



**E T ロボコン2015南関東地区大会
モデリングスペシャルセミナー**

**テーマ：コンピュータにモデルは描けるか？
～モデルの自動生成技術と必要性～**

パネルディスカッション

Takanari HASHIMOTO
2015年6月20日

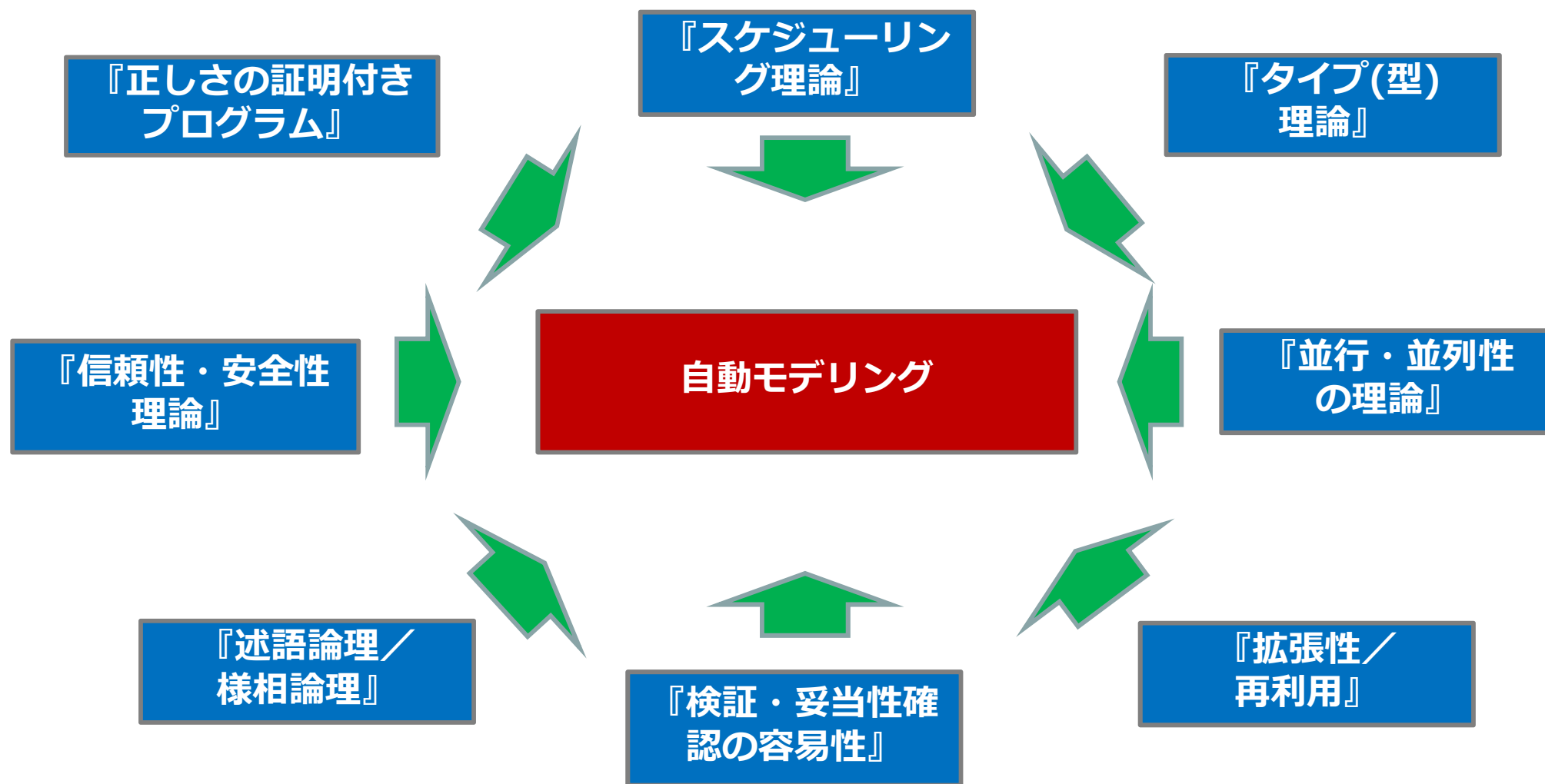
HASHIMOTO SOFTWARE CONSLUTING INTERNATIONAL INC.



TODOからみた 自動モデリング

自動モデリングへの展望～科学的SW開発

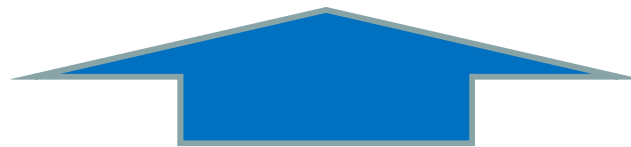
『自動モデリング』で考慮しなければならないテーマ



自動モデリングで実現しなければならないこと

実務から見て自動モデリングで保証すべきこと

- 信頼性の高いモデルであること
- インспекションやテスト工数の削減できること
- 高品質&高生産性の担保できること
- コード生成の自動化&テスト・シミュレーションが可能であること
- 客観的なデーターによる評価が可能であること
- 安全な再利用が可能であること
 - タイプ(型)安全であること
 - タイプ指向モデリングであること
 - リスコフ置換原理が保証されていること
 - モジュラー推論が可能であること



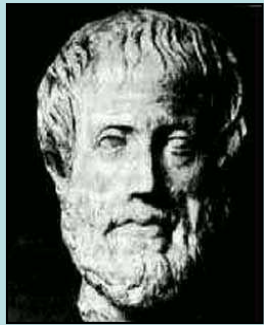
- 上記が達成できて初めて『自動的モデリング』と呼べる
- 決して単純では無く極めて困難と言える



科学的モデリングによる自動化 解説編

HASHIMOTO SOFTWARE CONSLUTING INTERNATIONAL INC.

科学的モデル／モデリングは数学・論理学



紀元前

...



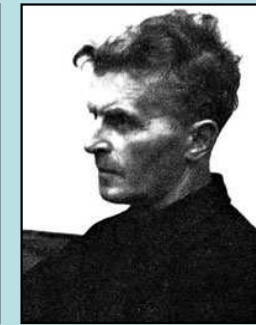
17世紀



...



20世紀

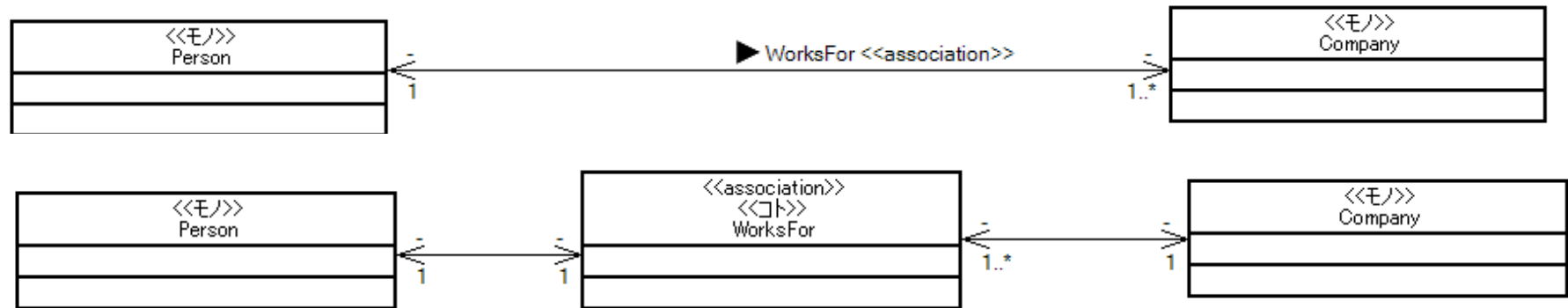


...

21世紀

モデリング技術は哲学・論理学・数学を基礎に持つ

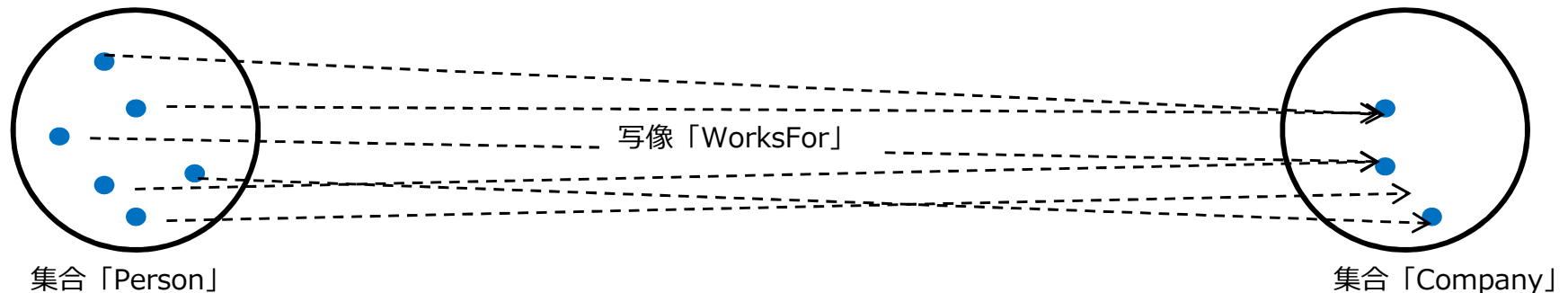
UML



命題関数

WorksFor(Person,Company)

集合
&
写像



分析・設計・実装の正当性(correctness)

プログラムの正当性(correctness)

- プログラムの正当性 :
 - **Hoareトリプル** : $\{P\} C \{Q\}$
 - インスペクションやテストをしなくても「**正しさ(correctness)**」を保証
 - **演繹的アプローチ**

正当性(correctness)の設計技法

モデルの正当性(correctness)の技法

- **Hoare**の理論は分析・設計の技法として利用されている
 - Design By Contract™
 - 不変条件(invariant)
 - 事前条件(pre-condition)
 - 事後条件(post-condition)
 - タイプ(型)置換原理

継承による『開放閉鎖原理』の実現

『開放閉鎖原理(OCP:Open-Closed Principle)』 By Bertrand Meyer

- 『開放閉鎖原理』を可能とするのは**継承機能だけ**である

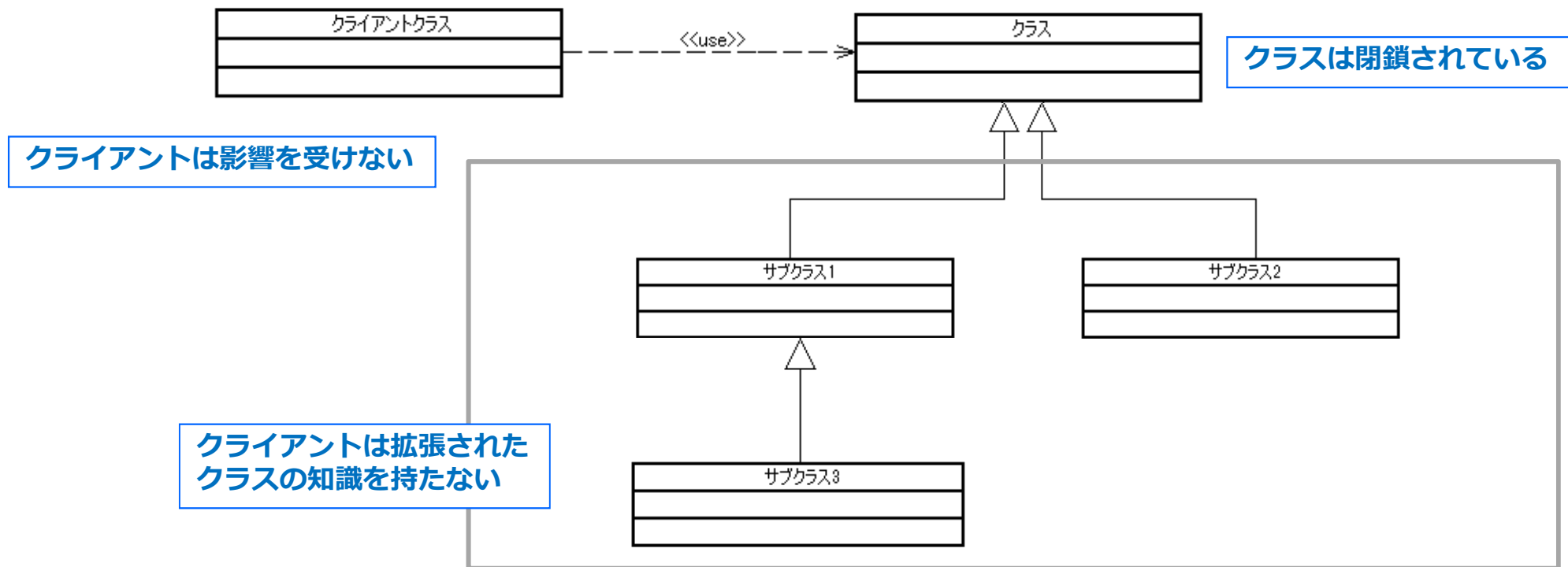
『開放閉鎖原理(OCP:Open-Closed Principle)』

- 『正しい設計には,継承を正確に使用する事が極めて重要である』 by Ian Joyner
- 『OO開発の威力の多くは継承の重要な概念によるものである』 by Herbert Toth

『開放閉鎖原則(Open-Close Principle)』

開放閉鎖原則(Open-Close Principle)[Meyer86]

