ソフトウェア工学 108－2
（1996．3．22）

ソフトウェア自動合成シェル SOFTEXSHELL を用いたドメイン指向ソフトウェア開発プロセス設計

佐藤 明良，友部 実，山之内 徹，渡辺 正信

NEC C&C 研究所

〒216 神奈川県川崎市宮前区宮崎 4-1-1

{*a-sato, tomobe, yamano, watanabe*}*@swl.cl.nec.co.jp*

本稿では、開発下流工程（設計からコーディング・テスト工程）を対象にして、開発方法や開発プロセスを開発する活動である“メソッドエンジニアリング”を支援する方法として、ソフトウェア自動合成シェル SOFTEXSHELL を用いた方法を提案する。また、チェーンストア管理システムの GUI ソフトウェア開発プロセス設計への適用を通して、メソッドエンジニアの特質、および、提案する方法の特徴について述べる。メソッドエンジニアは、特に、(1) 要求 / ドメイン分析能力、(2) メソッド設計評価能力、および、(3) 開発管理・メソッド普及能力が要求される。実際に適用した結果として、提案する方式は、メソッド設計結果の確認およびプロトタイピングを支援可能であり、また、ジェネレータによりメソッド普及も支援される。これらにより、適用例では結果として、アプリケーションエンジニアリングにおいて、生産性向上 (13.7 人年コスト削減) および納期短縮が達成され、提案方式の有効性を確認した。

Domain-Oriented Software Process Modeling using Software Synthesis Shell SOFTEXSHELL

Akiyoshi Sato, Minoru Tomobe, Toru Yamanouchi, Masanobu Watanabe

C&C Research Laboratories, NEC Corporation

4-1-1, Miyazaki, Miyamae-ku, Kawasaki, Kanagawa 216, Japan

{*a-sato, tomobe, yamano, watanabe*}*@swl.cl.nec.co.jp*

This paper proposes a method for *method engineering* using software synthesis shell SOFTEXSHELL, and describes an application for designing of software development process for a chain-store management system. Through this application, we examine the role and skills of the method engineer and the characteristics of the proposed method. The method engineer should have the ability of (1) requirements/domain analysis, (2) method design/evaluation, and (3) development management and method guidance. As a result of the application, the proposed method has been able to support method evaluation and method guidance for the application engineers. The development cost has been reduced by 13.7 man-years, and the development term has been shorten.

1 はじめに

ソフトウェア開発とは、単にターゲットとなるアプリケーションソフトウェアを開発することだけではなく、その開発プロセス設計や開発方法 (メソッド) 設計、および、ツール開発も含めて考えることが、開発生産性向上や品質確保のために、重要である。このような、アプリケーションソフトウェア開発に先立って、その開発方法およびプロセスや開発支援ツールを開発する活動はメソッドエンジニアリング (*method engineering*) [4] と呼ばれており、ターゲットソフトウェア自身を開発する活動であるアプリケーションエンジニアリング (*application engineering*) とは区別されている[1]。つまり、アプリケーションエンジニアはメソッドユーザである。

メソッドエンジニアは、アプリケーションエンジニアに対して、開発方法を提示し、どのように設計開発を進めればよいのかについて責任がある[2]。一般に、ウォータフォールモデルや OMT (*Object Modeling Technique*) 技法を用いるなどという一般論だけでは、実際にソフトウェア開発プロジェクトを効率的に遂行することは困難である。よりドメイン依存、または、状況依存として、顧客要求を満足するような、開発方法を具体的に設計開発する必要がある。つまり、例えば、ソフトウェア開発プロジェクトの最重要要件として納期重視があれば、機能や性能を多少落しても、短期間に開発を完了するような方法およびプロセスを用いるべきであるし、性能重視という要件ならば性能評価を随時詳細化しながら設計開発を進めることが重要となるであろう。

