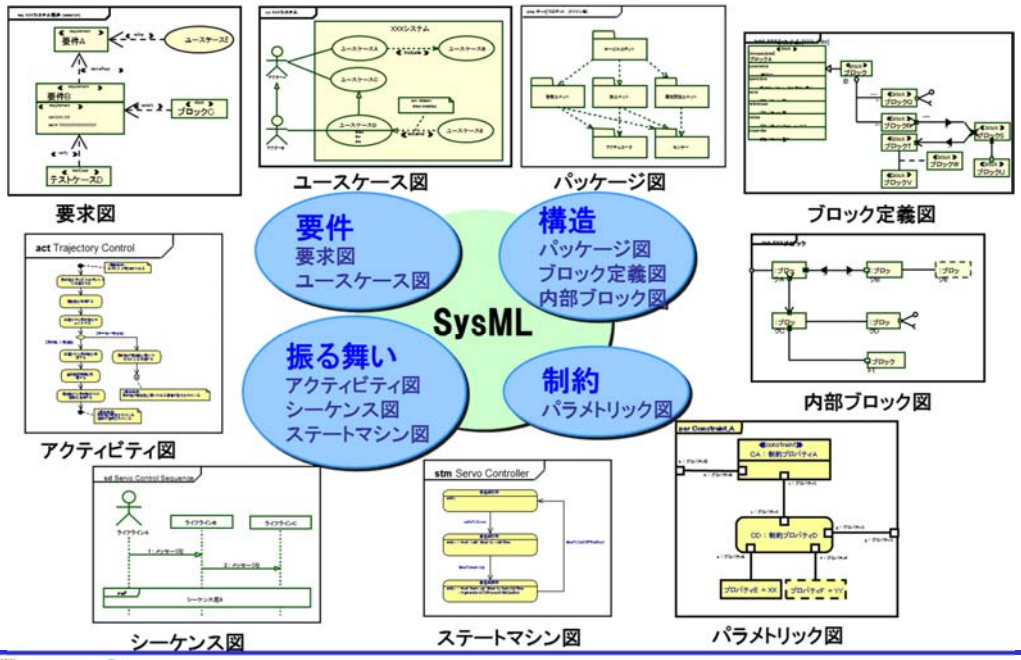


UMLの限界

- 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

SysML

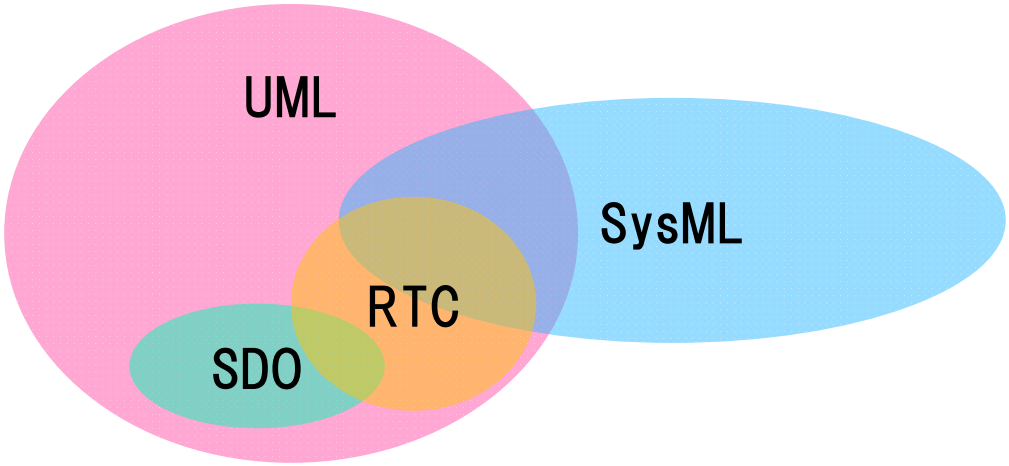
- System Modeling Language:システムモデリング言語
  - 「システム全体」の「モデル」を表現する言語(表記法＋意味)である。
  - 「UML」「オブジェクト指向」がベースとなっている。



SysML

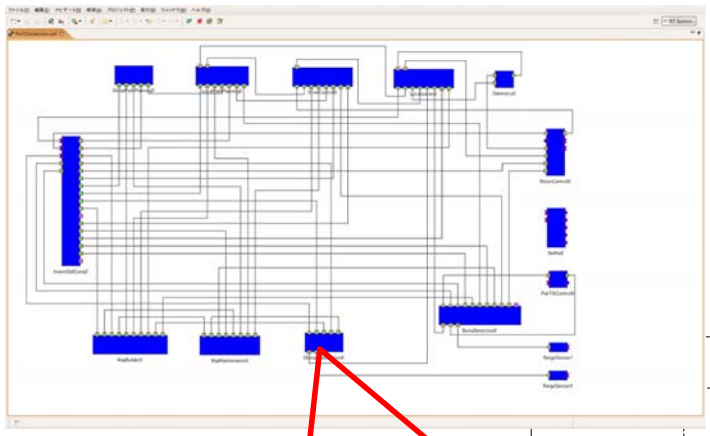
RTミドルウェアとの関係

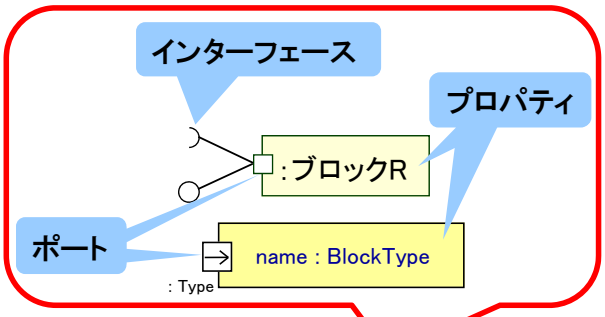
- RTミドルウェアの基本仕様：OMGにて国際標準化
  - RTC:Robotic Technology Component Specification
  - UML:Unified Modeling Language
  - SDO:Platform Independent Model and Platform Specific Model for Super Distributed Object
  - SysML:OMG Systems Modeling Language

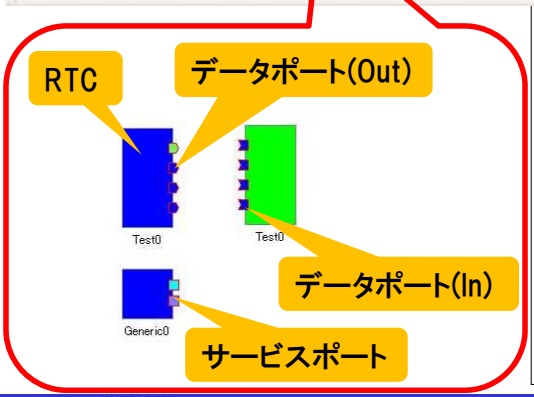


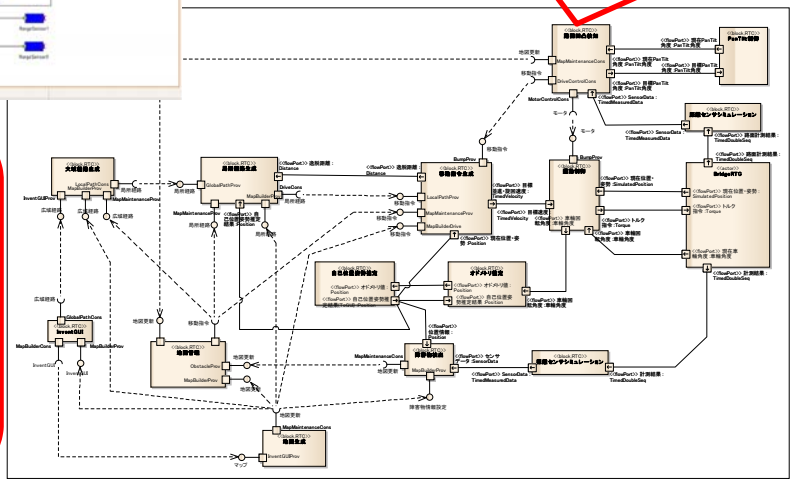
- 仕様として共通な部分が多い

RTシステムとSysMLの関係



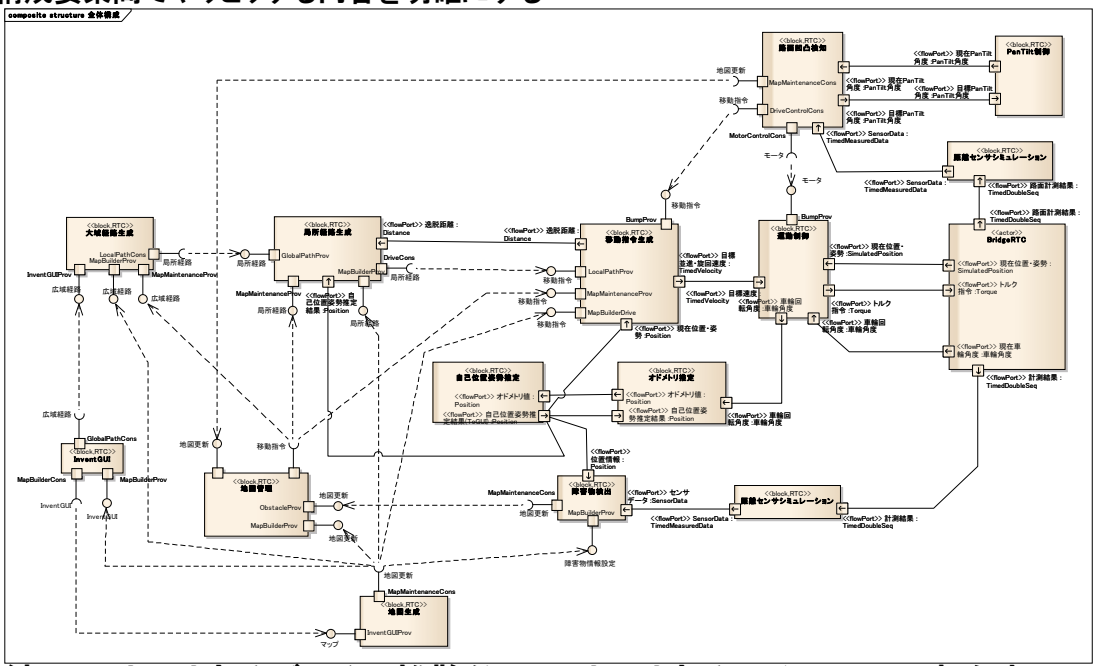






内部ブロック図

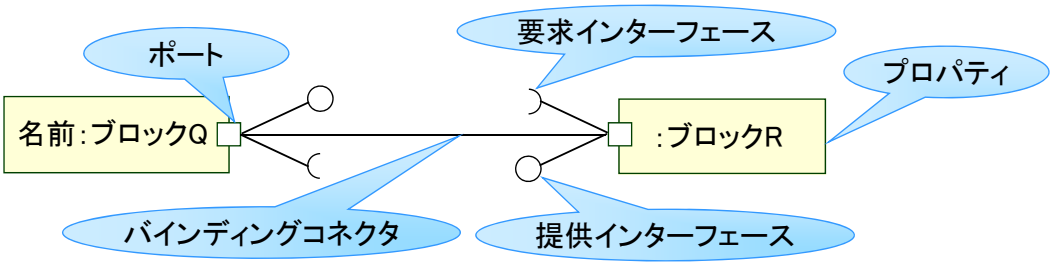
- システムを構成する要素間でやり取りする内容を表現
  - システム構成要素間の接続関係を明確にする
  - 構成要素間でやりとりする内容を明確にする