- クラス／コンポーネント／モジュールが：
 - 拡張可能**で**クラス／コンポーネント／モジュール**は**開放**されていること
 - クライアントに対して**クラス／コンポーネント／モジュール**は**閉鎖**されていること



『開放閉鎖原理』のメリット

『開放閉鎖原理(OCP:Open-Closed Principle)』 By Robert C.Martin

- 『開放閉鎖原理』を適用すれば、すでにモジュールがコンパイルされてバイナリ形式になっているものは、それがリンクされたライブラリーであろうと,DLLであろうと,Javaの.jarファイルであろうと手を触れる必要がない
- 『開放閉鎖原理』は**オブジェクト指向設計の核心**である
- 『開放閉鎖原理』に従う事でオブジェクト指向技術から得られる最大の利益を享受できる
 - 柔軟性
 - 再利用性
 - 保守性

「クラス」と「タイプ(型)」の区別

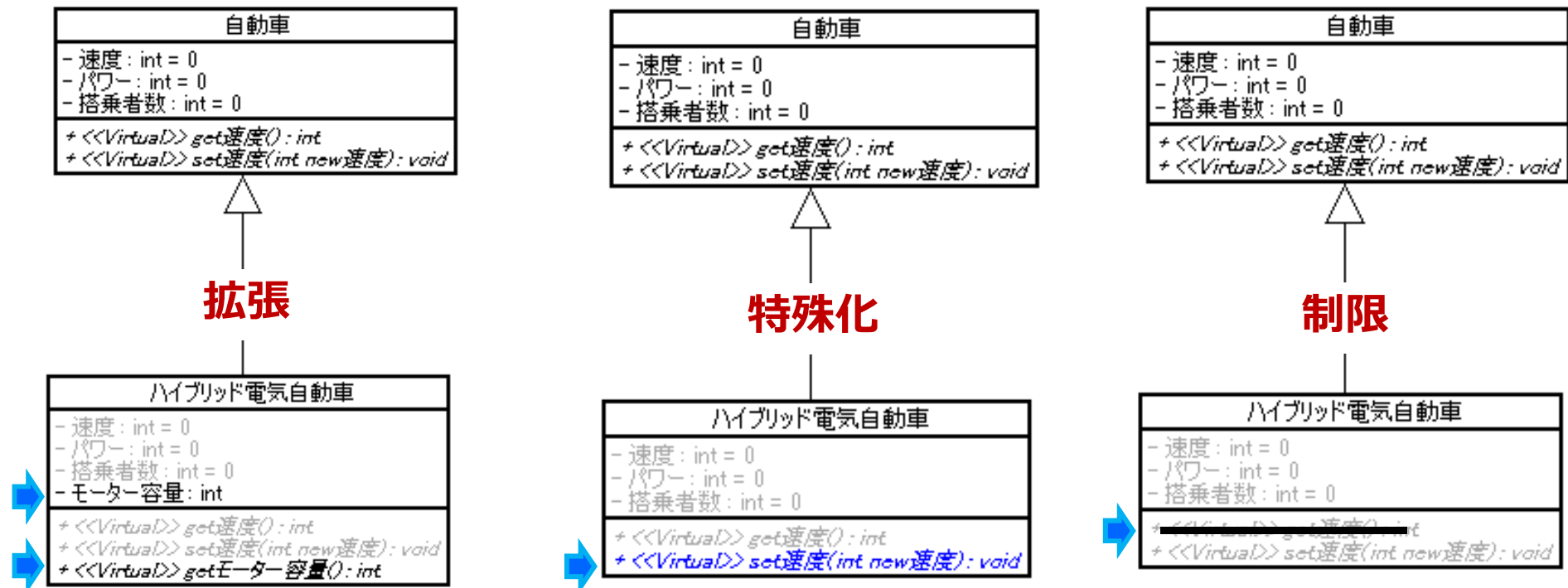
クラスとタイプ(型)の区別とタイプ(型)置換原理

- サブクラス(subclass)とサブタイプ(subtype)は明確に異なる概念である
- 継承階層内でサブクラスは任意のスーパークラス『タイプ(型)置換原理』を満たす必要がある
→ 「振る舞いサブタイプ(型)」
- 「振る舞いサブタイプ(型)」
→ サブクラスのオブジェクトをスーパークラスのオブジェクトの意味(振る舞い)を保証して代わりに利用できる「部分型(サブタイプ)」

タイプ継承の分類～拡張／制限／特殊化

タイプ継承によりサブタイプは下記のケースが発生する[Bla91]

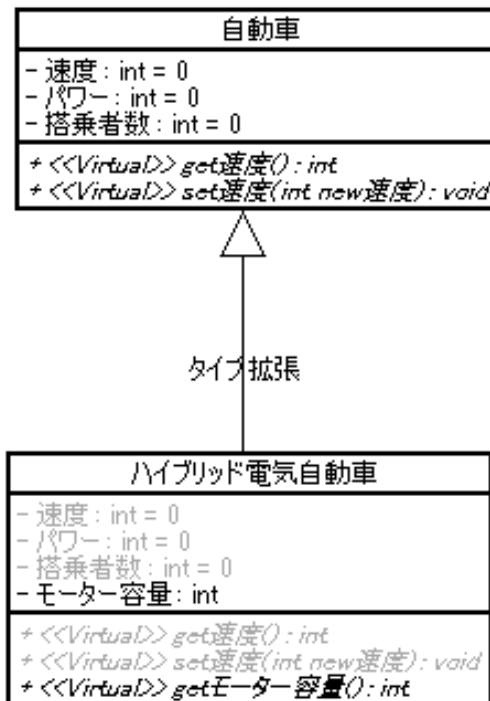
- **タイプ拡張(extension)**
 - サブクラスは新しい**属性・操作を追加**すること
- **タイプ特殊化(specialization)/再定義(override, redefine)**
 - サブクラスで最適化するために操作を**再定義して特殊化**すること
- **タイプ制限(restriction)**
 - スーパークラスの属性や操作の一部を**継承**すること



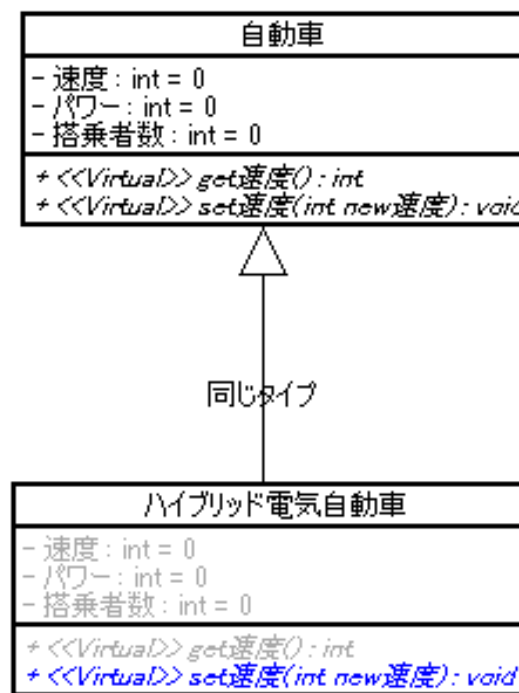
タイプ(型)継承とタイプ(型)関係

タイプ継承の分類	クラスから見た関係	タイプ(型)から見た関係
拡張	スーパークラス-サブクラス関係	スーパータイプ-サブタイプ関係
特殊化/再定義	スーパークラス-サブクラス関係	同じタイプ同士関係
制限(制約)	スーパークラス-サブクラス関係	サブタイプ-スーパータイプ関係

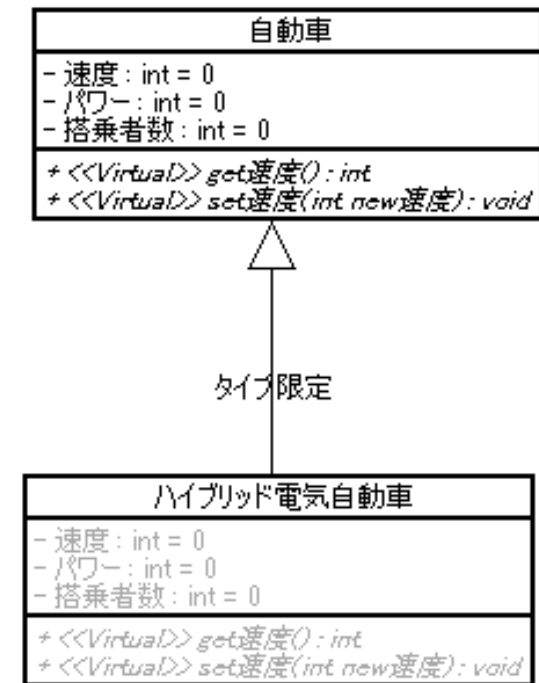
<スーパータイプ-サブタイプ>関係



<同じタイプ同士>関係



<サブタイプ-スーパータイプ>関係

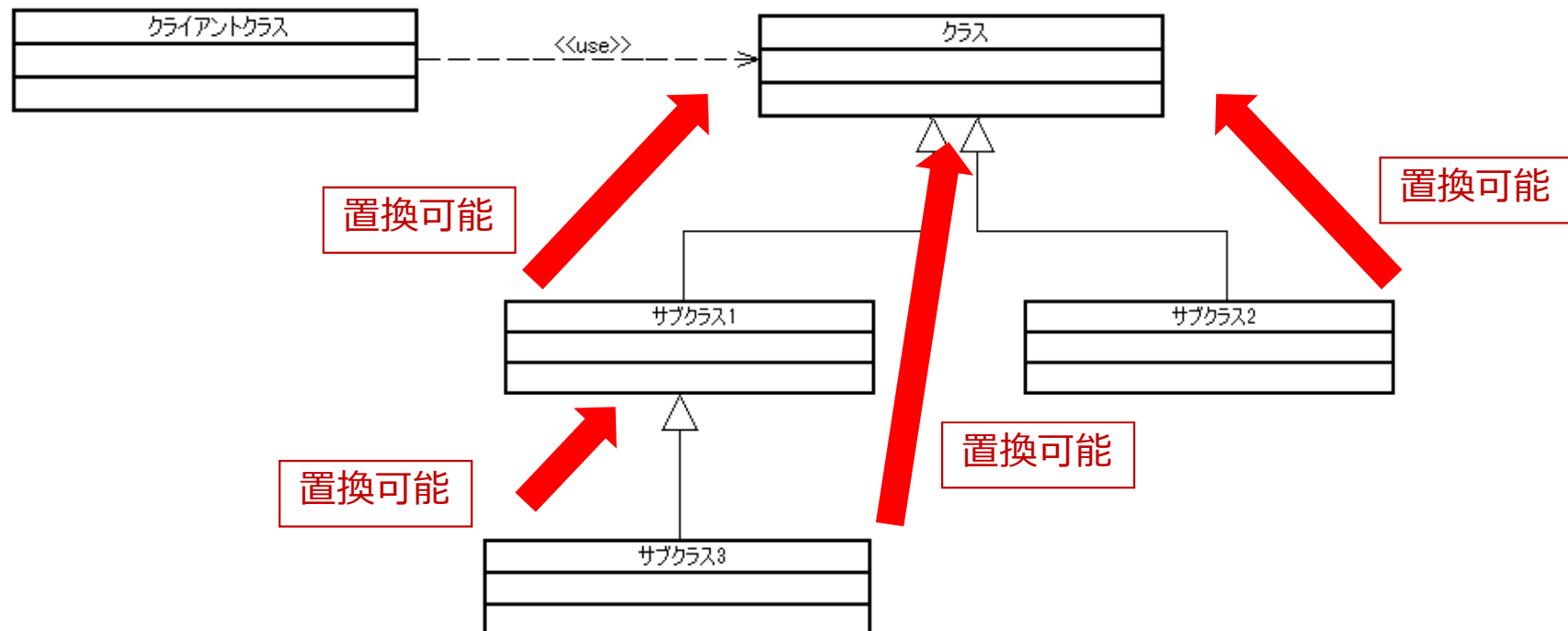


タイプ(型)置換原理(type substitution principle/type conformance principle)