本稿では、ソフトウェアアーキテクチャ[3]が決定した後、開発下流工程 (設計からコーディング・テスト工程) を対象にして、ドメインや状況に依存して、メソッドエンジニアリングを支援する方法として、ソフトウェア自動合成シェル SOFTEXSHELL[4] [7] [6] を用いた方法を提案する。また、この方法を大規模なチェーンストア管理システム開発へ適用した結果 [5] について述べる。特に、メソッドエンジニアの役割および期待される資質 (スキル) について議論し、提案する方法がどのように貢献するのかについて分析する。これらについて、我々が経験した実例を用いて説明する。

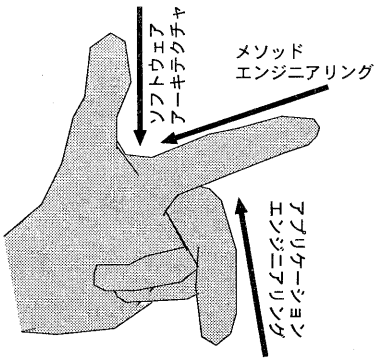

図 1: ソフトウェアアーキテクチャ、メソッドエンジニアリング、アプリケーションエンジニアリング

2 ソフトウェア自動合成シェル SOFTEXSHELL を用いたメソッドエンジニアリング

2.1 メソッドエンジニアリング

メソッドエンジニアリングとは、アプリケーションソフトウェアを開発する作業に先立って、その開発方法 (メソッド) や開発プロセス、ツールを設計開発する活動である (図 2)。つまり、顧客要求を入力として、ドメインや状況を考慮し、開発方法、プロセス、ツールを出力する活動である [4] [8]。

ここで、メソッド、プロセス、ツールとは、以下のような意味で用いている:

- メソッド = 開発者間の静的な関係、入出力インタフェース
- プロセス = 開発者のコラボレーション、開発作業のシナリオ
- ツール = 開発自動化機能、または、開発者の作業支援機能

つまり、メソッドは、静的に、どのような開発者がどのような形式 (構文や意味) を入出力とするのか

[1] CMU/SEI の *Model-Based Software Engineering* では、アプリケーションエンジニアリングと、ドメインエンジニアリングの 2 つの活動に分類している [8]。

[2] メソッドエンジニアリングの対象はアプリケーションエンジニアであり、アプリケーションエンジニアに対して、「あなたはコレコレをやりなさい」と指示する "人指し指" のような役割を果たす (図 1)。

[3] ソフトウェアアーキテクチャの設計は重要な問題であり (図 1)、メソッドエンジニアリングとも関連があるが、別稿にて議論することとする。

[4] 従来は、SOFTEX/S と呼ばれていたが、同じものである。

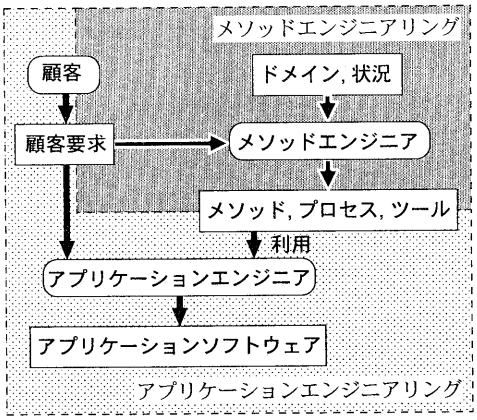

図 2: メソッドエンジニアリングの概念図

を規定する。プロセスは、時間軸を伴い、開発者間での協調作業や仕様記述からアプリケーションプログラムを開発する作業シナリオ (ワークフロー) を規定する。ツールは、開発者の作業を一部支援する機能であり、開発自動化の観点ではプログラムジェネレータや仕様チェッカなどに相当する。これらのメソッドエンジニアリングの結果 (メソッド、プロセス、ツール) を受けて、アプリケーションエンジニアは、顧客要求を満足するようなアプリケーションソフトウェアを開発する。