◆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

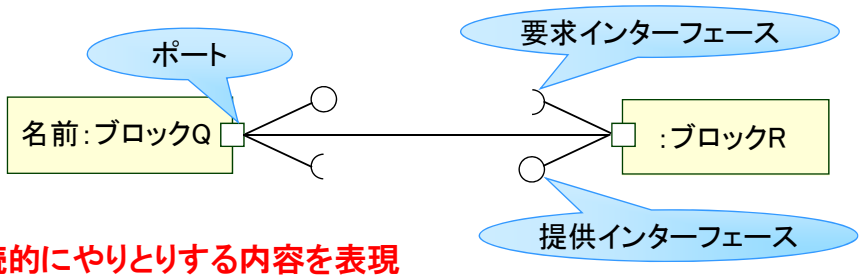
- プロパティ (Property)
  - ブロックに含まれる内部要素を構造的な役割で表現した要素
    - パート(Part)とも呼ばれる
- ポート (Port)
  - プロパティの内部と外部の境界を表現



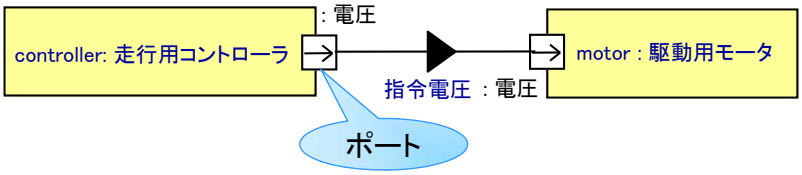
  - 実際に提供/要求している仕様は，インターフェースを用いて表現する
  - 離散的にやりとりする内容(イベントなど)を表現
- バインディングコネクタ (BindingConnector)
  - プロパティ間に何らかの関係や相互作用が存在することを表現

内部ブロック図の構成要素②

- 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



  - 連続的にやりとりする内容を表現



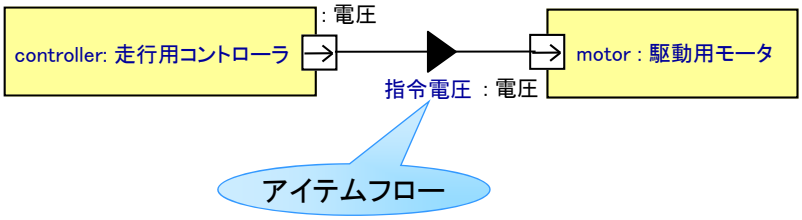
- In(入力)

- Out(出力)

- InOut(入出力)

内部ブロック図の構成要素③

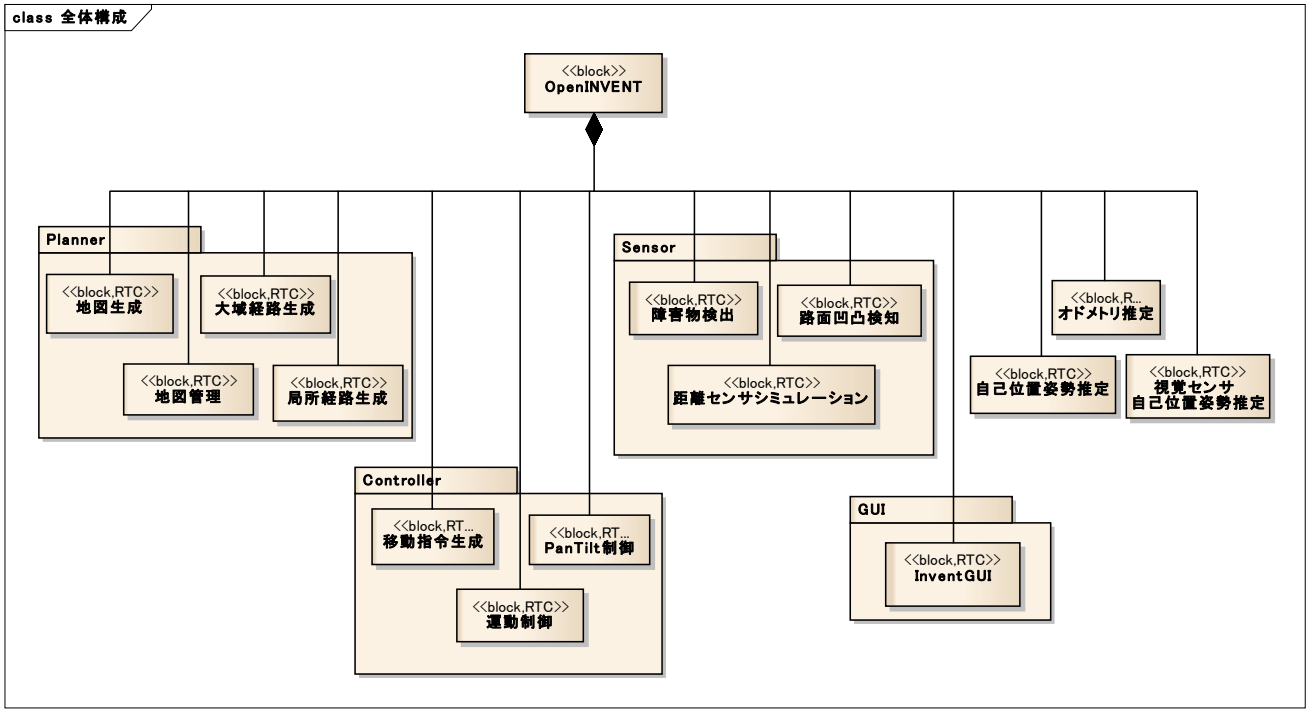
- アイテムフロー (ItemFlow)
  - プロパティ間を実際に流れている要素を表現



- 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

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

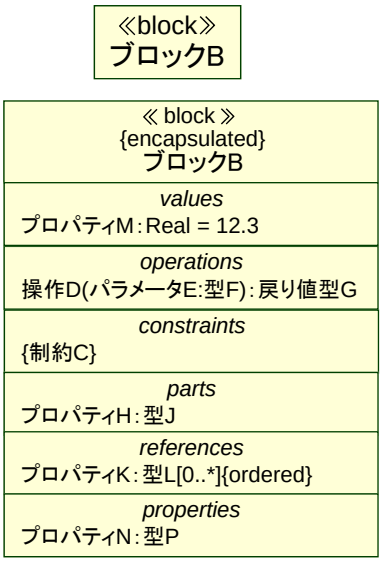
ブロック定義図の構成要素①

- 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

◆ 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>]

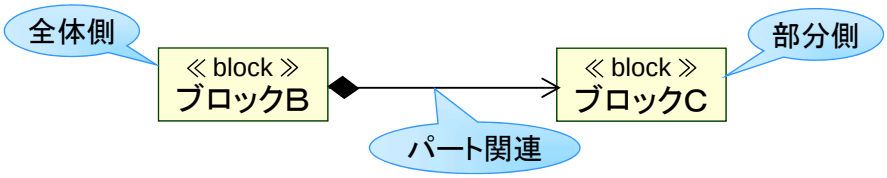
◆ 制約(constraints)
ブロックが満たすべき条件
{ 制約 }



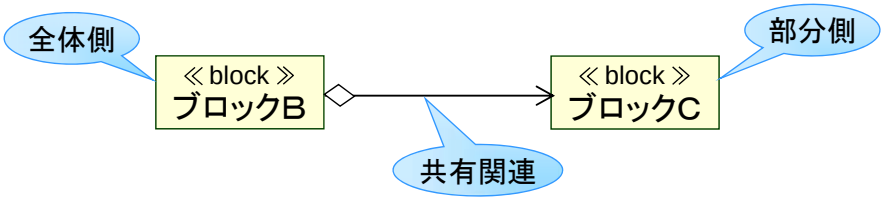
☑ ブロック定義図内で，ただの四角形が出てきた場合にはブロックとみなす

ブロック定義図の構成要素②

- パート関連 (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る
    - 「全体」が削除されると「部分」も削除される
    - UMLの「コンポジション」に相当

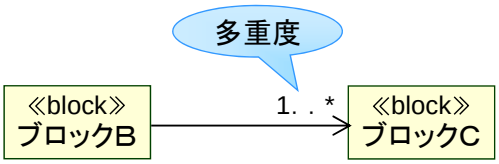


- 共有関連(SharedAssociation)
  - パート関連と同様に，ブロック間に「全体」-「部分」の関係があることを表現
  - 「部分」と「全体」のライフサイクルは異なるため，「部分」単独でも存在できる
    - UMLの「集約」に相当