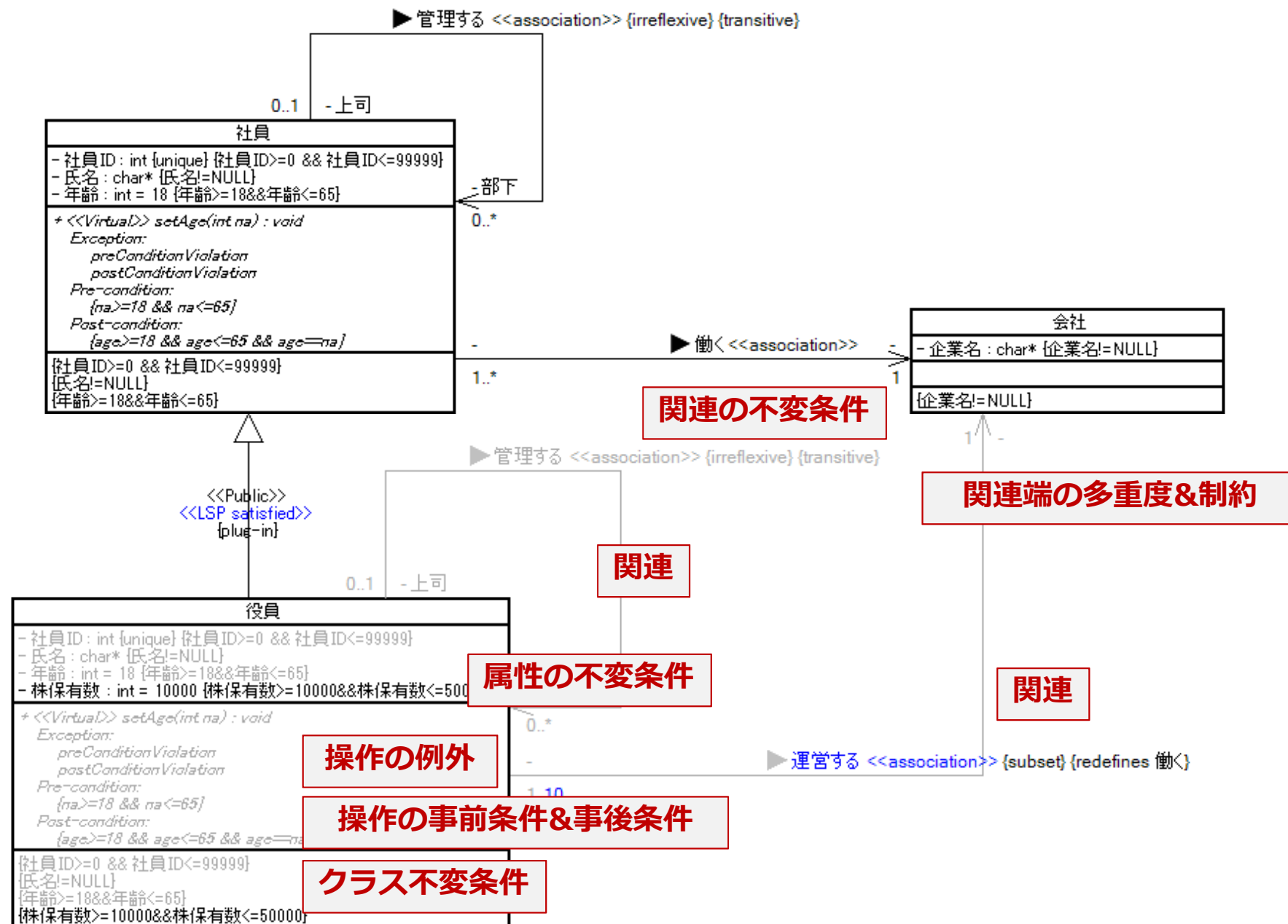
最重要基礎事項～タイプ(型)とクラスの区別とタイプ(型)置換原理

- **サブクラス(subclass)**と**サブタイプ(subtype)**は明確に異なる概念である
- 継承階層内でサブクラスは任意のスーパークラスと**タイプ(型)置換原理**を満たす必要がある

サブタイプのオブジェクトをスーパータイプのオブジェクトとして「意味を変えずに代わりに利用できる」こと



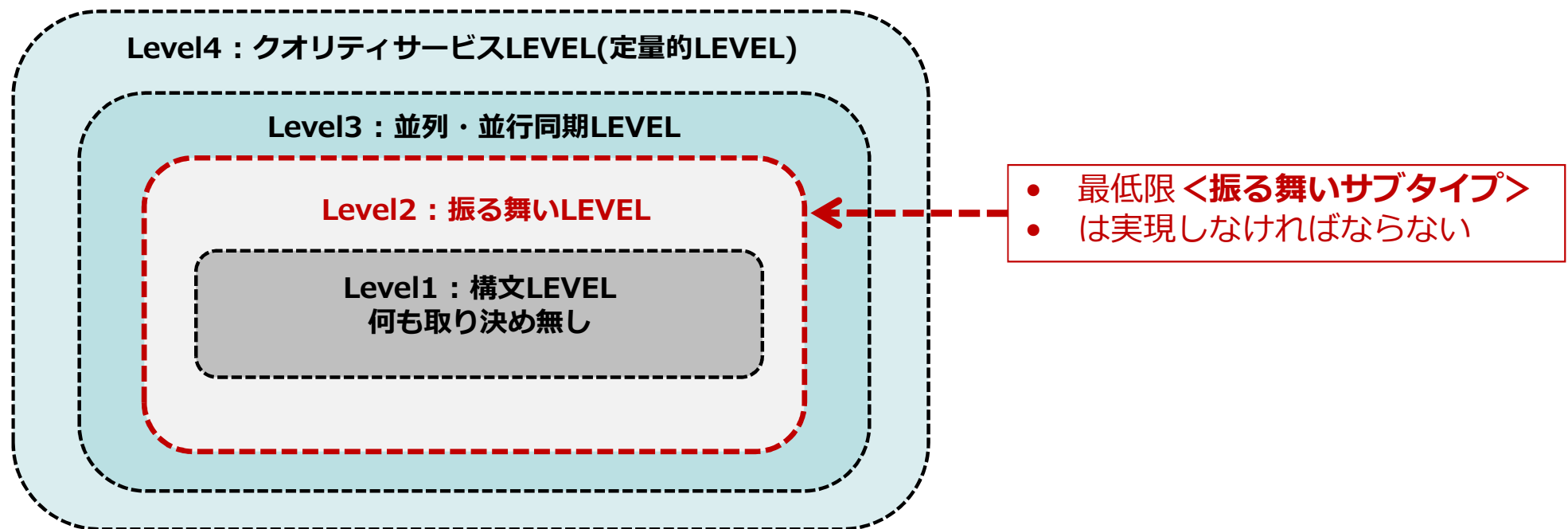
継承は全プロパティを継承する



タイプ(型)置換の難易度Level

タイプ(型)置換のLevelがある[Beugnard1999]

- Level1 : **構文的置換(syntactic proof)** : 構文
- Level2 : **振る舞い置換(behavioral proof)** : 意味
- Level3 : **処理順序置換(sequencing proof)** : 並列・並行性
- Level4 : **サービス品質置換(quality of service proof)** : 処理時間, サイズ空間, 安全性, etc



有名な『タイプ置換原理』の1つ～『リスコフ置換原理』

リスコフ置換原理[Liskov et al.,1994]の意味

- スーパータイプの**不変条件(invariant)**はサブタイプにおいて<強める>ことはできるが<弱める>ことはできない
 - **属性不変条件**
 - **クラス不変条件**
 - **SCinvariant $\Rightarrow$ Invariant**
- スーパータイプの**操作の事前条件(pre-condition)**はサブタイプにおいて<弱める>ことはできるが<強める>ことはできない
 - **Cpre $\Rightarrow$ SCpre**
- スーパータイプの**操作の事後条件(post-condition)**はサブタイプにおいて<強める>ことはできるが<弱める>ことはできない
 - **SCpost $\Rightarrow$ Cpost**
 - **(Cpre $\wedge$ SCpost) $\Rightarrow$ Cpost**

- サブクラスが『**振る舞いサブタイプ(型)**』となる規則
- クラスやコンポーネントの「**派生開発**」と「**再利用**」を**安全**に行える

リスク置換原理と論理空間(状態空間)

リスク置換原理

- スーパータイプの**不変条件(invariant)**はサブタイプにおいて **<強める>** ことはできるが **<弱める>** ことはできない
- スーパータイプの**事前条件(pre-condition)**はサブタイプにおいて <弱める> ことはできるが <強める> ことはできない
- スーパータイプの**事後条件(post-condition)**はサブタイプにおいて <強める> ことはできるが <弱める> ことはできない



スーパータイプとサブタイプの論理空間(状態空間)の空間の関係

- 同じ次元サブタイプはスーパータイプの「**論理空間(状態空間)**」に **<完全一致する>** or **<含まれる>** 必要がある
- サブタイプで「**拡張**」した部分は**置換原理の影響を受けない**
 - スーパーの「不変条件」に対して「**直交**」しなければならない



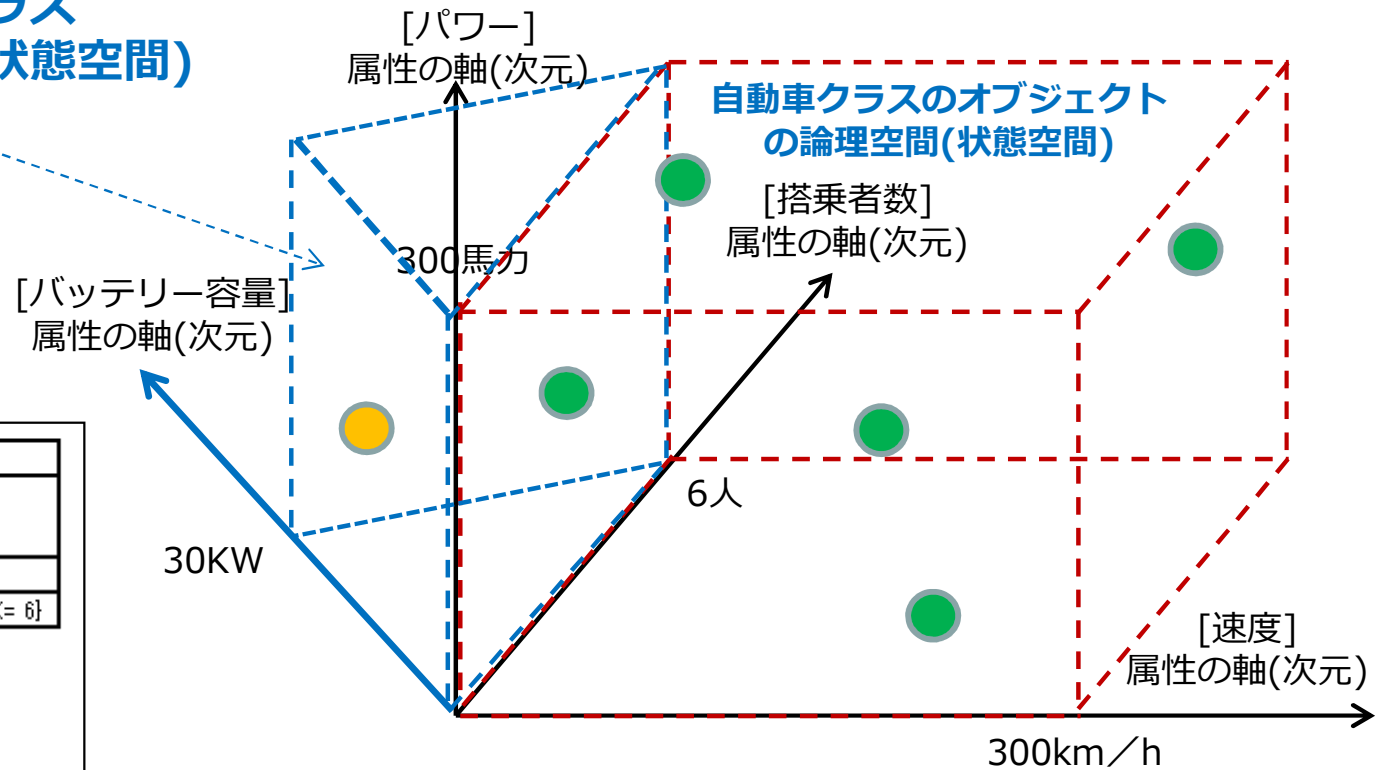
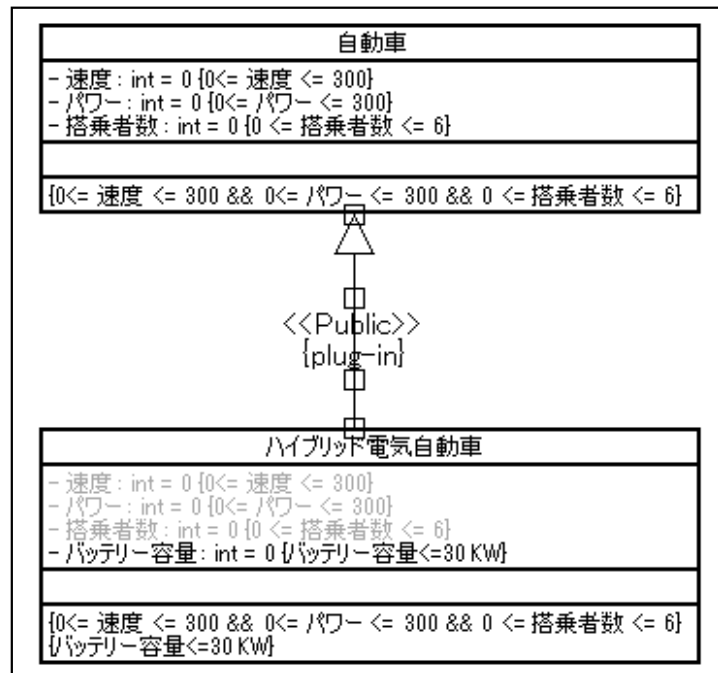
Design By Contract™(契約による設計)

