プロシューマ中心型ソフトウェアファクトリ
Prosumer-centric Software Factory

松本　吉弘

京都高度技術研究所

http://www5d.biglobe.ne.jp/~y-h-m/

プロシューマ

- 1972年に、Marshall McLuhan およびBarrington Nevittが、「Take Today」とう著書によって、生産的消費者（プロシューマ・prosumer）という概念を提案し、アルビン・トフラーが、著書「第三の波」や「富の未来」のなかで、これを引用した論説を行っていることはよく知られています。ソフトウェアの世界でも、Linux、Wikipediaなどは、プロシューマに近い人たちによって、ここまで推進されてきたと考えられます。今後も、これに類した形で推進されるFree/Libre Open Source Software (FLOSS)は、増えてくると考えられます。

これからの価値創生／再生

- GDP向上（安倍内閣政策）のためには
  - 生産活動より消費活動に価値創生／再生の可能性
  - 民生・消費者社会はアジャイル性大
  - ITに対しては一様化への懸念
    - ITは生産者／発注者(例：行政）が中心になっている
    - ITを消費者中心にする必要
  - だれがIT利用者か
  - 利用者のニーズをソフトウェアに反映する経路が長い

    利用者 → 発注者 → 製作者
  - 利用者が直接ニーズをソフトウェアに反映すべき
  - われわれのソフトウェアファクトリは
    - 米国の言うTime-to-marketではなく、Time-to-valueを目指す

顧客･利用者と開発者の新しい関係

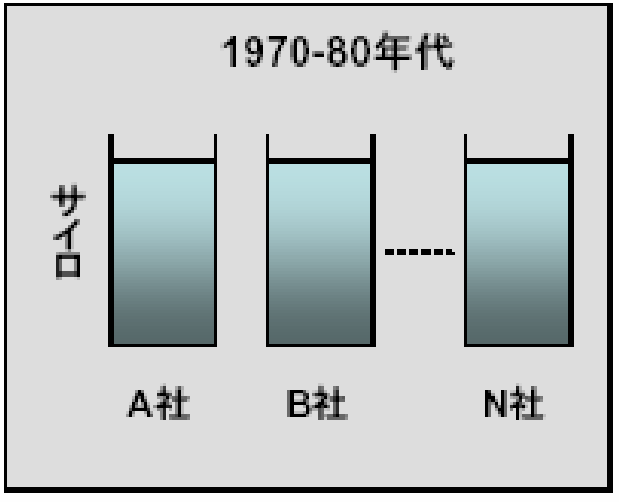
- 注文によって開発･保守されるアプリケーション･ソフトウェアにおいても、前後関係 (context)、問題、ニーズを直接肌で感じている顧客、またはユーザと称される人たちが、全部とまではいかなくても、一部を直接自分たちで責任をもって容易に、かつアジャイルに開発･保守できれば、より大きく事業利益の拡大に貢献できると考えられます。たとえば、毎日、販売価格を決定する方法を変更する必要がある量販店での価格決定ソフトウェア開発･保守などは、その対象になるのではないでしょうか。
- これからは、企業体は別個であっても、顧客･ユーザと開発者の垣根を取り払った新しいコラボレーション形態によるソフトウェア･サービス事業小集団 (筆者は、ソフトウェア･セル、またはアジャイルセルとよぶ) が生まれてくる時代と考えられます。
- 新しいソフトファクトリは、このような将来の事業形態を支援できるようなものにしたい、と考えます。