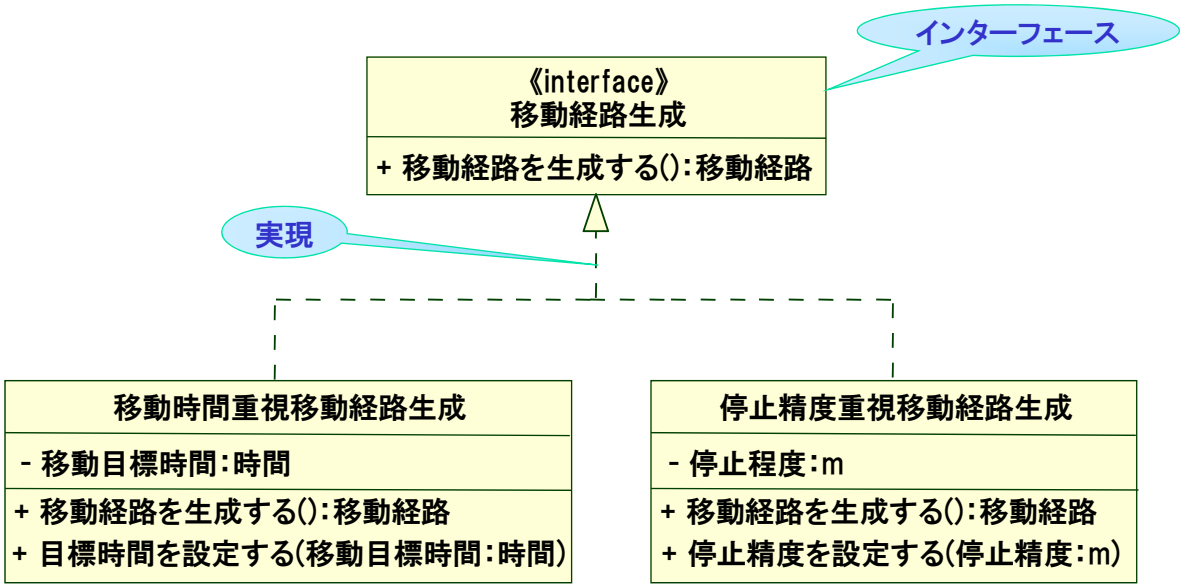
ブロック定義図の構成要素③

- 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



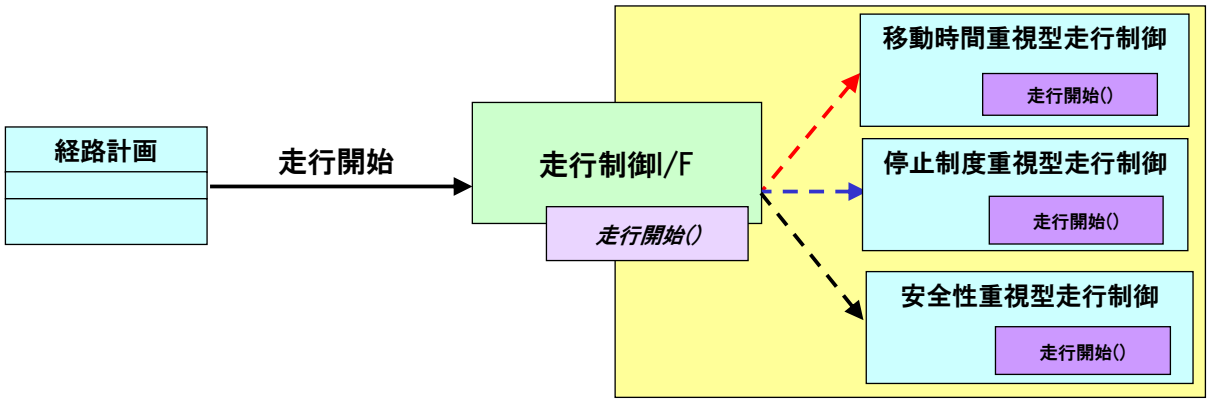
ブロック定義図の構成要素④

- 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



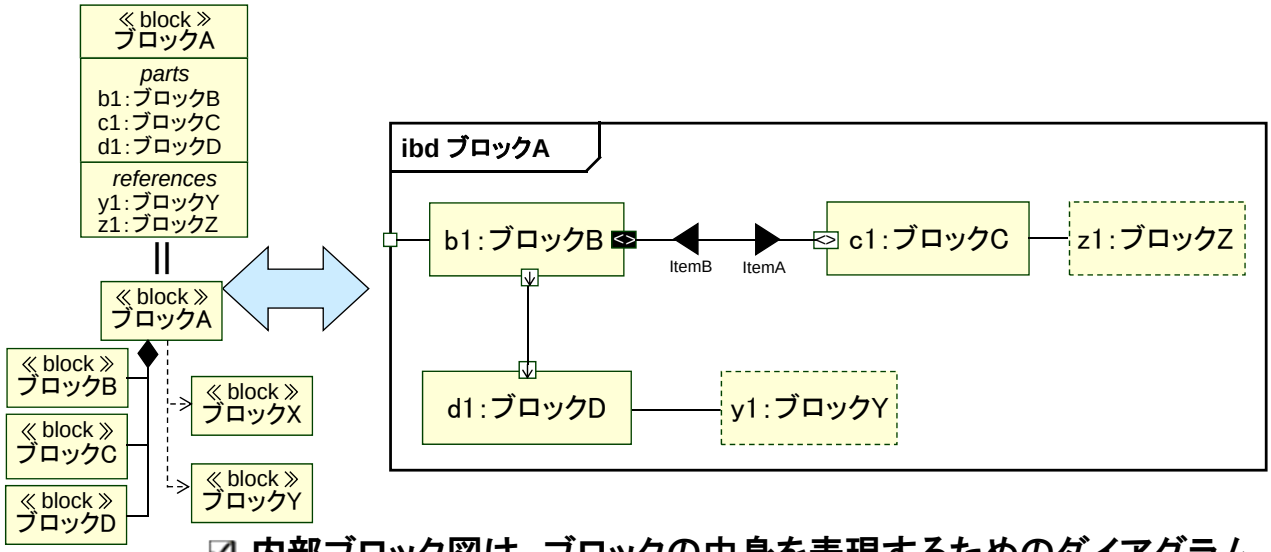
ポリモフィズム(多態性)

- 同一操作であっても振る舞いが変わること
  - 同じ名前の操作であっても，その内容がオブジェクトごとに異なること
  - 異なる操作を一つにまとめ，インターフェースを簡潔にすることができる
  - 利用する側の呼び出し方を変えずに，振る舞いを変える事ができる
  - インターフェースと実装の分離により実現



内部ブロック図とブロックの関係

- Parts：該当ブロックが保持している子要素
  - 枠線を実線で表記,ブロック定義図でパート関連で接続された要素
- References ： 他のブロックが保持している要素を参照
  - 枠線を点線で表記,ブロック定義図で共有関連で接続された要素

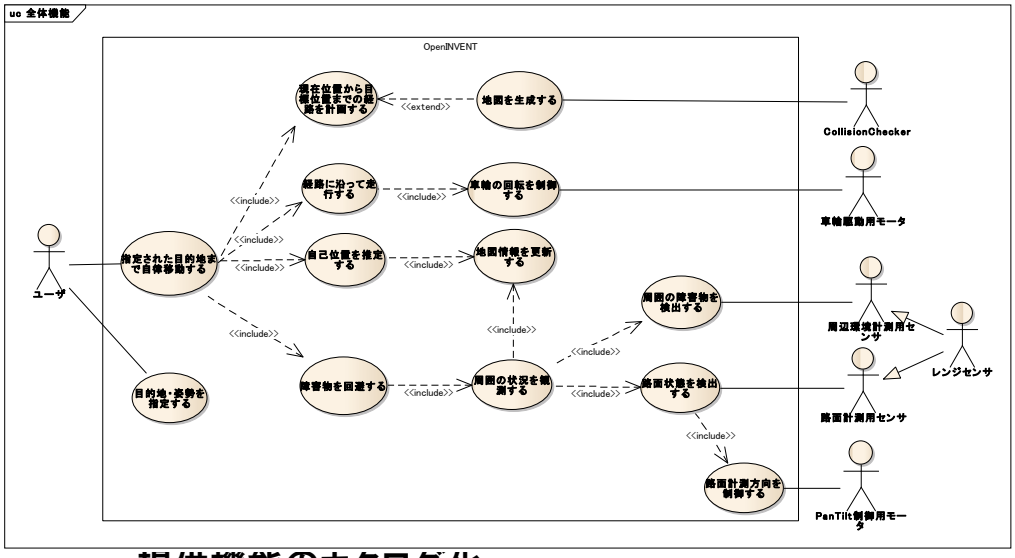


☑ 内部ブロック図は，ブロックの中身を表現するためのダイアグラム

☑ 表現対象のブロックを必ず指定する必要がある

ユースケース図

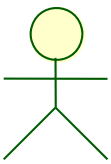
- 対象範囲(システム)が提供する機能の明確化
  - 開発対象の範囲(システム化範囲)を明確にする。
  - 機能とそのユーザの関係を明確にする。
  - 対象範囲(システム)が必要とする外部要素を明確にする。

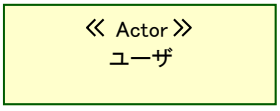


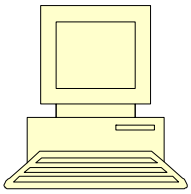
- ◆提供機能のカタログ化
- ◆開発を行う優先順位の決定にも利用される

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。
    - リモートコントローラ、ユーザ　・・・・○ 適切
    - DUALSHOCK2、田中さん　・・・・X 不適切
  - 複数の表記方法がある。


ユーザ



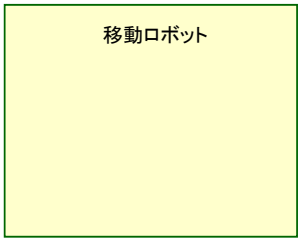

デバッグ用PC

☑ アクタの種類

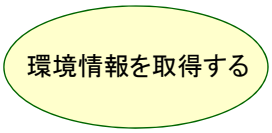
- ▶ プライマリ(主)アクタ
  - ▶ 対象(システム)に対して、自ら相互作用を開始するアクタ
    例)ユーザ、デバッグ用PC　など
- ◆ セカンダリ(副)アクタ
  - ◆ 対象(システム)からの要求に応じて、何らかの機能を提供するアクタ
    例)音声用スピーカ、プリンタ　など

