要求の仕様化に関する課題、プロセス及び手法

～必ずしも“厳密さ”ではなく
“意図”したものが実現できる要求定義を求めた活動の成果報告書～

2015年3月20日
安全性向上委員会

一般社団法人
組込みシステム技術協会
Japan Embedded Systems Technology Association

はじめに

- 安全性向上委員会は、2012年から3年間、要求の仕様化に関して、その課題、プロセス及び手法を調査する活動を行いました。この報告書は、その成果をまとめたものです。
- 当委員会は、情報セキュリティと機能安全をテーマとして活動していますが、セキュリティも安全も、その要求定義が曖昧では土台から崩れると気づき、この活動に着手しました。
- 活動期間中に、組込み総合技術展などで都度、概略報告を行ってきました。この報告書は、その発表資料に手を加え、要約解説を追加して、次のように構成されています：
  - 追加した要約解説
  - 発表資料の完全版

安全性向上委員会 SSQ-WG

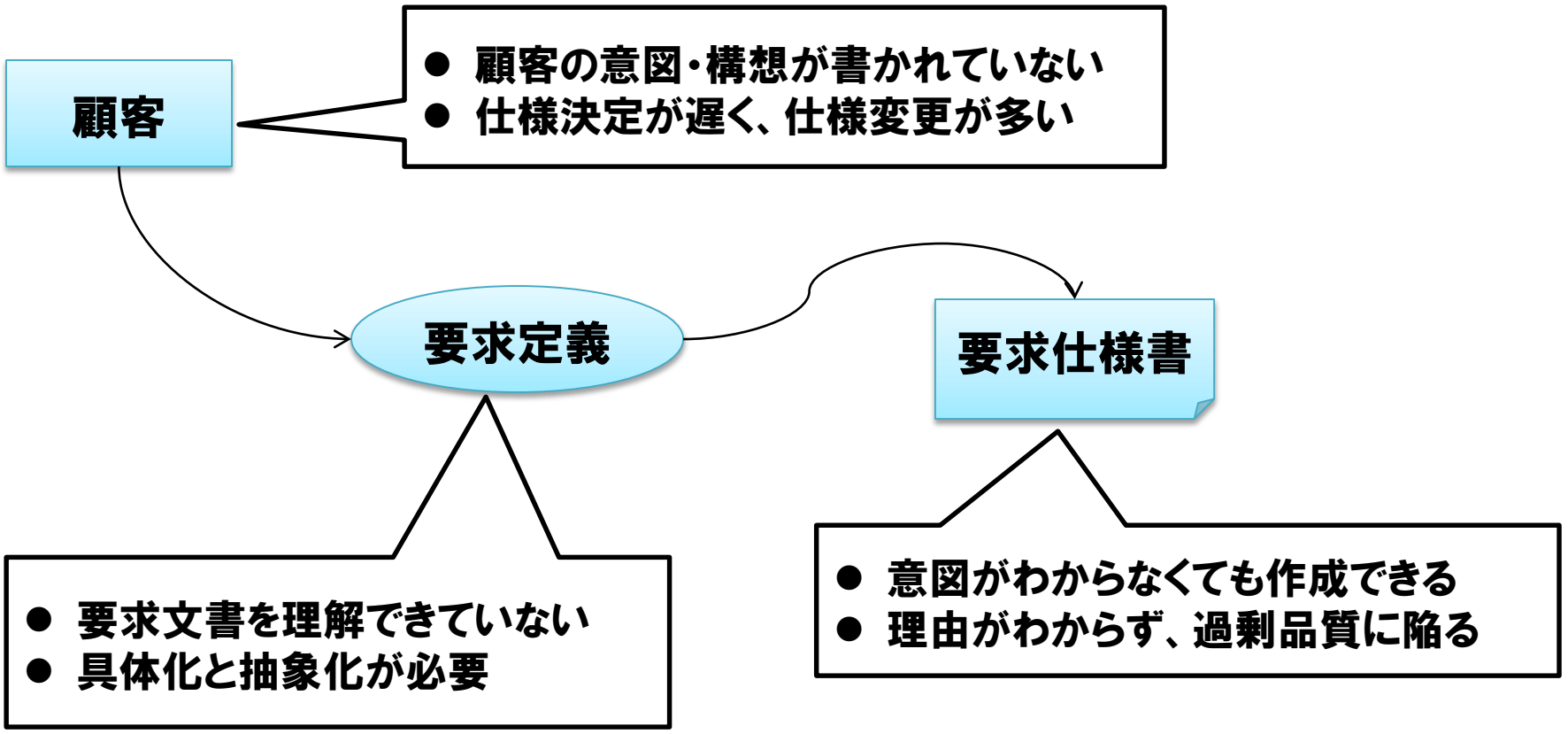
目次

1.要求の仕様化に関する課題

要約

- 要求の仕様化に関する開発現場における課題を、委員各社の現状を聞くことによって、引き出した。その結果は、「開発現場における課題まとめ」を参照。
- 興味深い物として、次の点を挙げる:
  - 顧客との関係では、顧客の意図や構想が示されないが、それでもソフトウェアは作成できてしまうという課題があった。仕様の間違いに気づきにくいことが問題。
  - 要求定義という作業では、具体化するという作業と抽象化するという作業の両方が必要になるという課題があった。つまり、設計者は、要求を具体的に分析、検討し、その結果を顧客が関心を持つ程度に抽象化して記述しなければならない。
- 各社からの聞き取り内容は、「開発現場における課題-1」から「-3」までを参照。

開発現場における課題まとめ



開発現場における課題-1

- 顧客仕様が詳細まで記載されているため、顧客の意図・構想が分からなくてもソフトがそれなりに作成出来てしまう。
  - 構想までは提示されないことも多い。
  - 制御意図の理解が困難、仕様の問題に気が付きにくい。
- 要求の背景や理由などがわからず、どうしても過剰品質に陥る。

開発現場における課題-2

- 仕様決定が遅く、仕様変更が多く、テストが十分できない。
  - 検証が設計段階できればよいが、しかし、
  - 使ってからでないと、悪い副作用が見えない。
- 発注者からの要求文書を十分には理解できていない。
  - それを基に作成する外部仕様書が不十分なものになる。
  - 機能の組合せに関して漏れが生ずる。
  - 対策:
    - 要求・仕様確認の徹底(署名付きの議事録)。
    - 要求把握スキル向上のための教育、ツール導入。

開発現場における課題-3

- 抽象的な要求を具体的に検討し、その結果を抽象化して顧客に提示することが難しい。
  - 顧客とは抽象化された内容(what,why)でコミュニケーションし、
  - 開発者とは具体化された内容(how)で。

2.目指す要求の仕様化プロセス像

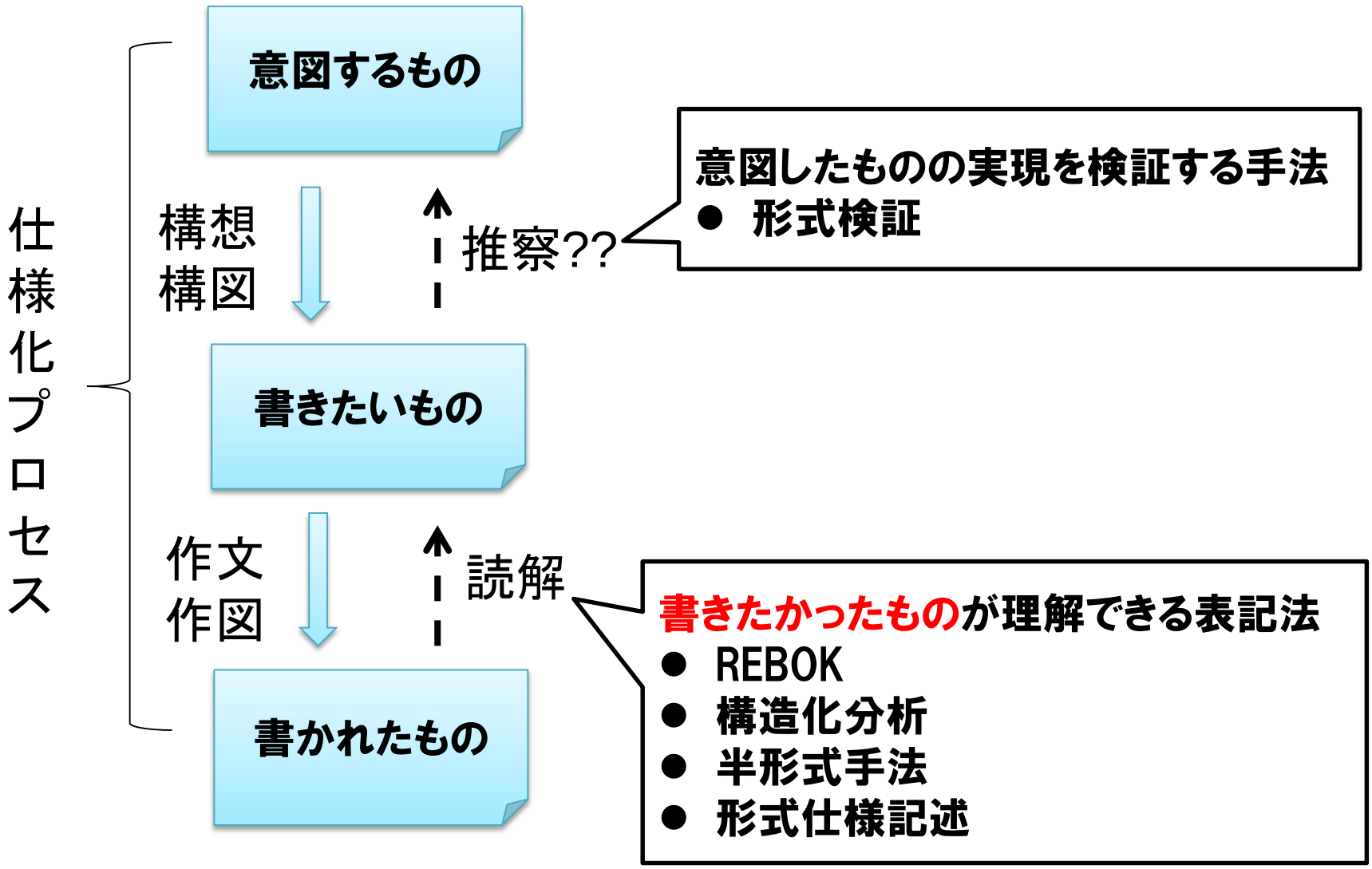
要約－1

- 要求を仕様化するプロセスは不可逆的である。書かれたものから書きたいものを、さらに意図するものを再現することは容易ではない。その再現を容易にするためには、仕様化プロセスに何らかの工夫が必要となる(「不可逆的な仕様化プロセス」を参照)。
- 発注者と受託者が、その要求仕様に関して合意に至るためには、双方が役割を果たし、決められた手順を踏むしかない。目指すべき手順は次のようになる:
    - 発注者が自然言語又は非形式的な表記法で要求を記述する。
    - 受託者が半形式手法又は形式手法を駆使して要求を分析する。
    - その結果を自然言語、半形式手法、又は形式手法で要求仕様として記述する。
    - 発注者は選択された手法で記述された要求仕様を理解し、確認する
- このプロセスがうまくいくためには、双方に半形式手法又は形式手法を読み書きできるスキルが必要になる(「仕様化プロセスの目標像」、を参照)。

要約-2

- 要求の仕様化は、発注者と受託者による共同作業であり、手順が進むに伴って仕様化のレベルが高まる。このプロセスにおいて、双方は次の役割を果たす(「発注者と受託者による共同プロセス」、「要求仕様のレベル」を参照):
  - イニシエータ:要求を獲得し、記述する(発注者)
  - アナリスト:要求を分析し、システム仕様を記述する(双方)
  - デザイナ:ソフトウェア仕様を記述する(受託者)
- 開発プロセスにおいて、イニシエータは発注者として要求を記述し、アナリストは共同作業として、システムの要求分析とアーキテクチャ設計を担い、デザイナは受託者としてソフトウェアの要求分析とアーキテクチャ設計を担う(「SSQエキスパート」を参照)。
- SSQエキスパートには役割と、タスク、スキル、知識が求められ、それを確実にするため、資格認定が必要となる(「SSQイニシエータ像」、「SSQアナリスト像」、「SSQデザイナ像」、「SSQエキスパート資格認定」を参照)。

不可逆的な仕様化プロセス

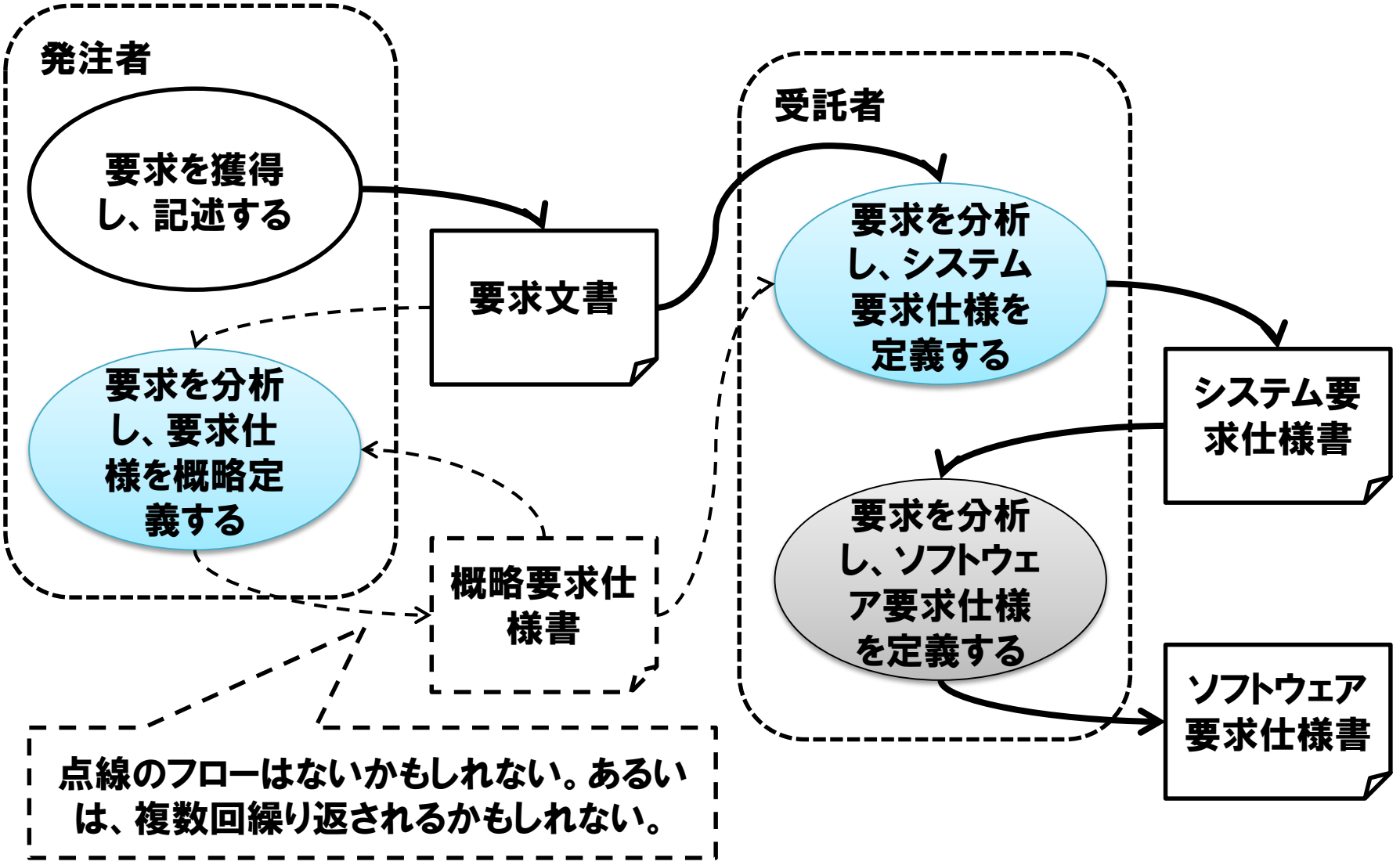


仕様化プロセスの目標像

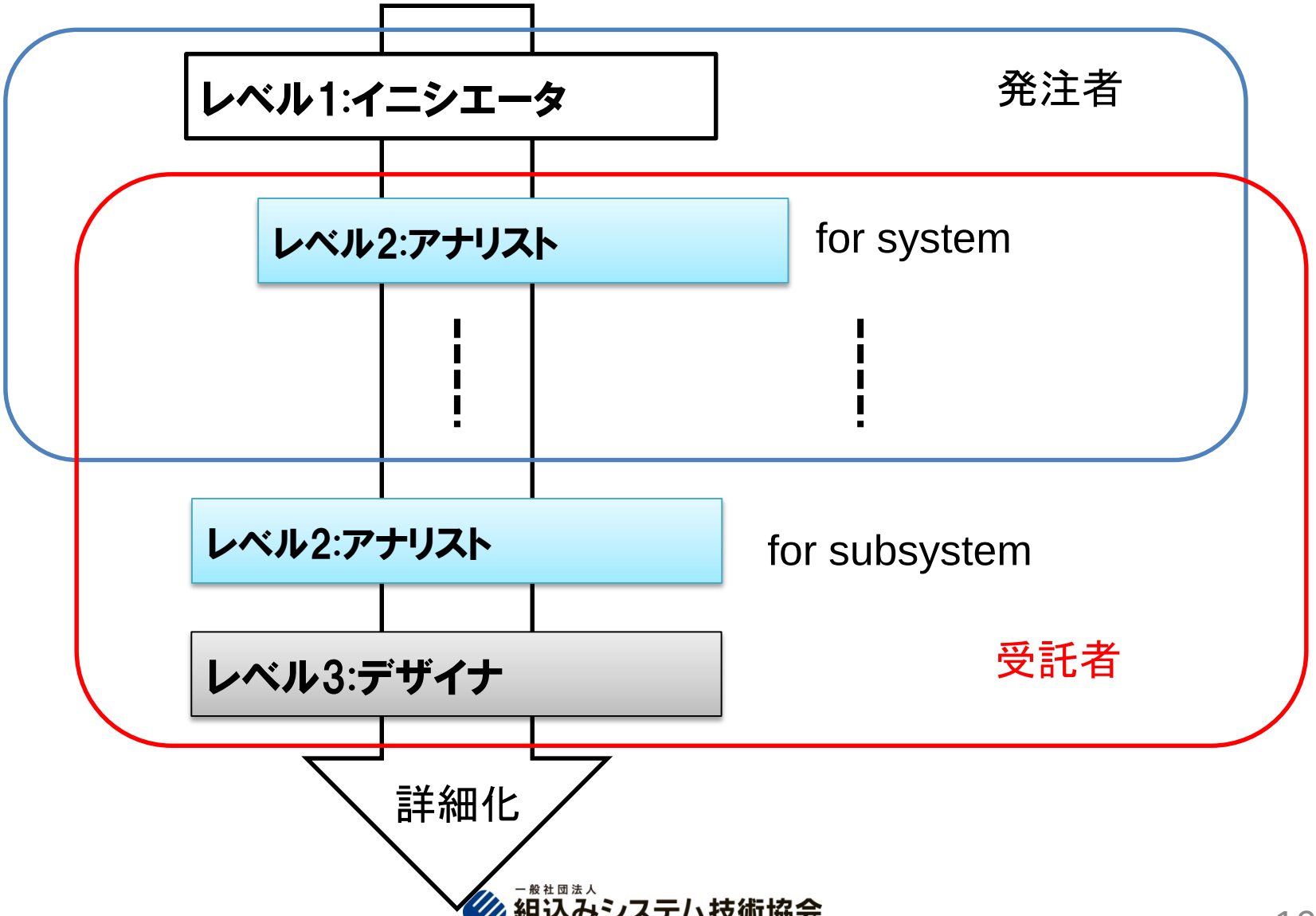
手順	作業項目		表記法とスキル			
	発注者	受託者	自然言語	非形式手法	半形式手法	形式手法
1	要求を記述		W	W		
2		要求分析	R&W	R	W	W
3		仕様書作成	W		W	W
4	仕様を確認		R		R	R
5	要求仕様を合意					

スキルの表記は、W: 書く能力、R: 読む能力

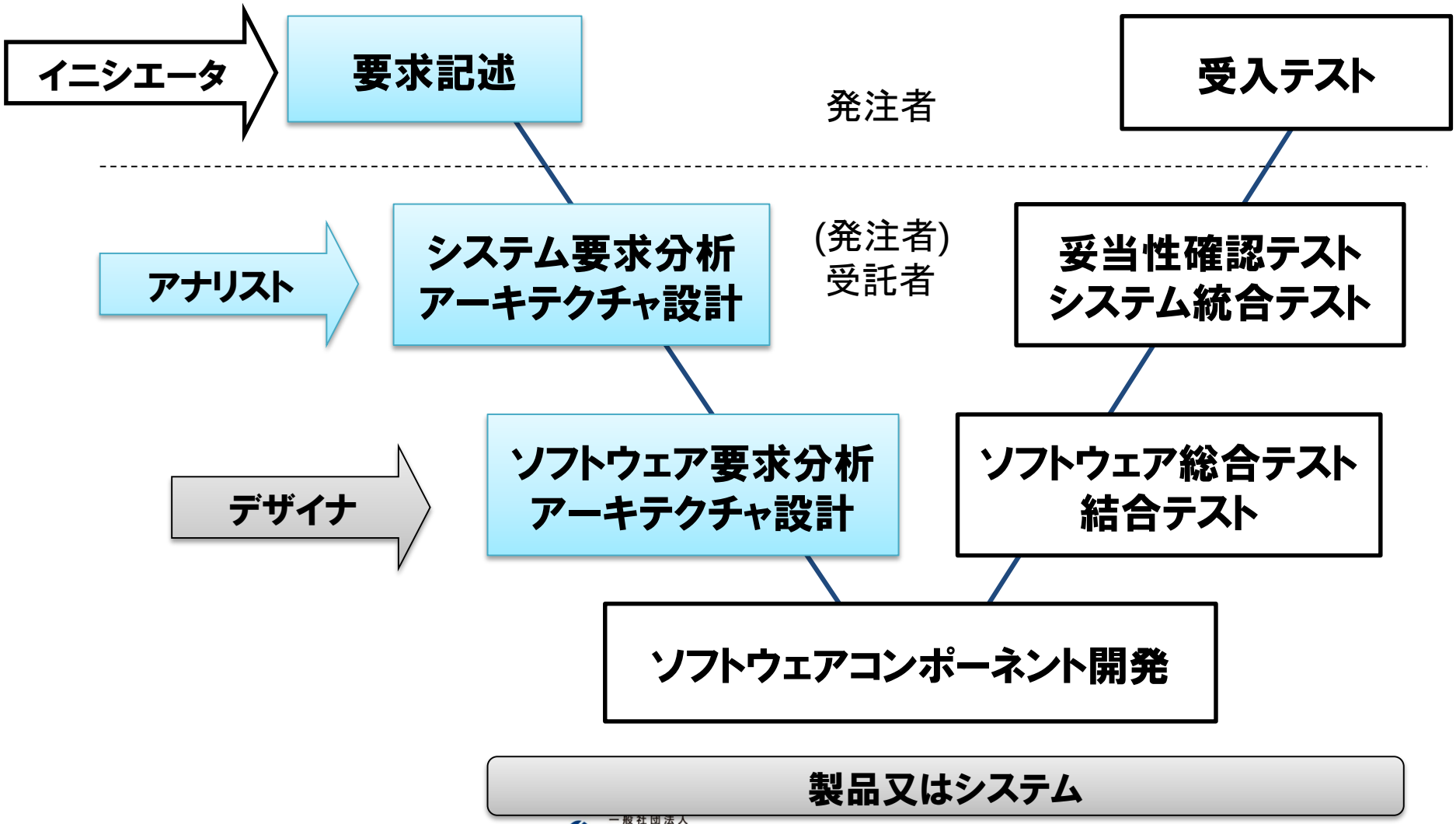
発注者と受託者による共同プロセス



要求仕様化のレベル



SSQエキスパート



SSQイニシエータ像

役割	高度な安全性又は情報セキュリティが求められる製品又はシステムを要求する。
	製品又はシステムに対する安全性又は情報セキュリティに関する要求文書を作成し、それに基づく要求仕様書を理解し、確認する。
タスク	● 製品開発
スキル	● 要求文書を自然言語で作成できる。 ● 半形式手法又は形式手法で書かれた要求仕様書を理解し、確認できる。
知識	● 機能安全に関する基本知識 ● 情報セキュリティに関する基本知識 ● 半形式手法又は形式手法に関する基本知識

備考:共通キャリアスキルフレームワーク(CCSF)を参考にしている。

SSQアナリスト像

役割	高度な安全性又は情報セキュリティが求められる製品又はシステムを開発する。
	製品又はシステムに対する安全性又は情報セキュリティに関する要求を分析し、その要求仕様を定義する。
タスク	● システム要求分析 ● システムアーキテクチャ設計
スキル	● 安全性又は情報セキュリティに関するシステム要求仕様を定義できる。 ● 安全性又は情報セキュリティに関するハザード又は脅威分析ができる。 ● ハザード又は脅威に対して安全度を分析し、評価することができる。 ● 半形式手法又は形式手法を用いて要求仕様書を作成できる。
知識	● 機能安全に関する実践知識 ● 情報セキュリティに関する実践知識 ● 半形式手法又は形式手法に関する実践知識

備考:共通キャリアスキルフレームワーク(CCSF)を参考にしている。

SSQデザイナ像

役割	高度な安全性又は情報セキュリティが求められるソフトウェアを開発する。
	製品又はシステムに対する安全性又は情報セキュリティに関する要求仕様を受けて、求められる安全性又は情報セキュリティを実現するソフトウェアを開発する。
タスク	● ソフトウェア要求分析 ● ソフトウェアアーキテクチャ設計
スキル	● 安全性又は情報セキュリティに関するソフトウェア要求仕様を定義できる。 ● 要求される安全性又は情報セキュリティを実現するソフトウェアアーキテクチャを設計できる。 ● ソフトウェア要求仕様及びアーキテクチャに関するハザード分析ができる。 ● 半形式手法又は形式手法を用いて要求仕様書及びアーキテクチャ設計書を作成できる。
知識	● 機能安全に関する実践知識 ● 情報セキュリティに関する実践知識 ● 半形式手法又は形式手法に関する実践知識

