Domain-Specific Modeling for Full Code Generation

派生開発/SPLE を支援

ドメインスペシフィック・モデリングと完全なコード自動生成

Juha-Pekka Tolvanen, Ph.D.
MetaCase

September 2010

内容

- ドメインスペシフィックモデリングについて
  - -UMLなどコードレベルのモデリングとの違いは？
- 産業界の実績･事例
- ドメインスペシフィック言語の構築･作り方
  - -ドメインコンセプトの特定とメタモデル化
- コードジェネレータの設定･構築
- 始めかた
  - -DSMに適切なドメイン、必要となるツールの機能
- まとめ、質疑応答

＜講師紹介＞　ユハ･ペッカ　トルバネン博士

MetaCase 社 CEO ユハ･ペッカ博士は、
DSMフォーラムを創立したメンバーでもあり、
モデル駆動開発手法とツールの発展、
とりわけドメイン･スペシフィック モデリング、
メタモデリング、コード生成の分野に
1991年から貢献。

ワールドワイドで多岐にわたる企業へのコンサルタントを通じた経験から書籍 Domain-Specific Modeling の執筆や70以上の論文を発表している。OOPSLA では2001年から DSMワーク ショップを運営し、今年10周年を迎える。また MetaCase社 からは、IEEE Software特集記事に〝ドメインスペシフィックモデリングのワーストプラクティス〟 が掲載された。　ユハ･ペッカ博士は、1998年に博士号を取得しフィンランドのユベスキュレ大学で助教授としてソフトウエア開発手法を教えています。

ドメインスペシフィックモデリングとは？

なぜ新しい取組みが求められるのか？
ドメインスペシフィックモデリング (DSM) について
実践的な成功例
産業界の事例

生産性の向上は停滞中、、、

- “The entire history of software engineering is that of the rise in levels of abstraction”

  “ソフトウェアエンジニアリングの歴史は抽象化レベルの上昇"

  Grady Booch

- 近年の汎用プログラミング言語ではもう生産性は上げられない
- UMLのようなコードの視覚化では生産性は上がらない

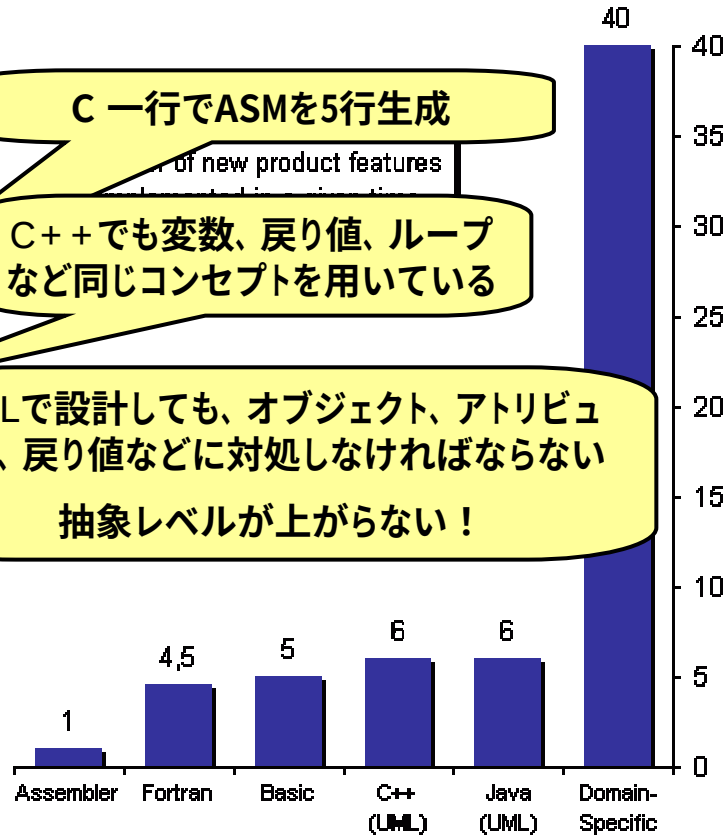
- 汎用ではなく専用のDSMなら
- さらに抽象レベルを上げて
- 完全な製品コードを生成できる

C 一行でASMを5行生成

C++でも変数、戻り値、ループなど同じコンセプトを用いている

UMLで設計しても、オブジェクト、アトリビュート、戻り値などに対処しなければならない

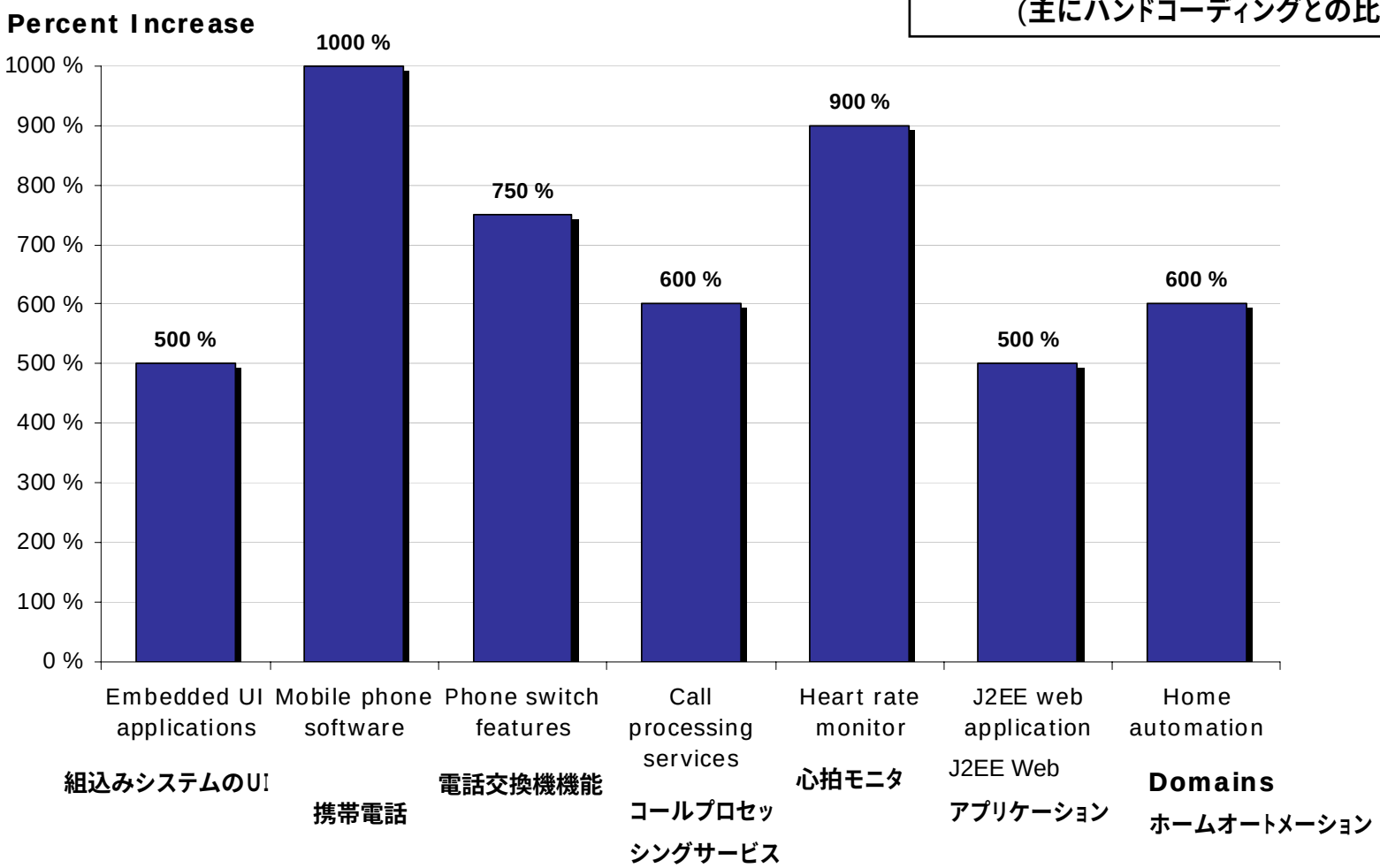
抽象レベルが上がらない！



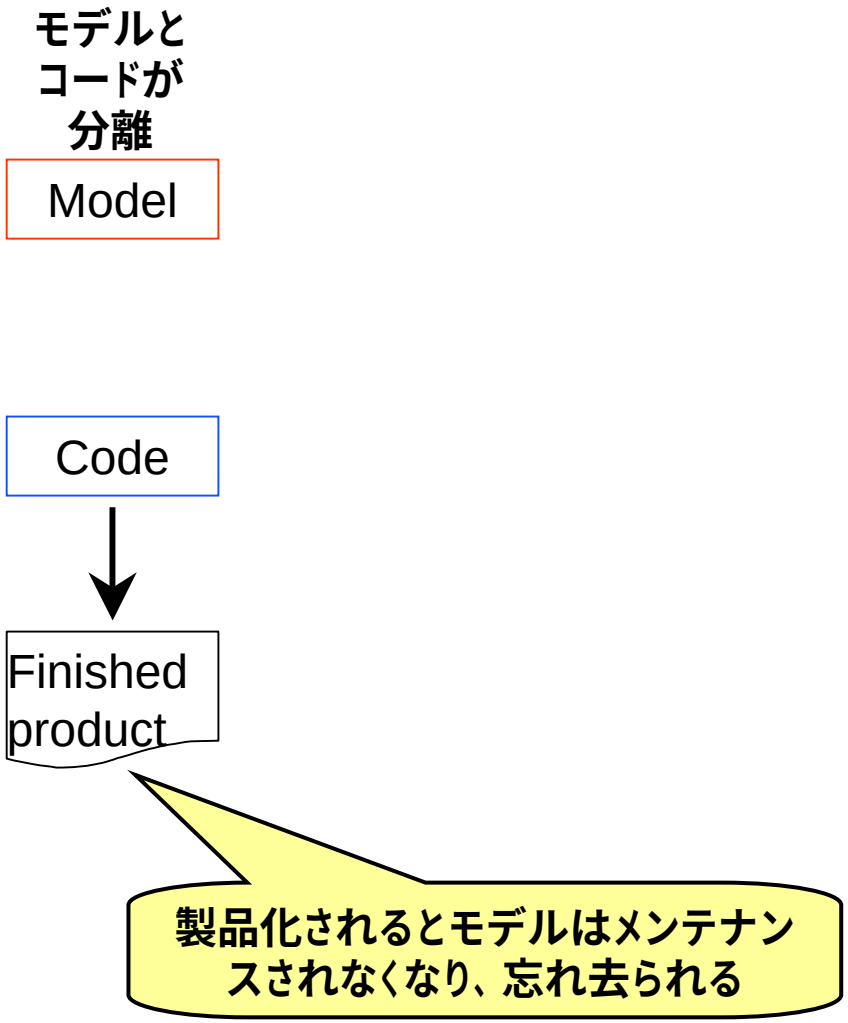
*Software Productivity Research & Capers Jones, 2002

DSM によって生産性を向上

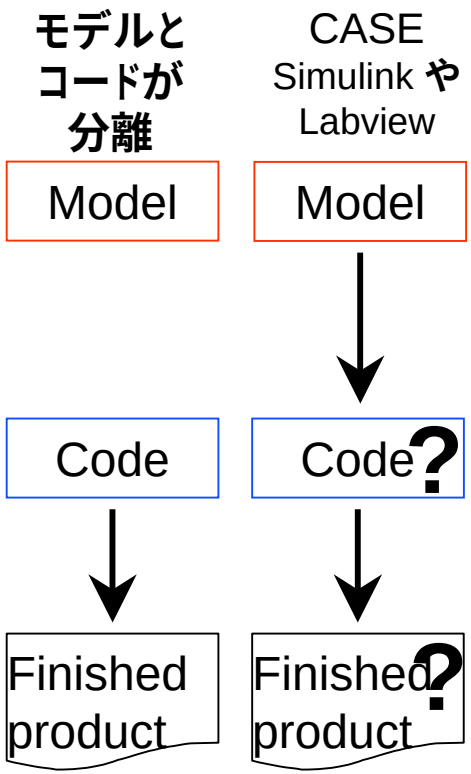
従来の開発手法との比較
(主にハンドコーディングとの比較)



DSMで飛躍的な成果
他のアプローチと異なったモデルの活用

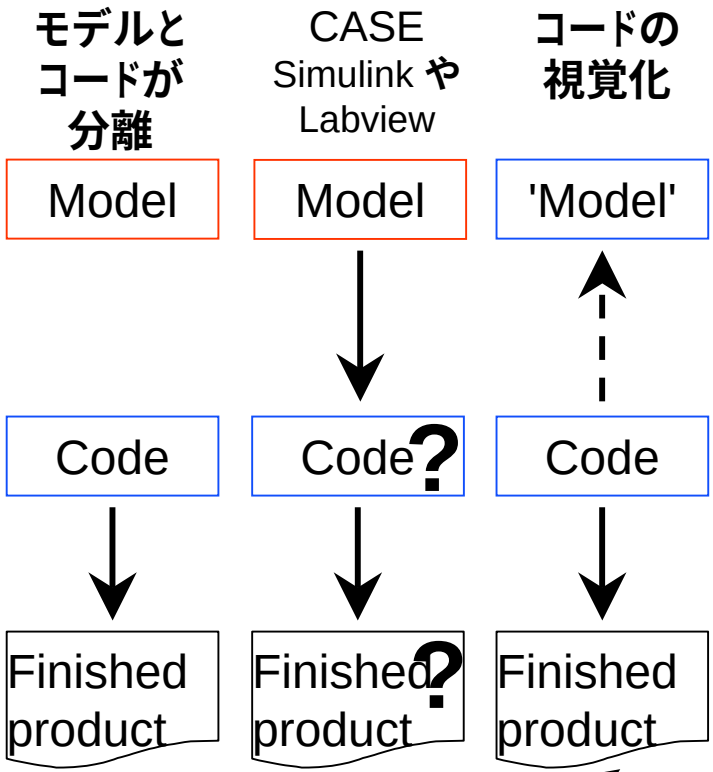


DSMで飛躍的な成果
他のアプローチと異なったモデルの活用



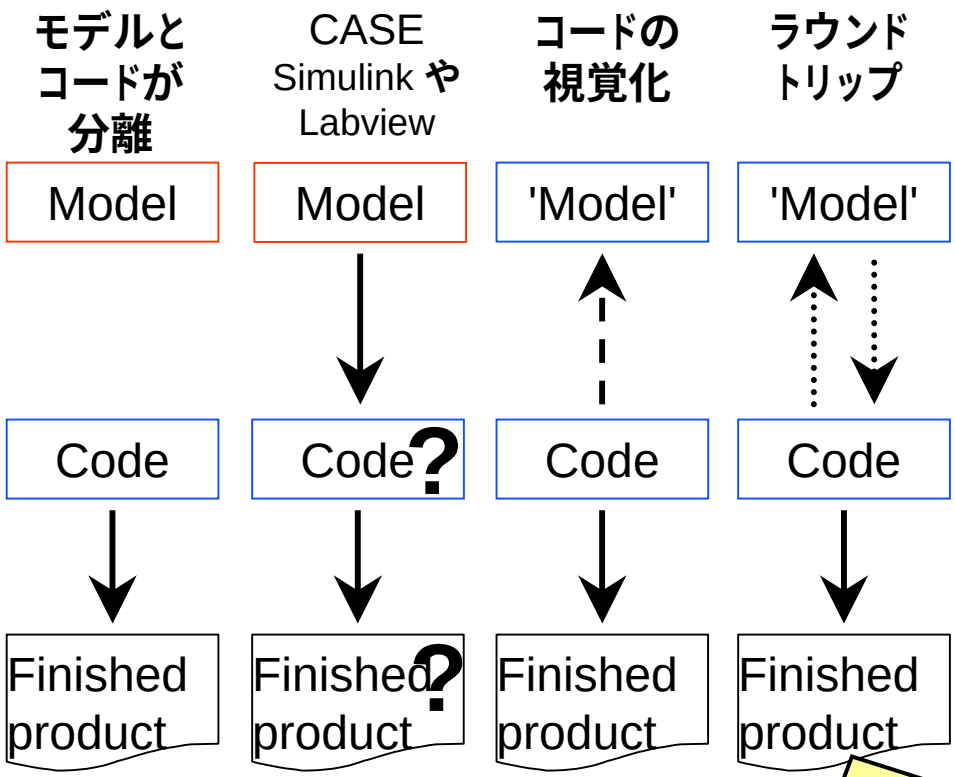
モデリングや生成コードが限定的。プロトタイプやシミュレーションには使えても、実製品レベルには十分ではない