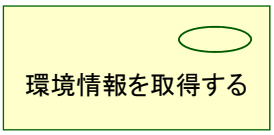
ユースケース図の構成要素②

- 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。



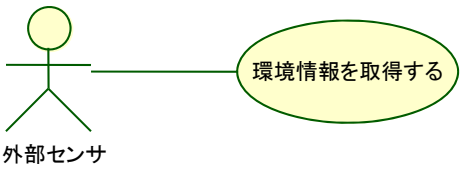
- ユースケース (UseCase)
  - 対象範囲(システム)の外部から見た機能(振る舞い)を表現する。
  - システムの内部処理(粒度の細かい処理)はユースケースではない。



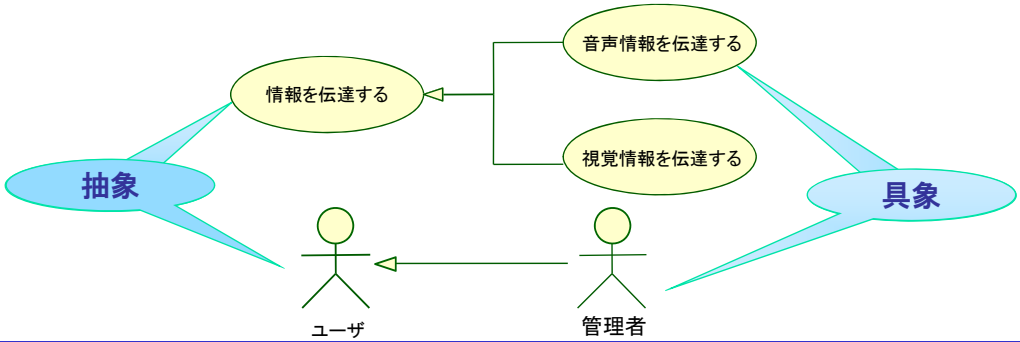


ユースケース図の構成要素③

- 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



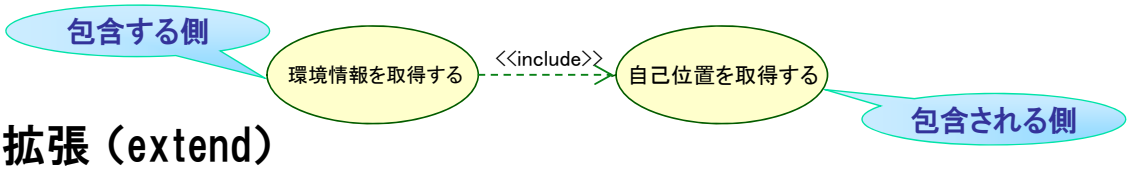
- 汎化 (Generalization)
  - 抽象的なものと具象的なものの関係を表現する。
    - 具象側は抽象側の特性を引き継ぐ
  - ユースケース、アクタの両方で使用できる。



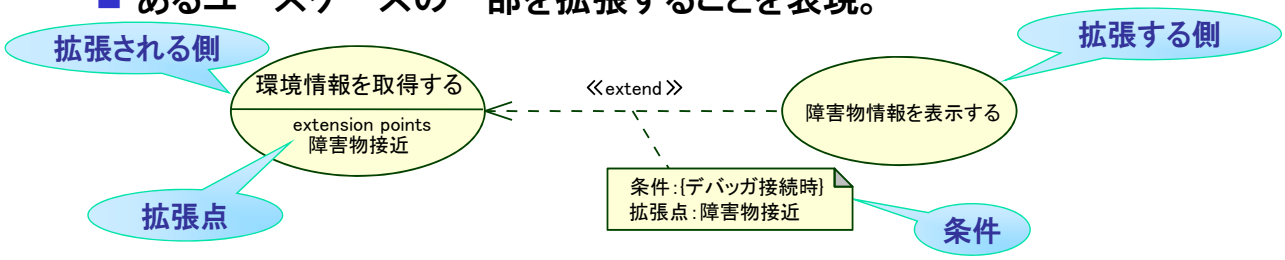
ユースケース図の構成要素④

- 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。



- 拡張（extend）
  - あるユースケースの一部を拡張することを表現。



◆「包含(include)」と「拡張(extend)」の違いについて

・包含：親ユースケース実行時に，必ず子ユースケースが呼び出される

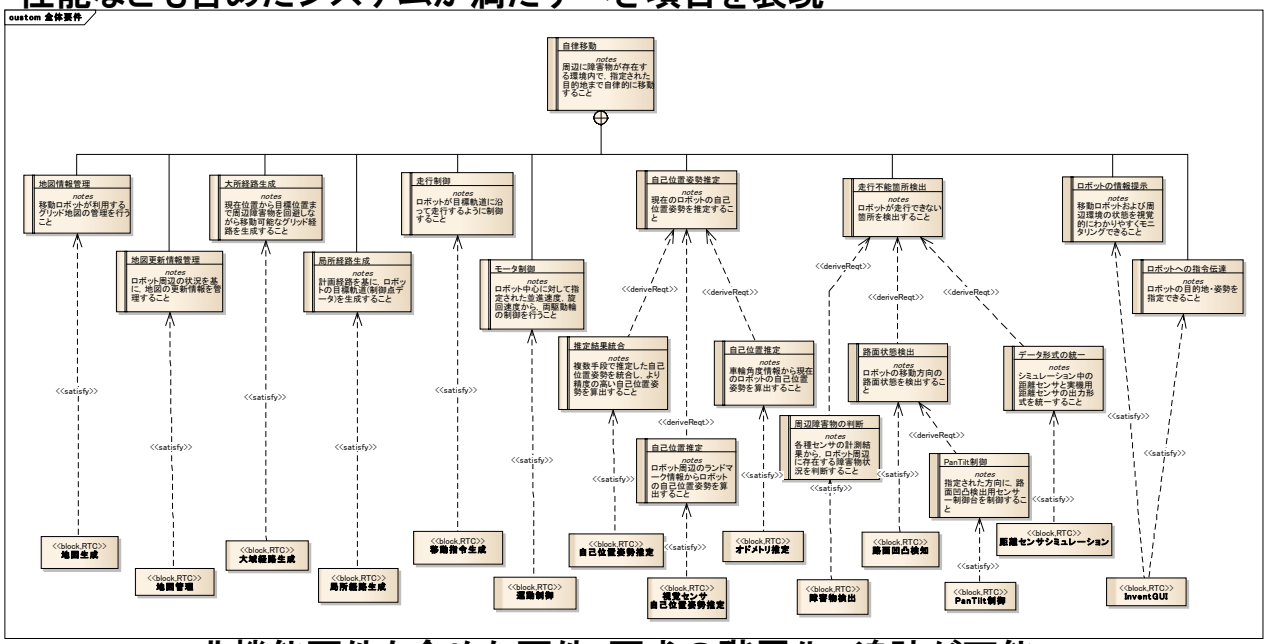
・拡張：親ユースケース実行時に，条件付きで子ユースケースが呼び出される

→呼び出される場所は拡張点で定義

呼び出される条件はノートに記載

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



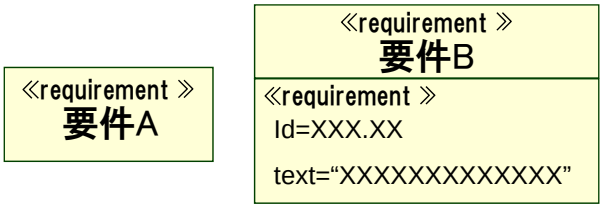
◆非機能要件も含めた要件・要求の階層化，追跡が可能

◆要件・要求の内容については，文章として記述

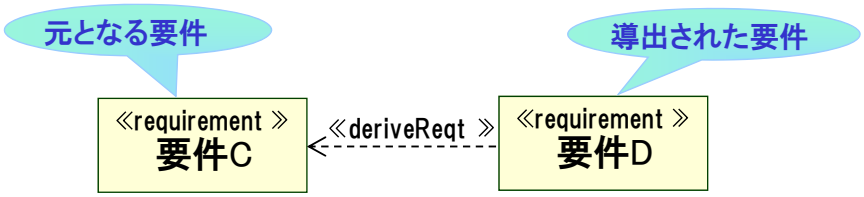
◆表形式での表現も可能

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



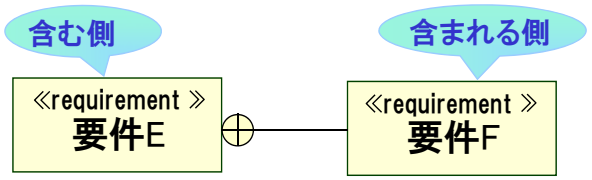
- 導出 (Derive Dependency)
  - ある要件から別の要件が導き出される関係



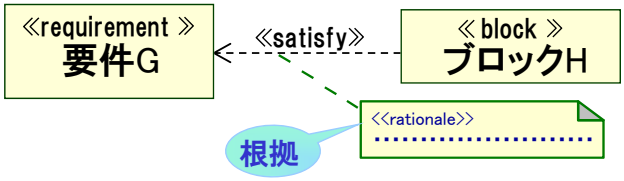
  - ユーザ視点の要件を技術的な要件に分解，整理
  - 抽象的な要件の具体化

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

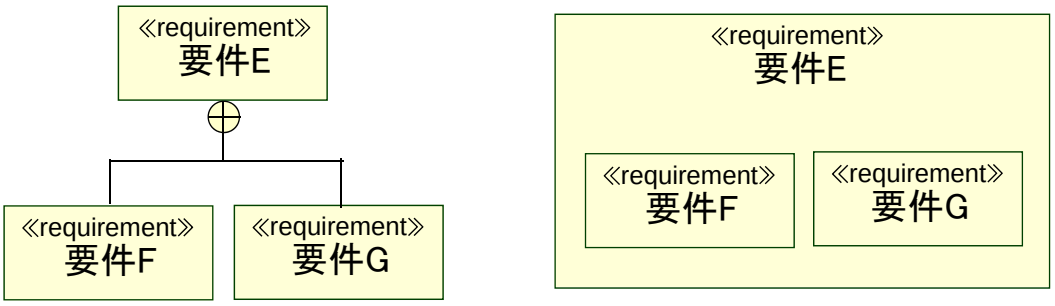


- 根拠 (Rationale)
  - 分析/設計中に行った決定の根拠を記述
    - 各要素間の関係などに付加



要求図作成時の注意点

包含と導出について

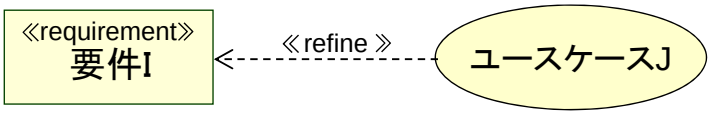


- 包含
  - 含む側，含まれる側が一体として扱われる
  - 他のモデルで要件を再利用する場合，全体をまとめて再利用する形
- 導出
  - 導出元，導出先はそれぞれ別の要件として扱われる
  - 他のモデルで要件を再利用する場合，個々の要件単独で再利用が可能

