ドメイン・モデリングの勘所

山田正樹/メタボリックス
masaki@metabolics.co.jp
2009.10.29

meta bolics

1

- ドメイン・モデリングとは何か
- ドメイン・モデリングを実践するための五つのポイント

2

ドメイン・モデリングとは何か

3

一言で言うと

現場から始める，一気通貫

モデリング

4

- 「システム」ではなく，「現実世界」 (例えば業務系の場合には業務) そのものをモデリングの最初の対象とする
- 最初のモデルを連続的に変換することによって, 最終製品までのすべての工程を貫く

5

- 例えば, 次のようなものはドメイン・モデルではない
  - 特定の機能の実現を目的とした設計モデル (= システム・モデル)
  - 要求仕様書代わりの要求モデル

6

ドメイン・モデリングの価値

7

- ドメイン・モデリングによって, 業務現場の知恵/知識をシステムの中に残す, 活かす

➡開発だけではなく, テスト, 運用, 保守, 拡張に業務現場の知恵/知識を再利用することができる

8

- 不安定で曖昧で変化しやすい要求や機能ではなく，より安定した現実世界をモデリングの基盤とする

➡より安定して，寿命の長いモデルを構築することができる

9

ドメインとは?

10

ドメイン・モデリング

11

- ドメイン = 領域, 領土
- ソフトウェアの分野では, 問題領域 (problem domain) と解決領域 (solution domain) を考え, ただドメインと言ったら問題領域を指すことが多い
- つまり, ある集団 (ユーザ, 顧客) が関心事とする問題の範囲

12

- モデリング言語としては, UMLでもいいし, オントロジやその他の (半形式的な) モデリング言語, 場合によっては宣言的なプログラミング言語でもよい

- 通常は問題領域固有の用語/概念を用いて, モデリングを始める

13

ドメイン・モデリングの参考書

14

- “**Domain-Driven Design:** Tackling Complexity in the Heart of Software” by Eric Evans, Addison-Wesley, 2004
  - Domain-Driven Design Quickly 日本語版
    - http://www.infoq.com/jp/minibooks/domain-driven-design-quickly
  - DDD難民に捧げるDomain-Driven Design のエッセンス
    - http://www.ogis-ri.co.jp/otc/hiroba/technical/DDDEssence/chap[1-3].html

15

- “**Object Design** - Roles, Responsibilities, and Collaborations” by Rebecca Wirfs-Brock and Alan McKean, Addison-Wesley, 2003
  - 日本語訳「オブジェクト・デザイン」

16

- “**Model Driven Architecture**: Applying MDA to Enterprise Computing”, by David. S. Frankel, Wiley & Sons, 2003
  - 日本語訳「MDA モデル駆動アーキテクチャ」

17

- 「ドメイン分析・モデリング - これからのソフトウェア開発・再利用 基幹技術」, 伊藤他, 共立出版, 1996
- 「システム仕様の分析学 - ソフトシステム方法論」, Brian Wilson, 共立出版, 1996
- 「コンバージェント・エンジニアリング入門」, デビッド・A・テイラー, トッパン, 1996

18

ドメイン・モデリングの五つのポイント

19

1. 共通言語としてのドメイン・モデル

2. ゆるいモデル駆動開発

3. 抽象化を忘れない

4. 汎用領域をメタモデル化する

5. 横断的関心事を抽出する

20

共通言語としての
ドメイン・モデリング

21

要点

ドメイン・モデルは, 全プロジェクト参加者, 特にユーザや顧客を含めて, 共通な認識をもつための土台である

したがって, 全プロジェクト参加者がドメイン・モデルを理解できるような工夫が必要である

22

そのために

勝手に造語したり, 既存の言葉に当てはめたりせず, まずドメインの言葉でモデリングを始める

モデルの変換に伴う言葉の変化を追跡できるようにする

- UMLでtrace abstraction, あるいはExcel表など

23

ドメインの言葉には

- ぶれ - 同じ言葉が違う意味に使われる
- ばらつき - 違う言葉が同じ意味に使われる
- 矛盾 - 言葉遣いが論理的に正確でない

などがある. これらは気付いたら記録していく

最初からすべてを集めようとする必要はない

業務改善や正しいモデルへのヒントになることが多い

24

顧客
やユーザがなかなか本気で
見て, 文句を言ってくれない
のも事実...