DSMで飛躍的な成果
他のアプローチと異なったモデルの活用



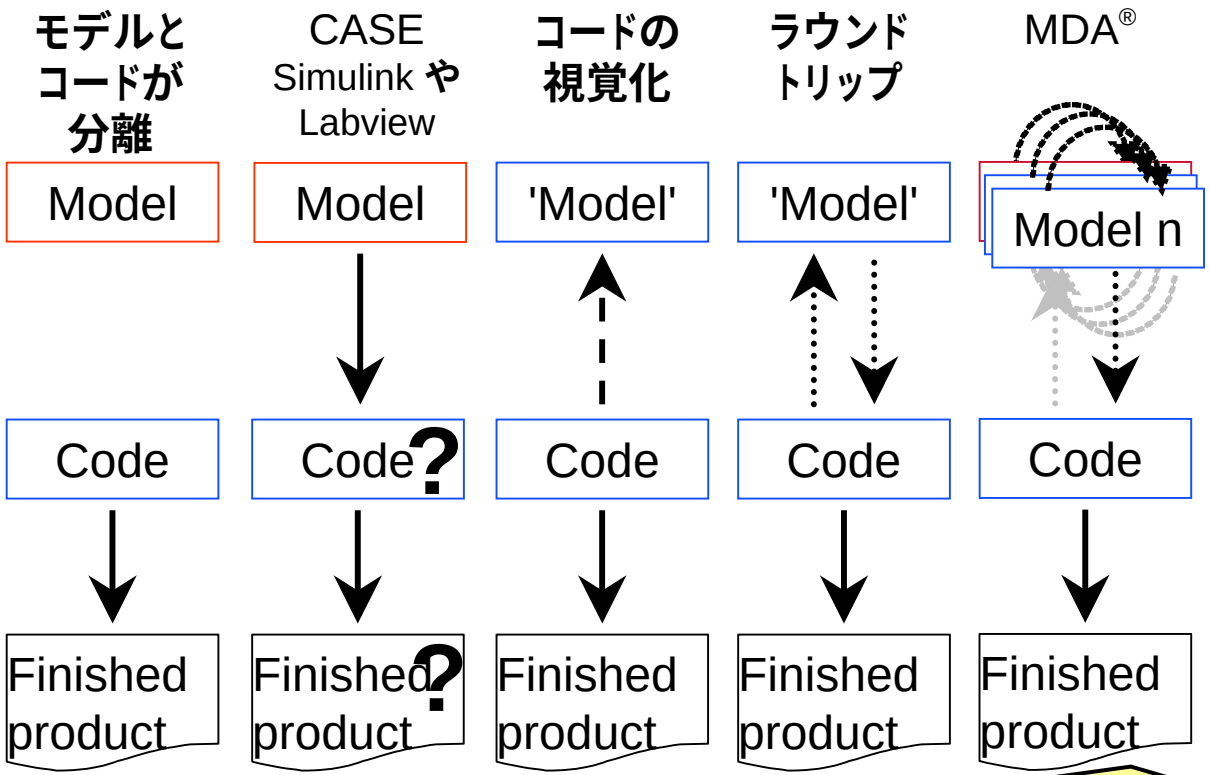
リバースエンジニアリング。コード構造を描写するだけであって問題空間を扱えない

DSMで飛躍的な成果
他のアプローチと異なったモデルの活用



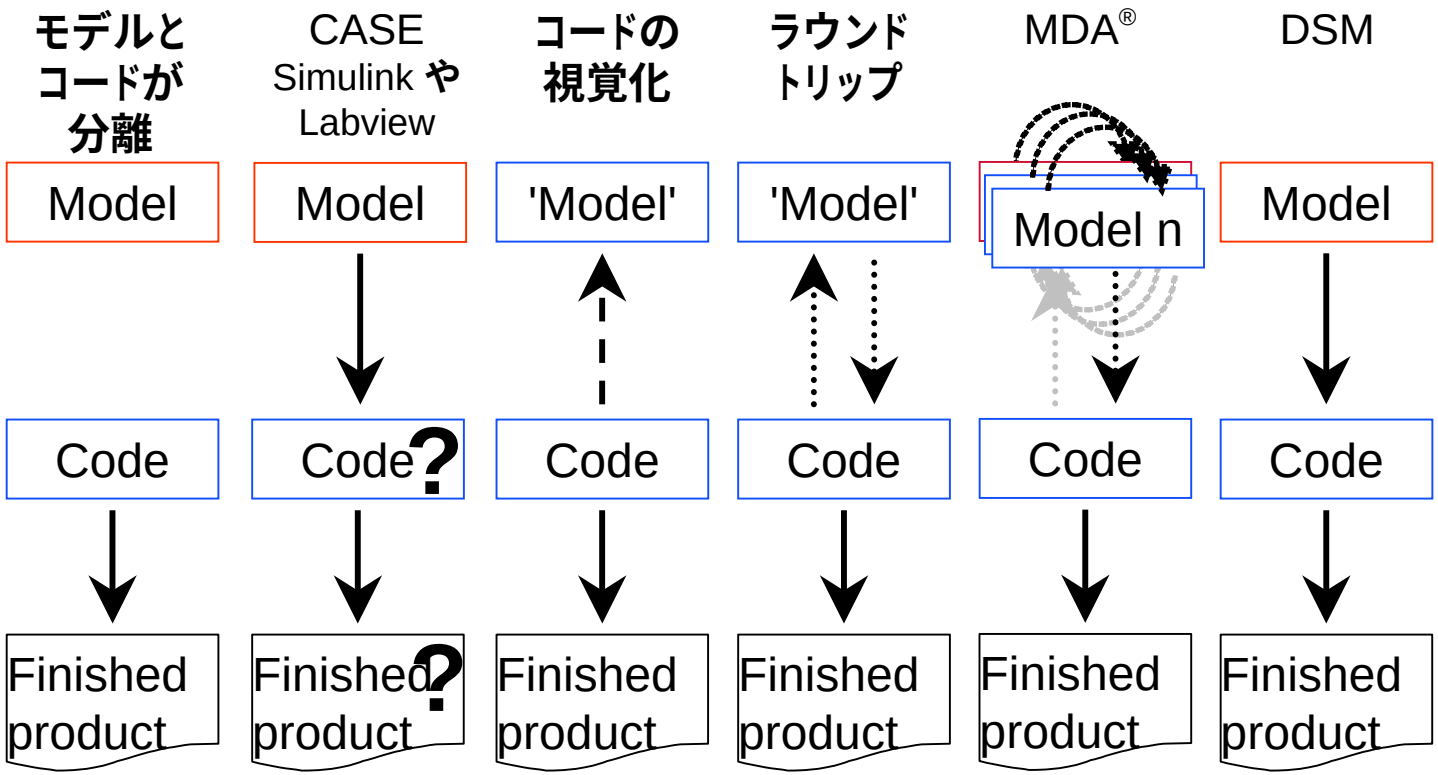
モデルとコード間で同じインフォメーションを自動同期させようとすること。UMLベンダが推奨していたが、UMLがコードと同期できない。せいぜいクラス図の一部のみ

DSMで飛躍的な成果
他のアプローチと異なったモデルの活用



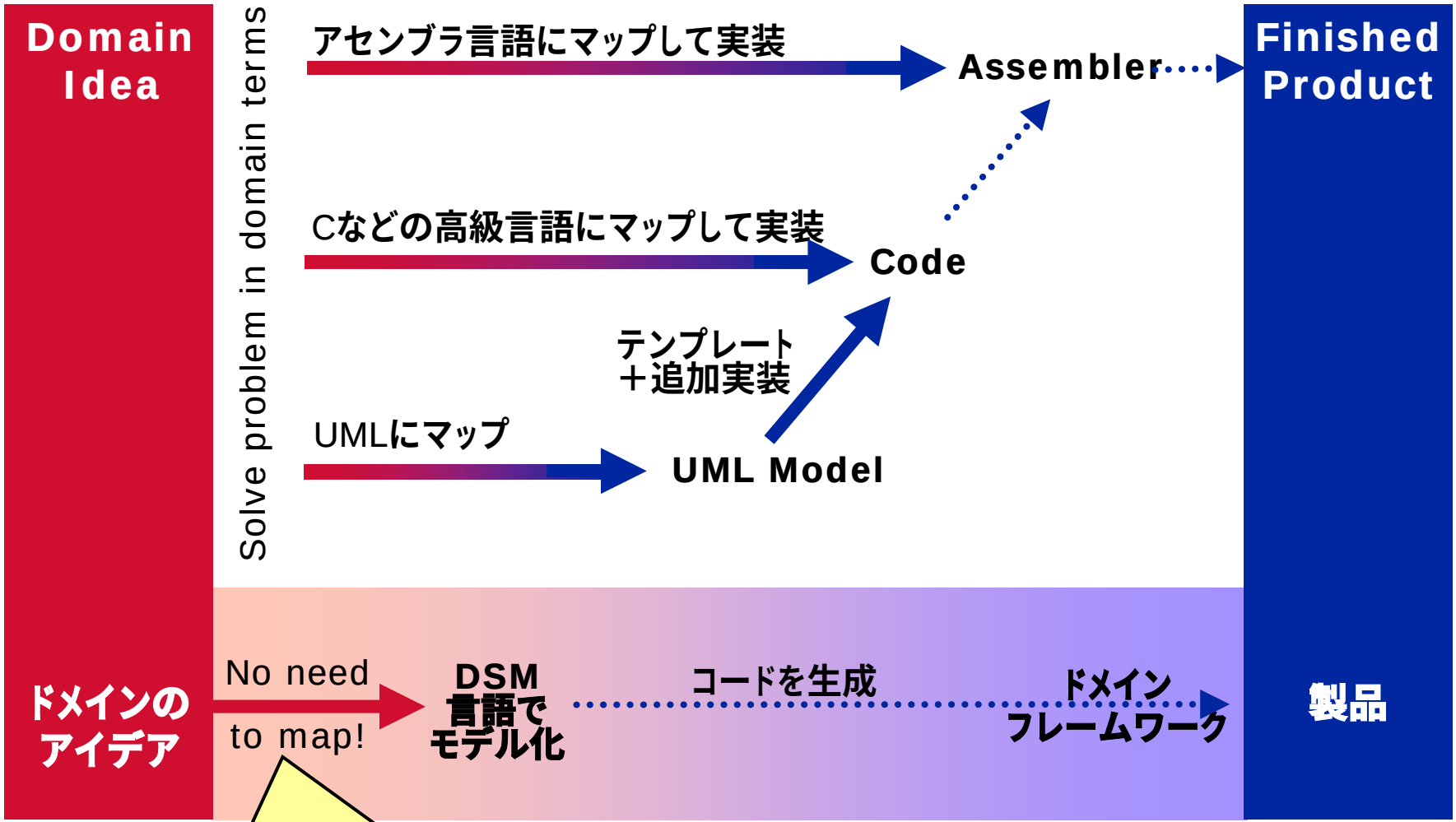
UMLのコード生成は制限があり、OMGは10年前にMDAを提唱。ハイレベルのモデルをローレベルに変換して、最後にコードへ。実践では使えない。ウォータフォール型のプロセスになり、例えばハイレベルのモデルが変更されても、ローレベルのモデルが容易にアップデートできない

DSMで飛躍的な成果
他のアプローチと異なったモデルの活用



モデルから完全なコードが生成され、それを変更する必要もない。2番目のCASEと同じに見えるかもしれないが、開発チームの望んだモデリング、コードを得ることができる。

製品機能をモデリング 対 コードレベルでモデリング

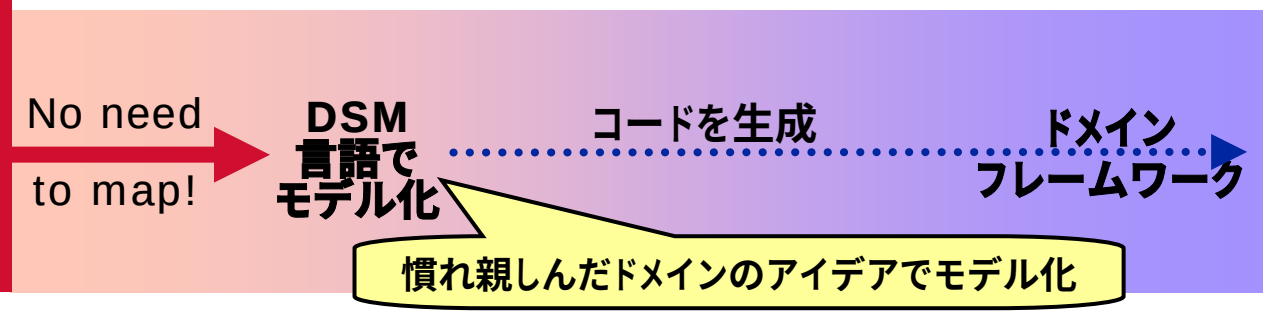


デジタル腕時計：デモ例

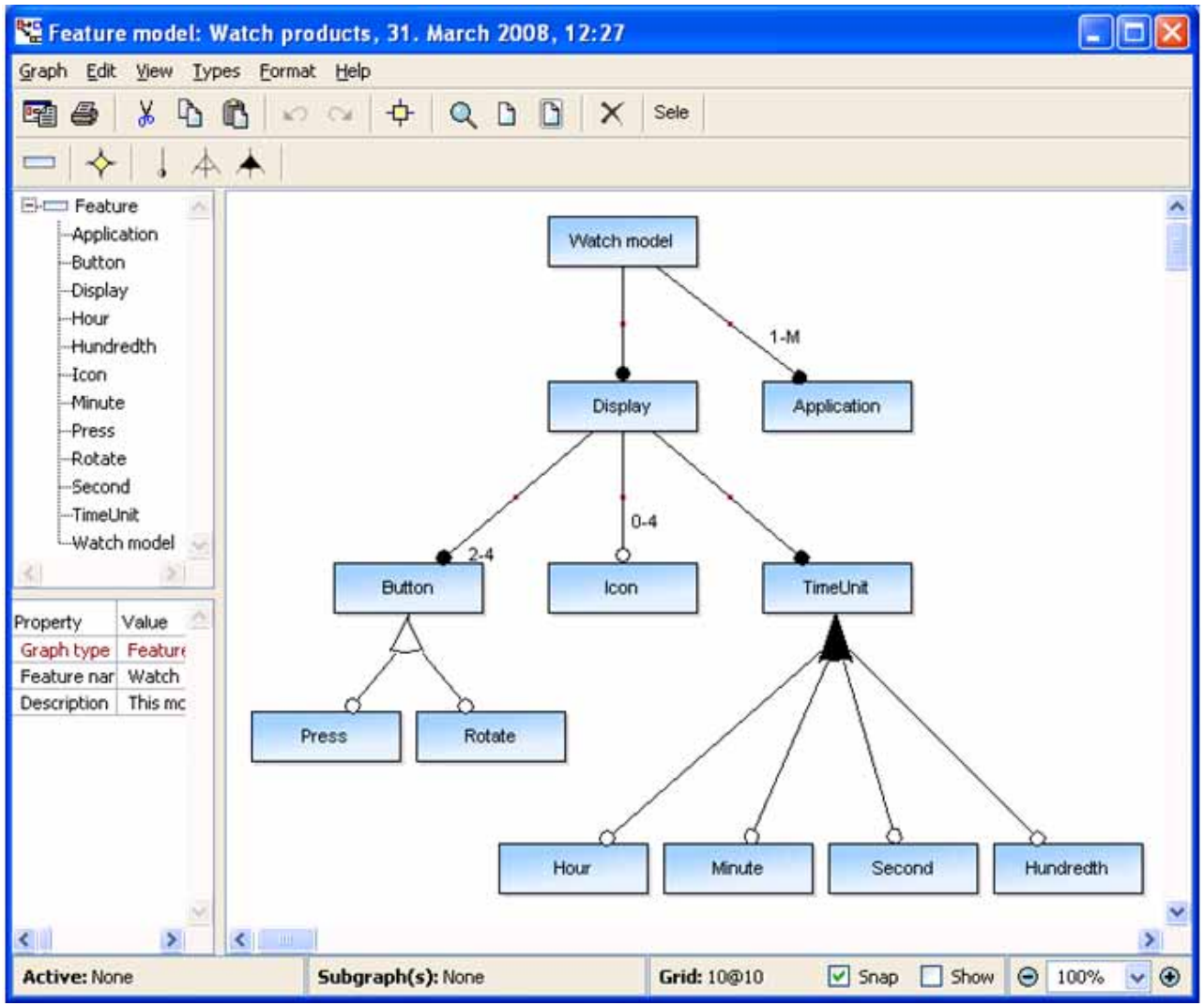
Domain Idea

- **製品ファミリー**
  - 機種展開：男性用，女性用，スポーツタイプ，子供向け，旅行向き，ダイビング用…
- **再利用可能なアプリケーション部品**
  - 時計，アラーム，タイマー，世界時計，ストップウォッチ…
- **複雑性をモデリング担当者から軽減させる**
  - Model-View-Controller を分離 (http://ja.wikipedia.org/wiki/Model_View_Controller)
  - リアルタイム性に関わる課題はフレームワーク内で対処
- **Java コードの生成**
  - MIDP や C など異なるプログラムコードも生成できる

Finished Product



プロダクトラインのフィーチャモデルも描けるが、、



DSM: ドメインスペシフィック モデリング

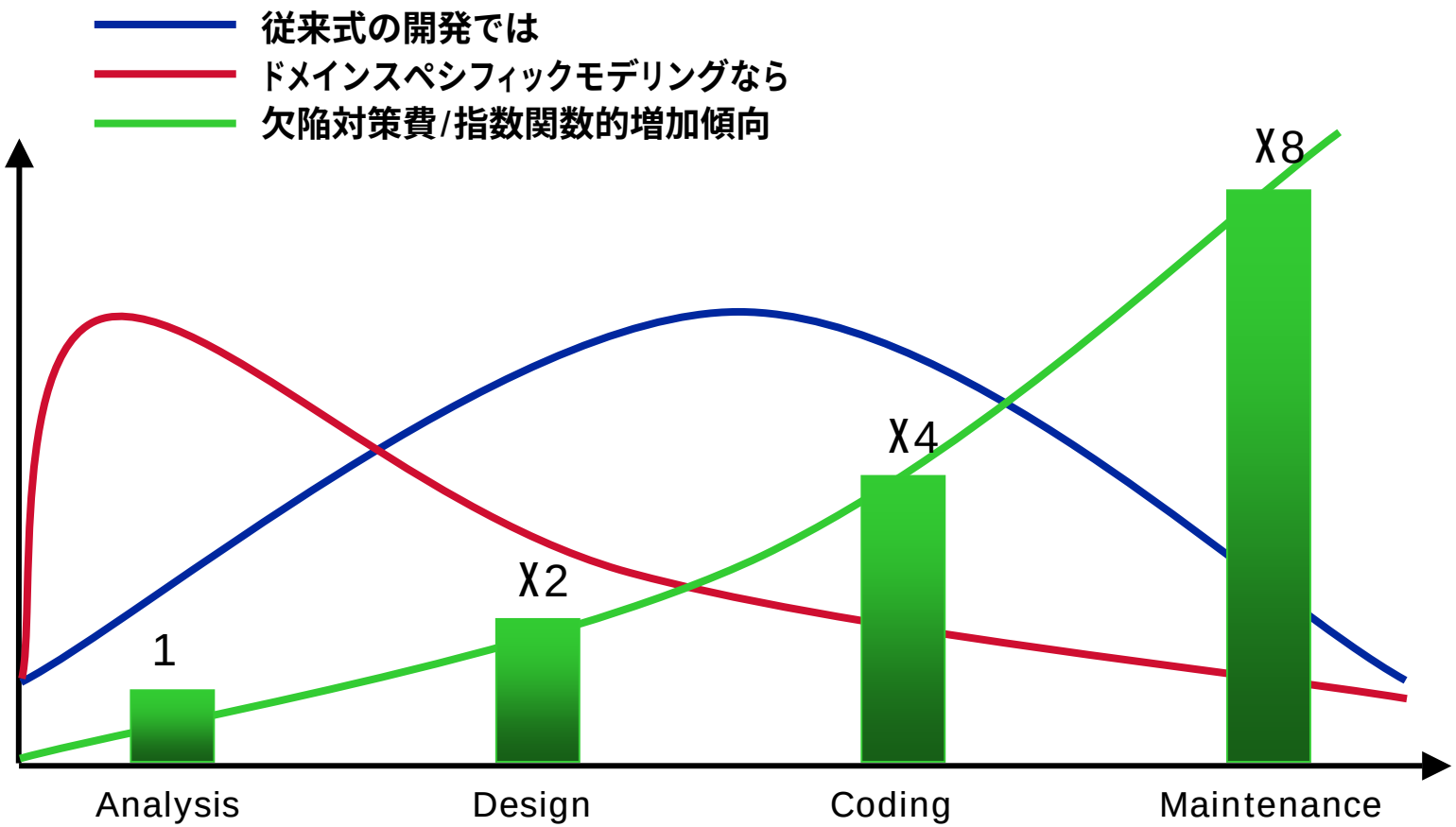
- **ドメインの知識を形（モデル）に（コードではなく）**
  - コード実装レベルから、抽象度を向上させる
  - ドメインの抽象概念を利用する
  - ドメインのコンセプトやルールをモデリング環境に組込む
  - 単一の製品系列にフォーカスする

- **実装担当者には、ドメイン専用用語を用いて開発してもらう**
  - ➔ 慣れ親しんだ専用の用語で開発
  - ➔ システムの課題解決に集中できる
  - ➔ 修正・改善など、問題への対処は一極集中で！
  - ➔ テスト工数を飛躍的に削減。一般に起こり易いエラーを排除できるので

DSM により欠陥を削減

- 典型的なコーディングエラーが起こりえないようになる
  - タイプミス: 通常コンパイル時に検出されるが、それでもランタイムのエラー要因となっている (有効ながら間違っている変数名など)
  - 非効率で粗末なアドホックなコード
  - 不必要で冗長なコード (メンテナンス時に使用され残されてしまったコードなど)
  - 連鎖反応: 相互依存関係によって一箇所の変更が他所の破壊に繋がる
  - リソース管理の課題: メモリの開放忘れ、リソース使用後の開放忘れ
- アーキテクチャ/プラットフォーム/フレームワークの課題に対処できる
  - アーキテクチャやフレームワークの間違った使用、放棄されてしまうことを回避
  - プラットフォーム/フレームワークへの準拠:最新版を活かすことを確実にする
- ソリューション内の欠陥を回避 (正しいものを作ることの支援)
  - 論理的なエラー　や
  - お粗末な設計　の回避

