MetaCase

ジェネレータとモデリング言語のコツ

うまくいっている組織のアプローチ

富士設備工業(株)電子機器事業部 浅野 義雄

Embedded Technology 2015／組込み総合技術展 小間番号:C-03

Introduction

- **ドメインスペシフィックモデリング言語は：**
  1. 開発の抽象レベルを問題空間に上げる
  2. 必要なコードをモデルから生成

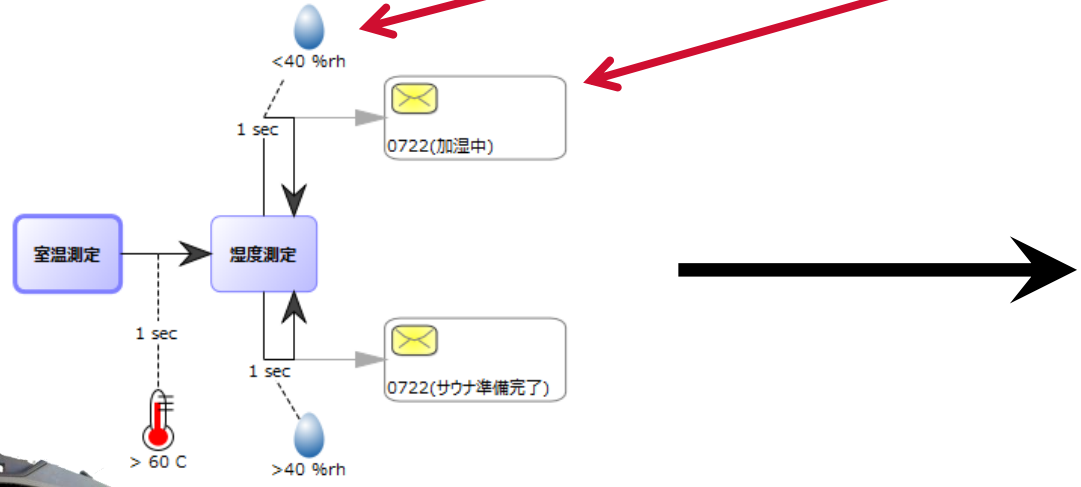
- この講演では、以下について紹介します
  - ドメインスペシフィックモデリング言語とジェネレータ
  - 事例
  - 避けるべきこと、取るべき手段についてのガイドライン

What is Domain-Specific Modeling?

- **ドメインの知識を用いる（コードではなく）**
  - ドメインのコンセプトやルールをモデリングに採用
  - 開発の抽象度をコード実装レベルから上げる
  - ドメインの抽象概念を採用
- **開発者は慣れ親しんだ用語を使うことができる**
  - 新しい言語を習う必要は無い
- **コードや様々な成果物に自動変換（自動生成）**
  - 最適なコード
  - 成果物への変換設定を自在にコントロール

Case 1: IoT device applications

IoTデバイスのセンサー(温度、湿度)やメッセージ送信機能など
コンセプトでモデリング



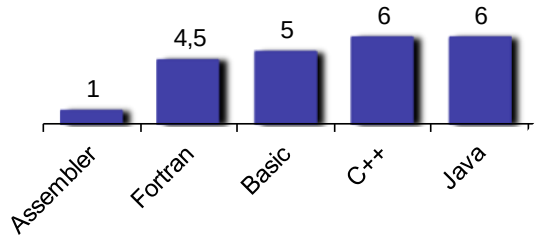
そしてIoTデバイスで実行するコードを生成


```
{
  "pId": "1",
  "apiVersion": "00.18",
  "initPuId": 1,
  "purposes": [
    {
      "puId": 1,
      "name": "Sauna",
      "initStId": 0,
      "states": [
        {
          "stId": 0,
          "name": "室温測定",
          "events": [
            {
              "evId": 0,
              "name": "",
              "actions": {
                "engine": {
                  "gotoStId": 1
                },
                "cloud": {
                  "sendEvent": false,
                  "sendPush": false
                }
              },
              "causes": [
                {
                  "sId": "0x00060100",
                  "threshold": {
                    "count": 1
                  },
                  "measurement": {
                    "log": false,
                    "interval": 1000
                  },
                  "thresholds": {
```

Why Domain-Specific Modeling?

- ソフトウエアエンジニアリングの歴史は開発の抽象レベルの向上であった
- 汎用のプログラミング言語は、新しいものでも生産性向上に寄与しない

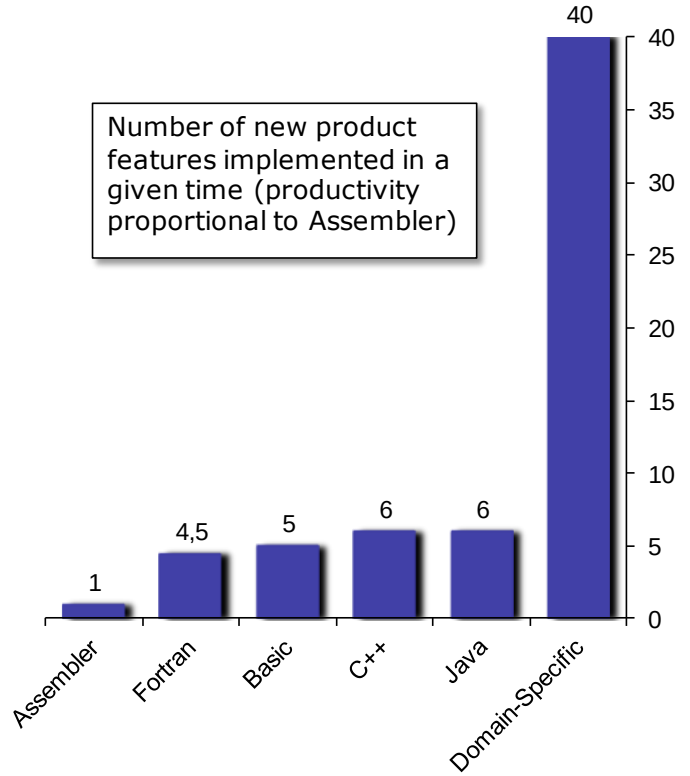
Number of new product features implemented in a given time (productivity proportional to Assembler)



*Software Productivity Research & Capers Jones, 2002

Why Domain-Specific Modeling?

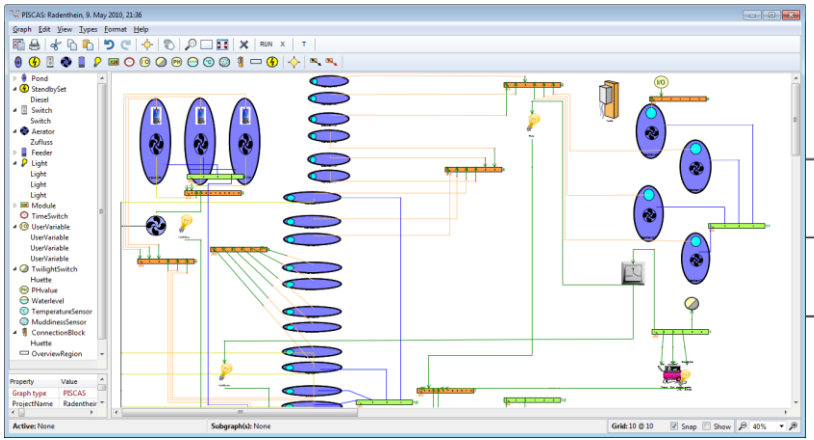
- ソフトウエアエンジニアリングの歴史は開発の抽象レベルの向上であった
- 汎用のプログラミング言語は、新しいものでも生産性向上に寄与しない
- ドメインスペシフィックな言語なら抽象レベルを上げて生産性が向上する

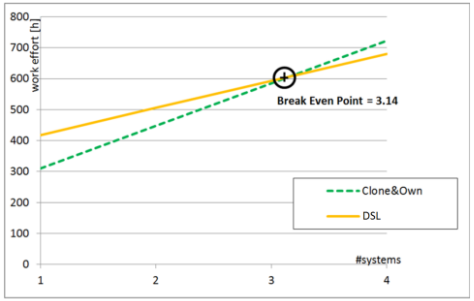


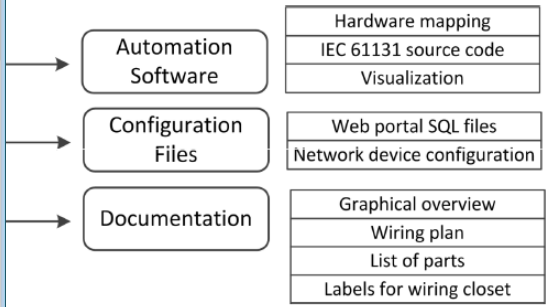
*Software Productivity Research & Capers Jones, 2002

Case 2: Fishfarm automation

- 養魚場の顧客と直接打合わせできるモデリング言語
- システム制御プログラムに加えて、機械設備や配線の設計図を生成
  => 工事業者に好評
- 3つ目のシステム開発で元が取れた

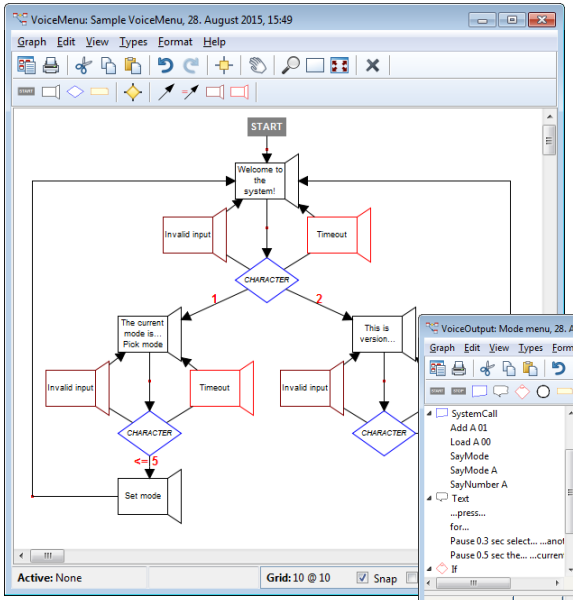




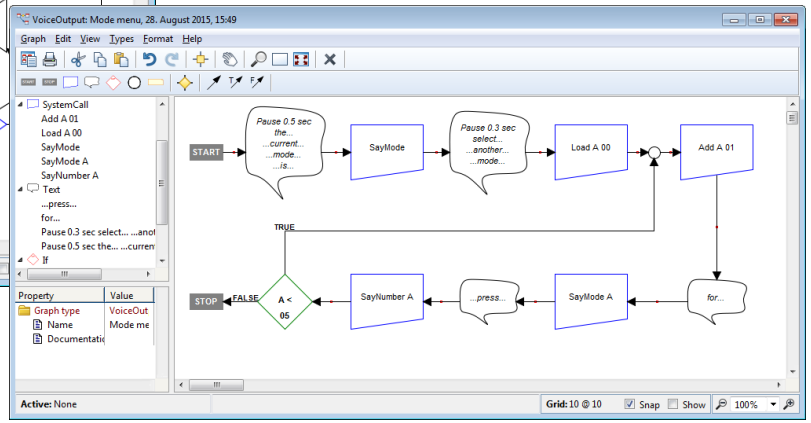


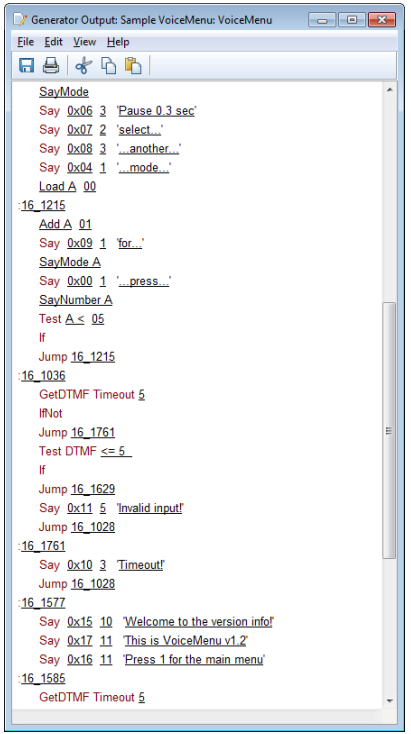
https://www.metacase.com/cases/hofernet.html