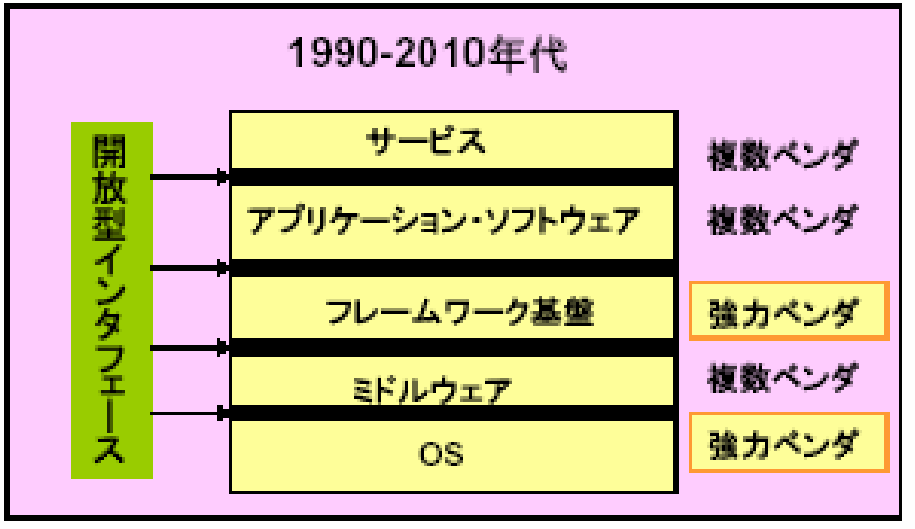
未来からの要請

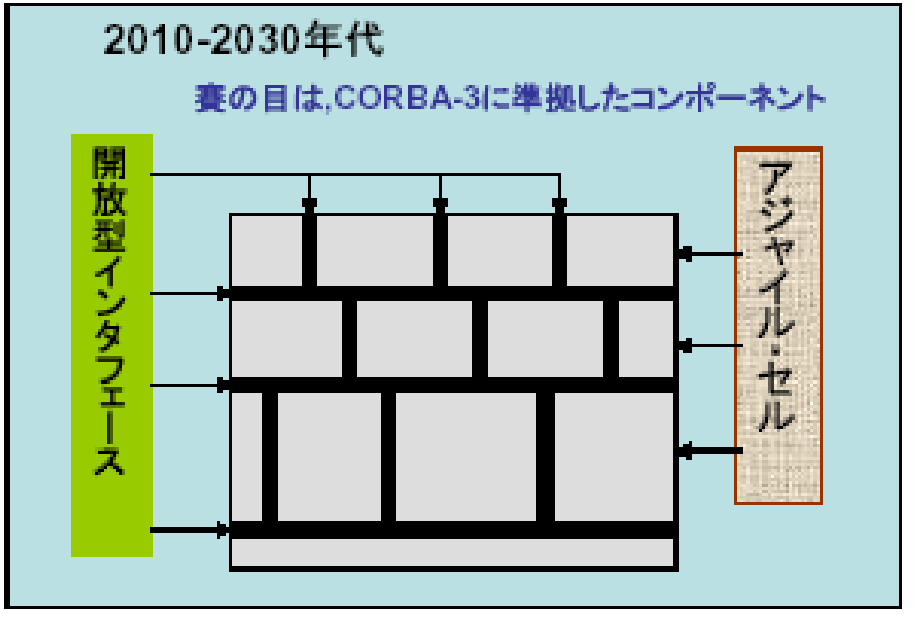
- 富の未来（アルビン・トフラー）の具体例
  - 消費者生産
    - スーパーマーケットで、買い物客が自分で勘定・支払い
    - ATM
    - Linux, Wikipedia

- ソフトウェアファクトリの未来
  - 消費者生産型ソフトウェアファクトリ　Prosumer-centric Software Factory
    - ニーズをもった利害関係者が直接手を下せるソフトウェアファクトリ
  - マス・カスタマイゼーション
    - 安価で個性のある製品をある特定のニッチに対して提供する
  - 利害関係者の自己組織化
    - コンポーネント組み合わせ爆発の回避
  - 自然定着
    - 随意移行
    - Minimally Invasive Transition (例：LSI Logic Inc.)