欠陥の起因とかかる費用の関係



Defect distribution and costs*

*Molina, P., Introducing MDD, Code Generation Conference 2010

ドメインスペシフィックモデリングとは？

なぜ新しい取組みが求められるのか？

ドメインスペシフィックモデリング (DSM) について

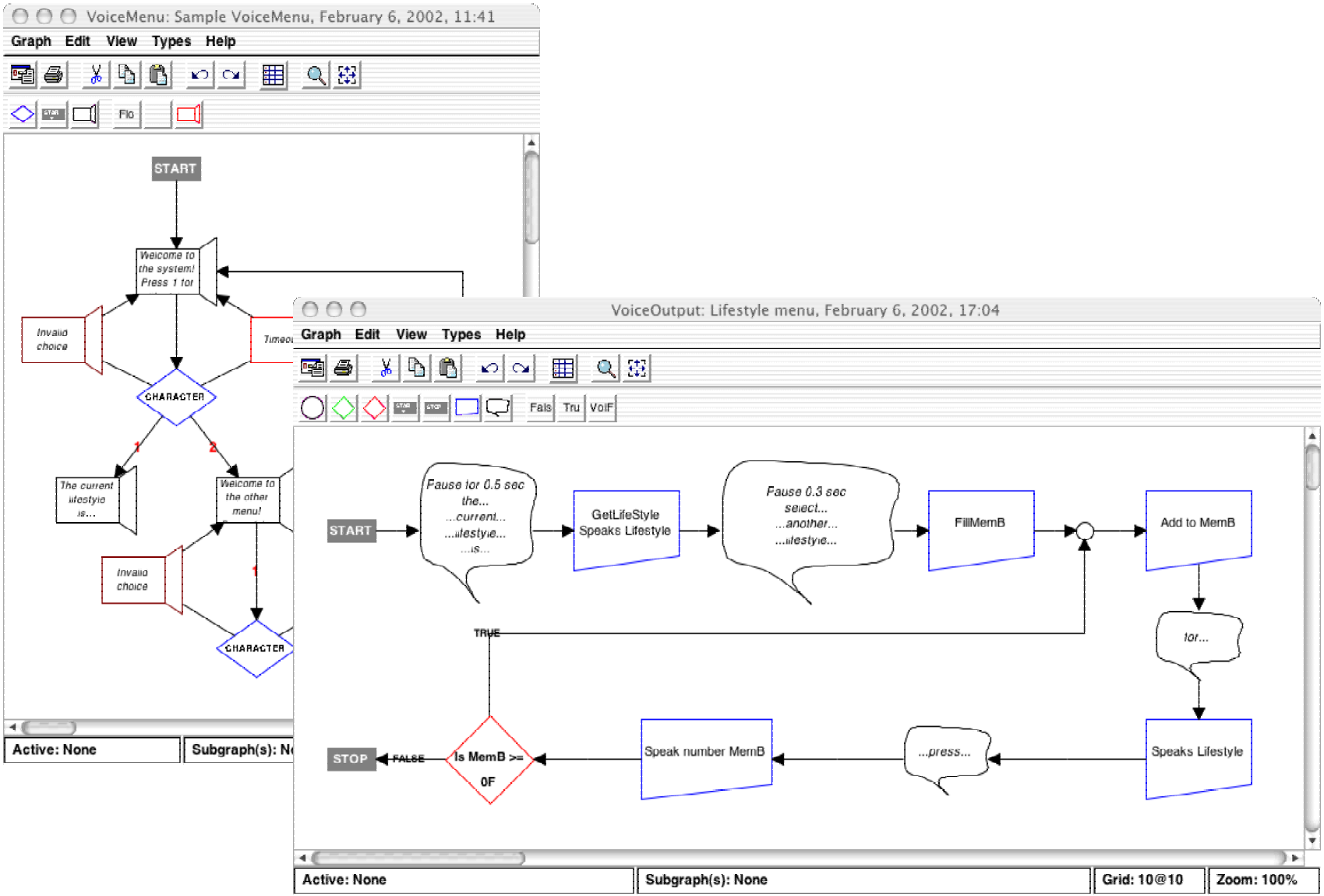
実践的な成功例

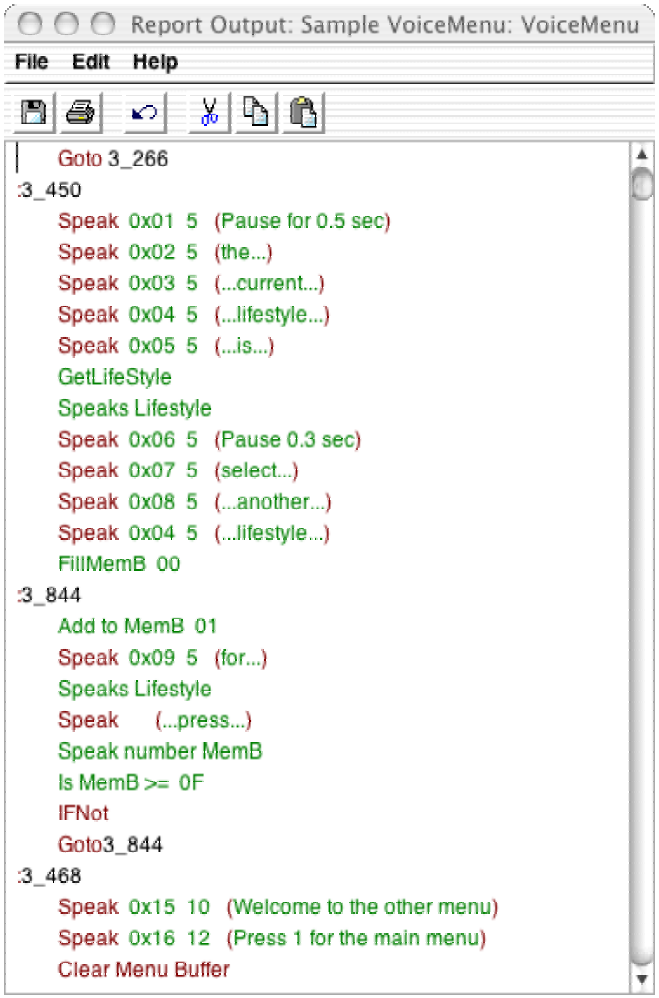
産業界の事例

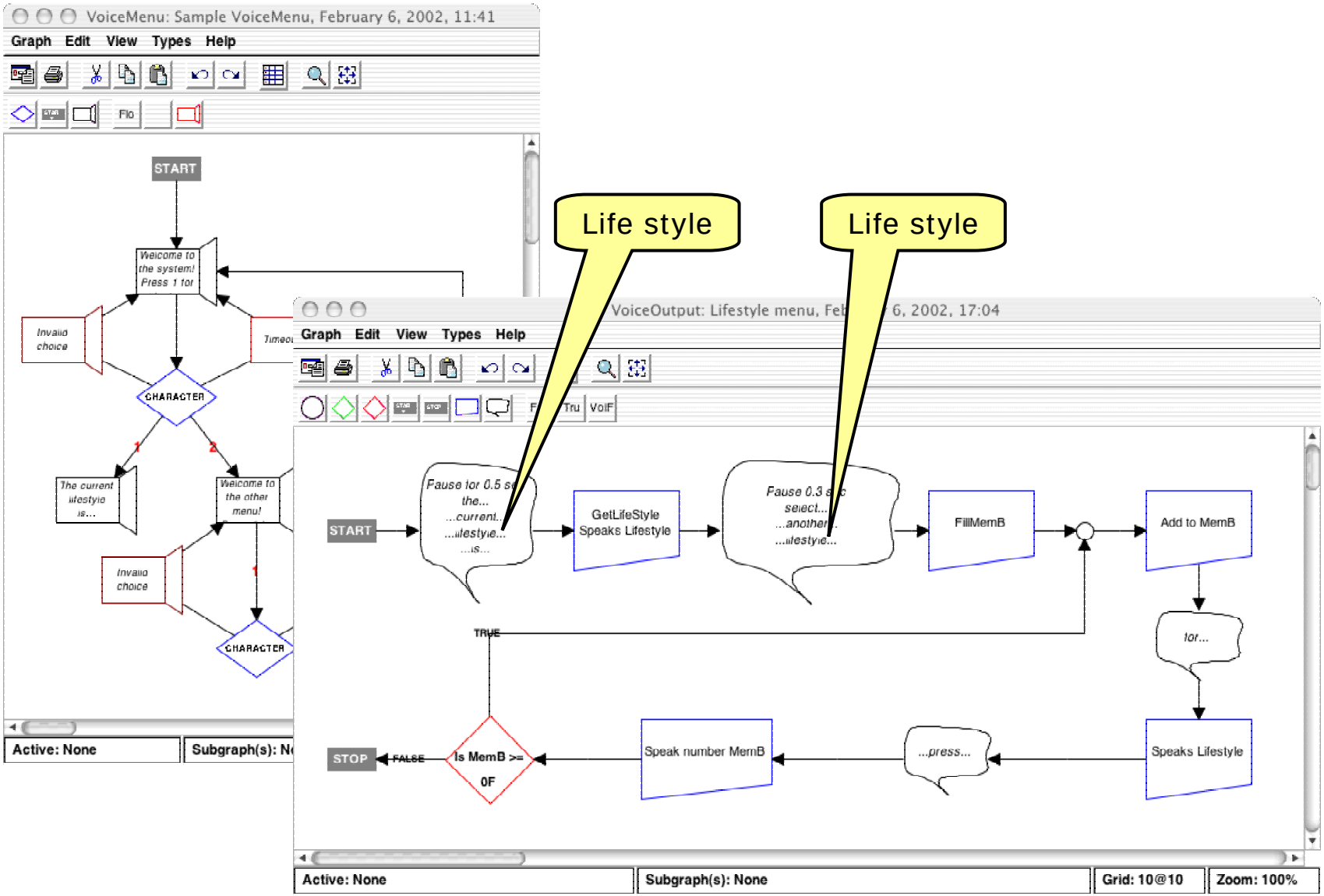
事例：マイクロコントローラの音声メニュー

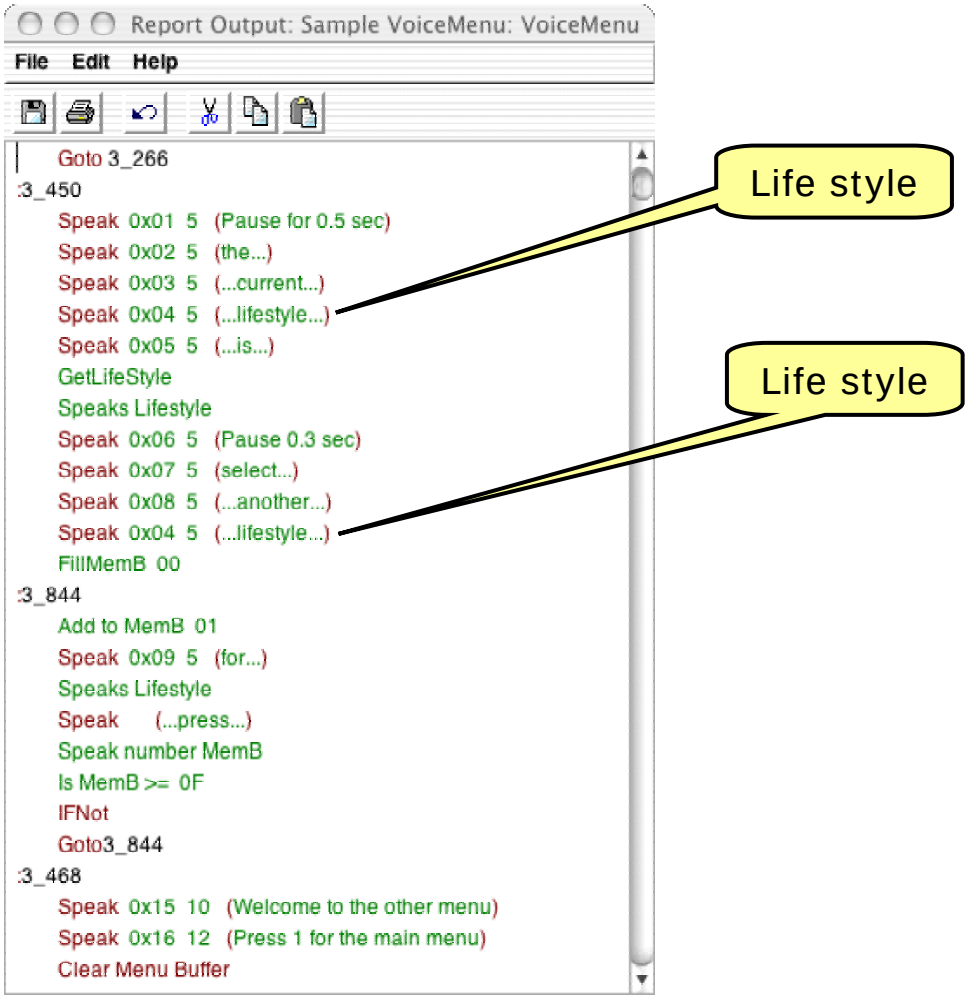
- 照明、ヒータ、アラーム等をリモートコントロールする為のホームオートメーションシステム
- 音声メニューはデバイスニーモニックに直結した（8ビット）アセンブラ風の言語でプログラムされる
- モデリング言語は全てのメニュー構造と個々の音声プロンプトを定義
- １００％のメニューコードを生成

➢ 開発時間が1週間から1日になった！


VoiceMenu: Sample VoiceMenu, February 6, 2002, 11:41
Graph Edit View Types Help
START
Welcome to the system! Press 1 for
Invalid choice
CHARACTER
The current lifestyle is...
Welcome to the other menu!
Invalid choice
CHARACTER
Active: None
VoiceOutput: Lifestyle menu, February 6, 2002, 17:04
Graph Edit View Types Help
Fals Tru VoIF
START
Pause for 0.5 sec the... ...current... ...lifestyle... ...is...
GetLifeStyle Speaks Lifestyle
Pause 0.3 sec select... ...another... ...lifestyle...
FillMemB
Add to MemB
for...
TRUE
STOP
FALSE
Is MemB >= 0F
Speak number MemB
...press...
Speaks Lifestyle
Active: None
Subgraph(s): None
Grid: 10@10
Zoom: 100%


Report Output: Sample VoiceMenu: VoiceMenu
File Edit Help
Goto 3_266
:3_450
Speak 0x01 5 (Pause for 0.5 sec)
Speak 0x02 5 (the...)
Speak 0x03 5 (...current...)
Speak 0x04 5 (...lifestyle...)
Speak 0x05 5 (...is...)
GetLifeStyle
Speaks Lifestyle
Speak 0x06 5 (Pause 0.3 sec)
Speak 0x07 5 (select...)
Speak 0x08 5 (...another...)
Speak 0x04 5 (...lifestyle...)
FillMemB 00
:3_844
Add to MemB 01
Speak 0x09 5 (for...)
Speaks Lifestyle
Speak (...press...)
Speak number MemB
Is MemB >= 0F
IFNot
Goto3_844
:3_468
Speak 0x15 10 (Welcome to the other menu)
Speak 0x16 12 (Press 1 for the main menu)
Clear Menu Buffer


VoiceMenu: Sample VoiceMenu, February 6, 2002, 11:41
Graph Edit View Types Help
START
Welcome to the system! Press 1 for
Invalid choice
CHARACTER
The current lifestyle is...
Welcome to the other menu!
Invalid choice
CHARACTER
Active: None
Life style
Life style
VoiceOutput: Lifestyle menu, Feb 6, 2002, 17:04
Graph Edit View Types Help
START
GetLifeStyle Speaks Lifestyle
FillMemB
Add to MemB
TRUE
for...
Speaks Lifestyle
...press...
Speak number MemB
Is MemB >= 0F
FALSE
STOP
Active: None
Subgraph(s): None
Grid: 10@10
Zoom: 100%

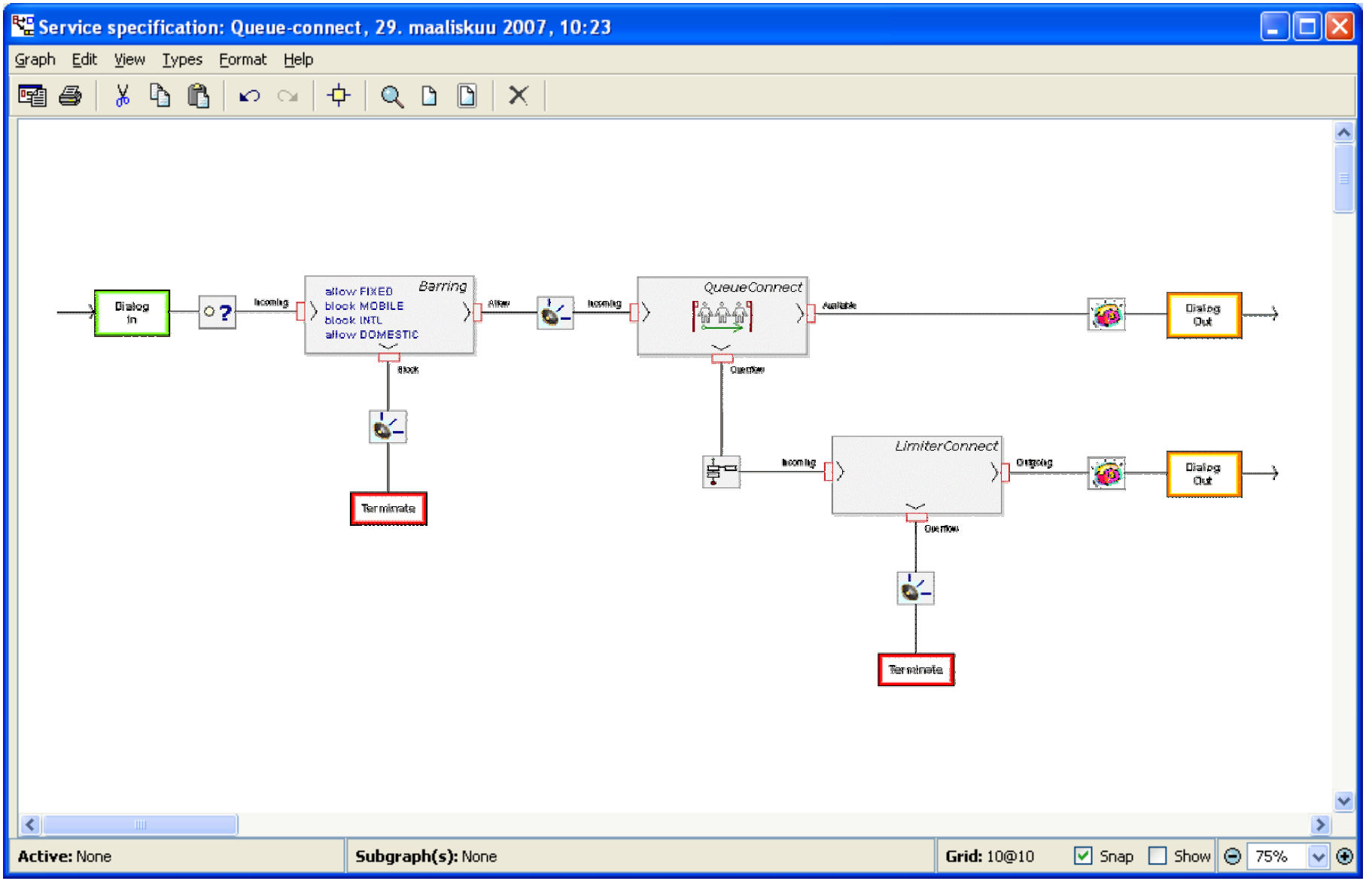

Report Output: Sample VoiceMenu: VoiceMenu
File Edit Help
Goto 3_266
:3_450
Speak 0x01 5 (Pause for 0.5 sec)
Speak 0x02 5 (the...)
Speak 0x03 5 (...current...)
Speak 0x04 5 (...lifestyle...)
Speak 0x05 5 (...is...)
GetLifeStyle
Speaks Lifestyle
Speak 0x06 5 (Pause 0.3 sec)
Speak 0x07 5 (select...)
Speak 0x08 5 (...another...)
Speak 0x04 5 (...lifestyle...)
FillMemB 00
:3_844
Add to MemB 01
Speak 0x09 5 (for...)
Speaks Lifestyle
Speak (...press...)
Speak number MemB
Is MemB >= 0F
IFNot
Goto3_844
:3_468
Speak 0x15 10 (Welcome to the other menu)
Speak 0x16 12 (Press 1 for the main menu)
Clear Menu Buffer
Life style
Life style

事例：IMSサービスの作成

- IPベースの通信サービスを即座に構成して展開
- サービスのコンセプトに従ってモデリングできる
- 一つのデザインから全ての成果物を得る
  - コード、構成情報、ドキュメント
  - サービスを実装できるフレームワーク、アプリケーションサーバ