Case 3: Voice Command in MicroController



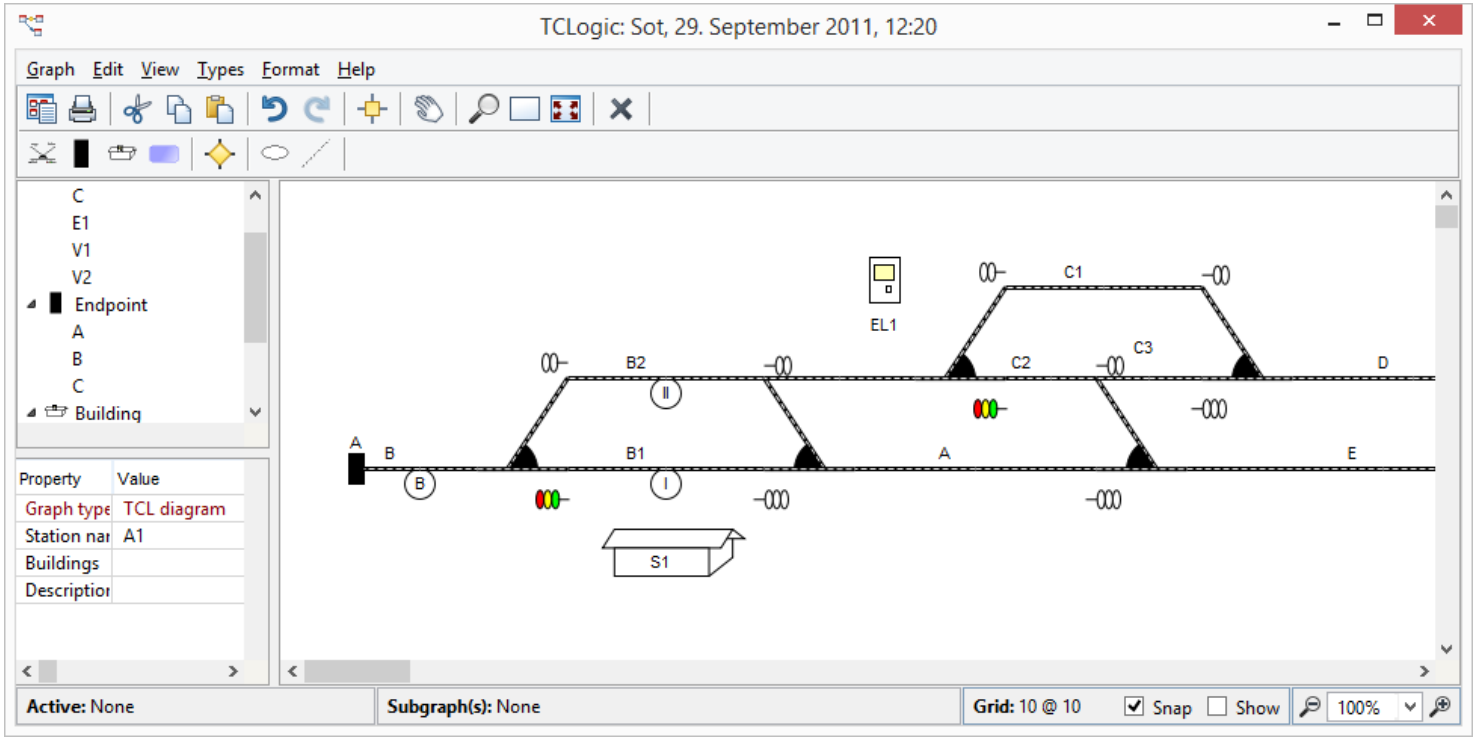
音声認識用の
8-bitマイコン
アセンブラコードを全て生成





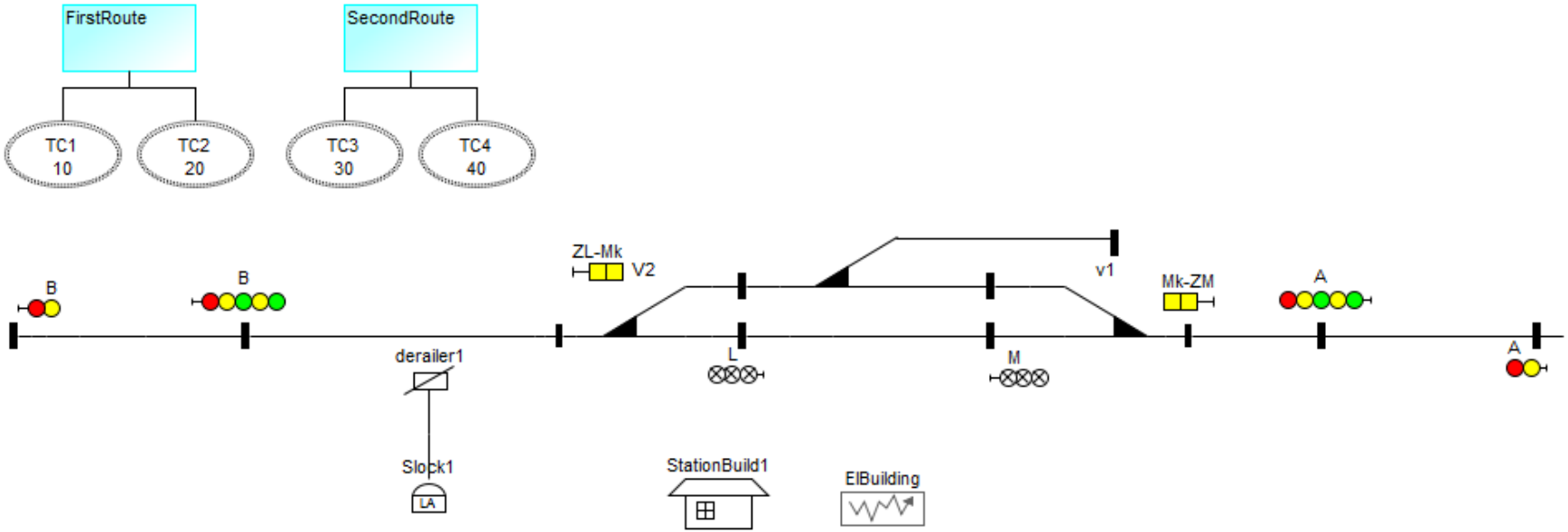
https://www.metacase.com/cases/microcontroller.html

Case 4: Railway interlocking

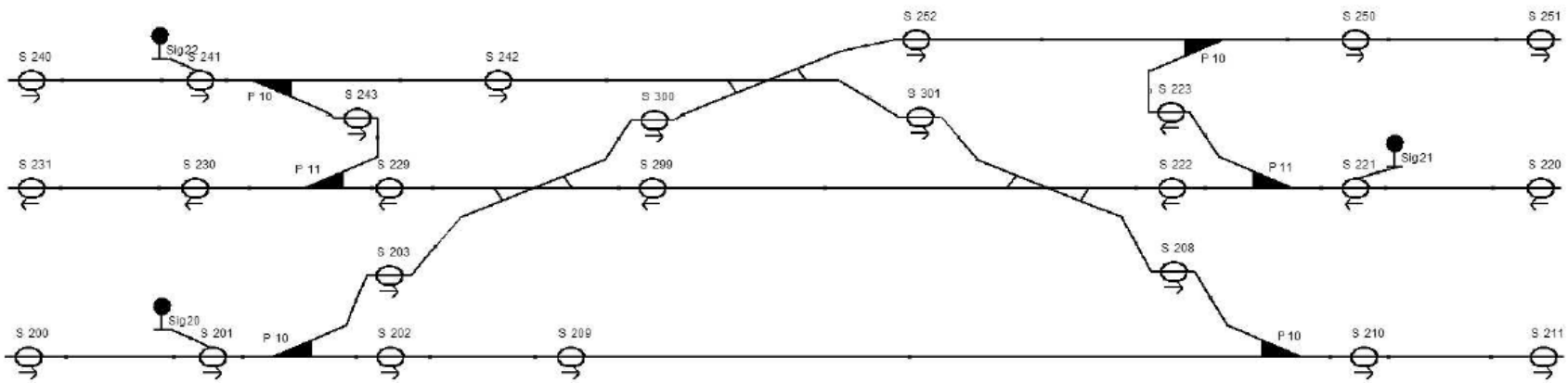


https://www.metacase.com/cases/railwayTrackControl.html

Station & Track DSL



Rail testing and verification



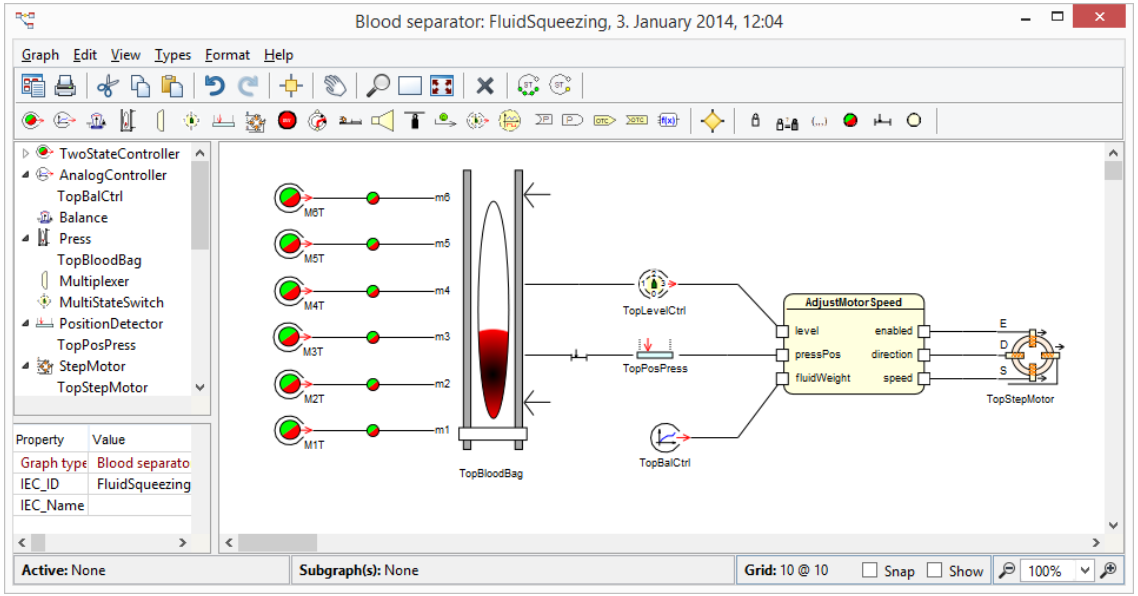
Route 1
Def: S200, S201, S202, S209, S210, S211
Start: Sig20:STRAIGHT
Signals:
Points: P102:STRAIGHT, P103:STRAIGHT,
Conflicts: R2, R6

Route 3
Def: S220, S221, S222, S299, S229, S230, S231
Start: Sig21:STRAIGHT
Signals:
Points: P118:STRAIGHT, P119:STRAIGHT,
Conflicts: R2, R4, R5, R6

Route 5
Def: S240, S241, S243, S230, S231
Start: Sig22:RIGHT
Signals:
Points: P100:RIGHT, P119:LEFT
Conflicts: R3, R6

Case 5: Blood separator machines

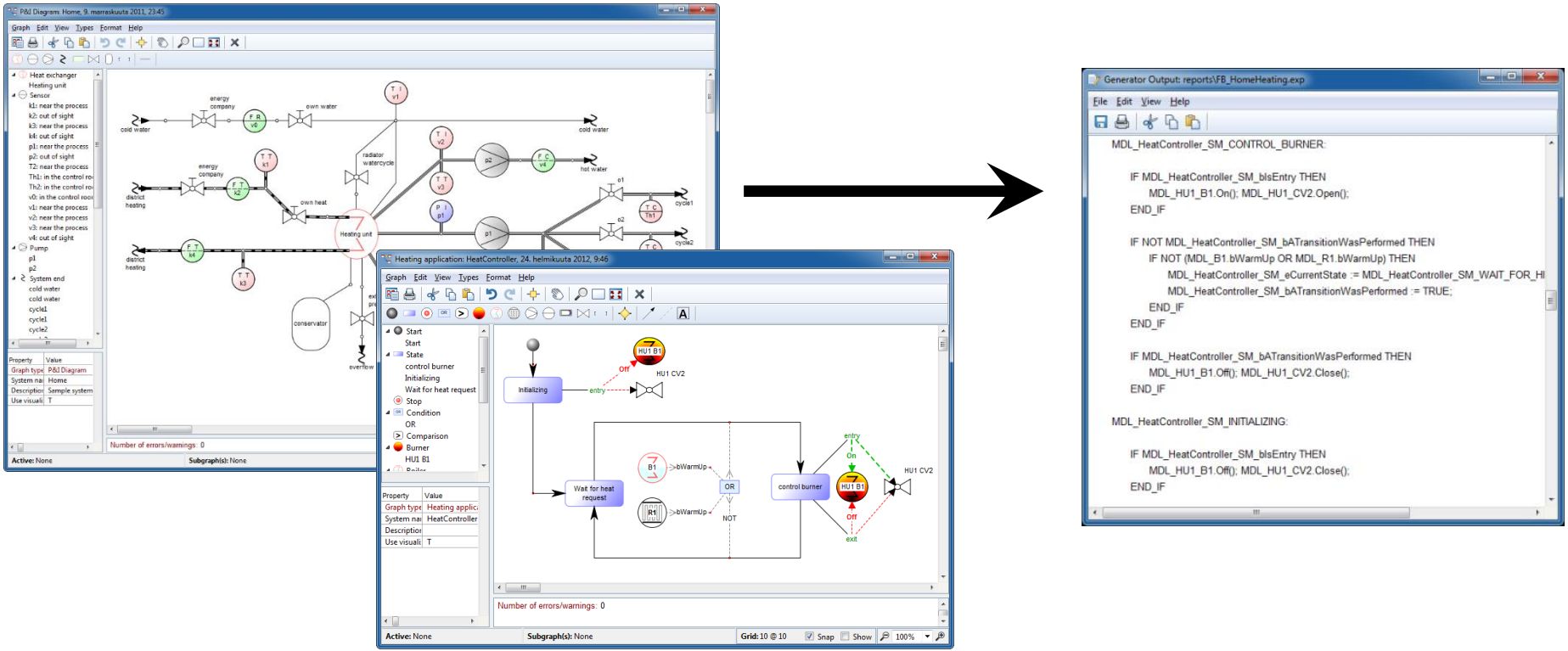
国際規格：IEC 61131-3（機能ブロックでPLCプログラムを組む構造化言語＝汎用）を
血液分離機のドメインコンセプトで拡張