ソフトウェア実装構成の変遷







垂直統合、水平統合からドメイン･コンポーネント統合へ

- サイロ型：　垂直統合　→　日本型
  - 未知ドメインを創出しなければならない問題に適する。
    - 東電との故障復旧システムに関する協働小集団による開発
    - 九電との故障検出推論システムに関する協働小集団による開発
    - 特定企業に利する場合、談合や独占禁止法違反として排除されるようになった（グローバライゼーション）。
- 共通水平基盤型：　水平統合　→　米国型
  - 産業･雇用の増進、ベンチャー企業の創出に適する。
    - 事業の水平分断をもたらし、変化への応答およびターンアラウンドを長くする。
- ドメイン･コンポーネント統合型：　単位ドメインと称する価値創出／再生を目的とする最小単位資源を開放型基盤上で統合する。
  - 単位ドメインは、ステークホルダが協働する小集団（筆者はセルとよぶ）によって開発･保守される。

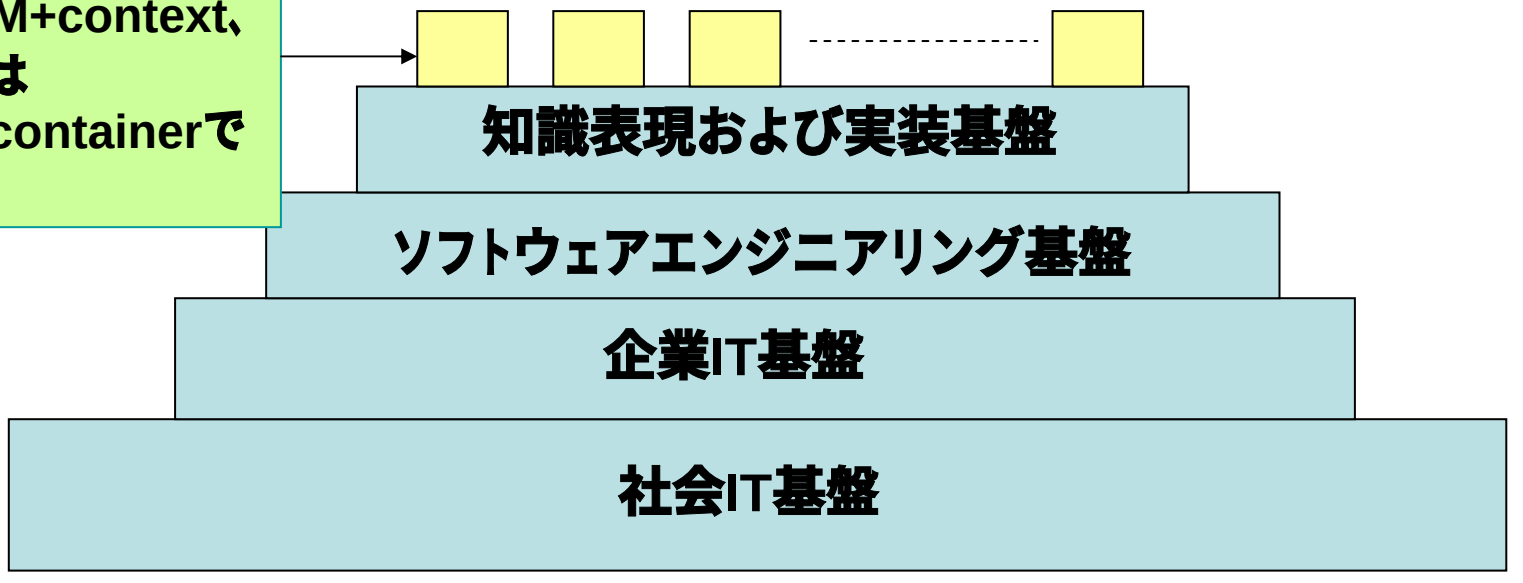
共通水平基盤型フレームワークの問題

- 問題： 数少ない強力な共通水平基盤提供者によって、システム･フレームワークが寡占される。
- 問題： システム･フレームワーク提供者が、コンポーネントまで提供しなければならないような商環境を形成する。
- 是正策：
  - ドメイン･コンポーネントを、基盤から完全に独立して開発･保守できるようにする。　→　CORBA Component Model (たとえば、COM+、COM context)
  - だれでも実現／実装できるような水平／垂直統合のための共通基盤の提供　→　たとえば、ウェブ／モーバイルクライアントで、グーグルが提供している環境
- ソフトウェアファクトリは： システム･フレームワークだけでなくドメイン･コンポーネントのテンプレートまでも提供し、その上で顧客／ユーザの価値創生／再生のための随意性を、高生産性／高品質／short time-to-valueで実現･実装するための環境提供

