開発方式の適切な選択と効果についての研究事例

存在従属グラフからRESTful Web サービスの生成（など）

井田 明男† 金田 重郎† 森本 悠介‡

†同志社大学大学院
‡同志社大学

発表概要

I. 自己紹介，問題意識など
II. オブジェクト指向モデリング指導の難しさ
III. エンティティの存在従属分析
IV. 存在従属分析の例題への適用
V. 存在従属グラフからリレーショナルモデルを構築する
VI. 存在従属グラフからユースケース記述を構築する
VII. 存在従属グラフを中心とした開発プロセス
VIII. 存在従属グラフからWebサービスを生成する
IX. まとめ

発表概要

I. 自己紹介，問題意識など
II. オブジェクト指向モデリング指導の難しさ
III. エンティティの存在従属分析
IV. 存在従属分析の例題への適用
V. 存在従属グラフからリレーショナルモデルを構築する
VI. 存在従属グラフからユースケース記述を構築する
VII. 存在従属グラフを中心とした開発プロセス
VIII. 存在従属グラフからWebサービスを生成する
IX. まとめ

自己紹介

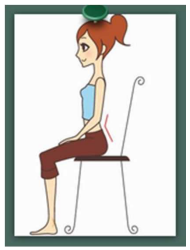
- 1959年11月生まれ，同志社大学文学部心理学専攻卒業
- 2012年までオージス総研に所属
  - モデルベース開発（オブジェクト指向とUMLを用いた開発）の普及活動
    - 企業に：証券会社，ITベンダ，複合機メーカ，医療機器メーカ，…に開発プロセステーラリングサービス，トレーニング，メンタンリングの提供
    - 大学に：同志社大学，関西大学，大阪電気通信大学，…に講義を提供
- 2012年から独立
  - 有限会社井田　代表取締役
  - 同志社大学理工学部　嘱託講師
- 関心事
  - オブジェクト指向モデリング，コード自動生成
    - 楽をするためには苦労をいとわないタイプです
  - ビジネスモデルの生成
    - BMG, BMNを読んで興味をもっています
    - ビジネスモデルとそれをサポートするITシステムの工学的なマッピング法が必要であると考えています

自己紹介（続き）

- 主な論文
  - 井田, 金田:要求分析のための日本語版ビリヤードボールモデルの提案，情報科学技術フォーラム講演論文集，vol.12, no.4, pp115-120, 2013-08-20
  - 井田, 金田:Applying Multidimensional Scaling for Responsibility Distribution between Objects—Linking Conceptual Activity Model and Class Diagram, 情報処理学会論文誌,vol.56,no.5, pp1882-7764, 2015-05-15
  - 井田, 金田, 熊谷, 藤本:存在従属性に着目した論理要件ロバストなドメインモデルの作成—ドメインクラス図をユビキタス言語として用いるために,情報処理学会論文誌情報処理学会論文誌,vol.56, no.5,pp1340-1350,2015-05-15
  - 井田, 金田, 熊谷, 矢野:エンティティの存在従属性に着目した機能規模の測定—アジャイルで網羅性の高い業務アプリケーションの見積り手法，情報処理学会論文誌 vol.57, no.5, pp.1399-1410, 2016-05-15
  - 矢野, 中西, 井田, 金田:存在従属性に基づくユースケース図の自動生成手法，電子情報通信学会論文誌 ,vol.J100-D, no.1 ,pp.14-27,2017-01-01
  - 井田, 金田, 森本:存在従属グラフからRESTful Webサービスの生成，情報システム学会　第12回全国大会・研究発表大会，ベストペーパ特別賞受賞,2016-11-12

自己紹介（続き）

- 最近のマイブーム
  - 長時間椅子に座っていることは，喫煙よりも健康に悪いと聞いてショックを受けた
  - 勝間和子氏を見習って立ち机を試してみたがどうも落ち着かない
  - そこで，極力正しい着座姿勢をとるように心がけている（ガイドライン）
  - また，60分に1回は立ち上がって，かかと落としを30回行うことにした（プロセス）
  - かかと落としは骨ホルモン:オステオカルシンの分泌を促す（成果物）
    - 骨ホルモンは，脳，肝臓，すい臓，腎臓などを活性化してくれるという（コロンビア大学ジェラルド博士）（効果）



背景・問題意識（その1）

- オブジェクト指向分析が日本人には直感方式
- 識別子を明記しない慣習があるUMLクラス図がそれを助長
  - 佐藤正美氏曰く:制約が表現されていないのはモデルではなくて単なる図式
- かといって正規化はボトムアップで収束が遅く敷居が高い？
  - 管理すべきデータ項目を集めてきてグループ分けし，グループ毎に主キー属性を定義，そして
    - Ⅰ）グループ内の繰り返し項目を分離
    - Ⅱ）主キー属性に部分関数従属している項目を分離
    - Ⅲ）主キー属性に推移関数従属している項目を分離
    - BC）非キー属性から主キー属性向けて関数従属している項目を分離
    - Ⅳ）自明でない多値従属性を排除
    - Ⅴ）結合従属性を残して分割しきる
- なにかよいモデリングガイドラインはないか？
  - 簡潔で，覚えやすく，実践しやすい

背景・問題意識（その2）

- アジリティを求めるあまり，木を見て森を見ない開発プロセスがある
  - 局所最適化されたアプリケーションが量産され，それらの相互接続性が著しく低いといった失敗例が多い
  - 全体の視点と実際に使われ維持される設計成果物の必要性
- 網羅性の低い要求分析（※次頁に参考資料）
  - 要求記述→アクター→ユースケース→ユースケース記述→クラス？
- 設計ドキュメントと実装成果物の乖離
  - 設計成果物＝実装成果物にならない限り
  - ほんとうに使われ，保守されるドキュメントはどうあるべきか

【参考】網羅性の低い要求分析

- ユースケースを手作業で識別した場合と存在従属グラフから自動生成した場合の量的の比較
  - ある地方自治体の住宅管理システムの調達仕様書を用いて

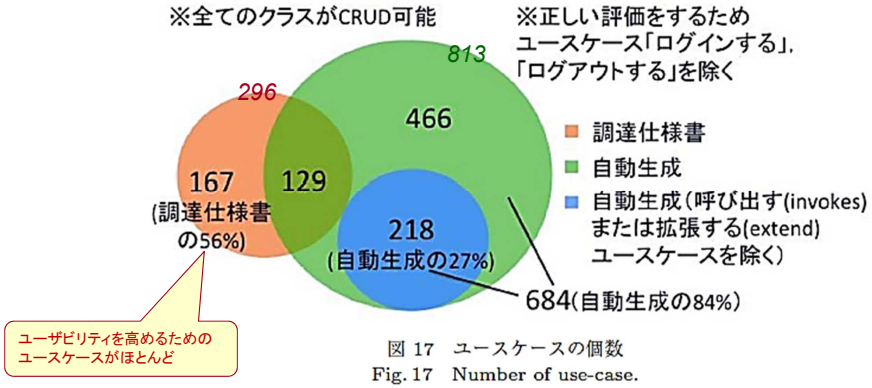
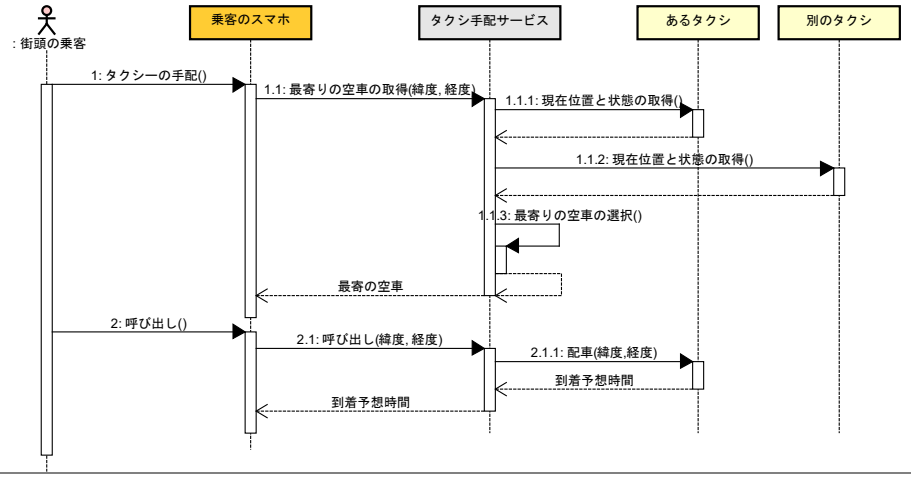


図 17　ユースケースの個数
Fig. 17　Number of use-case.

出所：矢野　寛将，中西　啓太，井田　明男，金田　重郎：存在従属性に基づくユースケース図の自動生成手法，
電子情報通信学会論文誌 D　Vol.J100-D　No.1　pp.14-272017/01/01

背景・問題意識（その3）

- IOTで社会を改善
  - 散在する製品やサービスの需要と供給をタイムリーに縁組
- システム間やデバイス間の連携がますます重要に
  - キーワードは，デバイス間Web連携と人工知能による精度の高い予測

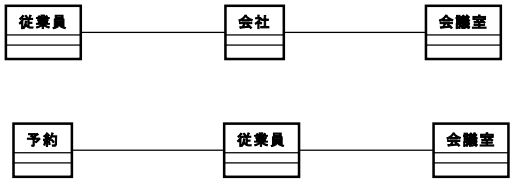


発表概要

I.　自己紹介，問題意識など
II.　オブジェクト指向モデリング指導の難しさ
III.　エンティティの存在従属分析
IV.　存在従属分析の例題への適用
V.　存在従属グラフからリレーショナルモデルを構築する
VI.　存在従属グラフからユースケース記述を構築する
VII.　存在従属グラフを中心とした開発プロセス
VIII.　存在従属グラフからWebサービスを生成する
IX.　まとめ

どうもうまく教えられません

私は学生にオブジェクト指向（OO）モデリングを教える中で，英語圏で育まれた手法の教科書をそのまま日本語に翻訳して用いても，どうも上手く教えられず，受講者のモデルも改善されない経験をしてきました．



日本人のOO初学者が作成しがちなクラス図

DOAが多少うらやましい

- DOAとOOAの大局的な対比（偏見含有注意）

項目	DOA	OOA
着眼点	属性（項目）主導型	実体主導型
アプローチ	ボトムアップ中心	トップダウン中心
モデリングプロセス	正規化	直感方式
識別子	複合主キー前提	オブジェクトIDと呼ばれる単独識別子を常に仮定

でも，実体主導で
収束の速いOOAも捨てがたい

正規形の概要

正規化レベル	概要
第1	非キー属性のうち繰り返しデータ項目などの集合要素を分離
第2	非キー属性のうち，主キーの一部だけに関数従属（部分関数従属）するものを分離
第3	非キー属性のうち，主キーから推移的に関数従属するものを分離
ボイス・コッド	非キー属性のうち，候補キーに対して部分あるいは推移関数従属するものを分離
第4	関係から多値従属性のある関係を取り出して分離
第5	関係のうち，結合従属性がある関係を 取り出して分離

日常使っている言語が違うから

よくよく考えてみると，英語ネイティブは，主語になる名詞が先ず思いつかないと，話し出せない言語を使って暮らしています．

それも，概念なのか，具体例なのか，単数なのか，複数なのか，はたまた既知なのか，未知なのか，を咄嗟に判断して，冠詞や後続する動詞を使い分けています．（これは恐るべき能力です）．

要するに英語話者は日常的にオブジェクト指向モデリングのトレーニングを積んでいるのです．

このことに気付けば，英語のOOの教科書では，何がインスタンスで何がクラスかなど，詳しく説明していないのも納得できます．（クラス図は英語なのです）

結果，英語のOOの教科書をそのまま日本語に訳して授業で用いても，直感方式，数少ない例を見て悟れ方式になってしまいます．

ありがちなのは，図書館の例を示して，ビデオレンタルショップの演習を行う．

佐藤正美氏曰く「生成規則を示せなければ，例示するしかない」
「私には，あるモノが見えるが，あなたにはそれが，見えないという不意打ちが起こってしまう」

日本語要求記述は存在文

一方，日本語はどうでしょうか？日本語には主語がありません．誰が，何を，どうした，ではなく，その結果として，事態がどのような状態にある，ことを描写します※

A:「私が，会議室を，予約しました．」

は不自然（恩着せがましく感じられる）な日本語文で，

B:「会議室は予約してあります．」（ここで会議室は主語ではなく，主題です）

が自然な日本語文です．

そして，AとBとでは，Aのほうがクラス図にしやすい文であることは，あきらかです．つまり，自然な日本語で記述された要求はクラス図化しにくいのです．

※ その結果，いつ，という時制の意識も希薄

「ある」日本語と「する」英語

- 「首討ち取ったり」
- 「首討ち取ってあり」
- わたしは，敵の主将の首を討ち取った，という状態で，いまここに存在しています．

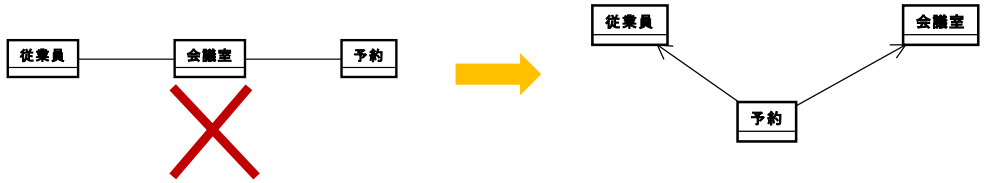
- 英語では，「I shot the captain of the enemy.」