https://www.metacase.com/cases/BloodSeparationMachine.html

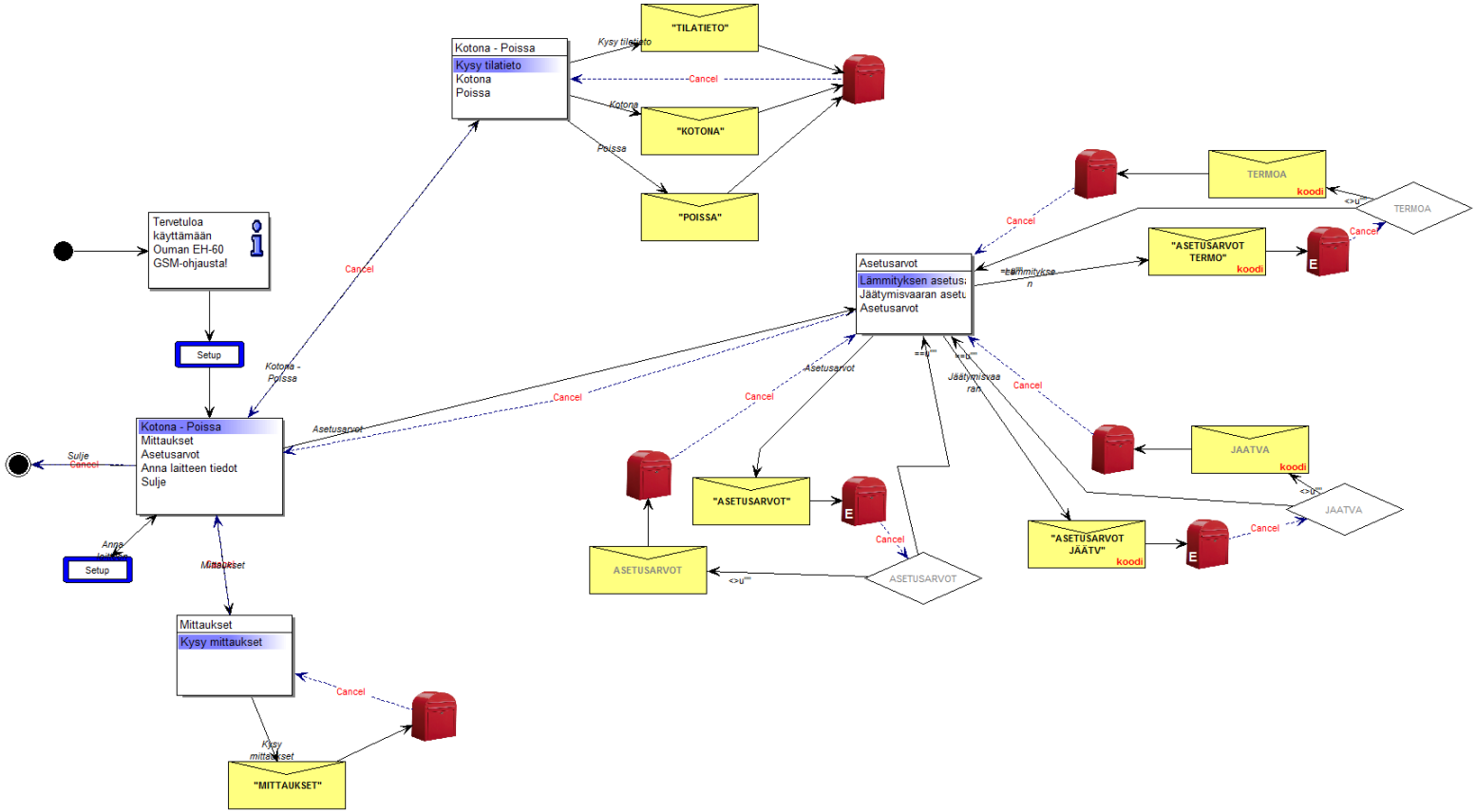
https://www.embedded.com/using-domain-specific-modeling-languages-for-medical-device-development/

Case 6: Heating systems & PLC code



https://www.metacase.com/cases/PLC_heating.html

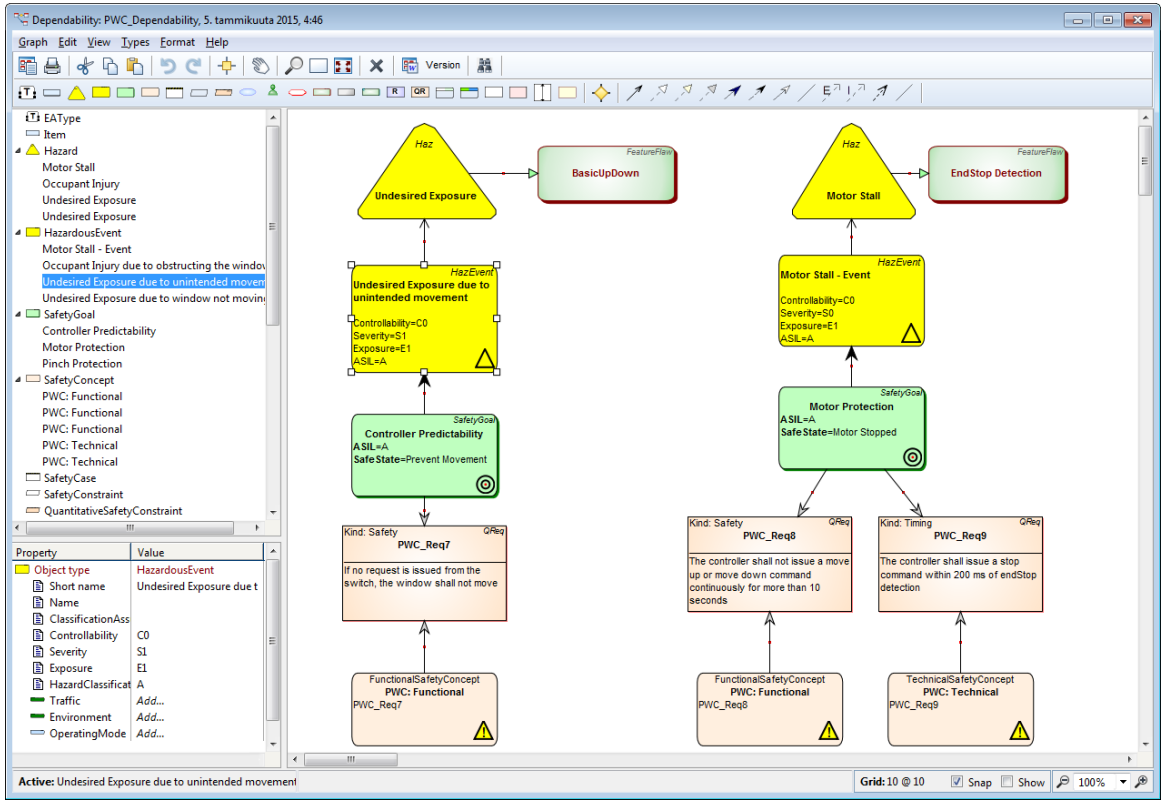
Heating system remote control



Case 7: Dependability and ISO26262

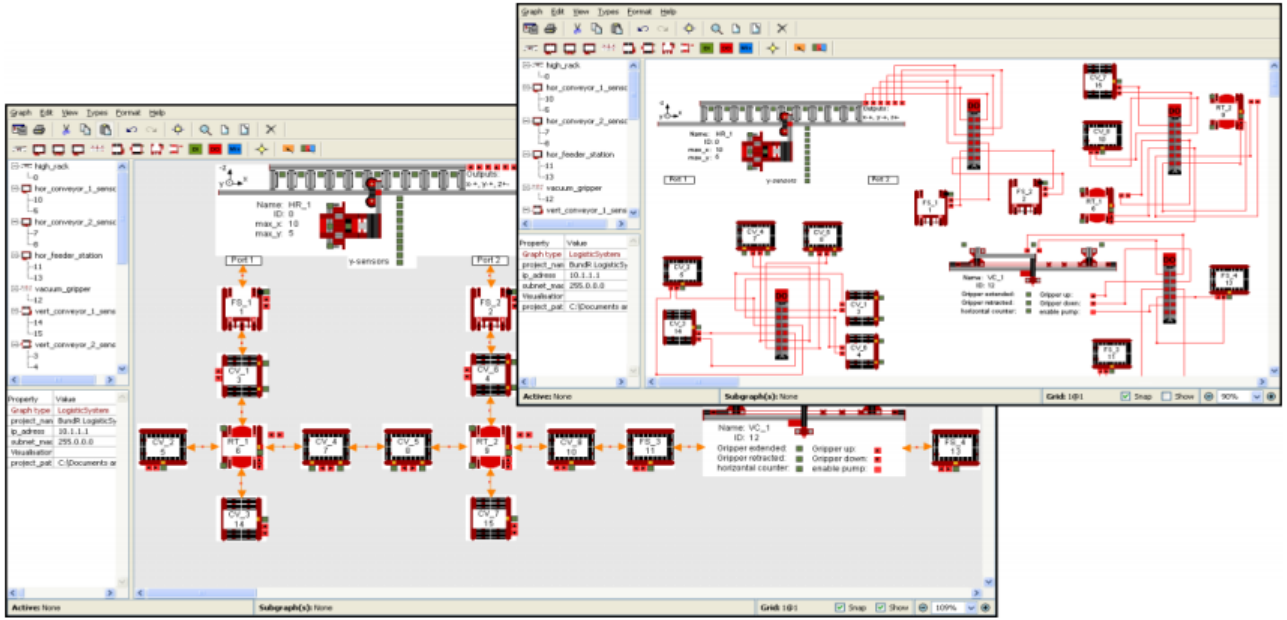
EAST-ADL はSysML（=汎用）を参考にして開発された、車載システムアーキテクチャ用のドメインスペシフィック言語。

プロダクトライン開発のフィーチャモデル、ISO 26262対応としてエラーモデルやディペンダブルモデル、Simulink、AUTOSAR 統合など

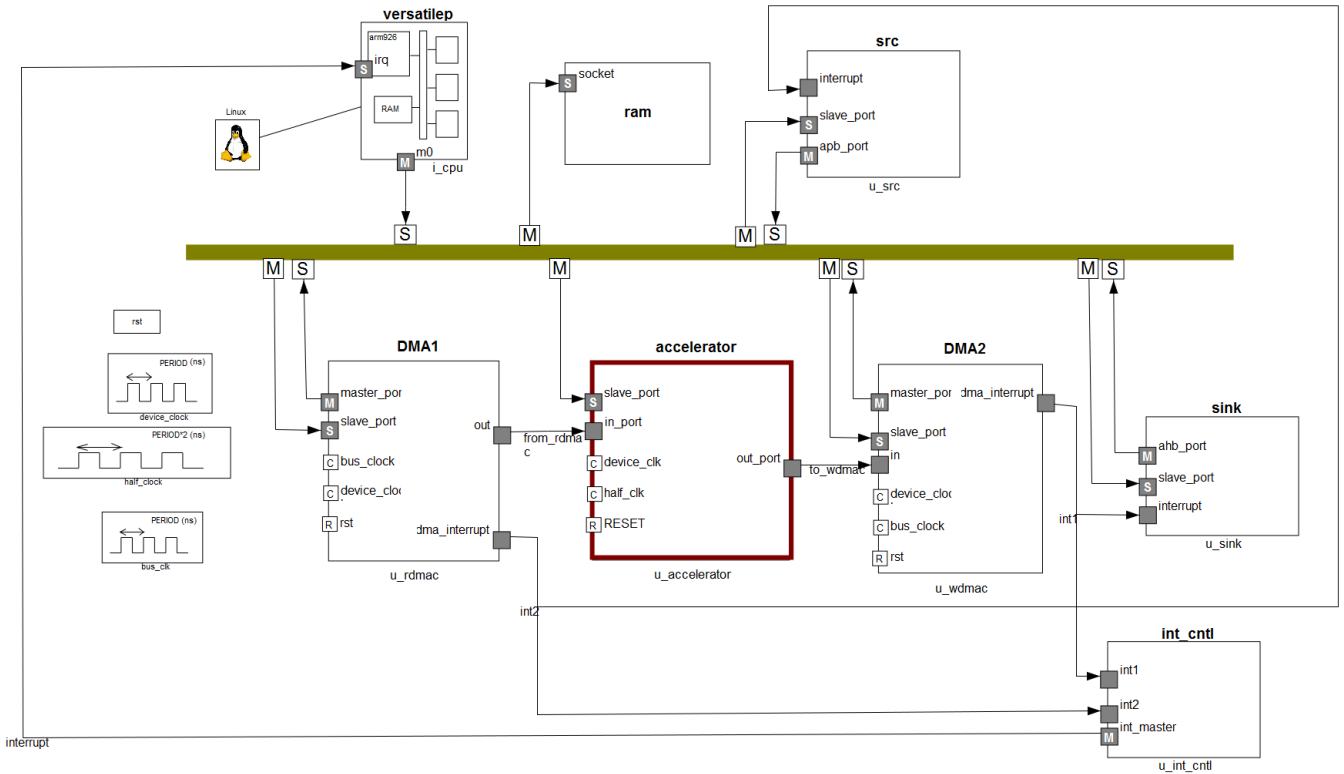


https://www.metacase.com/ja/solution/east-adl.html

Case 8: Warehouse automation systems

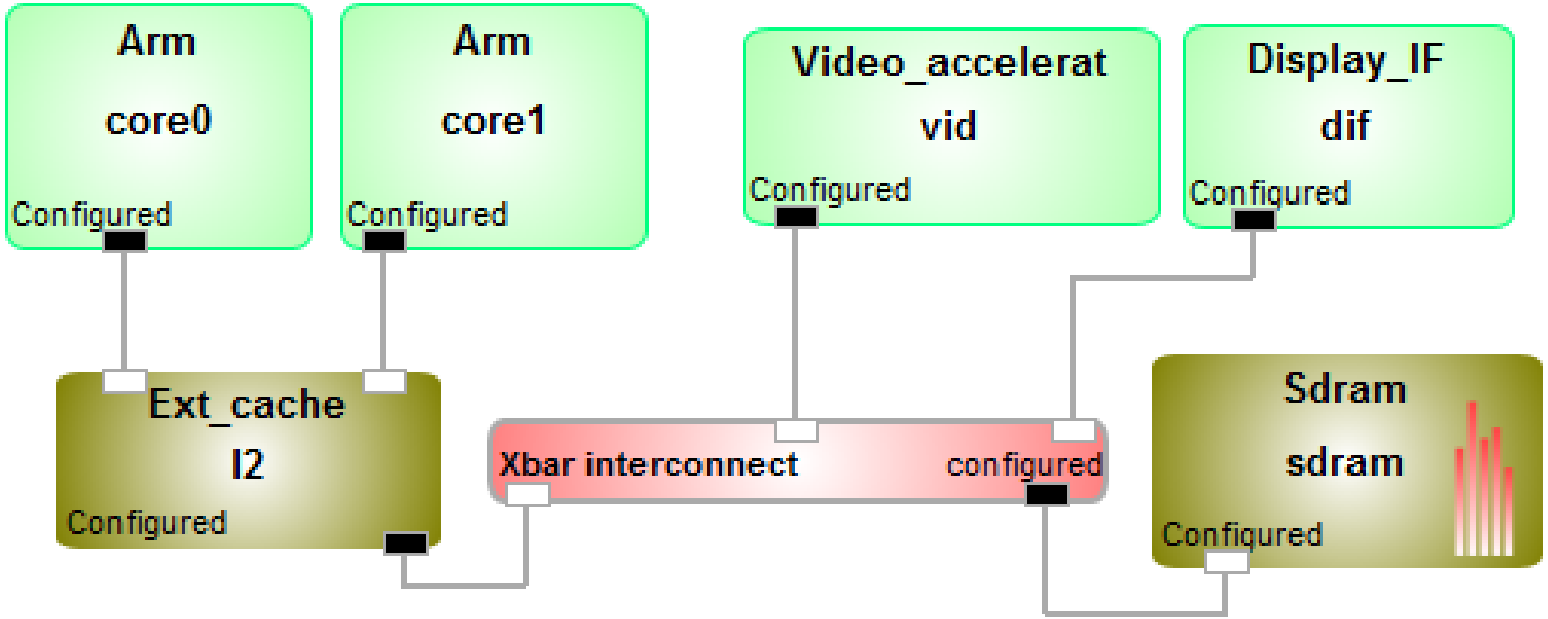


Case 9:Hardware (for System C)