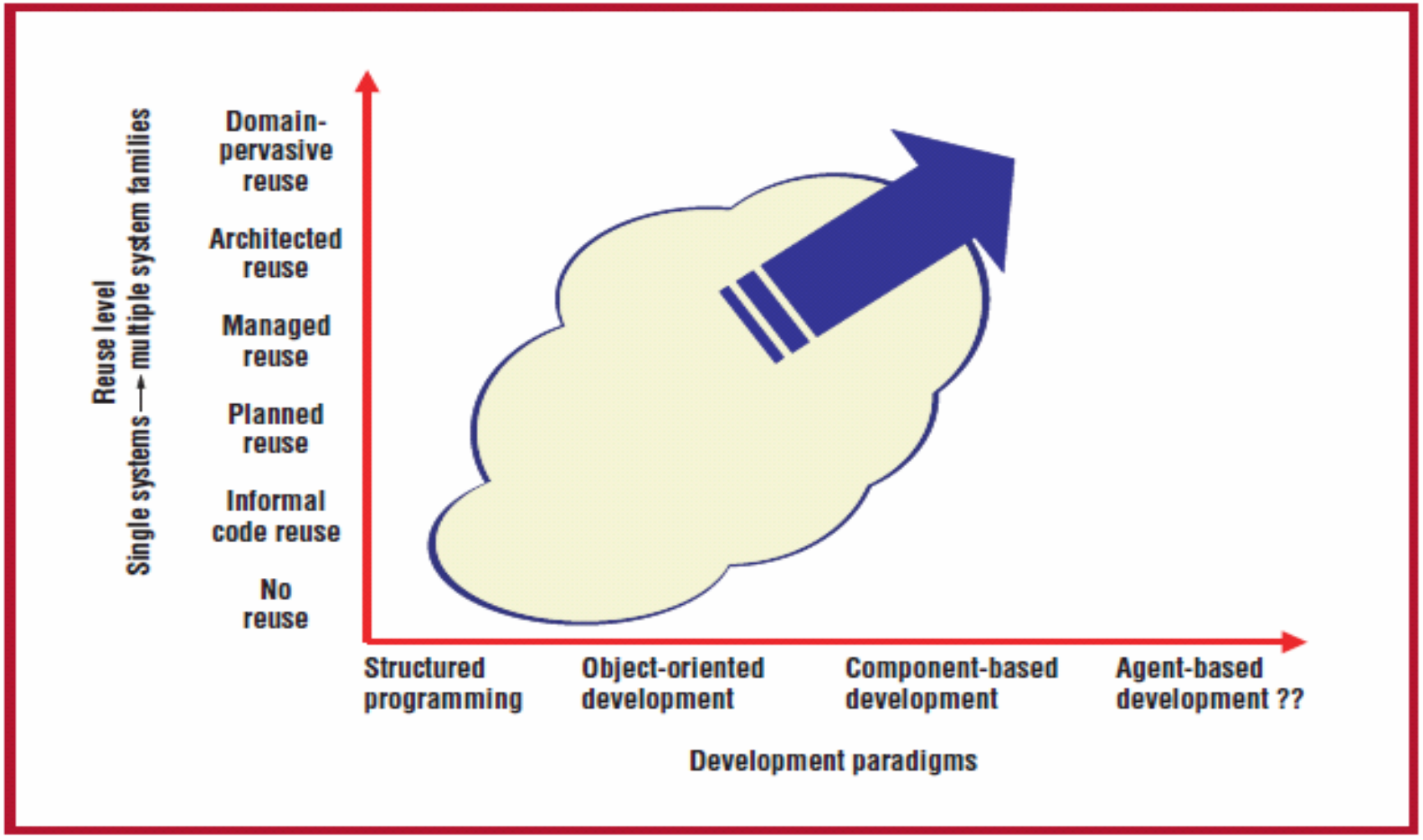
ソフトウェアファクトリ形成に必要な条件

- 必要な基盤が事前に形成されていること
- 基盤上で、ソリューションが自己組織化可能であること

CCMに基づいたコンポーネント（COM+context、またはEJBcontainerで実装）



ソフトウェアエンジアリングの動向



ソフトウェアエンジニアリング動向の変化

- 一般化 (generalization)追求から、領域特定化 (domain-specification)、資産化、資産利用エンジアリングへの転換

- 「ドメイン」がもつ、共通のフィーチャを重視
  - フィーチャ： 顧客の個々の事業価値項目と結びついた問題解決のための概念 →製品系列内では共有 (共通したフィーチャをもった製品族を製品系列とする)
  - デマンド： 需要、メタ・ニーズ

  「ソフトウェア要求」は、「問題」に対するソリューションを、高い抽象度における、なんらかのビューポイントから表現した記述である。