- サブタイプの**属性の不変条件／クラス不変条件**は,スーパータイプより **<強くする>** ことは可能

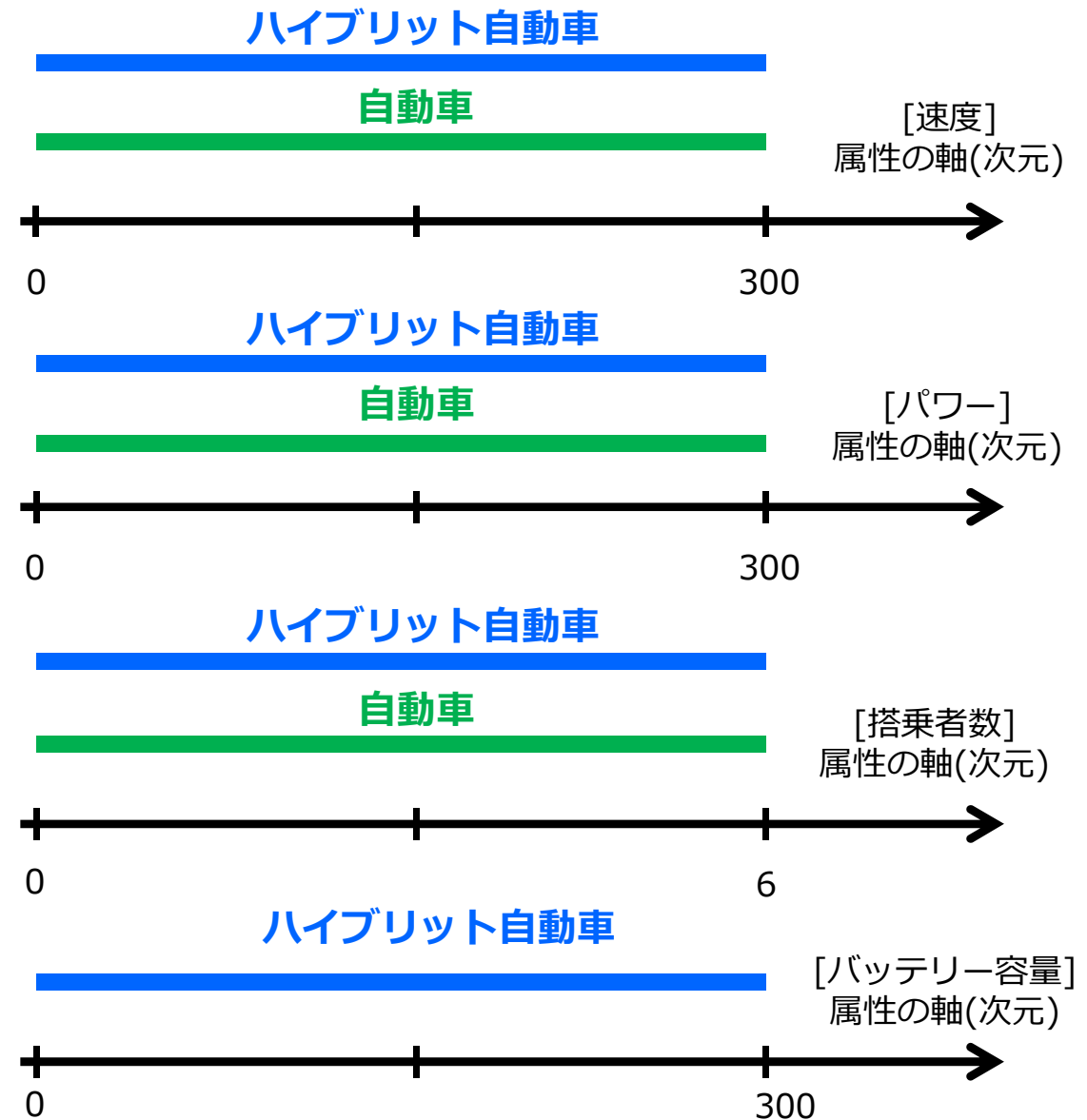
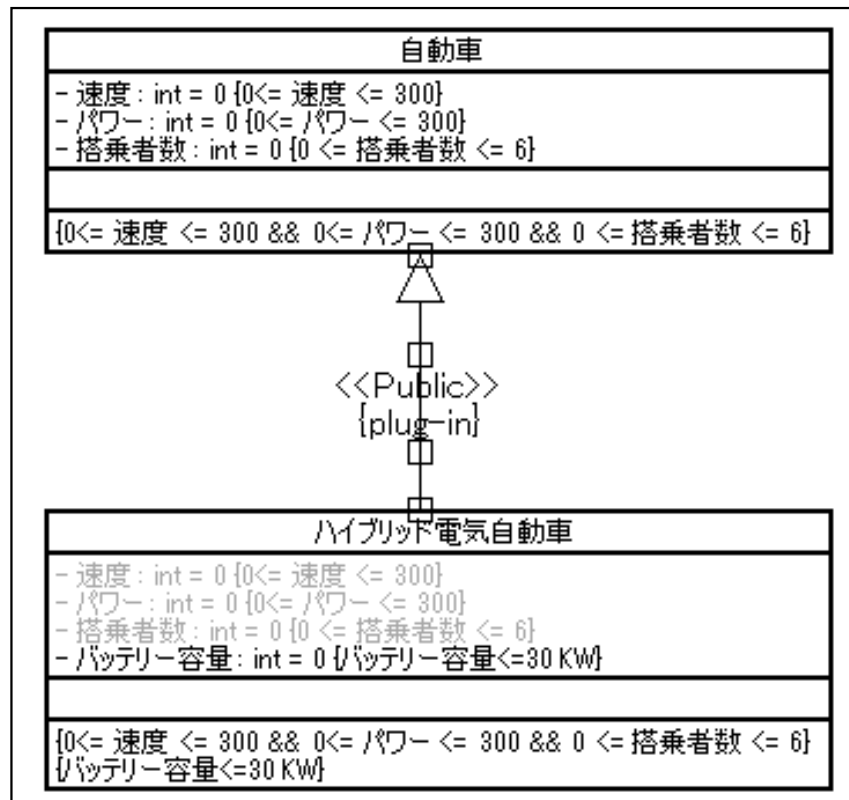
図解：Design By Contract™と振る舞いタイプ(型)置換①

ハイブリッド自動車クラス のオブジェクトの論理空間(状態空間)

タイプ(型)拡張
(サブタイプ電気自動車の
固有の属性)



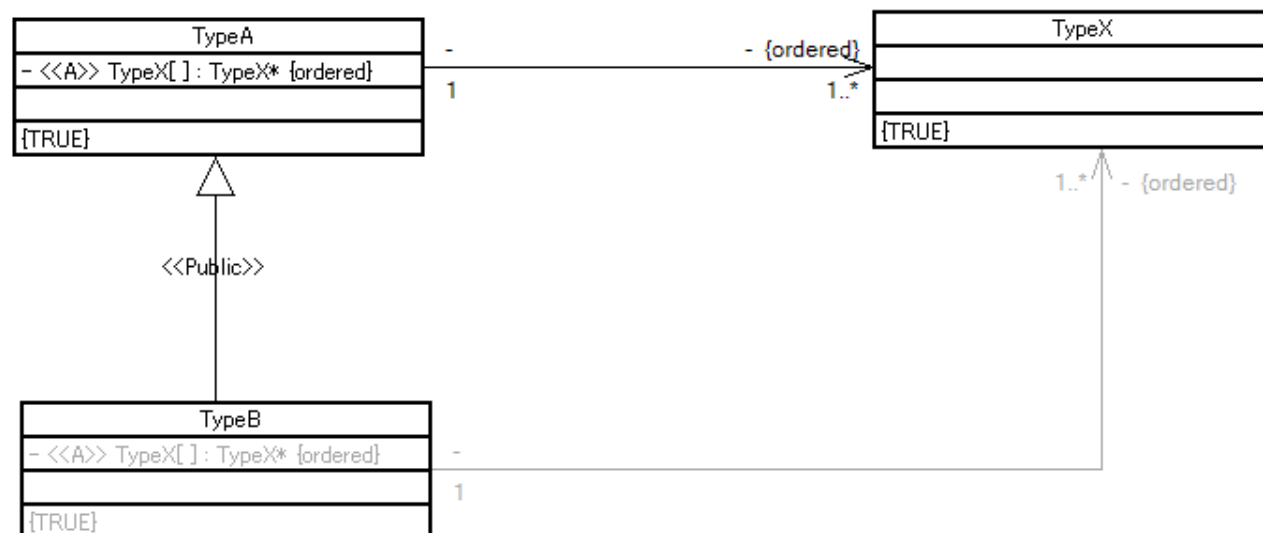
図解：Design By Contract™と振る舞いタイプ(型)置換②



関連・コンテナクラスの置換可能性

関連・コンテナクラスの置換可能性

- 2つのタイプType1とType2が同じタイプであるとき、Type1はType2に適合する
- Type1がType2のサブタイプであるときType1はType2に適合する
- Type1がType2に適合した場合に、コレクション(Type1)は、コレクション(Type2)に適合する
 - Set(Type1)とSet(Type2)
 - OrderedSet(Type1)とOrderedSet(Type2)
 - Sequence(Type1)とSequence(Type2)
 - Bag(Type1)とBag(Type2)
- タイプ適合は推移的である
- Bag(T), Sequence(T), Set(T)及びOrderedSet(T)はCollection(T)のサブタイプなので：
 - Bag(T), Sequence(T), Set(T)及びOrderedSet(T)間に他の関係は存在しない
- Set(T)はOrderedSet(T) 適合しない
- Bag(T)はSet(T)に適合しない
- Sequence(T)はBag(T)に適合しない
- Set(T)はBag(T)に適合する
- OrderedSet(T)はSet(T)とSequence(T)に適合する



再帰関連の制約の置換可能性

条件	定義
Reflexive	$(x,x) \in R$ for every $x \in S$
Non-reflexive	$(x,x) \notin R$ for every $x \in S$
Symmetric	If $(x,y) \in R$ then $(y,x) \in R$ for every $x,y \in S$
Non-symmetric	If $(x,y) \in R$ then $(y,x) \notin R$ for every $x,y \in S$
Transitive	If $(x,y) \in R$ and $(y,z) \in R$, then $(x,z) \in R$ for every $x,y,z \in S$
Non-transitive	If $(x,y) \in R$ and $(y,z) \in R$, then $(x,z) \notin R$ for every $x,y,z \in S$

- 『反射性』が「必要(required)」時は、 xRx が[TRUE]なる任意の x は受け入れられるべきである
- 『反射性』が「除外(excluded)」時は、 xRx が[TRUE]なる任意の x は拒否されるべきである
- 『反射性』が「無関係(irrelevant)」時は、いくつかの x は xRx について[TRUE]であり、いくつかの x は xRx において[FALSE]である様にすべきである
- 『対称性』が「必要(required)」時は、 $(xRy$ と $yRx)$ が[TRUE]となる任意の x,y を受け入れられるべきである
- 『対称性』が「除外(excluded)」時は、 $(xRy$ と $yRx)$ が[TRUE]となる任意の x,y を拒否しなければならない
- 『対称性』が「無関係(irrelevant)」時は、 $(xRy$ と $yRx)$ が[TRUE]となるいくつかの x と y は受け入れられるべきである、また $(xRy$ と $yRx)$ を満たさないいくつかの x と y は拒否されるべきである
- 『推移性』が「必要(required)」時は、 $(xRy$ と yRz と $xRz)$ が[TRUE]となる任意の x,y および z は、受け入れられるべきである
- 『推移性』が「除外(excluded)」時は、 $(xRy$ と yRz と $xRz)$ が[TRUE]となる任意の x,y および z は拒絶されるべきである
- 『推移性』が「無関係(irrelevant)」時は、 $(xRy$ と yRz と $xRz)$ が[TRUE]となるいくつかの x,y および z は、受け入れられるべきである、また、 $(xRy$ と yRz と $xRz)$ を満たさないいくつかの x,y および z は拒絶されるべきである

契約に基づく設計(Design By Contract)

クラス不変条件・事前条件・事後条件は**タイプ(型)**の一部である

契約に基づく設計(Design By Contract)	
クラス不変条件 (class invariant)	- クラス全体の不変条件 - クラスのインスタンスが満たさないといけない命題を表現している
属性の不変条件 (invariant)	クラスの属性が常に満たさなければならない条件
事前条件 (pre-condition)	- 操作を実行する前に成立しなければならない条件 (操作が呼び出された直後に判定される) - 事前条件を成立させる責任は操作を呼び出すクライアント側となる
事後条件 (post-condition)	- 操作が終了時に成立しなければならない条件(操作の最後で判定される) - 事後条件を成立させる責任は操作側となる

属性の不変条件(invariant)

属性の不変条件(invariant)

- 属性の制約

自動車
- 速度: int = 0 {0 ≤ 速度 ≤ 300} - パワー: int = 0 {0 ≤ パワー ≤ 300} - 搭乗者数: int = 0 {0 ≤ 搭乗者数 ≤ 6}
<pre>+ <<Virtual>> get速度(): int Pre-condition: {TRUE} Post-condition: {TRUE} + <<Virtual>> set速度(int new速度): void Pre-condition: {0 ≤ new速度 ≤ 300} Post-condition: {0 ≤ 速度 ≤ 300 && 速度 = new速度}</pre>
{0 ≤ 速度 ≤ 300 && 0 ≤ パワー ≤ 300 && 0 ≤ 搭乗者数 ≤ 6}

クラス不変条件(class invariant)

クラス不変条件(class invariant)

- 属性の不変条件の総和
- 属性間の不変条件
- クラス不変条件が定義されない場合：
 - この時の条件は{**true**}

自動車

- 速度: int = 0 {0 ≤ 速度 ≤ 300}
- パワー: int = 0 {0 ≤ パワー ≤ 300}
- 搭乗者数: int = 0 {0 ≤ 搭乗者数 ≤ 6}

+ <<Virtual>> get速度(): int

Pre-condition:

{TRUE}

Post-condition:

{TRUE}

+ <<Virtual>> set速度(int new速度): void

Pre-condition:

{0 ≤ new速度 ≤ 300}

Post-condition:

{0 ≤ 速度 ≤ 300 && 速度 = new速度}

{0 ≤ 速度 ≤ 300 && 0 ≤ パワー ≤ 300 && 0 ≤ 搭乗者数 ≤ 6}

事前条件(pre-condition)と事後条件(post-condition)①

事前条件(pre-condition)と事後条件(post-condition)

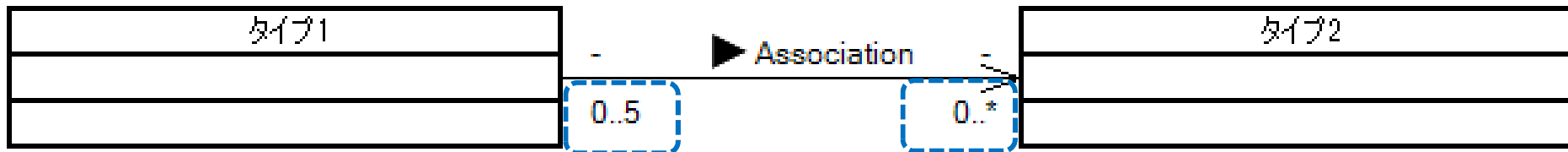
- 各操作に**事前条件(pre-condition)**と**事後条件(post-condition)**を定義
- 事前条件** :
 - 操作が意味をもつ条件を記載する
 - 開始条件
- 事後条件** :
 - 操作が保証する処理を記載する
- 事前条件(pre-condition)と事後条件(post-condition)が定義されない場合 :
 - この時の条件は{**TRUE**}

自動車
- 速度 : int = 0 {0 ≤ 速度 ≤ 300} - パワ- : int = 0 {0 ≤ パワ- ≤ 300} - 搭乗者数 : int = 0 {0 ≤ 搭乗者数 ≤ 6}
+ <<(int)>>get速度() : int Pre-condition: {TRUE} Post-condition: {TRUE}
+ <<(int)>>set速度(int new速度) : void Pre-condition: {0 ≤ new速度 ≤ 300} Post-condition: {0 ≤ 速度 ≤ 300 && 速度 = new速度}
{0 ≤ 速度 ≤ 300 && 0 ≤ パワ- ≤ 300 && 0 ≤ 搭乗者数 ≤ 6}

事前条件(pre-condition)と事後条件(post-condition)②

事前条件(pre-condition)と事後条件(post-condition)

- 関連は**定義域**と**値域**が存在する：
 - 数学の写像／関数
 - 論理学の述語
- **多重度の[0]**があることは：
 - **部分関数(partial function)**を意味する
 - 事前条件を持つ事を意味する
- 事前条件は操作の**特性方程式(characteristic equation)**
 - その操作の**定義域(domain)**を決定する



目の前にある危機～間違った「派生開発」と「再利用」

“非”科学的な再利用や派生開発のアプローチは悲惨な結果招く

- ソフトウェアの**再利用が原因の大参事**としてAriane5がある
- 事故原因調査結果[Lio96]:
 - スローされた例外をキャッチできない不具合がAriane4のソフトウェアコンポーネントあった
 - このコンポーネントの再利用が原因で宇宙船のソフトウェアが破壊された
- JzquelとMayerは、**事故の根本原因はコンポーネントの「契約」が存在しない事**と主張[JM97]
 - **「契約」**とはソフトウェア科学を利用した操作・クラス・コンポーネントの**「明確な意味の定義」**のことであり下記の「特性」で定義する:
 - **不変条件**
 - **操作の事前条件／事後条件／例外**
- JzquelとMayerは以下のように結論付けている:
 - クラスやコンポーネントの「契約」を定義せず「タイプ(型)置換原理」を満足しない「派生開発」と「再利用」は**完全な愚行**である



日本は現在も『**間違った危険な開発**』が主流であり横行している

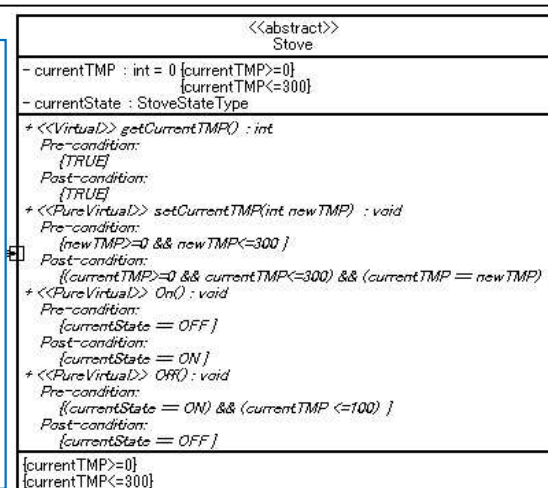


科学的モデリングによる自動化 デモ編

HASHIMOTO SOFTWARE CONSLUTING INTERNATIONAL INC.

『科学的モデリングツールHOPE』による作業の自動化

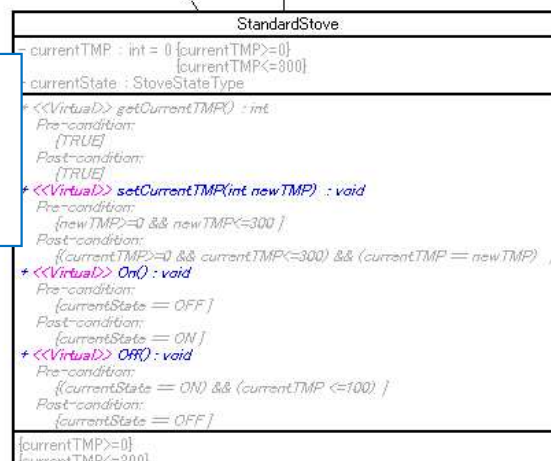
- 『科学的SW-Plug&Play』
『科学的SW-バリエーション・デベロップメント』を支援する
- 正確なモデルを迅速&自動的に作成・検証機能をもつ
 - 「振る舞いサブタイプ」自動定義機能
 - 演繹推論によるタイプ(型)整合性の自動判定機能
 - {redefine}{refine}自動判別&色表示機能
 - 自動コード生成
 - 自動ドキュメント生成支援機能
 - 静的関連・動的関連定義機能
 - etc



機能の実現部
・リスコフ置換原則を満たし「振る舞いサブタイプ」にすることが必要がある
・機能修正・拡張はサブクラス側で行う

機能の仕様部
・インターフェースとして機能する
・サブクラスで機能を具象的な実装するが「リスコフ置換原則」を満たす必要

- 作業効率を劇的に向上させるために
ツールによる自動処理を利用する
- ツールの機能でユーザーは難しい
理論・原理に精通していなくてもOK



関連仕様

全般 Room Stove

関連名 use

関連種類 静的関連 属性 <<assoc>> 動的関連 パラメーター <<param>>

プロパティ1 (制約) derived(派生) ☐

プロパティ2 (制約) reflective(反射関係) ☐ no transitive(推移関係) ☐

プロパティ3 (制約) union(和集合) ☐

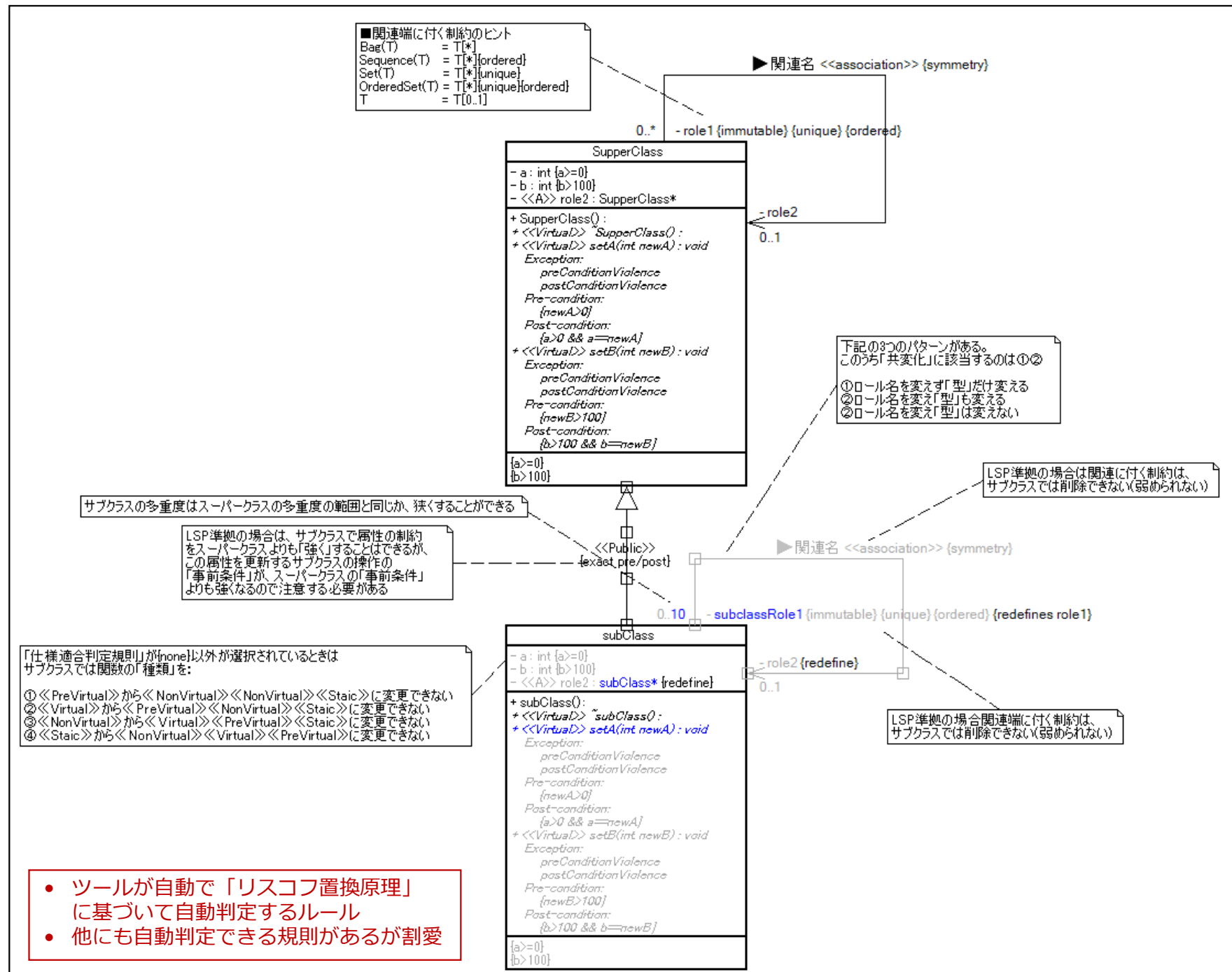
プロパティ4 (制約)