2.2 メタ CASE ツールとしてのソフトウェア自動合成シェル

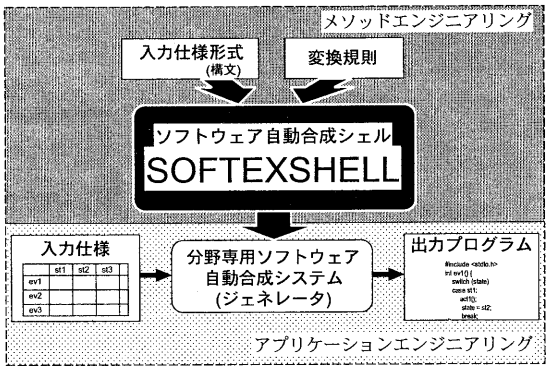

図 3: SOFTEXSHELL の概念図

メソッドエンジニアリングの方法として、ソフトウェア自動合成シェル SOFTEXSHELL [7] [6] を用いる方法を提案する。SOFTEXSHELL(ソフテックスシェル) は、筆者らが開発したシステムであり、多ソート項書換えシステム (*many-sorted term rewriting system*) [2] に基づき、入力構文定義 (シグネチャ定義) と変換規則定義 (項書換え規則) から、プログラムジェネレータを自動合成するシステムである (図 3)。プログラムジェネレータは、アプリケーションエンジニアリングにおいて用いられる CASE (*computer-aided software engineering*) ツールであるので、SOFTEXSHELL は CASE ツールを自動合成するメタ CASE ツールとして位置付けられる。

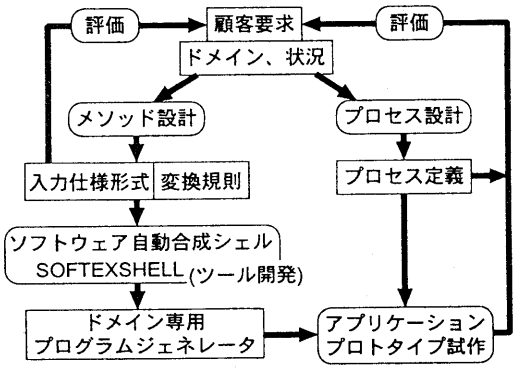

図 4: SOFTEXSHELL を用いたメソッドエンジニアリング

SOFTEXSHELL を用いたメソッドエンジニアリング方法は以下のようになる (図 4):

1. [メソッド設計] 顧客要求およびドメインや状況を入力とし、メソッド設計を行ない、顧客要求を満足するかを評価する。各開発者およびジェネレータの入出力構文、および、プログラムジェネレート手順を変換規則として定義する。
2. [プロセス設計] 顧客要求およびドメインや状況を入力とし、プロセス設計を行ない、顧客要求を満足するかを評価する。各開発者およびジェネレータがどのように協調して、どのようなシナリオで全体の開発が進むのかを時間軸で示したプロセスを定義する。
3. [ツール開発] 仕様チェッカやプログラムジェネレータなどの CASE ツールは、ソフトウェア自

動合成シェルによって、入力仕様形式定義と変換規則定義から自動合成される。

次節で上記の方法を、実際に、チェーンストア管理システム開発へ適用した例について説明する。

3 例 – チェーンストア管理システム開発プロセス設計

3.1 チェーンストア管理システム

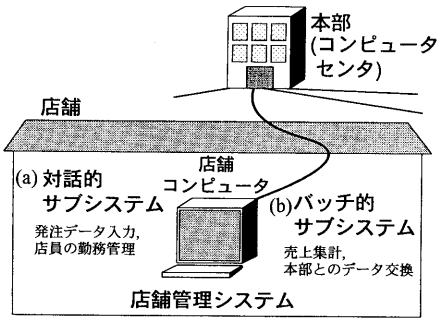

図 5: チェーンストアの店舗管理システム

チェーンストア管理システム (店舗管理システム) の概念図を図 5に示す。このシステムは、各店舗に設置されている店舗コンピュータ上で動作し、店員が、発注、在庫管理から勤務管理など日常行なわれる店舗管理の全てを行なう。それぞれの店舗コンピュータは本部 (コンピュータセンタ) のホストコンピュータとネットワークによって接続されている。