https://www.fuji-setsu.co.jp/products/MetaEdit/QEMU_SystemC.html

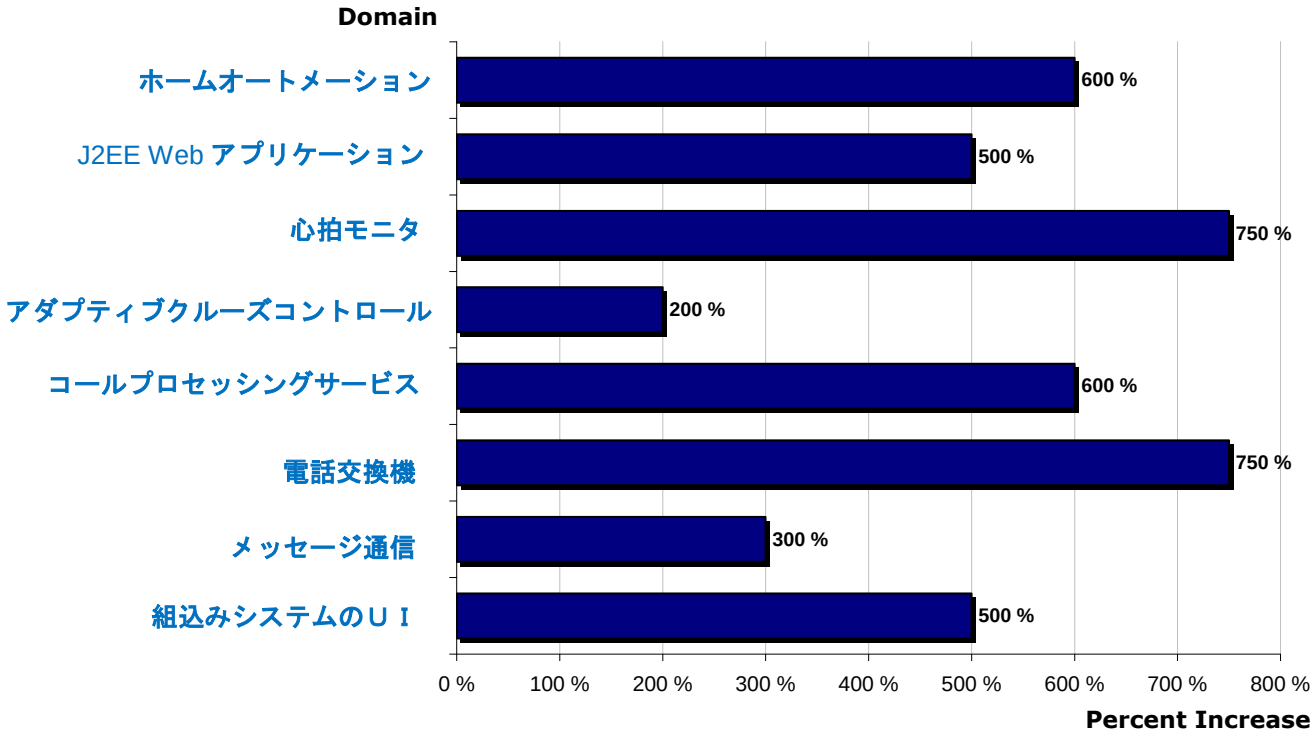
HW platform for virtual simulation



http://www.ijcce.org/papers/234-W0018.pdf

https://www.semanticscholar.org/paper/Domain-Specific-Front-End-For-Virtual-System-Vatjus-Anttila/939dd0e0ed1378240d6635717425d4ebd47a6101

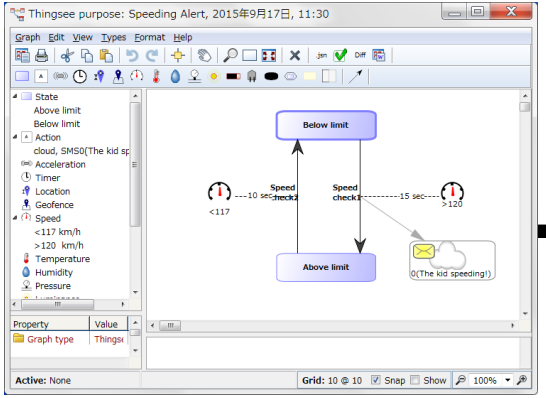
Productivity increase from DSM



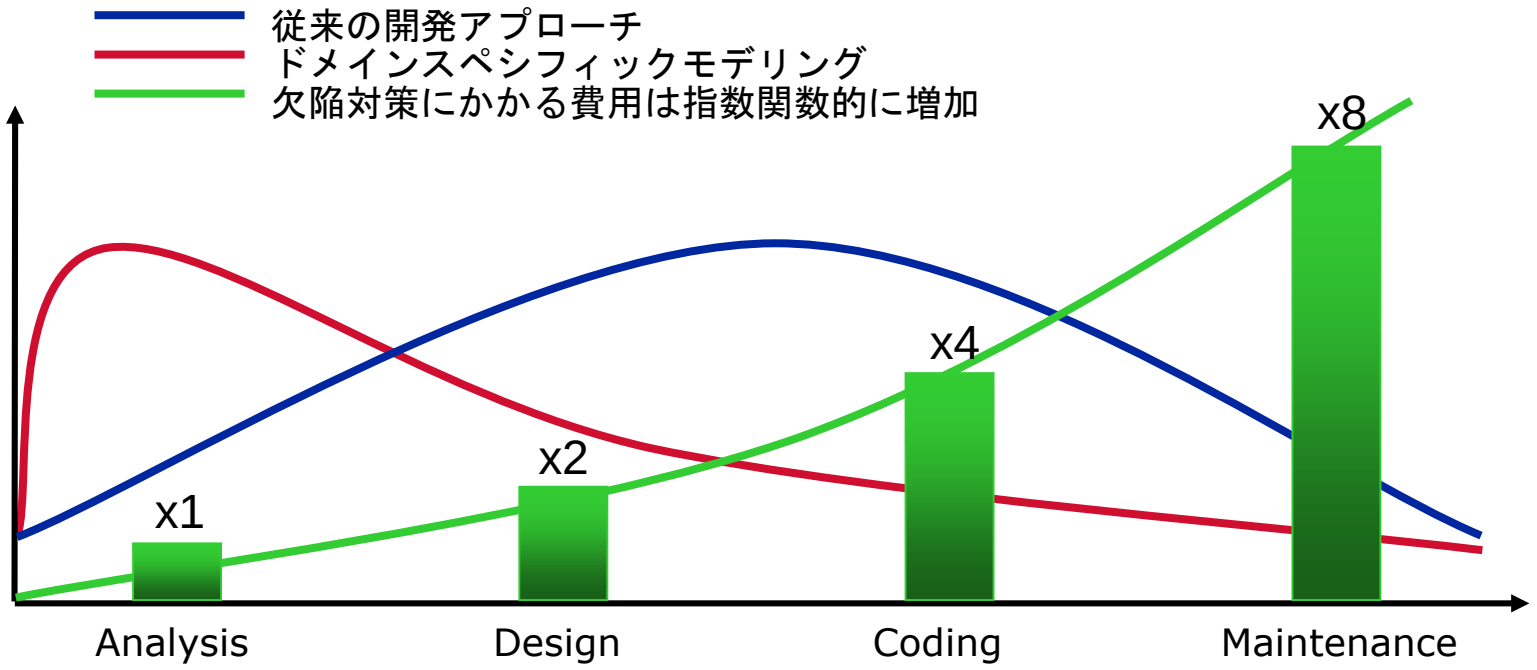
* Productivity proportional to earlier practice

Benefits

- 生産性
  - 少し描いて多くを生成
- 品質
  - 間違いを排除（ドメインのルールや制約で）
    - 使ってはいけないモデル部品、接続ミス
      多すぎる接続、不完全なモデルなど
  - 生成されるソースコード
    - タイプミス、データ値の欠如、誤った参照
      初期化忘れなどを排除できる

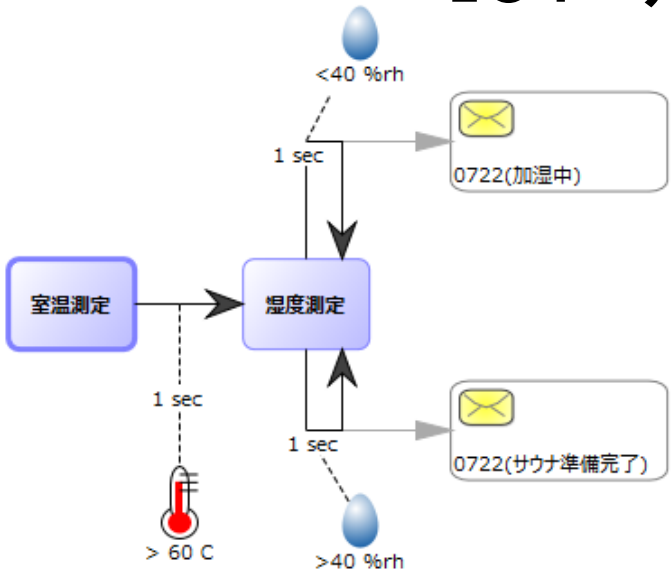


Defect distribution and costs*



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

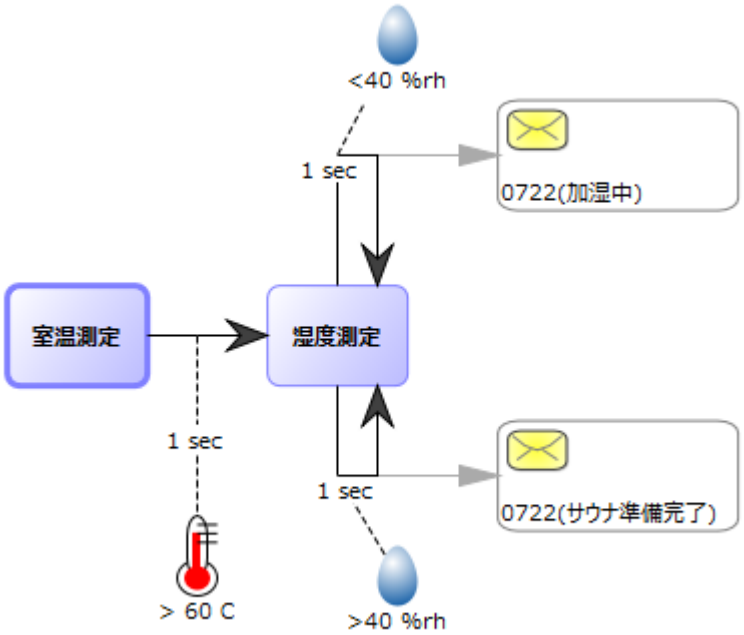
IoT デバイス for サウナ



https://www.fuji-setsu.co.jp/demo/DSMforSauna.wmv

Measuring productivity

Model: 7 elements, 3 relations

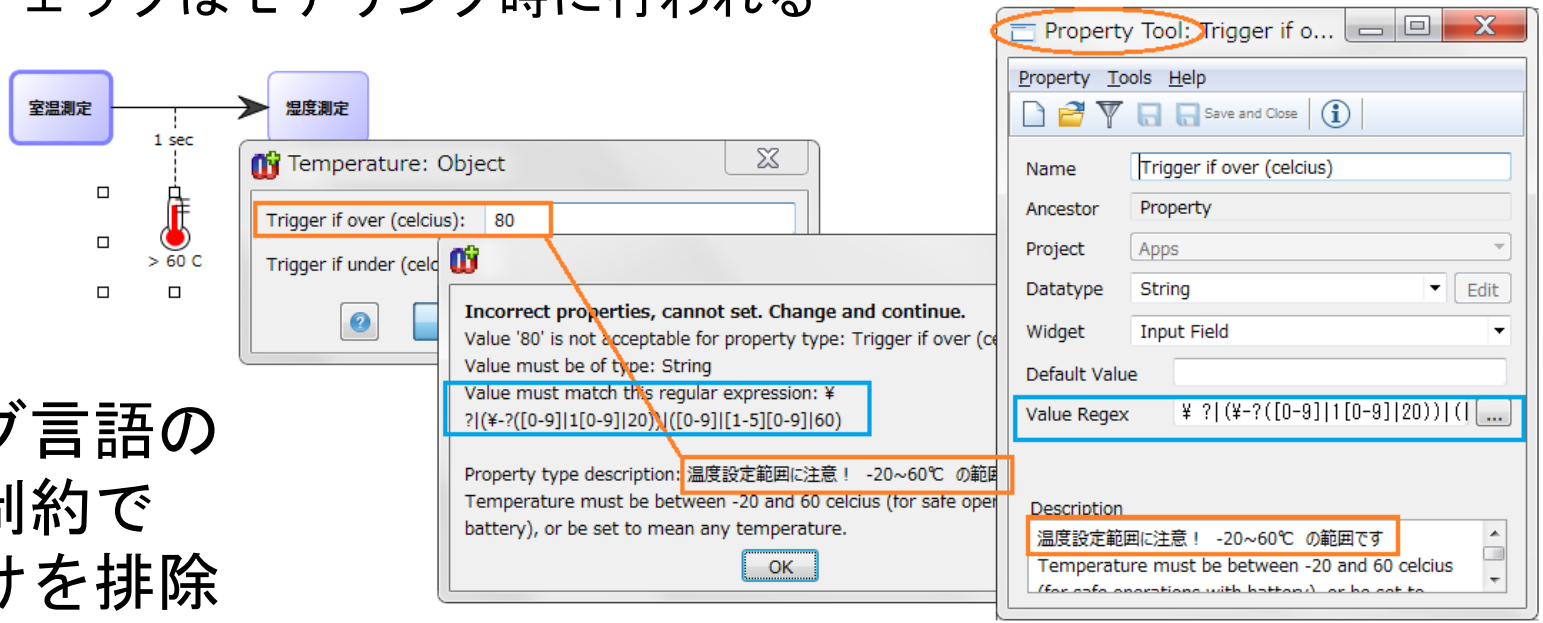


Generated code: 116 lines

```
{
  "pId": "1",
  "apiVersion": "00.18",
  "initPuId": 1,
  "purposes": [
    {
      "puId": 1,
      "name": "Sauna",
      "initStId": 0,
      "states": [
        {
          "stId": 0,
          "name": "室温測定",
          "events": [
            {
              "evId": 0,
              "name": "",
              "actions": {
                "engine": {
                  "gotoStId": 1
                },
                "cloud": {
                  "sendEvent": false,
                  "sendPush": false
                }
              },
              "causes": [
                {
                  "sId": "0x00060100",
                  "threshold": {
                    "count": 1
                  },
                  "measurement": {
                    "log": false,
                    "interval": 1000
                  },
                  "thresholds": {
```