関連名方向 ☐ ☐ ☐

誘導可能性 ☐ ☐ ☐

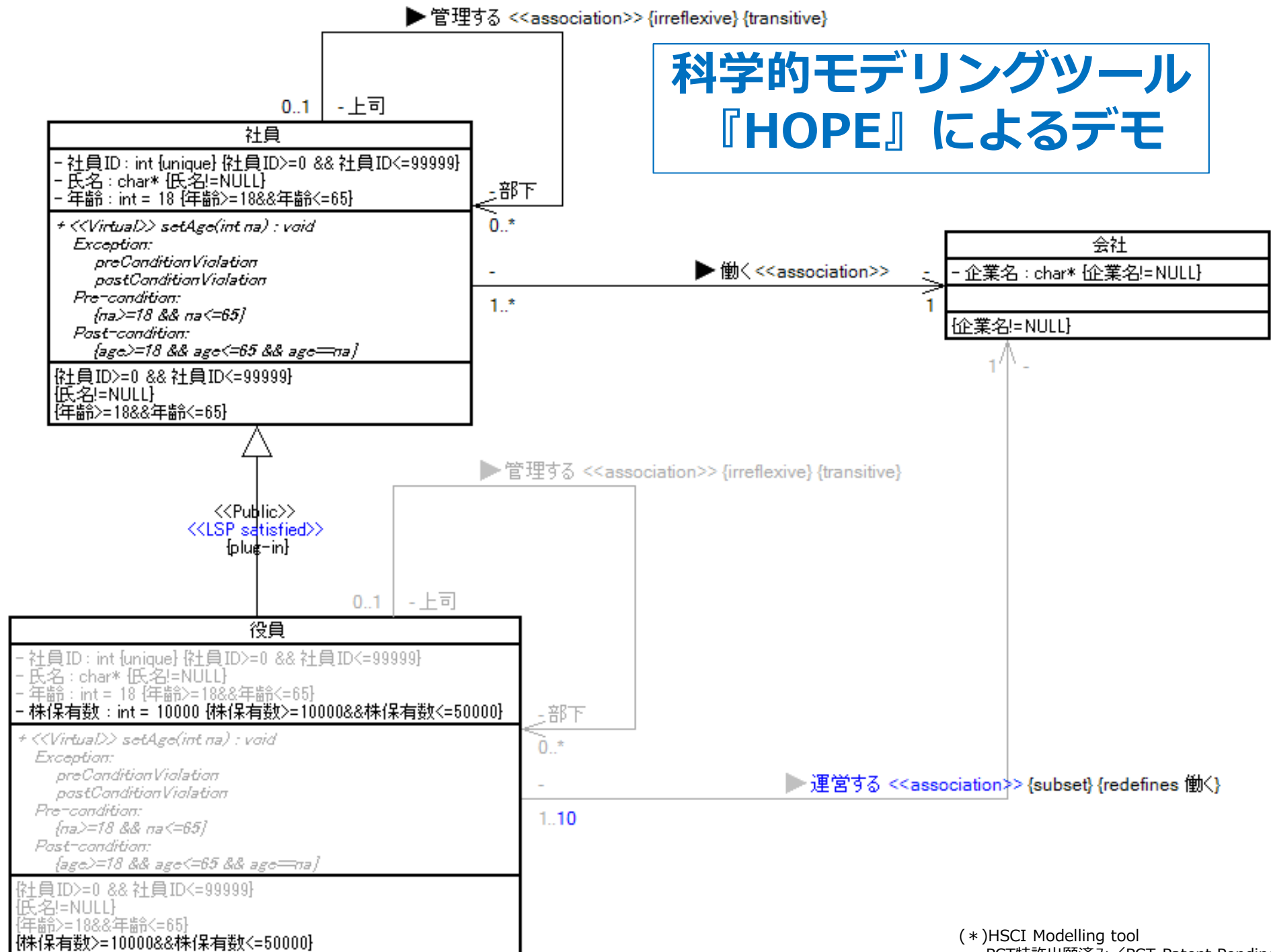
説明

ツールで『振る舞いサブタイプ(型)』の自動判定



『Auto Modeling Pilot Controller』 機能①

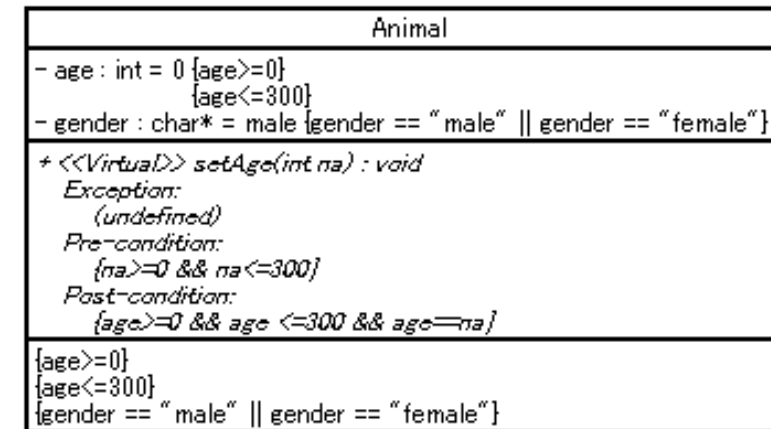
科学的モデリングツール 『HOPE』によるデモ



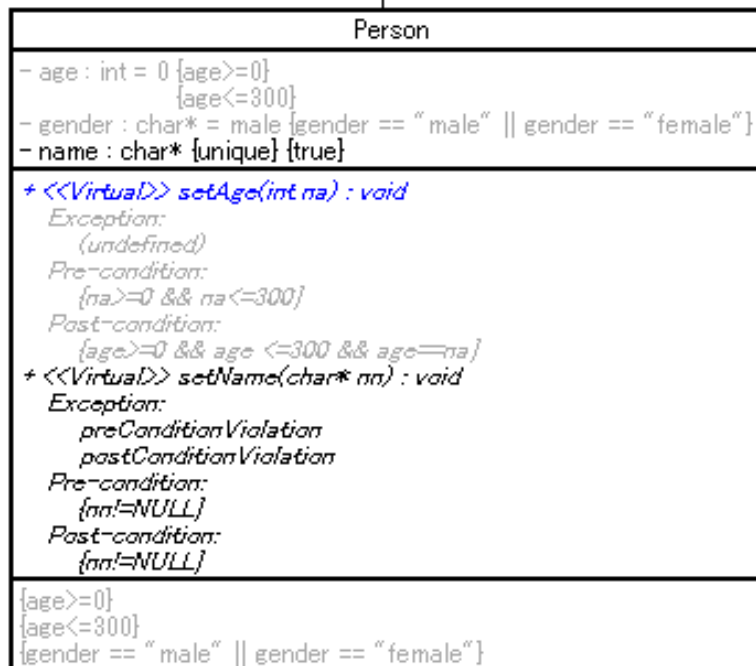
(*)HSCI Modelling tool
PCT特許出願済み/PCT Patent Pending

『Auto Modeling Pilot Controller』 機能②

科学的モデリングツール 『HOPE』 によるデモ



<<Public>>
<<LSP satisfied>>
[plug-in]



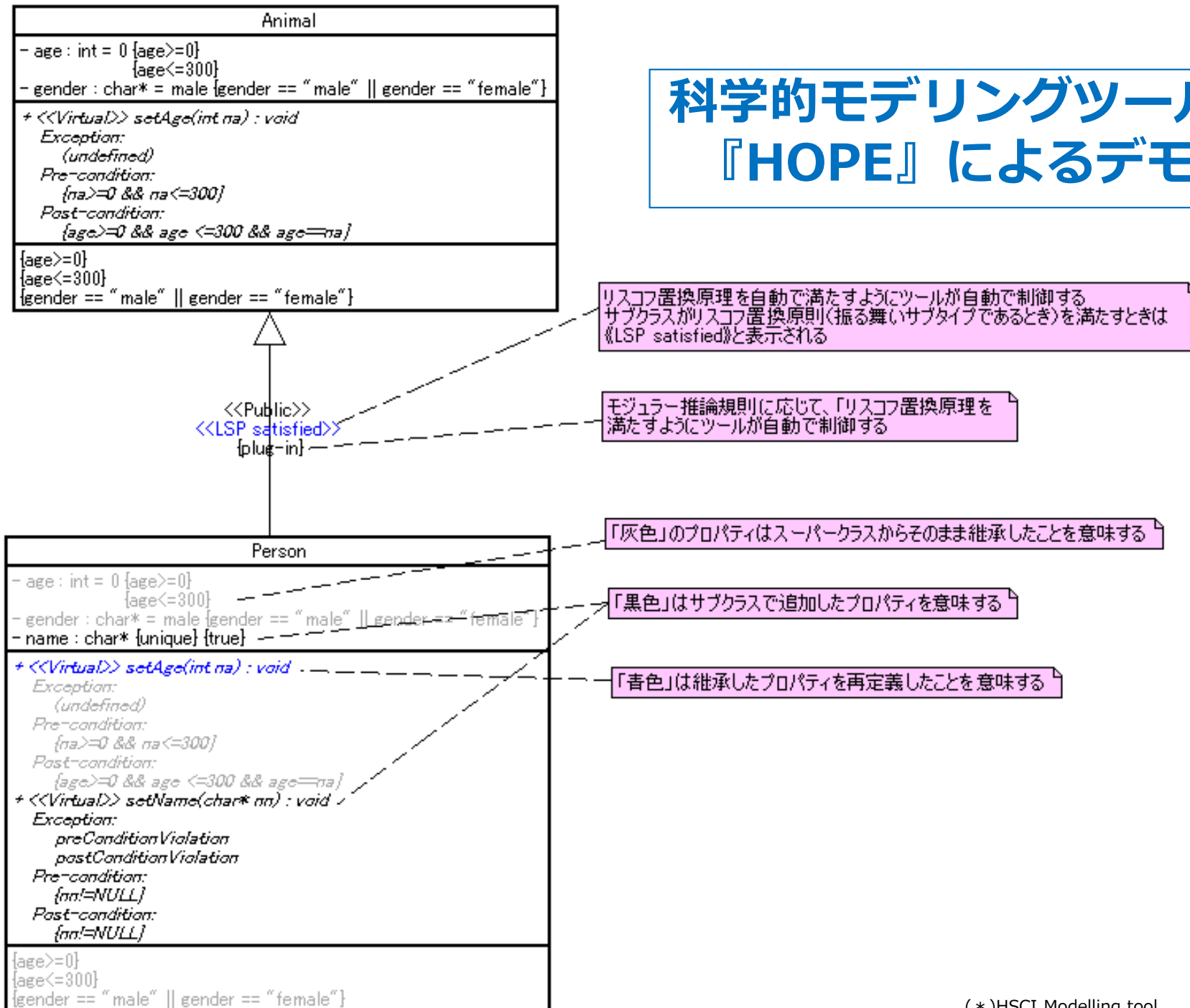
- 『Auto Modeling Pilot Controller』 機能はモジュー推論規則を選択するとONになる
- モジュー推論規則に応じて『Auto Modeling Pilot Controller』 機能が異なる

継承の可視性	☒ public	☐ protected	☐ private
表示 ☒			
仕様適合判定推論規則 (モジュー推論規則)			
表示 ☒			
☐ none	OFF: LSP depend on user		
☐ exact pre/post :	$(Cpre \Leftrightarrow SCpre) \wedge (SCpost \Leftrightarrow Cpost)$		
☐ exact pre :	$(Cpre \Leftrightarrow SCpre) \wedge (SCpost \Rightarrow Cpost)$		
☐ exact post :	$(Cpre \Rightarrow SCpre) \wedge (SCpost \Leftrightarrow Cpost)$		
☒ plug-in :	$(Cpre \Rightarrow SCpre) \wedge (SCpost \Rightarrow Cpost)$		
☐ weak plug-in :	$(Cpre \Rightarrow SCpre) \wedge (SCpre \wedge SCpost \Rightarrow Cpost)$		
☐ satisfies : (relaxed plug-in /plug-in compatibility)	$(Cpre \Rightarrow SCpre) \wedge (Cpre \wedge SCpost \Rightarrow Cpost)$		
☐ manual			

(*)HSCI Modelling tool
PCT特許出願済み/PCT Patent Pending

『Auto Modeling Pilot Controller』 機能③

科学的モデリングツール 『HOPE』によるデモ



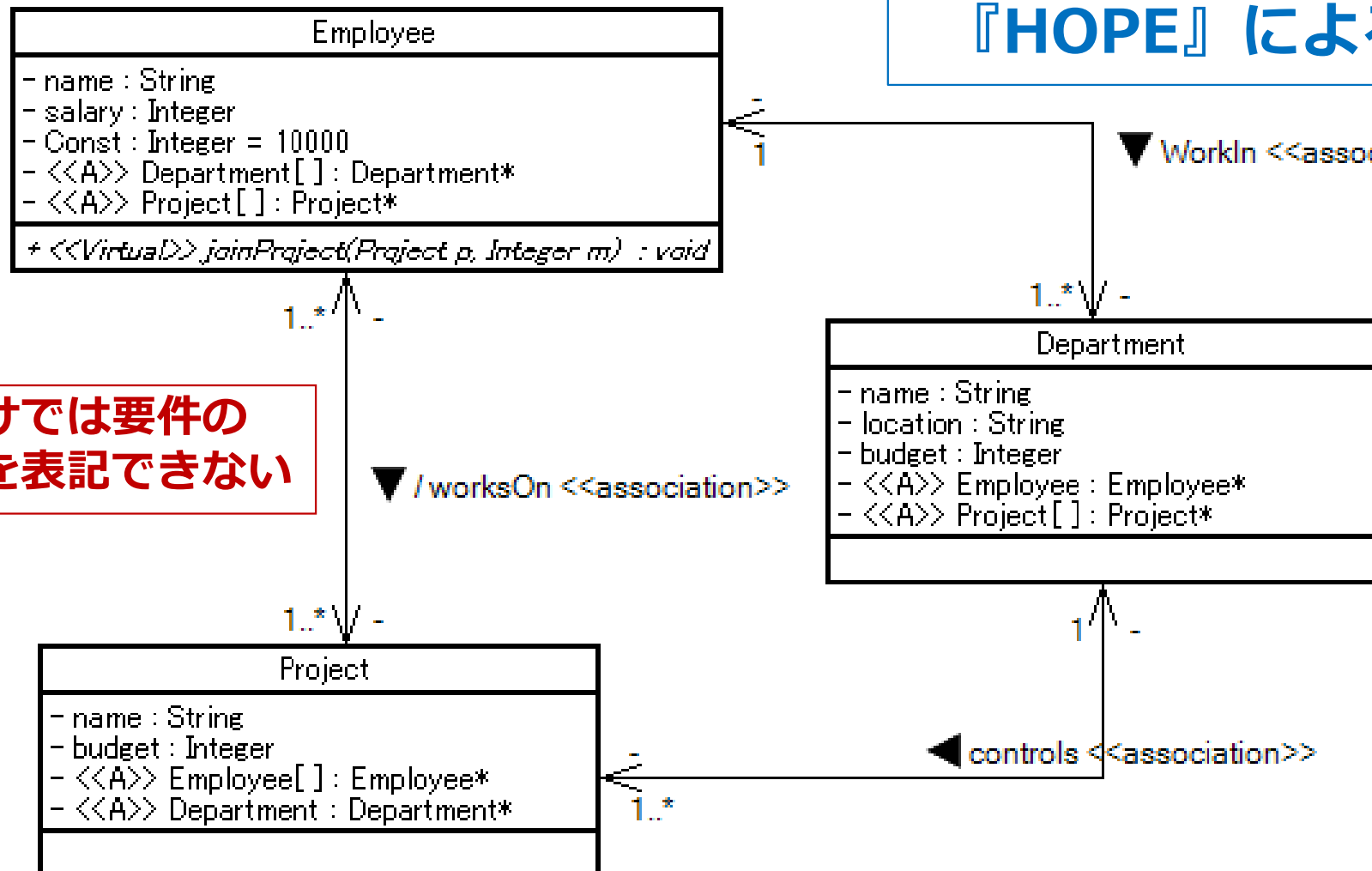
(*)HSCI Modelling tool
PCT特許出願済み/PCT Patent Pending