この店舗管理システムの機能は 2 つの機能から構成される: (a) 対話的サブシステム: 発注データ入力や店員の勤務管理など、 (b) バッチ的サブシステム: 売上集計処理やホストコンピュータからのマスタデータ送信など。特に、対話的サブシステムは、GUI (graphical user interface) を持つことが特徴であるため、 GUI プログラムおよびコールバックプログラムの開発が主となる。

この例題における、主たる顧客要求、および、ドメインおよび状況の特徴について以下に示す[5]:

- **顧客要求:** 生産性向上、納期短縮
- **ドメイン:** 複雑な GUI 画面が約-200 枚、X Window, UNIX ベースシステム
- **状況:** 開発者が COBOL プログラマ多数 (X Window や C 言語は未経験)、ソフトウェアアーキテクチャは新規開発[6]

一般的には、上記の顧客要求を満たすことは、困難な状況にある。なぜならば、 COBOL プログラマに X Window や C 言語の教育から行なうか、 COBOL ベースの GUI ビルダを開発するか、いづれにしても大きなコストを要するからである。

3.2 メソッド設計

まず、メソッド設計に於いて、メソッドエンジニアは、以下のような判断を下した:

- **[仮定 1]** 開発ツールは SOFTEXSHELL を用いて開発するので、低コストで作成可能
- **[仮定 2]** (GUI ビルダ利用で削減されるコスト) > (COBOL ベース GUI ビルダ作成コスト)

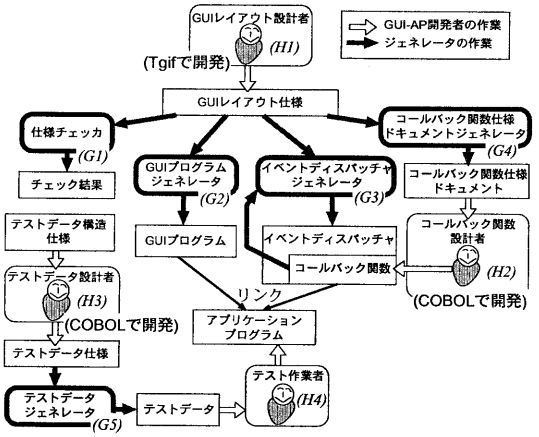

図 6: 店舗管理システム開発のメソッド概念図

つまり、メソッドエンジニアリングに多少のコストをかけても、アプリケーションエンジニアの能力をフルに発揮させれば、全体的なコストダウンおよび納期短縮につながるという判断 (仮定) をしたのであ

[5]ここに示したものは、説明の都合上、単純化している。

[6]実際には、メソッド設計と同時、または、これに先立って、ターゲットソフトウェアのソフトウェアアーキテクチャを決定する必要がある。これは重要な問題である。しかしながら、今回は、メソッドエンジニアリングとは別途に、アーキテクチャ設計を行なった。

る。メソッド設計の結果、図6のような体制とした。つまり、設計者として:

$H1$: GUI レイアウト設計者 (Tgif で開発)
$H2$: コールバック関数設計者 (COBOL で開発)
$H3$: テストデータ設計者 (COBOL で開発)
$H4$: テスト作業者

の 4 種類と、ジェネレータとして:

$G1$: 仕様チェッカ
$G2$: GUI プログラムジェネレータ
$G3$: イベントディスパッチャジェネレータ
$G4$: コールバック関数仕様ドキュメントジェネレータ
$G5$: テストデータジェネレータ

の 5 種類の入出力構文を決定し、それぞれの役割を決定した[7]。

図6に示す**メソッドの評価ポイント**としては:

- アプリケーションエンジニアリングにおいて、**COBOL プログラマの能力がフルに発揮されるために、高い生産性**が実現される
- **5 種類のジェネレータを導入すること**により、開発自動化され、**高い生産性**が実現される

である。これにより、上述したドメインや状況を考慮しても、顧客要求を満足すると評価 (予測) した。この評価は、COBOL プログラマの過去の標準的な生産性基準を基にした。また、ジェネレータ導入効果予測は、ジェネレータのプロトタイプを開発し、試用した経験を基にした。