- ドメインごとの知識資産、プロダクト資産、プロセス資産、サービス資産を基底とした開発・保守への移行

- 資産を使って要求に即応できるような、資産組み込み法の追求、およびそれを利用したファクトリ方式の確立

概念、ドメイン、およびフィーチャについては、別スライドで定義する。

ドメインとは

- ドメインとは、概念 (concepts) および知識 (knowledge) の、ある特定された領域 (area)のこと
- ドメインを定める目的：
  - ステークホルダを特定し、その満足度を最大にするため
  - 必要な資源を特定し、生産性およびプロダクト品質を最大にするため
- ドメインに含まれるもの：
  - この領域の実践分担者が理解できる概念
  - この領域に存在する問題を解決するために必要な知識

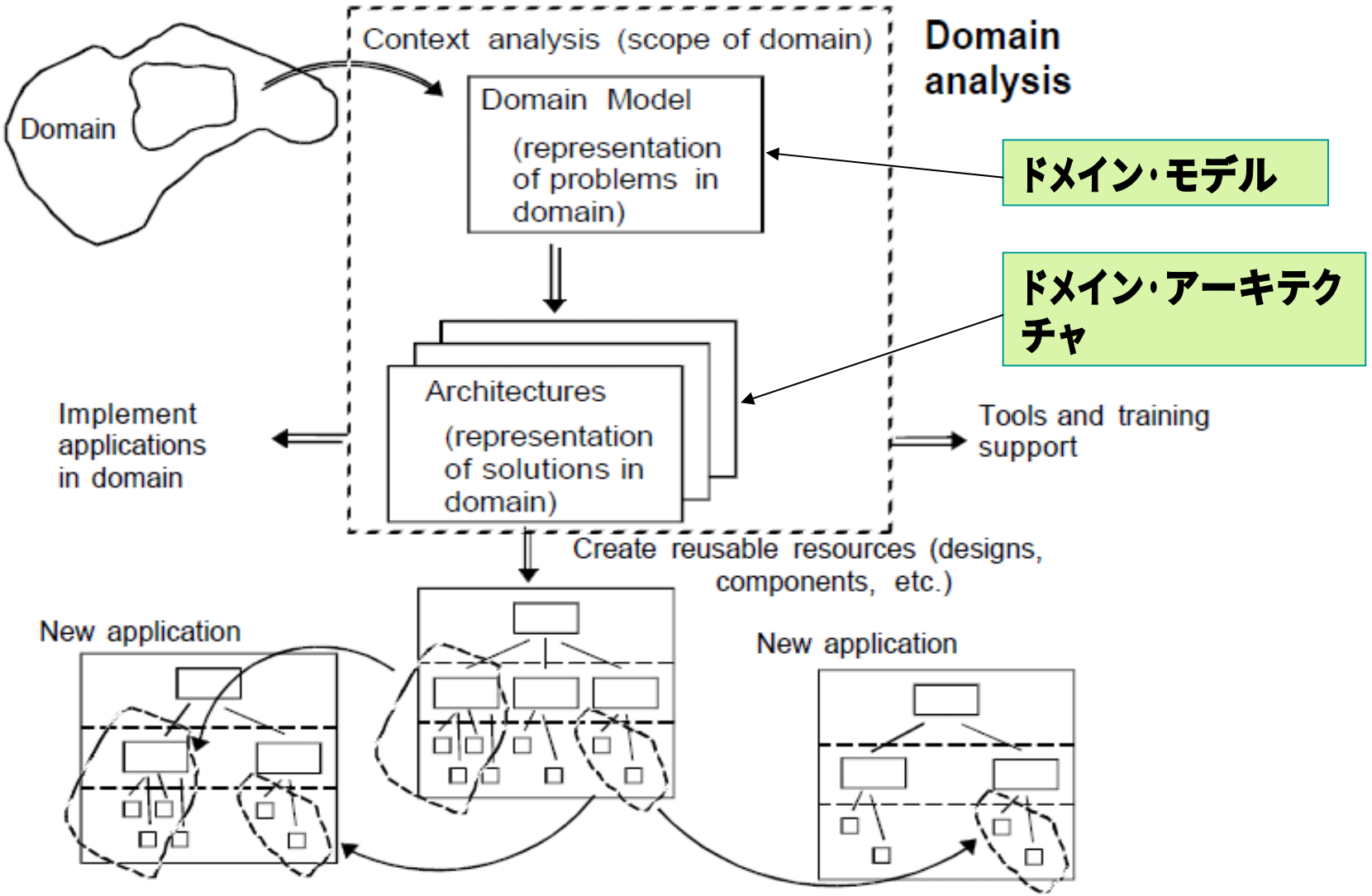
概念とは(concepts)