Measuring quality

- 例：誤った温度を指定するとエラーダイアログで適正範囲を示唆（間違ったモデルからコードを生成させない）
- このチェックはモデリング時に行われる



モデリング言語のルールや制約で漏れ・抜けを排除

Case 10: Industrial Process Plant Design

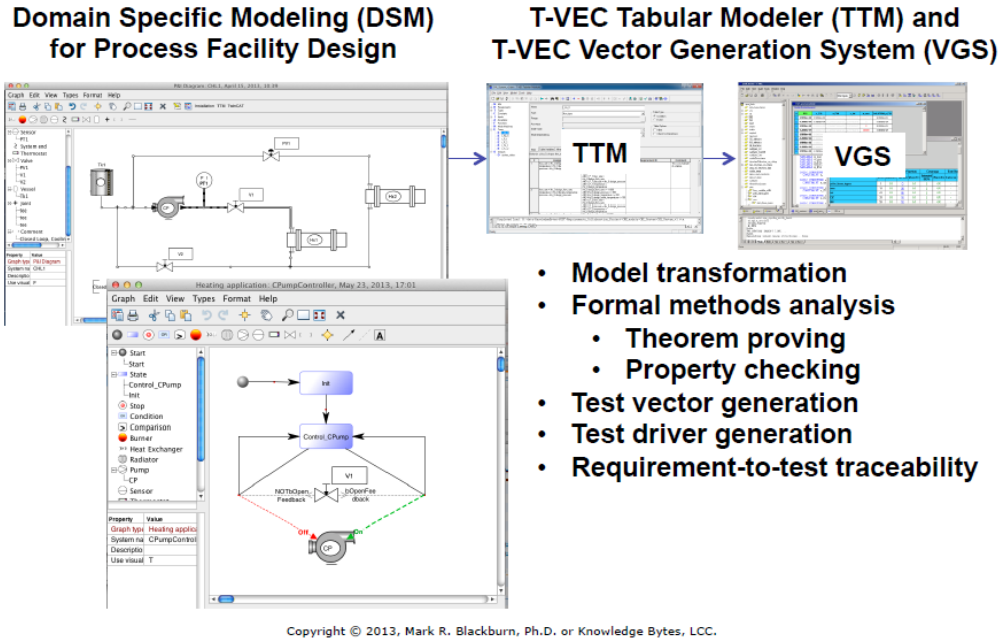
T-VEC 社 TTM
モデルベースドテスト

-仕様に矛盾が無いことを
定理証明(フォーマルメソッド)

-モデル(=仕様)からテストベクタ
(入力・期待値)を自動生成。

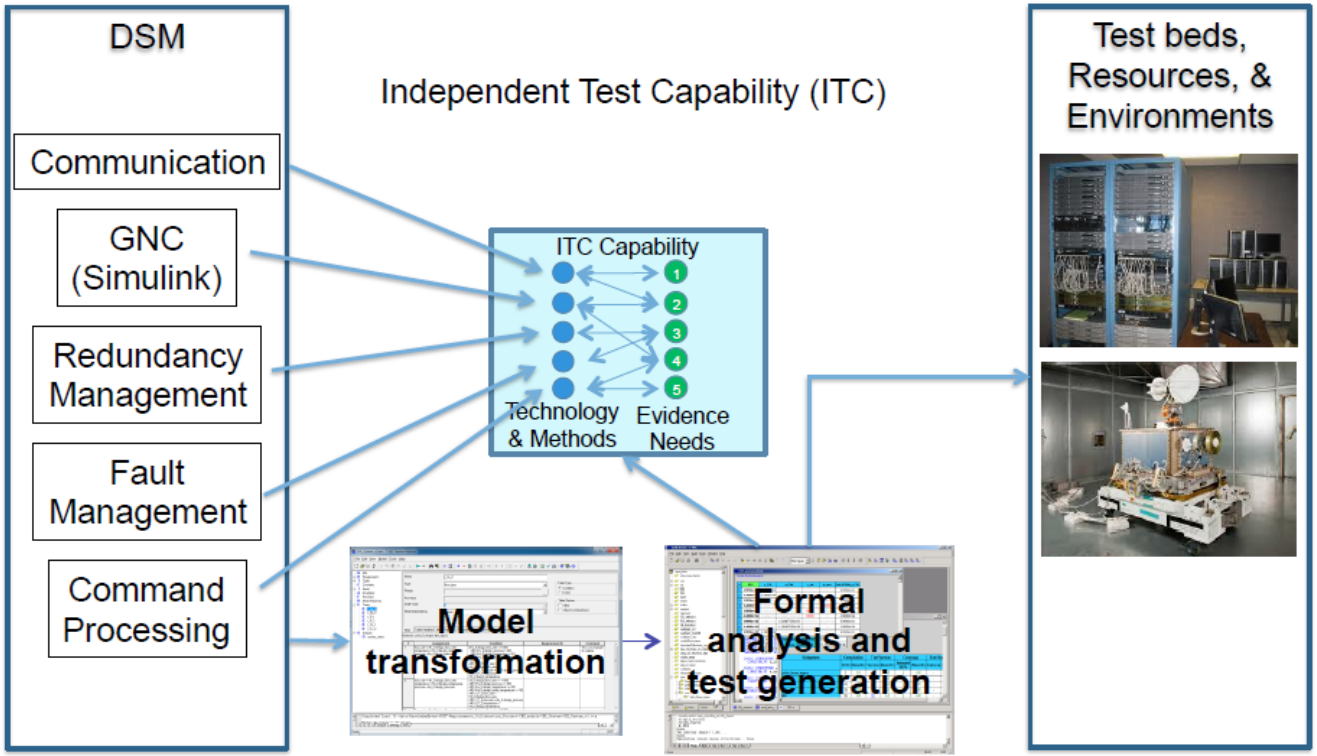
一致制の検証、トレーサビリティ
の工数を削減

=> DSM導入のモチベーションに



Virtual Design and Verification of Cyber Physical Systems : Mark Blackburn, Peter Denno*
Stevens Institute of Technology, Castle Point on Hudson, Hoboken, NJ, 07030, USA
National Institute of Standards and Technology, 100 Bureau Drive, Gaithersburg, MD, 20899-8265, USA
https://www.sciencedirect.com/science/article/pii/S1877050914000696

DSM Transformation to Formal Analysis and Test Tools Maps to ITC Capability

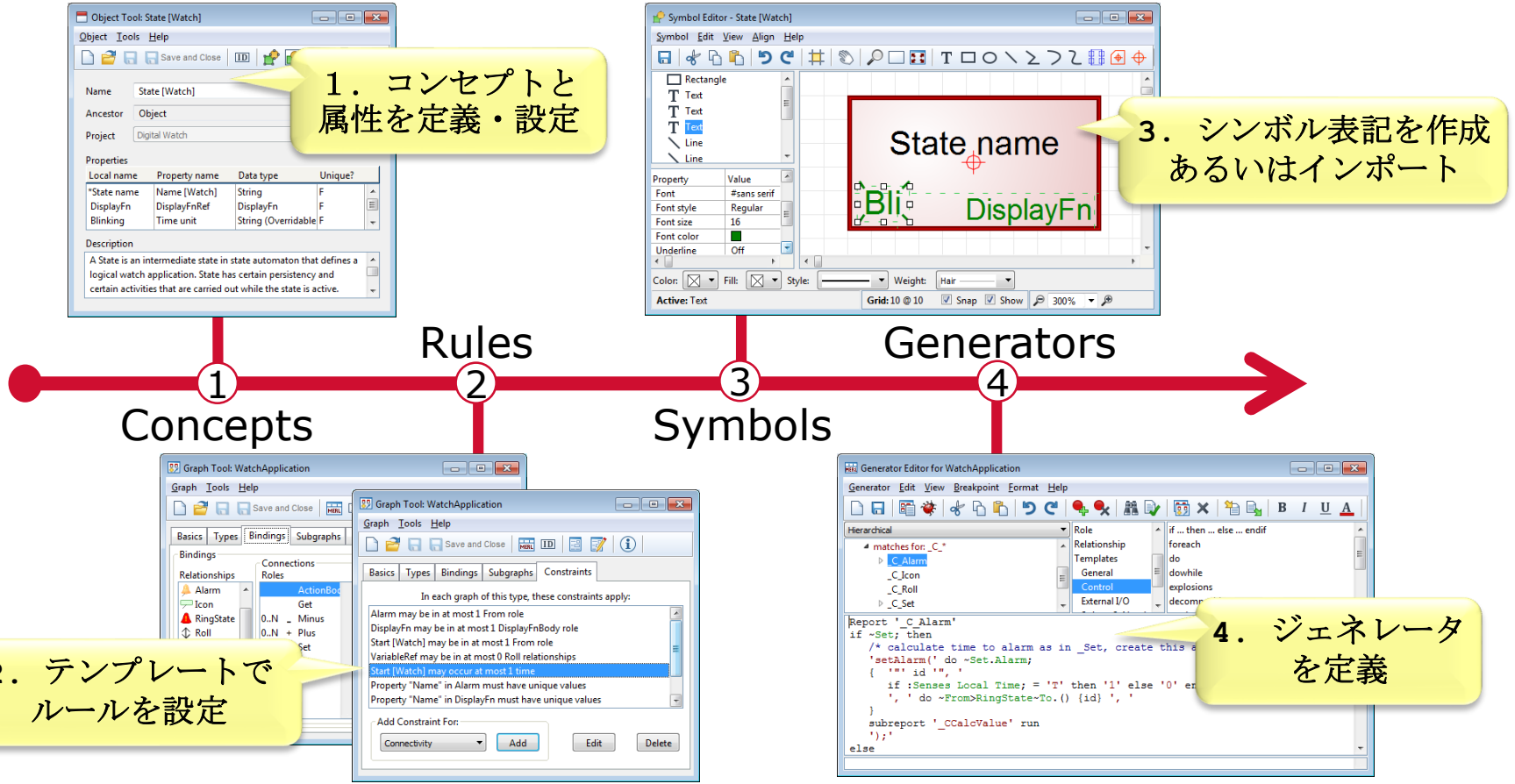


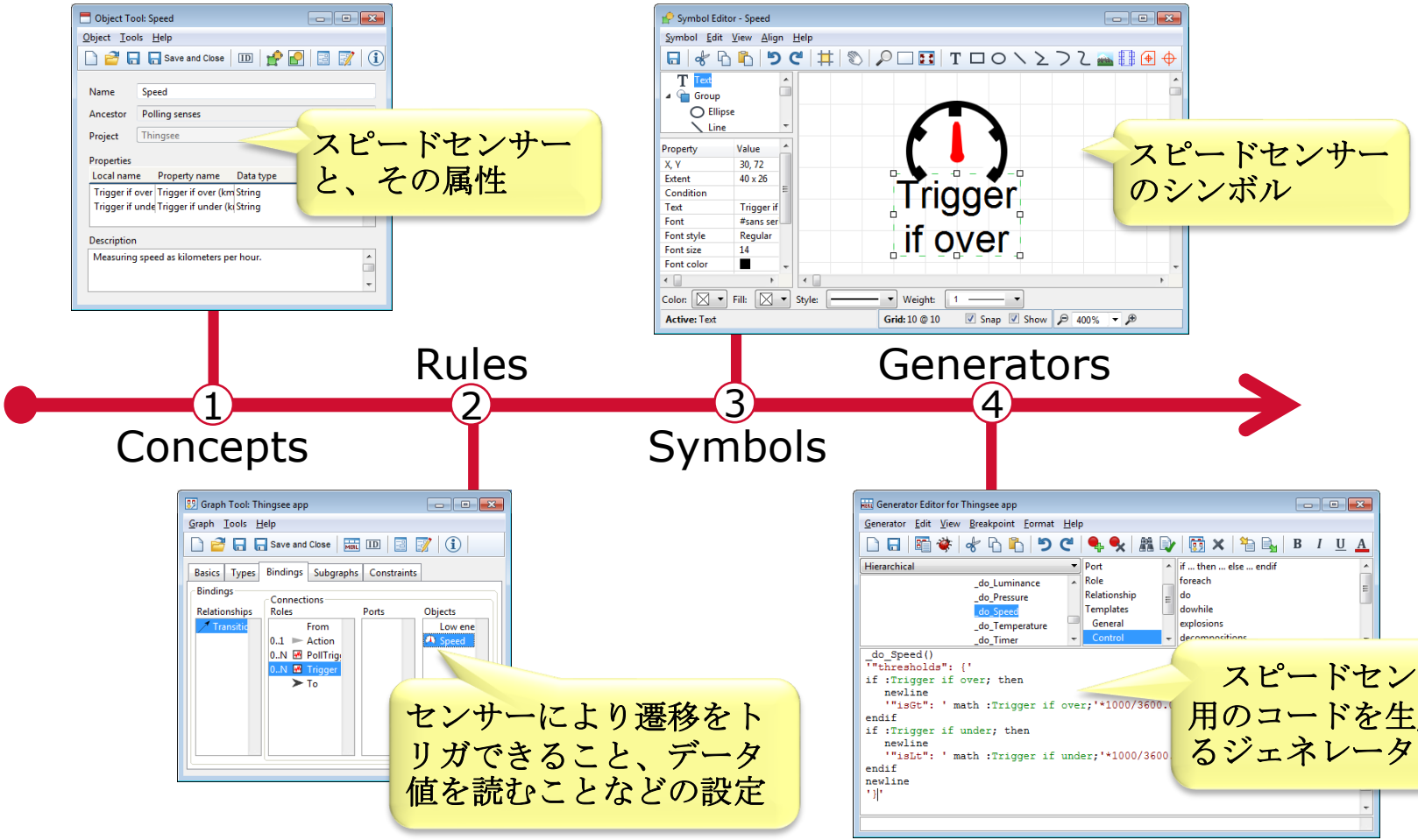
Copyright © 2013, Mark R. Blackburn, Ph.D. or Knowledge Bytes, LCC.