金田グループのモデリング・ガイドライン

- 金田先生のクラス図は英語である.
  - クラス名は，可算名詞で，そのインスタンスはカウント，識別可能であるべき
  - S+V+Oは，SがOを掌握して継続的に制御している関係
  - S+V+Cは，SはCのインスタンスかサブクラスである
  - 結果，クラス間の関係はHas_aとIs_aのいずれかしかない
    - 目的語が2つあるS+V+O1+O2 は，結果的にO1がO2をhas_aする関係になる
    - Vを実施した結果，SがOを持つか，SがOになる
  - 関連は，状態動詞である.
    - ただし，日本語の場合は状態動詞・動作動詞が文脈依存
  - 操作（メソッド）は，動作動詞である
    - ただし，動作の結果を記録すべき場合は，クラスとする
- 金田先生の英語の7文型とクラス図の対応付けが可能
  - 英文は時制に厳格だから，処格だけ特別扱いすれば済むのかもしれない
  - ただし，センテンス毎にクラス図を作ると大局観が得にくい
- 井田のインスタンス間の存在従属性に着目してみればどうか
  - インスタンスと属性値の存在従属性
  - インスタンス間の存在従属性

オブジェクト指向モデリング指導の難しさ

存在従属性に着目して日本語要求記述にあたると，大局観の問題が改善されることが分かってきました



予約の前に従業員と会議室の存在が必要

いくつか例題として採り上げ，存在従属性に着目したモデリングのアイデアやガイドラインについてお話します.

ガイドラインは重厚になりがちですが，7±2程度に留めることに注意を払いました.

ここまでの小括

- オブジェクト指向は英語圏で育まれた
- クラス図は英語（S+V+Oの文）である
- 日本語要求記述は存在文で構成されている
  - 例】「花屋は契約農家から花を仕入れ，花束に加工して顧客に販売している」
  - SVOが明確でない，時制も希薄
  - 日本語はクラス図ではない
- クラス図作成は日英翻訳に近い作業である
- けれども，存在従属関係に着目して日本語要求記述にあたると，この辺の事情は改善されることが分かってきた

発表概要

I. 自己紹介，問題意識など
II. オブジェクト指向モデリング指導の難しさ
III. エンティティの存在従属分析
IV. 存在従属分析の例題への適用
V. 存在従属グラフからリレーショナルモデルを構築する
VI. 存在従属グラフからユースケース記述を構築する
VII. 存在従属グラフを中心とした開発プロセス
VIII. 存在従属グラフからWebサービスを生成する
IX. まとめ

エンティティの存在従属分析

- 目的は，ドメインモデルとしての存在従属グラフを構築すること
- 存在従属グラフが手に入れば
  - ドメインについて業務屋さんとシステム屋さんが互いに理解できる
  - 識別子と多重度が迷いなく決まり，整合性要件を満たしたデータベースモデルに結びつく
  - 結果，知識のモジュラリティが高くなる
    - 知識の追加，更新，削除が問題なく行える
  - 存在従属グラフを起点に
    - ユースケースの生成ができる
    - ユースケース記述の生成ができる
    - 機能規模を計測することができる
    - 存在従属グラフをカプセル化するAPIを生成することができる
    - 他にもいろいろ使えるかも知れない

エンティティ：業務において重要な管理対象である物事（thing）
ドメイン：要求される知識が似ている業務領域
ドメインモデル：実装独立の本質的なモデル　ただし，実装に結びつかないドメインモデルは役に立たない
渡辺幸三氏曰く「アプリケーションというものは「テーブル構造から導出可能な多彩な影」でしかない」

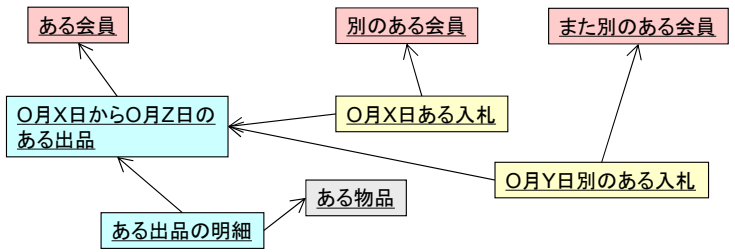
存在従属関係とは

- （業務要件上）あるインスタンスの存在が，別のインスタンスの先立つ存在を前提としているとみなせる場合，前者のインスタンスが属するクラスは，後者のインスタンスが属するクラスに存在従属している，と考えます
  - P.チェン（1977）はexistence dependencyを紹介，前者を弱実体，後者を強実体と呼んだ
  - けれども，P．チェンのexistence dependencyと我々の存在従属は若干異なる
- たとえば，オークションサイトのドメインであれば
  - 出品は，会員と，物品に存在従属している
  - 入札は，出品に存在従属している
  - 落札は，入札に存在従属している
  - 代金の支払は落札に存在従属している
  - 商品の発送は支払に存在従属している（前払いの場合）

などの存在従属関係が認められる，存在従属関係はドメインの至る所で認められる

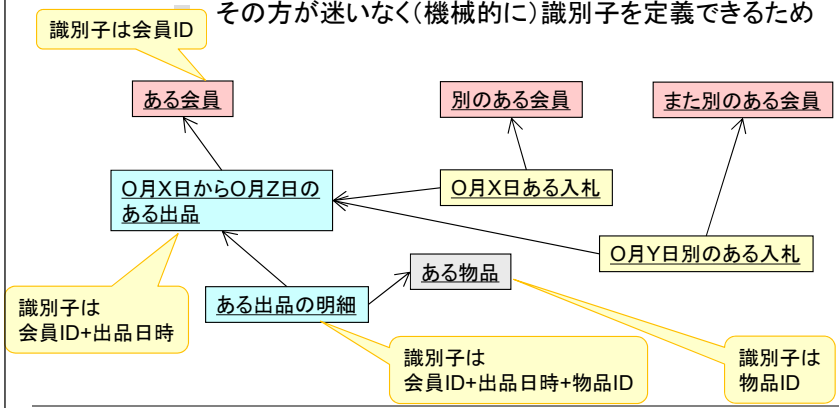
練習問題　インスタンス図を描いてみましょう

- 会員制のオークションサイトを考えます．
  - 会員でないと出品や入札はできません
  - 会員は入札期間を決めて出品します
  - 1回の出品で複数の商品個体を出品できます
  - 会員は出品に対して入札します



【参考】Peter ChenのExistence Dependency

- Peter Chenはエンティティ間の関係としてExistence Dependency（ED）とID Dependency（IDD：識別子従属性）を導入している
  - IDDは従属クラスの識別子の一部に従属先の識別子が含まれている関係
  - Peter ChenはEDはIDDを必ずしも含意しないとしている
  - 我々は存在従属であればIDDを含意させたいと考えている
    - その方が迷いなく（機械的に）識別子を定義できるため



存在従属分析における識別子のガイドライン

- イ)　あるデータ項目をあるクラスの属性とするには，その値がそのクラスのインスタンスに存在従属しかつライフサイクルが等しい場合に限る．
  - 属性値は，あるオブジェクトが生まれるともに生まれ，オブジェクトが存在している間は値は変化するとしても存在し続け，オブジェクトが消滅すると同時に消滅する値である
  - これを属性従属（Attribute Dependency）と呼んでいます
- ロ)　そのインスタンスが，独立して存在可能なクラスのIDには「必ず」単一属性のIDを与える．
  - オブジェクト指向のOID方式と同じ人工的なIDです．
  - 自然キー（業務で使われているキー）は使いません．自然キーは業務の都合で変更される可能性が高いからです．
  - 自然キーを非キー属性として持たせるのは問題ありません．

識別子のガイドライン

- ハ)　そのインスタンスが，独立して存在できない（存在従属な）クラスのIDには，「決して」単一属性のIDは与えず，「必ず」その存在根拠（前提）となるクラスのIDを含むIDを与える．結果，存在従属なクラスの識別子は複合主キーとなる．
- ニ)　そのインスタンスが，（リソースではなく）イベント※の場合は，上記のルールに加えて，そのクラスのIDに「必ず」生起時点をあらわす時刻または版（バージョン）のシリアル値を含める．
- ニ’）そのインスタンスが，（リソースではなく）期間リソース※の場合は，上記のルールに加えて，そのクラスのIDに「必ず」開始時点をあらわす時刻または版（バージョン）のシリアル値を含める．

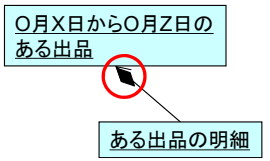
※受注のように時点が帰属するクラスのこと
※契約のように期間が帰属するクラスのこと

識別子のガイドライン

- ホ)　存在従属でないクラス間の関係は，汎化関係と単なる参照関係のみとする．
- i.　汎化関係の場合：サブクラスのIDは「必ず」スーパークラスと同じIDを共有する．たとえば，「プレミアム会員」のスーパークラスが「会員」クラスで，そのIDが「会員ID」であった場合は，「プレミアム会員」クラスのIDも「会員ID」となる．
- ii.　単なる参照関係の場合：参照元のインスタンスと参照先のインスタンスのそれぞれのライフサイクルは独立している．このような場合は，表記では参照元のクラスから参照先のクラスへの依存関係を定義し，RDBでは参照元は参照先のIDを外部キーとして保持するものとする．

属性従属関係

- 存在従属関係は，インスタンス間だけの関係ではない
- インスタンスとその属性値の間の関係も存在従属関係である
  - 属性従属関係（Attribute Dependency）と呼びましょう（存在従属の強い関係）
- そもそも属性値とは，
  - インスタンスが生まれるとともに生まれ，インスタンスが存在する間は存在し，インスタンスの消滅とともに消滅する値
- そうであれば，クラスとその属性の間の関係も存在従属関係であるべき
- たとえば，「井田明男」の「1959/11/05」，「A型」，「男」は属性値，「人」の「生年月日」，「血液型」，「性別」は属性
- 無数にあるので，業務要件に照らして，管理の必要がある属性だけをピックアップすることになっている



佐藤正美氏曰く「しかじかの個体（エンティティ）には，かくかくの性質（属性）が帰属する，という判断は，「本質的属性」という「哲学的にやっかいな論点ー存在論とか形而上学などーを考慮しなければならない」

展開の手順（存在従属分析手順）

- 【Step0】日本語要求記述をよく読む
- 【Step1】独立クラスを網羅する
  - オークションサイトであれば，会員，物品など
  - オークションサイト自体も独立クラスであるが，今はその中の業務をモデル化しているので，モデルには含まない
- 【Step2】独立クラスに存在従属するクラスを検討していく
  - 出品は，会員に存在従属する
  - 出品明細は，出品と物品とに属性従属する
  - 入札は会員と出品とに存在従属する
- 【Step3】存在従属関係以外の関係を見つける
  - 参照関係（後天的で取り換え可能な関係）
    - （落差物品の）発送はお届け先を参照する
  - 汎化関係（is_a関係，サブクラスのバリエーションを示す）
    - 一般会員，プレミアム会員は会員のバリエーションである

ここまでの小括

- 存在従属関係は「存在の前提となるクラス」←「依存するクラス」の関係
- 理解しやすく，ドメイン中の至る所でみつけられる
- 存在従属関係は「クラス」と「属性」の間の関係でもある
  - インスタンスに存在従属しないような値を属性値にしてはいけません
- 存在従属関係以外の関係としては，つぎの2種類がある
  - （単なる）参照関係　生まれた後で縁組みされた関係，依存先取り替え可能
  - 汎化関係（is_a関係，サブクラスのバリエーションを表現）
- 日本語要求記述から存在従属グラフを段階的に構築していく

発表概要

I. 自己紹介，問題意識など
II. オブジェクト指向モデリング指導の難しさ
III. エンティティの存在従属分析
IV. 存在従属分析の例題への適用
V. 存在従属グラフからリレーショナルモデルを構築する
VI. 存在従属グラフからユースケース記述を構築する
VII. 存在従属グラフを中心とした開発プロセス
VIII. 存在従属グラフからWebサービスを生成する
IX. まとめ

例題による手順とモデルの表記

【例題】

専門学校などで，1週間の時間割の策定を支援するアプリケーションを考える．
- 学級と曜日と時限が決まれば，授業が決まる．
- おのおのの授業には，講師，教室，そして教科が必要である．
- 教科によって担当できる講師や教室の要件（座学教室か実習教室か）に制約がある．
- 講師も非常勤の場合は，出講（登壇）可能な曜日と時間帯に制約がある．
- 教科もその学級で1週間で実施すべき教科が過不足なく網羅されなければならない．

例題による手順とモデルの表記

【Step1】 当該ドメインにおける独立クラスを網羅し，それぞれに識別子を付与する.