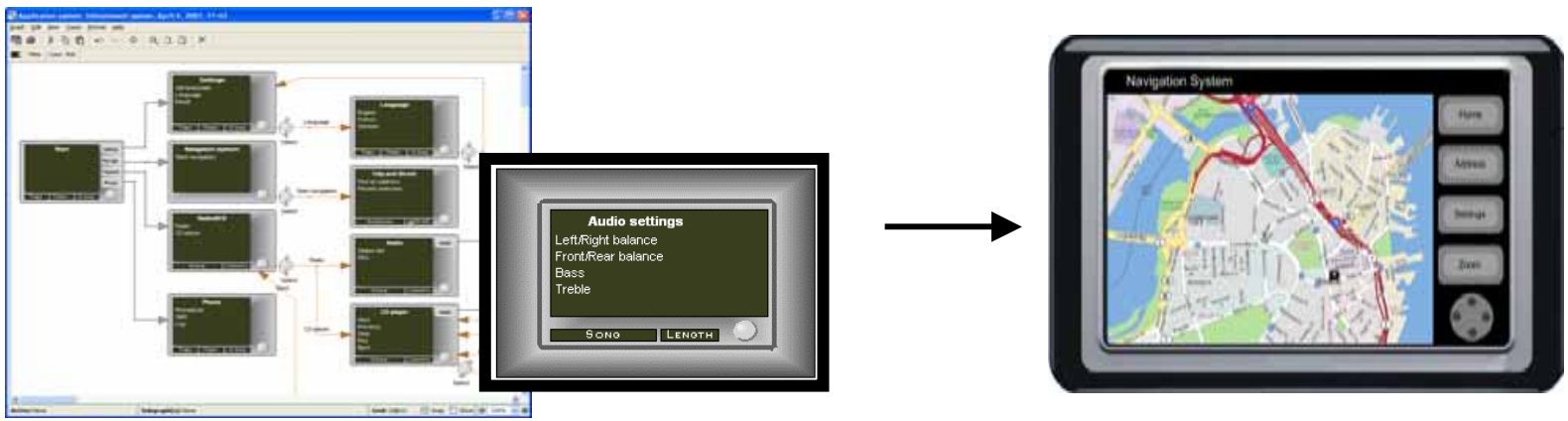
- 高い抽象度で迅速かつ容易にサービスを生成できる
  - SIPやHTTPなどに見られるような横断的な課題を排除することで

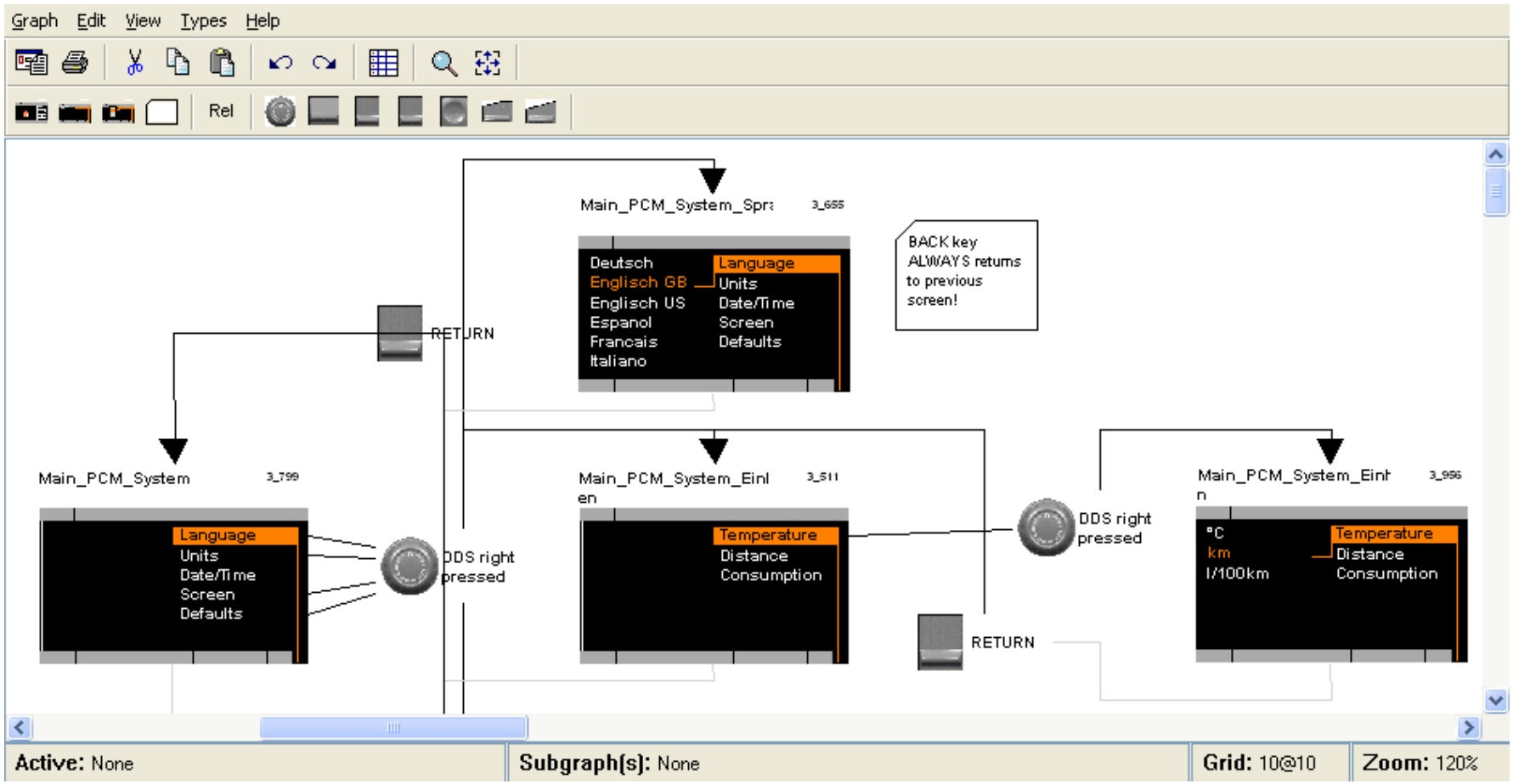
- 同様のドメイン：
  - ワークフロー
  - ビジネスプロセスモデル
  - テレコムネットワーク


Service specification: Queue-connect, 29. maaliskuu 2007, 10:23
Graph Edit View Types Format Help
Dialog In
?
Incoming
Barring
allow FIXED
block MOBILE
block INTL
allow DOMESTIC
Block
Terminate
Allow
Incoming
QueueConnect
Available
Dialog Out
Overflow
Incoming
LimiterConnect
Outgoing
Dialog Out
Overflow
Terminate
Active: None
Subgraph(s): None
Grid: 10@10
Snap
Show
75%

インフォテインメントシステム

- ハイレベルのHMI (ヒューマン・マシン・インタフェース) コンセプトを用いたシステム
  - ノブ、表示、イベント、対話式メニュー等
- プロトタイピングによる早期のフィードバックが可能
- HMIの一貫性を改善し、バリエーションに対応
- 製品のソースコードの品質を改善
- チーム内の複数の役割をまとめる
- 手書きのコードとコード生成を一体化させるために、既存のフレームワークとリンクする




Graph Edit View Types Help
Rel
Main_PCM_System_Spr
3_655
Deutsch
Englisch GB
Englisch US
Espanol
Francais
Italiano
Language
Units
Date/Time
Screen
Defaults
BACK key ALWAYS returns to previous screen!
RETURN
Main_PCM_System
3_799
Language
Units
Date/Time
Screen
Defaults
DDS right pressed
Main_PCM_System_Einh en
3_511
Temperature
Distance
Consumption
DDS right pressed
Main_PCM_System_Einh n
3_956
°C
km
l/100km
Temperature
Distance
Consumption
RETURN
Active: None
Subgraph(s): None
Grid: 10@10
Zoom: 120%

ドメインスペシフィックモデリングとは？

なぜ新しい取組みが求められるのか？
ドメインスペシフィックモデリング (DSM) について
実践的な成功例
産業界の事例

産業界から多くの実績が公開

- B.Braun; Medical Dialysis machines
- Bell Labs / AT&T / Lucent; 5ESS telecommunications switch,
- EADS: Tetra terminals
- Panasonic: embedded UI
- Honeywell; embedded software architectures
- Polar Electro: heart rate monitors
- ORGA; SIM toolkit & JavaCard
- Pecunet; B2B E-Business: insurance
- LexiFi; mlFi, financial contracts
- DuPont; Activity Modeling
- NASA ASE group; Amphion
- NASA JPL; embedded measurement systems
- Nokia; Mobile Phone product line
- USAF; Message Transformation and Validation
- ...

以下のリンクで公開されています

http://www.dsmforum.org/cases.html
(DSMフォーラム　www.DSMForum.org)

Experiences from practice

Panasonic

“標準的な開発手法に比較して、5倍の生産性向上が得られました”
http://www.fuji-setsu.co.jp/files/dsm07safa.pdf

POLAR
LISTEN TO YOUR BODY

“従来のアプローチと比較して、開発速度が7.5倍速くなり、コードや開発プロセスの品質を飛躍的に向上できた
http://www.metacase.com/cases/polar.html”

NOKIA
CONNECTING PEOPLE

“デザインから最終製品まで従来2週間必要であったモジュールが1日で開発できる。
http://www.fuji-setsu.co.jp/files/Nokia.pdf

EADS

“モデリング言語に組込まれるルールによりデザイン段階でエラーの要因が排除されるので自動生成されるコードの品質は高くなる”
http://www.fuji-setsu.co.jp/files/MetaEdit_EADS.pdf

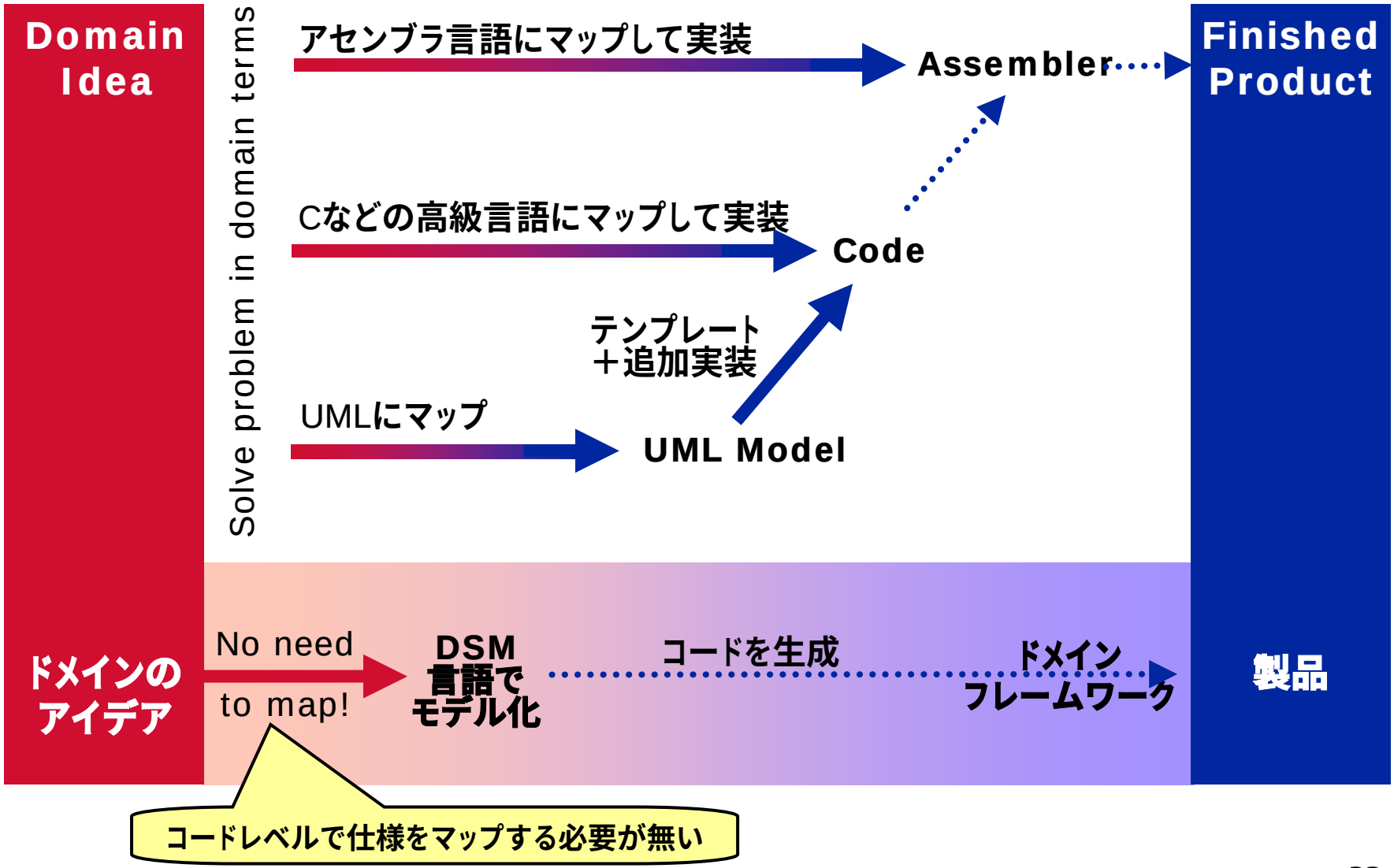
Building your modeling language

専用のモデリング言語の構築

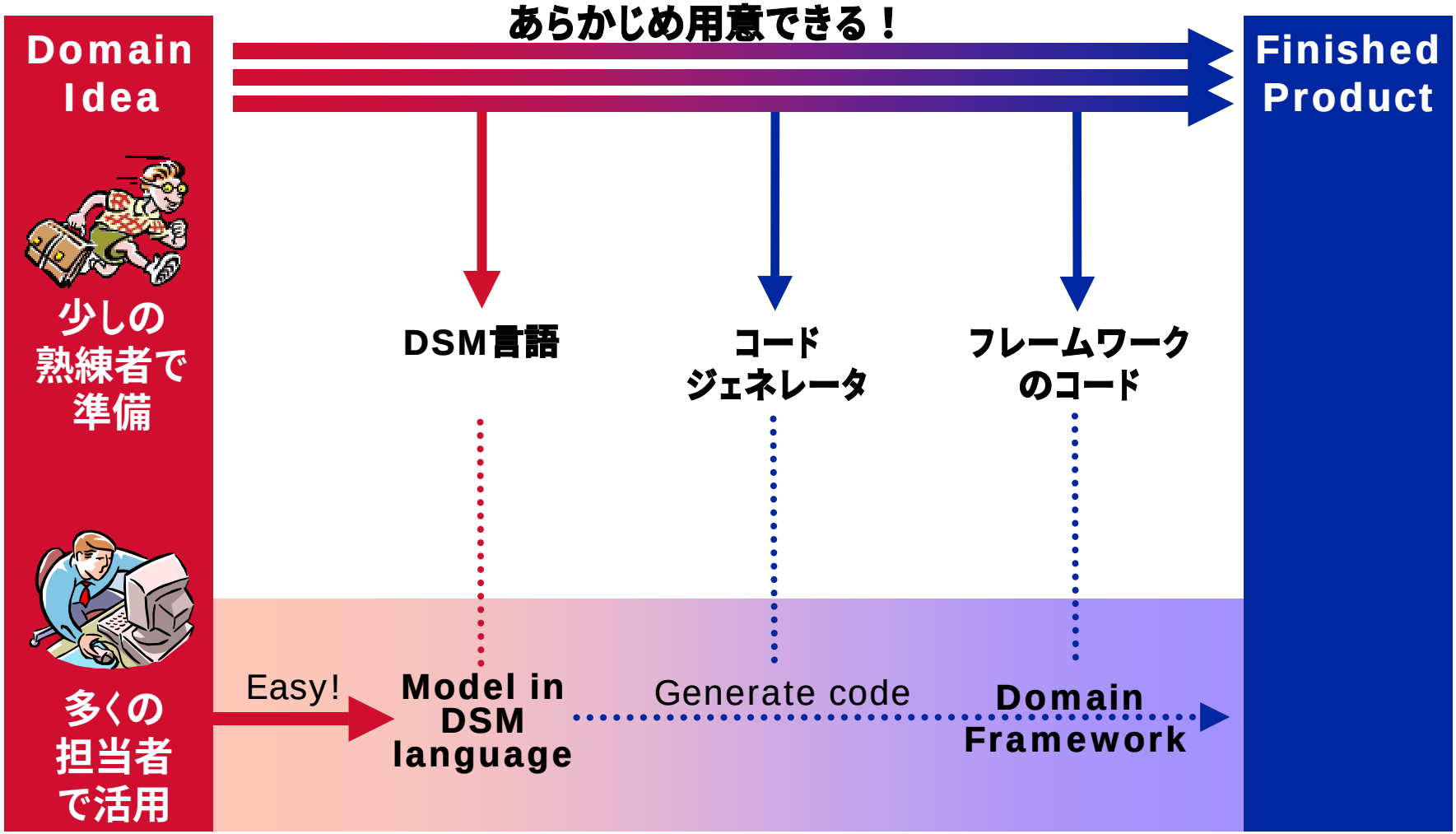
Identifying language concepts　言語のコンセプトを特定
Metamodeling　メタモデリング (モデル言語開発)
Language rules　モデリング言語のルール
Language notations　モデリング言語の表記

http://www.fuji-setsu.co.jp/files/jp-how2start.pdf

製品機能をモデリング 対 コードレベルでモデリング



DSM環境を構築・実装する方法



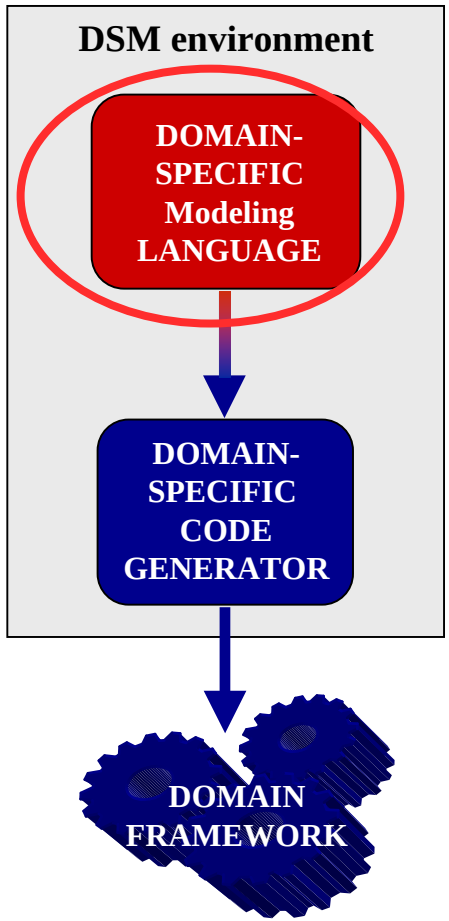
DSM 環境構築の手順

1. 抽象概念を特定する
   - コンセプトとそれらの連携について
2. メタモデルを定義
   - 言語のコンセプトとそれらのルール
3. 表記を作る･持たせる
   - モデルを表現･描写するもの
4. コード生成機能を定義
   - モデルから様々な成果物を生成させる（コード、デザイン、モデル解析結果、モデル評価の尺度、別モデルへの変換　など）

- 既存のコンポーネントやライブラリを洗練して利用する
- イテレーティブなプロセスで。サンプルを作ってみて確かめる
  - 一部分のみメタモデル化して、アプリモデルを描いて、ジェネレータを設定して、メタモデルを拡張して、さらにモデルを描いて、、、　と繰返して徐々に進化させる

モデリング言語の構築

- ＤＳＭ環境の最も重要な点
  - アプリケーション開発者に使用されること
  - ジェネレータとフレームワークは殆ど隠蔽されること
- 多くの場合、慣れ親しんだモデリングの考えを採用
  - ステートマシーン
  - フローモデル
  - データ構造　等