備考:共通キャリアスキルフレームワーク(CCSF)を参考にしている。

SSQエキスパート資格認定

SSQエキスパート研修

- ◆ 必修科目の終了
- ◆ 規定数以上の選択科目の終了

SSQコーチサービス

- ◆ 被コーチ時間が規定時間以上

3.要求の仕様化を支援するプロセスと手法

要約

- 書かれた仕様から意図するものを容易に再現できるためには、仕様化プロセスに工夫が必要である。その工夫を委員で分担して試みた。何を狙いとし、どのプロセスと手法を選択したかは、「仕様記述実験一覧」を参照。その中から次の4件を説明する:
  - システム要求設計技法による安全要求の記述
  - 物物関係分析法による安全制約の記述
  - 操作シナリオベース開発手法
  - SPINを活用する仕様検証

仕様記述実験一覧

狙い	題材	プロセス	手法	節番号
顧客合意		操作シナリオベース	VDM++	
意図の検証	車速制限		SPIN	
安全要求	電気ポット	システム要求設計技法	SysML	3.1
制約事項	イベント登録管理	物物関係分析法	VDM++	
安全制約	電気ポット	物物関係分析法	VDM++	3.2
仕様検証	モータのインバータ		SPIN	3.4
安全要求	電動支援自転車	操作シナリオベース開発	VDM++	3.3

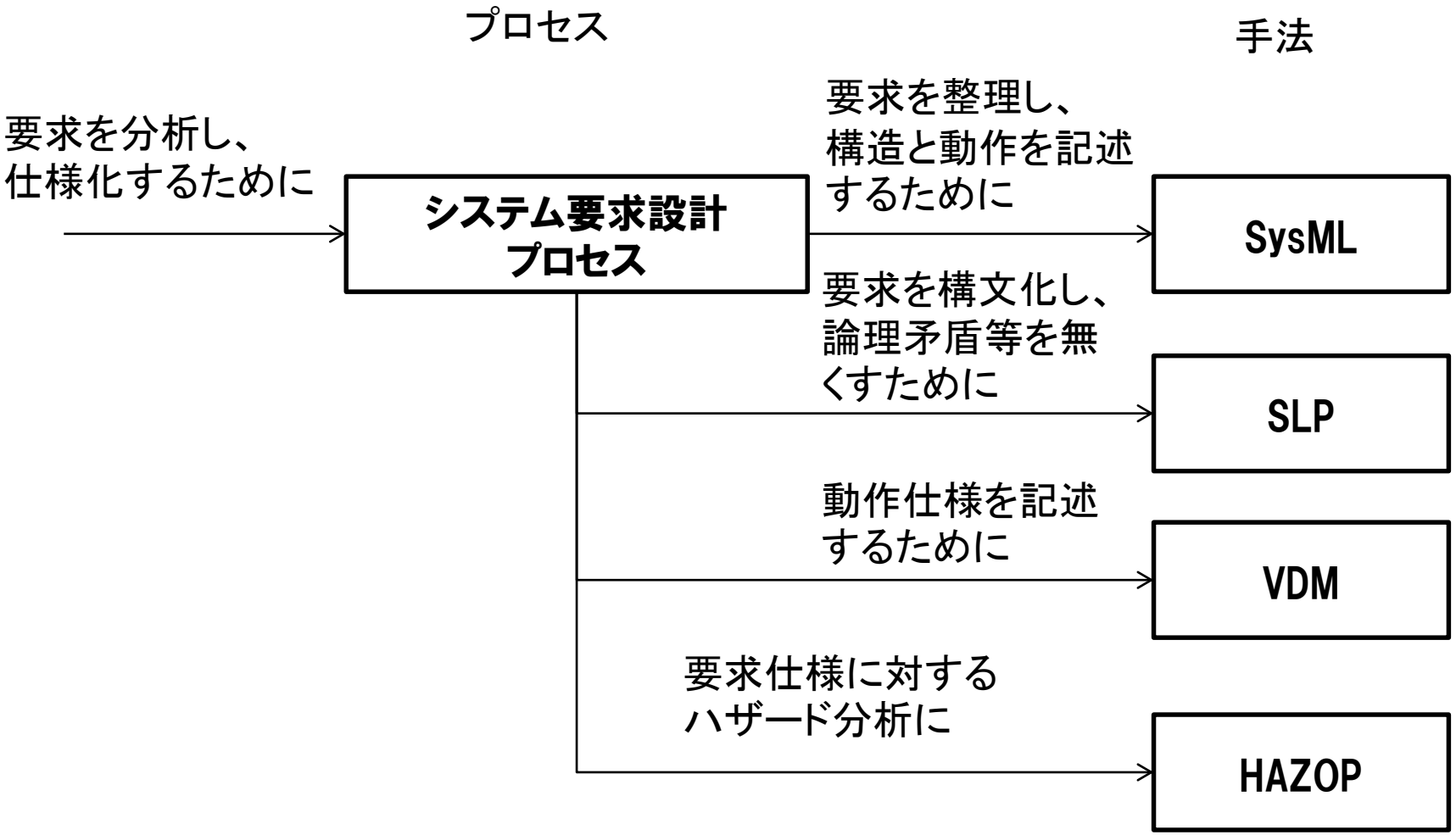
3.1 システム要求設計技法による安全要求の記述

システム要求設計技法の特徴

- システム又は製品に対する要求をもとにして、システムの構成要素に対する要求を定義するプロセスを示す。
- 標準的な開発プロセスモデルにおけるシステム要求分析とシステムアーキテクチャ設計に適用できる。
- 表記法としてSysMLを活用する。

RADT: システム要求設計技法

プロセスと手法



備考:SLPは自然言語を用いる要求記述言語の一種。VDMは形式手法の一種。

システム要求設計プロセスの概要

手順	作業項目	使用するSysML図
要求分析	要求を獲得する	要求図
	システムとその境界を決める	ブロック定義図
	システムの使われ方（機能）を定める	ユースケース図
	ユースケースの動作を表現する	シーケンス図 アクティビティ図 状態機械図
アーキテクチャ設計	システムを構成要素に分解する	ブロック定義図
	部品の相互作用を定義する	シーケンス図 アクティビティ図
	部品の相互接続を定義する	内部ブロック図
制約評価	システム特性に関する制約を獲得する	ブロック定義図 パラメトリック図
	性能等を評価し、アーキテクチャを修正	
要求割当て	構成要素の要求仕様を定める	ブロック定義図
	要求の追跡性を確立する	要求図

手順

1. 要求分析
    1. 要求を獲得する
    2. システムとその境界を決める
    3. システムの機能を定める
    4. ユースケースの動作を表現する
2. アーキテクチャ設計
    1. システムを構成要素に分解する
    2. 部品の相互作用を定義する
    3. 部品の相互接続を定義する
3. ハザード分析
4. 安全に関する要求定義
5. 要求分析とアーキテクチャ設計を繰り返す
6. ハザード分析の確認

(1) 要求分析

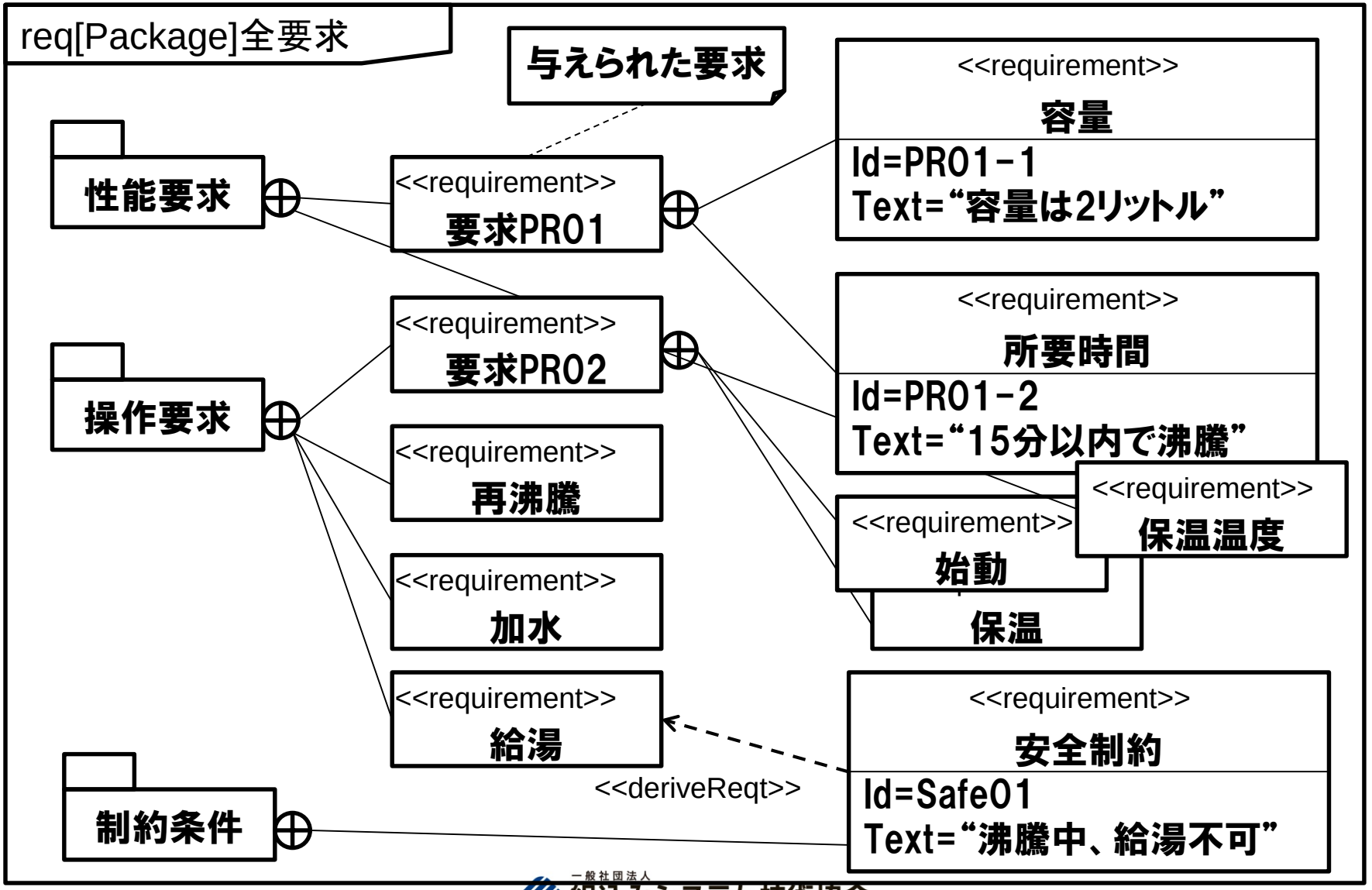
1. 要求を獲得する
   1. 電気ポットに関する要求を製品企画部門から入手する。
   2. 与えられた要求を上位方向に抽象化し、下位方向に詳細化し、過不足ないように要求図に整理する。
2. システムとその境界を決める
   1. 電気ポットの境界をシステムコンテキストとして表現する。
3. システムの機能を定める
   1. 操作に関する要求からユースケースを洗い出す。
4. ユースケースの動作を表現する
   1. 電気ポットの全体動作を状態機械図で記述する。
   2. 各状態における動作(振舞い)をアクティビティ図で記述する。

与えられた要求

電気ポットに関する要求

要求ID	要求事項
PR01	電気ポットの容量は2リットルとし、10度Cから沸騰するまでの時間は、15分以内とする。
PR02	電源コンセントをつなぐと、直ちに作動し、ヒータで加熱を始め、沸騰に達したら、90度Cに保温する。
PR03	再沸騰ボタンが押されたら、再沸騰を始める。
PR04	水が加えられ、温度が低下したら、再沸騰を始める。
PR05	保温中であれば、お湯を注ぐことができる。

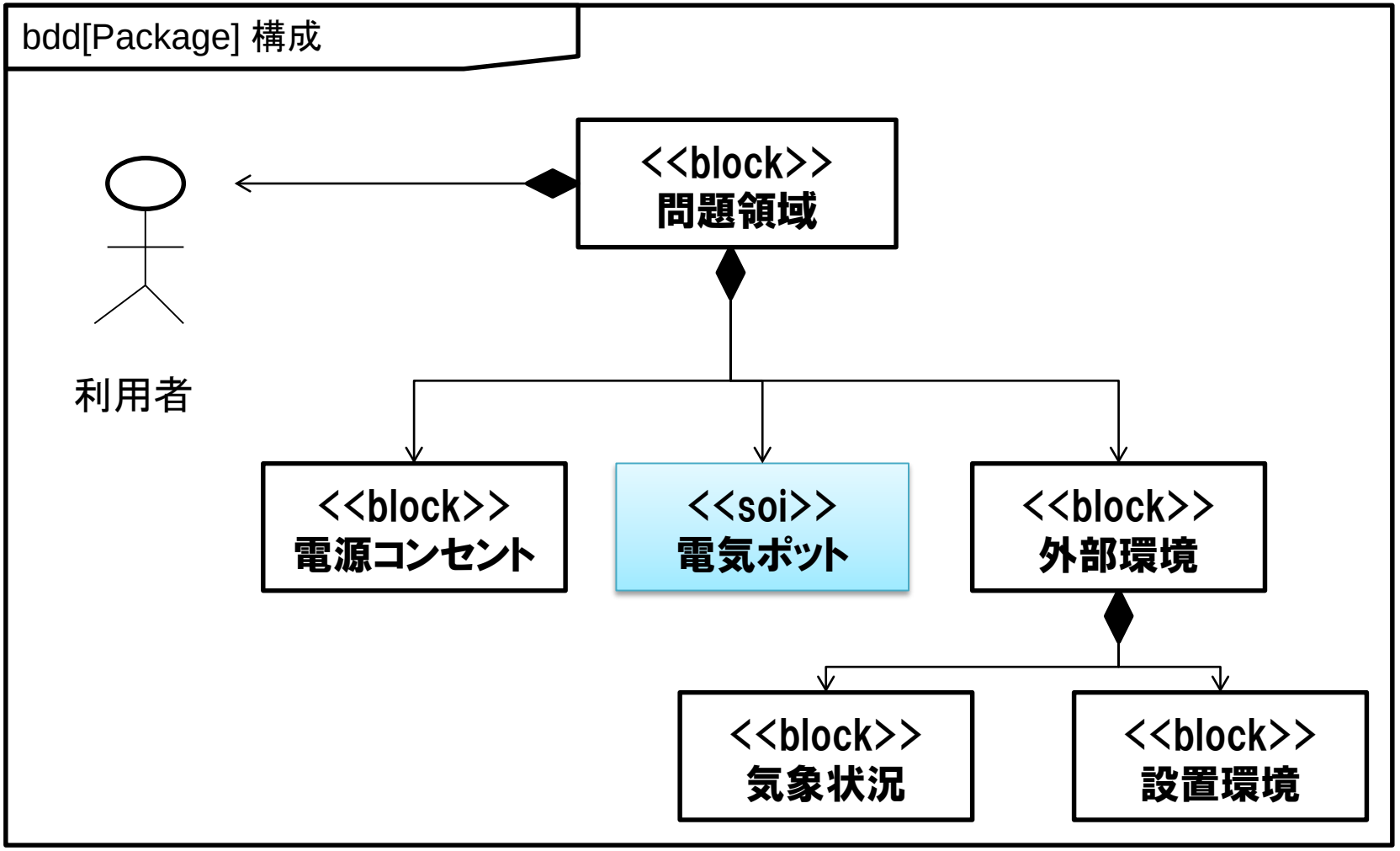
要求の整理



整理後の要素要求と派生要求

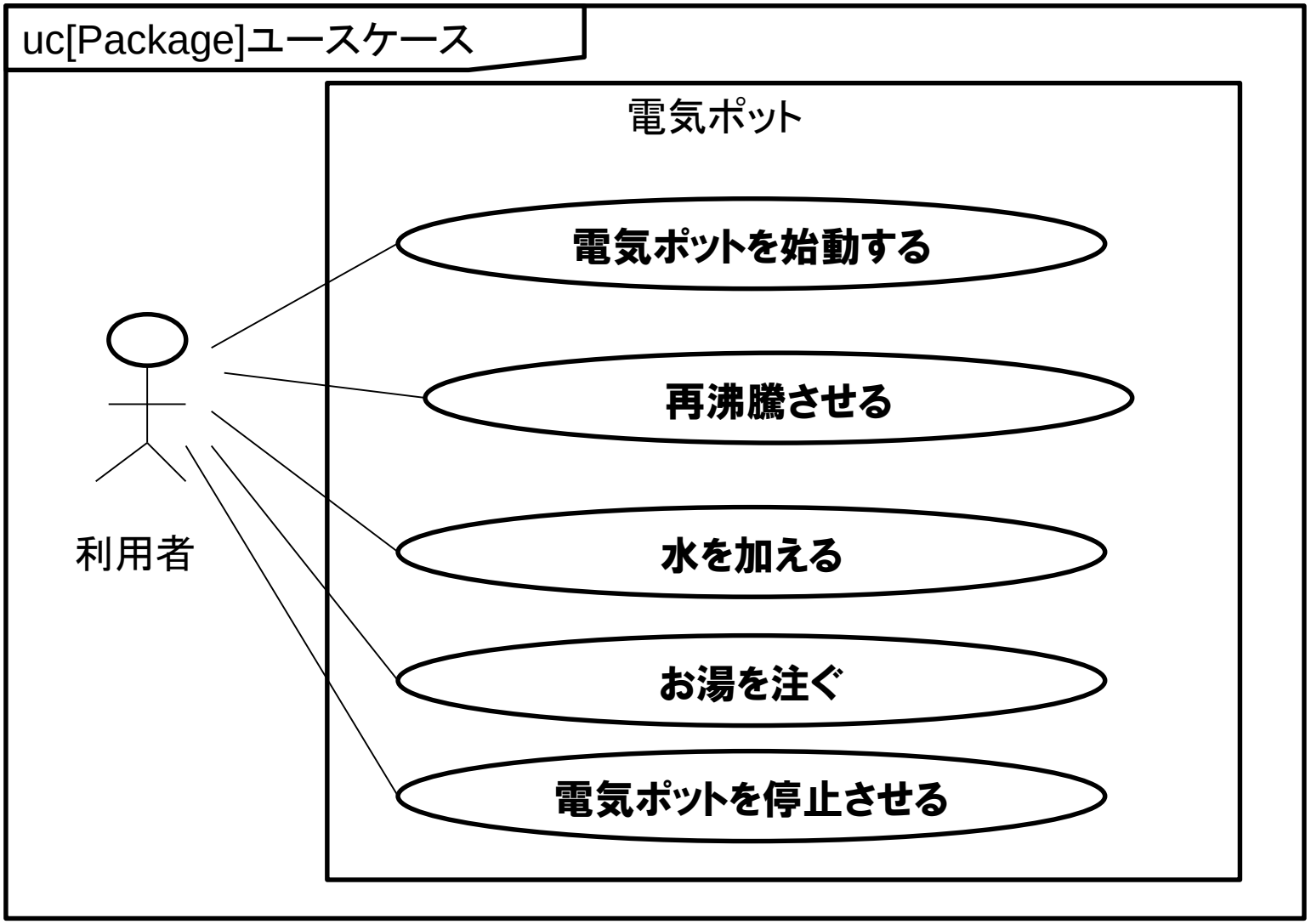
要求分野	要求ID	要求名	要求事項
性能要求	PR01-1	容量	電気ポットの容量は2リットルとする。
	PR01-2	所要時間	10度Cから沸騰するまでの時間は、15分以内とする。
	Perf01	保温温度	保温温度は90度C。(元の要求PR02から導出)
操作要求	PR02-1	始動	電源コンセントをつなぐと、直ちに作動し、ヒータで加熱を始める。
	PR02-2	保温	沸騰に達したら、保温する。
	PR03	再沸騰	再沸騰ボタンが押されたら、再沸騰を始める。
	PR04	加水	水が加えられ、温度が低下したら、再沸騰を始める。
	PR05	給湯	保温中に給湯ボタンを押せば、お湯を注ぐことができる。
	Ope01	停止	電源コンセントを抜くと、停止する。(省略されていた要求)
制約条件	Safe01	安全制約	沸騰中には給湯できない。(元の要求PR05から制約条件を導き出した)