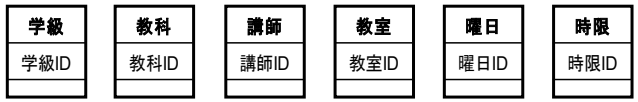


図 1 例題ドメインの独立クラス群

独立クラスには，その識別子として連番などの人工的な単一属性の主キーを付与しなければならない.

自然キーは業務で使っている識別子であるため，業務の都合により変更される

例題による手順とモデルの表記

【Step2】 独立クラスのインスタンスに存在従属するデータ項目（すなわち属性）を記述する

図 2 属性が追加された例題ドメインの独立クラス群

例題による手順とモデルの表記

【Step3】 当該ドメインの独立クラスに存在従属するクラスを網羅する

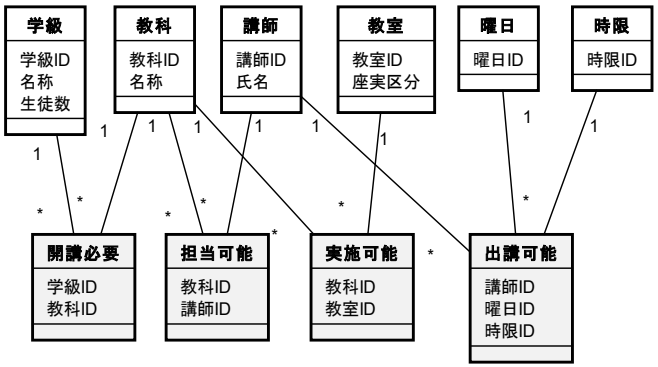


図 3 Step3実施後の例題のドメインモデル

例題による手順とモデルの表記

- 存在従属を明確に表現する
  - 識別子のガイドラインに従った上で，存在従属関係を明確に示せれば，識別子も多重度も自明のため省略可能

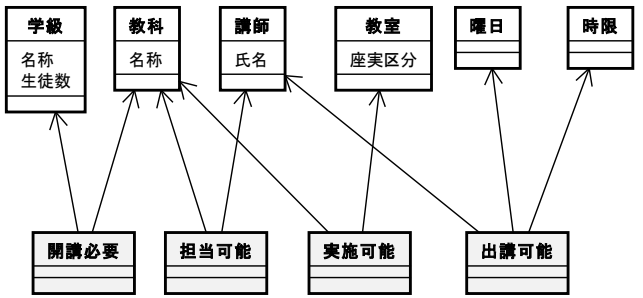


図 4 図3の提案記法による表記

存在従属性を表現するためにUMLのモデル要素である誘導可能性を使用している.
ドメインモデルは，ユースケース実現のモデリングを実施する以前のモデルであるため，誘導可能性がモデルに登場することはないため，このモデル要素を流用することにした.

【参考】例題による手順とモデルの表記

- 表記はコンポジションではだめですか？
  - 適切な場合と，そうでない場合がある

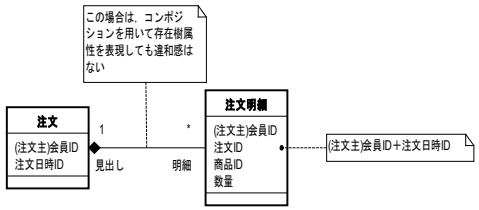


図 5 コンポジションによる存在従属性の表記（適切）

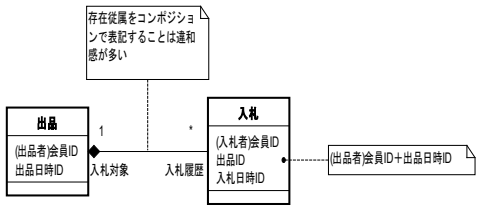


図 6 コンポジションによる存在従属性の表記（不適切）

見出し－明細パターンに見られる存在従属性を表現する場合はコンポジションが相応しい

入札は，あきらかに出品に存在従属する（存在しない出品に対する入札は不可能であるため）が，
コンポジションを用いて表記した場合は，
入札は出品の部分であるとは考えにくいため
違和感が残る

【参考】例題による手順とモデルの表記

- 多重度が厳密に1であれば，存在従属では？
  - そうとは限りません

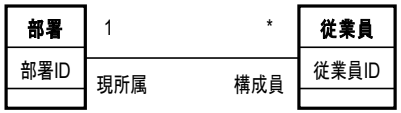


図 7 存在従属ではない多重度1の関連の例

このモデルは，従業員が部署に存在従属することを表しているのではなく，
「従業員たるものは常時どこか1つの部署に所属しなければならない」という業務ルールを表わしている．

例題による手順とモデルの表記

【Step4】 当該ドメインにおける従属クラスに，さらに存在従属するクラスを識別する

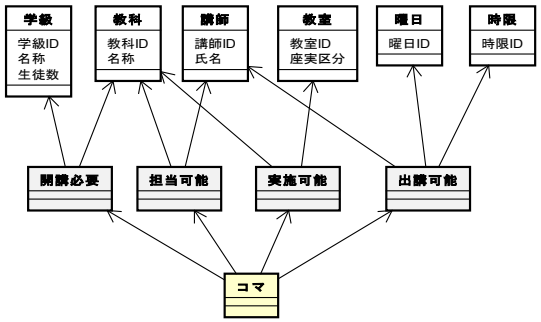


図 8 例題ドメインのStep4実施後のモデル

図8は，時間割の各コマが満たすべき論理要件を正確に反映している．
時間割のコマのすべてのインスタンスは，
ある学級において開講が必要な教科について，
その教科が担当可能でかつその時間帯に出講可能な講師と，
その教科を実施可能な教室の先立つ存在を前提として存在できる
これは，時間割策定支援アプリケーションの論理要件そのものである上，
このモデルは第４正規形を満たす

例題による手順とモデルの表記

- 提案手法はモデルの正規化レベルを自然に向上させる

表 2 提案手法の正規化レベルへの寄与

正規化レベル	概要	提案手法における対応
第1	非キー属性のうち繰り返しデータ項目などの集合要素を分離	対応なし．しかし，繰り返す属性は，発見に労を要しないため，発見次第直ちに別の実体として分離すれば問題なし
第2	非キー属性のうち，主キーの一部だけに関数従属（部分関数従属）するものを分離	実体に存在従属するデータ項目のみを属性とするため，当初から分離される
第3	非キー属性のうち，主キーから推移的に関数従属するものを分離	実体に存在従属するデータ項目のみを属性とするため，当初から分離される
ボイスコッド	非キー属性のうち，候補キーに対して部分あるいは推移関数従属するものを分離	実体に存在従属するデータ項目のみを属性とするため，当初から分離される
第4	関係から多値従属性のある関係を取り出して分離	このような関係は従属クラスとして識別されるため，当初から分離される
第5	関係のうち，結合従属性がある関係を 取り出して分離	このような関係は従属クラスとして識別されるため，当初から分離される

データモデル

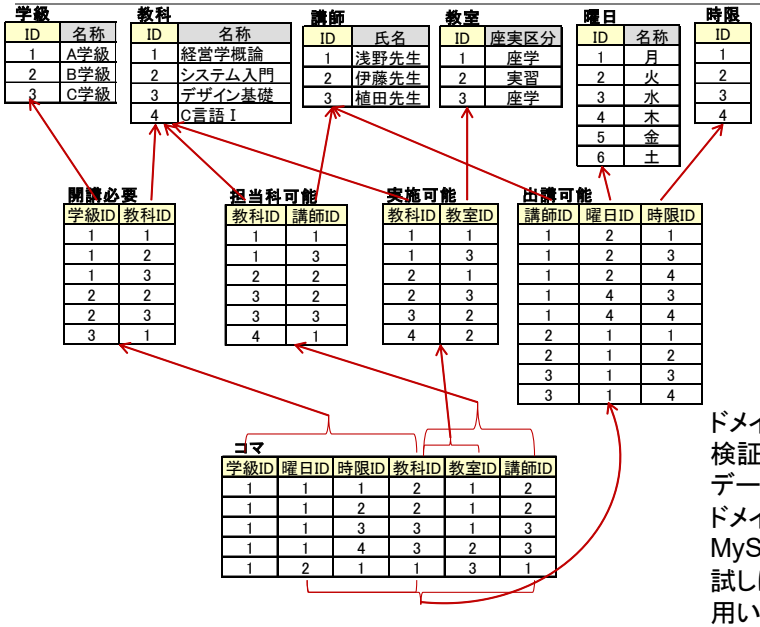


図 9 検証のために用いた例題のデータモデル

ドメインモデルの頑健性を検証するために作成したデータモデルの例

ドメインモデルから直接MySQLを使って実装し，試しにいくつかの具体値を用いてデータの追加，更新，削除を行ってみた．

試行結果

- 結果，データモデルは，以下の当り前の業務要件を愚直にクリアし，時間割策定支援のアプリケーション側の整合性検査ためのロジックを大幅に減らした．
  - 存在しないリソースの組み合わせは決して登録できない
  - 存在しない組み合わせからなる組み合わせも決して登録できない
  - あるリソースが使われる組み合わせが存在している限り，そのリソースは削除できない
  - ある組み合わせが使われている組み合わせが存在している限り，その組み合わせは削除できない

ここまでの小括

- 存在従属関係の考え方に存在従属した存在従属グラフを描くことでドメインの理解が深まった．
- 存在従属グラフはシンプルなので業務のエキスパートへの要求の確認にも使いやすいはず．
- 存在従属グラフから得られたデータモデルは，当たり前の業務（整合性）要件を愚直にクリアし，アプリケーション側の整合性検査ためのロジックを大幅に減らした．

発表概要

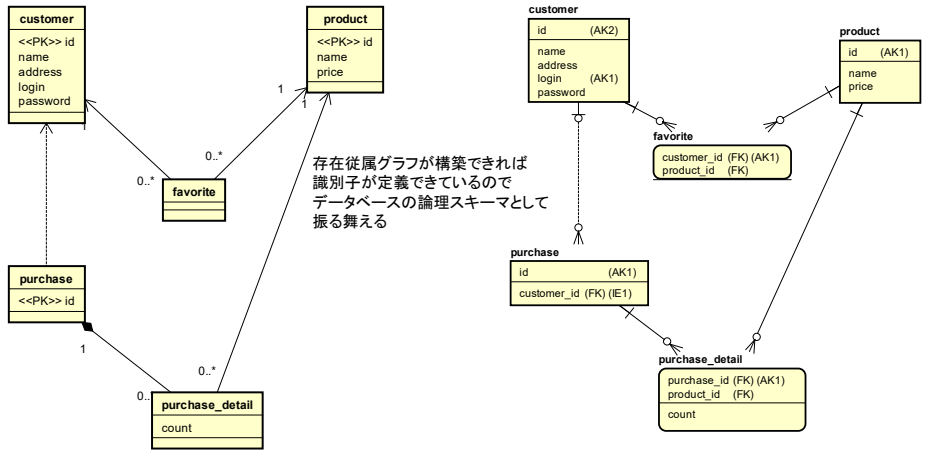
I. 自己紹介，問題意識など
II. オブジェクト指向モデリング指導の難しさ
III. エンティティの存在従属分析
IV. 存在従属分析の例題への適用
V. 存在従属グラフからリレーショナルモデルを構築する
VI. 存在従属グラフからユースケース記述を構築する
VII. 存在従属グラフを中心とした開発プロセス
VIII. 存在従属グラフからWebサービスを生成する
IX. まとめ

オブジェクトモデルとリレーショナルモデルの相違点

- 存在従属モデルはそもそもトップダウンアプローチで得られたオブジェクトモデルであるが，識別子のガイドラインに従えば，そのままリレーショナルモデルに変換可能である．存在従属モデルは，インスタンスの存在期間に関する制約を反映したモデルであるため，正しくモデル化を行えば多対多の関連は排除され，第4正規形以上の正規化レベルを持ったモデルが得られる．
- オブジェクトモデル
  - オブジェクトモデルでは，エンティティはクラスとして表現される
  - 関連には誘導可能性と呼ばれる方向の概念が存在することに注意が必要
- リレーショナルモデル
  - リレーショナルモデルではエンティティはリレーションとして表現される
  - リレーションは読み出し時に外部キーで結合するのが前提であるため，誘導可能性は双方向である
- 上記のような差異は，仮想表および仮想表に付随するサービスの層を設けることで吸収しやすい．