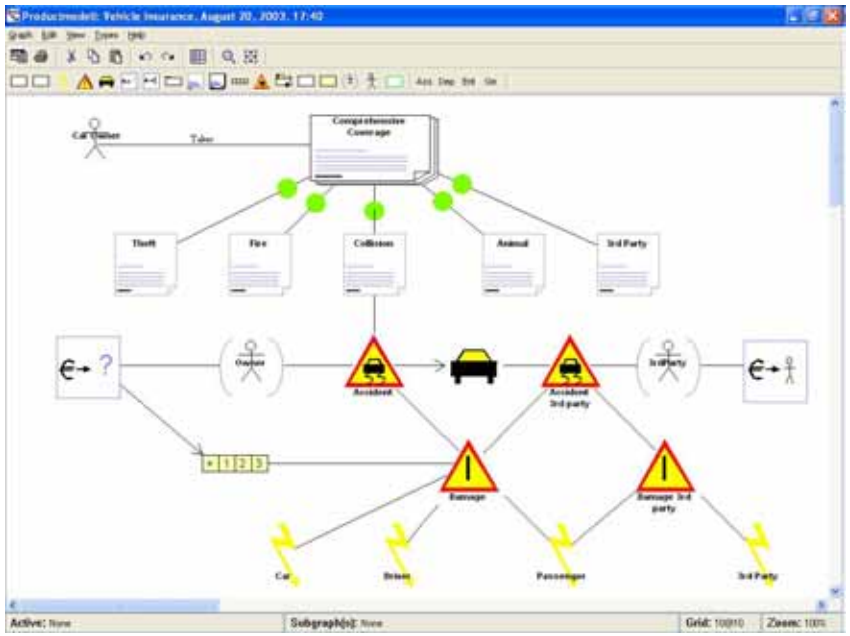
コンセプトを特定する方法・手掛かり

- DSMの始め方は？
  - 初心者には困難ですが
  - 20の事例から得られた取り組みに関する考察を紹介

- Initial analysis suggested five approaches:
  1. ドメインの熟練者のコンセプトを採用する
  2. 生成する成果物（言語）をベースにする
  3. 目に見える構造に着目する
  4. システムのルック・アンド・フィールから
  5. 変動点（バライアビリティ）から

1．ドメインの熟練者のコンセプトを採用する

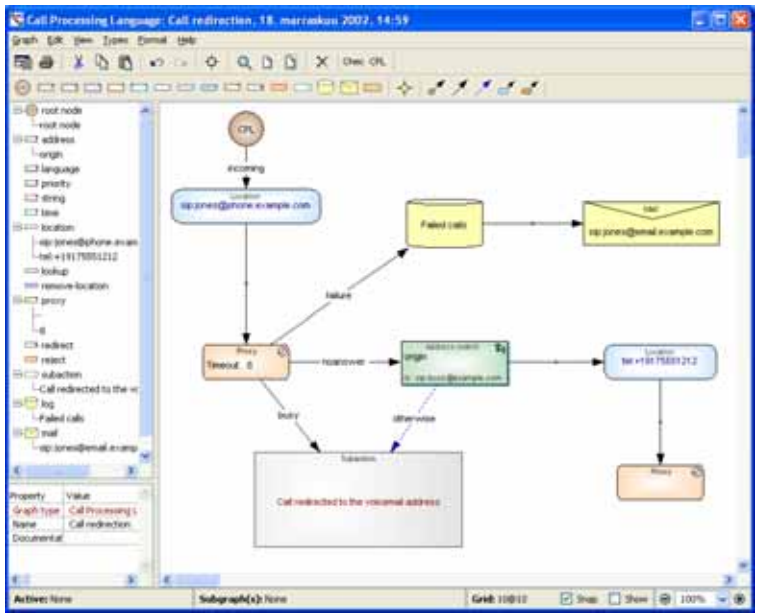
- ドメインの熟練者が考えるコンセプトを採用する手段
- シンプルなコード生成となり易い
- プログラマーでなくても活用できる



保険の構成設定言語/J2EE

2. 生成される成果物（言語）をベースにする

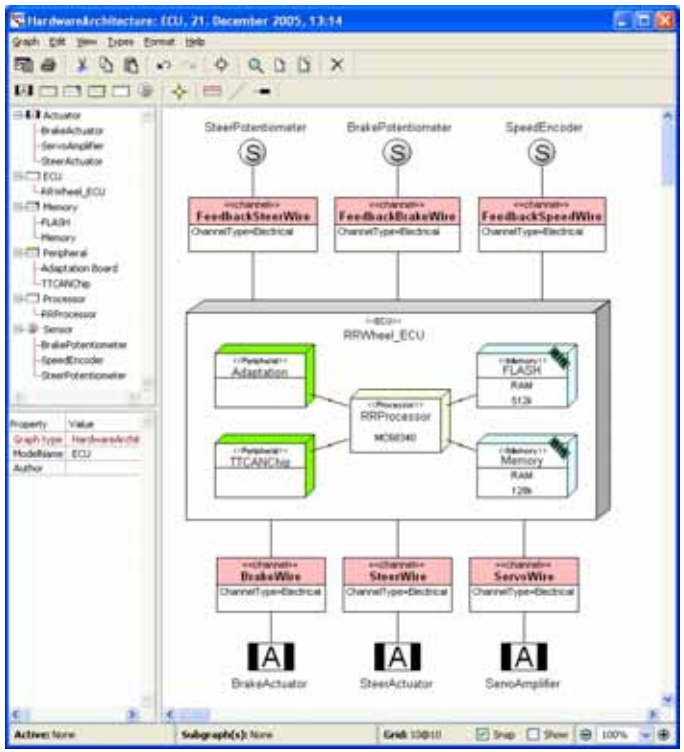
- コードの成果物の観点からモデリング構成を
- 課題：抽象度が下がってしまい易い
  - 結果生産性向上率は下がる
- DSL（ドメインスペシフィック言語）やXMLを対称にする場合には上手く行く
  - As opposed to generic 3GL　第三世代言語に比較して



インターネット通信/CPL

3．目に見える構造に着目する

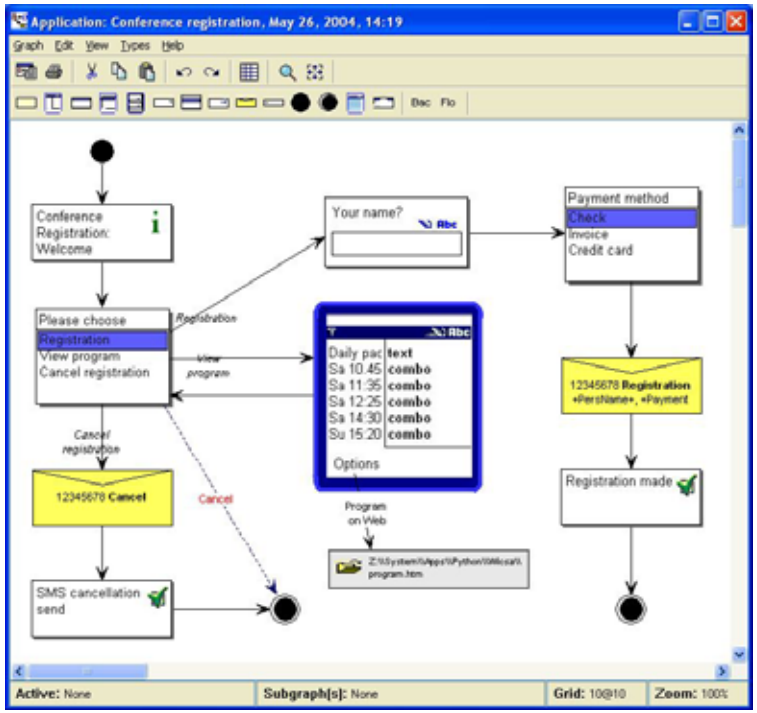
- 有形のシステムに適切
  - ネットワーク、物流システム、HWのアーキテクチャ、電車の制御、FA　など
- 目に見えるドメインのコンセプト
  - 特定し易い
  - 抽象度が高くなる
- コード生成のための
  他のモデルにもリンクできる



自動車の
HWアーキテクチャモデル

4. システムのルック・アンド・フィールから

- 最終製品から直接得られる情報
  - UI など
- ステートマシンなど　Model of Computation
  - データフロー、コントロールフロー、デシジョンツリー
  - Power of relationships　(プロパティを設定してリレーションシップを強化できる。状態遷移のトリガ情報、コンディションや起動させるアクションなど)
- ドメインコンセプトが見える
  - 特定し易い
  - 抽象度を上げられる
- ドメインフレームワークによってコードを隠蔽できる
  - モデル内にコードを書かない
  - 万が一必要であったとしても



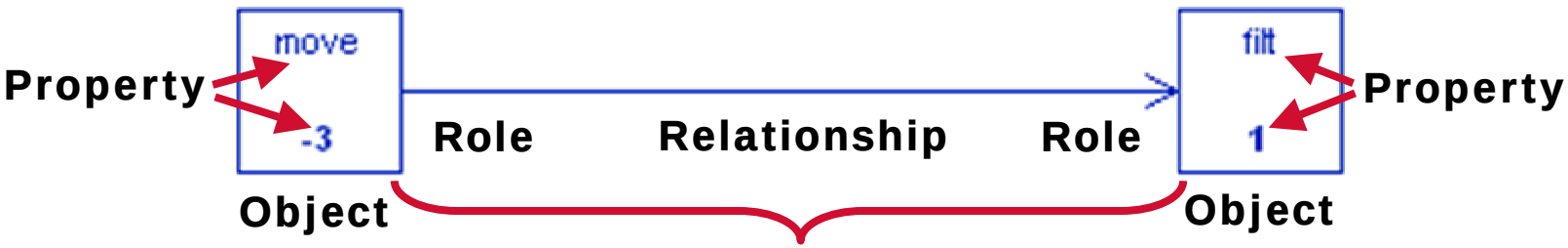
スマートフォン/Python

5. 変動点（バライアビリティ）から

- 言語のコンセプトを変動要素から
- モデラーはバリアントの選択
  - 構成、依存・排他関係、データ（パラメータ）など
- Infinite variability space (Czarnecki)　無限のバライアビリティスペース
  - フィーチャツリー構造では表現されない、あらゆるファミリーの可能性を提供できる
    （http://www.metacase.com/blogs/jpt/blogView?showComments=true&entry=3388313102）
- 一般にバライアビリティ解析をベースにDSMを構築することは容易ではない
  - バライアビリティはUI、コントロールなど様々な箇所に存在し複雑に入り組んだドメインを構成するので。言語が一旦出来ればモデリングはシンプルになる
- 将来に渡ってバライアビリティを予測できることが求められる⇒高いレベルの抽象度で

メタモデル内にコンセプトを定義する

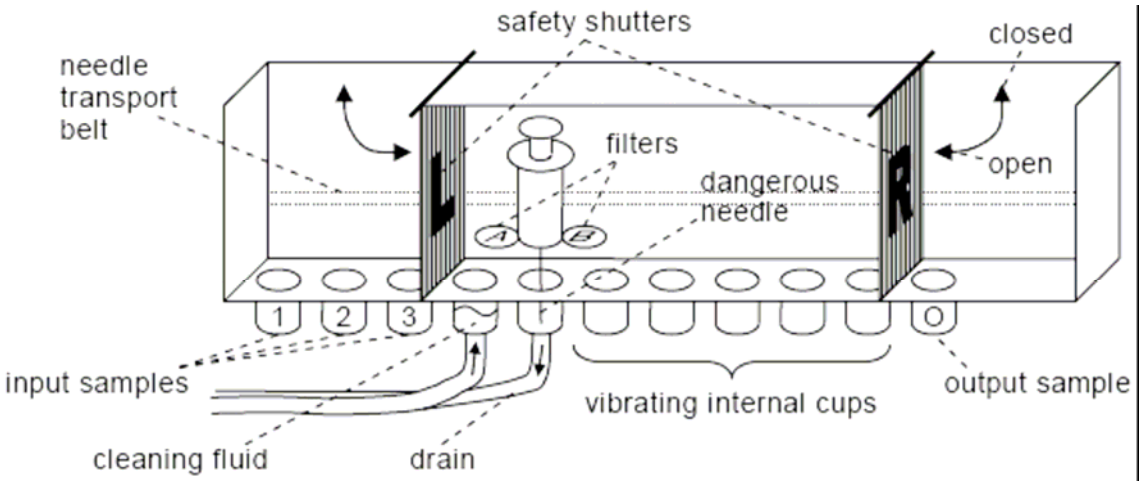
- メタモデルにより言語を形式化
- メタモデリングのツールは言語をテストできること (イテレーティブに)
- メタモデリング (モデル言語構築) でドメインのコンセプトは、以下の言語のコンセプトにマップできる
  - A **object** モデリングのオブジェクト
  - A **relationship** オブジェクト間のリレーションシップ
  - A **role** オブジェクト接続のためのロール
  - A **property** モデルエレメントのプロパティ
  - モデルの階層
  - モデルエレメントのリンク



Small example:
医療機器　ミキサー

例：医療機器 ミキサー　Medical mixing machine

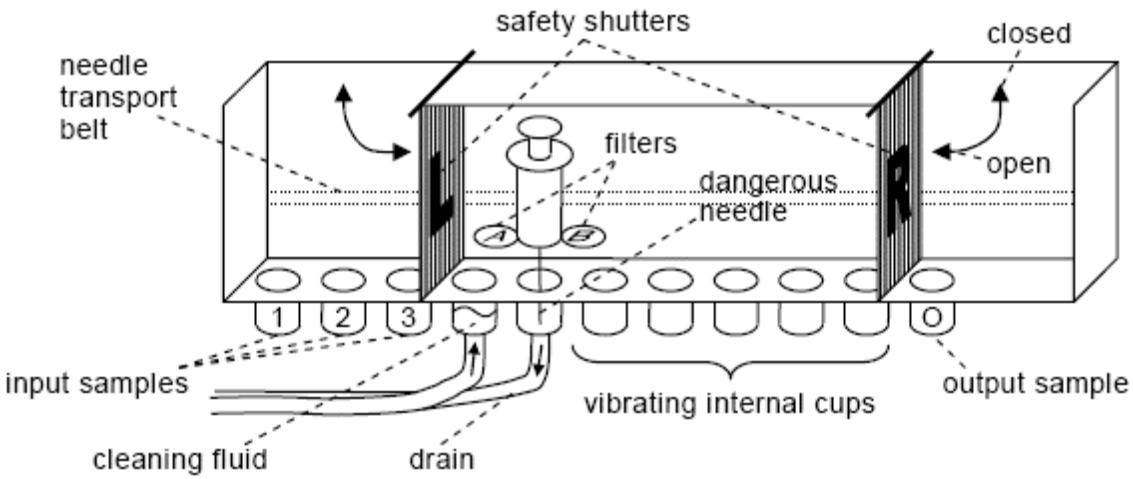
- 単一種の機械
- 製品バリアントはソフトウエアの構成で：混合処理方法で変える
- プラットフォームは11個のカップと注射器からなる
- マニュアル実装されるコードの品質が課題
  - Syringe broken, operator died, patient treatment error



http://www.fuji-setsu.co.jp/products/MetaEdit/MetaEdit_hands_on.html

生成される言語から始めてみると

- 言語のコンセプトを何処から得るか？



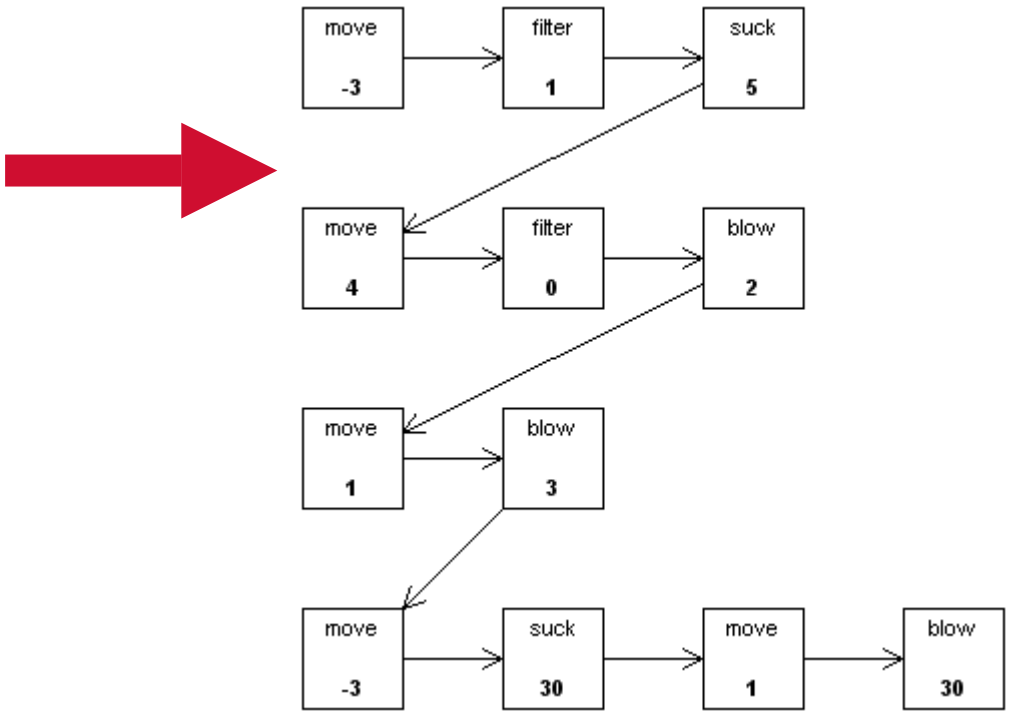
“filter A” を使い、”input sample” のカップ2から5滴分吸い上げ、カップ6に2滴、カップ7に3滴入れ、そして注射針を洗浄する”

```
01 move(-3); filt(1); suck(5);
02 move(4); filt(0); blow(2);
03 move(1); blow(3);
04 move(-3); suck(30);
05 move(1); blow(30);
```

Version 1

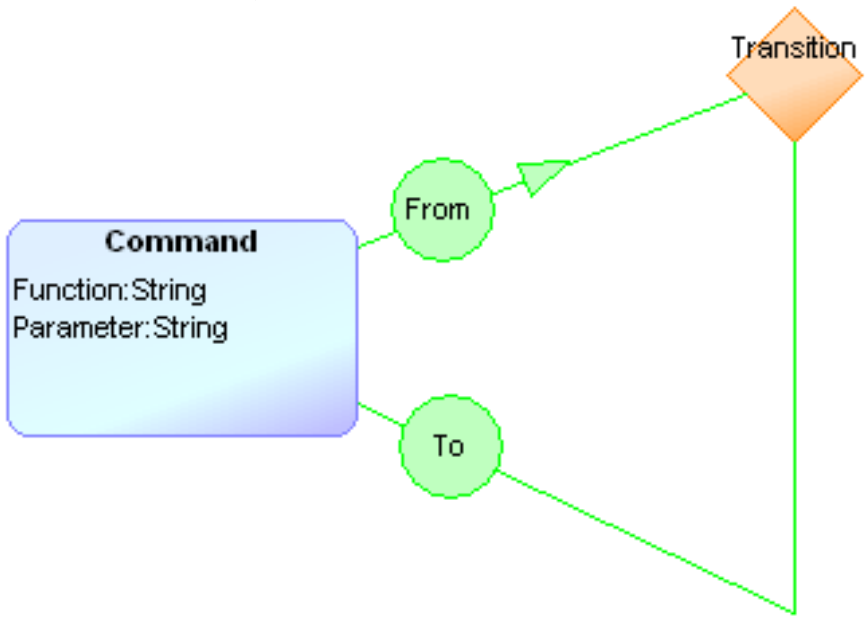
- 各コマンドは言語のエレメントにマップできる

```
move(-3);
filt(1);
suck(5);
move(4);
filt(0);
blow(2);
move(1);
blow(3);
move(-3);
suck(30);
move(1);
blow(30);
```