システムコンテキスト



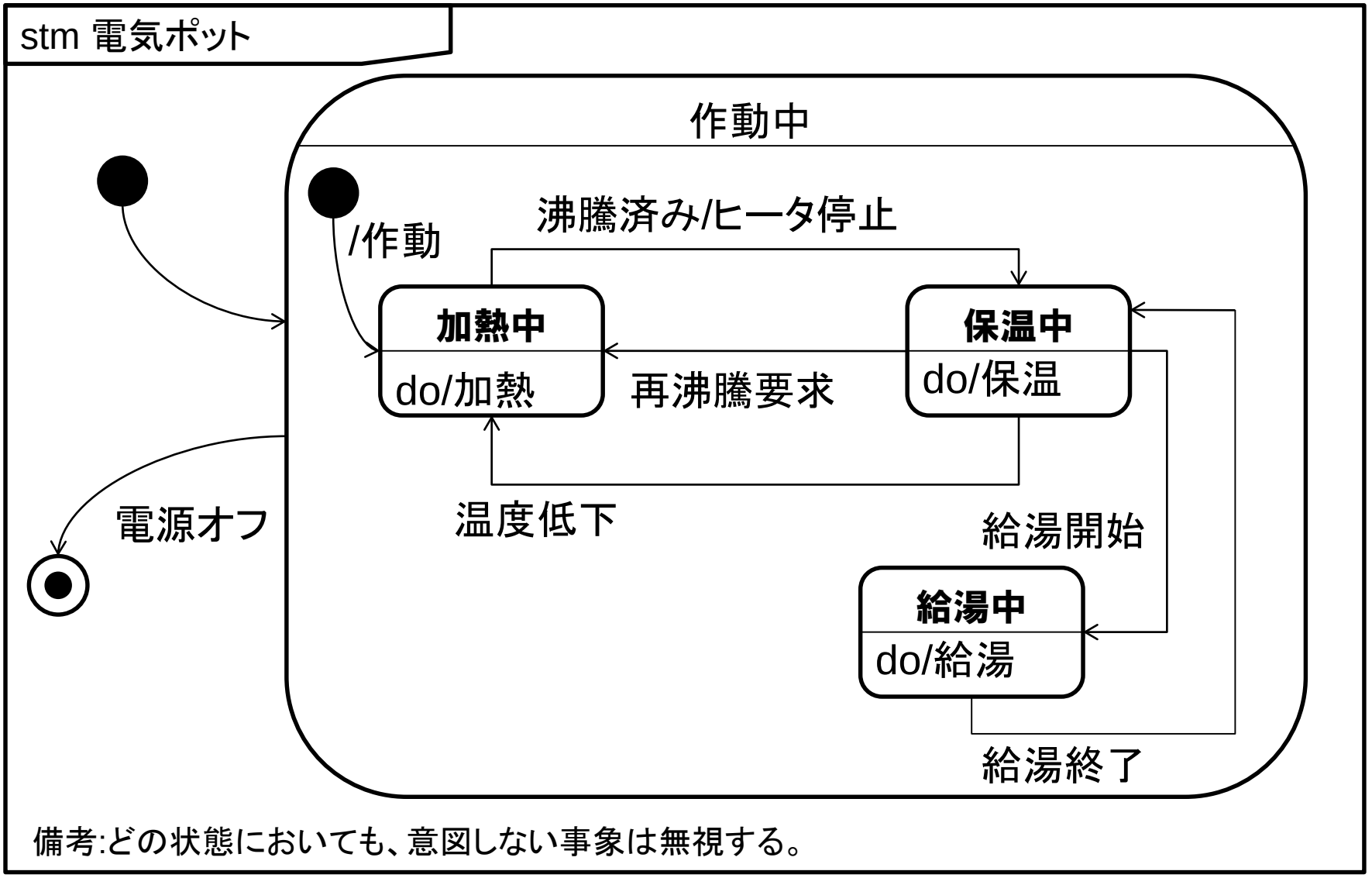
soi:System of Interest

ユースケース

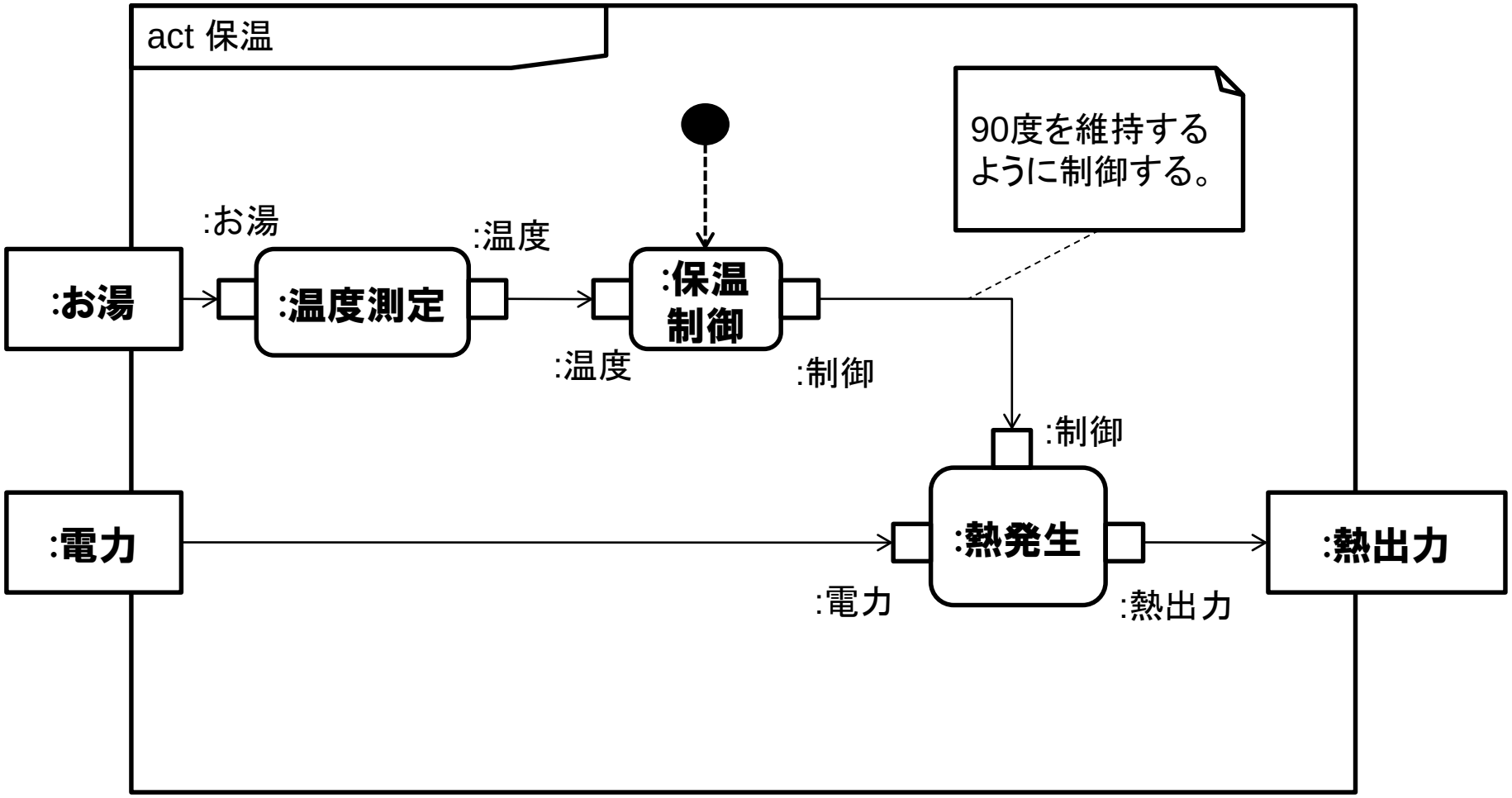


備考:操作に関する要求を基にしてユースケースを洗い出す。

状態機械図

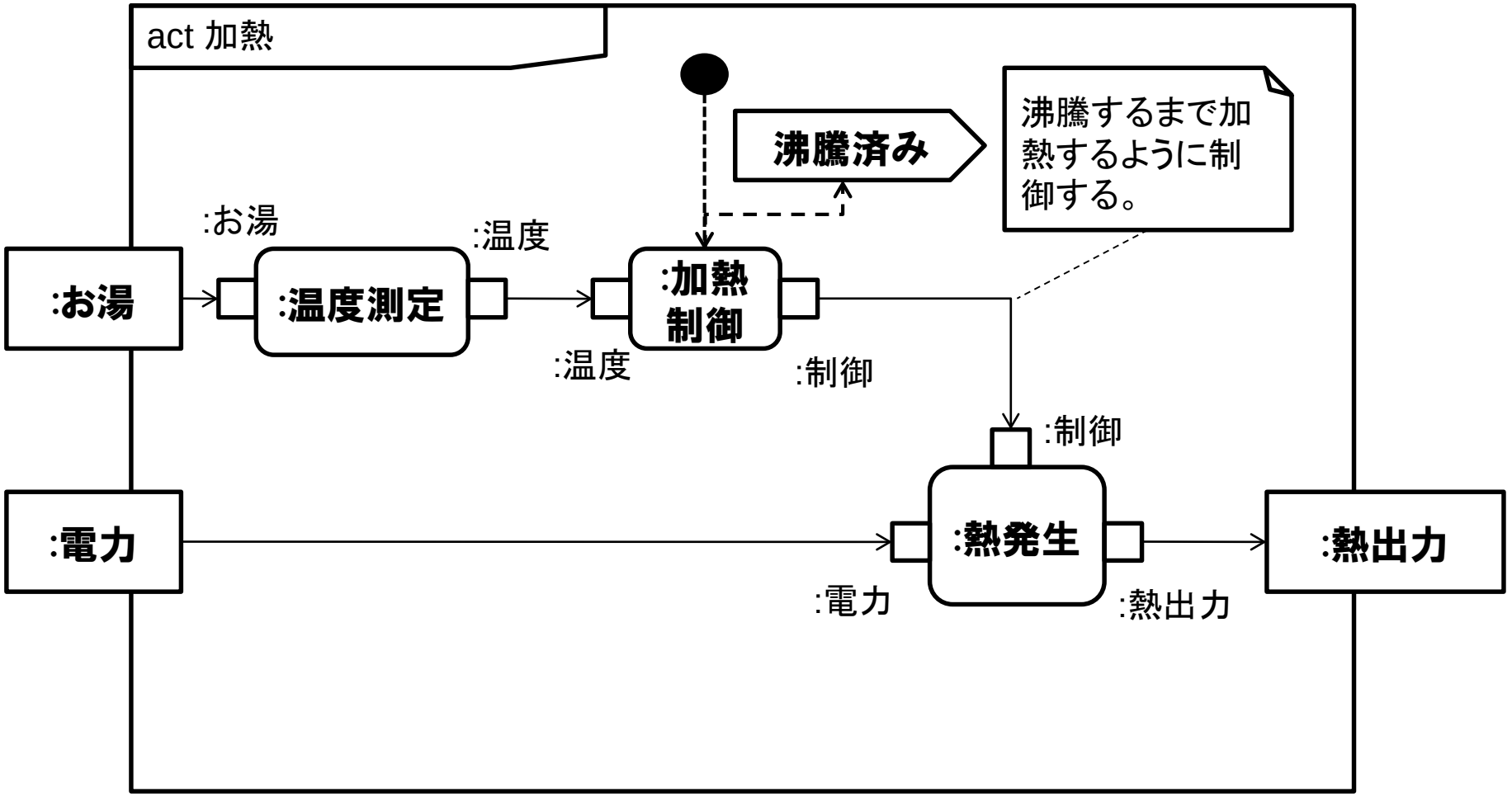


保温アクティビティ



備考:保温中の状態で実行される保温という処理をアクティビティ図で記述している。

加熱アクティビティ

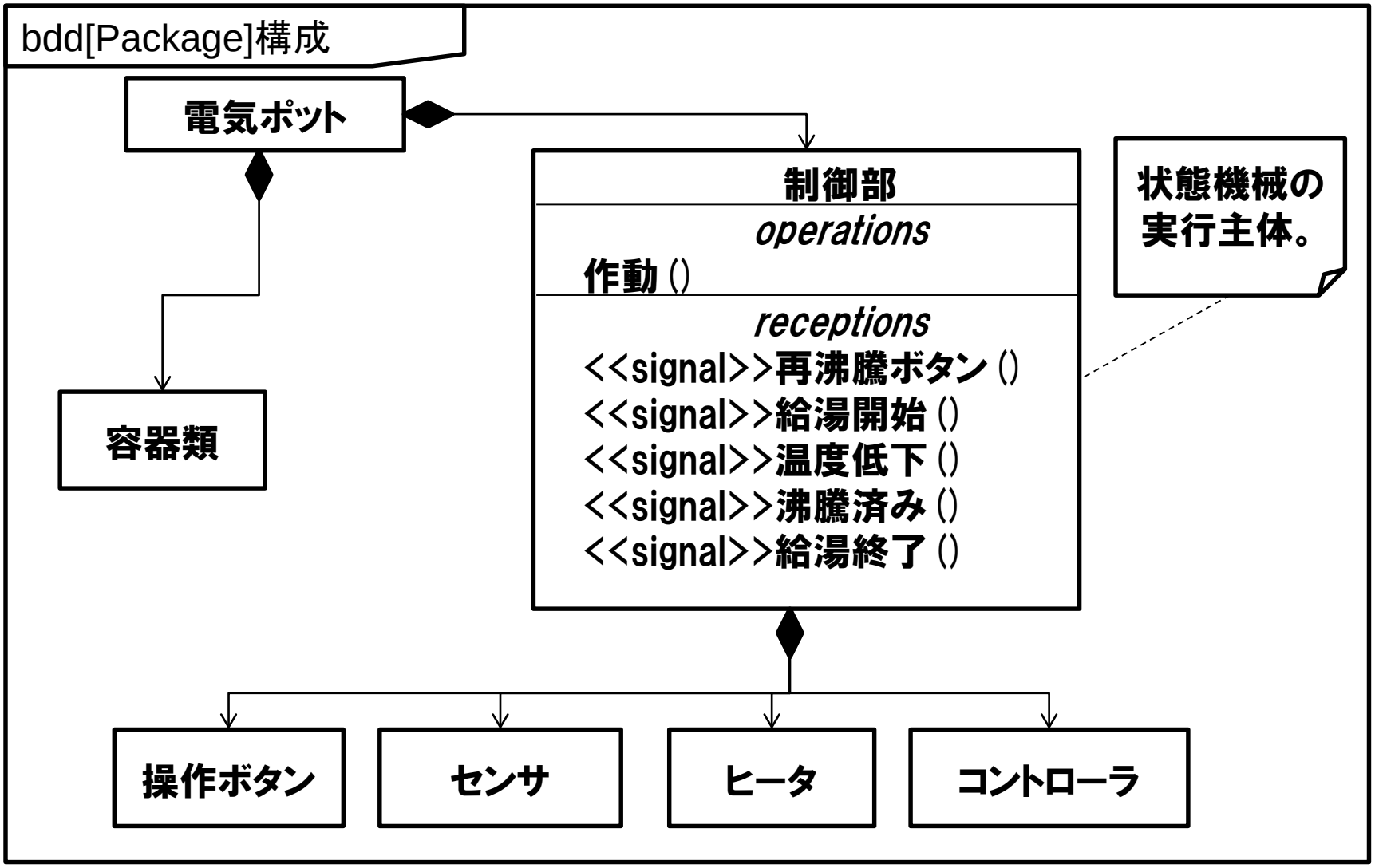


備考:加熱中の状態で実行される加熱という処理をアクティビティ図で記述している。

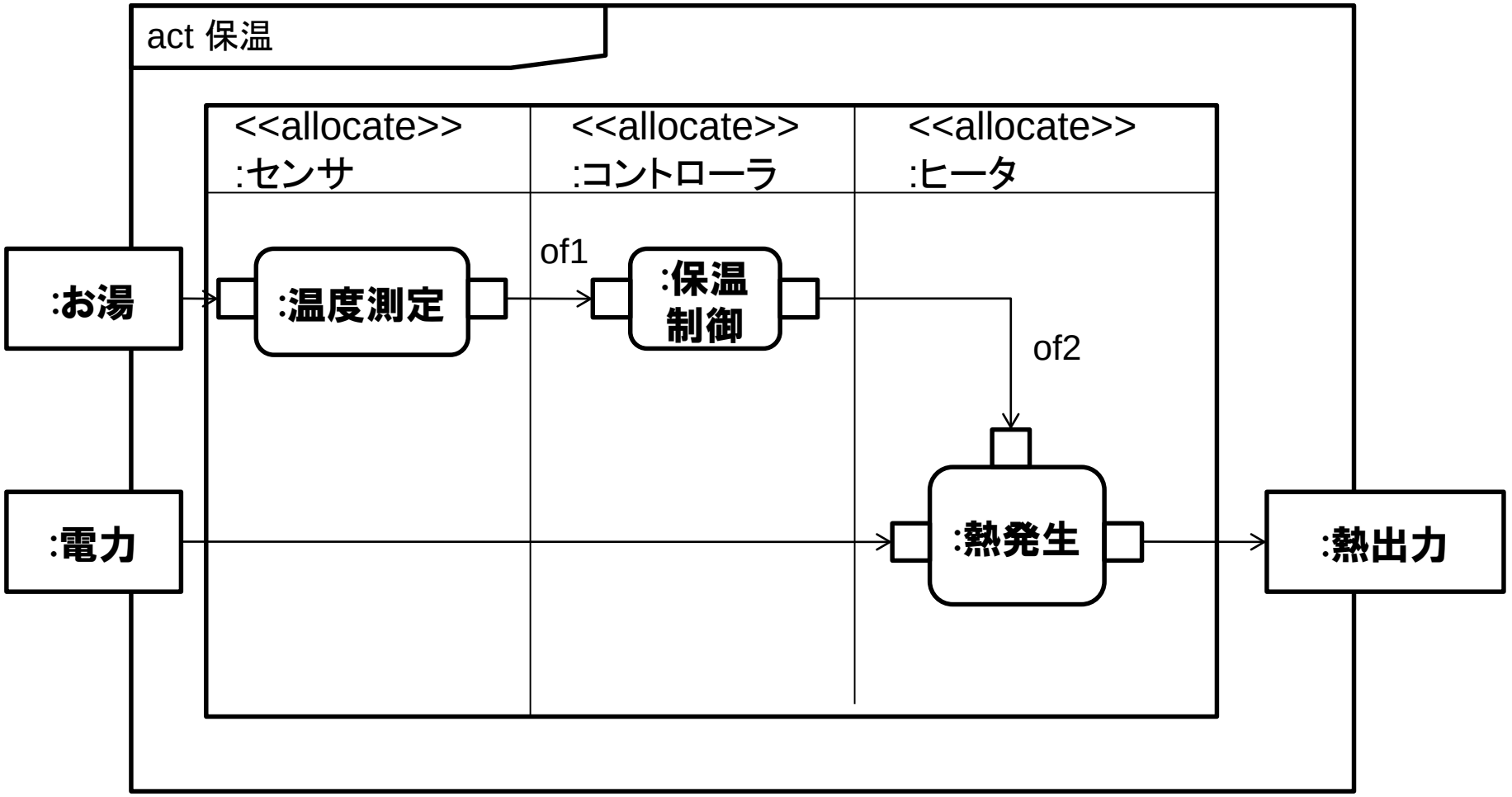
(2) アーキテクチャ設計

1. システムを構成要素に分解する
    1. 電気ポットを制御部と容器類に分け、制御部を構成する部品を記述する。
2. 部品の相互作用を定義する
    1. アクティビティ図における各処理(アクション)を、制御部を構成する部品に割当てる。
3. 部品の相互接続を定義する
    1. 制御部を構成する部品間をポートで接続し、部品間のデータを記述する。

システム構成

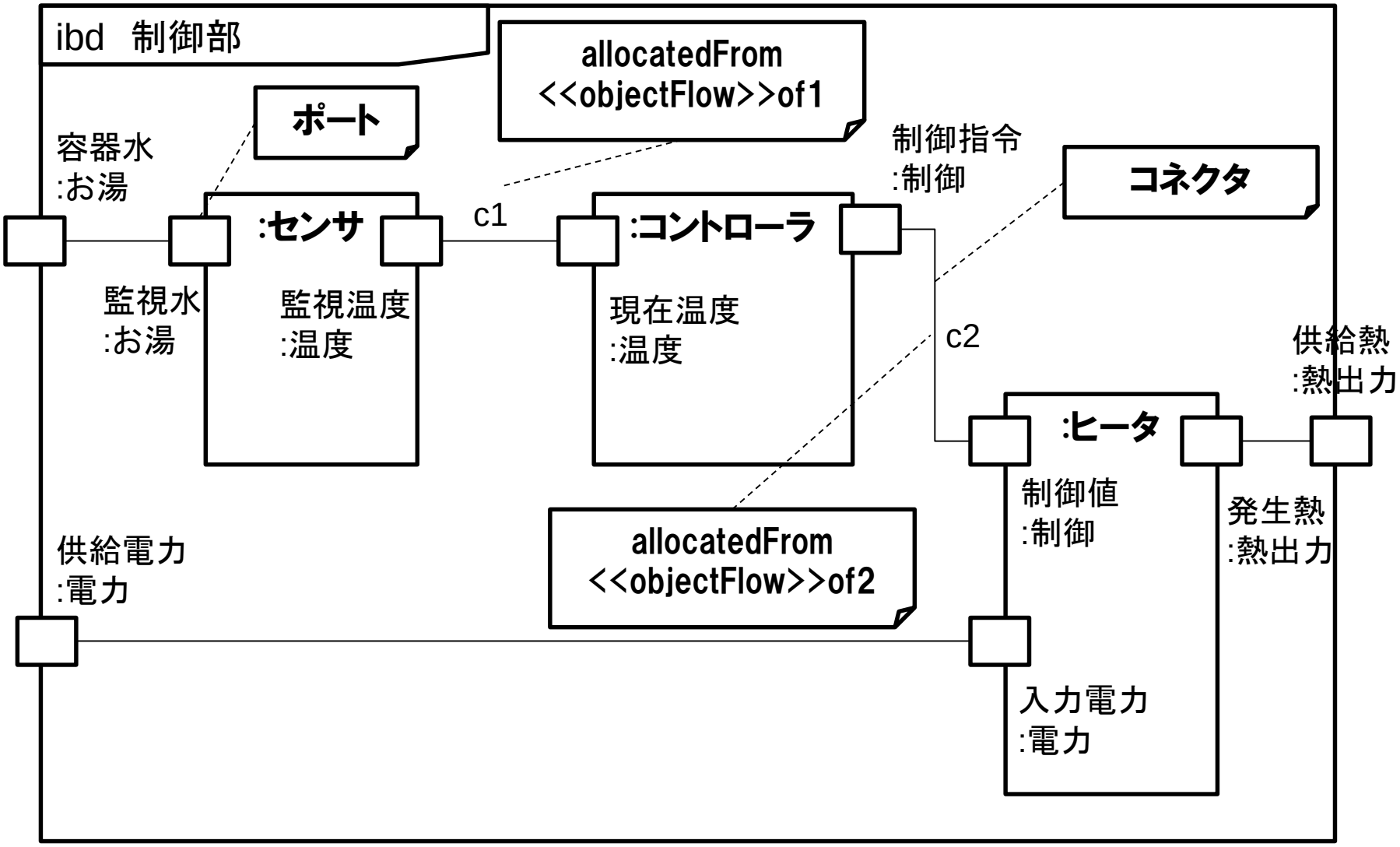


処理の割当て



備考:保温アクティビティを構成する処理をシステムを構成する部品に割当てる。

部品の相互接続



備考:部品のポートをコネクタで接続し、アクティビティ図におけるオブジェクトフローを割当てる。

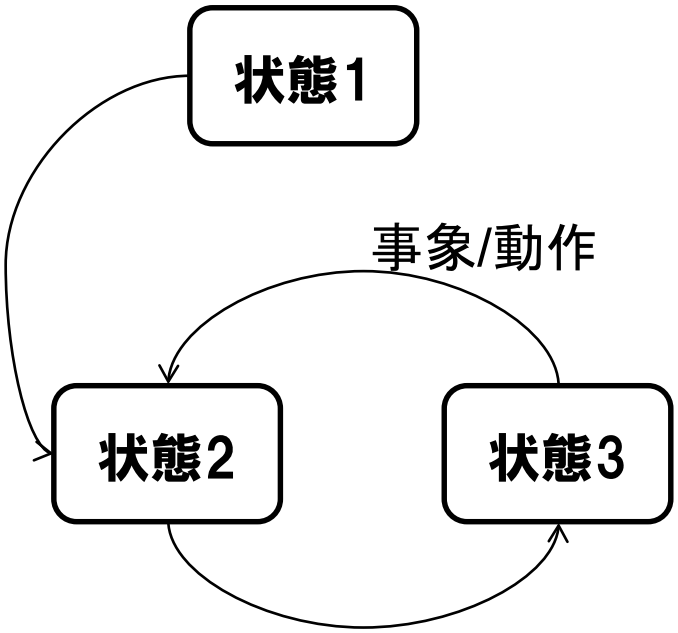
(3) ハザード分析と安全要求定義

1. ハザード分析
    1. 電気ポットの全体動作を示す状態機械に対して、HAZOP手法を用いてハザード分析を行う。
2. 安全に関する要求定義
    1. すべての識別されたハザードに対して、その安全要求を定義する。

HAZOP手法の実施手順

- 対象とするシステムと課題を定義する
- 事前準備
  - システムをいくつかの構成要素に分ける
  - 検討する設計意図からのずれ(逸脱)を用意する
  - ワークシートを用意する
- ある構成要素のある設計意図からのずれに対して、
  - それから引き起こされる事故を考える
  - ずれの原因を考える(あるいは、その逆)
  - 事故を防ぐ又は影響を緩和する安全機能を特定する
  - 不十分であれば、さらなるリスク低減策等を推奨する
- すべての構成要素のすべての設計意図からのずれが終わるまで繰り返す

状態遷移図におけるガイドワード



ガイドワード		遷移に対する解釈
No	否定	事象が未発生
More	量的変化	N/A
Less		N/A
As well as	質的変化	事象を誤検出
Part of		不完全な遷移 (動作が不完全)
Reverse	置換	N/A(Noに同じ)
Other than		別事象を誤認 別事象と同時発生

HAZOPによるハザード分析(1/2)

状態遷移: 加熱中 → 保温中
事象: 沸騰済み

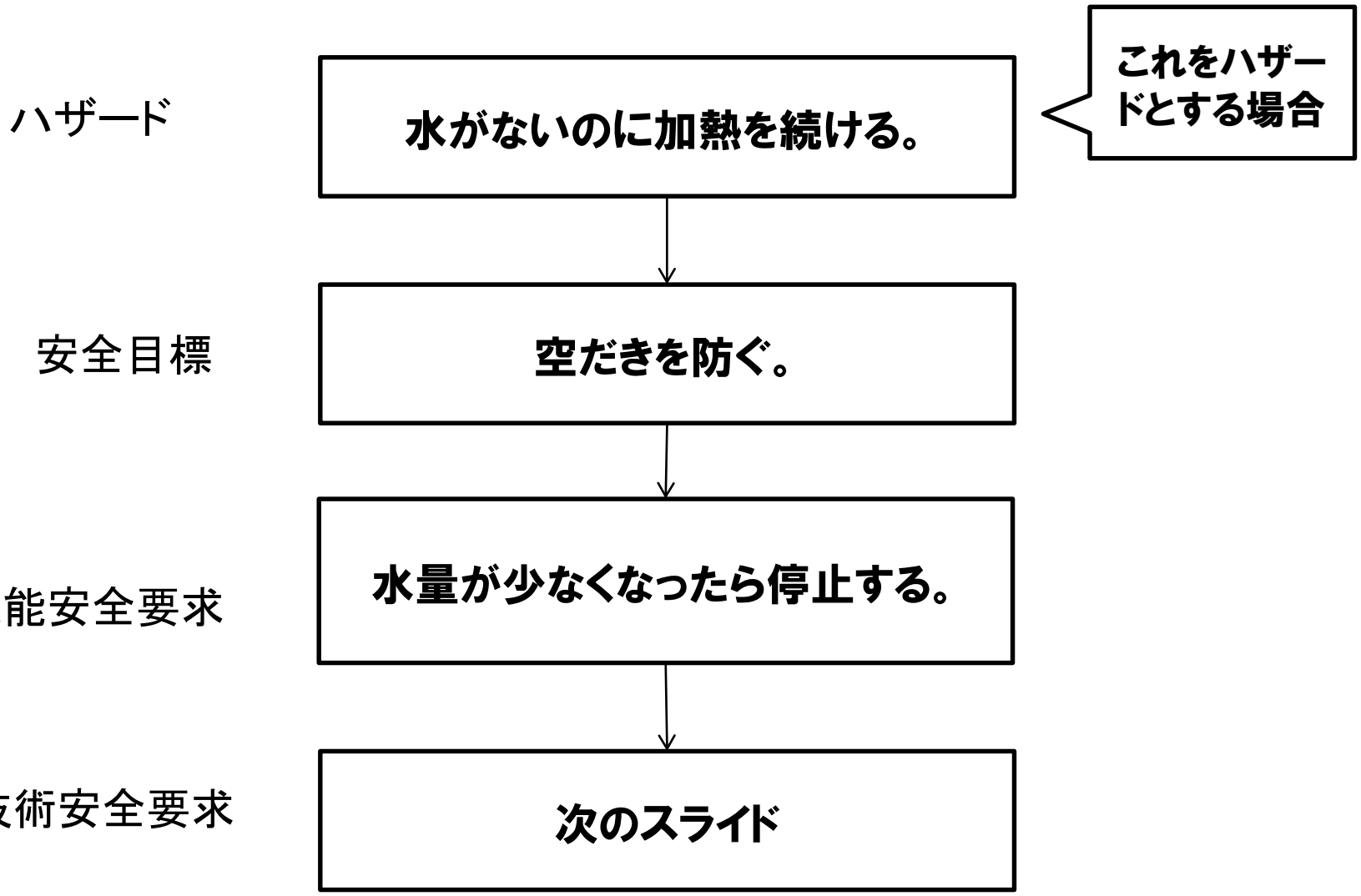
ガイドワード	原因と状況	結果と対策
No	沸騰に達したが、沸騰済み事象を発生しない。	加熱中状態で加熱が続く。吹き出すかもしれないし、水量が減って、空だきになるかもしれない。
As well as	沸騰していないけど、沸騰済みを誤検出。	保温中へ遷移し、沸騰に至らないまま、保温処理を行う。
Part of	ヒータを止めることができずに保温中へ遷移する。	保温中状態で加熱が続くが、保温処理がヒータを止めるはず。
Other than	AS well asに同じ	

HAZOPによるハザード分析(2/2)

状態遷移: 保温中 → 加熱中
事象: 再沸騰要求

ガイドワード	原因と状況	結果と対策
No	再沸騰を要求したが、再沸騰要求事象が発生しない。	保温処理を続ける。もう一度、再沸騰を要求すればよい。
As well as	再沸騰要求事象を誤検出。	余計な再沸騰が行われるだけ。
Part of	N/A	
Other than	実際は、温度低下事象が発生したが、これを再沸騰要求事象と誤認する。	結果的には再沸騰が行われ、問題はない。