3.3 プロセス設計

時間軸を考慮して、図7に示すような開発プロセスを設計した。

GUI レイアウト設計を、ボタン部品やグラフ部品などを配置する画面概要構成を設計する概要設計と、色や位置などを決定する詳細設計とに分離し、レイアウト詳細設計とコールバック関数設計およびテストデータ設計を並行して行なえるようにした。図7に示す**プロセスの評価ポイント**は:

- **並行開発**により、**納期短縮**が実現される
- **5 種類のジェネレータ利用**により、**高い生産性**および**納期短縮**が実現される

[7]実際には、各ジェネレータの入出力構文は、関数記号定義とソート定義から構成されるシグネチャ (*signature*) によって、インタフェースが定義される [5] [7]。

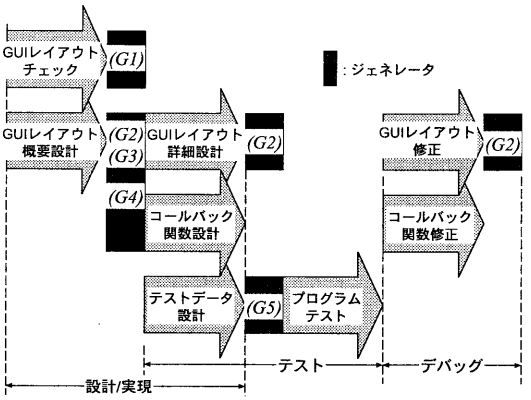

図 7: 店舗管理システム開発のプロセス概念図

である。これにより、上述したドメインや状況を考慮しても、顧客要求を満足すると評価 (予測) した。この評価は、試作したジェネレータを用いた上記プロセスの小規模実験を行なった結果による。

3.4 ツール開発

上述した 5 種類のジェネレータ **(ツール) 開発における評価ポイント**は、上述した 2 つの仮定がなり立つかどうかである。つまり、ツール開発コストが、適用効果 (ジェネレータ導入により削減されるコスト) より小さくなくては意味がないのである。

SOFTEXSHELL を用いたジェネレータ開発では、

- 項書換えシステムは副作用がないので、**変換規則モジュールの再利用が比較的容易**
- 変換途中結果は、項表現として表現されるので、**変換規則の段階的開発が比較的容易**

という特徴により、比較的小さいコストで開発できた [5]。実際、5 つのジェネレータは、図8に示すような 7 つのサブジェネレータ ($SG1$〜$SG7$) の組合せにより実現されており、サブジェネレータ共有 (5 つのジェネレータ間での再利用) により、これら開発コストは 14.5 人月であった (表1)。よって、結果として、上述した **[仮定 1] は満足された**といえる。[仮定 2] に関しては、後述するアプリケーションエンジニアリングでの評価結果で評価する。

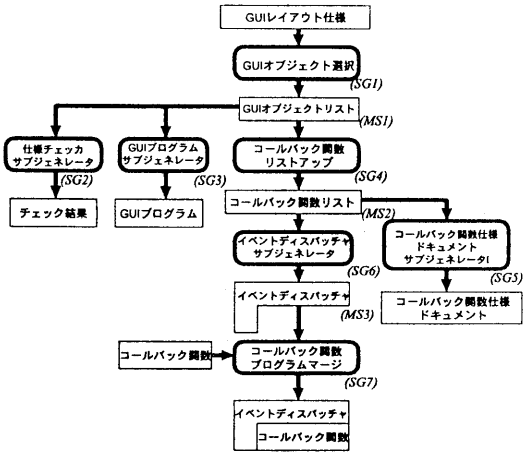

図 8: ジェネレータの構造

3.5 アプリケーションエンジニアリングでの評価

上記のメソッド、プロセス、ツールを用いて、チェーンストア管理システムの開発を行なったところ、(1) 開発したGUI画面数が189、(2) 5つのジェネレータで自動合成されたソフトウェアライン数が約40万行、(3) 従来比較で、13.7人年のコスト削減、生産性約3倍、(4) 納期内で完成、という評価結果となった[5]。生産性の算出根拠は、表1, 2, 3より、以下のようである:

(トータルコスト削減)
= (ジェネレータによる削減コスト) … (表2)
− (ジェネレータ開発コスト) … (表1)
− (ジェネレータ適用コスト) … (表3)
= 184.2人月 − 14.5人月 − 5.1人月
= 164.4人月 (= 13.7人年)

ここで、ジェネレータによる削減コストは、自動合成ソフトライン数と開発生産性標準(1人月 = 2.2Kライン)から計算した。従って、上述の **[仮定 2] も結果として満足された**といえる。GUI画面仕様記述例を、図9に示す。

4 メソッドエンジニアの特質とSOFTEXSHELLの貢献

提案したSOFTEXSHELLを用いたメソッドエンジニアリング方法を、実際のチェーンストア管理システム開発へ応用した経験から、メソッドエンジニアが持つべき特質について議論する。今回の経験から、メソッドエンジニアは、特に、以下の能力を持つ必要があることがわかった:

1. **[要求 / ドメイン分析評価]** 顧客要求やドメイン、状況の分析・予測・評価能力
2. **[メソッド設計]** メソッドやプロセスの提案、設計・評価能力
3. **[開発管理・普及]** 開発管理、プロジェクト管理能力 − 方法論の普及・徹底
4. **[基礎知識]** 方法論の知識、ツール開発能力

表 1: ジェネレータ開発コスト

Generators	Language	Generator size (rules)	Develop. cost (man-months)
$SG1$	rule	580 (8.8K lines)	5.5
$SG2$	rule	53 (0.4K lines)	0.5
$SG3$	rule	399 (3.5K lines)	3.0
$SG4$	rule	472 (5.1K lines)	1.5
$SG5$	rule	99 (0.9K lines)	0.3
$SG6$	rule	754 (9.9K lines)	2.0
$SG7$	C	532 lines	0.2
$G5$	rule C	33 (0.4K lines) 3055 lines	1.5
Total	rule C	2500 (29.0K lines) 3587 lines	14.5

表 2: 自動合成ソフトのサイズと予測効果 (コスト)

Generators	Language	Generated software size (K lines)	Ratio	Man-months (1 man-month = 2.2 K lines)
$G2$	C	76.0	18.7%	34.5
$G3$	COBOL	122.5	30.2%	55.7
	C	84.6	20.9%	38.5
$G4$	帳票	26.2	6.5%	11.9
$G5$	COBOL	96.0	23.7%	43.6
Total		405.3	100%	184.2

表 3: ジェネレータの適用サイズと適用コスト

Application function	Application size	Application costs
GUI layout design	189 画面	4.1 人月
Test data design	200 データ種類	1.0 人月

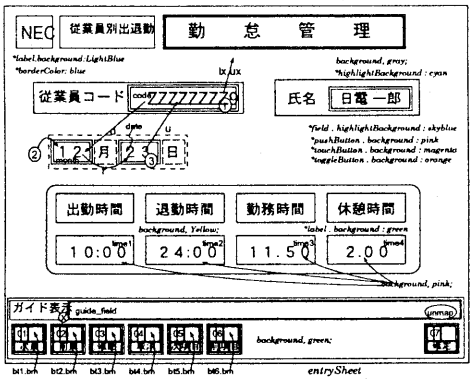

図 9: GUI 画面仕様記述例 (ジェネレータ G1 ～ G4 への入力)

これらは、それぞれ単独でも、要求工学 (*requirements engineering*)、ドメインエンジニアリング、プロジェクト管理、方法論 (オブジェクト技術やフォーマルメソッドなど) といった幅広い分野にまたがっているが、これらの全てに卓越した能力が要求される。しかしながら、これら能力をひとりの人間が全て持っている必要はなく、数名のグループで分担することは可能であるだろう。実際、今回のチェーンストア管理システム開発では、5 ～ 6 名のメソッドエンジニアが分担して担当した。