モデルの妥当性と制約条件

モデルで表現できない制約条件

- ① 部署には少なくともプロジェクトの数だけの従業員がいる
- ② 多くのプロジェクトに従事する従業員の方が給料が高い
- ③ プロジェクトの予算は部署の予算を超えない
- ④ プロジェクトに従事する従業員はプロジェクトを管理する部署に属していなければならない

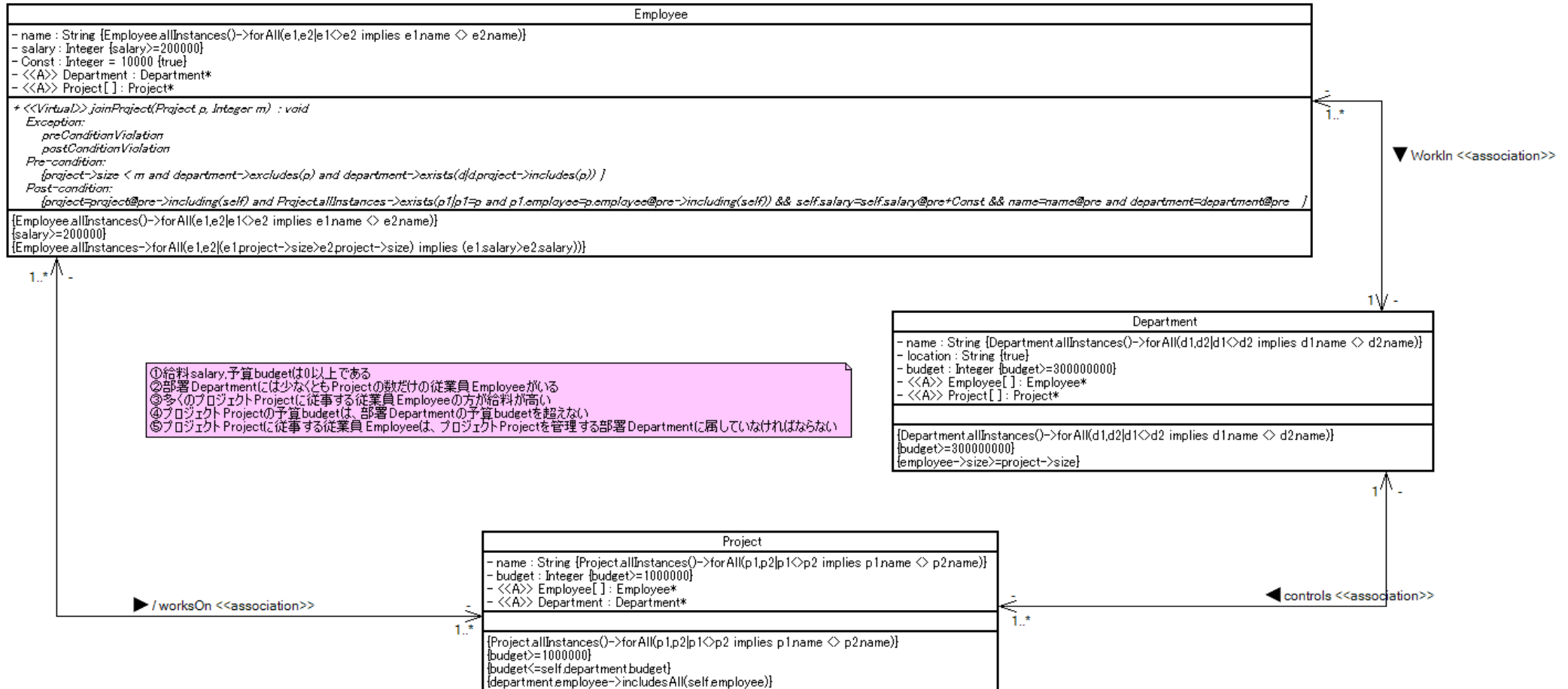
科学的モデリングツール 『HOPE』によるデモ



図だけでは要件の
制約を表記できない

UML&OCLによるモデルの妥当性と制約条件

科学的モデリングツール 『HOPE』によるデモ



UML&OCLによるモデルの妥当性と制約条件

属性の不変条件

Emplo:

- name : String {Employee.allInstances()->forAll(e1,e2|e1◇e2 implies e1name ◇ e2name)}
- salary : Integer {salary>=200000}
- Const : Integer = 10000 {true}
- <<A>> Department : Department*
- <<A>> Project [] : Project*

操作の事前条件(写像の定義域の条件)

* <<Virtual>> joinProject(Project p, Integer m) : void

Exception:

preConditionViolation
postConditionViolation

操作の例外

Pre-condition:

{project->size < m and department->excludes(o) and department->exists(d|dproject->includes(o)) }

Post-condition:

{project=project@pre->including(self) and Project.allInstances->exists(o1|o1=o and o1.employee=o.employee@pre)}

{Employee.allInstances()->forAll(e1,e2|e1◇e2 implies e1name ◇ e2name)}

{salary>=200000}

{Employee.allInstances->forAll(e1,e2|(e1project->size>e2project->size) implies (e1.salary>e2.salary))}

クラス不変条件

操作の事後条件

(*)HSCI Modelling tool
PCT特許出願済み/PCT Patent Pending

OCL(USE)例①

-- Employeeクラス

class Employee

attributes

name : String

salary : Integer

init : 10000

operations

joinProject(p:Project, m:Integer) : OclVoid

pre: project->size < m and department->excludes(p) and department->exists(d|d.project->includes(p))

post: project=project@pre->including(self) and Project.allInstances->exists(p1|p1=p and

p1.employee=p.employee@pre->including(self)) and self.salary=self.salary@pre+Const and name=name@pre and department=department@pre

constraints

inv : Employee.allInstances()->forAll(e1,e2|e1<>e2 implies e1.name <> e2.name)

inv : salary>=200000

inv : Employee.allInstances->forAll(e1,e2|(e1.project->size>e2.project->size) implies (e1.salary>e2.salary))

end

-- Departmentクラス

class Department

attributes

name : String

location : String

budget : Integer

operations

constraints

inv : Department.allInstances()->forAll(d1,d2|d1<>d2 implies d1.name <> d2.name)

inv : budget>=300000000

inv : employee->size>=project->size

end

コード生成モード ☐ C

☐ C++

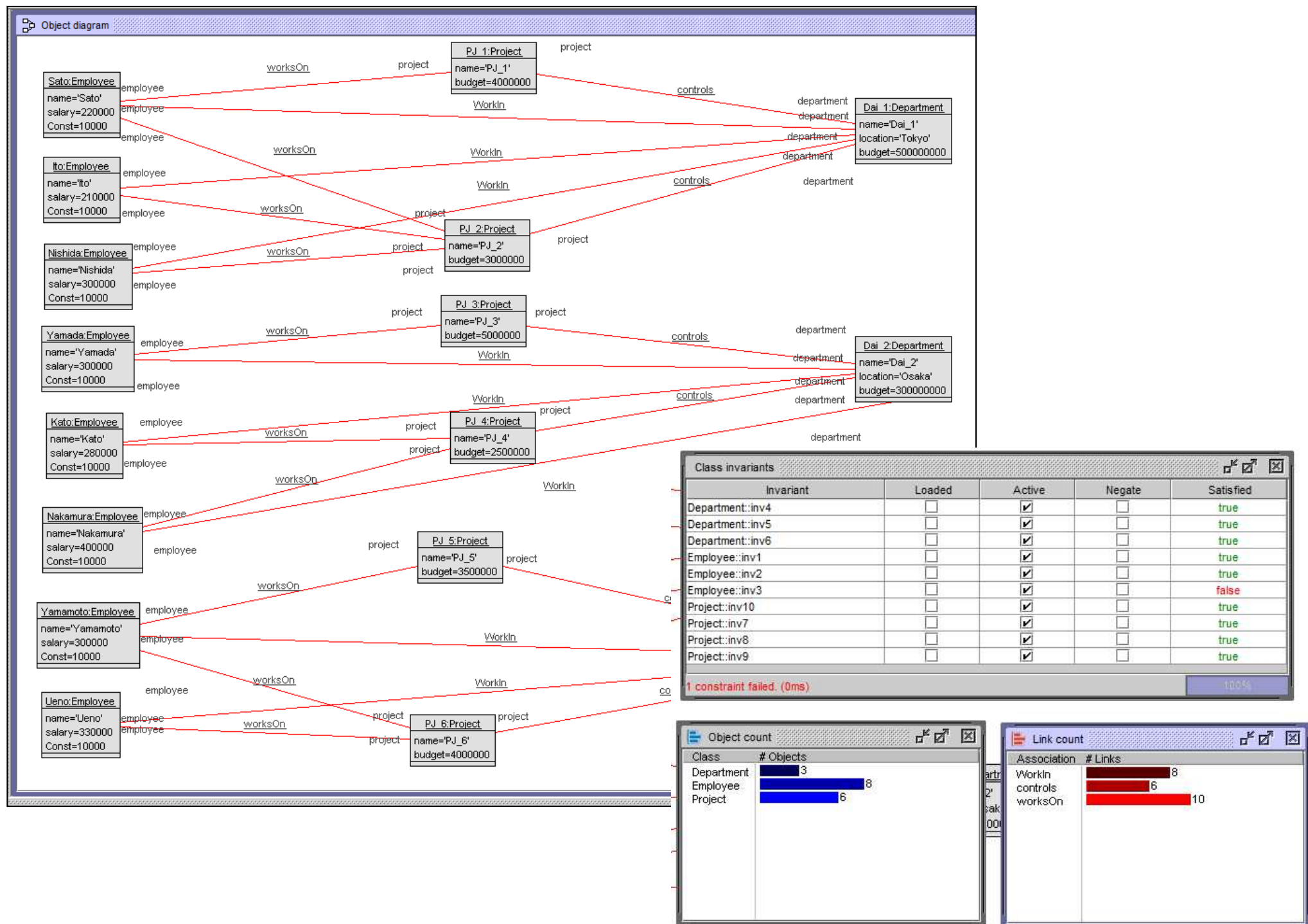
☒ USE(OCL)

コード生成

OCL(USE)例②

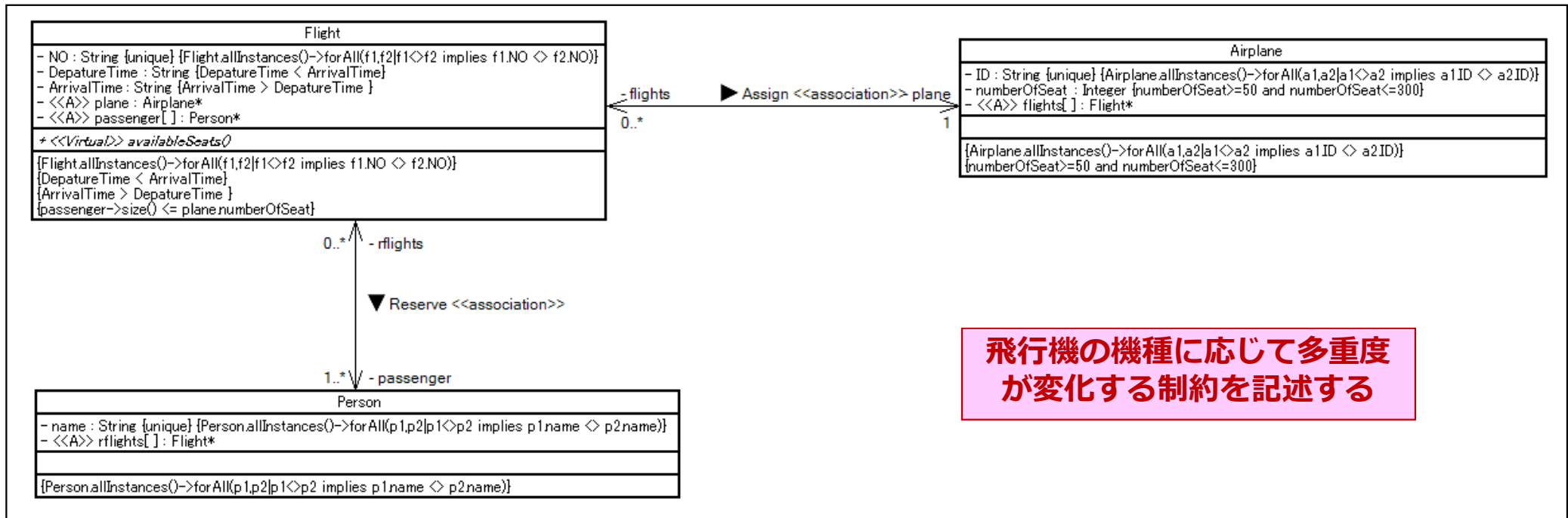
```
-----  
-- Projectクラス  
-----  
class Project  
  attributes  
    name : String  
    budget : Integer  
  operations  
  constraints  
    inv : Project.allInstances()->forAll(p1,p2|p1<>p2 implies p1.name <> p2.name)  
    inv : budget>=1000000  
    inv : budget<=self.department.budget  
    inv : department.employee->includesAll(self.employee)  
end
```

モデルの妥当性と制約条件をシミュレーション



動的多重度①

科学的モデリングツール 『HOPE』によるデモ



飛行機の機種に応じて多重度
が変化する制約を記述する

動的多重度②

-- Flightクラス

```
class Flight
  attributes
    NO : String
    DepatureTime : String
    ArrivalTime : String
  operations
    availableSeats(f:Flight) : Integer
      pre: false
      post: false
  constraints
    inv : Flight.allInstances()->forall(f1,f2|f1<>f2 implies f1.NO <> f2.NO)
    inv : DepatureTime < ArrivalTime
    inv : ArrivalTime > DepatureTime
    inv : passenger->size() <= plane.numberOfSeat
end
```