初期のモデルは, 顧客やユーザとレビューする

シンプルであっても正確なモデルが必要

見るだけで理解が難しいモデルは, 顧客やユーザの目の前で実際に動かしてチェックする

モデルを動かせる環境が必要

その場で変更できればなおよし

25

ソフトウェアは, 最終的にはすべて「言語」の問題である

26

ゆるいモデル駆動開発

27

要点

ドメイン・モデリングにモデル駆動開発は重要な役割を果たすが, あまりに形式的なMDDはドメインの知識を失ったり, 共通の理解を得られない可能性がある

ドメイン・モデリングには「ゆるめ」のMDDが望ましい

28

そのために

高価で難解で高度な技術の必要なMDDツールではなく, できるだけ初期のモデルでもそれなりに, 簡単に, 素直に動くツールを用意する

必要に応じて, UMLと限定せず, 多言語の混成 (polygrot) を許す

29

クラス図

Ruby on RailsやGrailsなどのスカッフォールドの仕組み

Naked Objectパターン

http://www.nakedobjects.org/ ほか

オントロジ・ツール

Protégé http://protege.stanford.edu/

ほか

30

アクティビティ図, 状態機械図など

- ワークフロー・エンジンやビジネス・プロセス・エンジン
  - 例 http://labs.jboss.com/jbossjbpm/
  - 例 http://www.jboss.org/drools/
- 状態機械エンジン
  - 例 http://smc.sourceforge.net/

31

ユーザ・インタフェイス・プロトタイピング・ツール

スクリプト系の動的なプログラミング言語とDSL (ドメイン固有言語) を組み合わせれば, 問題ごとに自作してもそれほど手間ではない

32

抽象化を忘れない

33

要点

ドメイン・モデリングは, 技術に偏向する可能性からは免れやすいが, 逆にドメインべったりのモデリングになりやすい

システム・モデリングと同様に, ドメイン・モデルも適度な抽象化が必要である

同時に抽象化の過程も追跡可能とすべきである

34

そのために

最初から抽象的なモデルを作るべきではない

as-isのモデルをある程度作り上げて, それを関係者がコミットしてから, リファクタリングを行う

モデルの版管理や追跡のリンクを行う

35

ここで行うリファクタリングは, 技術のための最適化や実現上の都合に対する適合ではなく, 概念上の抽象化でなければならない

技術の言葉が紛れ込んできたら, 間違った抽象化を行っている可能性がある

36

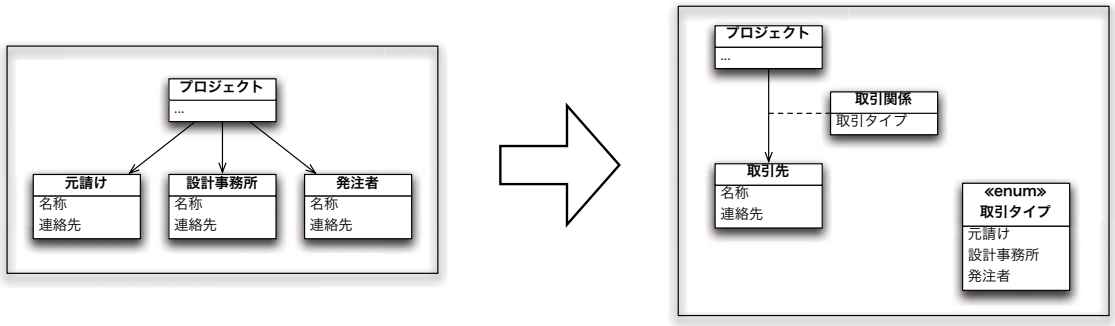
抽象化の視点は複数あり得る

関係者には, 何のための抽象化か, 抽象化によって何が得られるかを説明しなければならない

関係者の合意が得られなければ, 過度の抽象化か, 間違った視点からの抽象化が行われている恐れがある

37

例



38

説明

* ドメイン・モデルでも，「ドメインそのまま」でなく，抽象化を行ってよい

39

利益

* 抽象化によって，ドメイン・モデルの見通しをよくできる
* ドメインそのものに，抽象化による新たな価値を付与することができることもある

40

文脈

- モデルとして, 冗長であったり, より一般化が可能であったり, 抽象化によって見通しをよくすることができる場合

41

方法

- システム・モデルに対する抽象化 (モデルのリファクタリング) の方法と基本的には同じである
- ただし, 目的は異なる

42

制約, 注意

* 抽象化の結果, 現在ドメインに存在しないもの, あるいはドメインに存在する価値がないものが出現してはならない
* その判定にはドメインの構成員の同意が必要である