- 概念とは [清水83]：
  - カテゴリ内の事物の共通特性を抽象したもの
  - 事物のカテゴリ名を指すもの
- 概念をモデルの視点から表現するための理論
  - 定義的特性理論 (Propositional representation)
    - カテゴリは、カテゴリごとに個別に必要、かつ十分な定義的記述集合によって表象され、これによってその概念が示すカテゴリへの所属が、明確に決定できる。
  - 特徴的特性理論 [Smith81]
    - カテゴリは、特性の分布に由来する不均質な内的構造をもち、境界は不鮮明であり、family resemblance 、または確率的モデルによって識別する。
    - 特性の型に、Features (定性的) 、およびDimensions (定量的) があるとする。
- 理論ベースの概念理論 (Exemplar modeling)
  - 省略

Domain Engineering

- Establish he boundaries of the target domain and its relationships to other domains.
- Develop a domain model that captures essential features, capabilities, concepts, and functions in an enabled representation format, of which commonalities and differentials should be made clear.
- Develop a domain architecture from defined viewpoints in an enabled representation format.
- Specify assets belonging to the domain, where assets include documented requirements, designs, software codes, test cases, guidelines and manuals.

Domain Model vs. Domain Architectures



Domain Analysis and Design

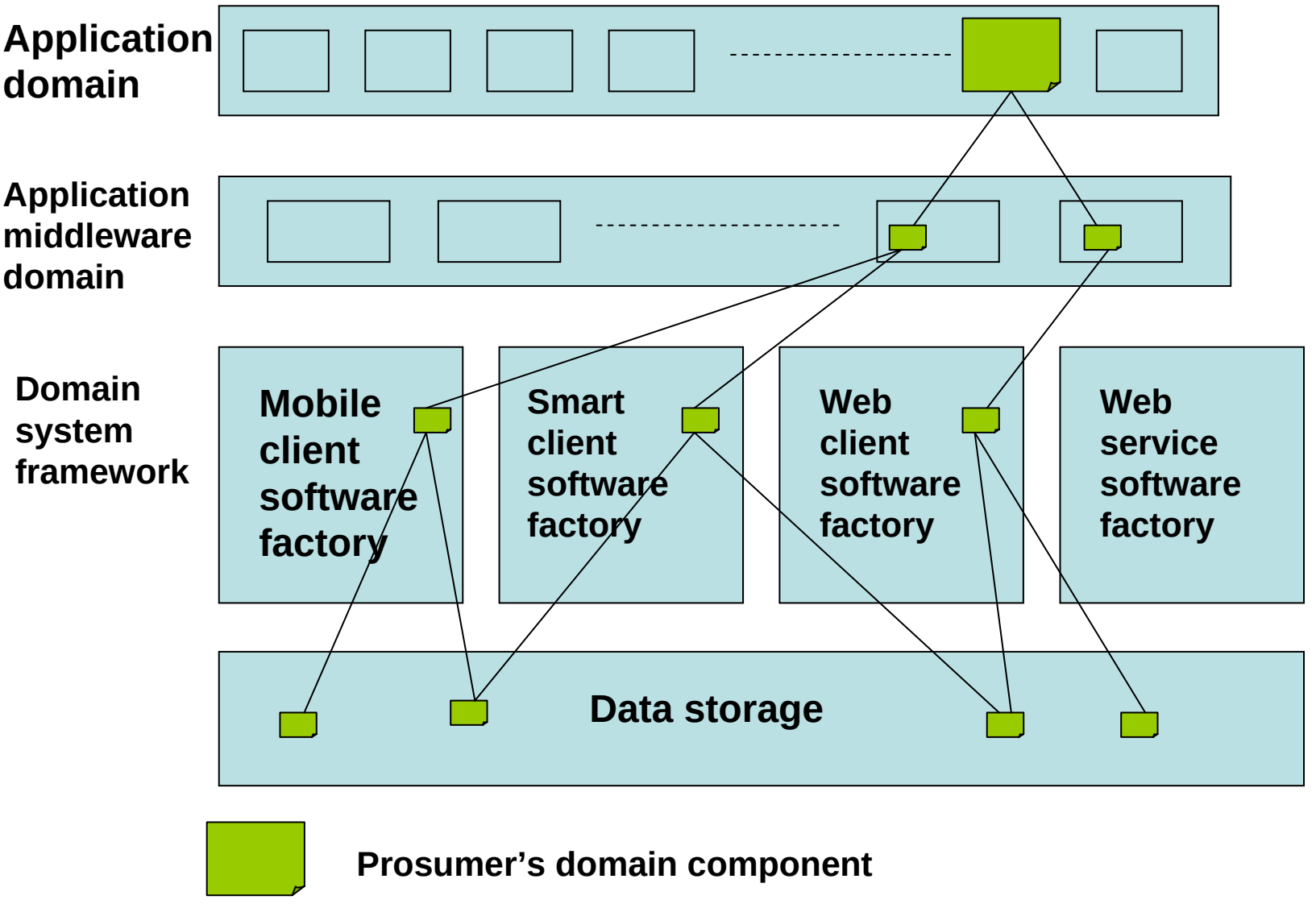
- **Context Analysis (The ISO/IEC JTC1 FCD 12207 includes the context analysis in the domain analysis)**
  - **Define the boundary of the domain.**
  - **Identify the problem (not solutions), such as current and anticipated needs.**
    - **e.g. This year, we need to increase the earning (sales) 1.3 times as much as that of the last year by introducing CRM.**
- **Domain Analysis**
  - **Describe the domain model – the representation of the problem -- features, capabilities, concepts, and functions.**
  - **The domain model must be validated.**
- **Domain Design**
  - **Describe the domain architecture – the representation of a family of systems that consist of software components, their external properties, and their relationships with one another, including their commonalities and variabilities.**
  - **The domain model should be mapped to the domain architecture.**