ハザードと安全要求



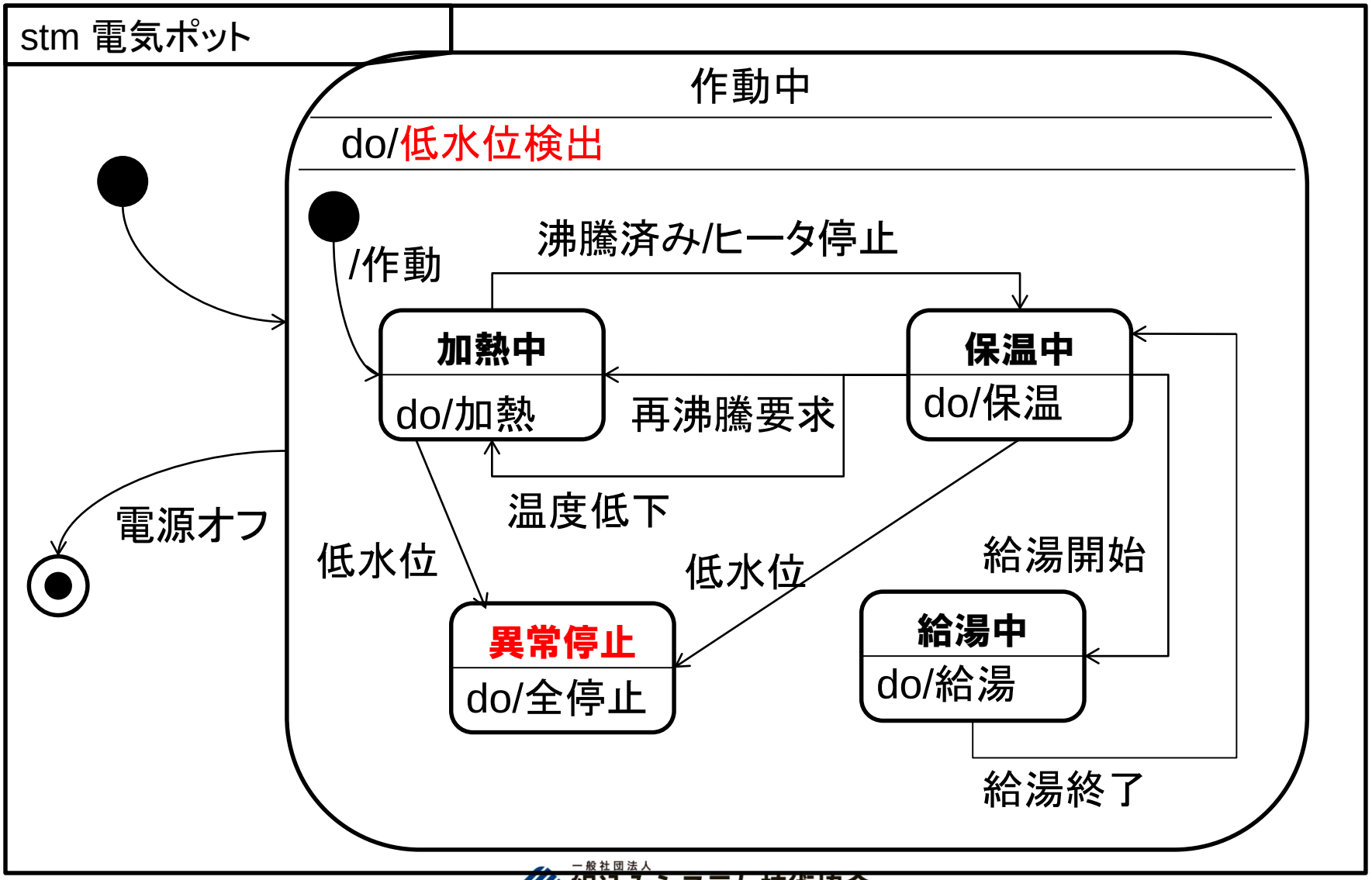
技術安全要求

要求番号	要求事項
TSR01	水位センサと低水位判定ソフトウェアによって低水位を検出する。
TSR02	水位センサは所定水位の水の存在を常時検知する。
TSR03	水位センサは、故障すると、水はないという状態を示すものとする。
TSR04	水量が全体の10%を切れば、低水位とする。
TSR05	低水位判定ソフトウェアは、水位センサの状態を調べ、水はないという状態であれば、低水位という事象を発生する。これを定期的に繰り返す。
TSR06	低水位事象を検出したら、ヒータを止め、電気ポットを停止させる。

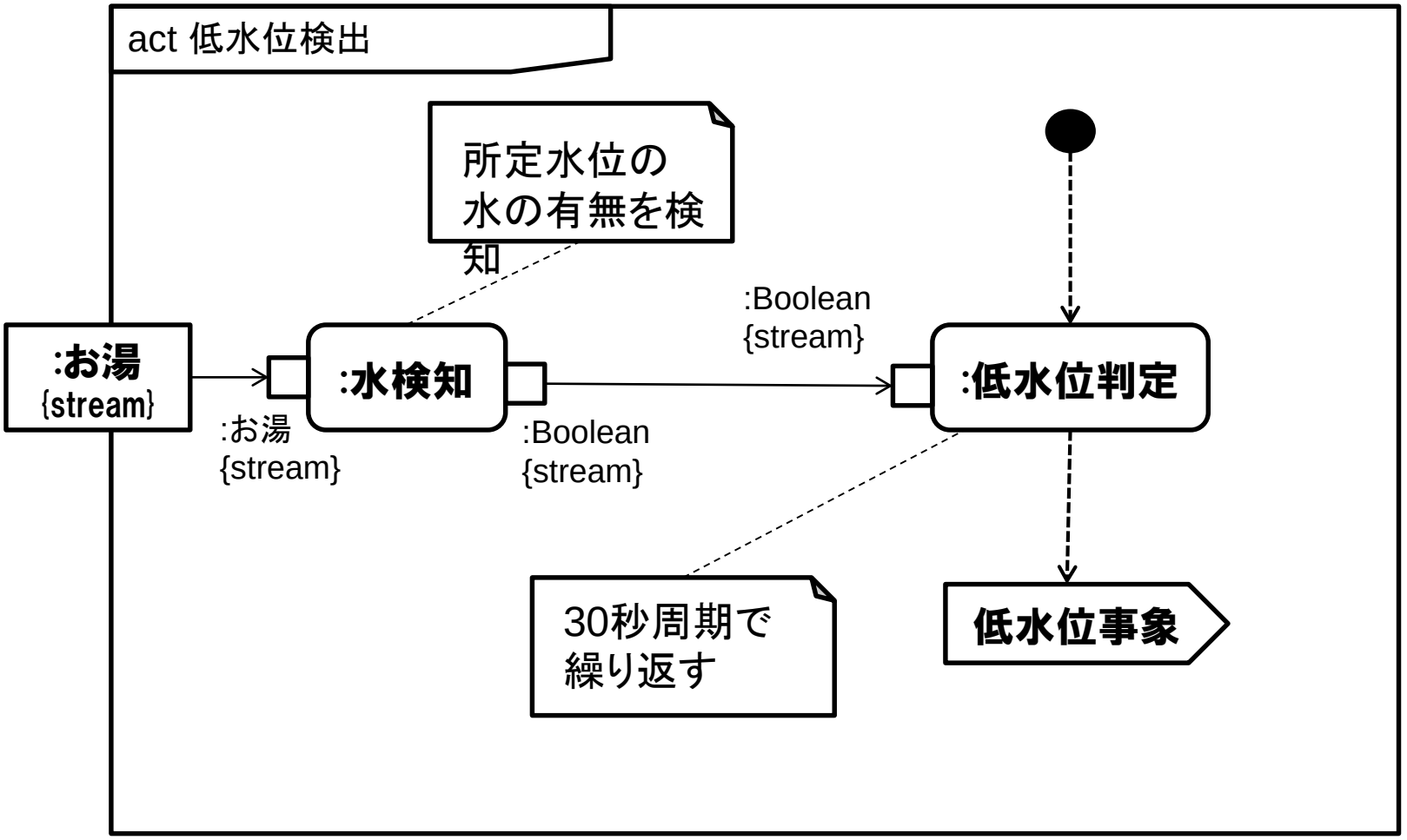
(4) 要求分析と設計の再実施

1. 要求分析
    1. 電気ポットの動作に安全要求を反映し、それに対応して状態機械を見直す。
    2. 追加される低水位検出のための動作(振舞い)に関して、その処理とデータの流れを記述する。
2. アーキテクチャ設計
    1. システム構成に水位センサを追加する。
    2. 水位センサとコントローラに追加された処理(アクション)を割当てる。
3. ハザード分析の確認
    1. ハザード分析を再実施し、対策後にハザードがないことを確認する。

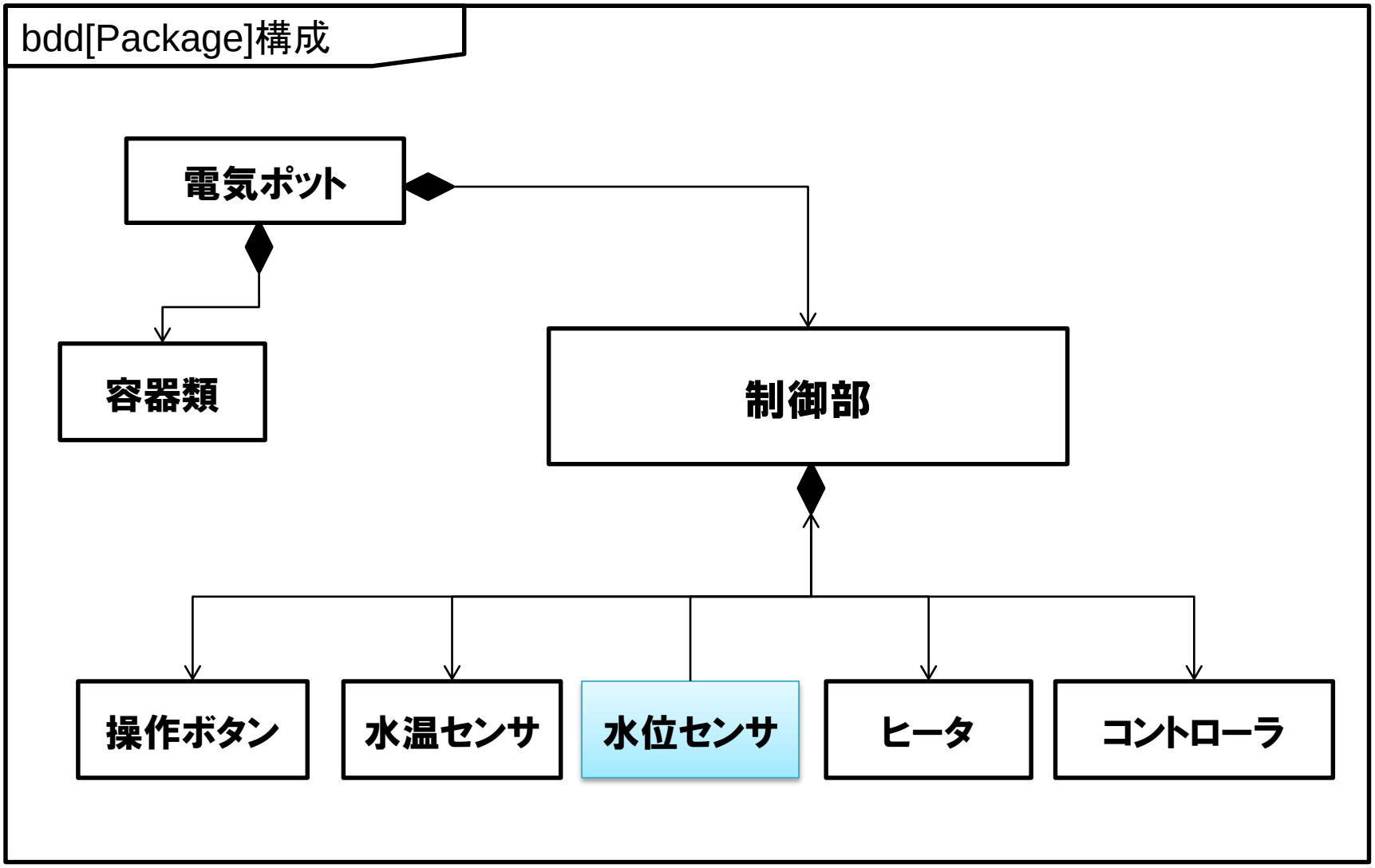
対策後の状態機械図



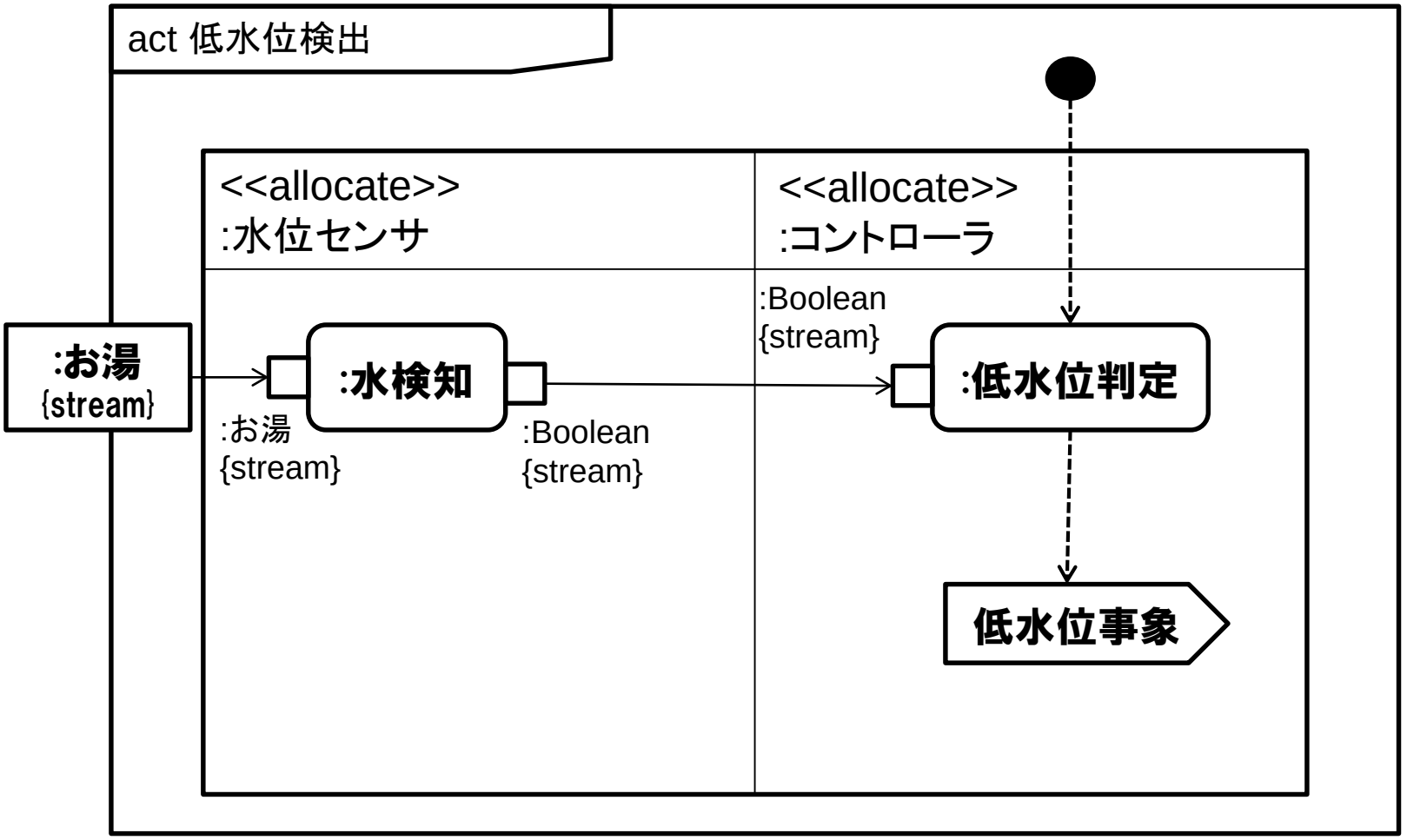
追加される低水位検出アクティビティ



対策後のシステム構成



追加される処理の割当て



再ハザード分析(1/3)

状態遷移: 加熱中 → 保温中
事象: 沸騰済み

ガイドワード	原因と状況	結果と対策
No	沸騰に達したが、沸騰済み事象を発生しない。	加熱中状態で加熱が続くが、水量が減っても、低水位検出が作動すれば空だきは防げる。
As well as	沸騰していないけど、沸騰済みを誤検出。	対策前と同じ。
Part of	ヒータを止めることができずに保温中へ遷移する。	対策前と同じ。
Other than	低水位事象を沸騰済み事象と誤認し、ヒータを止めて保温中へ遷移する。	保温処理が続くが、低水位検出は、低水位を定期的に検出するから、再度、低水位事象が発生するはず。

ハザードのないことを確認

再ハザード分析(2/3)

状態遷移: 加熱中 → 異常停止
事象: 低水位

ガイドワード	原因と状況	結果と対策
No	低水位になっても、低水位事象が発生しない。	加熱が続くが、低水位検出は、低水位を定期的に検出する。水位センサは、故障すると低水位を示すフェースセーフ設計となっている。
As well as	低水位事象を誤検出。	不必要に異常停止するだけ。
Part of	N/A	
Other than	沸騰済み事象を低水位事象と誤認。	不必要に異常停止するだけ。

再ハザード分析(3/3)

状態遷移: 保温中 → 加熱中
事象: 温度低下

ガイドワード	原因と状況	結果と対策
No	加水しても、温度低下事象が発生しない。	再沸騰しないが、再沸騰を要求すればよい。
As well as	温度低下事象を誤検出。	余計に再沸騰するだけ。
Part of	N/A	
Other than	低水位事象を温度低下と誤認し、加熱中へ遷移する。	加熱中状態で加熱が続くが、低水位検出は、低水位を定期的に検出する。

考察

■ 要求定義手法

- 電気ポットの与えられた要求から、要求分析とアーキテクチャ設計を行って、その結果として要求を記述した。
- 提供する機能をシステムの構成要素に割当てるには、この手法は有効である。

■ SysML

- システムの要求を体系的に整理できるので、漏れを防ぐには有効である。
- ハードやソフトにとらわれずに、システムを要素に分解でき、この構成要素によってシステムの機能を記述できることがわかった。
- これで書かれた設計資料はハザード分析に役立つことを確認した。

3.2 物物関係分析法による安全制約の記述

使用するプロセスと手法

仕様化プロセス:
物物関係分析法(注)による形式仕様記述法

ステップ1:要求を読み、分析する

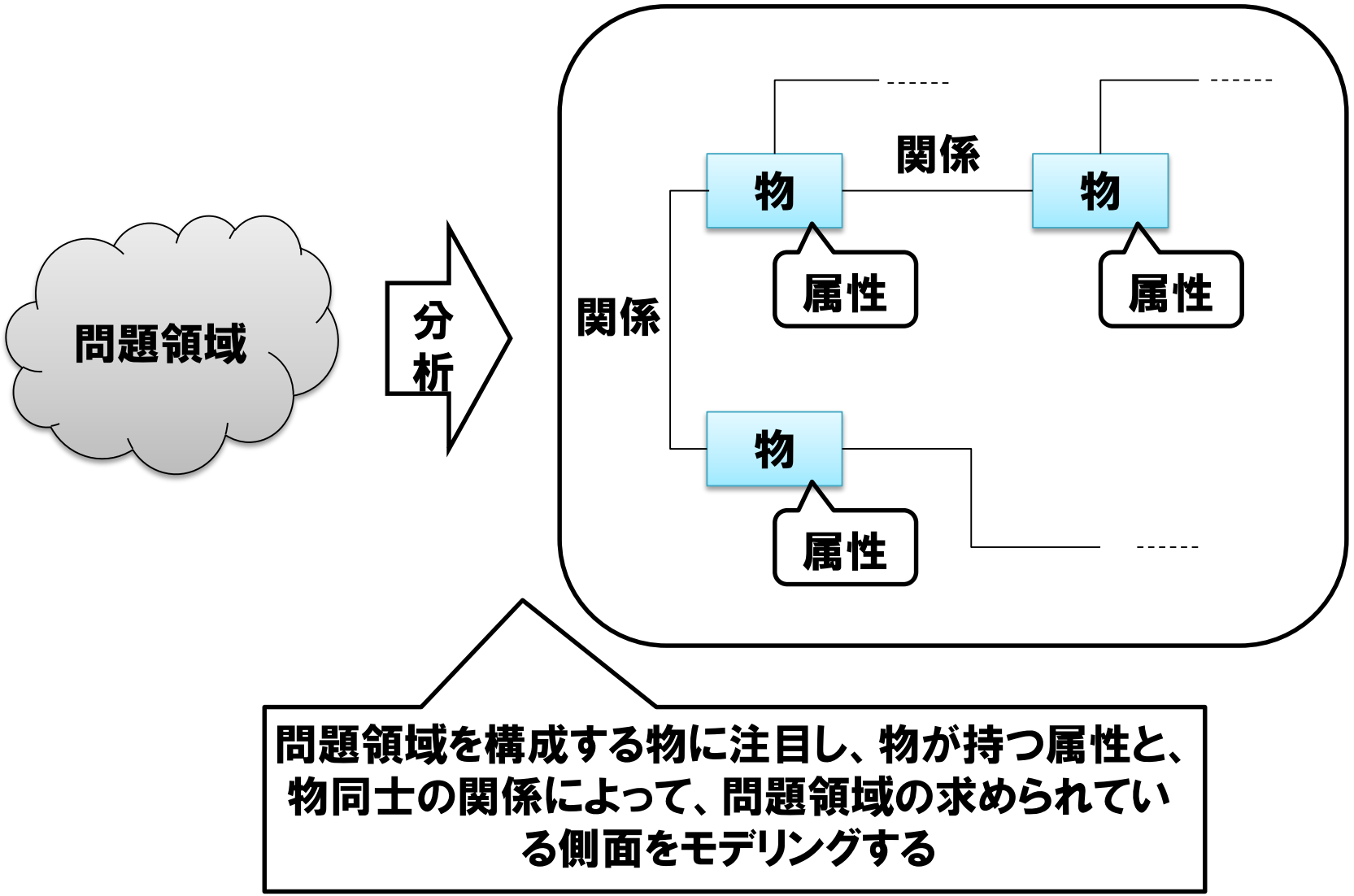
ステップ2:システムの構造をクラス図で表現する

ステップ3:クラス図をVDM++で記述する

ステップ4:仕様記述の内部一貫性をツールで検証する

注: クラス定義のために新しく考案した手法。

物物関係分析法



問題領域を構成する物に注目し、物が持つ属性と、物同士の関係によって、問題領域の求められている側面をモデリングする

クラス図作成の手順

1.対象とする機能を決める

2.物とその属性を洗い出す

名詞を拾い出し、
物と属性に分ける

3.物同士の関係を分析し、制約を明確にし、
クラス図を書いてみる

システムコンテキストを決める

4.クラス、属性、関連を見直す

5.クラスに操作を追加し、機能の定義を試みる

6.機能を記述できなければ、ステップ4に戻る

見直し観点:
クラス->属性
属性->関連
属性->クラス
関連->クラス
クラスの分解
クラスの併合

題材:電気ポット

与えられた要求

要求ID	要求事項
PR01	電気ポットの容量は2リットルとし、10度Cから沸騰するまでの時間は、15分以内とする。
PR02	電源コンセントをつなぐと、直ちに作動し、ヒータで加熱を始め、沸騰に達したら、90度Cに保温する。
PR03	再沸騰ボタンが押されたら、再沸騰を始める。
PR04	水が加えられ、温度が低下したら、再沸騰を始める。
PR05	保温中であれば、お湯を注ぐことができる。

物とその属性を洗い出す

名詞を拾い出し、物と属性に分ける

要求ID	要求事項
PR01	電気ポットの容量は2リットルとし、10度Cから沸騰するまでの時間は、15分以内とする。
PR02	電源コンセントをつなぐと、直ちに作動し、ヒータで加熱を始め、沸騰に達したら、90度Cに保温する。
PR03	再沸騰ボタンが押されたら、再沸騰を始める。
PR04	水が加えられ、温度が低下したら、再沸騰を始める。
PR05	保温中であれば、お湯を注ぐことができる。