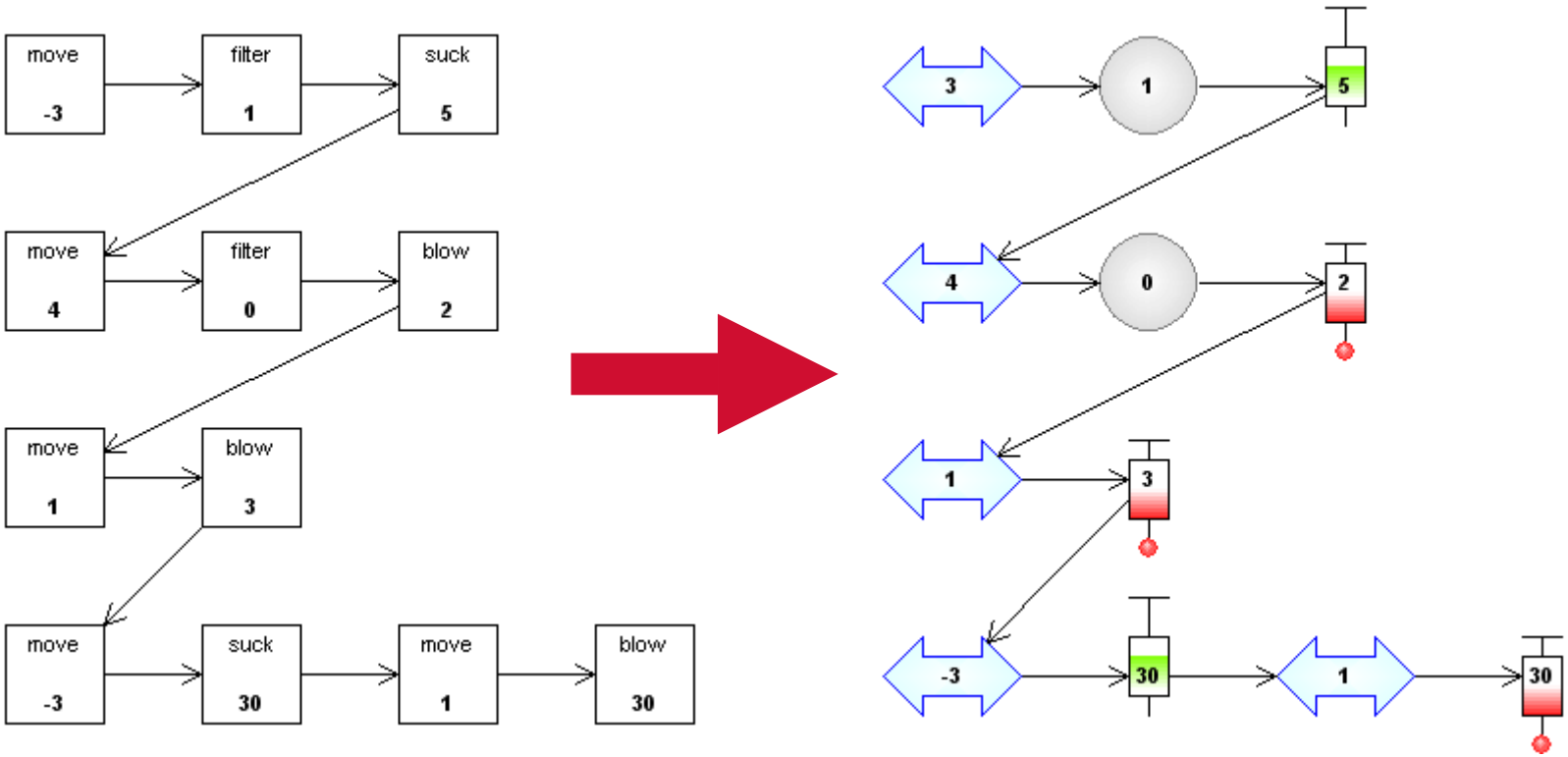


Version 1:メタモデル

- Command オブジェクトは、パラメータを用いたファンクションコール (move、filter、suckなど)
- Transitionリレーションシップや From、Toロールは、命令の処理手順などを定義

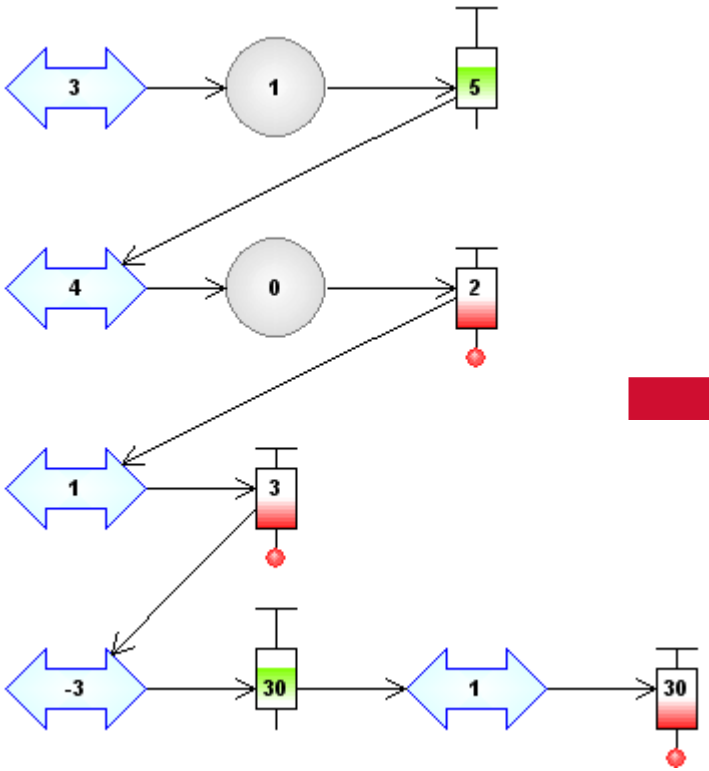


Version 1:
明確で意味のあるシンボルを持たせる



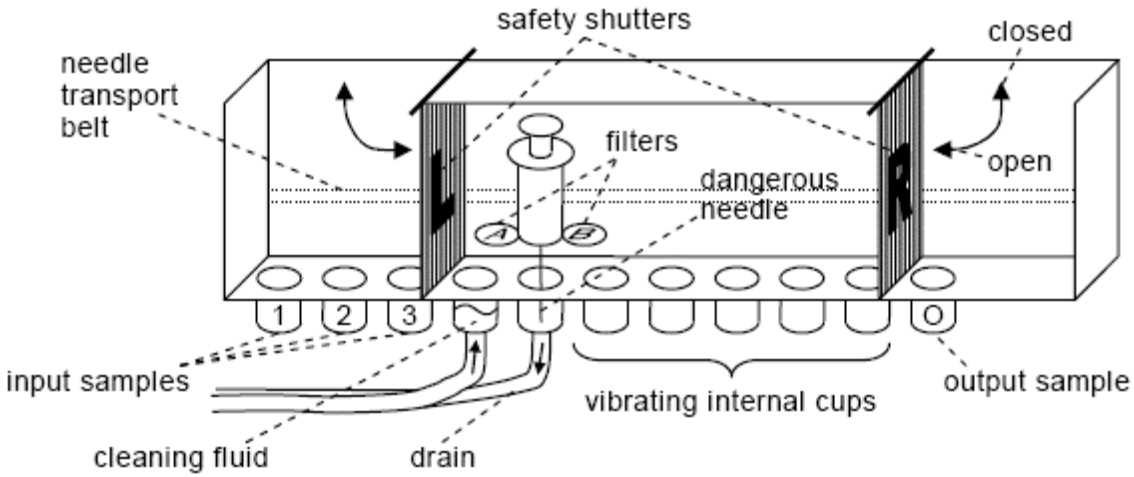
Version 1: コード生成の動作

- **各**Objectからコマンドを書き出す
- リレーションシップ（->）の順に次のObject




```
move(-3);
filt(1);
suck(5);
move(4);
filt(0);
blow(2);
move(1);
blow(3);
move(-3);
suck(30);
move(1);
blow(30);
```

言語のコンセプトを何処から得るか？



“filter A” を使い、”input sample” のカップ2から5滴分吸い上げ、カップ6に2滴、カップ7に3滴入れ、そして注射針を洗浄する

```
01 move(-3); filt(1); suck(5);
02 move(4); filt(0); blow(2);
03 move(1); blow(3);
04 move(-3); suck(30);
05 move(1); blow(30);
```

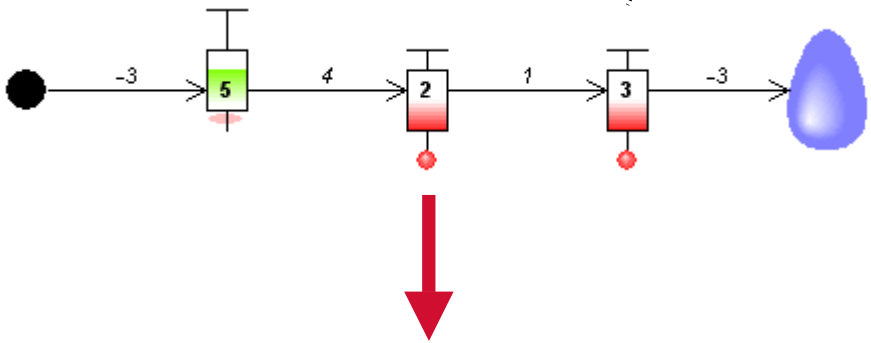
Version 2: より高い抽象度

“filter A” を使い、”input sample” のカップ2から5滴分吸い上げ

カップ6に2滴

カップ7に3滴入れ

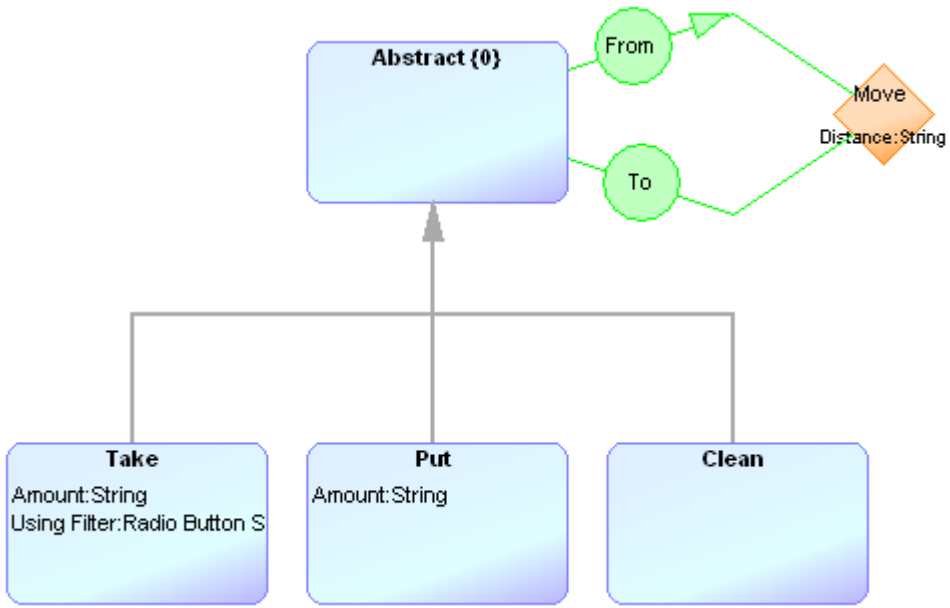
そして注射針を洗浄する



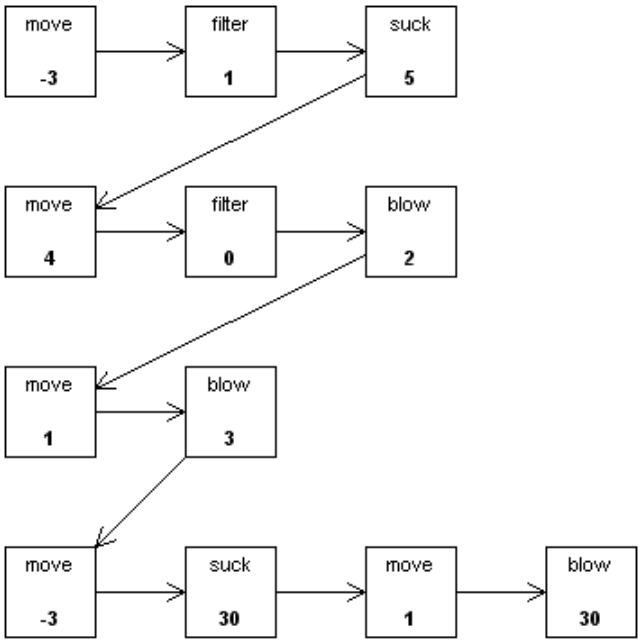

```
01 move(-3); filt(1); suck(5);
02 move(4); filt(0); blow(2);
03 move(1); blow(3);
04 move(-3); suck(30);
05 move(1); blow(30);
```

Version 2: 新しいメタモデルは

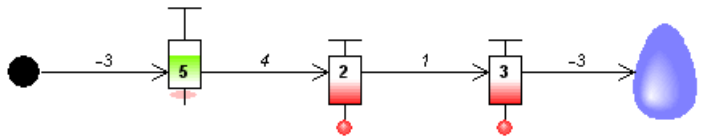
- **モデリングのオブジェクト**: Take, Put, Clean
- **リレーションシップ**: Move
- **ルール**:
  - **フィルターを介してのみ**Take (**吸上げ**) **の処理ができる**
  - **注射針洗浄の詳細は隠蔽**



そのモデリングの効果は



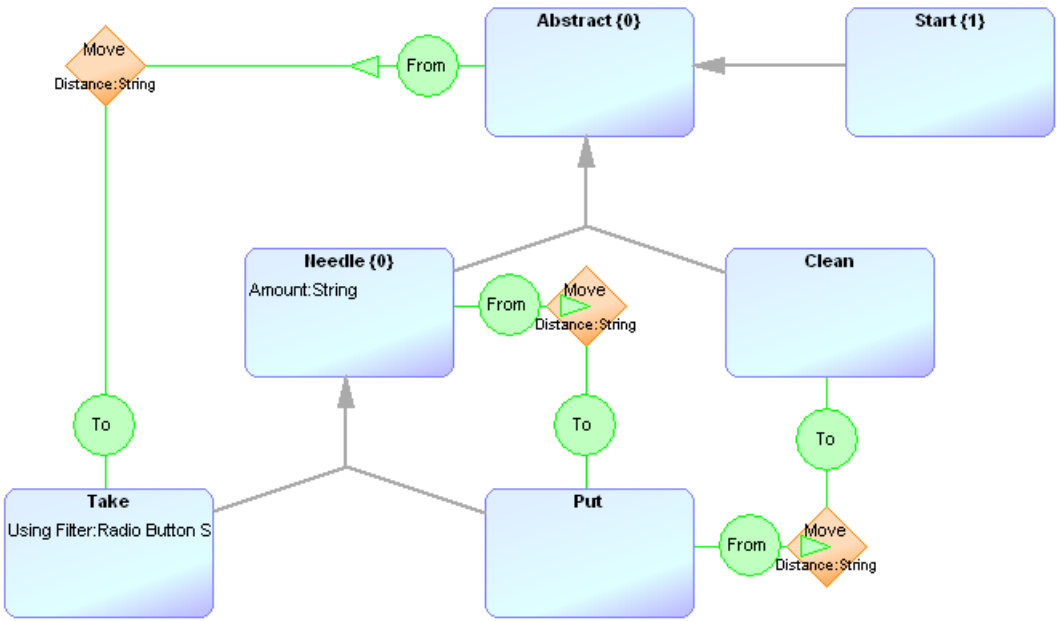
- 12 objects
- 11 relationships
- 24 properties
- **47 elements in total**
- **全部で47エレメント**



- 5 objects
- 4 relationship
- 7 properties
- **16 elements in total**
- **全部で16エレメント**

Version 2: ルールを追加すると

- スタートのための単一箇所を指定するスタートオブジェクト
- Takeは何処からでも行える(Abstract)
- PutはTakeかPutの後に起こり得る
  - TakeやPutを行うNeedleスーパタイプを介して
- Clean はPutの後のみ



ルールについて

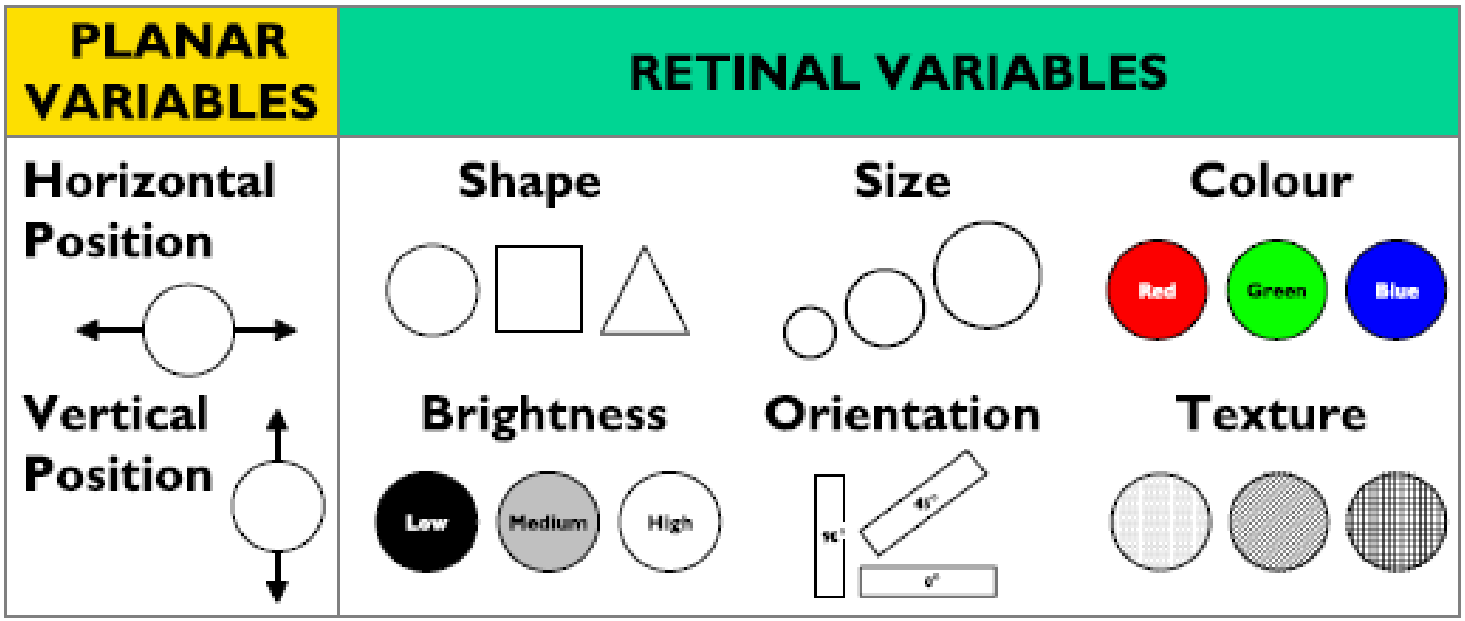
- 言語内にドメインのルールを持たせる
  - 間違ったモデルが描けないように
  - データの不足を通知させる
  - モデルの一貫性を保つ
- 違反、間違いをモデラーに見えるように･わかるような工夫を取る
  - ダイアログでモデリング中に
  - 別のモデルチェック画面で
  - エラー、データの不足箇所をハイライトして通知する
- 別のモデルチェックを走らせることもできる
  - コード生成前に
  - モデルチェックの結果をドキュメント化
  - バージョン管理前に

表記を定義する

- 使い易いモデリング言語として」採用されるために必須
- シンボルは箱型から写真まで様々
  - 実際のドメイン上の形態に似ていることが望ましい
  - 最悪なのは全てを箱型の中に持たせて、テキストで違いを現そうとすること
  - デザイン情報はスペースを侵食するので、妥協も必要
- 表記を一から作ろうとしないこと
  - 良く知った既存のエレメントを採用すること（and, or, start, stop など）
- ヒント：モデリングするユーザに表記定義の相談をする
  - 大きな効果を期待できる
  - モデルを読む（見る）ことになる人のことも留意する
    - マネージャ、テスト担当者、顧客、セールス担当も！