要求図の構成要素③

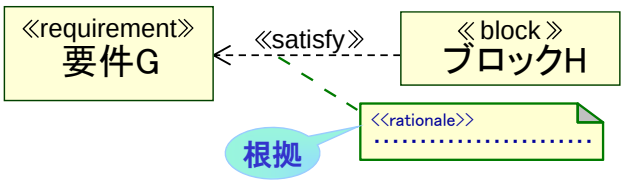
詳細化 (Refine Dependency)

- 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)

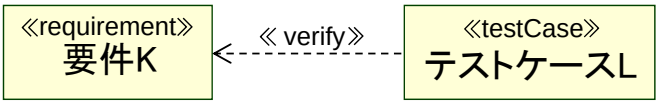
- 要件とその要件を実現する要素の関係



  - 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

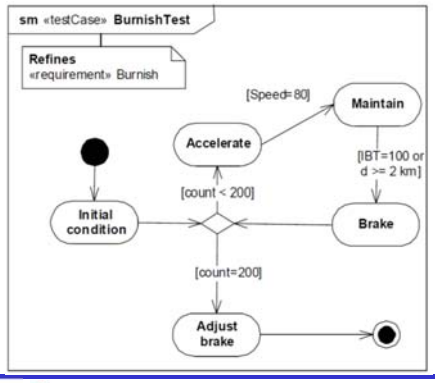
要求図の構成要素④

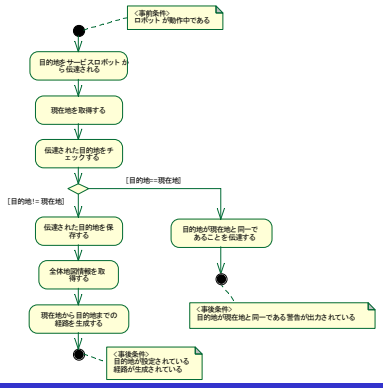
- テストケース（TestCase）
  - ある要件が満たされているかどうかを判断するための手法・手順を表現
    - 要件として定義した内容の確認手順など