識別子設定の例

顧客と製品は独立クラスである
お気に入りは，顧客と製品の間に存在従属する
注文は，購買を参照する
購買明細は，購買に属性従属し製品に存在従属する



存在従属グラフ　　ERD（IE表記）に変換

難しい正規化理論を表に出さず

- ガイドラインではインスタンスに存在従属する値だけが属性値
  - 結果，属性値はインスタンスに完全関数従属するものだけ（BC正規形）
  - クラスに部分関数従属や推移関数従属するようなテータ項目は最初から，その属性にはされない
  - たとえば製造番号（シリアル番号）のように製品個体の属性とすべきデータ項目が製品種類の属性とされたり，逆に型番のような製品種類の属性とすべきデータ項目が製品個体の属性にされてしまうような誤りも回避される
  - また，自明でない多値従属性も分け入るスキがない（後述）

自明でない多値従属性も分け入るスキがない

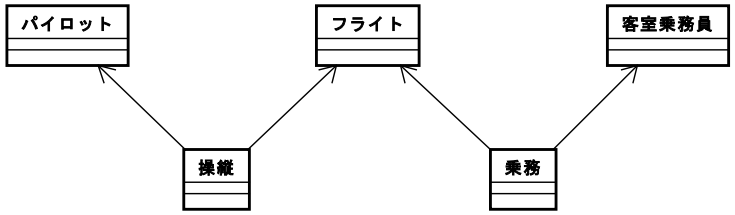
自明でない多値従属性の例：

- あるフライト（f1）にパイロット2名がアサインされました（p1,p2）
- そのフライト（f1）に客室乗務員3名もアサインされました（c1,c2,c3）
- この状況を記録すべく，つぎのようにタプル化しました※
  (f1,p1,c1), (f1,p1,c2), (f1,p1,c3),
  (f1,p2,c1), (f1,p2,c2), (f1,p2,c3)
- フライトf1が決まれば，パイロットp1とp2が決まる（多値従属）
  フライトf1が決まれば，客室乗務員c1とc2とc3が決まる（多値従属）
  けれどもパイロット（p*）と客室乗務員（c*）との間には何の関係もない
  - タプル（リレーションには）には，フライト↠パイロット｜客室乗務員　という自明でない多値従属関係が混入しています

※アサインを漠然と捉えているとこのようなタプルが生み出されてしまう．
パイロットはフライトを操縦するためにアサインされた，
客室乗務員をフライトに乗務するためにアサインされた，と捉える

結果，第4正規形が得られる

- けれども，我々の存在従属分析では最初から次のようにモデル化できる：



執拗な存在従属性の点検作業は，正規化と変わらない

ここまでの小括

- 存在従属グラフが構築できれば，識別子は機械的に設定できる
- 存在従属グラフからリレーショナルモデルへの変換作業はストレートフォワードな作業となる
- 存在従属の考え方が理解できたなら，あなたは強力なモデリングのガイドラインを手に入れたことになる

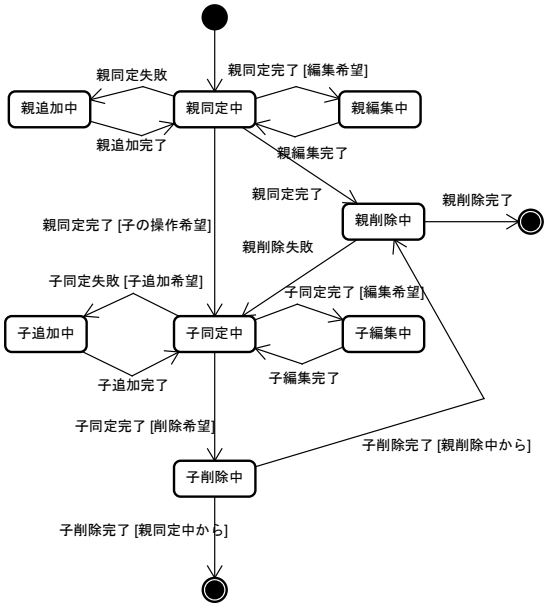
発表概要

I. 自己紹介，問題意識など
II. オブジェクト指向モデリング指導の難しさ
III. エンティティの存在従属分析
IV. 存在従属分析の例題への適用
V. 存在従属グラフからリレーショナルモデルを構築する
VI. 存在従属グラフからユースケース記述を構築する
VII. 存在従属グラフを中心とした開発プロセス
VIII. 存在従属グラフからWebサービスを生成する
IX. まとめ

存在従属グラフからユースケース記述を構築

- ユースケース記述としてのイベントフローを書くのは，初学者には難しいとされるが，存在従属グラフができてしまえば，論理的に導ける
- 存在従属グラフの歩き方
  - 生成するとき
    - 独立クラスのインスタンスを生成する
    - 生成したインスタンスのIDを使って従属クラスのインスタンスを生成する
  - 参照／更新するとき
    - 独立クラスのインスタンスを同定する
    - 同定したインスタンスのIDを使って従属クラスのインスタンスを同定する
  - 削除するとき
    - 独立クラスのインスタンスを同定する
    - 同定したインスタンスのIDを使って従属クラスのインスタンスを同定する
    - 従属クラスのインスタンスを削除する
    - （必要に応じて）独立クラスのインスタンスを削除する

存在従属グラフを辿る際のユースケースの状態



これはHATOASリンクの
生成にも使えそう！

- 親は従属先のクラスのインスタンス，子は従属クラスのインスタンス

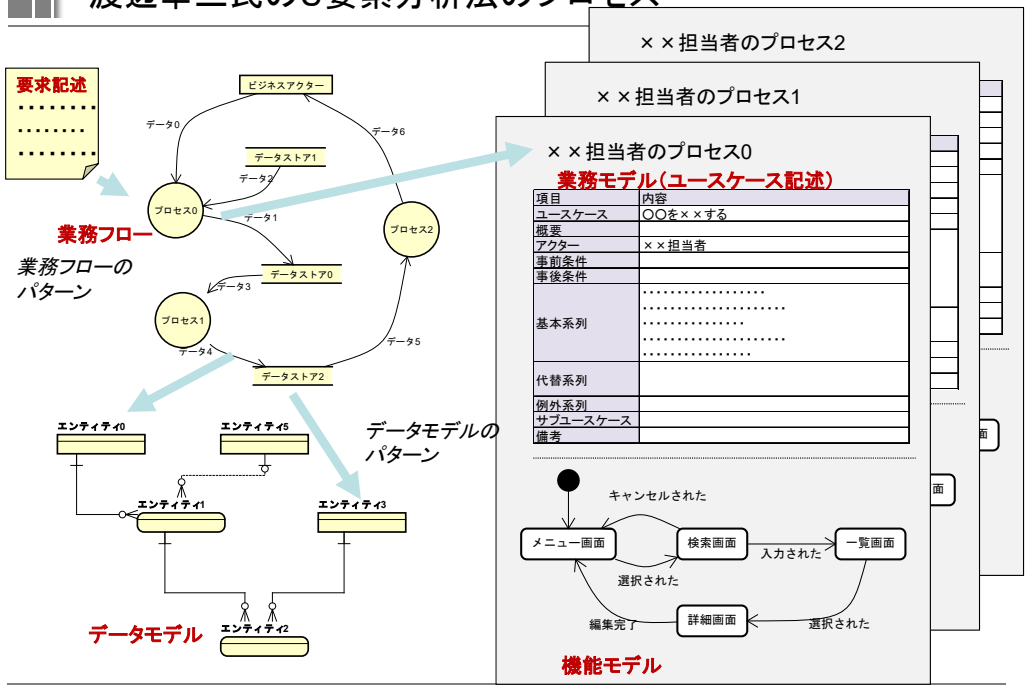
これまでの小括

- 存在従属グラフが構築できれば，ユースケース記述も機械的に導ける（だろう）
- ただし，代替フローや例外フローについては未検討
- また，以下の制約を考慮する必要がある
  - 相互排他性制約
    - 生成される子はクラスAかクラスBのいずれかのインスタンスでなければならない
  - 相互包含性制約
    - 生成される子はクラスAとクラスBの両方のインスタンスでなければならない

発表概要

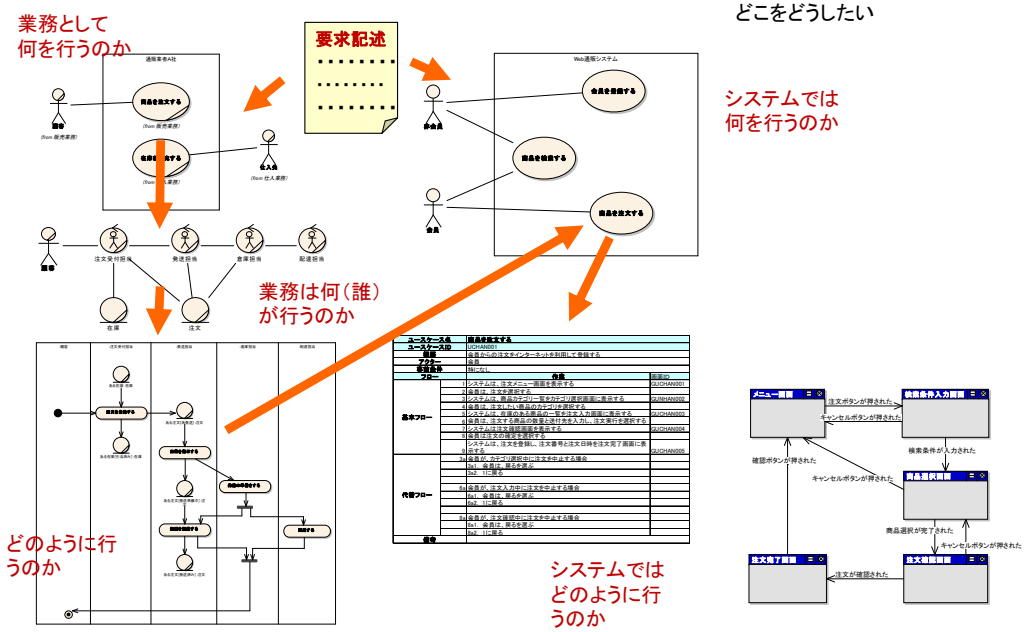
I. 自己紹介，問題意識など
II. オブジェクト指向モデリング指導の難しさ
III. エンティティの存在従属分析
IV. 存在従属分析の例題への適用
V. 存在従属グラフからリレーショナルモデルを構築する
VI. 存在従属グラフからユースケース記述を構築する
VII. 存在従属グラフを中心とした開発プロセス
VIII. 存在従属グラフからWebサービスを生成する
IX. まとめ

渡辺幸三氏の3要素分析法のプロセス

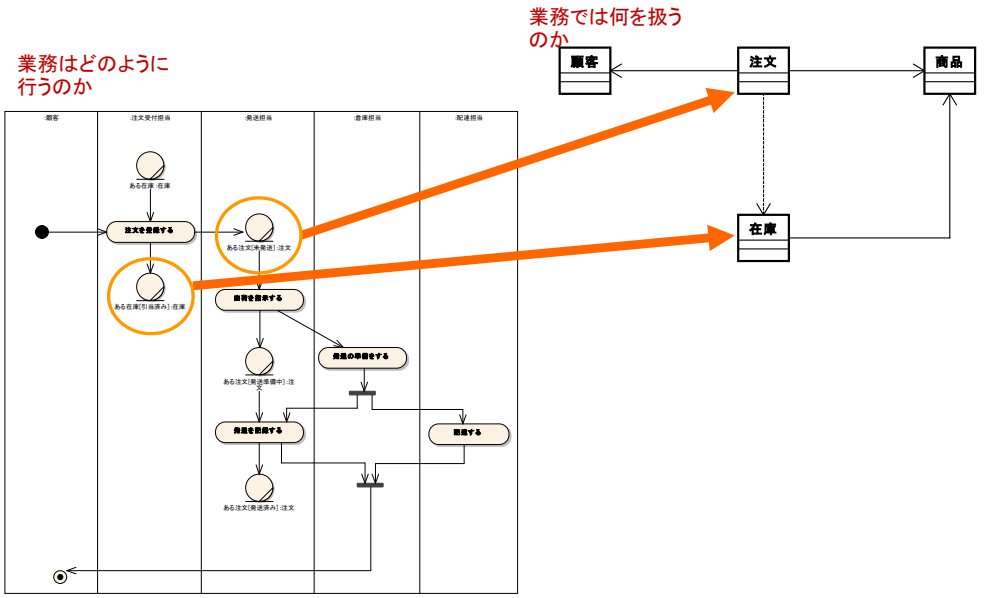


RUPのプロセス(1/2)