電気ポット:
容量、沸騰時間、
制御状態

電源コンセント:
接続状態

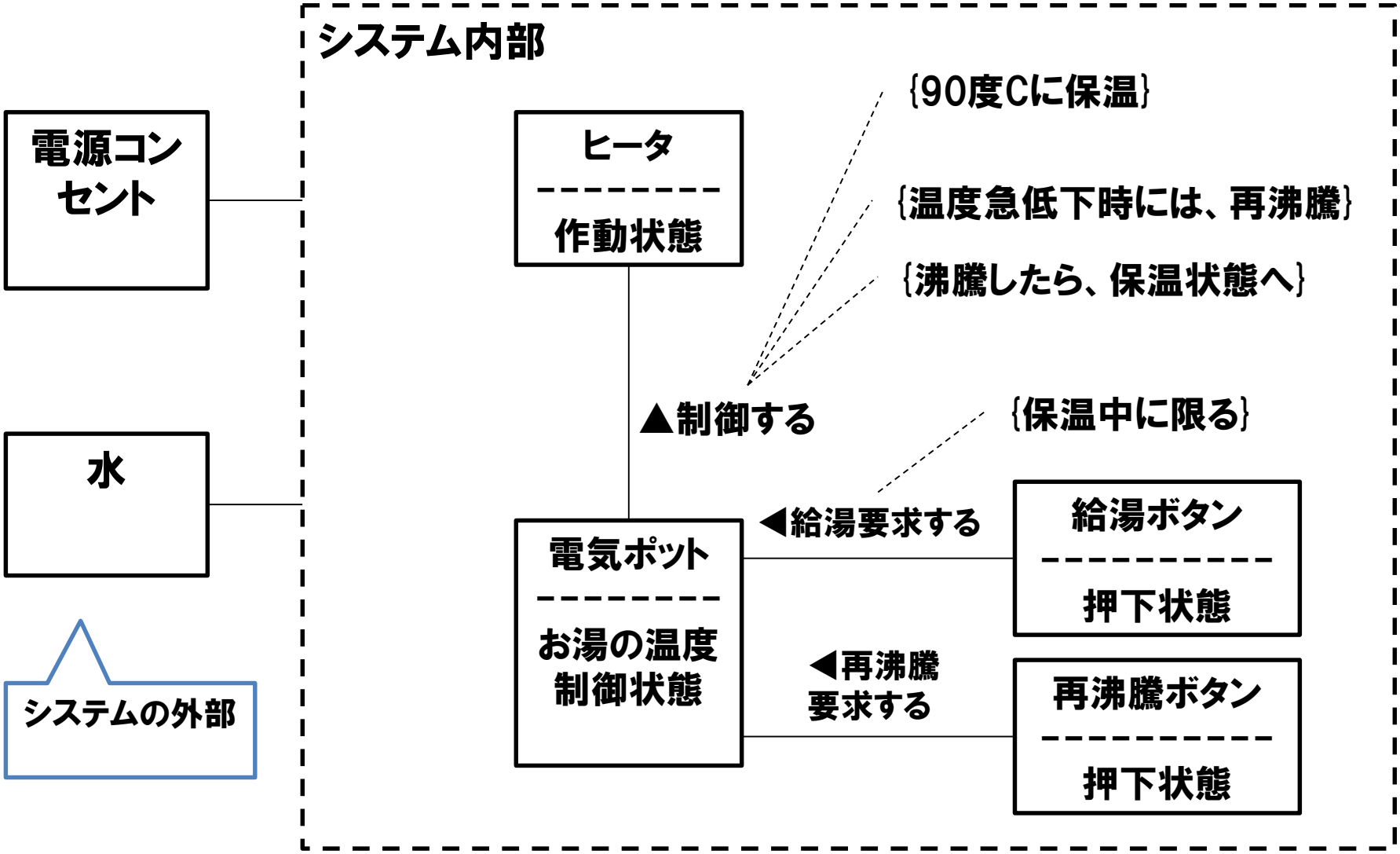
ヒータ:
作動状態

再沸騰ボタン:
押下状態

水:
温度

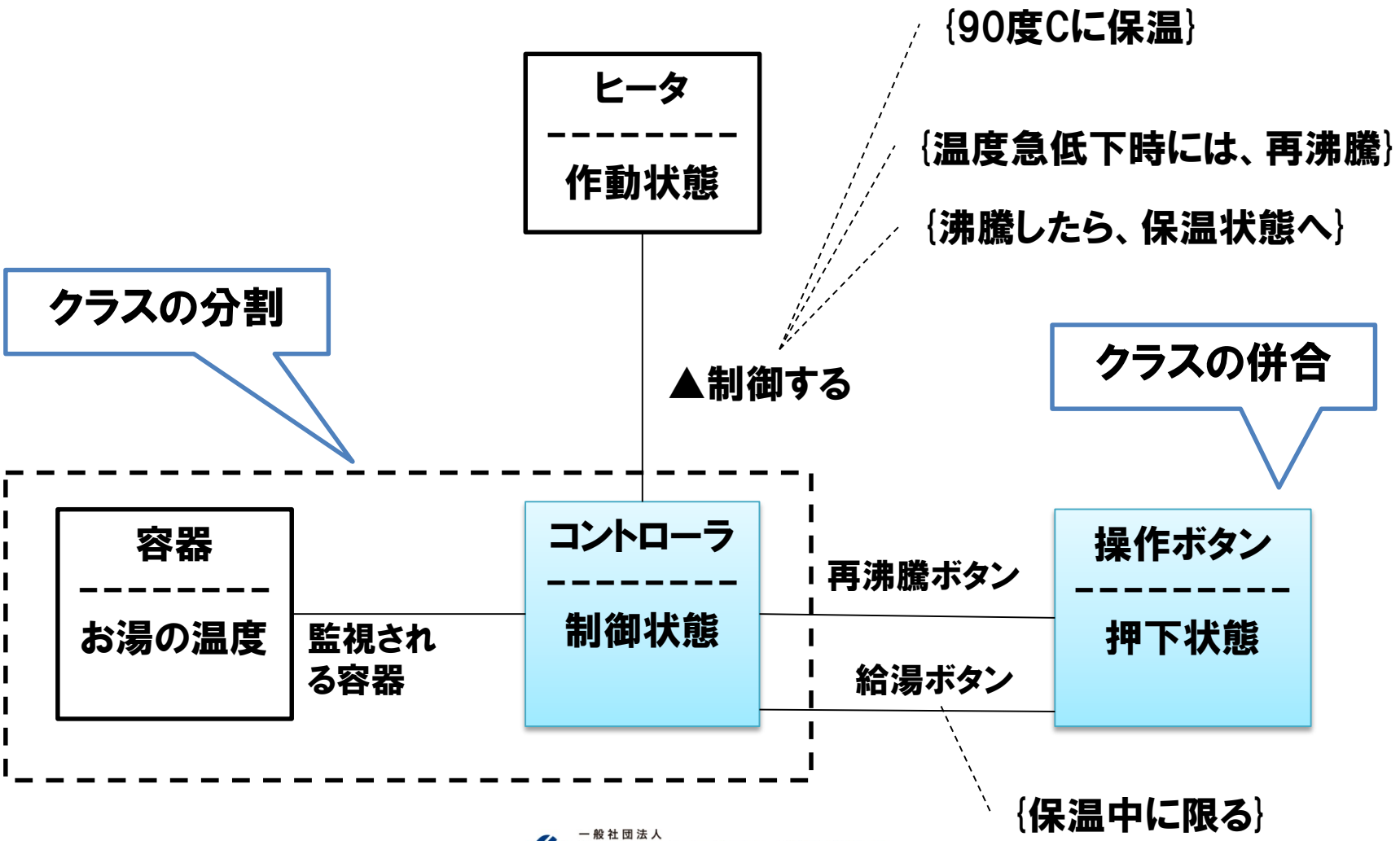
給湯ボタン:
押下状態

物同士の関係を分析する

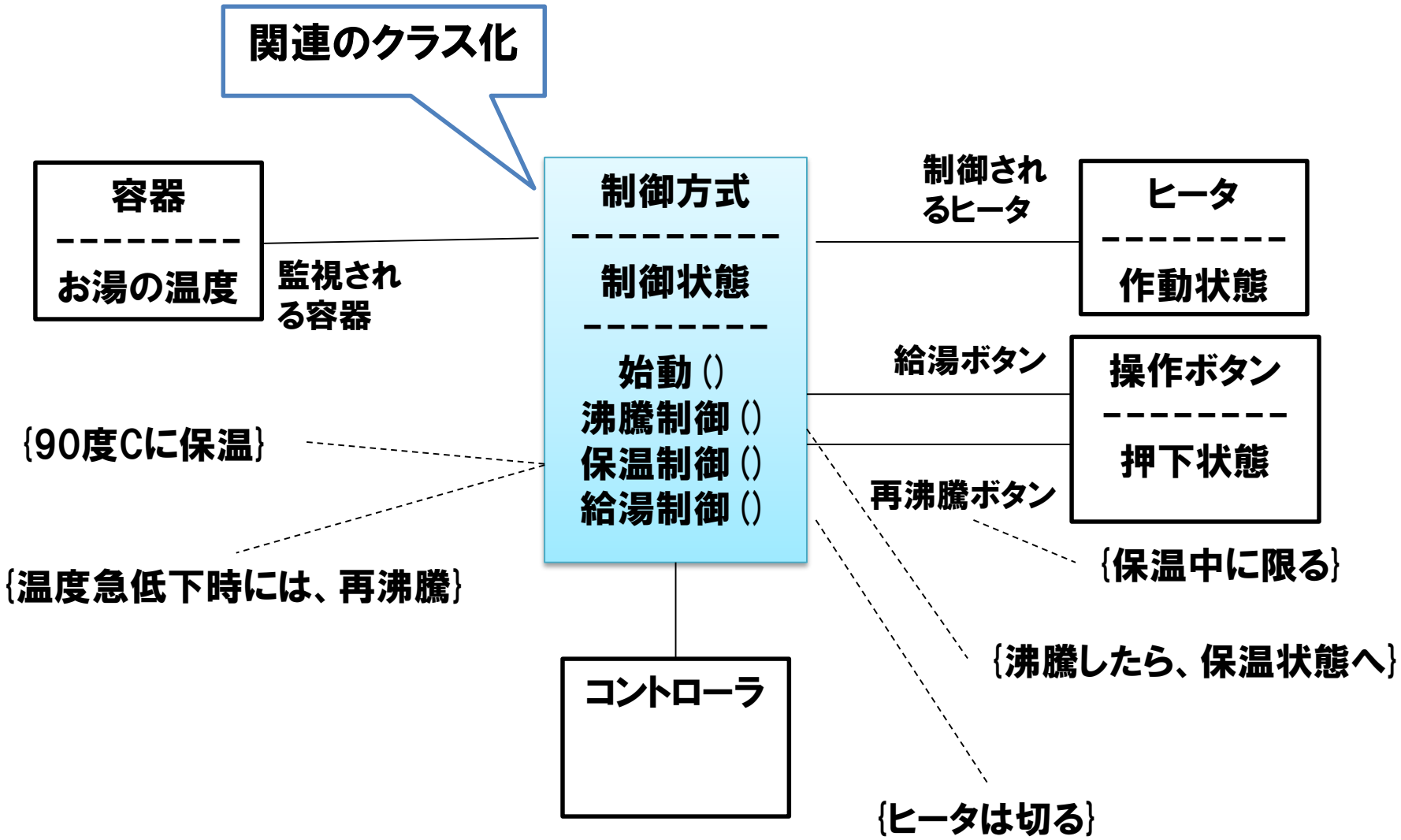


ただし、ここでは性能関係は除外。

クラス、属性、関連の見直し



最終的なクラス図



ヒータ、操作ボタン

```
class ヒータ
instance variables
  public 作動状態 : <on> | <off> := <off> ;
end ヒータ

class 操作ボタン
instance variables
  押下状態 : <on> | <off> := <off> ;
operations
  public 押下を知る: () ==> <on> | <off>
  押下を知る() == is not yet specified
      post RESULT = 押下状態 ;
end 操作ボタン
```

容器

```
class 容器
types
  public 水温 = nat;
instance variables
  お湯の温度 : 水温 := 0 ;
operations
  public 水温を知る: () ==> 水温
  水温を知る() == is not yet specified
       post RESULT = お湯の温度 ;
  public 沸騰しているか  : () ==> bool
  沸騰しているか() == is not yet specified ;
end 容器
```

制御方式

```
class 制御方式
types
  public 制御状態型 = <沸騰> | <保温> | <給湯> | <init> ;
instance variables
  public 制御状態 : 制御状態型 := <init> ;
  制御されるヒータ : ヒータ := new ヒータ () ;
  再沸騰ボタン : 操作ボタン := new 操作ボタン () ;
  給湯ボタン : 操作ボタン := new 操作ボタン () ;
  監視される容器 : 容器 := new 容器 () ;
operations
  public 始動: () ==> ()
  始動 () == is not yet specified
     pre 制御状態 = <init>
     post 制御状態 = <沸騰>
  ;
```

沸騰制御、給湯制御

```
public 沸騰制御: () ==> ()
 沸騰制御() == is not yet specified
     pre 制御状態 = <沸騰>
     post 監視される容器.沸騰しているか() and
             制御されるヒータ.作動状態 = <off> and
             制御状態 = <保温>
 ;
 public 給湯制御: () ==> ()
 給湯制御() == is not yet specified
     pre 制御状態 = <給湯>
     post 給湯ボタン.押下を知る() = <off> and
             制御されるヒータ.作動状態 = <off> and
             制御状態 = <保温>
 ;
```

{沸騰したら、保温状態へ}

{ヒータは切る}

保温制御

```
public 保温制御: () ==> ()
 保温制御() == is not yet specified
     pre 制御状態 = <保温>
     post def t = 監視される容器.水温を知る() in
             if 給湯ボタン.押下を知る() = <on> then 制御状態 = <給湯>
             elseif 再沸騰ボタン.押下を知る() = <on> then 制御状態 = <沸騰>
             elseif 保温限界か(t) then 制御状態 = <沸騰>
             else (保温範囲か(t) and 制御状態 = <保温>)
      ;
 保温限界か : 容器`水温 ==> bool
 保温限界か(t) == is not yet specified
 ;
 保温範囲か : 容器`水温 ==> bool
 保温範囲か(t) == is not yet specified
    post if t >= 85 and t<=95 then RESULT = true
            else RESULT = false
 ;
end 制御方式
```

{保温中に限る}

{温度急低下時には、再沸騰}

{90度Cに保温}

コントローラ

```
class コントローラ
-- for test
operations
  public 状態制御する : () ==> ()
  状態制御する() ==
    (dcl cont : 制御方式 := new 制御方式() ;
     while true do
       cases cont.制御状態 :
       <init> -> cont.始動() ,
       <沸騰> -> cont.沸騰制御() ,
       <保温> -> cont.保温制御() ,
       <給湯> -> cont.給湯制御()
       end
     ) ;
end コントローラ
```

これは仕様ではなく、一つの実装例であり、テスト用。

考察

■ 安全制約

- 物物関係分析法を用いて、システムを構成する要素とその属性を識別し、構成要素をクラスとするクラス図を作成した。
- 構成要素間の関係を分析し、クラス間の関連に制約を認識した。
- 破られることによって安全侵害を起こす制約が、安全制約である。

■ VDM++

- システムを状態機械と見なして、各状態における動作をクラスの操作に対応させ、その仕様を、事前条件と事後条件として記述した。
- 安全制約は、操作の事後条件として記述できる。

3.3 操作シナリオベース開発手法

はじめに

上流工程における妥当な仕様作成プロセスの構築

- 後工程を後戻り無く効率的に進めるには、上流工程を重視する必要がある。
- ソフトウェア開発の失敗は上流工程に主原因がある。
- 経験と勘のみに頼るのではなく、ツールも併用して合理的に妥当な仕様を作成したい。

後から分かってみると、きわめて自明なんだけど、最初はどうしてよいかわからなかったよ！
何か良い方法はないかね？

良い要求仕様とは？

良い仕様記述の条件

- エンドユーザの意図を汲み取り、実現する現実的な解となっていること。
- 立場、背景が異なる人間が読んでも、ブレのない解釈が可能なこと。
- 実装の縛りは最小限となっていること。

良い要求仕様とは？

抽象度の高い仕様記述

- 良いか悪いか比較的判断し易い。
- 玉虫色（解釈の幅が大きい）。
- 例外が未考慮。

具体的すぎる仕様記述

- 記述量が膨大になり、理解しづらい。
- 意図が伝わりにくくなる。
- 実装まで規定しがちで、それが適切でない場合、困難（変更に対する重い慣性）を伴う。

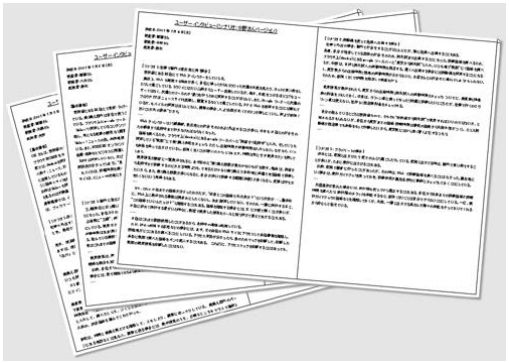
良い要求仕様とは？

どこに問題があるのか？

- 要求仕様を実装（精密に定義）して初めて不適切であることが分かる。
- 曖昧にしていたこと（後で考えることにしていたこと）が問題を引き起こす。

どうすれば良い要求仕様を作れるか？

操作シナリオベース開発手法



要求仕様

操作シナリオ

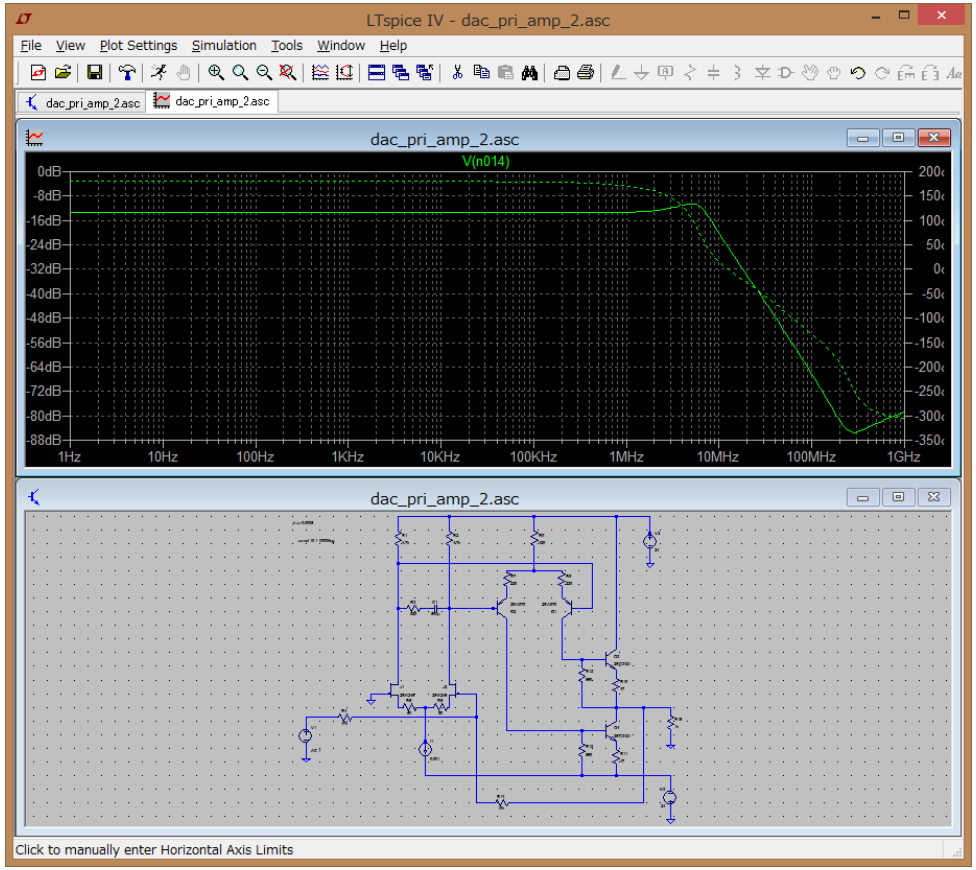
- 適用する場面（操作シナリオ）を考えて要求仕様の適切さを検証する。例外を考慮する。
- 要求仕様を操作の具体例で考える（具体的なものは考え易い）。

注）操作：「○○する」と言う動詞で表されるもの。

どうすれば良い要求仕様を作れるか？

操作シナリオベース開発手法

■ 要求仕様のシミュレーションを行って事前検証する。



LTSpiceを用いたシミュレーション例

電子回路設計では、SPICEシミュレータを用いて、事前に動作を検証してから試作を行うことにより、生産性の向上が図られる。

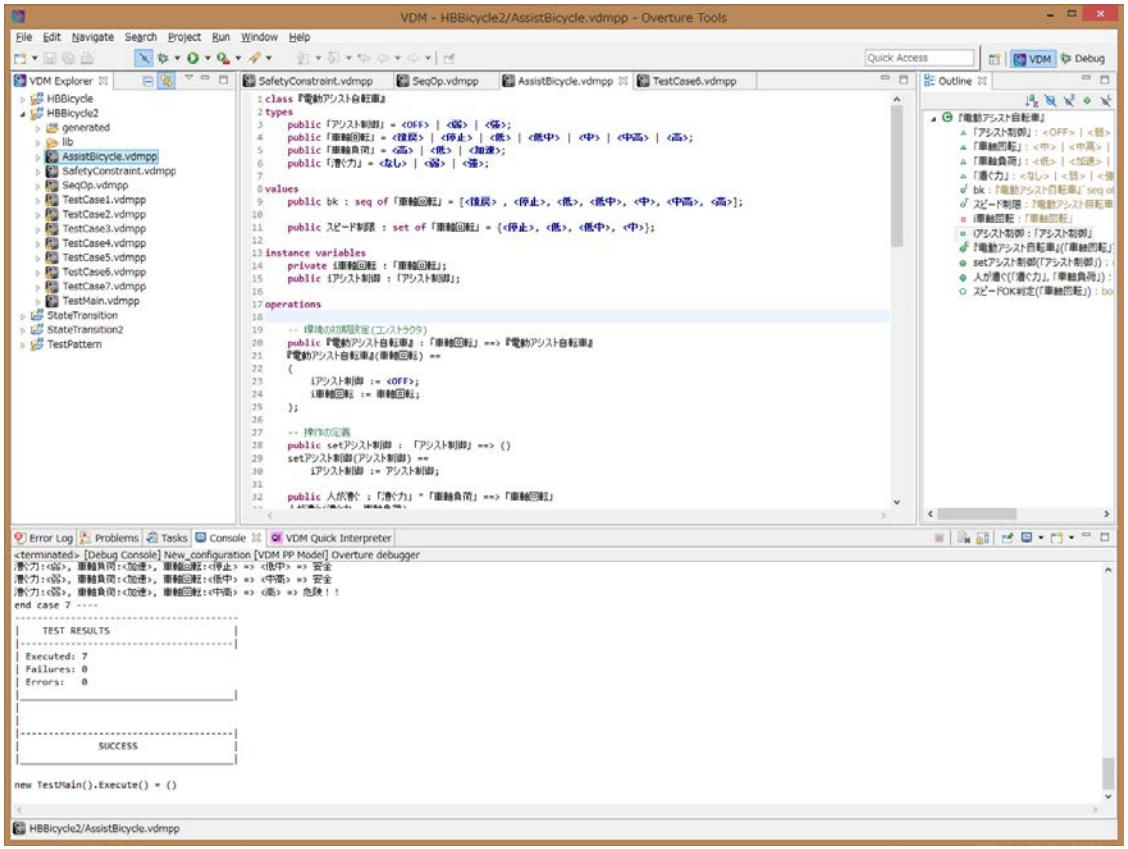
要求仕様のシミュレータは？

形式手法 VDM++

どうすれば良い要求仕様を作れるか？

操作シナリオベース開発手法

Overture Toolsを用いて要求仕様をシミュレーションする！



操作シナリオを作成し、そのシナリオに沿って要求仕様のシミュレーションを行い、振る舞いをチェックする。

どうすれば良い要求仕様を作れるか？

操作シナリオベース開発手法

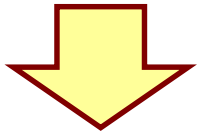
- 形式手法であるVDM++を用いて要求仕様の振る舞いシミュレーションを行う。
- 要求仕様をVDM++で書き、具体的な適用例をケーススタディ・シナリオとしてVDM++で実行し、検討する。

- 形式手法は、論理（モデル作成）が正しくないとエラーが発生する。
- 「とりあえず後回し」は通用しない。
- VDM++で書くことにより、モノの本質が分かって来る（思考するための有効なツール）。

提案

良い要求仕様を作成するには、

- 操作シナリオを作成する。
- 操作シナリオを形式手法（VDM++）を使って仕様シミュレーションを行う。
- シミュレーションの結果により要求仕様を修正することを繰り返す。



試行錯誤

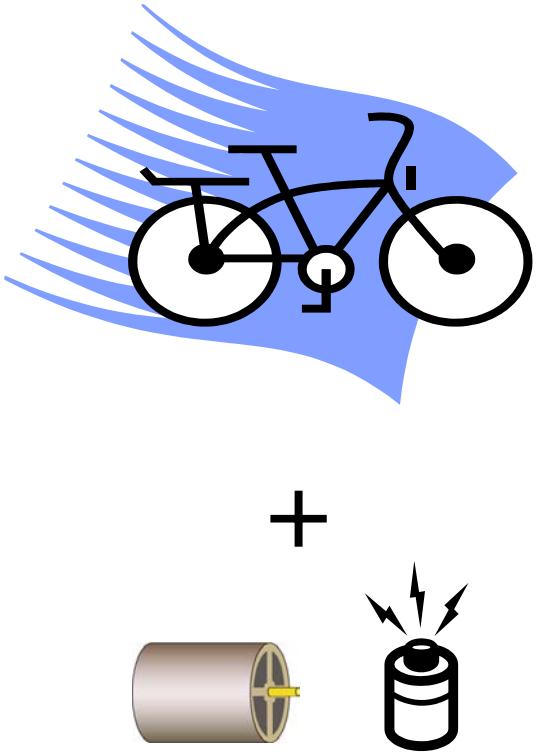
次第に良い要求仕様に近づける。

思考実験

思考実験課題

電動アシスト自転車の安全に関する仕様を
「**操作シナリオ開発手法**」で検討する。

電動アシスト自転車（１）

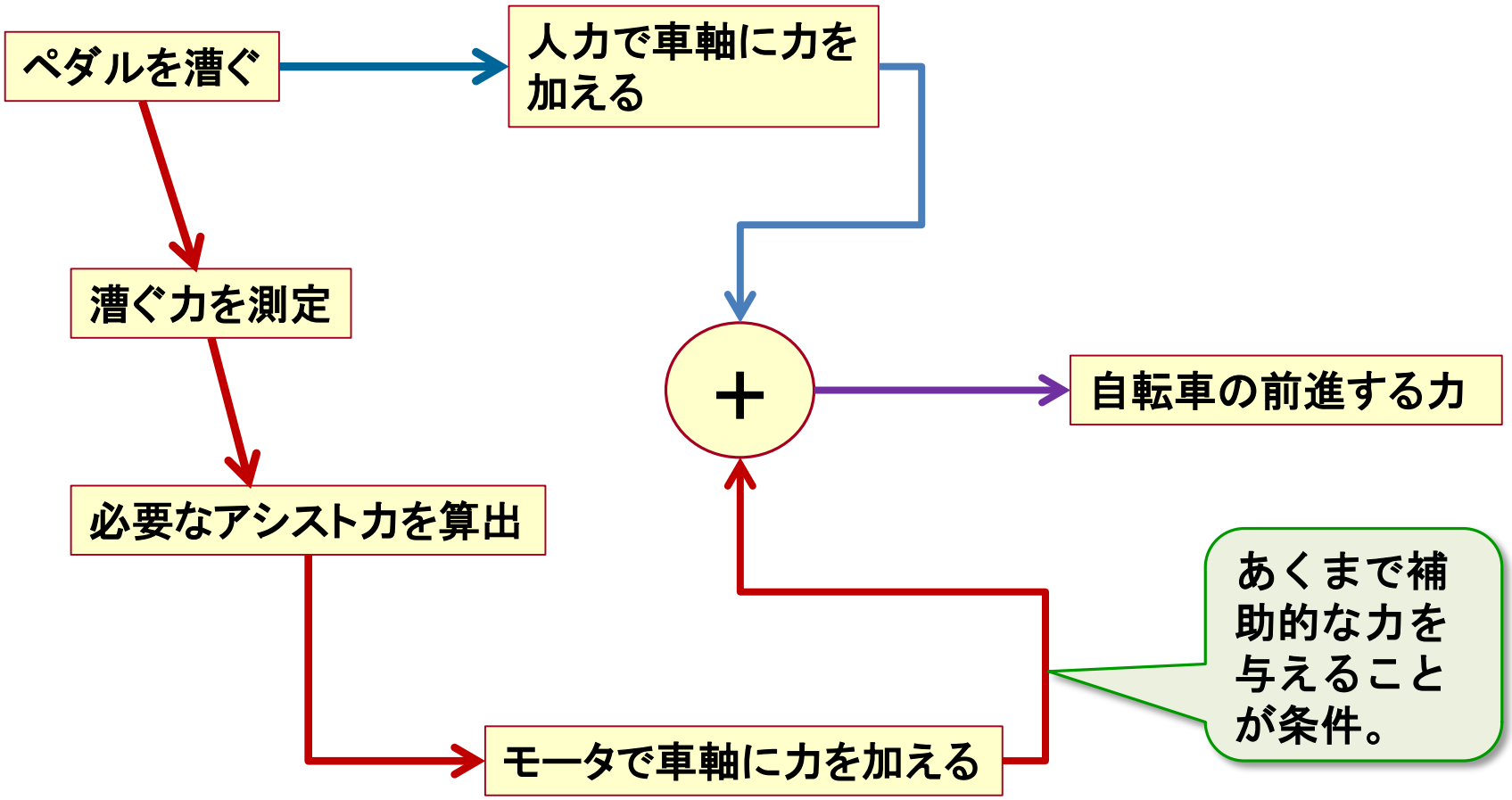


一般的な特徴

1. 自転車にモータとバッテリをつける。
2. ペダルを漕ぐ力をアシストする。
   （上り坂が苦にならない）
3. スピードは出ない。
   （早く走ることが目的でない）
4. モータだけでは自走不可。
   （免許不要、自転車扱いにするため）