- 検証（Verify）
  - 要件とその要件が満たされているか確認するためのテストケースとの間の関係

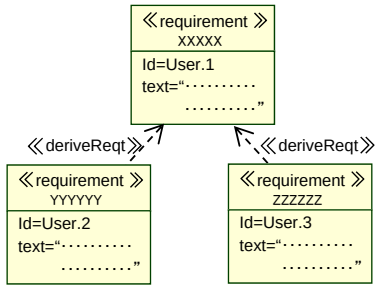
◆実際のテスト手順については，別途詳細化



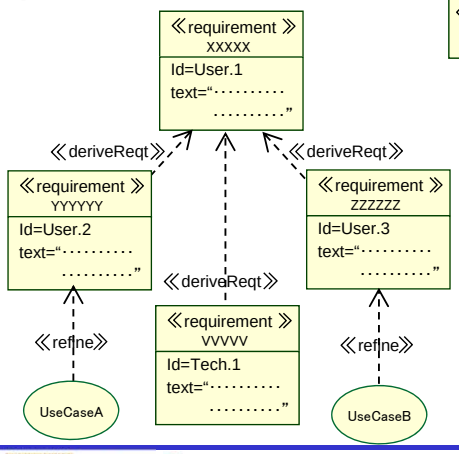


要求図

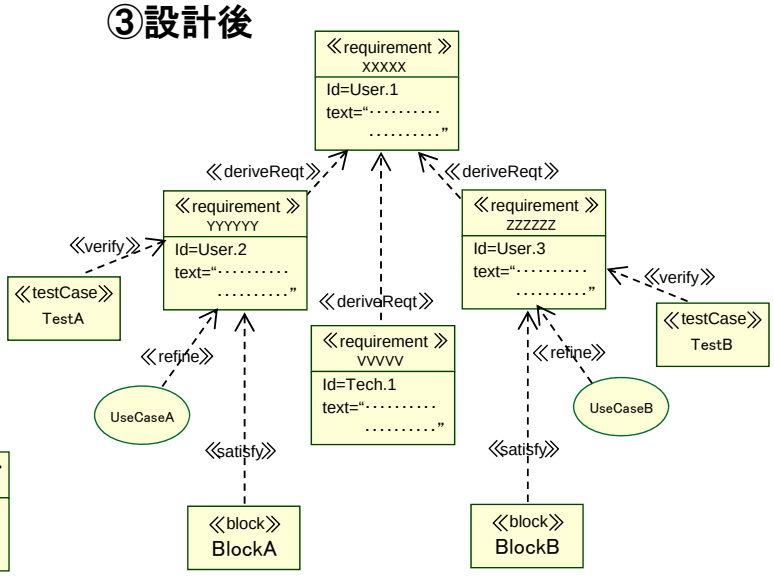
①要求分析時



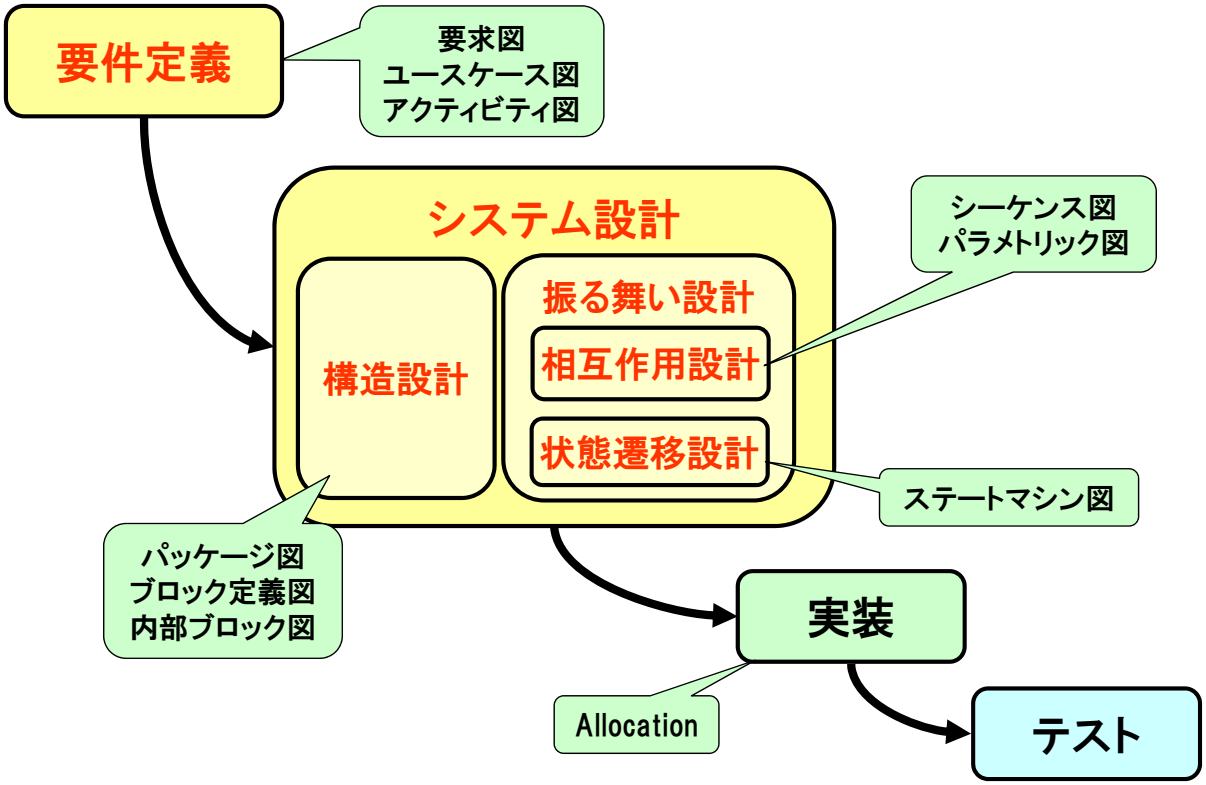
②機能分析後



③設計後

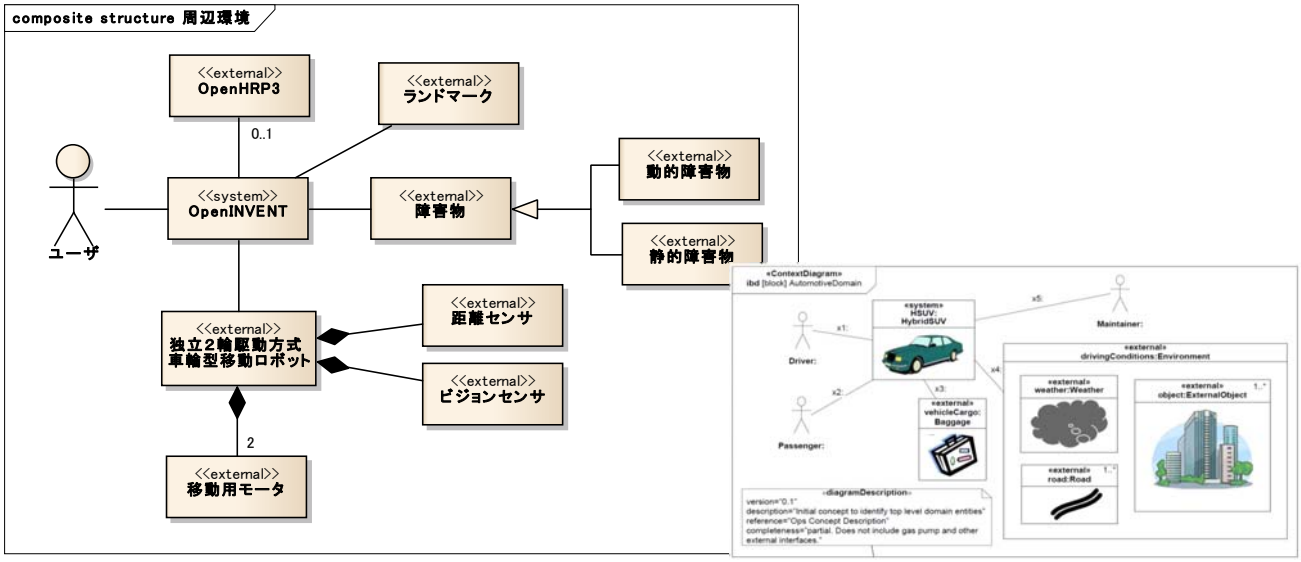


システム開発の流れ



コンテキスト図

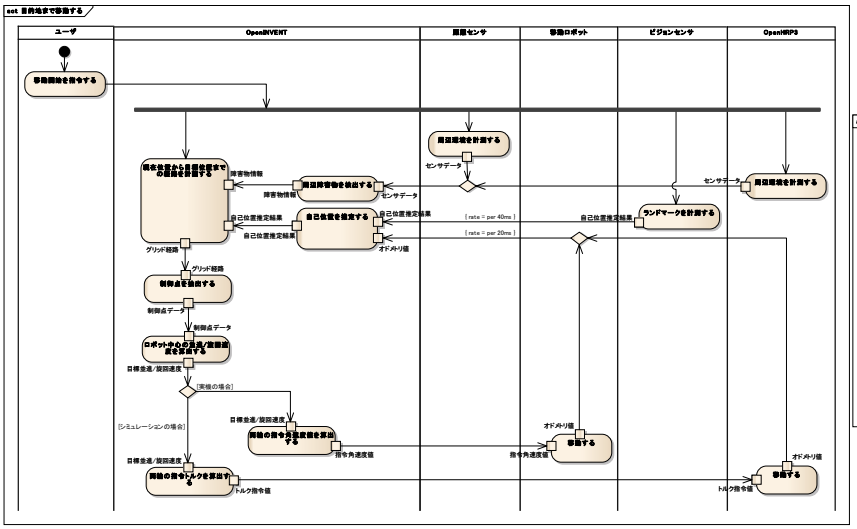
- システムが動作する環境を表現

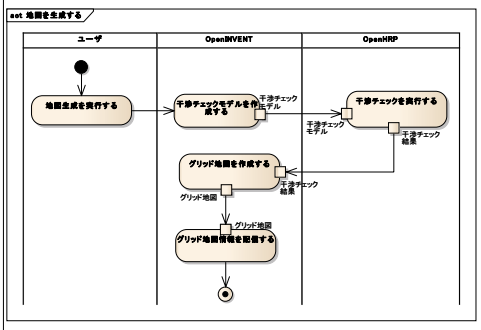


- ◆対象システムと相互作用する要素，影響を与える要素を抽出する
- ◆以降の分析/設計作業にて，検討する必要のある要素を洗い出す
  - ◆ 記述していない要素については，検討対象外
    - ☑ SysML仕様内で明示的に定義されたダイアグラムではない

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

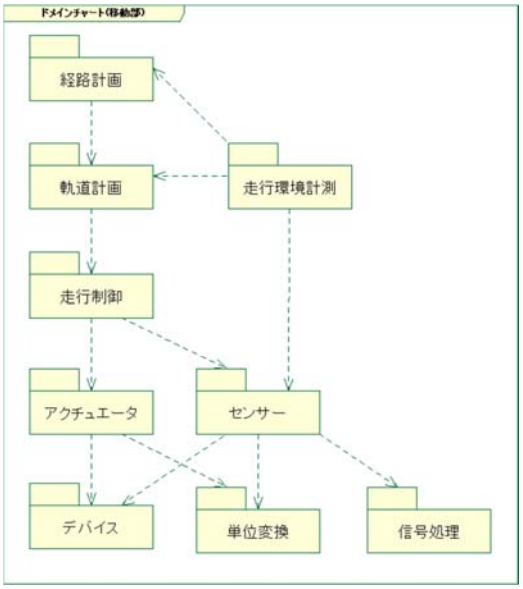




- ◆処理の実行順序，各処理の実行主体を表現
- ◆各処理間でやり取りするデータを表現
- ◆粒度に応じて処理を階層化して表現することも可能

パッケージ図

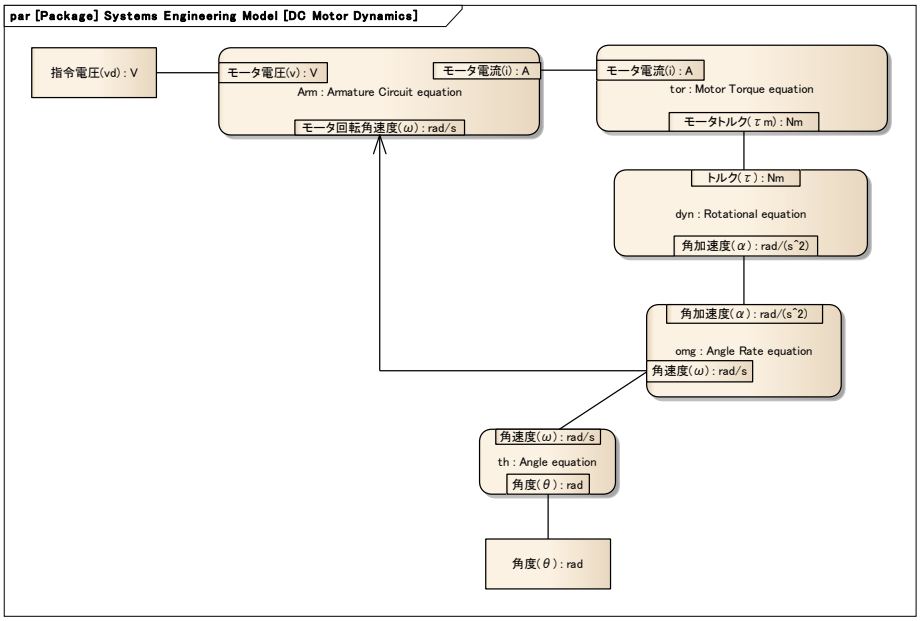
- 静的なおおまかな構造および構造間の関係を表現
  - パッケージは再利用を行う際の一番大きな単位であり，モデル要素をグループ化する単位となる



- ☑ 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

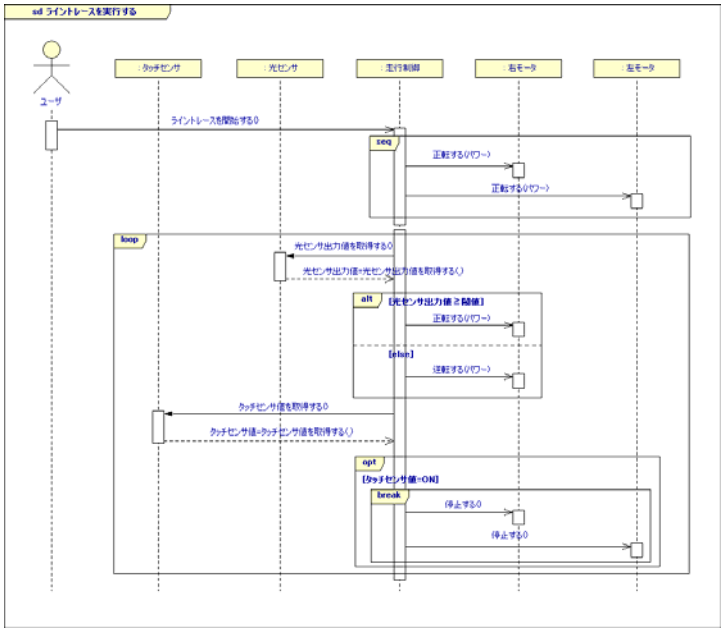
- システムに付随する制約の関係を表現
  - 各ブロックで定義された制約間の関係を表現
  - ブロック線図と似た内容を表現することが可能