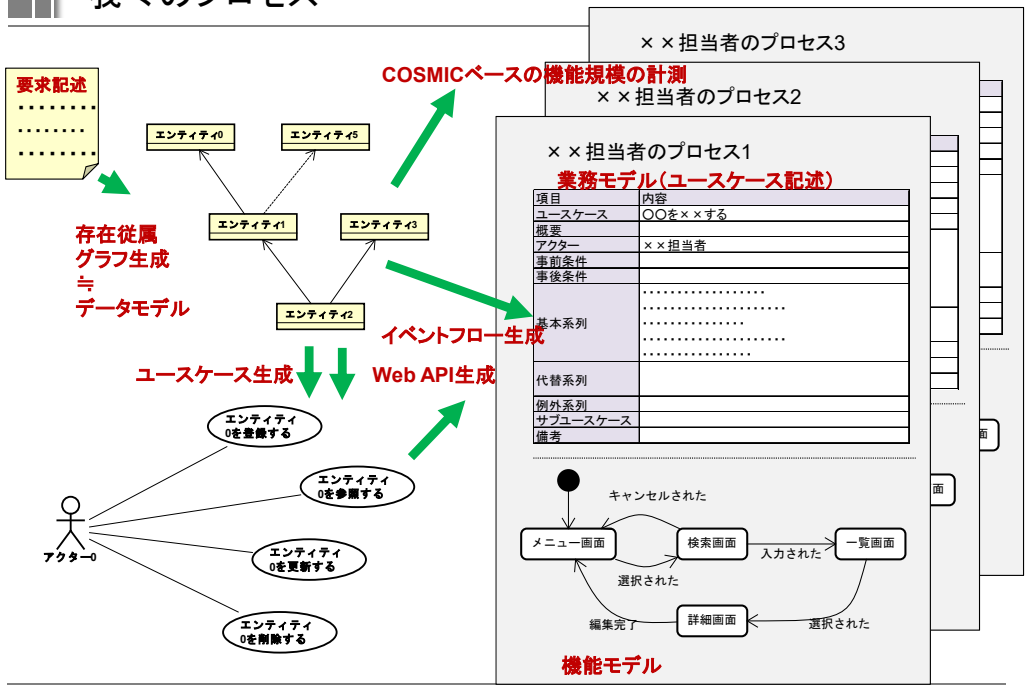
要求記述は新築とリフォームで異な
新築の場合：何をどうしたい
リフォームの場合：現システムの
どこをどうしたい



RUPのプロセス(2/2)



我々のプロセス



発表概要

I. 自己紹介，問題意識など
II. オブジェクト指向モデリング指導の難しさ
III. エンティティの存在従属分析
IV. 存在従属分析の例題への適用
V. 存在従属グラフからリレーショナルモデルを構築する
VI. 存在従属グラフからユースケース記述を構築する
VII. 存在従属グラフを中心とした開発プロセス
VIII. 存在従属グラフからWebサービスを生成する
IX. まとめ

サービス指向

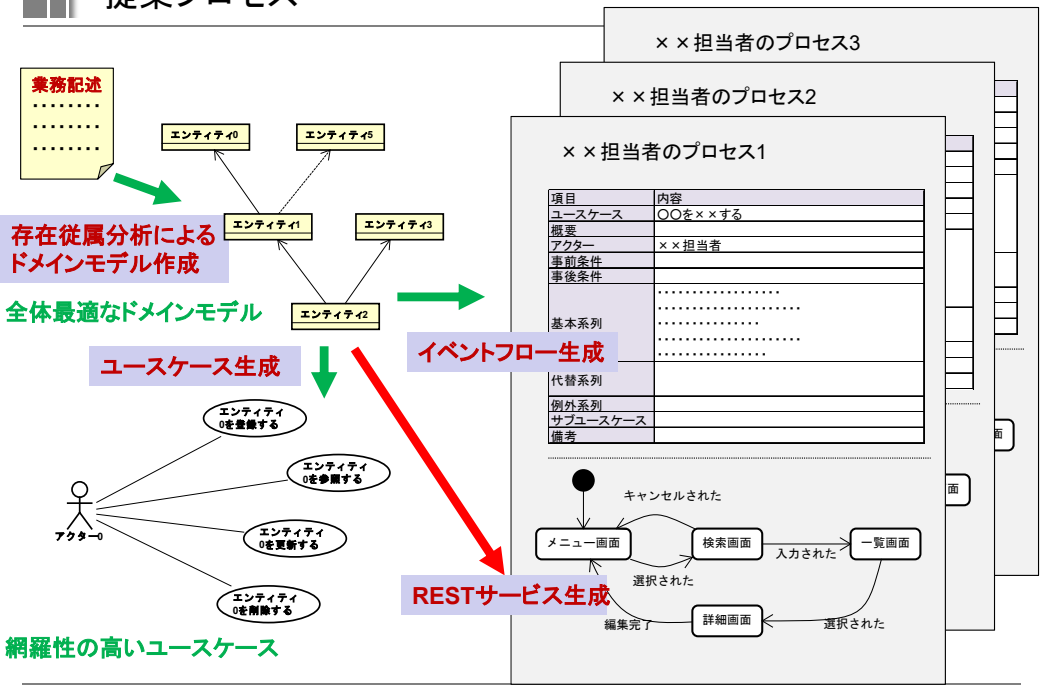
- SOAからMSA（Microservices Architecture）
  - SOAが要求する技術は「大Webサービス」と呼ばれ，SOAPやWSDLなどWS-*と称される一連の仕様群を前提としていた
  - MSAは，Webの文化から生まれた軽量なWeb APIをサービス群の主な実現技術にするため，それらの連携にESBのような重量級のミドルウェア前提ではない
- RESTfulサービス・・・MSAの実現手段の最右翼
  - 世界中のあらゆる機械が互いに通信しあうことを可能にする技術
  - Roy Fieldingが，ウェブについて書いた論文で初めて紹介され，開発者の間で急速に認知され使われるようになった
  - APIの仕様が合えば，サーバおよびクライアントの実装言語を問わず連携可能
  - ただし，APIの設計は自由度が高い
    - URIの命名，HTTPメソッド，パラメータの与え方，結果の受け取り方

RESTの原則と設計レベル

RESTfulアーキテクチャの原則	説明	
		LEVEL0 HTTPの使用
アドレス可能なリソース	リソースは，RESTにおいて情報とデータの主要な抽象である．個々のリソースはURI（Uniform Resource Identifier）を通じてアドレス可能でなければならない	LEVEL1 リソース概念の導入
制約された統一インターフェース	リソースを操作するための明確なメソッドの小さなセットを使用する．URIには「アクション」パラメータを使用せず，Webサービスは POST, GET, PUT, DELETEなどのHTTPのメソッドのみを使用する	LEVEL2 HTTPの動詞の導入
表現指向	サービスとの対話には，そのサービスが提供する表現を使用する．1つのURIによって参照されるリソースは，複数のフォーマット（表現）を持つことができる	
ステートレスな通信	すべてのHTTPリクエストが完全に分離している．クライアントがリクエストを送信する際には，サーバがリスエストを処理するために必要なアプリケーションの状態をすべて含めなければならない	
アプリケーション状態エンジンとしてのハイパーメディア	サーバはリンク付きの表現を返すことで，最新の戻り値の結果に応じてクライアントが次に送信可能なリクエストについての示唆を与えることができる	LEVEL3 HATEOAS概念の導入

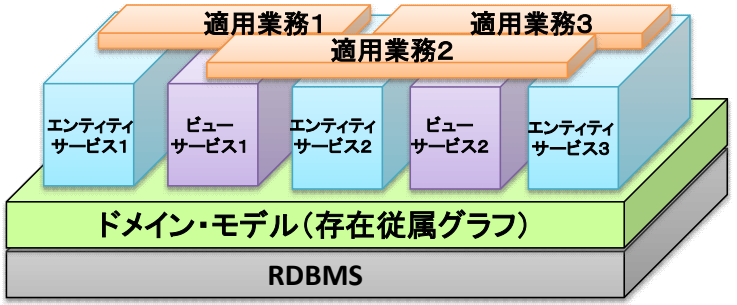
http://martinfowler.com/articles/richardsonMaturityModel.html

提案プロセス



提案アーキテクチャ

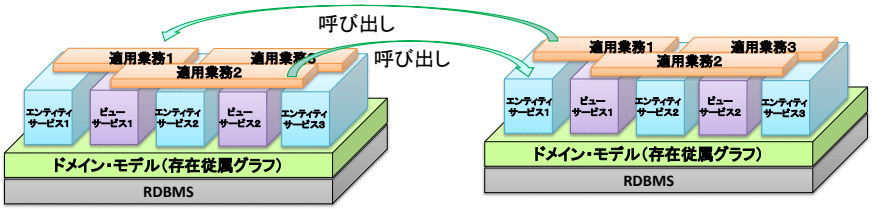
- 存在従属グラフを中心としたアーキテクチャ
- エンティティサービスとビューサービスを自動生成
  - Swaggerの援用によってドキュメントも半自動生成
- 適用業務の変更，ドメインモデルの変更はサービスのレイヤで吸収



存在従属グラフ中心のアーキテクチャ
RDBMSの使用を前提

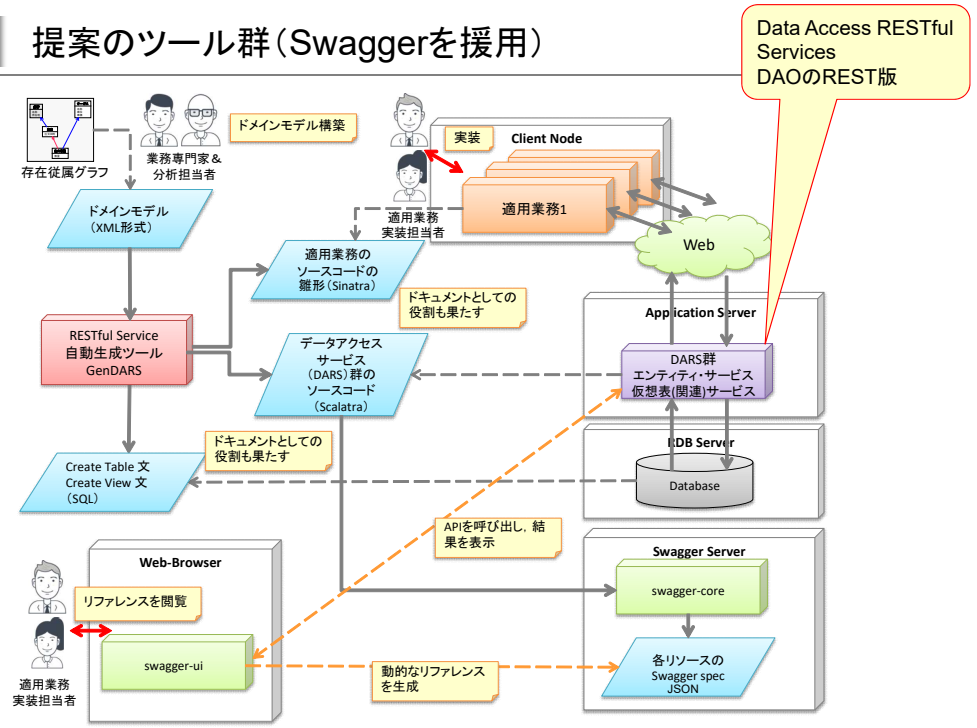
提案アーキテクチャ

- ドメイン間のインタラクションも容易
- APIはドメインモデルをカプセル化している
- 適用業務から他のドメインのサービスを呼び出す
  - 呼び出し方がわからなければ呼び出せないため，APIリファレンスが必須
  - ただし，所詮はWeb越しの関数呼び出しなので適用業務が求めるものを具体的にイメージする必要がある



カプセル化されたドメイン間の連携イメージ

提案のツール群（Swaggerを援用）

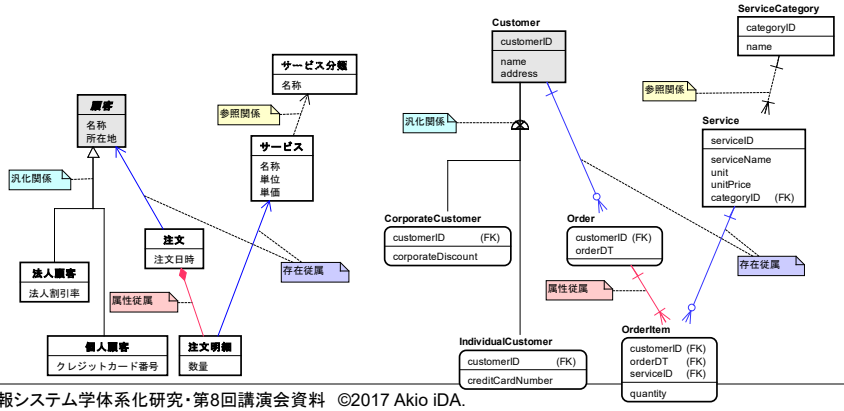


生成が必要なサービスを検討する

- どんなAPIがあれば良いのか？次のグラフについてアプリケーション（適用業務の視点で）検討してみる．

1. 親の生成
2. 親の同定→子の生成，参照，更新，削除
3. 親の削除

- 制約に照らして適切な順序関係



具体例で考える

- 独立エンティティについてのサービス呼び出し例

(1) 顧客が自分自身を個人顧客として登録する場合
①顧客は，適用業務に氏名，住所，クレジットカード番号などの属性値を提示して，自分を個人顧客として登録することを依頼する．
②適用業務は，その処理の中でサービスPOST Customerを呼び出して，顧客インスタンスを新規登録する．入力された属性値群はリクエスト・ボディに格納されて渡される．新規登録した結果，新規のcustomerIDが採番されるため，それを顧客に示す．

■ 従属エンティティについてのサービス呼び出し例

(2) 顧客が商品を注文する場合
①顧客は顧客登録時に示されたcustomerIDを適用業務に入力する.
②適用業務はGET　Customer／{customerID}を呼び出して，顧客の属性値を取得する．そして，それらの属性値に問題がなければ，PUT　Order／{customerID}＋{orderDT}を呼び出して，新規注文（見出し）を生成する．ここで，orderDTのDTはDateTimeの略で注文時点のタイムスタンプ値であるとする． UNIXエポックからの経過ミリ秒数
③顧客は注文可能な商品の一覧を適用業務に要求する.
④適用業務はGET　Productを呼び出して，提供可能な商品の一覧を取得し，顧客に示す.
⑤顧客は利用したい商品をいくつか選択して適用業務にそれらの注文を依頼する
⑥適用業務は選択された商品の数だけ，PUT　OrderItem／{customerID}＋{orderDT}＋{productID}を呼び出して，新規の注文に属性従属させる注文明細を生成する.