Steps for defining a DSM solution

1. 抽象概念を見出す
   - コンセプト、およびそれらがどのように相互作用するか
     （例： IoT のセンサーと、それに伴うアクション）
2. メタモデルを定義
   - 言語のコンセプトとルール
3. 表記を定義
   - モデルを描写し表現するもの
4. ジェネレータを定義
   - モデルから様々な成果物を生成したり、モデルの解析を行える

＝＞ イテレーティブなプロセスで（サンプルで試して進化させる）
- 一部分を定義して、それを試して、次を追加して、、


1．コンセプトと
属性を定義・設定
3．シンボル表記を作成
あるいはインポート
Rules
Generators
1
2
3
4
Concepts
Symbols
2．テンプレートで
ルールを設定
4．ジェネレータ
を定義

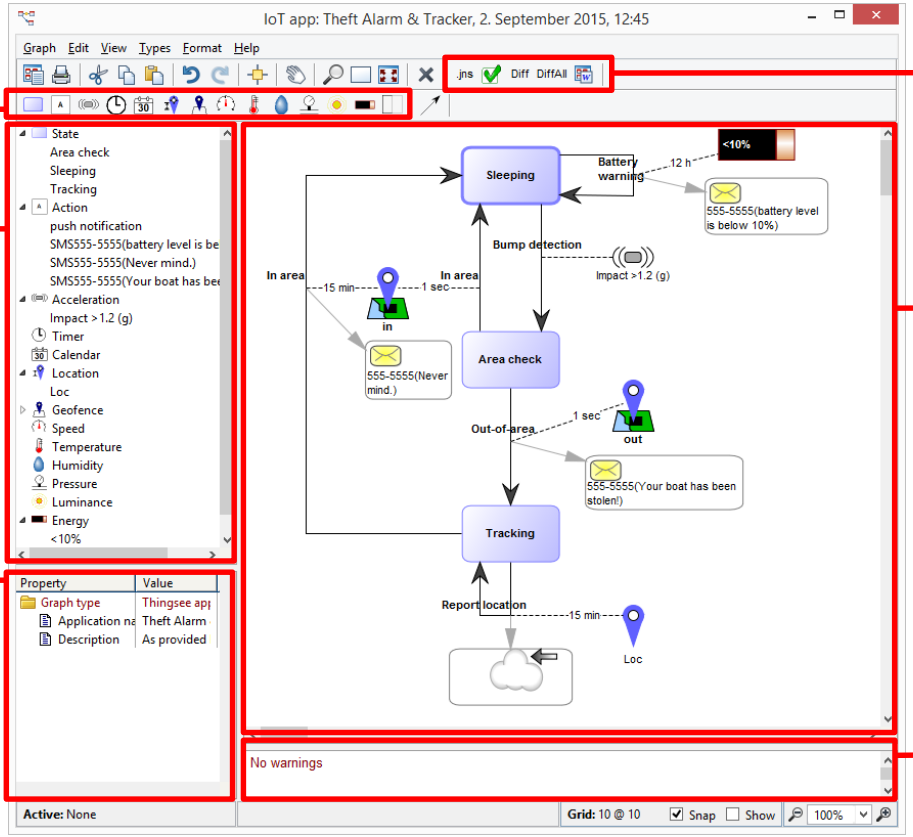

Object Tool: Speed
Object Tools Help
Save and Close
Name Speed
Ancestor Polling senses
Project Thingsee
Properties
Local name Property name Data type
Trigger if over Trigger if over (km String
Trigger if unde Trigger if under (k String
Description
Measuring speed as kilometers per hour.
スピードセンサーと、その属性
Symbol Editor - Speed
Symbol Edit View Align Help
Text
Group
Ellipse
Line
Property Value
X, Y 30, 72
Extent 40 x 26
Condition
Text Trigger if
Font #sans ser
Font style Regular
Font size 14
Font color
Trigger if over
Color: Fill: Style: Weight:
Active: Text
Grid: 10 @ 10 Snap Show 400%
スピードセンサーのシンボル
Rules
Generators
1
Concepts
2
3
Symbols
4
Graph Tool: Thingsee app
Graph Tools Help
Save and Close
Basics Types Bindings Subgraphs Constraints
Bindings
Relationships
Transiti
Connections
Roles
From
0..1 Action
0..N PollTrig
0..N Trigger
To
Ports
Objects
Low ene
Speed
センサーにより遷移をトリガできること、データ値を読むことなどの設定
Generator Editor for Thingsee app
Generator Edit View Breakpoint Format Help
Hierarchical
_do_Luminance
_do_Pressure
do_Speed
_do_Temperature
_do_Timer
Port
Role
Relationship
Templates
General
Control
if ... then ... else ... endif
foreach
do
dowhile
explosions
_do_Speed()
'"thresholds": {'
if :Trigger if over; then
newline
'"isGt": ' math :Trigger if over;'*1000/3600
endif
if :Trigger if under; then
newline
'"isLt": ' math :Trigger if under;'*1000/3600
endif
newline
'}'
スピードセンサー用のコードを生成するジェネレータ

Resulting solution for IoT app dev

言語のコンセプト
とそれらのアイコン

ツリー表示
（カスタマイズ可能）

属性表示
（カスタマイズ可能）



ジェネレータ各種
（コード、文書、テストなど）

モデリング画面

モデルチェックやエラー、
ウォーニングを表示

Characteristics of successful DSMLs

- 言語のコンセプトを問題空間から得ること。
  解決空間（コード）のコンセプトでは無く
  - UML のクラス、継承、コードの視覚化では無く
- 特定ドメインにのみスコープの範囲を限定する
  - 狭いほど良い、必要に応じて拡張できる
- モデルの開発、アップデート、チェックに要する作業は最小限に
  - モデルチェックやリファクタリングを支える仕組みが必要
- 顧客や利害関係者間の意思疎通をサポートできること

ドメインスペシフィック言語とジェネレータ開発のコツと避けるべきこと

- We investigate the following aspects:
  1. Quality of language, rules, generators
  2. Incremental language development
  3. Evolution and maintenance of languages
  4. Generator development
  5. Generator and transformation speed
  6. Scalability (large models, multiple engineers)

Quality of resulting language (& tool)

- モデリング言語の出来の良さや品質は作り方に大きく依存
- コンセプト、ルール、表記など定義がひとつに統合されていること
- UMLはコンセプトのルールを無くしてしまった、、 

言語の定義が分断されていると

- コンセプトはメタモデルに（MOF）
- ルールは制約言語で（OCL）
- 表記はシンボル定義に（テキスト）
- モデル変換はコードで（QVTやXSLT）
- ツールの各種機能もコードで（Java）

言語の定義が統合されると

- 一部分への変更が全てに反映
  - ルールや制約に対して
  - シンボルに対して
  - ジェネレータに対して
  - ツールやアイコンに対して

361 errors in UML 2.0: Bauerdick et al, in Procs of UML 2004, LNCS 3273, Springer, 2004
320 errors in UML 2.3: Wilke & Demuth, 2010, journal.ub.tu-berlin.de/eceasst/article/download/669/682

MOF とOCLのリンク等に300以上の欠陥があることが調査され公表されている。OMGによるUMLの仕様が悪いという事（UMLツール屋さんのせいではないけれど、、）

Incremental language development

- **最初から完璧な言語を用意することはできない**

➔ 言語を使用するユーザの意見を取り入れて段階的に開発することが最善の方法

- 言語がユーザの関わり無く定義される
- 言語の定義が紙上のスペックのみ（直接試せない）
- 単純なスキーマ言語のように定義が部分的（コンセプト、ルール・制約、表記等の定義が分断）
- モデリングの成果物への影響が考慮されないで言語を更新する

- 言語を使用するユーザが直接関わる
- 実際に使用して試す（コードを生成させて実行して確認する）
- 言語の変更時に他の部分への影響がチェックされる

Evolution (of language & models)

- **メンテナンスは開発の多くを占める**
- CodeGen 2011で紹介された保険用のドメイン固有言語はアップデートされてからモデルの更新に5カ月も費やした

アップデートの良くないシナリオ

1. 言語変更
2. ツールを変更
3. アップデートをパッケージ化してシェア
4. アップデートを全員がインストール
5. 既存モデルをアップデート

アップデートの良いシナリオ

1. 言語を変更すれば、
   - 自動的にシェアされて、
   - ツールも自動でアップデートされて、
   - モデルも自動でアップデートされる

Generator development process

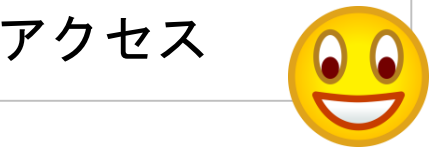
- ジェネレータの開発担当は多くをマスターしなければならない：
  メタモデル（モデルも）、ジェネレータ言語、
  生成対象言語とそのライブラリー

分断されてしまうと、、

- メタモデルはXML スキーマ
- モデルはXML
- ジェネレータはXSLT
- 生成対象は.x 形式

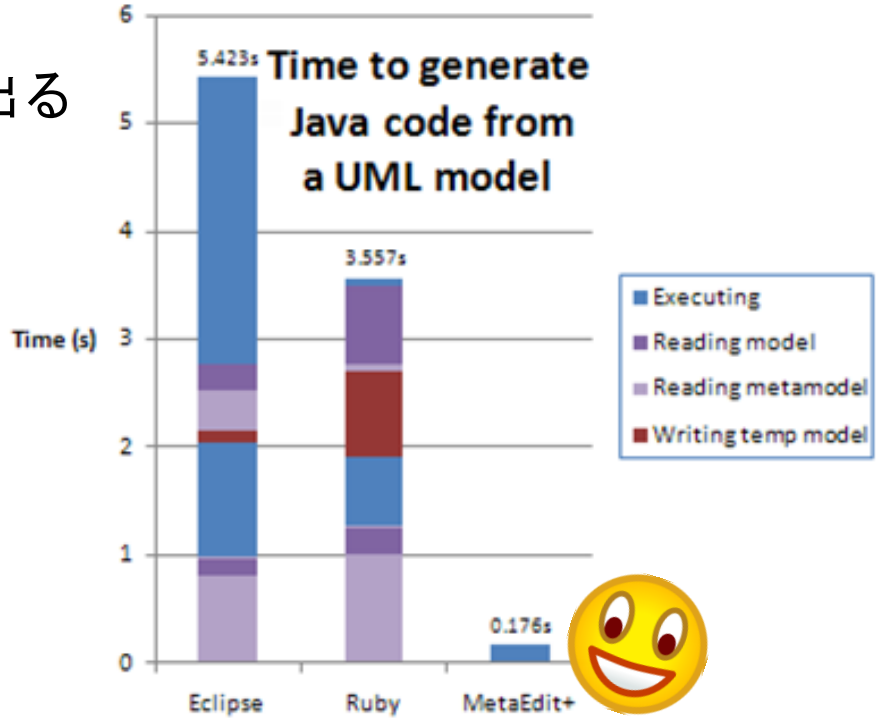
統合環境下では、

- ジェネレータ定義内からメタモデルにアクセス
- 生成されたコードからソースとなったモデルにリンク
- ジェネレータのデバッグ時にモデルにアクセス



Generator speed

- 用いるツールや手法で顕著な差が出る
- 4coreのPentiumマシンで
  一晩かけても間に合わない
- 原因は無駄な手順
  - モデルを読んで
  - メタモデルも読んで
  - 一時的なモデルをM2Mで生成して
  - 一時的なモデルからコード生成



Cuadrado & Molina, Building Domain-Specific Languages for Model-Driven Development, IEEE Software, 2007, https://www.computer.org/csdl/magazine/so/2007/05/s5048/13rRUIM2VzW

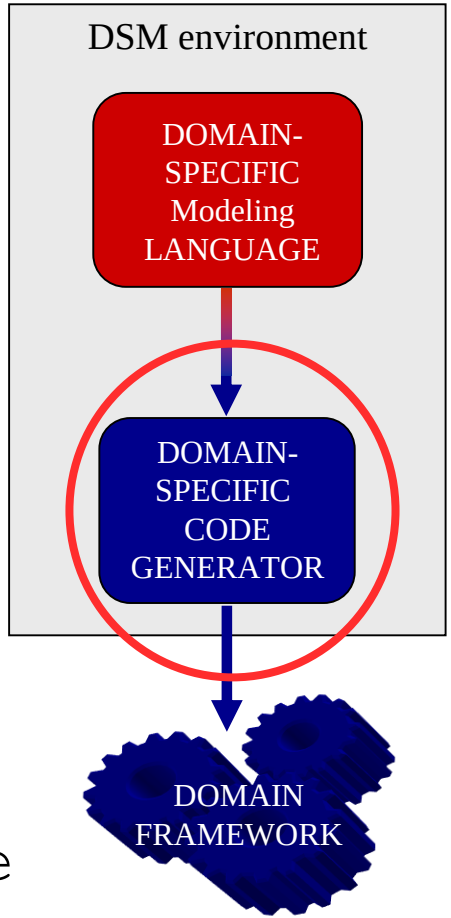
& Kelly, S., blog: https://www.metacase.com/blogs/stevek/blogView?entry=3385914921