コード生成モード ☐ C

☐ C++

☒ USE(OCL)

コード生成

-- Airplaneクラス

```
class Airplane
  attributes
    ID : String
    numberOfSeat : Integer
  operations
  constraints
    inv : Airplane.allInstances()->forall(a1,a2|a1<>a2 implies a1.ID <> a2.ID)
    inv : numberOfSeat >= 50 and numberOfSeat <= 300
end
```

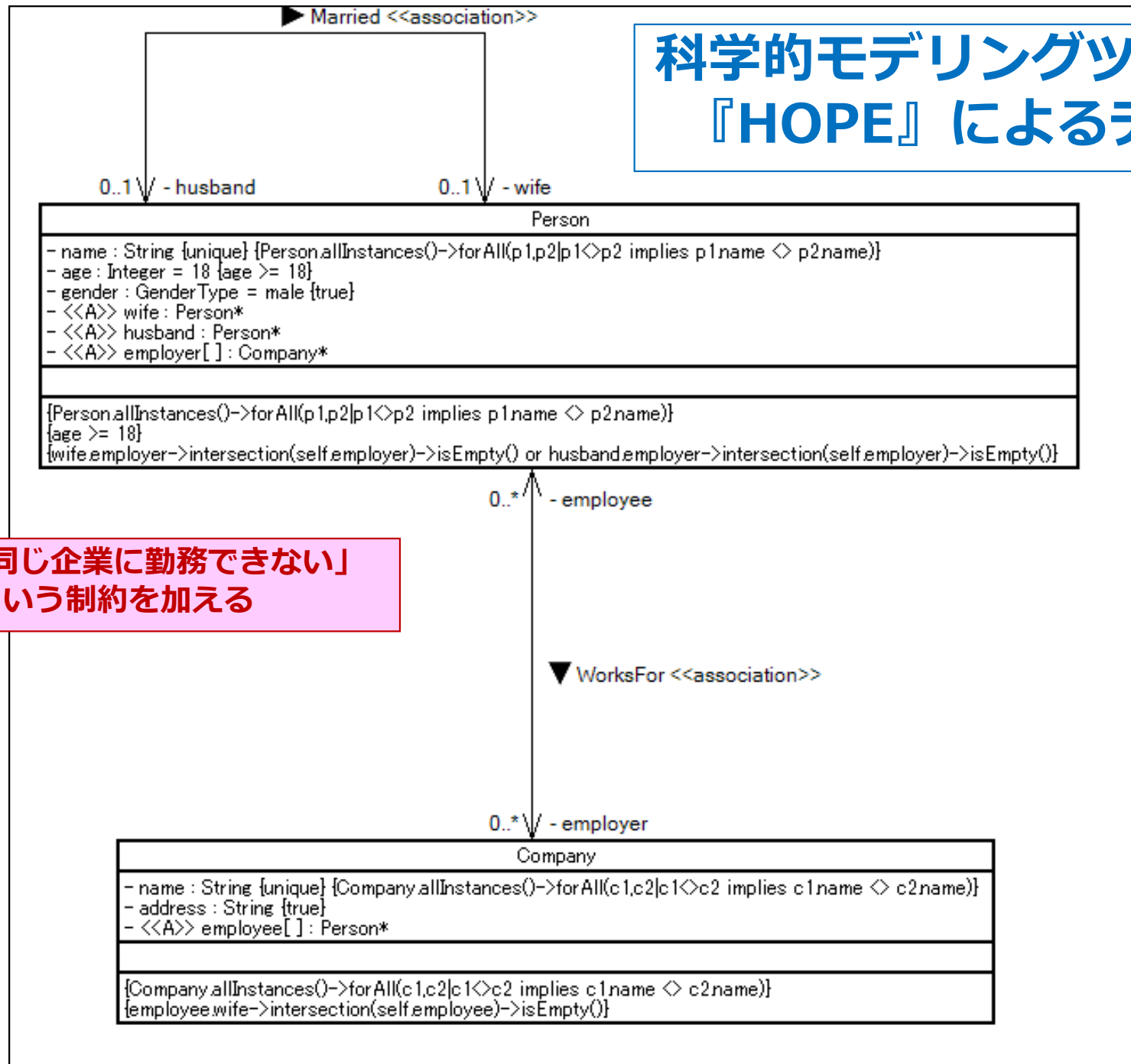
-- Personクラス

```
class Person
  attributes
    name : String
  operations
  constraints
    inv : Person.allInstances()->forall(p1,p2|p1<>p2 implies p1.name <> p2.name)
end
```

複雑な制約を持つ関連①

科学的モデリングツール 『HOPE』によるデモ

「夫婦が同じ企業に勤務できない」
という制約を加える



複雑な制約を持つ関連②

-- Personクラス

```
class Person
  attributes
    name : String
    age : Integer
    init : 18
    gender : GenderType
    init : #male
  operations
  constraints
    inv : Person.allInstances()->forAll(p1,p2|p1<>p2 implies p1.name <> p2.name)
    inv : age >= 18
    inv : self.wife.employer->intersection(self.employer)->isEmpty() or self.husband.employer->intersection(self.employer)->isEmpty()
end
```

コード生成モード ☐ C

☐ C++

☒ USE(OCL)

コード生成

-- Companyクラス

```
class Company
  attributes
    name : String
    address : String
  operations
  constraints
    inv : Company.allInstances()->forAll(c1,c2|c1<>c2 implies c1.name <> c2.name)
    inv : self.employee.wife->intersection(self.employee)->isEmpty()
end
```

HSCIの技術コラム

技術の解説は下記のURLを参考にしてください

<http://hsc-i.com/TechnologyColumn.html>

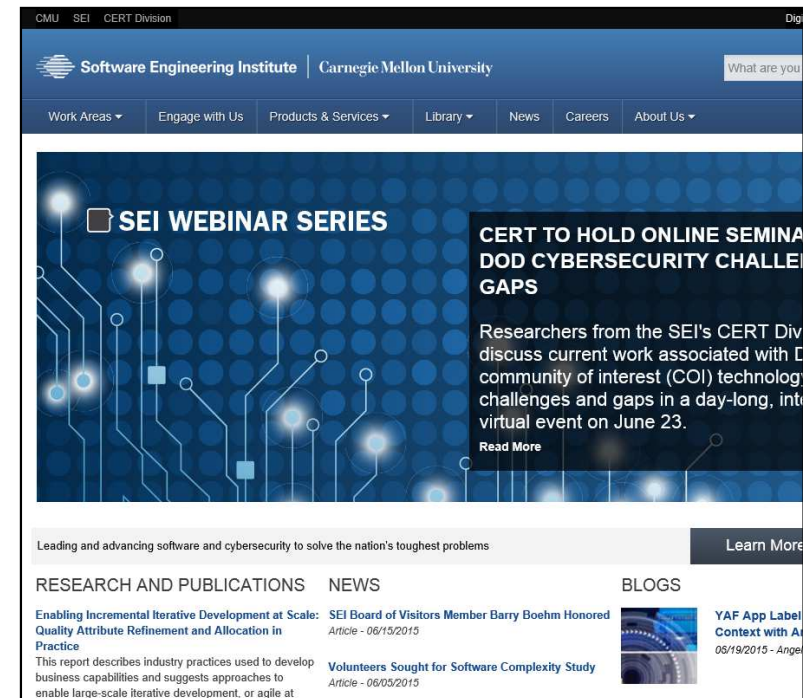
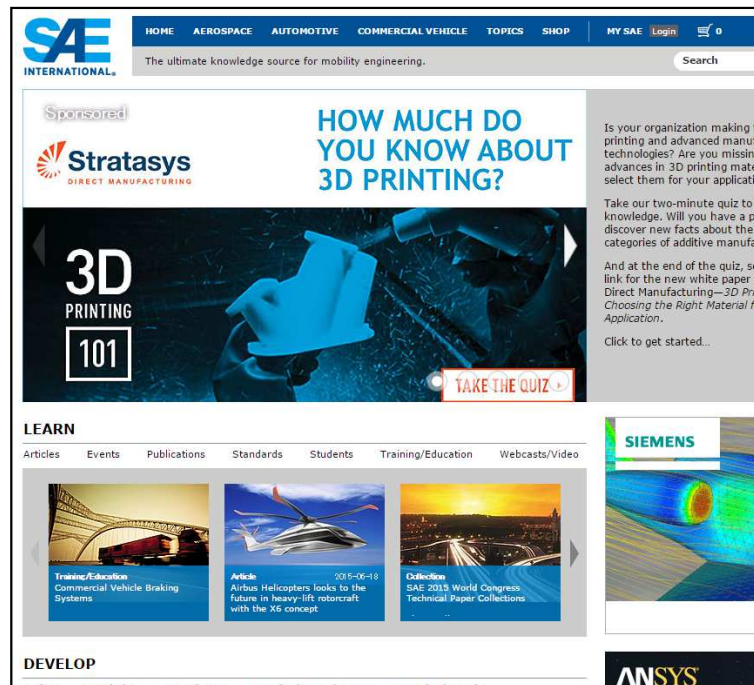


海外の自動化とモデリング の動向

科学的モデルベース開発の発展と加速

システムズ・エンジニアリングの加速

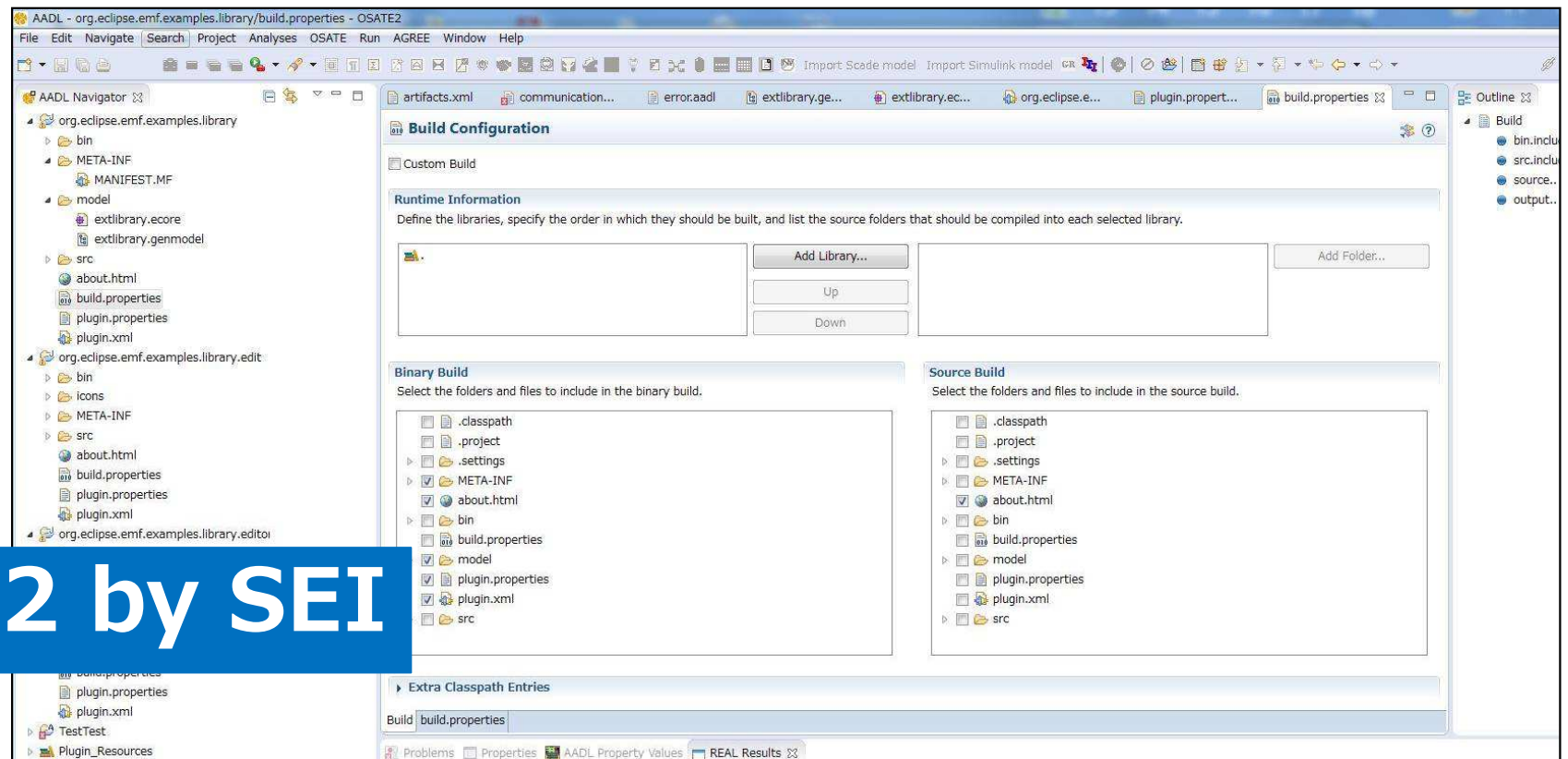
- アメリカ国防高等研究計画局(Defense Advanced Research Projects Agency)
 - 略称はダーパ (DARPA)
- SAE International
 - <http://jp.sae.org/>
- カーネギーメロン大学ソフトウェア工学研究所(SEI)
- NASA





UML/SysML + AADL

UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems



OSATE2 by SEI

Thank You