■ 従属エンティティについてのサービス呼び出し例

(3) 顧客が注文を変更する場合
①顧客は，適用業務にcustomerIDを入力して，自分が発した注文を照会する.
②適用業務は，GET　OrderCutomer／{customerID}を呼び出して，当該顧客の注文一覧を取得し，顧客に示す．なお，OrderCustomerはエンティティではなく関連（仮想表）に対するDARSである.
③顧客は，変更したい注文を1つ選択して適用業務に注文の変更を依頼する.
④適用業務は，GET　Orderproduct／{customerID}＋{orderDT}を呼び出して，注文明細（注文された商品の名称と数量および料金）の一覧を取得し，顧客に示す.
⑤顧客はここで，注文明細の内容を変更，または，注文明細の追加，または，注文明細の削除を行うことができる.
(ア) ある注文明細の数量を変更する場合
適用業務は，PUT　OrderItem／{customerID}＋{orderDT}＋{productID}を呼び出して，当該明細を更新する
(イ) ある注文明細の商品を変更する場合
適用業務は，DELETE　OrderItem／{customerID}＋{orderDT}＋{productID}を呼び出して，当該明細を削除するとともに，PUT　OrderItem／{customerID}＋{orderDT}＋{productID}を呼び出して，変更後の商品IDの明細を追加する
(ウ) 新規に注文明細を追加する場合
適用業務はPUT　OrderItem／{customerID}＋{orderDT}＋{productID}を呼び出して，変更後の商品の明細を追加する
(エ) ある注文明細を削除する場合
適用業務は，DELETE　OrderItem／{customerID}＋{orderDT}＋{productID}を呼び出して，当該明細を削除する

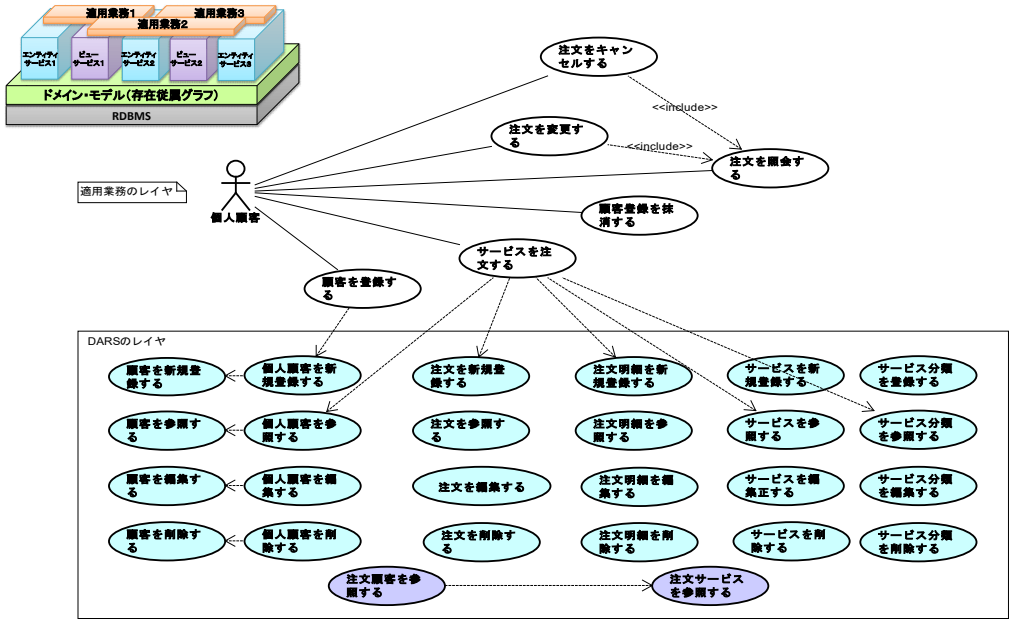
■ 独立エンティティのインスタンスを削除する

(5) 顧客が自分自身の顧客登録を抹消する場合
①顧客は，適用業務にcustomerIDを入力して，自分の顧客登録の抹消を依頼する.
②適用業務は，DELETE　Customer／{customerID}を呼び出して，顧客登録の抹消を結果を提示する．ただし，注文を持っている顧客はその注文の決済が完了するまで，自分の顧客登録を抹消することはできない.

■ 関連に関するサービスを呼び出す例

(4) 顧客が注文をキャンセルする場合
①顧客は，適用業務にcustomerIDを入力して，自分が発した注文を照会する.
②適用業務は，GET　OrderCutomer／{customerID}を呼び出して，当該顧客の注文一覧を取得し，顧客に示す．なお，OrderCustomerはエンティティではなく関連（仮想表）についてのDARSである.
③顧客は，キャンセルしたい注文を1つ選択して適用業務に注文のキャンセルを依頼する.
④適用業務は，DELETE　Order／{customerID}＋{orderDT}を呼び出して，指定された注文を削除する.

具体例から理解できたサービスのレイヤ構造



一般化して考える

■ 独立エンティティのためのサービスの一般形

操作対象	サービスの機能	キー値の採番	HTTP操作	Pathパターン	リクエストのボディ	レスポンスのボディ	レスポンス・ヘッダのステータス	備考
独立エンティティ	インスタンスの新規登録	サーバ側で連番を自動採番	POST	エンティティ名	新規登録されるインスタンスの属性と属性値の表現	新規登録されたインスタンスの属性と属性値の表現	201 Created 415 Unsupported Media Type 404 Not Found	この操作は安全でもべき等でもない
	特定のインスタンスの照会	なし	GET	エンティティ名/{id}	空	該当したインスタンスの属性と属性値の表現	200 OK 204 No Content 404 Not Found	照会の操作はすべて安全かつべき等である 存在しない{id}が指定された場合はエラーとなる
	特定のインスタンスの編集	なし	PUT	エンティティ名/{id}	編集するインスタンスの属性と属性値の表現ただし、キー属性とその値はなくてもよいし、あっても無視される	編集されたインスタンスの属性と属性値の表現ただし、キー属性とその値はなくてもよいし、あっても無視される	200 OK 204 No Content 415 Unsupported Media Type 404 Not Found	編集の操作はすべて安全ではないが、べき等である 存在しない{id}が指定された場合はエラーとなる
	特定のインスタンスの削除	なし	DELETE	エンティティ名/{id}	空	削除されたインスタンスの属性と属性値の表現	204 No Content 500 Internal Server Error 404 Not Found	削除の操作はすべて安全ではないが、べき等である 存在しない{id}が指定された場合、または、そのインスタンスを参照しているオブジェクトが存在する場合はエラーとなる

一般化して考える

■ 独立・従属エンティティ共通のサービスの一般形

操作対象	サービスの機能	キー値の採番	HTTP操作	Pathパターン	リクエストのボディ	レスポンスのボディ	レスポンス・ヘッダのステータス	備考
独立・従属エンティティ共通	指定範囲のインスタンスの照会	なし	GET	エンティティ名/{from}/{to}	空	該当したすべてのインスタンスの属性と属性値の表現の一覧	200 OK 404 Not Found	最初のインスタンスの位置を0番目とし、fromで指定された位置のインスタンスからtoで指定された位置のインスタンスまでが含まれる
	すべてのインスタンスの照会	なし	GET	エンティティ名/	空	該当したすべてのインスタンスの属性と属性値の表現の一覧	200 OK 404 Not Found	照会の操作はすべて安全かつべき等である
	インスタンスの件数照会	なし	GET	エンティティ名/count	空	件数	200 OK 404 Not Found	件数を照会する操作はすべて安全かつべき等である
	すべてのインスタンスの削除	なし	DELETE	エンティティ名/				あえて実装しない

一般化して考える

■ 従属エンティティについてのサービスの一般形

操作対象	サービスの機能	キー値の採番	HTTP操作	Pathパターン	リクエストのボディ	レスポンスのボディ	レスポンス・ヘッダのステータス	備考
従属エンティティ	インスタンスの新規登録	クライアント側で既知のためクライアント側が指定	PUT	エンティティ名/{id1}+{id2}+{id3}+…+{idn}	新規登録されるインスタンスの属性値の表現	新規登録されたインスタンスの属性と属性値の表現	201 Created 415 Unsupported Media Type 404 Not Found	この操作は安全ではないがべき等である
	特定のインスタンスの照会	なし	GET	エンティティ名/{id1}+{id2}+{id3}+…+{idn}	空	該当したすべてのインスタンスの属性と属性値の表現の一覧	200 OK 204 No Content 404 Not Found	照会の操作はすべて安全かつべき等である
	特定の親に従属するインスタンスの照会	なし	GET	エンティティ名/{id1}+{id2}+{id3}+…+{id(n-1)}	空	該当したすべてのインスタンスの属性と属性値の表現の一覧	200 OK 204 No Content 404 Not Found	複合識別子の属性がn個の場合、識別子は最低1個からn−1個までを+記号で連結して指定できる。途中の識別子の省略はできない
	指定範囲のインスタンスの照会	なし	GET	エンティティ名/{from}/{to}	空	該当したすべてのインスタンスの属性と属性値の表現の一覧	200 OK 404 Not Found	先頭を0番目としてfrom番目からto番目のインスタンスの一覧を照会

関連のためのサービス

■ 関連(仮想表)についてのサービスの一般形

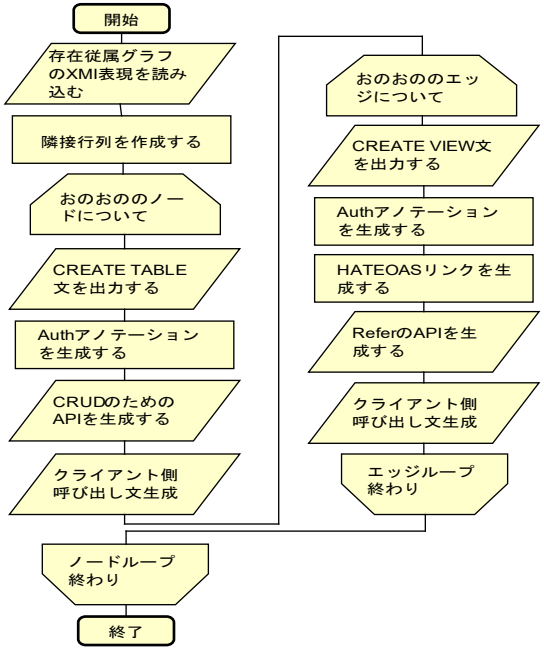
操作対象	サービスの機能	キー値の採番	HTTP操作	Pathパターン	リクエストのボディ	レスポンスのボディ	レスポンス・ヘッダのステータス	備考
仮想表(ビュー)	特定のインスタンスの照会	なし	GET	仮想表名/{id1}+{id2}+{id3}+…+{idn}	なし	該当したインスタンスの属性と属性値の表現	200 OK 204 No Content 404 Not Found	仮想表の名称は隣接行列のエンティティ名をfrom, toの順に結合したもの
	特定の親に従属するインスタンスの照会	なし	GET	仮想表名/{id1}+{id2}+{id3}+…+{id(n-m)}	なし	該当したすべてのインスタンスの属性と属性値の表現の一覧	200 OK 204 No Content 404 Not Found	複合識別子の属性がn個の場合，識別子は最低1個からn-1個までを+記号で連結して指定できる．途中の識別子の省略はできない
	指定範囲のインスタンスの照会	なし	GET	仮想表名/{from}/{to}	なし	該当したすべてのインスタンスの属性と属性値の表現の一覧	200 OK 404 Not Found	最初のインスタンスの位置を0番目とし，fromで指定された位置のインスタンスからtoで指定された位置のインスタンスまでが含まれる
	すべてのインスタンスの照会	なし	GET	仮想表名/	なし	該当したすべてのインスタンスの属性と属性値の表現の一覧	200 OK 404 Not Found	照会の操作はすべて安全かつべき等である
	インスタンスの件数照会	なし	GET	仮想表名/count	なし	件数	200 OK 404 Not Found	照会の操作はすべて安全かつべき等である