43

参照

* 「アナリシスパターン」, マーチン・ファウラー, アジソン・ウェスレイ, 1998

44

汎用領域をメタモデル化する

45

要点

抽象化を行う対象のうち, 多くの事例に共通するような汎用領域は, メタモデル化を行うことによって, よりよい見通しを立てたり, ベスト・プラクティスを導入できる場合がある

ただし, メタモデル化によって, ドメイン固有の知識を消したり, 矮小化してはならない

46

そのために

抽象化に際して, 汎用性のある領域である場合には, メタモデル化を検討する

いくつかの方略がある

- 既存メタモデルの再利用
- 既存メタモデルの拡張
- 新規メタモデルの作成

47

既存メタモデルの再利用

もっともコストが低く, メタモデル自身がそれなりの吟味を経ているはずなので高い有用性が期待できる

その代わり, ドメインによっては適合しない場合もある. そのような場合には, そのメタモデルに無理矢理押し込めてはいけない

48

既存メタモデルの拡張

その分野に既にメタモデルが存在するが,
現ドメインにそのまま使えない場合には,
メタモデルの拡張を検討する

優れたメタモデルならば, 比較的簡単に拡張が可能である

汎用性と固有性のバランスをとらなければならない

49

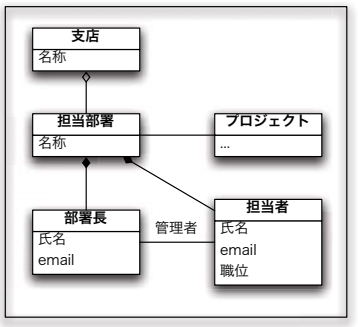
新規メタモデルの作成

既存メタモデルの拡張では十分でなかったり, その分野でメタモデルが存在しない場合には自前でのメタモデル作成を検討する

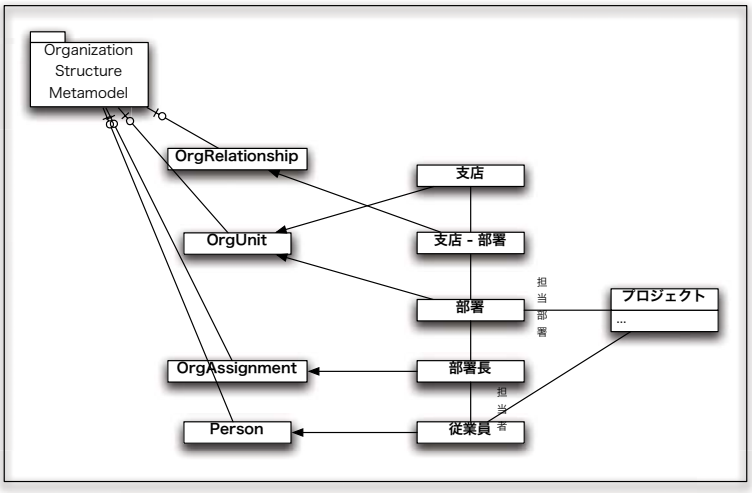
ただし, そのためにコストが掛かること, よいメタモデルには時間が掛かることなどを考慮の上, メタモデル化が引き合うか, そもそも必要かを考えなければならない

50

例



実際のモデリングにはステレオタイプなどを使って描きます



51

その他の例

- Business Motivation Model
- Organization Structure Metamodel
- Business Process Definition Metamodel
- その他ドメイン固有メタモデル
- あまりOMGに振り回されるのも考え物だが

52

説明

* ドメイン・モデルの中に, 汎用的な領域を発見したら, メタモデルの適用を検討する

53

利益

* メタモデルを用いることにより, 抽象化のベスト・プラクティスを利用できる
* ドメイン・モデルの見通しをよくしたり, 完成度を高めることができる

54

文脈

* 多くのドメイン・モデルで本質が類似しており, 共通に用いられるような領域がある場合

55

方法

* 対象となる領域を特定し, 適用可能な既存のメタモデルを探す
* あるいは既存のメタモデルをドメイン・モデルに会わせて拡張する
* あるいは新規のメタモデルを作成する