Domain-Specific Generator

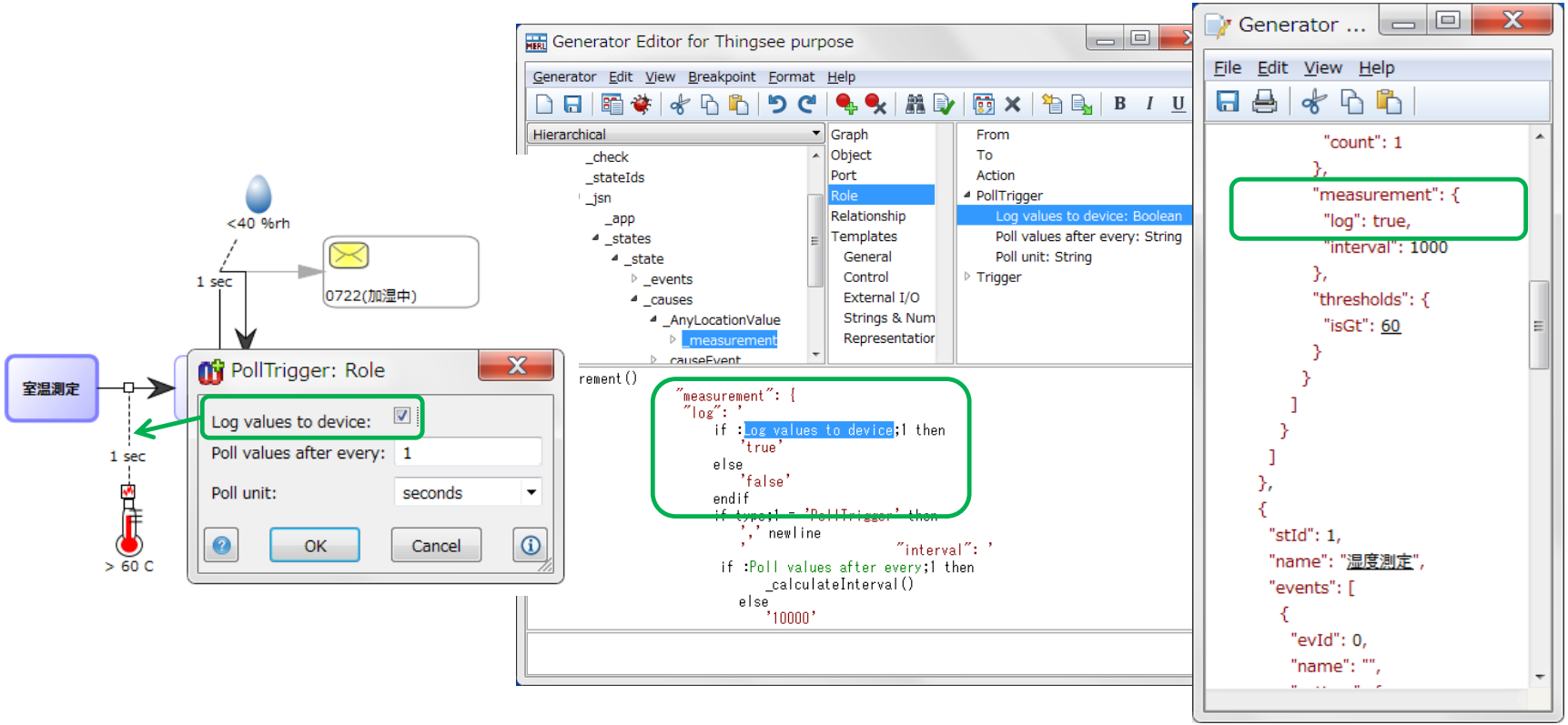
- モデルを巡回して要素から文字列に変換
  1. モデルを巡回解析
     → メタモデル定義に従ってナビゲーション
  2. 必要な情報を抽出
     → モデル内の各種エレメントのデータにアクセス
  3. アクセスしたデータをコードに変換
     →あらかじめ定義されたセマンティックスとルールに従って
  4. 様々な出力フォーマット
     → 出力フォーマットを定義可能

- ASCIIテキスト、コード類、設定ファイル、ビルドスクリプト、doc、メトリクスなど
- モデルのグラフィックファイル（.png, .gif or .pct)

MERL: MetaEdit+ Reporting Language



Domain-Specific Generator



MetaEdit+ DSMのジェネレータ定義について

https://www.fuji-setsu.co.jp/files/DefiningGeneratorswithMetaEditPlus.pdf

Scalability, Collaboration

- 開発にコラボレーション（共同作業）は付き物
- そしてMDD では、非常に多くのダイアグラムやモデルを管理しなければならない

シングルユーザレベル

- １つのXMLファイルを
- １人が編集して
- 後から比較(diff)とマージして
- １つの大きなモデルを開くのに数分かかる

コラボレーション対応ツール

- プロジェクトで共通のリポジトリ
- 共同作業で編集して
- 比較（diff）とマージは必要無く
- レイジーローディングを使って、莫大なエレメントを扱える

コラボレーション開発の技術革新
https://www.fuji-setsu.co.jp/files/Collaborative_modeling_and_metamodeling_JP.pdf

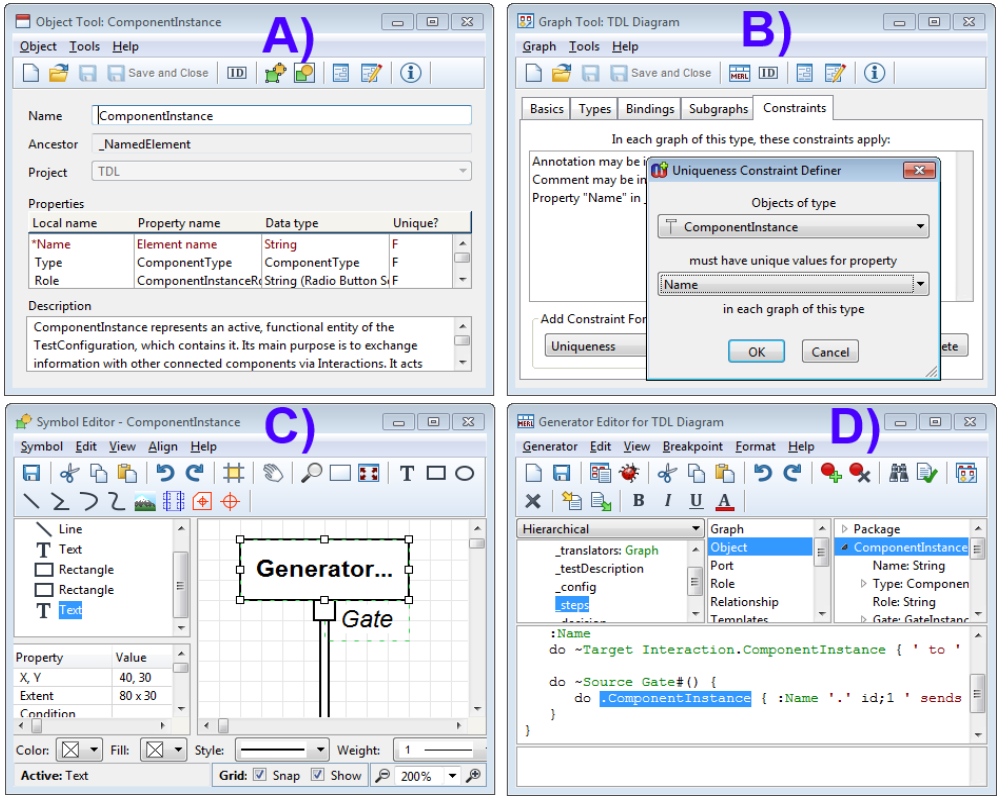
Why collaboration on language engineering?

- ひとつのモデリング言語の定義は、複数のパーツで構成される: 抽象構文、ルール、制約、表記など
  - 一人で全てを満たすことは容易ではない
- モデリング言語定義の正しさ、一貫性、完全性が求められる
- 単一のモデリング言語では不十分で、複数のモデリング言語を統合して使用する必要がある
  - それゆえ複数の担当者が関わることになる

Collaborative metamodeling ETSIのTDL (テスト記述言語)例

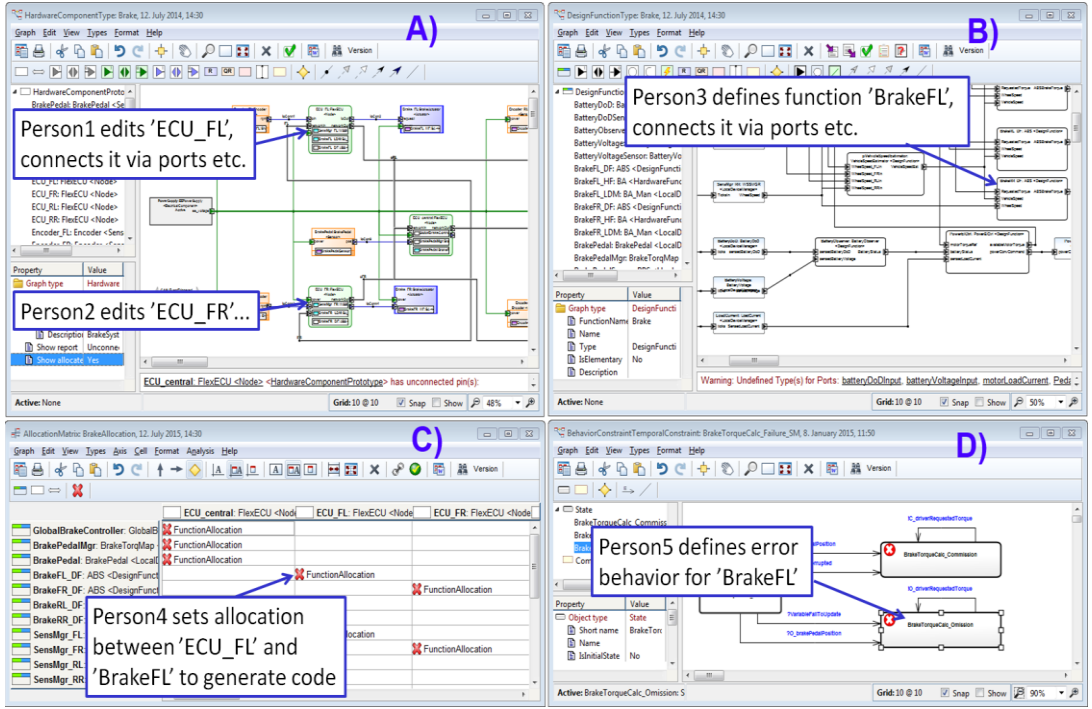
A）ComponentInstanceエレメントの抽象構文 、B）制約やルールの設定、C）ComponentInstanceの具象構文（シンボルなど表記）、D）コンポーネントのインスタンスをアクセスするTDLのジェネレータの定義

全てのモデリング言語パーツはツール内で上手く統合されるので、何らかの変更を自動反映することやトレースすることができる（例えば抽象構文への変更に対して、制約 B）、表記 C）、ジェネレータ D）へ）



Collaborative modeling
車載システムのモデル開発の共同作業

A) 2人の担当者が同じモデル図内の異なるハードウエア部品を編集。同時にB)でシステムの論理コンポーネントをアーキテクトが定義していて、C)ではコードを生成させるために、A)とB)でまさしく編集されているハードウエアと論理アーキテクチャ間のアロケーションを定義している。そしてD)では機能安全担当がB)で定義される論理コンポーネントのエラーモデルを定義。

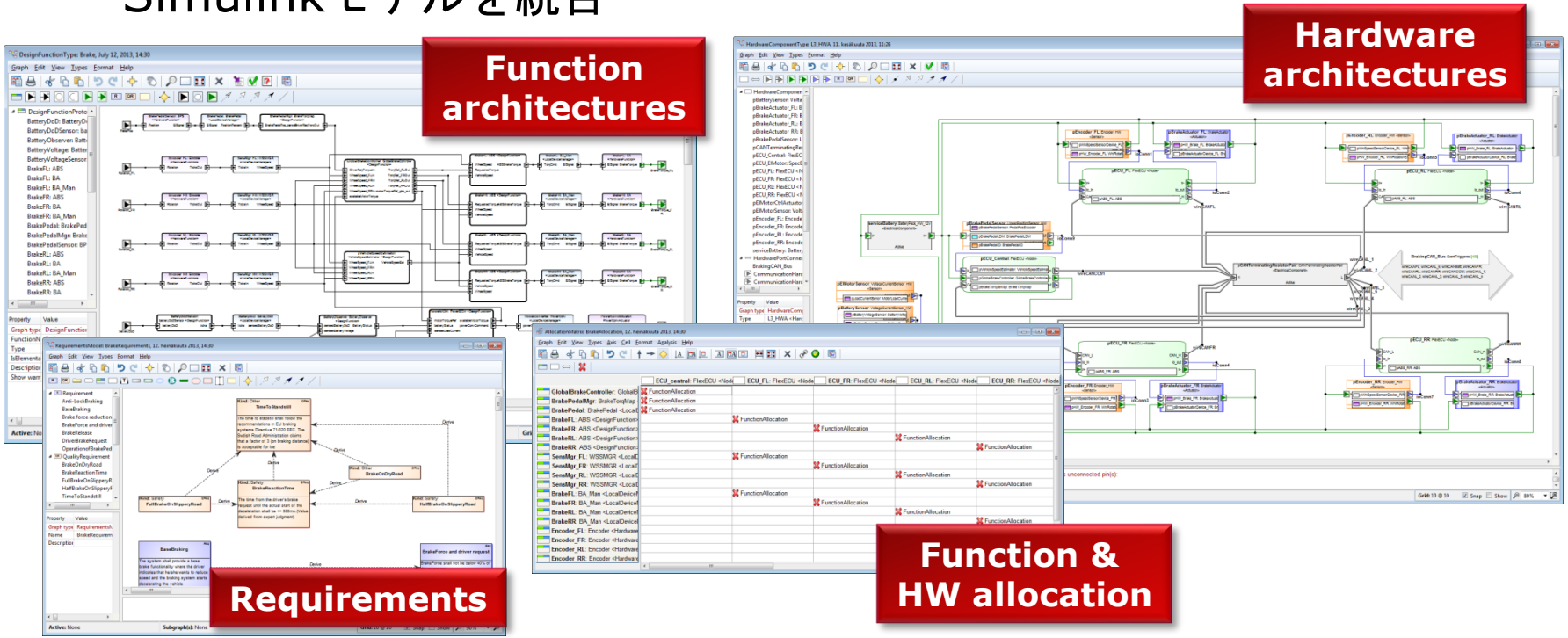


Benefits of collaborative modeling

- 同じデザインスペースで同時並行して作業ができる
  - フィードバックや意見を求めたい場合は、メンバー全員が同じモデル（あるいはモデルエレメント）を即座に参照して、アップデートすることができる
- 比較（diff）とマージ作業で衝突を避けるための労力や時間が必要無い
- 必要となる全てのモデル情報を得ることができるので開発が加速される
- 異なるモデリング言語の様々なビューの作業を、モデルレベルに統合できる
- モデルビューやエレメント間でトレース可能
- モデルは早期段階でチェックして検証される。これは単一ダイアグラム内に留まるのではなく、異なる言語を用いて開発される大きなモデルに対しても同じ