電動アシスト自転車（２）

操作シナリオベース開発手法



電動アシスト自転車は、バッテリとモータを搭載した*自転車*　とする。

課題解決手順（１）

要求仕様をモデルを使って考える。
操作を中心に考える。

- **仕様を単純化するために、箇条書きで考える。**
- **定量的では無く、定性的に考える。**注）

注）
最初から無暗に精密に考えることは不要。
本質に関係ないところで苦労することになる。
精密化は後段階に回す。

課題解決手順（２）

注）定量的な表現とは？

自転車の速度： $v_0 + \alpha * t$

走行距離： $v_0 * t + (1/2) * \alpha * t^2$

時間的に正確にシミュレーションしないと意味ある値とならない！
質量とか力とかのパラメータも実際に近い値でないと意味ある結果が出ない。

ここで行おうとしている仕様のチェックには向かない。

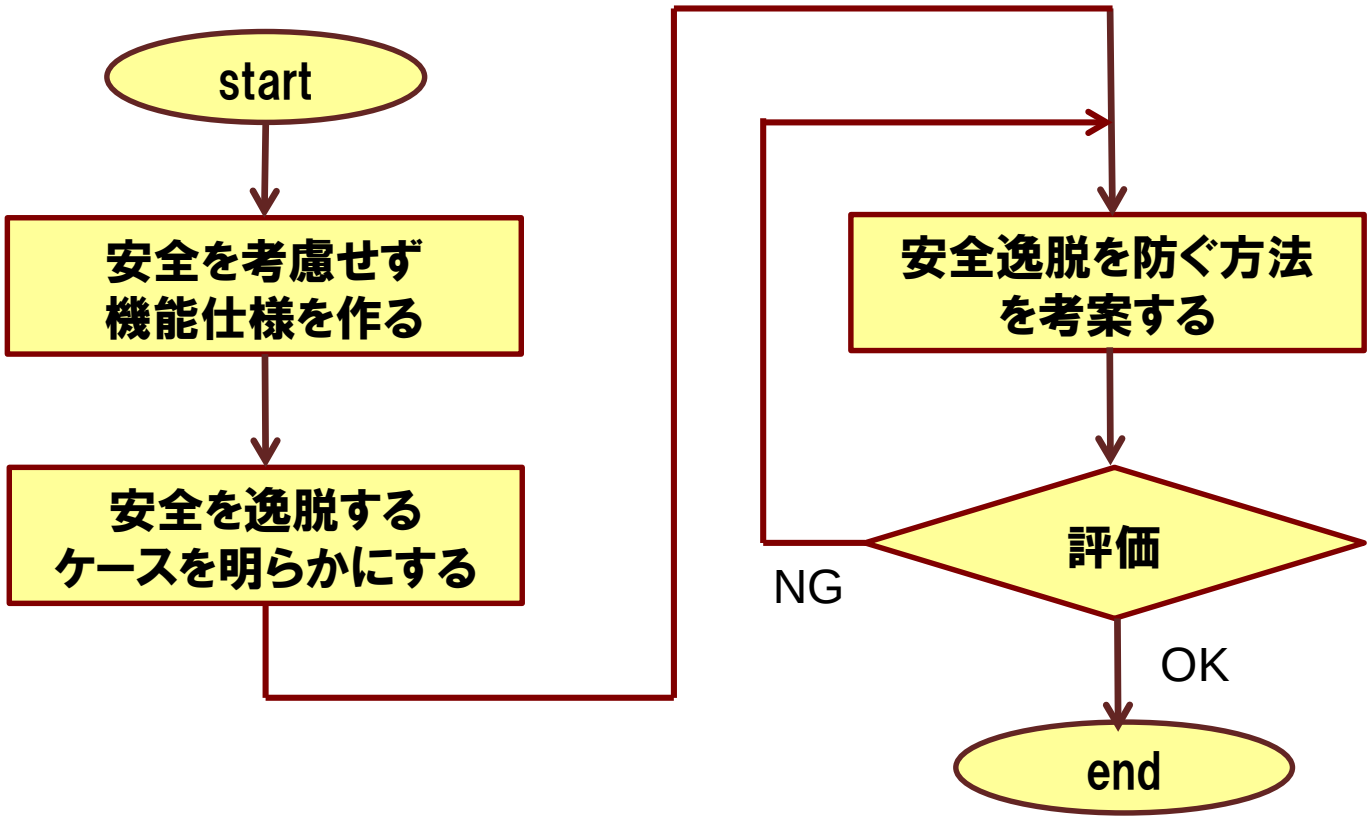
課題解決手順（３）

操作ベースのシナリオをベースとして、

（１）安全考慮なしの仕様を作成する。
（２）仕様を検証する。
（３）安全を脅かす反例を作成する。
（４）安全機能を追加する。
（５）安全機能を検証する。

課題解決手順（４）

安全を考える手順フロー



課題解決手順（５）

操作シナリオ作成の視点

1. 電動アシストなしで、停止状態から自転車を漕ぎ、加速する。
   -- リファレンスとなる動きをモデル上で表す。

2. 電動アシストを活かし、停止状態から自転車を漕ぎ、加速する。
   -- 電動アシストを活かすと、電動なしリファレンスより楽に安全に加速可能となることを表す。

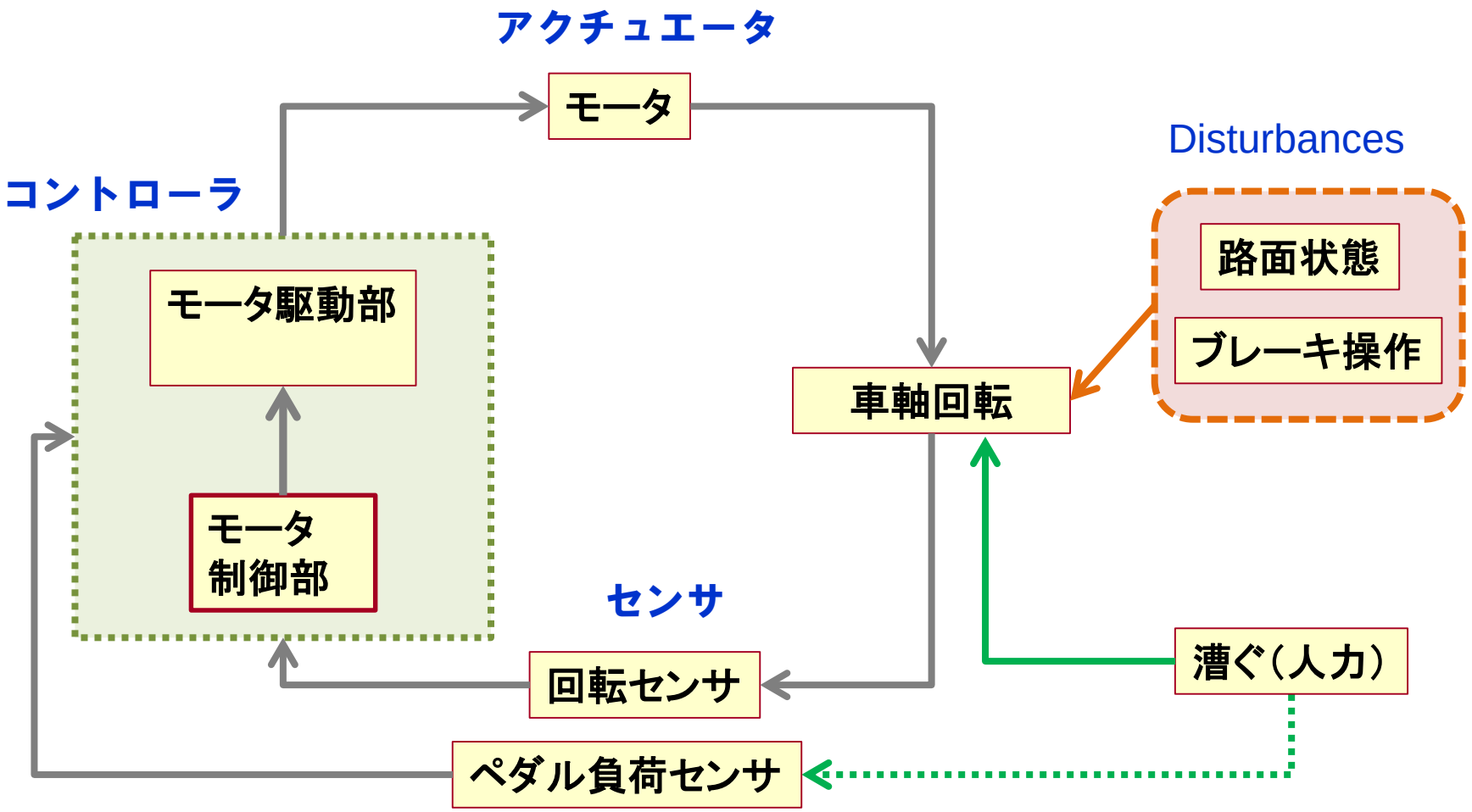
3. 環境条件としては、平地、上り坂、下り坂を想定し、電動アシストあり、なしを比べる。

4. 電動アシスト使用時にブレーキを掛けた場合、電動アシストなしと同等の動きとなることを確かめる。

電動アシスト自転車モデリング（１－１）

操作シナリオベース開発手法

電動アシスト自転車制御システム



電動アシスト自転車モデリング（１－２）

とりあえず、モデル図に従ってVDM++で機能仕様を
モデル化してみよう！

電動アシスト自転車モデリング（１－３）

状態の定義

types

「自転車速度」= <停止> | <低> | <中> | <危険速度>;
「漕ぐ力」= <なし> | <弱> | <強>;
「道路勾配」= <平坦> | <上り坂> | <下り坂>;
「ハンドル操作」= <右> | <直進> | <左>;
「進行方向」= <前方右> | <前方直進> | <前方左> | <後方>;
「ブレーキ操作」= <弱> | <通常> | <強>;

「アシスト機能」= <有効> | <無効>;

定性的、概念的な状態を表現する！
保持する状態と操作時の操作の両方

電動アシスト自転車モデリング（１－４）

インスタンス変数の定義

instance variables

public　i自転車速度　:「自転車速度」:= <停止>;
public　i道路勾配　:「道路勾配」　:= <平坦>;
public　i進行方向　:「進行方向」　:= <前方直進>;
public　iアシスト機能　:「アシスト機能」:= <無効>;

保持する状態を格納する変数の定義

注）それぞれの状態の数は少なくても、状態を保持する変数の数が増大すると、
その組合せ数は膨大な数になる！

電動アシスト自転車モデリング（１－５）


```
public 漕ぐ：「漕ぐ力」 ==> ()
漕ぐ(力) ==
    let 現在速度：「自転車速度」 = i自転車速度
    in
        if 力 = <なし> then (
            iアシスト機能 := <無効>;
            if i道路勾配 = <上り坂> then (
                i自転車速度 := set速度マイナス(現在速度);
            ) else if i道路勾配 = <平坦> then (
                skip; -- 速度変化なし
            ) else if i道路勾配 = <下り坂> then (
                i自転車速度 := set速度プラス(現在速度);
            )
```

アシスト機能は漕がないと無効化される。

電動アシスト自転車モデリング（１－６）

操作シナリオベース開発手法

```
) else if 力 = <強> then (
    if i道路勾配 = <上り坂> then (
            if iアシスト機能 = <有効> then (
                i自転車速度 := set速度プラス(set速度プラス(現在速度))
            ) else (
                i自転車速度 := set速度プラス(現在速度);
            )
    ) else if i道路勾配 = <平坦> then (
            if iアシスト機能 = <有効> then (
                i自転車速度 := set速度プラス(set速度プラス(現在速度))
            ) else (
                i自転車速度 := set速度プラス(現在速度);
            )
    ) else if i道路勾配 = <下り坂> then (
            if iアシスト機能 = <有効> then (
                i自転車速度 := set速度プラス(set速度プラス(set速度プラス(現在速度)))
            ) else (
                i自転車速度 := set速度プラス(現在速度);
            )
    )
    iアシスト機能 := <有効>;
);
```

アシスト機能は強く漕ぐと有効化される。

電動アシスト自転車モデリング（１－７）

．．．しかし、複雑すぎて手におえない！

考慮することが多すぎ、シミュレーションするには**多くの仮定**が必要になる。

単に、要求仕様をモデル図に従ってVDM++で記述するだけではダメ！

モデリングの手法を工夫する必要がある！

電動アシスト自転車モデリング（２－１）

操作シナリオベース開発手法

モデリング上の問題点

- 人間の操作と電動アシストシステムとの関係が複雑
- 人間の操作（ペダルを漕ぐ）ことと、スピードの関係のパラメータが多い

パラメータが多く、扱い辛い

いろいろ仮定しないと進まない！

- 電動アシストシステムと人間との関係を**単純化したモデル**が必要

⇒ STAMPを使ってモデルを整理してみる。

電動アシスト自転車モデリング（２－２）

STAMP Process Models

1. 安全は、「故障」問題としてとらえるのではなく、コントロール問題として扱う。

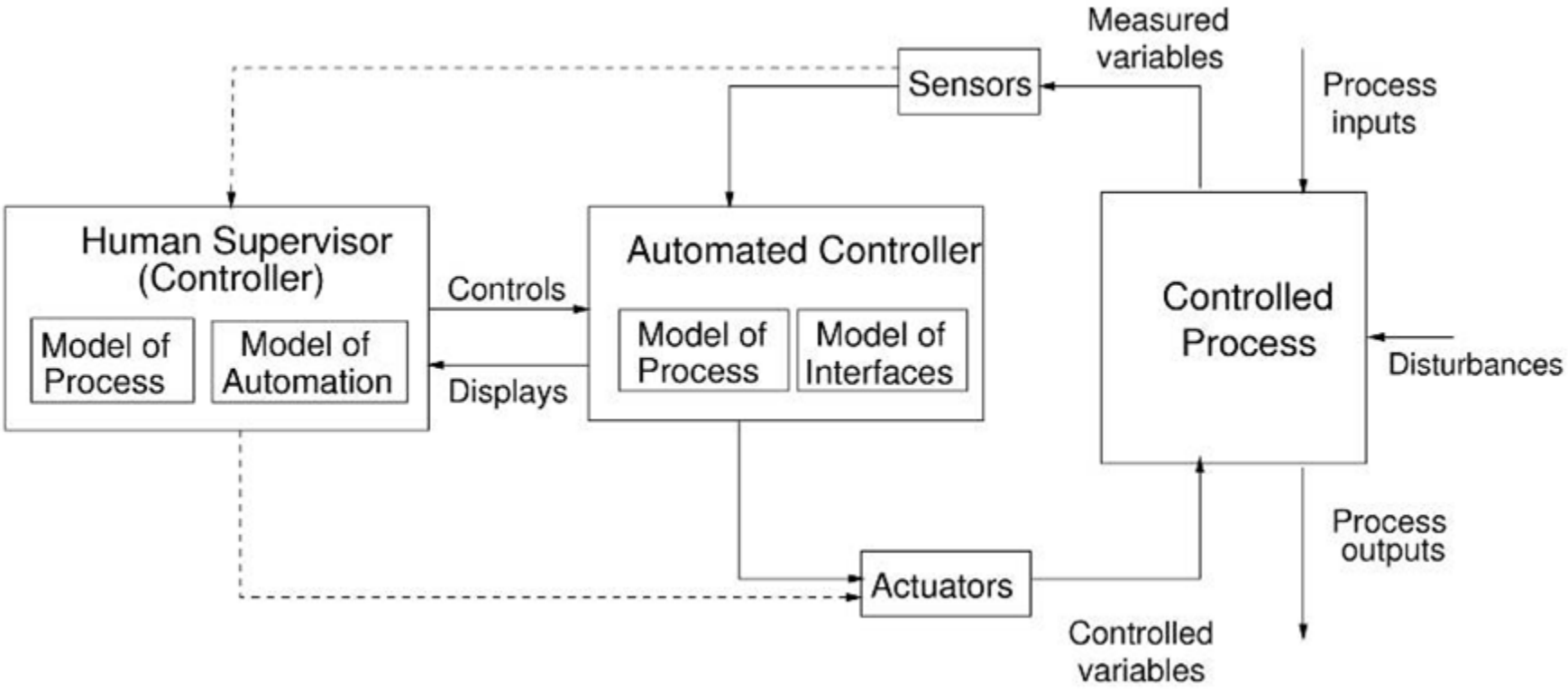
2. 事故は単なるイベントまたはイベントの連鎖ではない。
   - 複合した動的なプロセスを含む
   - 人間、機械、環境の間の相互作用によって生じる

注）Prof. Nancy Leveson、Future Trends in Process Safety プレゼン資料より引用

電動アシスト自転車モデリング（２－３）

操作シナリオベース開発手法

STAMP Process Models

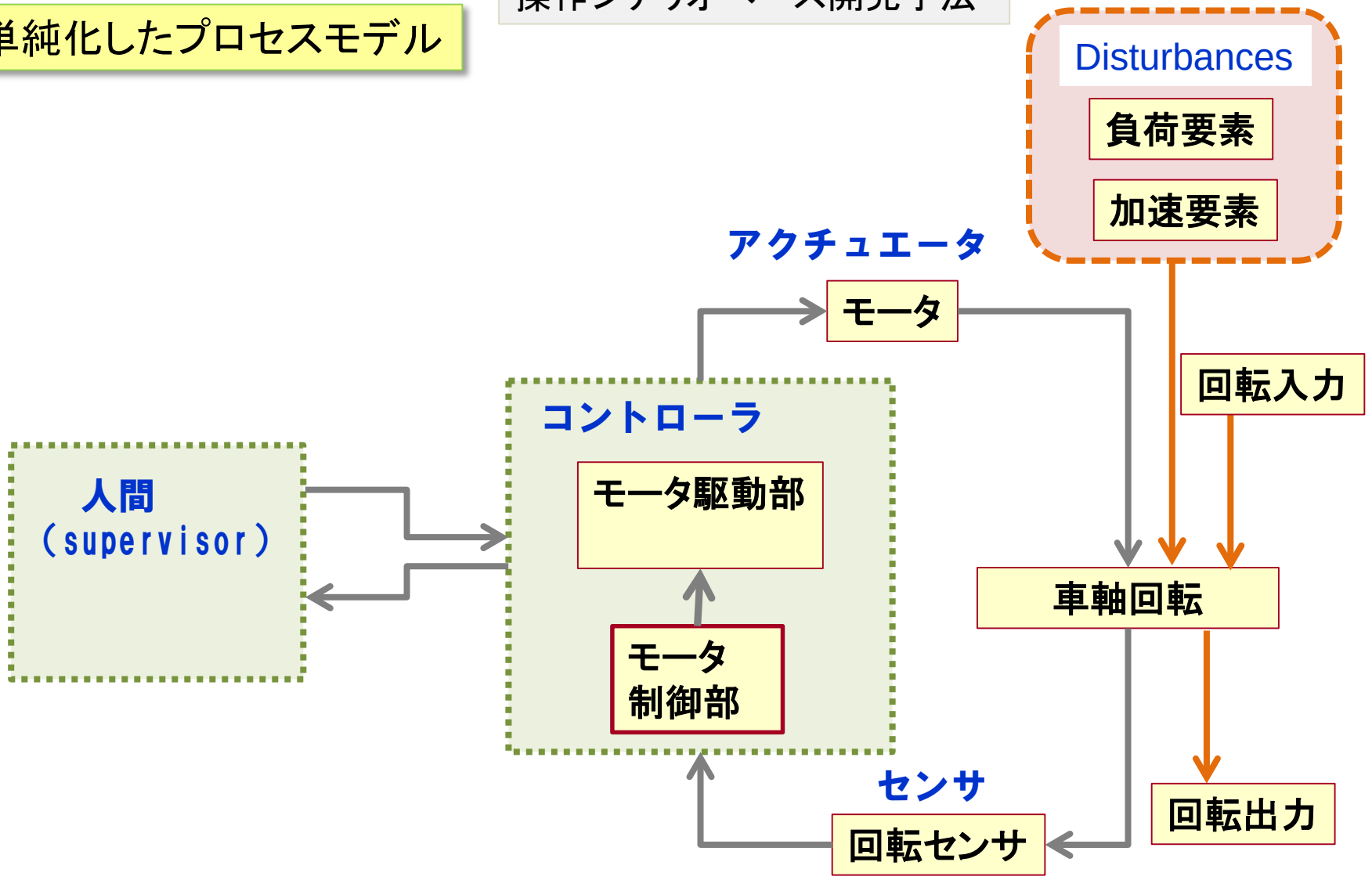


注）Prof. Nancy Leveson、Future Trends in Process Safety プレゼン資料より引用

電動アシスト自転車モデリング（２－４）

操作シナリオベース開発手法

単純化したプロセスモデル



モデリング手順（再記）

1. 安全考慮なしでVDM++の仕様を書く。
2. シナリオに沿ってシミュレーション実行を行い、仕様を確認する。
3. 危険状態になるシナリオを書く。
4. 安全となるように安全仕様を追加する。
5. 安全仕様を含めてシミュレーション実行を行い、安全なことを確認する。

VDM++のクラス定義構成

```
class『電動アシスト自転車』

    types
        -- 型(ここでは状態)の定義
    values

    instance variables
        -- インスタンス変数(状態を保持する変数)の定義

    operations
        -- インスタンス変数を変更する操作の記述

    functions
        -- 関数の記述
    traces

end『電動アシスト自転車』
```

VDM++モデリング – 状態の定義

状態の定義

```
types
    public「アシスト制御」= <OFF> | <弱> | <強>;
    public「車軸回転」= <後戻> | <停止> | <低> | <低中>|<中> | <中高> |<高>;
    public「車軸負荷」= <高> | <低> | <加速>;
    public「漕ぐ力」= <なし> | <弱> | <強>;

values
    public bk : seq of「車軸回転」=
        [<後戻> , <停止>, <低>, <低中>, <中>, <中高>, <高>];
```

定性的、概念的な状態を表現する！
保持する状態と操作時の操作の両方

VDM++モデリング – 保持する状態の定義

インスタンス変数の定義

```
instance variables
        private  i車軸回転 :「車軸回転」;
        public   iアシスト制御 :「アシスト制御」;
```

保持する状態を格納する変数の定義

注）それぞれの状態の数は少なくても、状態を保持する変数の数が増大すると、
その組合せ数は膨大な数になる！

VDM++モデリング – 関数

関数の定義

```
functions
   public static setプラス[@T] : seq of @T * @T -> @T
   setプラス(s, x) ==
                  let h = hd s,  s2 = tl s
                  in
                      if h = x
                      then
                            if len s2 > 0  then  hd s2
                            else h
                       else setプラス[@T](s2, x);
```

汎用的に使えるプラス、マイナス関数を定義しておく！

VDM++モデリング – 関数

操作の記述

```
operations
  -- コンストラクタ
  public 『電動アシスト自転車』:「車軸回転」==>『電動アシスト自転車』
  『電動アシスト自転車』(車軸回転) ==
  (
        iアシスト制御 := <OFF>;
        i車軸回転 := 車軸回転;
  );
  -- 操作の定義
  public setアシスト制御 : 「アシスト制御」==> ()
  setアシスト制御(アシスト制御) ==
              iアシスト制御 := アシスト制御;
```

VDM++モデリング – 保持する状態の操作定義

操作シナリオベース開発手法

```
public 人が漕ぐ : 「漕ぐ力」*「車軸負荷」==>「車軸回転」
人が漕ぐ(漕ぐ力, 車軸負荷) ==
(dcl  rot : 「車軸回転」:= i車軸回転,   -- アシストが無い場合の結果
     rot2 : 「車軸回転」;  -- アシストを行った場合の結果
     IO`print("漕ぐ力:");IO`print(漕ぐ力);IO`print( ", 車軸負荷:" );IO`print(車軸負荷);
     -- アシストが無い場合
     cases 漕ぐ力 :
          <なし> -> rot := SeqOp`setマイナス[「車軸回転」](bk,rot),
          <弱>  -> (cases 車軸負荷 :
                         <高> -> rot := SeqOp`setマイナス[「車軸回転」](bk,rot),
                         <低> -> rot := SeqOp`setプラス[「車軸回転」](bk,rot),
                         <加速> -> rot := SeqOp`set2プラス[「車軸回転」](bk,rot)
                    end;),
          <強>  -> (cases 車軸負荷 :
                         <高> -> rot := SeqOp`setプラス[「車軸回転」](bk,rot),
                         <低> -> rot := SeqOp`set2プラス[「車軸回転」](bk,rot),
                         <加速> -> rot := SeqOp`set3プラス[「車軸回転」](bk,rot)
                    end;)
     end;
     rot2 := rot;
```

車軸負荷と漕ぐ力に応じて速度の変化を記述してみる。

VDM++モデリング – 保持する状態の操作定義

続き

アシストが有効な場合、車軸回転をアシスト追加する。

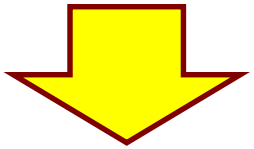
```
-- アシストを「ON」にすると車軸の負荷に関わり無く車軸の回転を制御できるようになる。
cases iアシスト制御 :
      <OFF> -> skip,
      <弱> ->  -- アシスト
            (cases 漕ぐ力 :
                  <なし> -> skip,
                   <弱>, <強> ->  rot2 := SeqOp`set2プラス[「車軸回転」](bk,rot2)
             end;)
       <強> ->  -- アシスト
            (cases 漕ぐ力 :
                  <なし> -> skip,
                  <弱>, <強> ->   rot2 := SeqOp`set3プラス[「車軸回転」](bk,rot2)
              end;)
 end;
 IO`print(“, 車軸回転:” );IO`print(i車軸回転);IO`print( “ => “ );IO`print(rot2);
 i車軸回転 := rot2;
 return  rot2;
);
```