視覚的に異なったものを活用する

- シンボルには視覚的に異なる様々なものを採用すると良い

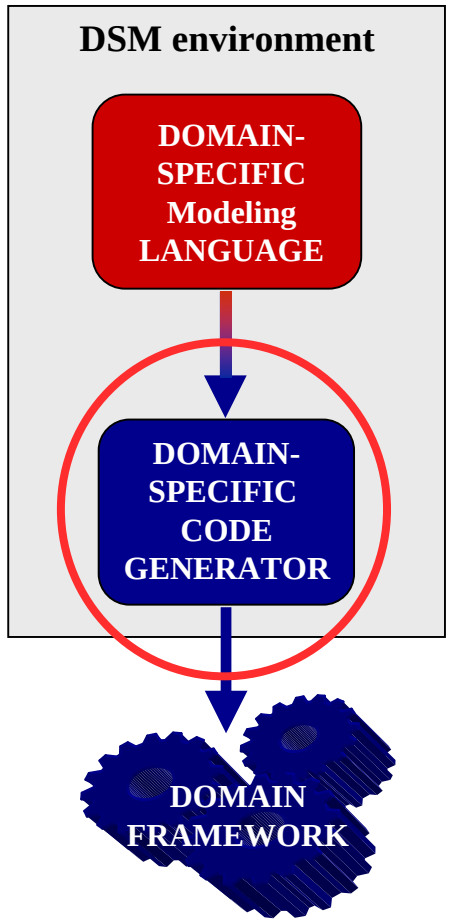


Building a Generator
ジェネレータの設定 (構築)

How generators work? **ジェネレータの動作について**
How to design a generator? **ジェネレータの設計方法**
Generator example **ジェネレータの例**

ジェネレータ

- ジェネレータで、モデルから、目的の出力に変換する
  1. モデルをくまなく解析 → モデルをナビゲーションする
  2. 必要な情報を取り出す → モデル内のデータにアクセスする
  3. コードに変換する → 変換のためのセマンティックスとルール
  4. 幾つかの出力フォーマットを使用 → 出力フォーマットを定義可能
- Three generator approaches
  - フィックス (固定型) (Simulink、Labview など)
  - カスタマイズ
  - ドメインスペシフィック ジェネレータ



ジェネレータの設計方法　1

- **完全なコード生成（１００％を目標に）ジェネレータを作成**
  - **生成されたコードを修正しない**
    - **コンパイル後にアセンブラを修正しますか？**
  - **代わりにジェネレータやフレームワークを修正する**
    - **ラウンドトリップに起因する問題が無くなる**
    - **アセンブラコードを修正して、結果をCに反映させるようなことをしますか？**
- **出来るだけ小さなコードを生成するようにする**
  - **グルーコード（接続用のコード）だけにする、後はドメインフレームワークとプラットフォームに任せる**
- **読みやすいコードを生成する**（“good looking”）
  - **コードをデバッグしたり、シミュレータで実行したり、ジェネレータを修正する時に必要になる**
  - **コーディングスタンダードに準拠する、コメントを含める、コードを生成したモデルを参照する為のデータを埋め込む**

ジェネレータの設計方法 2

- 専用のジェネレータを作成
  - 汎用的なジェネレータを作ろうとすると大抵失敗する
- ジェネレータを出来るだけ単純にする方法
  - 製品ごとで変わる部分 (変動性) はモデル言語側へ
  - 低レベルな製品間で共通する部分 (共通性) はフレームワークへ
- 変更を反映する為にジェネレータをモジュール構造にする
  - モデリング言語、生成されるファイル、モデリングコンセプト等をベースにしたジェネレータ構造
  - 共通の要求に対して共通のジェネレータサブルーチンを使う

"Here's one I made earlier"

- マニュアルで動作するコードを書いてみる
  - マニュアルで上手く出来たことだけ自動化できる
  - 小賢しいプログラムではなく、単純なプログラムを心掛ける
- システムそのものを表現するモデルを構築する
  - 主要な3～4のオブジェクトタイプを1つのインスタンスにする事を目指す
- ジェネレータの観点でコードを解析する
- ジェネレータにとって最も簡単なのは固定値と固有値
  - 常に同じ出力になる固定テキスト
  - モデルの名前など直結した値
- Between the extremes is the meat of the generator:

  ジェネレータの肝:
  - モデルの値に依存した、条件セクションや呼び出し処理
  - 繰り返しセクション(モデル要素毎に一回等)

autobuild の重要性

- Autobuild = **モデルから直接テスト実行**
  1. モデル、サブモデルから全ファイルを生成
  2. ファイルに対してコンパイル実行
  3. アプリを起動する･実行する
     - 必要ならエミュレーション環境で
     - 実行状況をモデル視覚化させることもできる (ターゲットに依存)
- **工数削減、マニュアルによるエラーの回避、一貫性を維持できる**
- **コンテキストスイッチから脳を開放すること！**
  - モデルはアプリケーションを高いレベルで記述
  - コードのことを考慮に入れないで済む

Domain Framework
ドメインフレームワーク

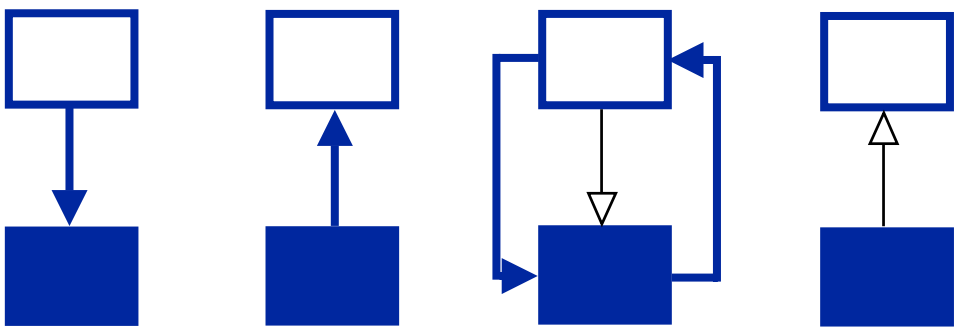
http://www.fuji-setsu.co.jp/files/Framework.pdf

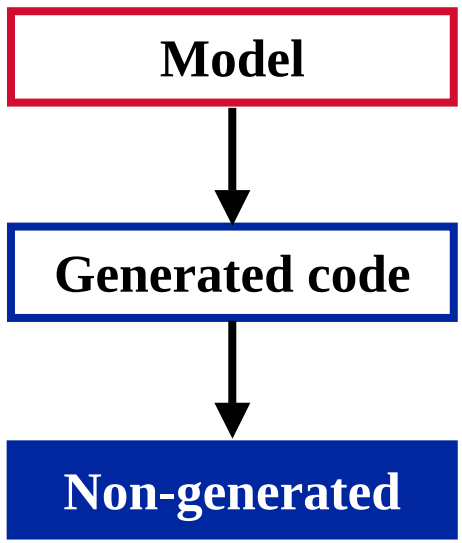
なぜドメインフレームワークなのか？

- 実装内における重複箇所を改善する（共通性）
  再利用の自動化。手作業で書かれたコード内の重複部分に対して
- ＤＳＭによって得られる広い観点・長期的視野
  - １人の開発者が1アプリケーションにファイルダイアログ呼び出しを2つ程度書くような場合は、
    - フレームワーク関数を用意するに及ばない－＞コピー＆ペーストで十分
  - ＤＳＭでは、1人の開発者が1つのジェネレータを書けば、１００のアプリケーションが作れるようになる
    - 大きな効果がフレームワーク関数により得られる：シンプルでコンパクトになる
- プラットフォームの詳細を隠蔽できる
  - プラットフォームの進化やバグによる影響を回避する事が出来る
  - フレームワークが単一のプラットフォームのみならず複数で利用できるようになる
    - モデル、ジェネレータ、生成されるコードは変わらない
    - フレームワークへのインターフェースは変わらない
    - 複数のフレームワークバージョンを切り替えるだけ

生成されたコードと他のコードの結合

- 異なったレベルのコードが統合される
  - 他で生成されたコード
  - ドメインフレームワーク、部品、プラットフォーム関数
  - 手書きのコード
- 生成されたコードとそれ以外のコードの分離
  - 別のファイルにする（或いはファイル内で、別のセクションにする）のが最良
- 生成されるコードは
  - 既存のコードを呼んだり、データ構造のインスタンスを生成できる
  - 既存のコードから呼ばれることが出来る
  - 既存のコードのサブクラスになれる
  - 基本クラスを形成できる





Where to apply – and how?

何処に？　どうやれば？

Where to apply? どこに活用できる？

- **繰返し行われている開発作業**
  - 作業の大半が既存製品のそれと同様 (あるいは複数製品が並行して開発されている)
- **ドメインの熟練者が必要**
  - プログラマーで無くても関わることが出来る

- **一般に以下が該当する:**
  - 製品ファミリー (プロダクトライン開発)
  - プラットフォームベースの開発
  - 構成による開発
  - ビジネスルールの定義･設定
  - 組込みデバイス

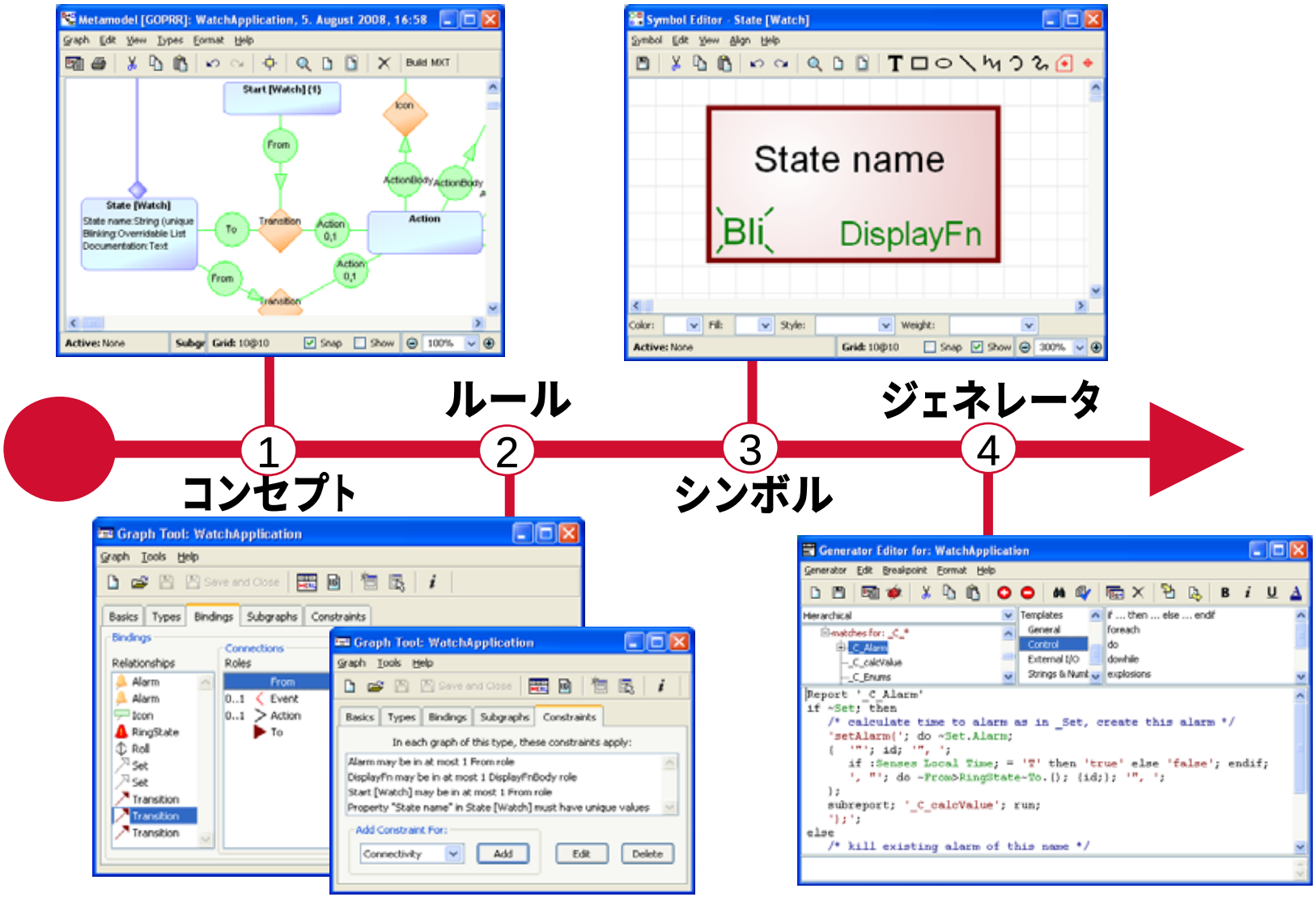
DSMに適したドメインを選択する

- 複数の候補があるならベストなものを最初に選ぶ

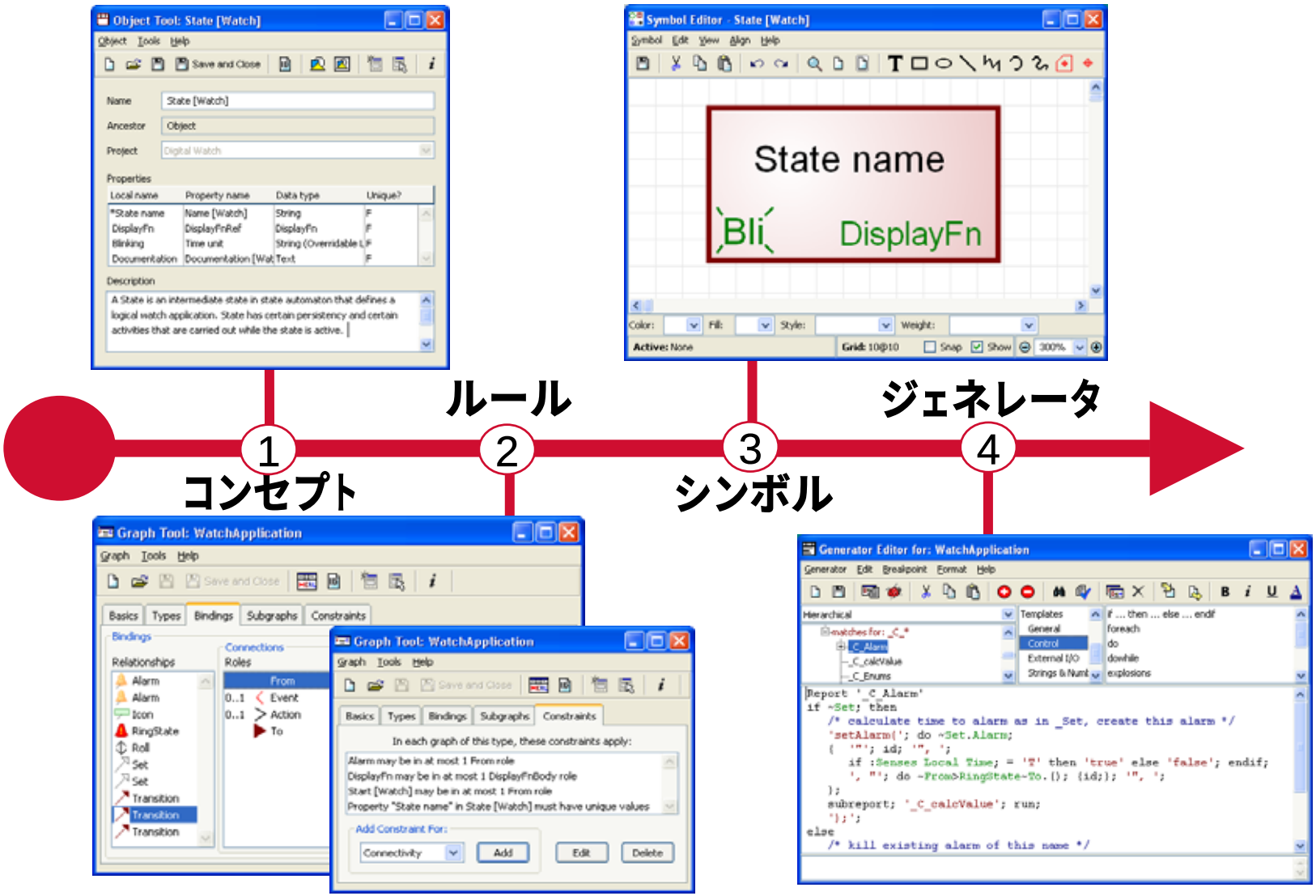
組織上の決定要因:

- 社内でターゲットとするビジネス領域を熟知しているか？
- 多くの開発者がいますか　？
- 社外組織との関係は低いか？
- 他の候補となるドメインと関連するか？
- 顧客ごとのカスタマイズの範囲は？

MetaEdit+ による DSM構築

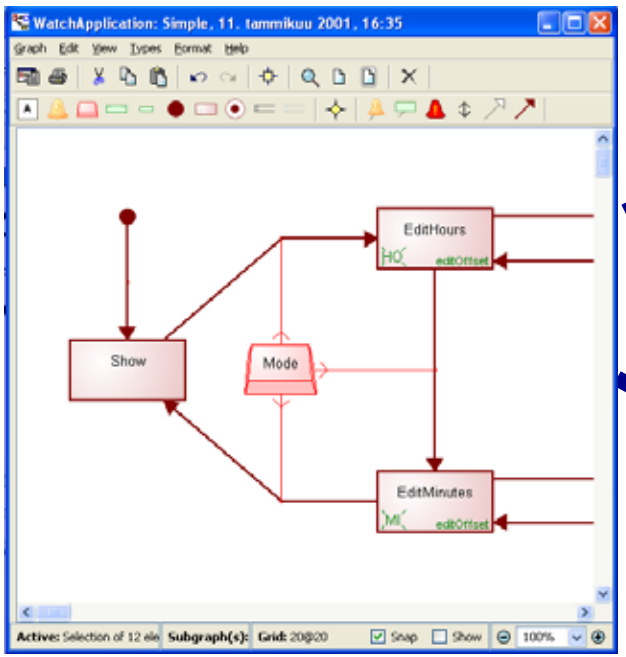


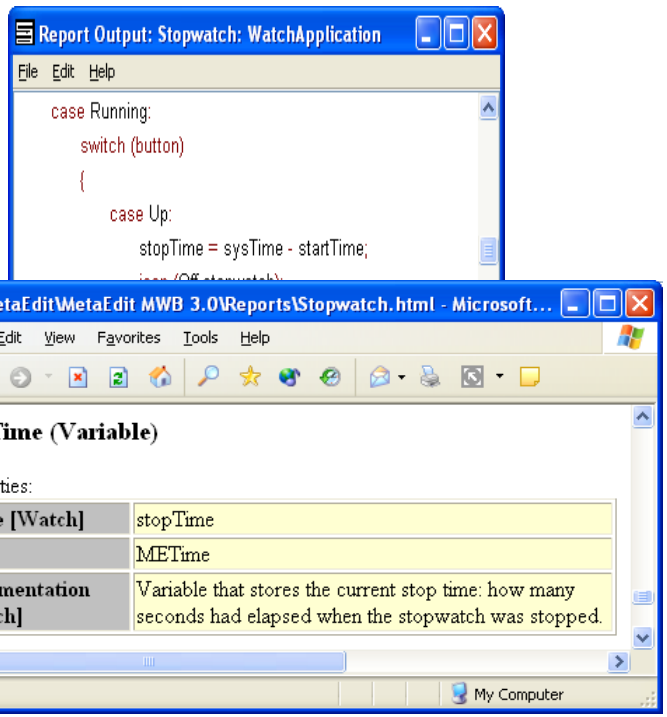
MetaEdit+ による DSM構築



MetaEdit+： プログラミングの必要なくDSM環境を構築できる

- エディタ (ダイアグラム，マトリックス，テーブル形式で)，ブラウザー，コードジェネレータ,マルチユーザ，マルチプロジェクト，マルチプラットフォーム
- 簡単かつ安全に言語をメンテナンス (修正･追加) 、進化させることが出来て、共有できる。
  - 最新のモデリング言語を共有し、既存モデルをアップデートできる