次に、提案するメソッドエンジニアリング方法における、SOFTEXSHELL の貢献について議論する。SOFTEXSHELL が提供する、項書換えおよびジェネレータの 2 つの側面から論じる。

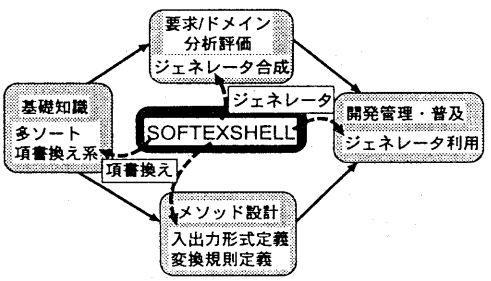

図 10: メソッドエンジニアの特質と SOFTEXSHELL のメソッドエンジニアリングへの貢献

[項書換え] — 各設計者やジェネレータの入出力形式を定義する方法として、多ソート項による定義方法を提供する。つまり、仕様記述言語やダイアグラムを、多ソートシグネチャ (*many-sorted signature*) として定義する方法を規定している。今回のチェーンストア管理システム開発での、ジェネレータ $G1 \sim G4$ の入力仕様形式の例では、図 11に示すような "従業員コード入力フィールド" は、以下に示すような関数記号宣言、および、項表現に対応する。また、「2 重矩形 (box) として描かれた仕様記述は、ラベルウイジェット (label) として認識する」という変換規則は以下のようになる (Xn は変数を表す)。

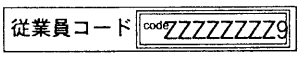

図 11: 入力仕様定義例

```
// 関数記号宣言
op label: name x y labelX labelY fontSizeName label
      borderColor background height width → label.
op field: name x y borderColor fontSizeName template
      borderWidth shadowWidth height width → field.
// 項表現
label(label52x128, 48, 124, 12, 8, medium, "従業員コード",
      blue, LightBlue, 53, 345).
field(code, 204, 128, blue, large, "ZZZZZZZZ9",
      1, 4, 45, 181).
// 2 重矩形 (box) を ラベル (label) として認識する変換規則
eq group(obj(box, X1,X2,X3,X4,X5,X6,X7,X8,X9,X10)
         obj(box, Y1,Y2,Y3,Y4,Y5,Y6,Y7,Y8,Y9,Y10))
     = label(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10).
```

項書換えにより、メソッドエンジニアは、メソッド設計 (およびプロセス設計) において、各ツール間の結合性や設計者が作成する仕様に関して型検査を行なうことが可能となり、**メソッド設計結果の確認 (validation)** を行なうことができる。

[ジェネレータ] — メタ CASE としての SOFTEXSHELL は、ジェネレータ自動合成機能のために、メソッド (およびプロセス) を迅速にプロトタイプ可能とする。従って、「メソッド設計→顧客要求評価」のプロセスを迅速に実行することが可能となり、**メソッドエンジニアリングのプロトタイピング**を支援する。

また、ジェネレータは、開発手順を自動化・ブラックボックス化して内在していると考えられるので、

これをアプリケーションエンジニアリングで利用することは、メソッド (の一部) を自動的に利用することに相当する。つまり、ジェネレータ利用は、**メソッドの普及を支援する**ことになる。

5 関連研究

Imperial College の Bashar Nuseibeh ら [3] [4] は、*Multi-Perspective Software Development* におけるメソッドエンジニアリング方法を提案している。5 つのスロットを持つ *ViewPoints* [1] と呼ばれるフレームワークのうち、仕様表現形式 (*Style*) と開発プロセスのモデル (*Work plan*) を決定する作業をメソッドエンジニアが行ない、残りの 3 つである仕様記述 (*Specification*)、ドメイン記述 (*Domain*) および、開発履歴 (*Work record*) の作成をアプリケーションエンジニアが行なう。*Viewer* とよぶ *ViewPoints* を作成するエディタをメタ CASE ツールとして提案しているが、SOFTEXSHELL のようなジェネレータ開発を自動化する機能は提供されない。