VDM++モデリング – 安全制約の定義


```
class『安全制約』
values
        public 安全スピード : set  of『電動アシスト自転車』`「車軸回転」 =
                        { <停止>, <低>, <低中>, <中>, <中高> };
operations
        public  static  print安全判定 : 『電動アシスト自転車』`「車軸回転」 ==> ()
        print安全判定(スピード) ==
        (
              if  is安全スピード(スピード)
              then  IO`println( “ => 安全” )
              else  IO`println( “ => 危険！ ！” );
         );
functions
       --  安全スピード判定
      public static  is安全スピード : 『電動アシスト自転車』`「車軸回転」 -> bool
      is安全スピード(スピード) ==
              exists  x  in  set 安全スピード  &  スピード = x;
end『安全制約』
```

操作シナリオの作成（１）

- ありそうな通常動作の操作シナリオを作成してみる。
- VDM++上で仕様をシナリオに沿ってシミュレーション実行してみる。



結果が妥当かどうか判断し、妥当でない場合、モデルを修正する。

具体的な操作シナリオを作成することで、何が本質なのかが分かって来る。

操作シナリオの作成（２）

シミュレーションする操作シナリオ

		強く漕ぐ	弱く漕ぐ
アシストなし	平地走行		
	上り坂		
	下り坂		
アシスト弱	平地走行		
	上り坂		
	下り坂		
アシスト強	平地走行		
	上り坂		
	下り坂		

操作シナリオの作成（３）


```
class TestCase1 is subclass of TestCase
operations
    public TestCase1: seq of char ==> TestCase1
            TestCase1 (nm) == setName (nm);
    protected runTest: () ==> ()
    runTest () ==
    (dcl 自転車 : 『電動アシスト自転車』,
         車軸回転 : 『電動アシスト自転車』`「車軸回転」 ;
         IO`print ("start case 1 ----:");IO`println (getName ());

                ここにテストケースを書く


         IO`println ("end case 1 ----");
     );
end TestCase1
```

操作シナリオの作成（４）

テストケース（実行シナリオ）の例

```
IO`println ("アシストなしで停止状態から強く漕いでみる！");
自転車 := new 『電動アシスト自転車』 (<停止>);
自転車.setアシスト制御 (<OFF>);
『安全制約』`print安全判定 (自転車.人が漕ぐ ( <強>, <低> ) );;

IO`println ("アシスト弱で停止状態から強く漕いでみる！");
自転車 := new 『電動アシスト自転車』 ( <停止> );
自転車.setアシスト制御 ( <弱> );
『安全制約』`print安全判定 (自転車.人が漕ぐ ( <強>, <低> ) );

IO`println ("アシスト強で停止状態から強く漕いでみる！");
自転車 := new 『電動アシスト自転車』 ( <停止> );
自転車.setアシスト制御 ( <強> );
『安全制約』`print安全判定 (自転車.人が漕ぐ ( <強>, <低> ) );
```

操作シナリオの作成（５）

テストケースの実行結果例

```
**
** Overture Console
**
start case 1（アシストなし） ----:(Case 1)
平地：アシストなしで停止状態から強く漕いでみる！
漕ぐ力:<強>, 車軸負荷:<低>, 車軸回転:<停止> => <低中> => 安全
上り坂：アシストなしで停止状態から強く漕いでみる！
漕ぐ力:<強>, 車軸負荷:<高>, 車軸回転:<停止> => <低> => 安全
下り坂：アシストなしで停止状態から強く漕いでみる！
漕ぐ力:<強>, 車軸負荷:<加速>, 車軸回転:<停止> => <中> => 安全
平地：アシストなしで停止状態から弱く漕いでみる！
漕ぐ力:<弱>, 車軸負荷:<低>, 車軸回転:<停止> => <低> => 安全
上り坂：アシストなしで停止状態から弱く漕いでみる！
漕ぐ力:<弱>, 車軸負荷:<高>, 車軸回転:<停止> => <後戻> => 危険！！
下り坂：アシストなしで停止状態から弱く漕いでみる！
漕ぐ力:<弱>, 車軸負荷:<加速>, 車軸回転:<停止> => <低中> => 安全
end case 1 ----
```

操作シナリオの作成（６）

- 本件の場合、シミュレーションの実行回数と車軸のスピードの関係が適切になっているかどうか確かめる。
- NGの場合、「車軸回転」の分け方と「人が漕ぐ」操作の内容を修正する。

モデルは 現実のすべてをシミュレーションする必要はない。

モデルは その適用限界を把握して使用することにより単純化する。

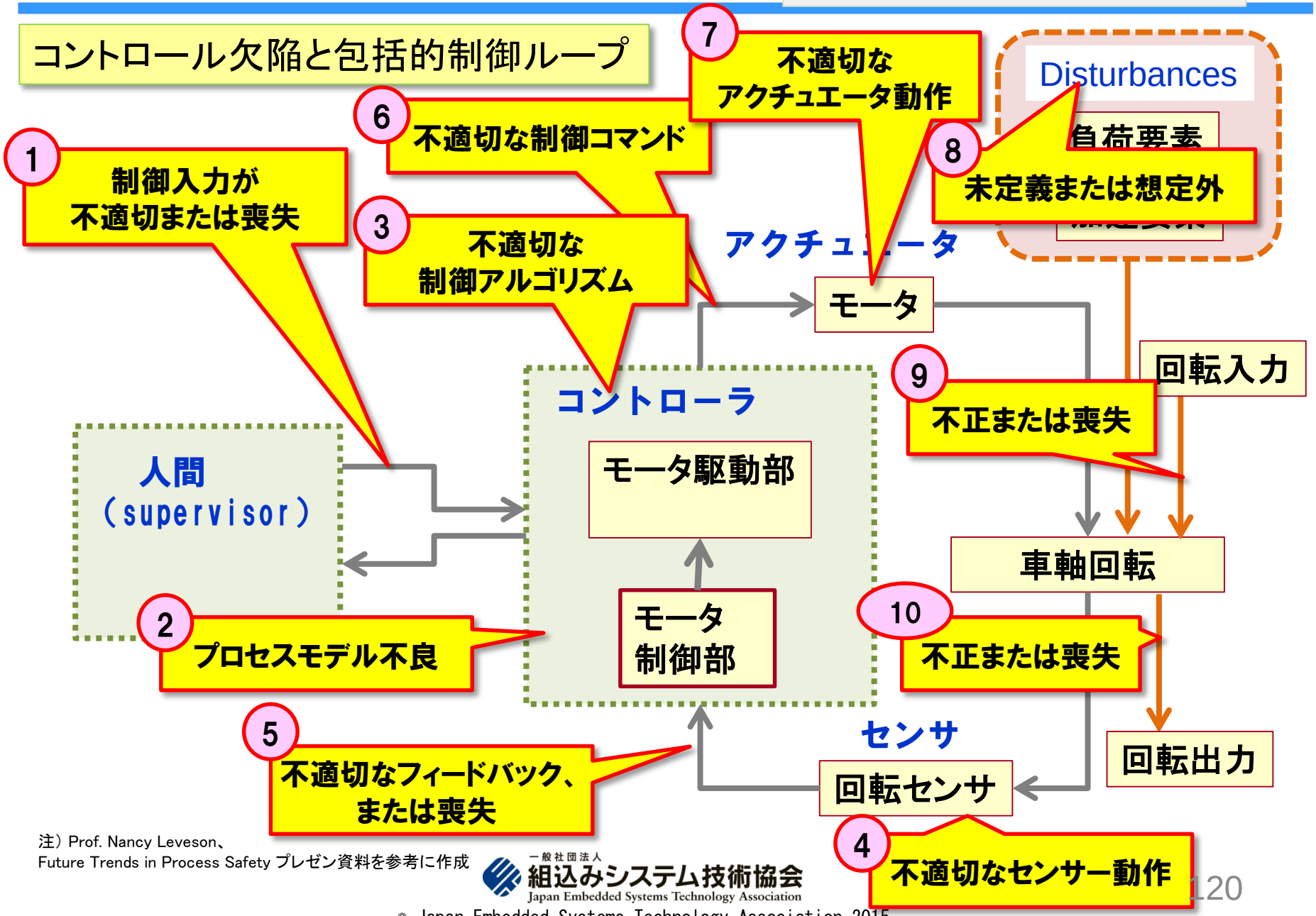
基本ケースについて妥当な結果が出るようにモデルを修正する。

テストケースのシナリオとシミュレーション結果

基本機能のシミュレーション結果

		強く漕ぐ	弱く漕ぐ
アシストなし	平地走行	安全	安全
	上り坂	安全	危険(逆行)
	下り坂	安全	安全
アシスト弱	平地走行	安全	安全
	上り坂	安全	安全
	下り坂	危険	安全
アシスト強	平地走行	危険	安全
	上り坂	安全	安全
	下り坂	危険	危険

ハザード分析（１）



注）Prof. Nancy Leveson、
Future Trends in Process Safety プレゼン資料を参考に作成

ハザード分析（２）

ハザード分析例

注）「（安全側）」は状況による。

No	部位	要因	影響	結果	故障の分類
1	人間の操作	不適切または喪失	電動アシスト動作をONにできない。	電動アシスト機能が使えない。	安全側
			電動アシスト動作をOFFにできない。	意思に反して電動アシストが効いてしまう。	危険側
2	コントローラ・ソフト	プロセスモデル不良	ブレーキ操作が考慮されていない。	ブレーキを掛けたときの動きが保障されない。	危険側
3	コントローラ・ソフト	不適切なアルゴリズム	モータ出力が大き過ぎる。	スピードが出過ぎる。	危険側
			モータ出力が小さ過ぎる。	電動アシストが効かない。	安全側
4	回転センサ	不適切なセンサー動作	車軸の回転を検出しない。	漕がなくてもモータが効く。	危険側
			車軸の実際の回転より少なく検出する。	スピードが出過ぎる。急加速発生。	危険側
			車軸の実際の回転より多く検出する。	電動アシストの効きが良くない。	（安全側）
5	センサ信号	不適切なフィードバックまたは喪失	車軸の回転情報が得られない。	急加速発生。	危険側
6	モータ制御信号	不適切な制御コマンド	モータが動作しない。	電動アシストが効かない。	安全側
7	モータ制御信号	不適切なモータ動作	モータ出力が指令値に達しない。	電動アシストの効きが良くない。	（安全側）
8	車軸回転外力	未定義または想定外の外乱	ブレーキ操作が考慮されていない。	ブレーキを掛けたときの動きが保障されない。	危険側
9	回転入力	回転入力の不正または喪失	車輪の回転が車軸に伝わらない。	スピードが検出できない。	危険側
10	回転出力	回転出力の不正または喪失	車軸の回転を車輪に伝えられない。	モータが空回りする。	（安全側）

ハザード分析（３）

どうすれば危険側故障の対策が可能か？

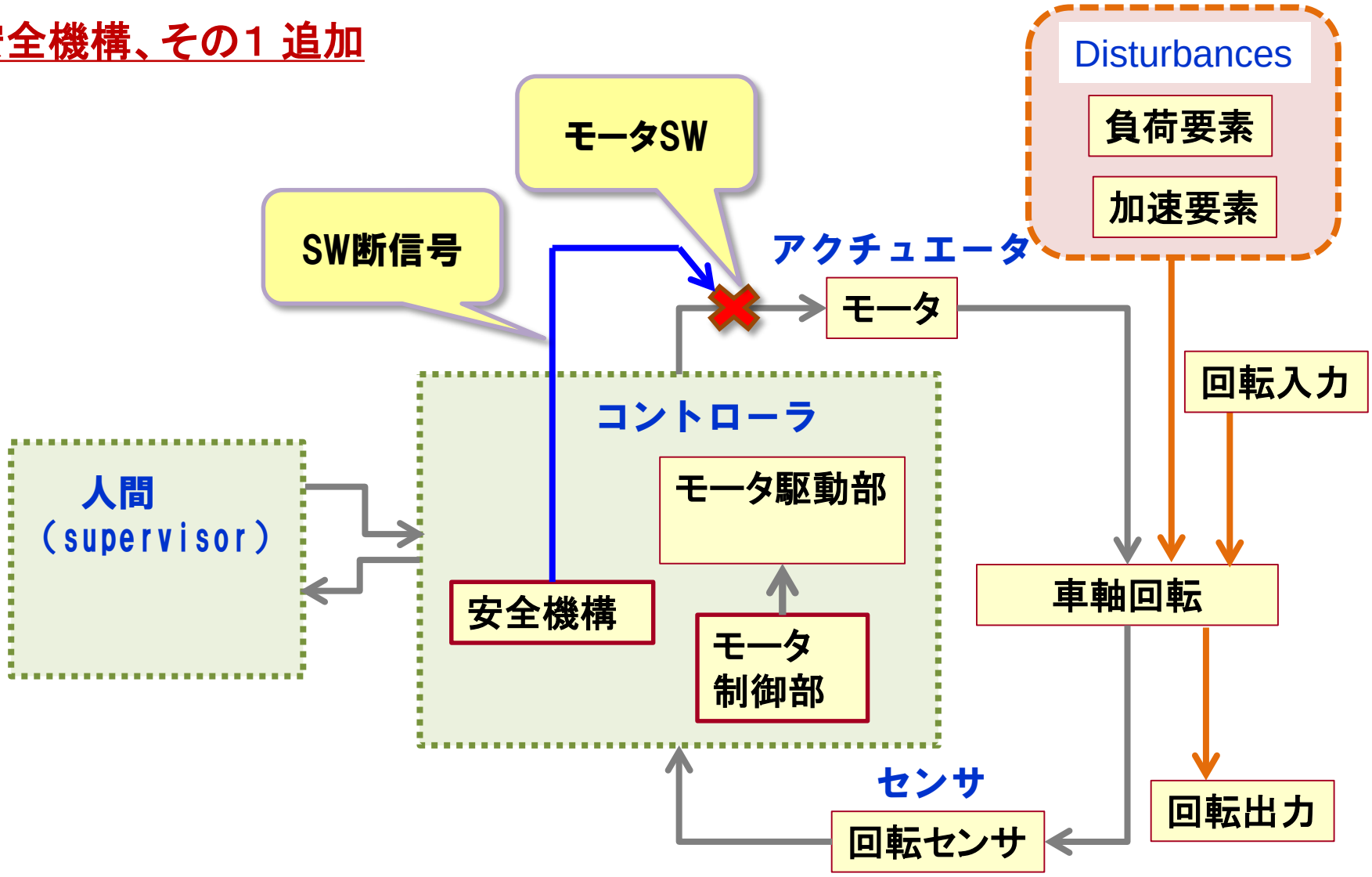
安全機構、その１：

危険スピードが出ないようにアシストを制限する。
危険スピードに達した場合、アシストを無効化する。

No	部位	要因	影響	結果	故障の分類	安全機構	
						その1	
1	人間の操作	不適切または喪失	電動アシスト動作をOFFにできない。	意思に反して電動アシストが効いてしまう。	危険側	？？	
2	コントローラ・ソフト	プロセスモデル不良	ブレーキ操作が考慮されていない。	ブレーキを掛けたときの動きが保障されない。	危険側	？？	
3	コントローラ・ソフト	不適切なアルゴリズム	モータ出力が大き過ぎる。	スピードが出過ぎる。	危険側	安全スピードに抑えられる	
4	回転センサ	不適切なセンサー動作	車軸の回転を検出しない。	漕がなくてもモータが効く。	危険側	？？	
			車軸の実際の回転より少なく検出する。	スピードが出過ぎる。急加速発生。	危険側	？？	
5	センサ信号	不適切なフィードバックまたは喪失	車軸の回転情報が得られない。	急加速発生。	危険側	？？	
8	車軸回転外力	未定義または想定外の外乱	ブレーキ操作が考慮されていない。	ブレーキを掛けたときの動きが保障されない。	危険側	？？	
9	回転入力	回転入力の不正または喪失	車輪の回転が車軸に伝わらない。	スピードが検出できない。	危険側	？？	

安全機構の追加

安全機構、その1 追加



安全機構の追加

■ アシストを使用した場合のスピードの出方が、アシストを使用しない場合と同等にする。

```
values
    public スピード制限 : set of 「車軸回転」 = {<停止>, <低>, <低中>,<中>};

functions
    public スピードOK判定 : 「車軸回転」 -> bool
    スピードOK判定 (rot) ==
            exists x in set スピード制限 & rot = x;
```

車軸回転スピードの判定関数を作成し、操作に組み込む

安全機構の追加

アシストを使用すると安全制約に違反する場合、
アシストを無効化する。

```
cases  iアシスト制御：
    <OFF> -> skip,
    <弱> ->  -- アシスト
          (cases 漕ぐ力：
              <なし> -> skip,
              <弱>, <強> ->
                    (    rot2 := SeqOp`set2プラス[「車軸回転」](bk,rot2);
                      if not スピードOK判定(rot2)  then  rot2 := rot;  )
          end;),
    <強> ->  -- アシスト
          (cases 漕ぐ力：
              <なし> -> skip,
              <弱>, <強> ->
                    (    rot2 := SeqOp`set3プラス[「車軸回転」](bk,rot2);
                      if not スピードOK判定(rot2)  then  rot2 := rot;  )
          end;)
end;
```

安全機構の追加

安全機構付加のシミュレーション結果(<停止>状態～)

		強く漕ぐ	弱く漕ぐ
アシストなし	平地走行	安全	安全
	上り坂	安全	危険(逆行)
	下り坂	安全	安全
アシスト弱	平地走行	安全	安全
	上り坂	安全	安全
	下り坂	安全	安全
アシスト強	平地走行	安全	安全
	上り坂	安全	安全
	下り坂	安全	安全

停止状態からアシストを有効にすると安全範囲が広がる！

安全機構の追加

安全機構付加のシミュレーション結果（車軸回転<中>状態～）

		強く漕ぐ	弱く漕ぐ
アシストなし	平地走行	危険	安全
	上り坂	安全	安全
	下り坂	危険	危険
アシスト弱	平地走行	危険	安全
	上り坂	安全	安全
	下り坂	危険	危険
アシスト強	平地走行	危険	安全
	上り坂	安全	安全
	下り坂	危険	危険

走行中にアシストを有効にしても、危険はアシストなしと同程度！

安全機構の追加

- 車軸の回転速度を監視し、管理値逸脱で、アシストを無効化することで安全になることが分かった。

- 安全の確保する場合のアシストの無効化は、**車軸の回転速度のみのパラメータを監視**することで実現可能である（道路環境、アシストの強弱は不要）。

スピードの監視を行うことでOKか ？？？

注）安全機構のその2をまだ考慮していないということでは無い！

安全機構の追加

漕ぎ続けることをシミュレーションすると・・・・

```
IO`println("平地：アシスト強で停止状態から強く漕いでみる！");
自転車 := new 『電動アシスト自転車』( <停止> );
自転車.setアシスト制御( <強> );
『安全制約』`print安全判定(自転車.人が漕ぐ( <強>, <低> ));
『安全制約』`print安全判定(自転車.人が漕ぐ( <強>, <低> ));
『安全制約』`print安全判定(自転車.人が漕ぐ( <強>, <低> ));
```

```
平地：アシスト強で停止状態から強く漕いでみる！
漕ぐ力:<強>, 車軸負荷:<低>, 車軸回転:<停止> => <低中> => 安全
漕ぐ力:<強>, 車軸負荷:<低>, 車軸回転:<低中> => <中高> => 安全
漕ぐ力:<強>, 車軸負荷:<低>, 車軸回転:<中高> => <高> => 危険！！
```

がんばって漕ぐと危険スピードに到達してしまう！

想定外の結果

安全機構の追加

何故？？？

VDM++の結果を解釈すると。。。

- 頑張って漕いで、アシスト動作の閾値を超えるとアシストは動作しないが、**自力で安全速度を超えることが可能**となる。
  （走行中にアシストを有効にするケースも同様）

人間の想像力は、固定概念をなかなか越えられないが、ロジックは正直！分かった後で考えると「マットウ」なことだけど・・・・。

電動アシストは、人間の力のアシストにより、人間の力を拡大するがアシストにできる安全機能もシステム的な限界がある。

安全機構の追加

さらに

- 安全機構のその２を追加して危険側故障に対応する。
- 安全機能その２でも危険側故障が残る場合、さらにその３を追加する。・・・・・

- 安全機構は、コストと危険のバランスを考えて実施する範囲を決める（商品の便利性とリスク、価格のバランス）。

思考実験結果のまとめ

- 要求仕様は、形式手法と操作シナリオの組み合わせを用いて試行錯誤を繰り返すことで段階的に完成度を高める。
- モデリングの極端なケースの操作シナリオ実行により、システム的な欠陥に気づくことができる。
- モデルは現実を単に抽象したものではなく、目的に合わせて取捨選択したものが必要。
- 操作シナリオが期待通りに動かないのは、モデルが不十分なのではなく、人間の思考が間違っているケースもある。

まとめ（１）

まとめ（２）

- 要求の背景にある「意図」を操作シナリオで表現することにより、「意図」が顕在化できる。

- 要求仕様とその操作シナリオとが一体化することにより、「意図」を実現する「要求」をブレなく理解可能。

- VDM++等の形式手法を用い、操作シナリオをシミュレーションすることにより、要求の論理的な正しさを担保することができる。

今後の課題

- 形式手法でシミュレーションしながら要求仕様を作成する技法スタイルの確立。

- ベースとなる仕様に「但し書き」（例外）を取り込みながら、要求仕様を完成に持って行くためのスタイル、手法。

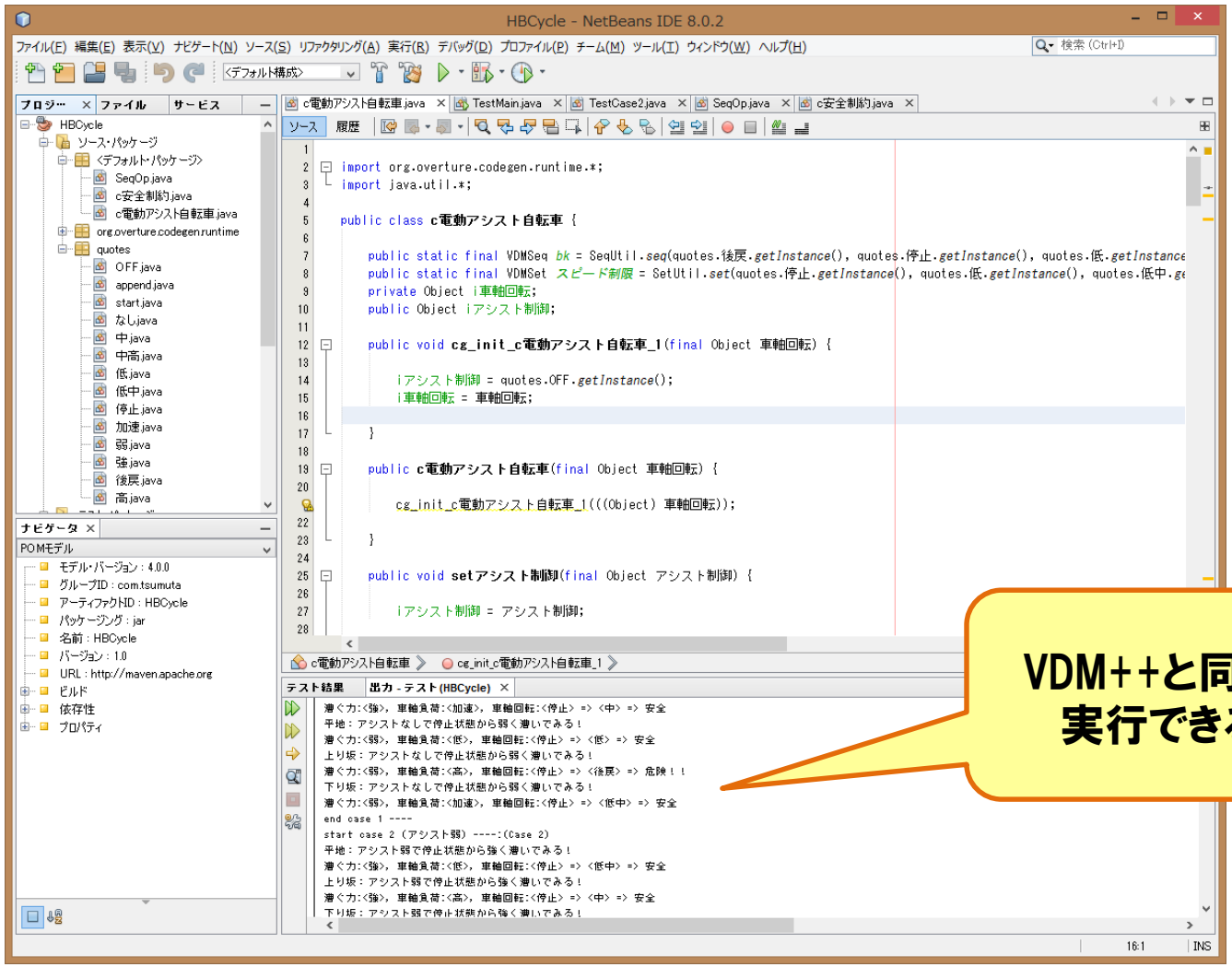
付録 － VDM＋＋からJavaソースの生成

■ Overture ツールを用いてVDM++の仕様記述から Java言語ソースを自動生成してみる。

ただし、以下の作業を行う。

1. 生成したソースをIDEに読み込めるようにmavenでプロジェクトを作成し、配置する。
2. Eclipse、NetBeans等で作成したmavenプロジェクトを開き、ソースを自動整形する（インデント等が乱れているのを修正）。
3. 『』、「」はJavaコンパイラが受け付けないので取り除く。（修正はIDEの機能を用いることで簡単に可能）
4. OvertureのJava実行用ランタイムをプロジェクトに加える。（org.overture.codegen.runtimeパッケージ）
5. Junit（単体テストツール）をプロジェクトから参照可能にする。（UNITテストを使用する場合）