DSMを構築し活用するためのツール

Eclipse EMF, Eclipse GEF

- DSMに必要なツールを得るための方法　6通り
  1. 一から専用のモデリングツールを実装する
  2. フレームワークをベースにして専用のモデリングツールを実装する
  3. メタモデル、モデリングツールのスケルトンを生成、コードを加える
  4. メタモデル、フレームワーク上に完全なモデリングツールを生成
  5. メタモデル、ジェネリックなモデリングツールのコンフィグレーションを出力
  6. モデリングとメタモデリングの環境を統合する　(MetaEdit+)
- **良いツールなら数週間で**
  - モデリングツールとジェネレータの構築は半自動
  - DSM構築を支援し成功に導く
  - ドメインのデザインを介してDSMをテストできる（イテレーティブに）
- **良いツールはDSMLを変更して、その変更を反映させられる**
  - モデリングツールにも
  - 既存のデザインモデルにも

http://www.metacase.com/ja/featurelist.html

DSMを構築し活用するためのツール

- DSMに必要なツールを得るための
  1. 一から専用のモデリングツールを実装する
  2. フレームワークをベースにして専用のモデリングツールを実装する
  3. メタモデル、モデリングツールのスケルトンを生成、コードを加える
  4. メタモデル、フレームワーク上に完全なモデリングツールを生成
  5. メタモデル、ジェネリックなモデリングツールのコンフィグレーションを出力
  6. モデリングとメタモデリングの環境を統合する (MetaEdit+)

Eclipse GMF, Microsoft DSL tools

- 良いツールなら数週間で
  - モデリングツールとジェネレータの構築は半自動
  - DSM構築を支援し成功に導く
  - ドメインのデザインを介してDSMをテストできる (イテレーティブに)
- 良いツールはDSMLを変更して、その変更を反映させられる
  - モデリングツールにも
  - 既存のデザインモデルにも

http://www.metacase.com/ja/featurelist.html

DSMを構築し活用するためのツール

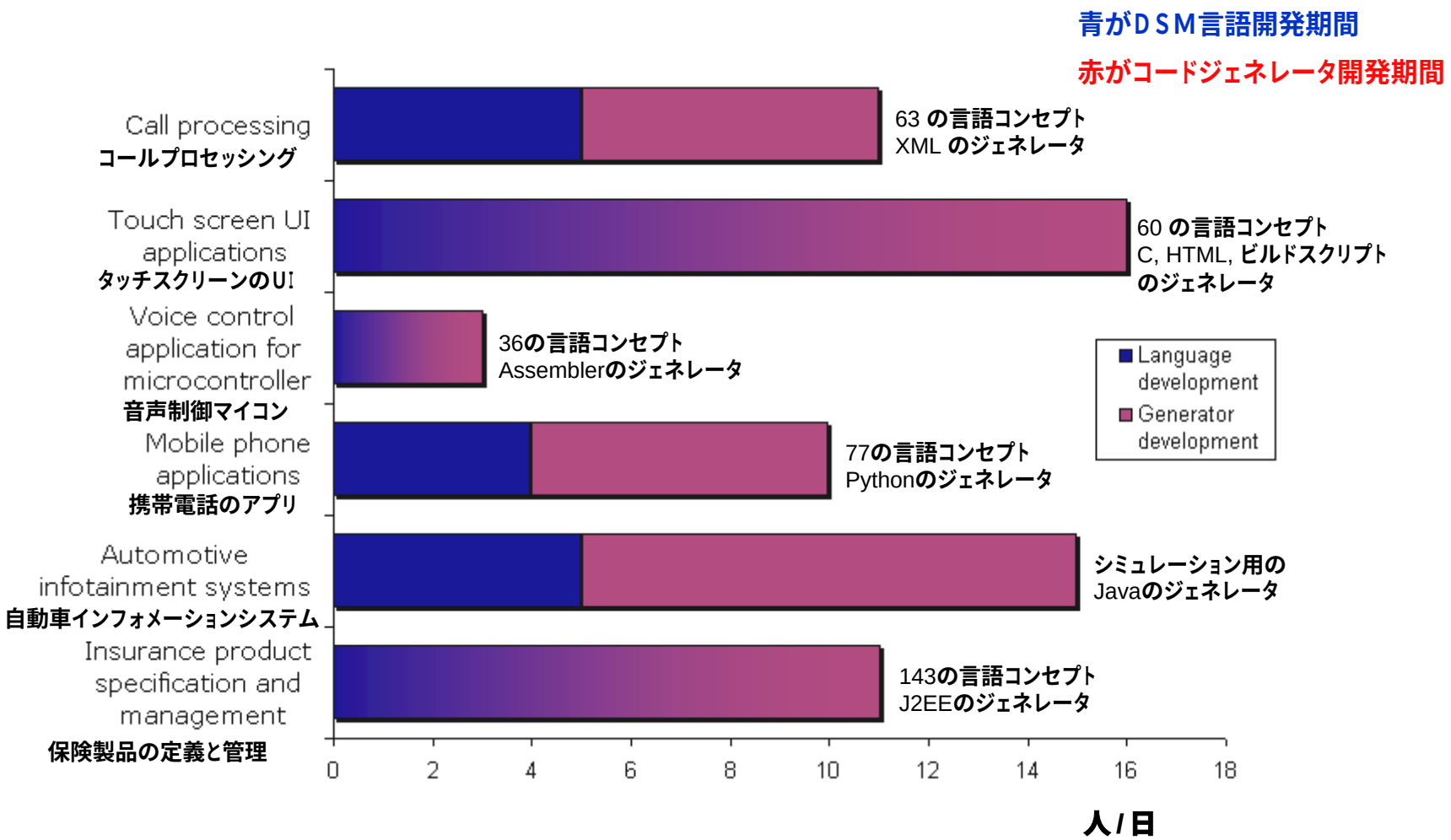
- DSMに必要なツールを得るための方法　6通り
  1. 一から専用のモデリングツールを実装
  2. フレームワークをベースにして専用のモデリングツールを実装する
  3. メタモデル、モデリングツールのスケルトンを生成、コードを加える
  4. メタモデル、フレームワーク上に完全なモデリングツールを生成
  5. メタモデル、ジェネリックなモデリングツールのコンフィグレーションを出力
  6. モデリングとメタモデリングの環境を統合する　(MetaEdit+)

GME 、MetaEdit 1.0旧バージョン

- 良いツールなら数週間で
  - モデリングツールとジェネレータの構築は半自動
  - DSM構築を支援し成功に導く
  - ドメインのデザインを介してDSMをテストできる（イテレーティブに）
- 良いツールはDSMLを変更して、その変更を反映させられる
  - モデリングツールにも
  - 既存のデザインモデルにも

http://www.metacase.com/ja/featurelist.html

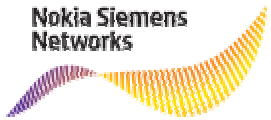
DSM 構築に要した期間



http://www.fuji-setsu.co.jp/products/MetaEdit/blogs.html

What engineers say

“DSM構築に時間･工数はかからなかった。そして熟練者の数週間の貢献により、全ての開発者が熟練者並みに開発できるようになった" Antti Raunio



“MetaEdit+を活用すれば1週間でDSLを構築できた“
Jukka Manninen

“MetaEdit+によるモデリングは快適で非常に簡単。同じ結果をEclipse のGMFで得ることに比較して"
Ulf Hesselbarth

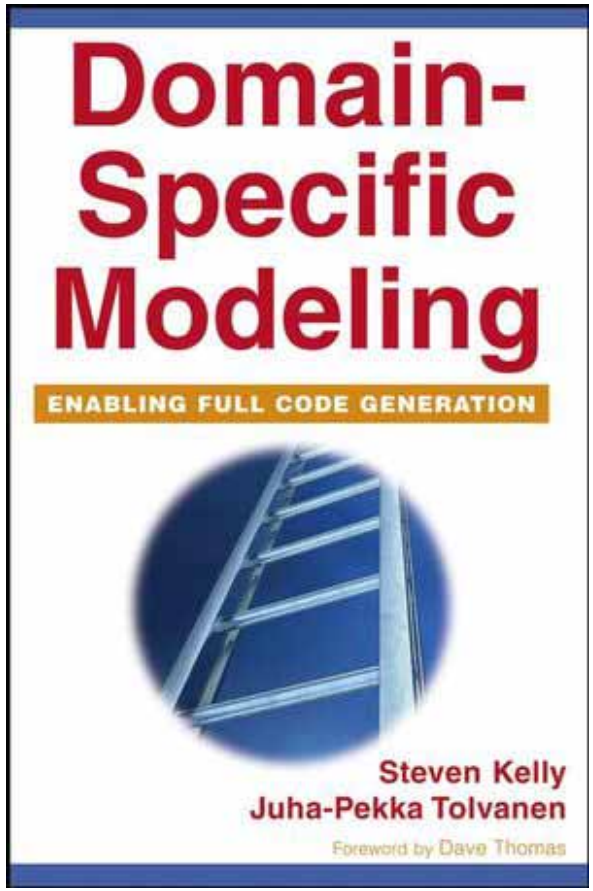
“MetaEdit+ は最も柔軟なツールで専用の言語シンタックスを直ちに定義することができた”
David Narraway

まとめ

- ドメインスペシフィックモデリングで生産性が本質的に改善できる（5～10倍）
- DSMを活用することで熟練者の能力をチームメンバーが最大限に活用できる
- 良い言語開発環境はDSM環境構築の費用対効果が高い
- 実装したものモデル化したもの何であれ生成対象にできる
  - 生成対象にならないのは、まだ書かれていないものだけ
- MetaEdit+は評価され実践で証明されたテクノロジー
  - 数百のDSMを構築した実績
- DSM環境を構築することは素晴らしき喜びになる
  - The ultimate refactoring! 究極のリファクタリング

Further reading

- **Domain-Specific Modeling: Enabling Full Code Generation**
  Wiley, 2008
  http://dsmbook.com



Further reading

IEEE Software 特集記事
ドメインスペシフィックモデリング
Vol. 26, No. 4 July/August 2009

ドメインスペシフィックモデリングの
ワーストプラクティス (良くない事例集)

http://www.metacase.com/papers/WorstPracticesForDomain-SpecificModeling_JA.html

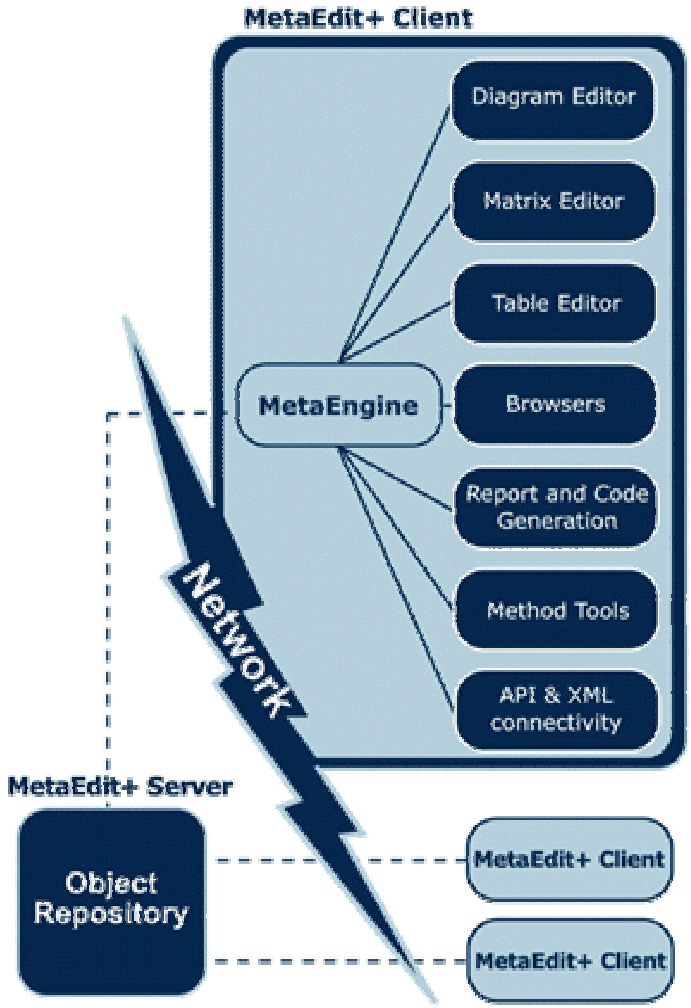
ＤＳＭ言語構築に活用される MetaEdit+

- メタモデリング、モデリング環境、コード生成機能の統合化
  - 完全に融合され、既存ツールチェインと統合できること
- DSM言語を厳密なルール、コンストレインツにより形式化
  - モデル、モデル間の整合性チェック
- 進化を支援できる
  - ＤＳＭ言語の修正･拡張が容易で、既存アプリモデルが破壊されることなくアップデートできる
- マルチDSM言語
- 特定ツールベンダーに固定されない
  - モデルやメタモデルは、ＸＭＬなど標準のファイル形式でインポート･エクスポートできるなど

マルチユーザーとマルチプラットフォーム

- Windows
- Linux
- Solaris
- HP-UX
- Mac OS X

- **ツールの統合**
  1. SOAP／Webサービス／.NETを使ったAPI
  2. XMLファイル
  3. システムコール／コマンドラインでの利用



MetaEdit+ 優位性

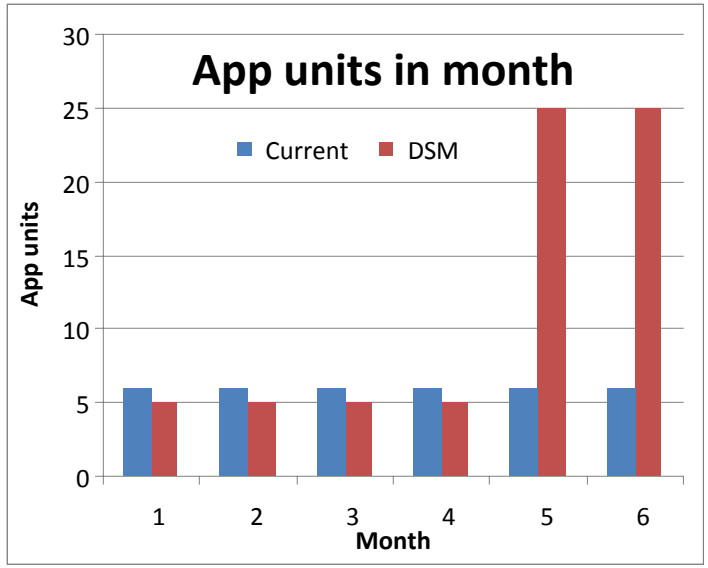
- ＤＳＭ環境構築には、通常２～３週間、大規模システムでも２～３人月。
  従来のモデル駆動開発では準備に少なくとも１～２年はかかると言われています。

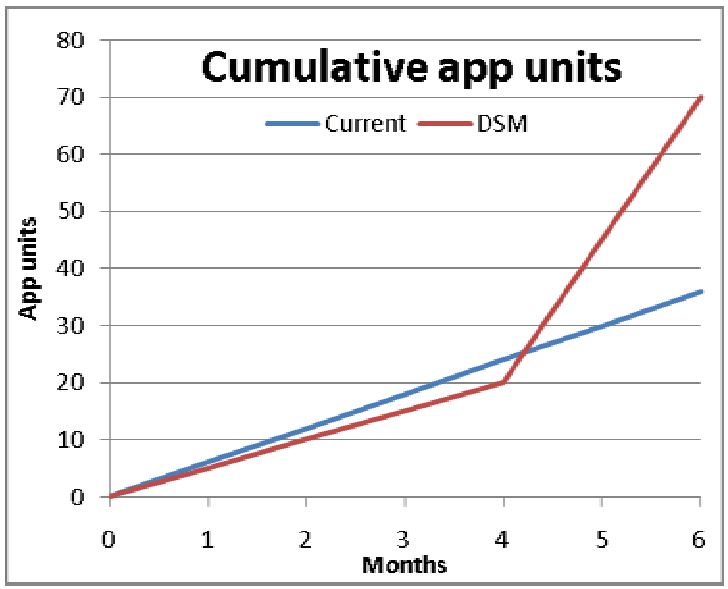
- なぜか？

- 高い抽象度の部品
- プロパティで派生を追加できる
- リポジトリベースで管理されるため、追加･修正などが柔軟

DSMの経済効果

- DSM以前:
  - 6 人の開発者
  - 1ユニット/1人月のアプリケーション開発
- 1～4ヶ月目: ＤＳＭ言語構築期間
  - 1 人がＤＳＭ言語とコードジェネレータを構築
  - 残り5人は従来通りに開発 (1ユニット/月)
  - ➢ トータル２０アプリケーション
    (従来通りに6人でやれば２４アプリケーション)
- 5ヶ月目: ＤＳＭの展開
  - 5倍 の生産性
  - 1 人はＤＳＭ言語のメンテナンス専任
  - 5 人がＤＳＭ活用、 5ユニット/1人月
  - ➢ 45 のアプリケーションユニット
    (従来通り6人でやれば３０アプリケーション)
- 6ヶ月目
  - ➢ 70 のアプリケーションユニット
    (従来通り6人でやれば３６アプリケーション)





ＤＳＭの経済効果: ROI

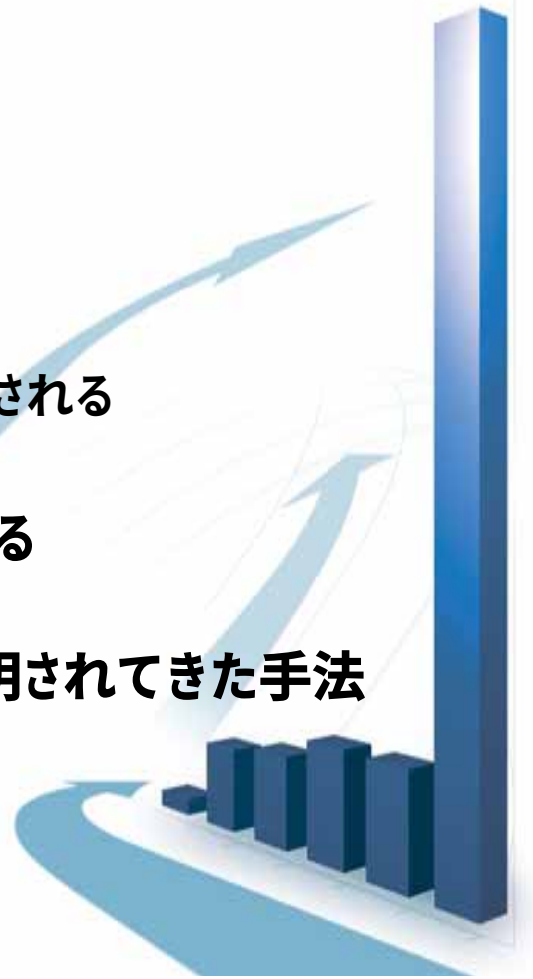
- １アプリユニットの価値 = 5000€
  - 主に開発者の人件費
- 開発者は効率を上げることができた
  - 25 000€ /月の成果を上げる
  - 20 000€ /月の上積み効果
- 開発者の一人当たりの費用:
  - MetaEdit+ のレンタル費 250€/月
  - 4ヵ月後からレンタル開始
- １アプリユニットに掛かる単価は75％ダウン
  - 5000€ から 1250€
  - 人件費とレンタル費用の合計

月:	6	12
上積みされたアプリ	34	148
上積みされた価値	170 000	740 000
追加必要経費	2 500	10 000
投資収益率	**6700%**	**7300%**

6人月の人件費	30 000
1ヶ月のDSMツールレンタル費	1 250
DSMツール込みの総費用	31 250
DSMによる月間総ユニット数	25
DSMによる1ユニットの単価	**1 250**

まとめ

- 生産性は抽象度を上げる事で実現できる
- モデリング言語、コードジェネレータ
  をカスタマイズする事で自動化を実現できる
- DSM 環境による組織の役割
  - チーム内のエキスパートがＤＳＭ環境を構築する
  - チームメンバーにより真のモデル駆動開発が達成される
    (エキスパートのノウハウを共有することにもなる)
- ＤＳＭ環境構築はエキスパートの意欲をかきたてる
  (モチベーションの向上)
- MetaEdit+ は、15年以上に渡って評価され証明されてきた手法
  - 100以上の実績



Thank you!

jpt@metacase.com
www.metacase.com

USA:
MetaCase
5605 North MacArthur Blvd.
11th Floor, Irving, Texas 75038
Phone (972) 819-2039
Fax (480) 247-5501

Europe:
MetaCase
Ylistönmäentie 31
FI-40500 Jyväskylä, Finland
Phone +358 14 641 000
Fax +358 420 648 606

国内：富士設備工業株式会社　電子機器事業部

担当：浅野　E-MAIL：info@fuji-setsu.co.jp

www.fuji-setsu.co.jp