☑ 内部ブロック図の1種

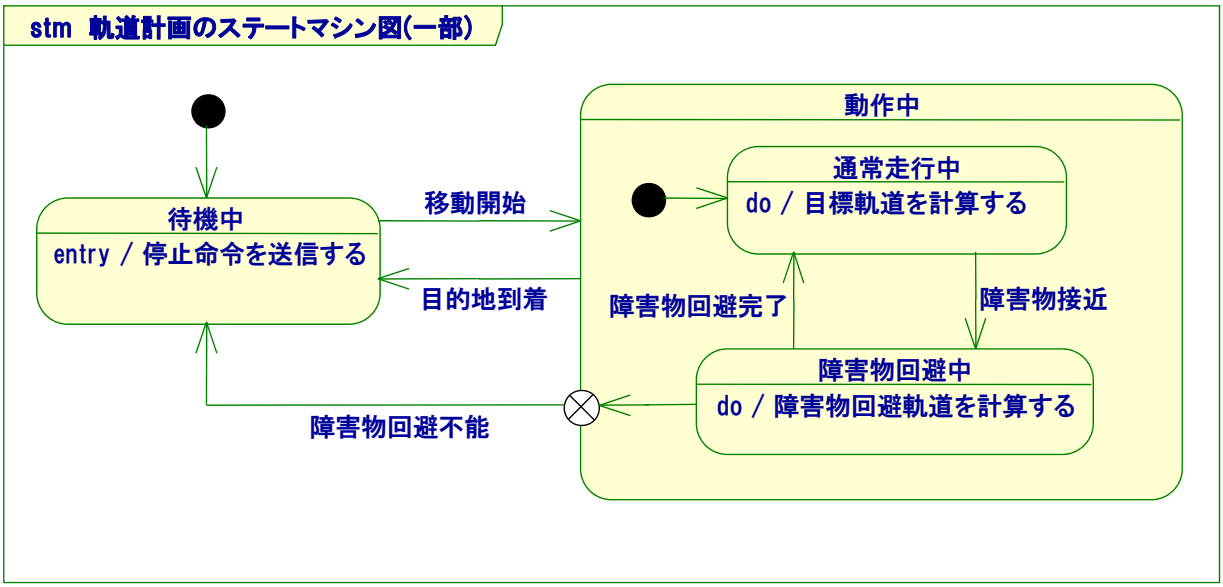
シーケンス図

- 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



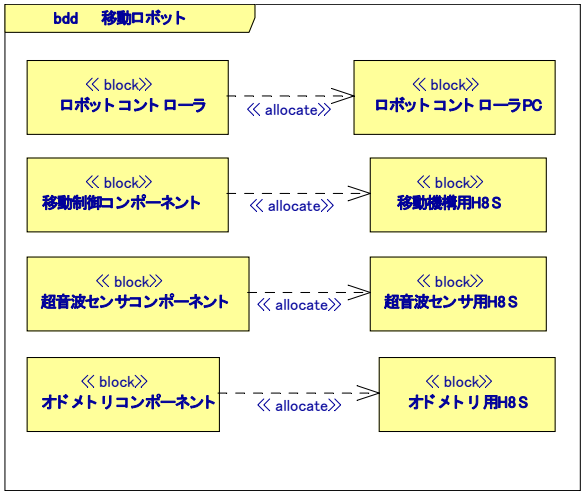
ステートマシン図

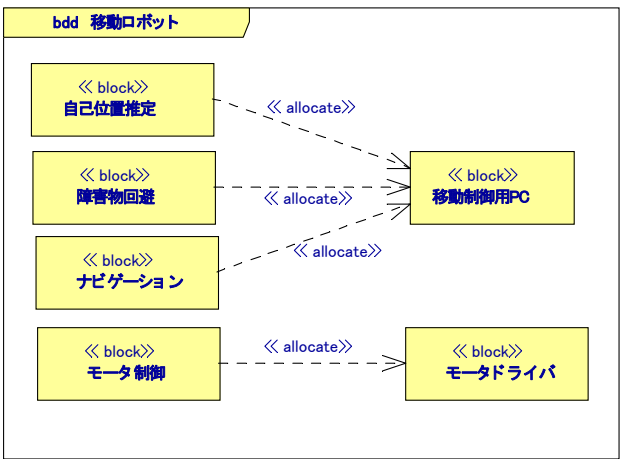
- ある要素が生成されてから破棄されるまでの状態の移り変わりを表現
  - 状態が変化する流れと，そのきっかけを表現可能
  - 各状態で実行する振る舞い(アクティビティ)も表現可能



アロケーション

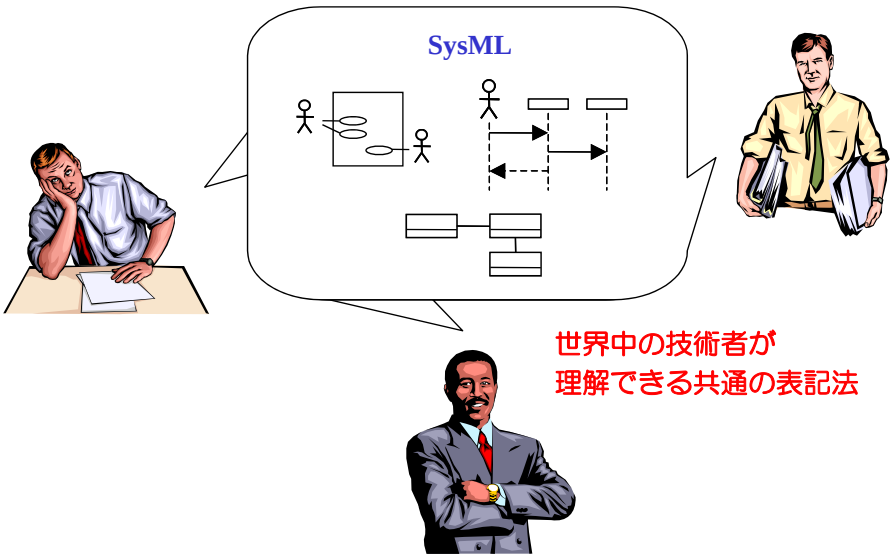
- 要素間の対応関係を表現
  - ソフトウェアコンポーネントのノード配置，アクティビティのブロックへの振り分けなどを表現
  - 「機能」と「構造」の分離を実現