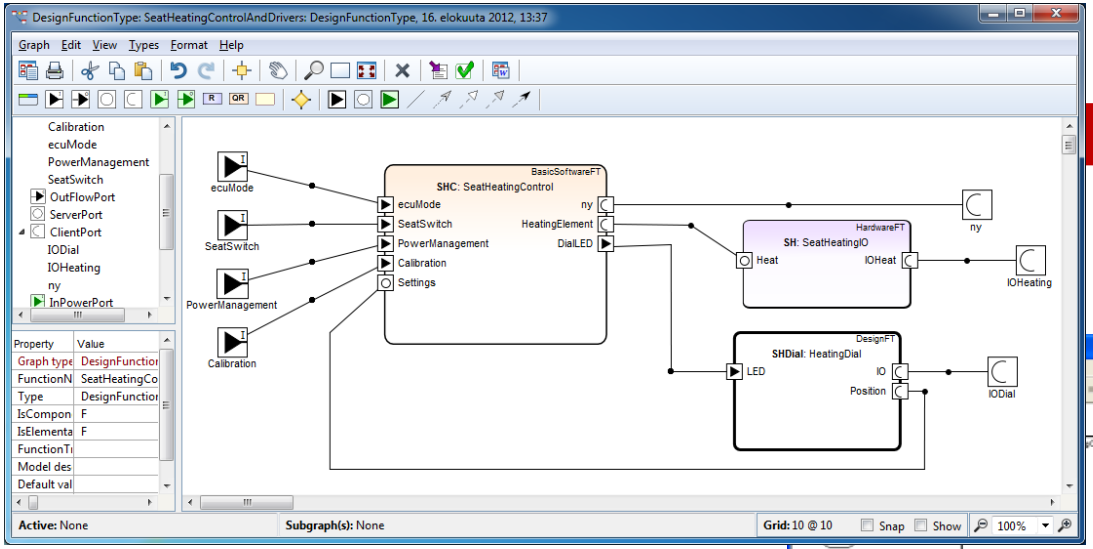
Case 11: Automotive architecture design

- MetaEdit+ で様々なデザインアーキテクチャビューや Simulinkモデルを統合



https://www.fuji-setsu.co.jp/products/MetaEdit/EAST-ADL.html

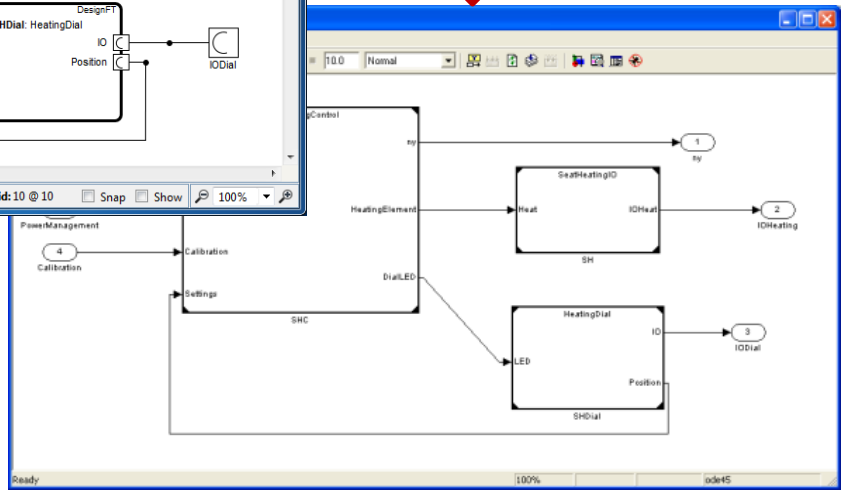
Generates Simulink from architecture



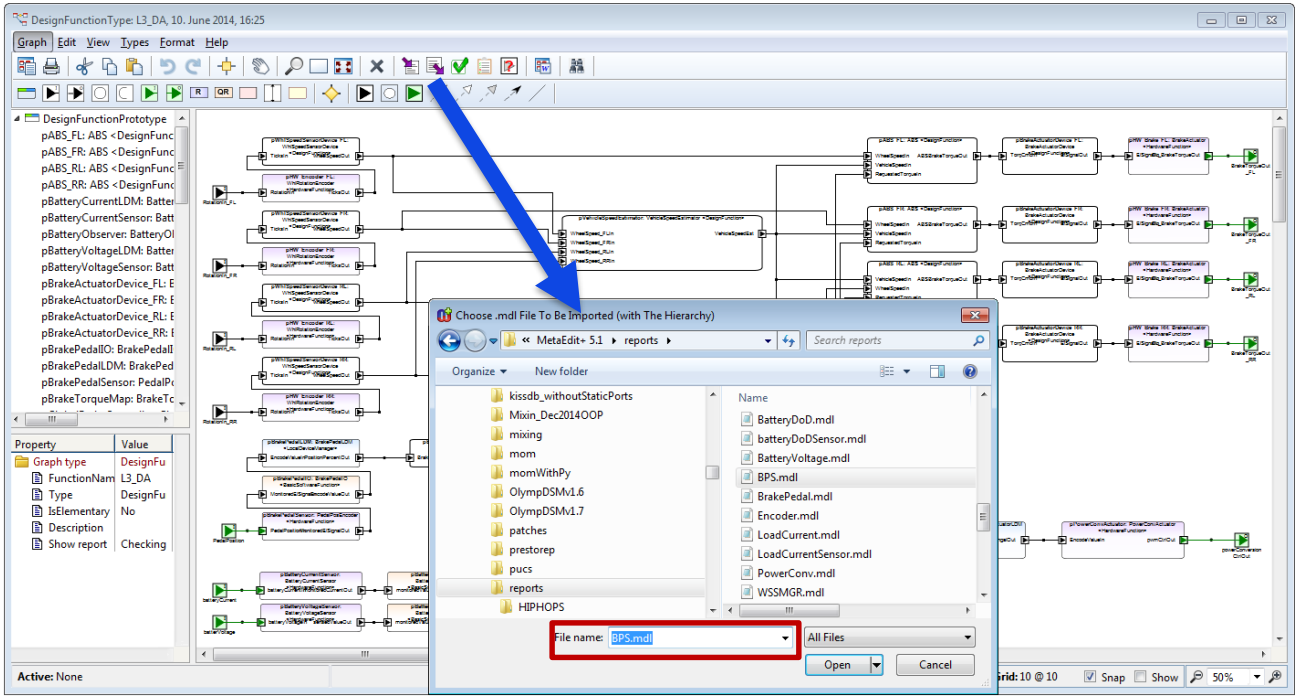
MetaEdit+

Creates automatically Simulink models via API or directly .mdl files

Simulink



Importing from Simulink



Benefits of architecture design with MetaEdit+

- 重複作業を排除
  - 変更は全ての成果物に同時反映され通達もされる
- コラボレーション開発
  - 比較（diff）とマージの必要無く、共同開発できる
- 全プロセスに渡ってのトレーサビリティ
  - 要件からコードまで全ての成果物でバイディレクショナルに
- モデル変換・コード生成
  - Simulinkモデル、Cコード、各種ドキュメント
- 様々なツールと連携
  - 定理証明、テストベクタ生成など
- 組織ごとの需要に応じてカスタマイズ
  - ルールや制約の追加、各種ジェネレータ、ツール統合など MetaEdit+なら柔軟に対応できる

SysML with MetaEdit+

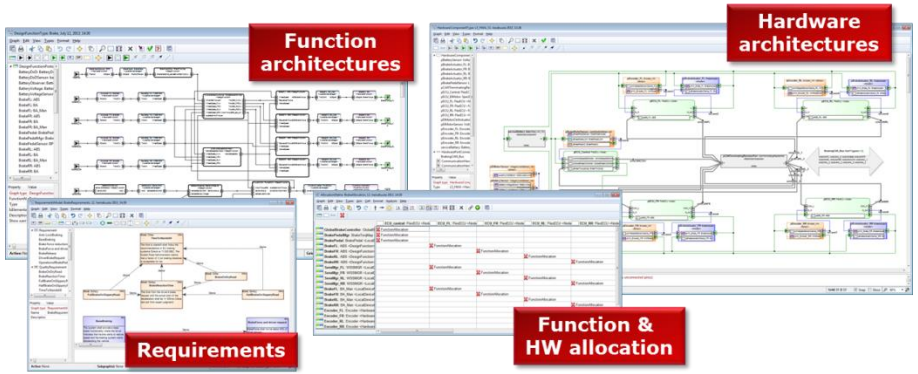
- 車載システム

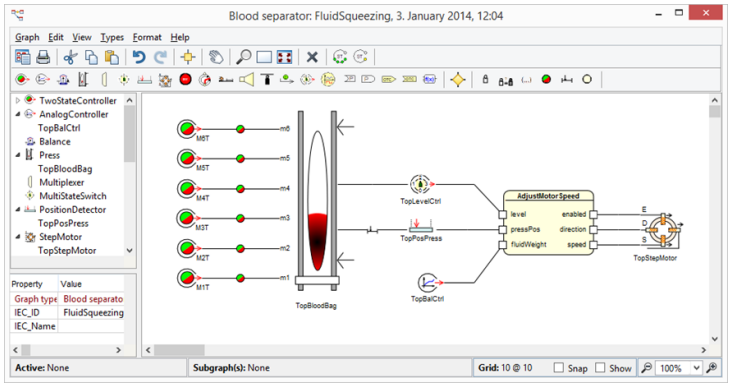
=> EAST-ADL へ

- 車載以外

=> MetaEdit+でSysML を各ドメインのコンセプトで拡張

=> ツールは既存UMLツールのまま、各ドメインのコンセプトで拡張 （Case 6 のように）



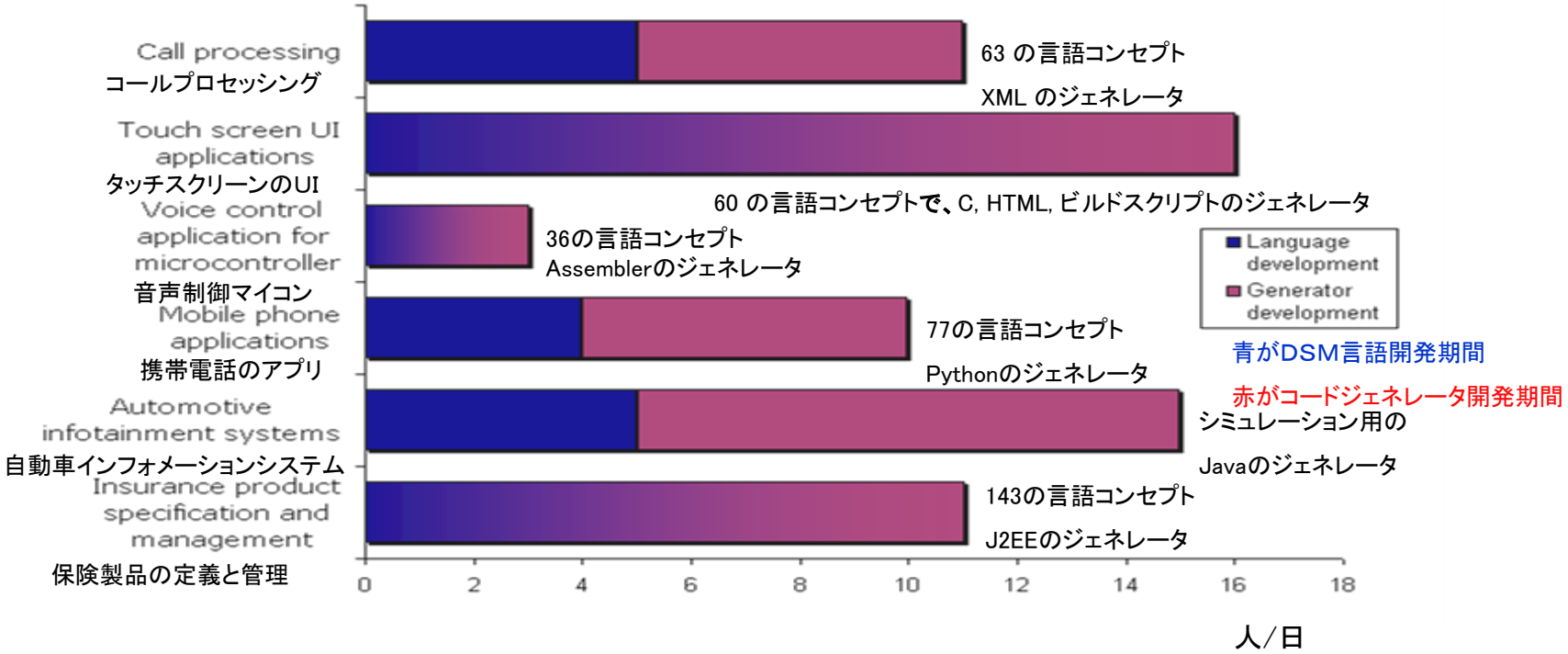


Summary

- ドメインスペシフィックモデリングで、生産性と品質を飛躍的に向上
- 専用のモデリング言語とジェネレータを作るなら、必要な部分のみを定義して、残りはツールで自動化させること
- Special attention to:
  - 言語定義の品質を改善し続ける
  - ユーザが関与するインクリメントな開発
  - ドメインの継続的な変化に備える
  - ジェネレータの開発プロセスと生成速度に留意する
  - コラボレーション開発、大規模モデルを支える拡張性

DSMの開発には数年・数億円かかる？

適切なツール（ MetaEdit+ ）なら通常数週間



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

MetaCase

本講演は Code Generation conference の基調講演を基にしています

The Business Cases for Modeling and Generators
https://www.infoq.com/presentations/modeling-language-generator/

MetaCase

Thank you!

ET2015:小間番号(C-03)
富士設備工業(株)ブースで展示しています

Industry experiences

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た”

EADS “モデリング言語に備えたルールでモデルのエラーを排除できたので、生成するコード品質が明らかに改善された”

POLAR “従来のアプローチと比較して、開発が7.5倍速くなり、生成されるコードだけでなく、開発プロセスの質も飛躍的に改善した”

DENSO “MetaEdit+ を用いることで、ソフトウエア開発の外部委託を削減できるようになった。
またAUTOSAR のバージョンが変更されても、それに対する修正が簡単に行えて、実装とテストの作業が短縮された”