付録 － VDM＋＋からJavaソースの生成



付録 － VDM＋＋からJavaソースの生成

自動生成Javaソースの評価

1. 動作するソースが得られる。

2. 実用的かどうかは疑問あり。
   （コレクションの扱い等が非効率）

3. 日本語でVDM++を作成するとJavaソースも日本語のものが生成される（実行可能だが・・・）。

3.4 SPINを活用する仕様検証

形式検証(SPIN)による仕様記述実験

◆モータ制御用インバータの起動処理(プリチャージ制御)を題材として、形式検証(SPIN)による仕様記述(仕様検証)実験を行う。

1. 形式検証(SPIN)概要
2. 仕様検証手順
3. 仕様検証例
4. 考察

形式検証（SPIN）の概要-1

活用目的と開発プロセスの関係

目的 / プロセス	仕様記述	コード生成	特性検証	バグ検出
要求分析	VDM Event-B Z	B SCADE	LTSA UPPAAL SPIN	
設計				
実装				ESC/Java2 Spec# VARVEL
テスト	注1		注1	

注1:テストケース生成。

SPINは特性検証、
バグ検出に使用可能

形式検証（SPIN）の概要-2

SPINは、G.J.Holazmann博士が中心となって1980年代に開発した、ソフトウェアの信頼性を向上させるための技術である。

デザインやプログラムの正しさを自動的に検査する形式検証ツールの中で、もっとも実用的なものである。

オートマトン、時相論理、グラフ・アルゴリズムの3つの基礎要素からなる総合ツールである。

公開されているため、誰でも利用が可能である。

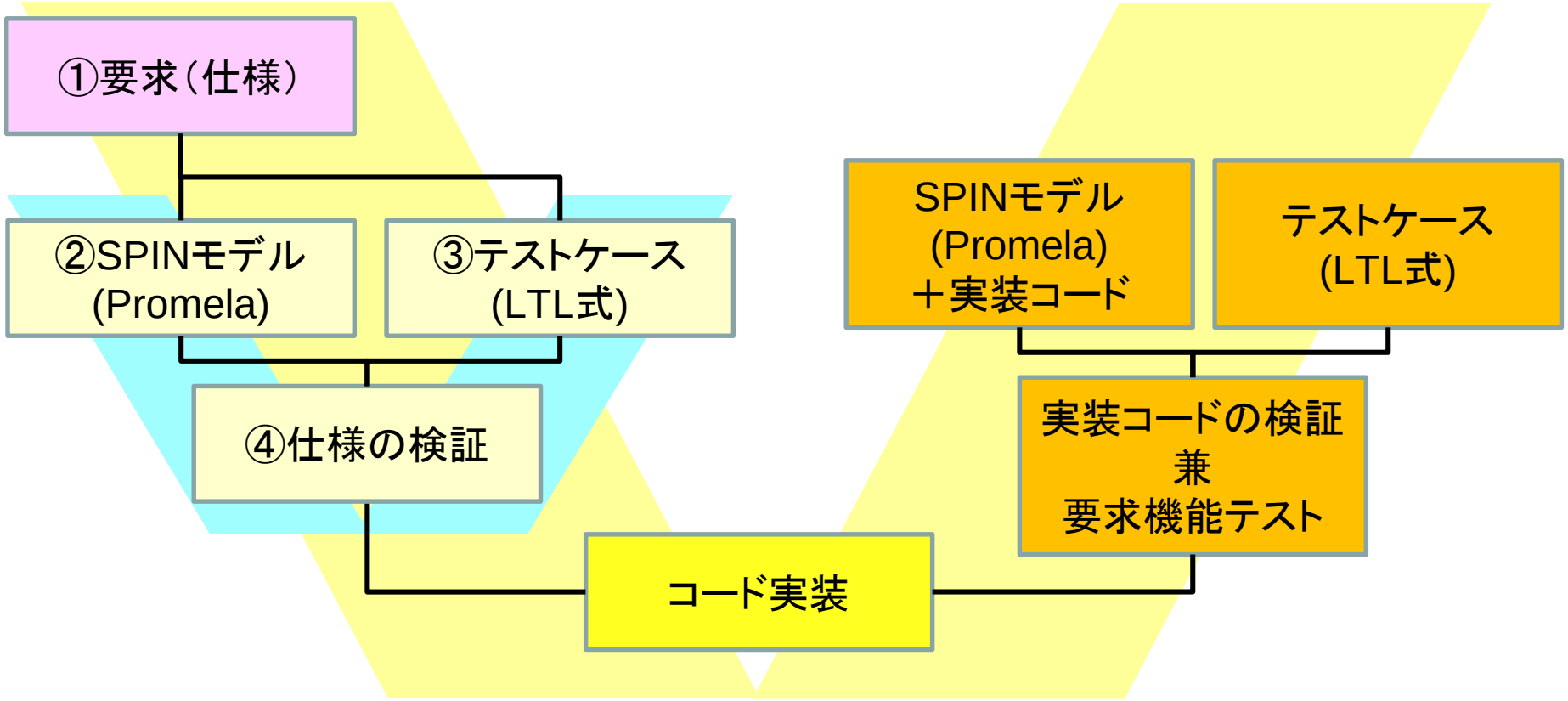
出展：SPINモデリング検査　中島　震著

仕様検証手順

・SPINを用いた仕様検証手順

①要求（顧客仕様）の受領
②SPINモデル（Promela言語）の作成
③テストケース（LTL式）の作成
④仕様の検証

} ②～④で仕様検証を繰返す



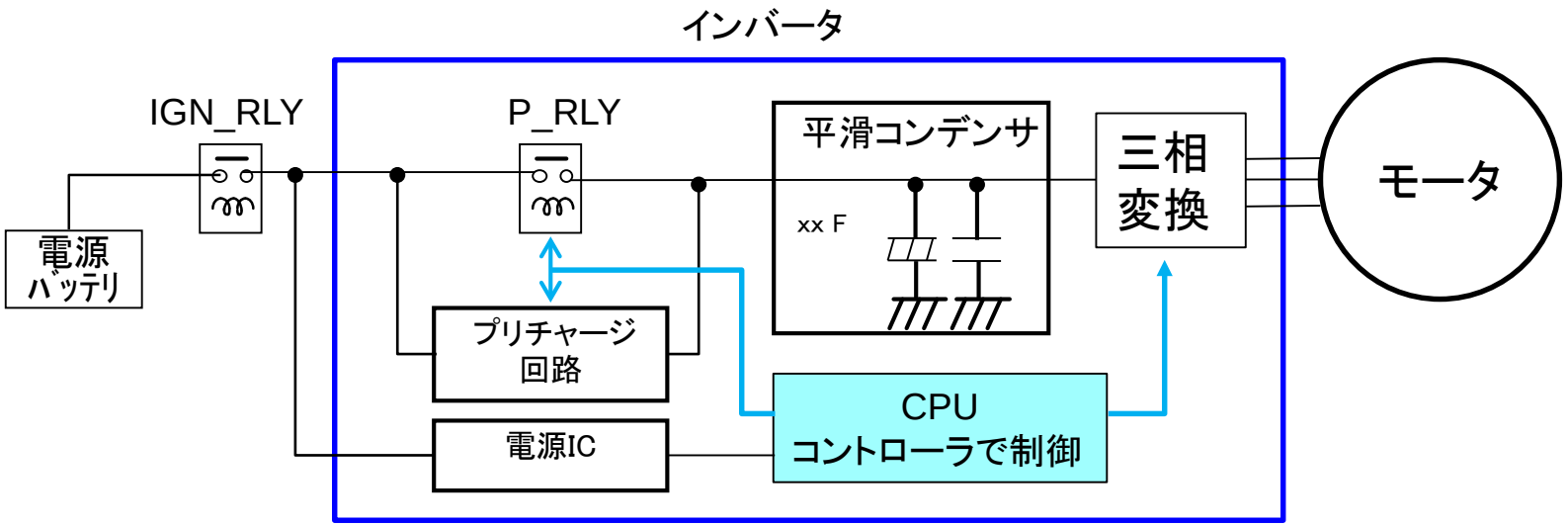
仕様検証例－1

◆モータ制御用インバータのプリチャージ制御

【要求仕様】
- IGN_RLYがOFF→ONのとき、プリチャージ回路を通じてコンデンサへ充電を行う。
- コンデンサ電圧＞入力電圧×0.9のとき、プリチャージ回路をOFFしP_RLYをONする。
- プリチャージ時間＞規定時間のとき、プリチャージ異常と判断する。

【機能要求】
- プリチャージが規定時間以内に完了できること。

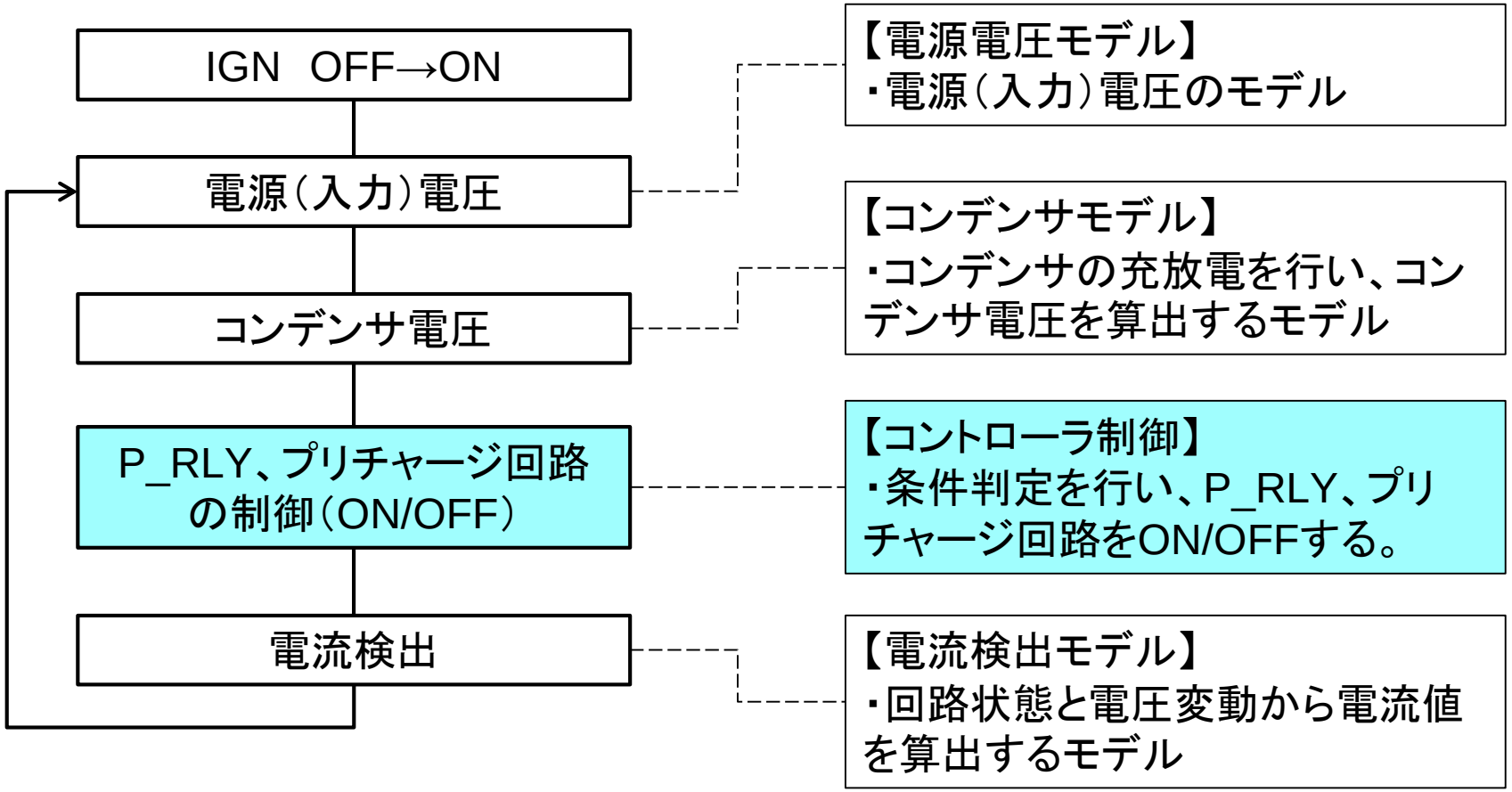
【安全要求】
- IGN_RLYへ流れる突入電流が規定値以下であること。



仕様検証例－2

モデルの動作方式検討
⇒周期的動作

モデル動作内容検討



仕様検証例－3

各モデルの動作は、変更・変化点（変化する/させるモノ）及び安全の観点から振る舞いを検討する。

モデル	変更・変化点	安全観点 （参考モデルでは未対応）
電源（入力）電圧モデル	電源（入力）電圧 （変動）	高電圧 低電圧 センサ故障
コンデンサモデル	コンデンサ電圧 （演算）	コンデンサショート センサ故障
コントローラ制御(P_RLY、プリチャージ回路)	P_RLY、プリチャージ回路 （ON/OFF）	P_RLYのON固着 P_RLYのOFF固着 プリチャージ回路断線
電流検出モデル	電流値 （演算）	突入電流 センサ故障

仕様検証例－4

◆モータ制御用インバータのプリチャージ制御 記述例

Promelaモデル

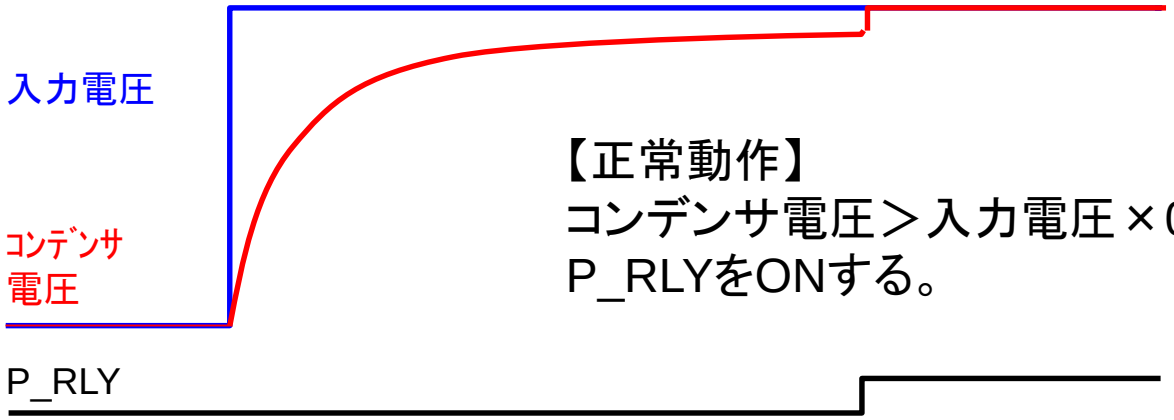
```
mtype = { off, on };
mtype fIgnRly;    /* IGN_RLY */
mtype fPRly = off;    /* P_RLY */
mtype fPRlyold = off;    /* P_RLY前回値 */
mtype fPrechgErr = off; /* プリチャージ異常フラグ */
mtype fCErr = off; /* コンデンサ異常フラグ */

/* コンデンサモデル用 */
int Rpre = 10000; /* プリチャージ回路抵抗[mΩ] */
int Rnrm = 50;    /* 通常回路抵抗[mΩ] */
int C = 10;     /* コンデンサ容量[mF] */
int Vin = 14000;  /* 入力電圧[mV] */
int dt = 1;     /* 単位時間[msec] */
int t = 0;  /* 経過時間[msec] */
int Q = 0;  /* 現在の電荷量[0.001Q] */
int dQ = 0; /* 変化分の電荷量[0.001Q] */
int Vc = 0; /* コンデンサ電圧[mV] */
int VcOld = 0; /* コンデンサ電圧[mV]前回値 */
int i = 0; /* 突入電流[mA] */

active proctype prechg()
{
/*** 初期設計 ***/
  fPrechgErr = off;

  fIgnRly = on;
```

```
/*** ループ ***/

  do
  :: true ->
    /* battery（14V, 13V, 3Vで変動する） */
    atomic{
      if
      ::(Vin == 14000) -> skip
      ::(Vin == 14000) -> Vin = 13000
      ::(Vin == 14000) -> Vin = 3000
      ::(Vin == 13000) -> Vin = 14000
      ::(Vin == 13000) -> skip
      ::(Vin == 13000) -> Vin = 3000
      ::(Vin == 3000) -> Vin = 14000
      ::(Vin == 3000) -> Vin = 13000
      ::(Vin == 3000) -> skip
      fi;
    };
    /* コンデンサ故障(GNDショート) */
    atomic{
      if
      ::(fCErr == off) -> skip
      ::(fCErr == off) -> fCErr = on
      ::(fCErr == on) -> skip
      fi;
    };

  /* コンデンサモデル ---> */
    atomic{
      if
      ::(fCErr == on) -> Vc = 0;
      ::else ->
```

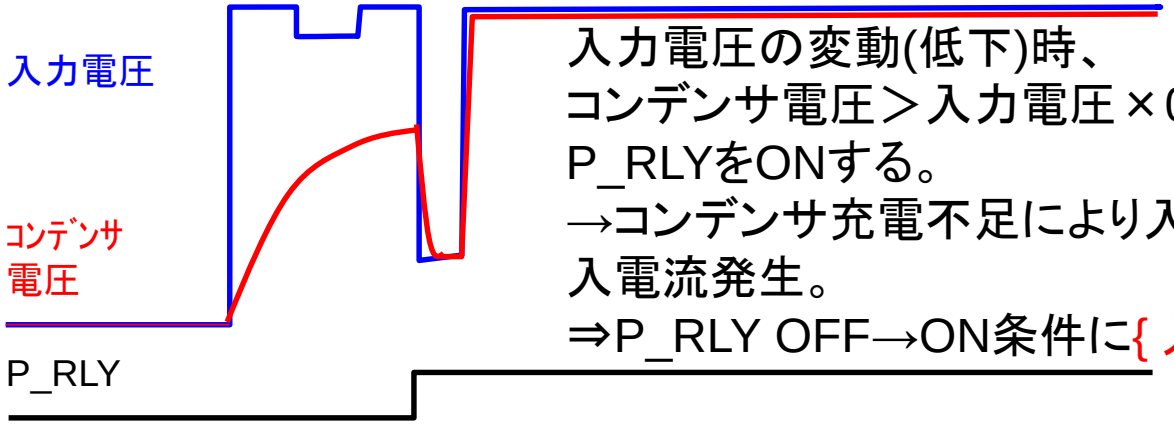
```
/* IGN ONのとき */
        if
        ::(fIgnRly == on) ->
        /* P_RLY OFFのとき */
          if
          ::(fPRly == off) ->
            VcOld = Vc;
            /* プリチャージ処理実施 */
            dQ = (Vin-Vc)*dt*1000/Rpre;
            Q = Q + dQ;
            Vc = Q / C;
          /* P_RLY ONのとき */
          ::(fPRly == on) ->
            i = C * (Vin - Vc)/dt;
             /* 突入電流i＝C＊dv/dt */
            VcOld = Vc;
            if
            ::(fPRlyold == off) -> Vc = Vin; Q = Vc * C
            ::else ->
              dQ = (Vin-Vc)*dt*1000/Rnrm;
              Q = Q + dQ;
              if
              ::((Vin>Vc) && (Vin < Q / C ) ) ->
                 Vc = Vin; Q = Vc * C
              ::((Vin<Vc) && (Vin > Q / C ) ) ->
                 Vc = Vin; Q = Vc * C
              ::else -> Vc = Q / C
              fi;
          fi;
        fi;
```

仕様検証例－5

◆正常動作（上段）と検出された動作（下段）



【正常動作】
コンデンサ電圧＞入力電圧×0.9でプリチャージ完了し、
P_RLYをONする。



入力電圧の変動(低下)時、
コンデンサ電圧＞入力電圧×0.9でプリチャージ完了し、
P_RLYをONする。
→コンデンサ充電不足により入力電圧の変動（上昇）で突入電流発生。
⇒P_RLY OFF→ON条件に{ 入力電圧が正常範囲内 }
の条件追加

仕様検証例-6（検証結果反映）

◆モータ制御用インバータのプリチャージ制御

【制御仕様】

・IGN_RLYがOFF→ONのとき、プリチャージ回路を通じてコンデンサへ充電を行う。

・{コンデンサ電圧＞入力電圧×0.9}**且つ{ 入力電圧が正常範囲内}**のとき、
プリチャージ回路をOFFしP_RLYをONする。

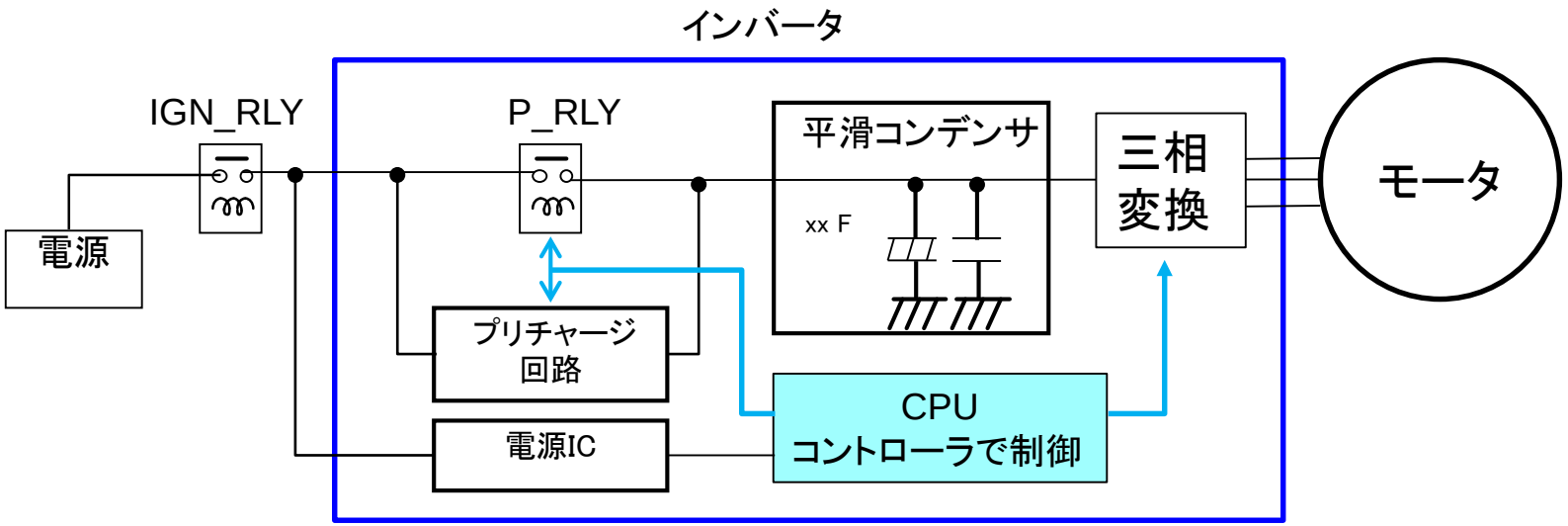
・プリチャージ時間＞規定時間のとき、プリチャージ回路異常と判断する。

【機能要求】

・プリチャージが規定時間以内に完了できること。

【安全要求】

・IGN_RLYへ流れる突入電流が規定値以下であること。



考察

今回の例では電源電圧の変動という、ある意味当たり前の条件が抜けていることを指摘したが、要求仕様作成者が当たり前と考えていることは、得てして要求仕様として記載されていないことが多い。

書かれていないことが伝わらないのは不具合の典型例であるが、仕様検証を用いてシミュレーションを試行錯誤することで、ロジックエラーについても、コンパイルエラーのように気付くことが出来るのではないかと考える。

参考資料

◆Basic Spin Manual
http://spinroot.com/spin/Man/Manual.html

◆Basic Spin Manual翻訳版
http://www.asahi-net.or.jp/~hs7m-kwgc/spin/Man/Manual_japanese.html

◆SPINモデル検査　中島　震著

◆モデル検査[初級編] ―基礎から実践まで4日で学べる―
産業技術総合研究所システム検証研究センター　著

おわりに

- 本活動は、開発者が要求仕様書の品質に悩まされていることから出発しています。しかし、かといって、そのサブタイトルが示すように、要求仕様書は“厳密”でありさえすればよいという訳ではなく、“意図”のようなものにも配慮すべきではないかというものです。
- 活動は「意図」を十分解明したとはいえませんが、活動としては仕様記述の方法を従来の方法も含めて比較検討しました。いくつかの重要な形式手法も調査の対象にしています。また抽象的な議論ばかりではなく、実際の事例をもって調査研究したところにも特色があります。要求開発は「要求獲得⇔分析⇔仕様化⇔妥当性確認」と進む訳ですが、本活動で示された成果は、この要求開発のプロセスを見直すにあたり参考になると思います。
- 「意図」に関していえば、意図は要求には「理由や背景」を書くべきであるとするXDDPなどの見解と重複します。意図を書くことにより、記述した内容がよく伝わったり、記述ミスの指摘があったり、またよりよい提案があったりします。仕様記述の量も減らすことができるのではないでしょうか。これは意図のどういう作用なのだろうというところに関心があります。
- もしかしたら知識には階層的な構造があり、擬人的にいえば、意図を投げかけると「それは自分に関係がある」と知識たちが手を挙げるのかもしれません。知識を用語化し、用語を階層的にうまく組み立てることができれば、限られたドメインの中ではありますが、意図によって仕様の領域を特定できるかもしれません。例えば、ここでドメイン知識は或る製品に関すること(要求仕様）であり、意図はその製品の型番に対応するかもしれません。そして、このようなドメイン知識の実現は「オントロジー」技術に依るのかもしれません。
- 委員会では組合員各位の参加を求めています。いろいろ議論しましょう。

安全性向上委員会委員長 漆原 憲博

執筆者一覧

委員長	漆原憲博(株式会社ジェーエフピー)
SSQ-WGリーダー	中村 洋 (株式会社レンタコーチ)
SSQ-WGサブリーダー	積田 恵一 (東芝システムテクノロジー株式会社)
SSQ-WGサブリーダー	能登 祐二 (日本プロセス株式会社)

要求の仕様化に関する課題、プロセス及び手法

2015/3/20 発行

発行者　一般社団法人 組込みシステム技術協会
東京都中央区日本橋浜町1丁目8-1
TEL: 03(5821)7973 FAX: 03(5821)0444
URL: http://www.jasa.or.jp