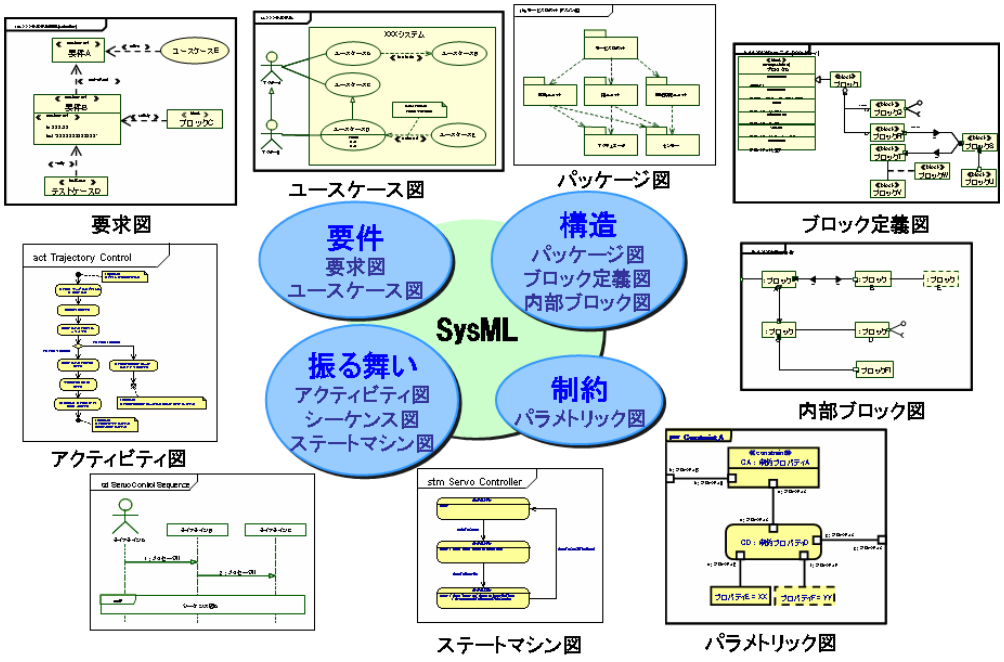
SysMLの特徴

- 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



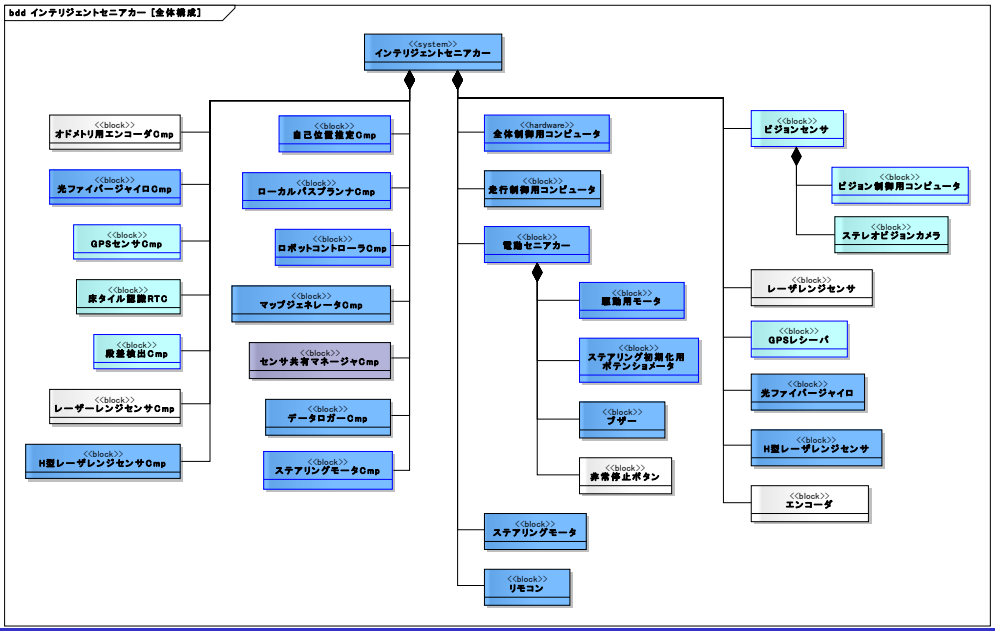
SysMLの特徴

- 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



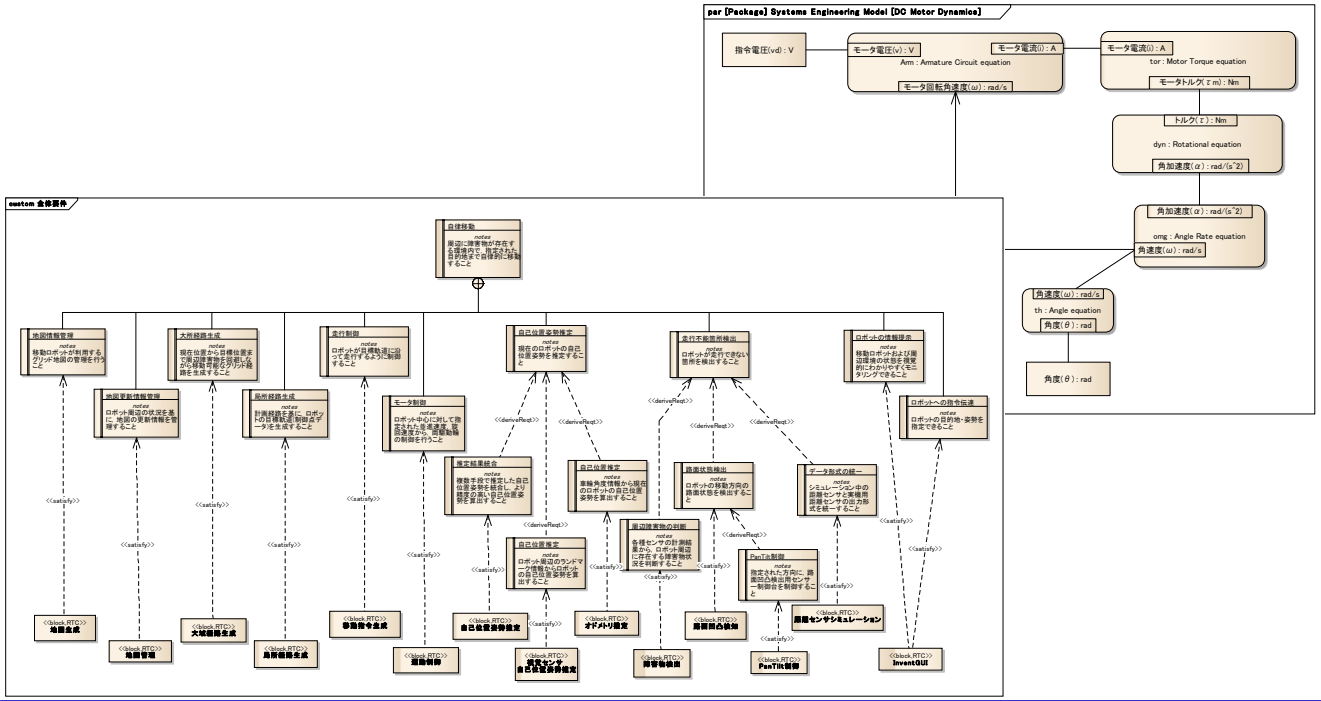
SysMLの特徴

- 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



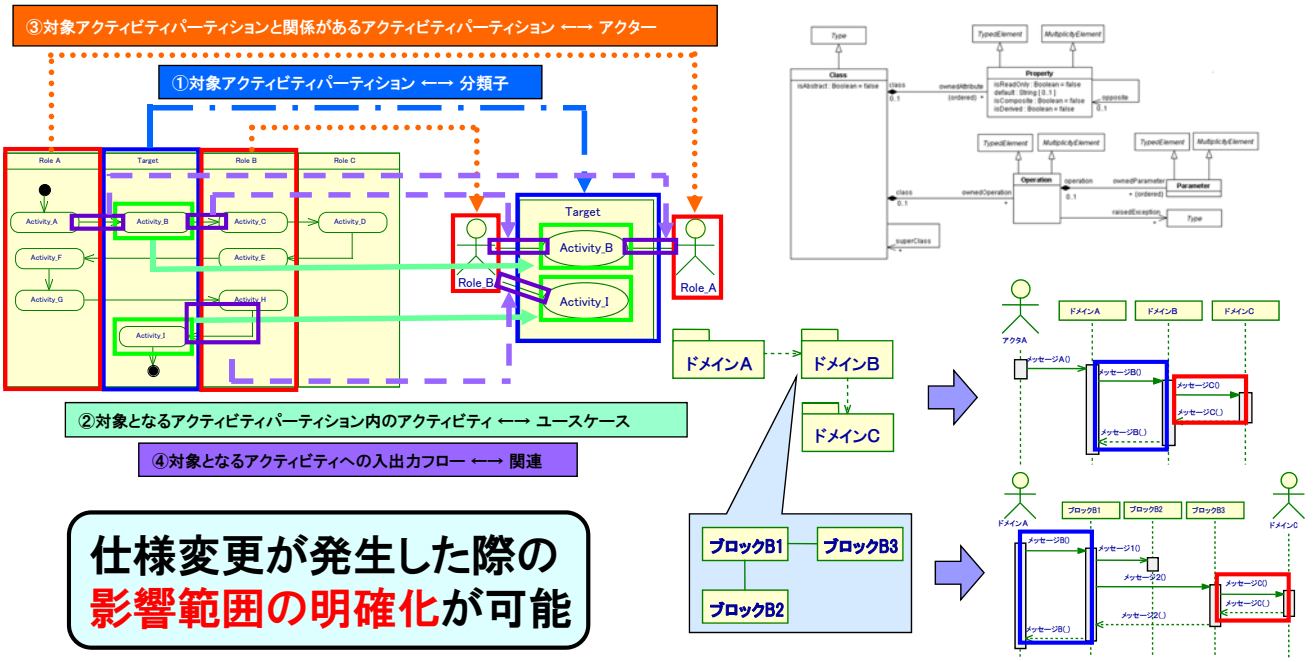
SysMLの特徴

- 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

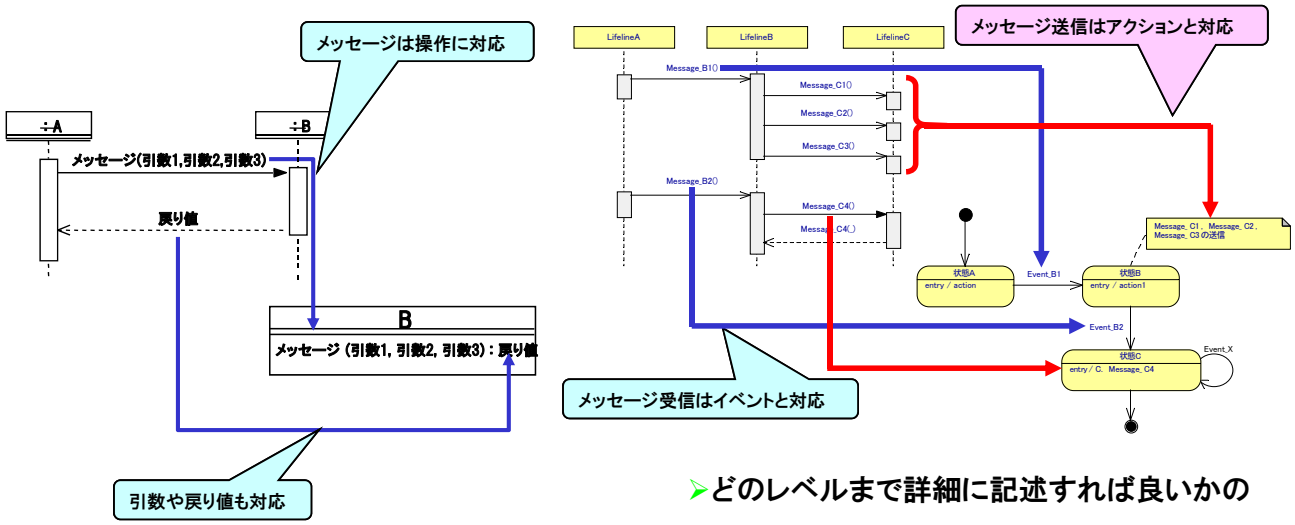
何を目的としてモデルを作成しているのか？
何の検討を行いたいのか？何を明確に整理したいのか？
を常に意識する事が重要

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

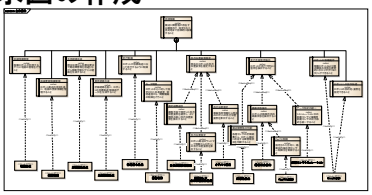
- ダイアグラム間，工程間の繋がりを意識する
  - ダイアグラム間の関係はある程度決まっている
  - 成果物，モデルを後工程でどのように利用するかを意識する



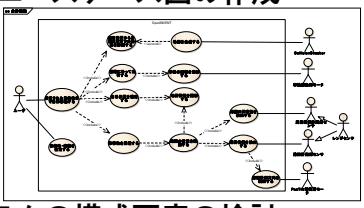
➢どのレベルまで詳細に記述すれば良いかの判断材料にもなる

RTSystem開発時のSysML利用法

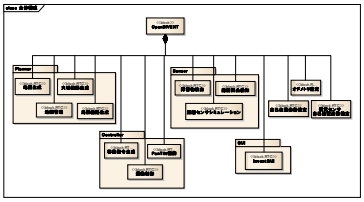
01. システムが「何を」実現すべきなのかを検討
  - 要求図の作成



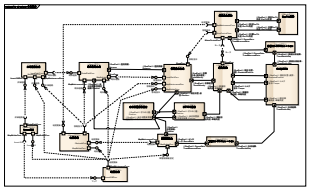
02. システムが実現すべき機能の検討
  - ユースケース図の作成



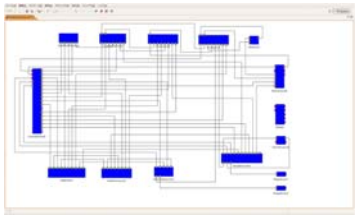
03. システムの構成要素の検討
  - ブロック定義図の作成

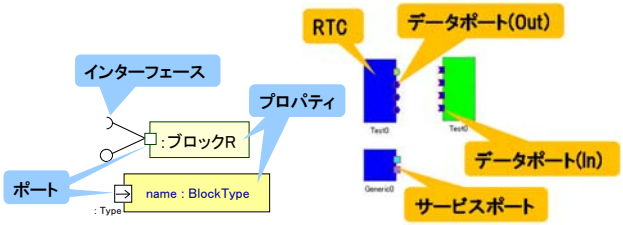


04. 構成要素間でやり取りするデータの検討
  - 内部ブロック図の作成



05. RTC,RTSystemへのマッピング





まとめ

- SysMLとは？
  - オブジェクト指向の考え方に基づき，システム全体をモデリングするための表記法
    - あくまでも「表記法」であるため，使い方に注意・工夫が必要
    - ハードウェア，ソフトウェア両方を含めた表現が可能
      - ハードウェア，ソフトウェアの切り分けを検討する際にも利用可能
  - 様々な視点からシステムを表現することが可能
    - 4種類9つのダイアグラムを用意
    - 上位のユーザ要求から，下位の制御系設計まで対応可能

◆全てのダイアグラムを必ず利用しないといけない訳ではない
  ◆ 必要に応じて部分的な導入・利用も可能

RTミドルウェア SUMMER CAMP 2014

モデリング講習会

日時：2013年7月25日(金)
場所：早稲田大学 理工キャンパス55号館
S棟2階第3会議室

RT MIDDLEWARE
RT-MIDDLEWARE SUMMER CAMP 2014