CMU/SEI では、*Model-Based Software Engineering*(MBSE) と呼ばれる技術を開発している [8]。MBSE では、ソフトウェア資産 (*Software Assets*) を開発するドメインエンジニアリングと、そのソフトウェア資産を利用してアプリケーションソフトウェアを開発するアプリケーションエンジニアリングの、2 つを区別している。ドメインエンジニアリングは、本稿で提案するメソッドエンジニアリングに相当し、(1) ドメイン分析、(2) ドメイン設計、(3) ドメイン実現、のプロセスから構成されている。ドメイン設計とは、設計モデル開発プロセスであり、ドメイン実現とは仕様記述言語やジェネレータや再利用コンポーネントを開発するプロセスである。提案する SOFTEXSHELL を利用する方法は、このうち、ドメイン設計およびドメイン実現を支援することに相当する。

6 おわりに

ソフトウェア自動合成シェル SOFTEXSHELL を用いたメソッドエンジニアリング方法を提案し、チェーンストア管理システムの GUI ソフトウェア開発プロセス設計への適用を通して、メソッドエンジニアの特質、および、提案する方法の特徴について述べた。メソッドエンジニアは、特に、(1) 要求 / ドメイン分析能力、(2) メソッド設計評価能力、および、(3) 開発管理・メソッド普及能力が要求されることがわかった。実際に適用した結果として、提案する方式は、メソッド設計結果の確認およびプロトタイピングを支援可能であり、また、ジェネレータによりメソッド普及も支援される。これらにより、適用例では結果として、アプリケーションエンジニアリングにおいて、生産性向上 (13.7 人年コスト削減) および納期短縮が達成され、提案方式の有効性を確認した。

謝 辞

本研究を進めるにあたり、御支援をいただきました 日本電気株式会社 高橋利彦 本部長、大埜 嵩 部長、田中淳一 課長、村岡四郎 課長、および、山本昌弘 所長に深く感謝いたします。また、土' 方雅之氏には熱心に議論いただき、有意義なコメントをいただきました、深く感謝いたします。

参考文献

[1] Finkelstein, A., Gabbay, D., Hunter, A., Kramer, J. and Nuseibeh, B.: Inconsistency Handling in Multiperspective Specifications, *IEEE Transactions on Software Engineering,* Vol. 20, No. 8, August, 1994.

[2] Klop, J. W.: Term Rewriting Systems: A Tutorial, *Bulletin of the European Association for Theoretical Computer Science,* No. 32, 1987.

[3] Nuseibeh, B.: Meta-CASE Support for Method-Based Software Development, *Proc. of 1st International Congress on Meta-CASE,* Sunderland, UK, 1995.

[4] Nuseibeh, B., Finkelstein, A. and Kramer, J.: Method Engineering for Multi-Perspective Software Development, *Information and Software Technology Journal,* Butterworth-Heinemann, February, 1996.

[5] Sato, A., Tomobe, M., Yamanouchi, T., Watanabe, M. and Hijikata, M.: Domain-Oriented Software Process Re-engineering with Software Synthesis Shell SOFTEX/S, *Proc. of the 10th Knowledge-Based Software Engineering Conference (KBSE-95),* pp. 97-104, 1995.

[6] 友部 実, 佐藤 明良, 山之内 徹: 項書換えシステムに基づくソフトウェア自動合成システムの実現, 情報処理学会プログラミング研究会, Vol. 95, No. 114, 95-PRO-4, 4-3, pp.13-18, 1995.

[7] Yamanouchi, T., Sato, A., Tomobe, M., Takeuchi, H., Takamura, J. and Watanabe, M. : Software Synthesis Shell SOFTEX/S, *Proc. of the 7th Knowledge-Based Software Engineering Conference (KBSE-92),* pp. 28-37, 1992.

[8] *Model-Based Software Engineering,* Software Engineering Institute, CMU, *http://www.sei.cmu.edu/technology/mbse/*