仮想表およびそれに付随するサービスを生成する目的は，
オブジェクトモデルとリレーショナルモデルのギャップを埋めることと，
派生属性値の計算(たとえば数量に単価を掛けて金額を求めるなど)
の小さなビジネスロジックをここに持たせたうえで，適用業務が要求するビューを
効率的に構築するためのいわゆる正規化崩しの作業を不要としたいがためである．

ここまでの小括

- 適用業務から他のドメインのサービスを呼び出す
- 呼び出し方がわからなければ呼び出せないため，APIリファレンスが必須
- ただし，所詮はWeb越しの関数呼び出しなので適用業務が求めるものを具体例で想定し，その後APIを一般化して整理した
  - 独立エンティティについて4タイプ(CRUD)
  - 独立・従属エンティティ共通で3タイプ(Rのバリエーション)
  - 従属エンティティについて4タイプ(CUとRのバリエーション)
  - 関連について5タイプ(Rのバリエーション)

生成のアルゴリズム



入力：存在従属グラフのXMI表現例

```
<?xml version="1.0" encoding="UTF-8"?>
<xmi:XMI xmlns:uml="http://www.omg.org/spec/UML/20110701" xmlns:xmi="http://www.omg.org/spec/XMI/20110701">
  <uml:Model name="sampleStore" xmi:id="_UeoaUKCFEeax5e04-MYM_w">
    <packagedElement name="Customer" xmi:id="_UeoaUaCFEeax5e04-MYM_w" xmi:type="uml:Class">
      <ownedAttribute name="id" type="_UeoabqCFEeax5e04-MYM_w" visibility="private" xmi:id="_UeoaUqCFEeax5e04-MYM_w" xmi:type="uml:Property"/>
      <ownedAttribute name="customername" type="_Ueoab6CFEeax5e04-MYM_w" visibility="private" xmi:id="_UeoaU6CFEeax5e04-MYM_w" xmi:type="uml:Property"/>
      <ownedAttribute name="email" type="_Ueoab6CFEeax5e04-MYM_w" visibility="private" xmi:id="_UeoaVKCFEeax5e04-MYM_w" xmi:type="uml:Property"/>
      <ownedAttribute name="phone" type="_Ueoab6CFEeax5e04-MYM_w" visibility="private" xmi:id="_UeoaVaCFEeax5e04-MYM_w" xmi:type="uml:Property"/>
      <ownedAttribute name="password" type="_Ueoab6CFEeax5e04-MYM_w" visibility="private" xmi:id="_UeoaVqCFEeax5e04-MYM_w" xmi:type="uml:Property"/>
      <ownedAttribute association="_UeoabKCFEeax5e04-MYM_w" name="" type="_UeoaYaCFEeax5e04-MYM_w" visibility="private" xmi:id="_UeoaV6CFEeax5e04-MYM_w" xmi:type="uml:Property"/>
    </packagedElement>
    <packagedElement name="Category" xmi:id="_UeoaWKCFEeax5e04-MYM_w" xmi:type="uml:Class">
      <ownedAttribute name="id" type="_UeoabqCFEeax5e04-MYM_w" visibility="private" xmi:id="_UeoaWaCFEeax5e04-MYM_w" xmi:type="uml:Property"/>
      <ownedAttribute name="name" type="_UeoabqCFEeax5e04-MYM_w" visibility="private" xmi:id="_UeoaWqCFEeax5e04-MYM_w" xmi:type="uml:Property"/>
    </packagedElement>
    <packagedElement clientDependency="_UeoaaqCFEeax5e04-MYM_w" name="Product" xmi:id="_UeoaW6CFEeax5e04-MYM_w" xmi:type="uml:Class">
      <ownedAttribute name="id" type="_UeoabqCFEeax5e04-MYM_w" visibility="private" xmi:id="_UeoaXKCFEeax5e04-MYM_w" xmi:type="uml:Property"/>
      <ownedAttribu
      <ownedAttribu
      <ownedAttribu                                                                                                    xmi:type="uml:Property"/>
    </packagedEleme
    <packagedElemer
    <packagedElemer
      <ownedAttribu
      <ownedAttribu
      <ownedAttribute name="orderDate" type="_UeoabqCFEeax5e04-MYM_w" visibility="private" xmi:id="_UeoaZKCFEeax5e04-MYM_w" xmi:type="uml:Property"/>
      <ownedAttribute name="quantity" type="_UeoabqCFEeax5e04-MYM_w" visibility="private" xmi:id="_UeoaZaCFEeax5e04-MYM_w" xmi:type="uml:Property"/>
      <ownedAttribute name="status" type="_UeoabqCFEeax5e04-MYM_w" visibility="private" xmi:id="_UeoaZqCFEeax5e04-MYM_w" xmi:type="uml:Property"/>
      <ownedAttribute name="complete" type="_UeoaCKCFEeax5e04-MYM_w" visibility="private" xmi:id="_UeoaZ6CFEeax5e04-MYM_w" xmi:type="uml:Property"/>
      <ownedAttribute association="_UeoaaCFEeax5e04-MYM_w" name="" type="_UeoaW6CFEeax5e04-MYM_w" visibility="private" xmi:id="_UeoaaKCFEeax5e04-MYM_w" xmi:type="uml:Property"/>
      <ownedAttribute association="_UeoabKCFEeax5e04-MYM_w" name="" type="_UeoaUaCFEeax5e04-MYM_w" visibility="private" xmi:id="_UeoaaaCFEeax5e04-MYM_w" xmi:type="uml:Property"/>
    </packagedElement>
    <packagedElement client="_UeoaW6CFEeax5e04-MYM_w" supplier="_UeoaWKCFEeax5e04-MYM_w" xmi:id="_UeoaaqCFEeax5e04-MYM_w" xmi:type="uml:Dependency"/>
    <packagedElement memberEnd="_UeoaX6CFEeax5e04-MYM_w _UeoaaKCFEeax5e04-MYM_w" name="" xmi:id="_Ueoaa6CFEeax5e04-MYM_w" xmi:type="uml:Association"/>
    <packagedElement memberEnd="_UeoaV6CFEeax5e04-MYM_w _UeoaaaCFEeax5e04-MYM_w" name="" xmi:id="_UeoabKCFEeax5e04-MYM_w" xmi:type="uml:Association"/>
    <packagedElement name="PrimitiveTypes" xmi:id="_UeoabaCFEeax5e04-MYM_w" xmi:type="uml:Package">
      <packagedElement name="int" xmi:id="_UeoabqCFEeax5e04-MYM_w" xmi:type="uml:PrimitiveType"/>
      <packagedElement name="string" xmi:id="_Ueoab6CFEeax5e04-MYM_w" xmi:type="uml:PrimitiveType"/>
      <packagedElement name="boolean" xmi:id="_UeoacKCFEeax5e04-MYM_w" xmi:type="uml:PrimitiveType"/>
    </packagedElement>
  </uml:Model>
</xmi:XMI>
```

Astah XMI Export Plugin使用

クラス図のモデル情報をXMI形式のファイルにエクスポートするプラグイン

出力：Scalatraのコード（サーバ側API実装）

- ScalatraはVer.2.2からSwaggerをサポートするためのライブラリである scalatra-swaggerを提供している
- 最新版は2.5.0（2017/2/21現在）
- REST APIであれば，どの言語/フレームワークで実装されていてもSwaggerのクライアントを利用することができる

Scalatraのコードのイメージ

- ルーティングはHTTPメソッドとURLパターンをペアで記述する
- リクエストに最初にマッチしたルーティングのブロックが実行される

```
get '/Entity' do
  .. インスタンスをありったけ参照する ..
end

get '/Entity/:id' do
  .. インスタンスを高々1件参照する ..
end

post '/Entity' do
  .. インスタンスを生成する ..
end

put '/Entity/:id' do
  .. インスタンスの値を更新する ..
end

delete '/Entyty/:id' do
  .. インスタンスを削除する ..
end
```


Scalatraのコードの具体例

```
import org.scalatra._
import org.scalatra.json._
import org.json4s._

class SwaggerTestController extends ScalatraServlet with JacksonJsonSupport {

  protected implicit val jsonFormats: Formats = DefaultFormats
  private[this] val productRepo = new ProductRepository

  before() {
    contentType = formats("json")
  }

  get("/product") {
    params.getAs[String]("name") match {
      case Some(name) => productRepo getProductByName name
      case None => productRepo.allProducts
    }
  }

  post("/product") {
    parsedBody.extractOpt[Product] match {
      case Some(product) =>
        productRepo addProduct product
        Ok()
      case _ => halt(400, "invalid params")
    }
  }

  notFound {
    halt(404)
  }
}
```

（リクエストの）HTTP メソッド（“リソースのURI”） {
    応答するレスポンスの値を作る処理
}

生成の仕組みの概要

- 隣接行列を生成する

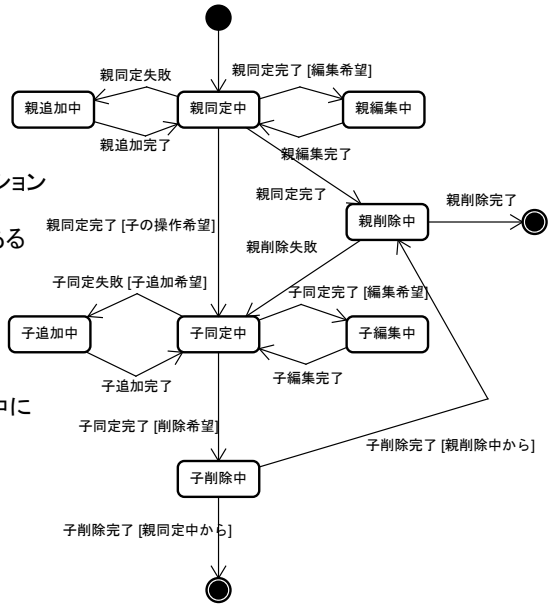
to / form	Customer	Corporate Customer	Individual Customer	Order	OrderItem	Service	Service Category
Customer				↓			
Corporate Customer	汎化						
Individual Customer	汎化						
Order	存在従属↑						
OrderItem				属性従属		存在従属↑	
Service					↓		参照↑
Service Category						↓	

- scalatraのテンプレートコードに対して文字列置換を実行する
  - クラスの数だけクラス名を置換してクラスを生成する
  - 存在従属と参照のある部分には仮想表の定義とHATEOASリンクを生成
  - 属性の数だけ属性名および型の生成
  - POST，GET，POST，DELETEに対する応答操作の置換

HATEOAS（Hypermedia As The Engine Of Application State）の生成

RESTの原則によれば，
APIはクライアントに対して呼び出しのナビゲーション
インタラクションのための
ハイパーメディア・コントロールを提供すべきである

具体的には，今のリクエストの結果に応じて
次に呼び出し可能なリンク情報をレスポンスの中に
埋め込むことで実現される



親は従属先のクラスのインスタンス，子は従属クラスのインスタンス

Swaggerの援用

- Swaggerの存在を知る
- Swaggerは魅力的に感じられる
  - Open API InitiativeにはGoogle, Microsoft, IBMなどが参画
  - 見た目が綺麗なリファレンス
  - 実行できるリファレンス
  - チームのメンバに喜んで使ってもらえそう
- しかしながら，Swagger definitionを手作業で編集するのはしんどい
  - 実装コードを書いた方が楽かも知れない
- 自動生成すればよいのでは？
  - テンプレートに対する少し高度な文字列置換でなんとか

Swaggerの概要

- 人間が読みやすく機械が実行可能なドキュメントを提供するプロジェクト
  - Longmanの現代英英辞典のようにカラフルなドキュメントUI
- 4つの主なツールで構成

The major core tools are:

Tool	Description
Swagger Core	Java-related libraries for generating and reading Swagger definitions
Swagger Codegen	Command-line tool for generating both client and server side code from a Swagger definition
Swagger UI	Browser based UI for exploring a Swagger defined API
Swagger Editor	Browser based editor for authoring Swagger definitions in YAML or JSON format

Swaggerの4つの主なツール

- swagger-core
  - REST APIの実装からSwagger definitionを生成するためのライブラリ
- swagger-ui
  - Swagger definitionを参照して，それを基にHTMLリファレンスを生成するWebクライアントアプリケーション
- swagger-editor
  - Swagger definitionを編集るためのエディタ
  - ブラウザ上で動作
- swagger-codegen
  - Swagger definitionからクライアント側，サーバ側両方のコードを生成するコマンドラインツール

人間可読で実行可能

- Swagger-ui

user : Operations about user — Show/Hide | List Operations | Expand Operations

POST	/user	Create user
POST	/user/createWithArray	Creates list of users with given input array
POST	/user/createWithList	Creates list of users with given input array
GET	/user/login	Logs user into the system
GET	/user/logout	Logs out current logged in user session
DELETE	/user/{username}	Delete user
GET	/user/{username}	Get user by user name
PUT	/user/{username}	Updated user

展開したり，折り畳んだり

user : Operations about user — Show/Hide | List Operations | Expand Operations

POST /user — Create user

Implementation Notes

This can only be done by the logged in user.