56

制約, 注意

* メタモデルの適用によって, ドメイン・モデル固有の性質を失ったり, 過度の一般化を行わないようにする
* メタモデルの適用には, コストが掛かる面があることを意識する

57

参照

* http://www.omg.org/
* 他

58

横断的関心事を抽出する

59

要点

通常は, 名詞的存在や分野内で抽象化を考えることが多いが,

形容詞的存在や

分野を横断する関心事の

抽象化を考えることによって, よりよいドメイン・モデルを構築できる場合がある

60

そのために

ドメイン・モデルの中で異なる分野にも関わらず, 共通する「性質」, 特に「形容詞」的な性質 がないかを考える

これを「横断的関心事」(アスペクト) と呼ぶ

実装上のアスペクトではなく, 概念上のアスペクトであることに注意

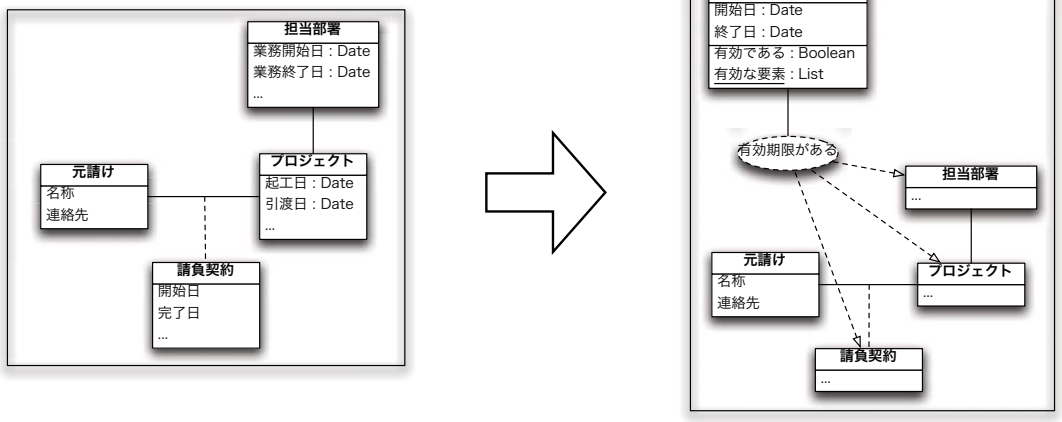
61

横断的関心事は構造よりも, 振る舞いに関連する抽象化で, これは概念上だけでなく, 実現上も有用な抽象化である

横断的関心事に注目することによって, 今まで見逃していた抽象化を行うことができる

62

例



今のところ横断的関心事を表す標準の記法はありません

63

その他の例

* 識別可能な
* 分類可能な
* 位置に紐付いた
* 版管理された
* 永続的な
* 期日のある
* 説明の付いた
* コメント可能な
* テンプレート付きの
* 状態を持つ
* .....

64

説明

* 異なる概念の間に共通する性質がある場合, それを横断的関心事 (アスペクト) として抽出する

65

利益

* ドメインを「そのまま」表したドメイン・モデルでは, 異なる概念に共通する (ドメイン上の) 性質を見逃しがちだが, これも重要なドメイン概念である

66

文脈

* ひととおりのドメイン・モデリングが済み, 次のレベルのモデリングに移行するとき

67

方法

* 異なる概念に共通する性質を他のモデルや, 例を参考にして検討する
* それらは形容詞で表される場合が多い
* ドメインにとってはメタレベルの概念である場合も多い

68

制約, 注意

* aspect4jなどのような実装レベルのアスペクトとは異なることに注意
* Rails, Grailsなどでは, メタ・プログラミングによって実現に対応付けることもできる

69

制約, 注意

* ドメインにとってメタになってくるので, どうしても「解釈」の視点が入ってくるが, 行き過ぎないようにする
* ドメインのアーキテクチャを決めると考えるとやりやすいかも知れない

70

参照

* 「ビジネスパターンによるモデル駆動設計」, Pavel Hruby, 日経BPソフトプレス, 2007
* http://grails.org/
* Ruby on Railsの`acts_as_*`プラグイン

まとめ

- ドメイン・モデリングは役に立つ
- ドメイン・モデリングは (DDDの時代からさらに) 進化し続けている
- ドメインの抽象化, ドメインのアーキテクチャ構築を (システムに引きずられずに) 行うことが重要である
- それがモデル駆動開発につながる

73