Parameters

Parameter	Value	Description	Parameter Type	Data Type
body	(required)	Created user object	body	Model / Example Value

Parameter content type: application/json

```
{
  "id": 0,
  "username": "string",
  "firstName": "string",
  "lastName": "string",
  "email": "string",
  "password": "string",
  "phone": "string",
  "userStatus": 0
}
```

Response Messages

HTTP Status Code	Reason	Response Model	Headers
default	successful operation		

Try it out!

実行してレスポンスを確認することができる

POST /user/createWithArray — Creates list of users with given input array

Swagger editor

definitionはYAML形式で編集

右側にプレビューが表示される

File ▾ Preferences ▾ Generate Server ▾ Generate Client ▾ Help ▾ — ✔ Processed with no error

```
  license:
    name: Apache 2.0
    url: http://www.apache.org/licenses/LICENSE-2.0.html
host: samplestore.swagger.io
basePath: /v2
schemes:
  - http
paths:
  /products:
    post:
      tags:
        - product
      summary: Add a new product to the store
      description: ""
      operationId: addProduct
      consumes:
        - application/json
        - application/xml
      produces:
        - application/json
        - application/xml
      parameters:
        - in: body
          name: body
          description: Product object that needs to be added to
            the store
          required: false
          schema:
            $ref: "#/definitions/Product"
      responses:
        "405":
          description: Invalid input
      security:
        - samplestore_auth:
          - write_products
          - read_products
    put:
      tags:
```

/products

POST /products — product

Summary: Add a new product to the store

Parameters

Name	Located in	Description	Required	Schema
body	body	Product object that needs to be added to the store	No	▼Product { id: integer, category: ▶Category { }, name: string *, status: ▶ string }

Responses

Code	Description
405	Invalid input

Security

Security Schema	Scopes
samplestore_auth	write_products read_products

Try this operation

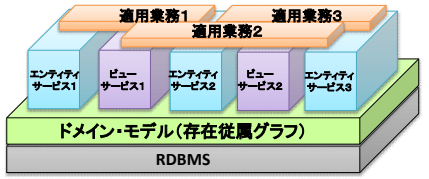
PUT /products — product

Summary: Update an existing product

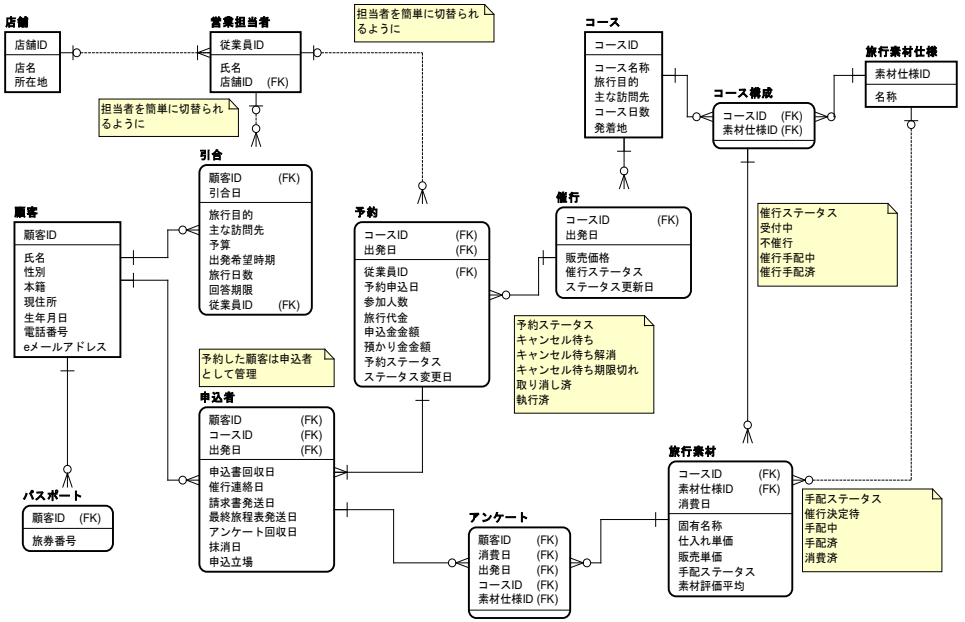
Parameters

実際のプロジェクトで使ってみた結果

- 旅行会社の販売支援システムの一部に適用（次頁にモデル）
  - 13のエンティティに対して103のエンティティサービスを生成
  - 16の関連に対して77のビューサービスを生成
- サービス群は
  - ドメインモデルの変更のバッファとして機能する
    - ドメインモデルの仕様変更
    - 適用業務の仕様変更
- 存在従属グラフの構築時の心的負担も軽減
- 関連（ビュー）サービスの考え方はオリジナリティが高い



旅行会社のパッケージツアー販売支援ドメインの存在従属グラフのERD表現



提案手法と従来手法の定性的比較

開発方法論	ドメイン分割とドメインモデルの構築	WebサービスAPI設計のガイドライン	Webサービス自動生成	WebサービスAPI利用のための設計ドキュメント
アジャイル開発(たとえばスクラム)＋NetBeans サーバ側の実装言語はJavaでWebサービスはJSR 339: JAX-RS 2.0: The Java API for RESTful Web Servicesの仕様に準拠する	個別に実施する. スモールリリースが前提のためリリース対象についてアクターの識別, ユースケースの識別, ユースケース毎のユースケース記述の作成, ユースケース記述からのクラスと関連の識別, ユースケース記述からのWebサービスAPIの抽出が行われる. 局所最適化されたクラス図とWebサービスAPIだけが識別される恐れがある.	ガイドラインが存在しないため, 命名規則も粒度も統一感のないWebサービスAPIが定義される恐れがある.	エンティティに対するCRUD操作のみ. あとは手作業で追加していく. ここでもガイドラインが存在しないため命名規則も粒度も統一感のないWebサービスAPIが追加されていく恐れがある.	手動で用意する必要があるため, 設計ドキュメントと実装成果物の内容が乖離する恐れあり.
エンタープライズアジャイル開発(DAD)＋NetBeans+提案手法 サーバ側の実装言語はJavaでWebサービスはJSR 339: JAX-RS 2.0: The Java API for RESTful Web Servicesの仕様に準拠する	方向付けフェーズにて実施. 業務要求記述から先に存在従属のクラス図(存在従属グラフ)を作成し, そこから導かれるノードとエッジに関するWebサービスAPIを一気に自動生成. 存在従属グラフの構築には, 業務が扱う概念のインスタンスの存在期間の依存関係に着目して行う. またWebサービスAPIの生成は提案ツールを用いて自動的に行うことができる.	提案手法で定義した次のガイドラインが存在する: ・アクセス権についてのガイドライン ・HTTPメソッドの使用についてのガイドライン ・エンティティ・サービス設計のガイドライン ・ビュー・サービス設計のガイドライン ・既存のサービスに新たなサービスを追加するためのガイドライン ・既存のサービスの変更と廃止についてのガイドライン ・HATEOASリンクの生成に関するガイドライン	エンティティに対するCRUD(新規作成, 参照, 更新, 削除)操作とエンティティ間の関係に対するR(参照)操作を自動生成. 足りないものは手作業で追加していくが, WebサービスAPI設計のガイドラインが存在するため, 命名規則や粒度に統一感のないWebサービスAPIが追加されていくリスクが少ない.	クライアント側のコード見本がサーバ側API生成時に自動生成されるためリファレンスマニュアルとしての設計ドキュメントと実装成果物の乖離が生じない.

発表概要

I. 自己紹介, 問題意識など
II. オブジェクト指向モデリング指導の難しさ
III. エンティティの存在従属分析
IV. 存在従属分析の例題への適用
V. 存在従属グラフからリレーショナルモデルを構築する
VI. 存在従属グラフからユースケース記述を構築する
VII. 存在従属グラフを中心とした開発プロセス
VIII. 存在従属グラフからWebサービスを生成する
IX. まとめ

筆者らの想いと願い

- 筆者らは, 予てから日本語問題記述からドメインモデルを工学的に導く研究を進めているが, これらの研究に今回の提案を加えることで, さらにドメインモデルが使いやすいものなることを願っている.
- ドメインモデルの構造(存在従属グラフ)だけではなく, ブラウザでAPIを呼び出すと登録されているインスタンス群が表示されるので, ドメインモデルの不備を見つける用途にも使えると思います.
- 提案手法がオブジェクト指向手法において正規化と同等の役割を果たし, ドメインエキスパートとシステムエンジニアが共同で参加するモデリング作業の一助となれば幸いである.

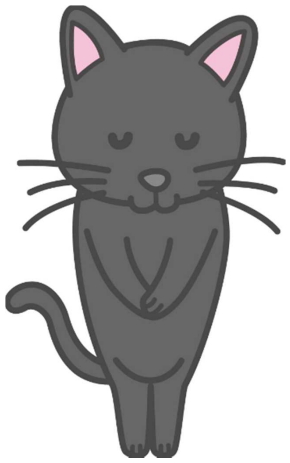
今後の課題

- プラットフォームレベルで
  - 期間リソースをうまく扱えるようにしたい
  - 相互排他性制約と相互包含性制約にうまく対応したい
  - HATEOASリンクを実用的にしたい
  - 若い衆に実装を任せて，人様にご使用いただける程度の完成度になれば，GitHubなどで公開したい

参考文献

- [1] Sam Newman：マイクロサービスアーキテクチャ，オライリージャパン（2016 / 2）
- [2] R. T. Fielding：Architectural Styles and the Design of Network-based Software Architectures, Irvine，CA（2000 / 8）
- [3] S. Ambler, M. Lines：ディシプリンド・アジャイル・デリバリー - エンタープライズ・アジャイル実践ガイド，翔泳社（2013 / 6）
- [4] L. Richardson, S. Ruby：RESTful Webサービス，オライリージャパン（2007 / 9）
- [5] B. Burke：JavaによるRESTfulシステム構築，オライリージャパン（2010 / 8）
- [6] D. Leffingwell：アジャイルソフトウェア要求―チーム，プログラム，企業のためのリーンな要求プラクティス，翔泳社（2014 / 2）
- [7] Think IT：エンタープライズ・アジャイルってなんだろう？ 3つの視点と2つのフレームワーク，https://thinkit.co.jp/story/2014/10/07/5294（2014 / 10）
- [8] Philippe Kruchten：ラショナル統一プロセス入門 第3版，アスキー（2004 / 11）
- [9] J. Crupi, D. Malks：J2EEパターン 第2版，日経BP社（2005 / 5）
- [10] P. Chen：The Entity Relationship Approach to logical Database Design，QED information science, Wellesley（1979 / 1）
- [11] M. Snoeck, G. Dedene：Existence dependency: The key to semantic integrity between structural and behavioral aspects of object types，IEEE Transactions of Software Engineering，24（4），pp 233-251（1998 / 4）.
- [12] 金田重郎，井田明男，酒井孝真，熊谷聡志：日本語仕様文からの概念モデリングガイドライン－行為文と関数従属性に基づくクラス図の作成，電子情報通信学会論文誌 D，Vol.J98-D，No.7，pp.1068-1082（2015 / 7）

参考文献

- [13] 井田明男，金田重郎，熊谷聡志，藤本明莉：存在従属性に着目した論理要件ロバストなドメインモデルの作成－ドメインクラス図をユビキタス言語として用いるために，情報処理学会論文誌，56巻5号 pp 1340-1350（2015 / 5）
- [14] 西沢直木：astah* professional 活用ガイド，翔泳社（2013 / 3）
- [15] 矢野寛将，中西啓太，井田明男，金田重郎：存在従属性に基づくユースケース図の自動生成手法～公営住宅管理システムを例として～，電子情報通信学会，知能ソフトウェア研究会，信学技報115(487), pp 25-30,（2016 / 3）
- [16] The OAuth 2.0 Authorization Framework draft-ietf-oauth-v2-31
- https://tools.ietf.org/html/draft-ietf-oauth-v2-31
- [17] J. Martin, C. McClure：ソフトウェア構造化技法 - ダイアグラム法による，近代科学社（1986 / 11）
- [18] C. Date：データベース実践講義－エンジニアのためのリレーショナル理論，オライリージャパン（2006/2）
- [19] Swagger - The World's Most Popular Framework for APIs.,
- http://swagger.io/

ご清聴ありがとうございました.