これから必要となるドメインに対応するソフトウェアファクトリ

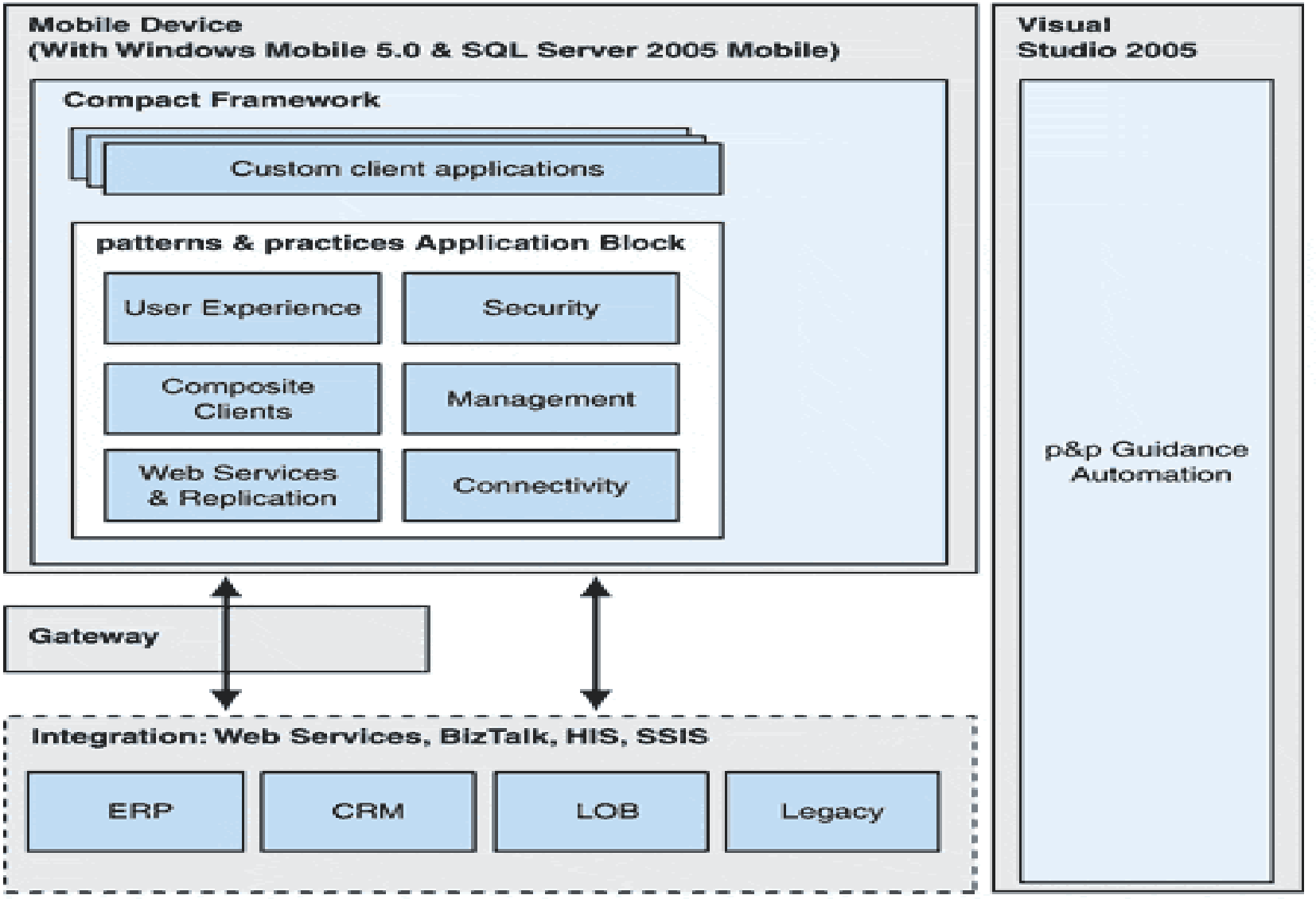
- **現在：**
  - **Web client software factory**
  - **Smart client software factor**
  - **Mobile client software factory**
  - **Web service software factory**

- **これから：**
  - **AUTOSAR (JASPER) software factory**
  - **RFID client software factory**
  - **NGN-grid client software factory**

プロシューマ中心型ソフトウェアファクトリが求めるシステムフレームワーク



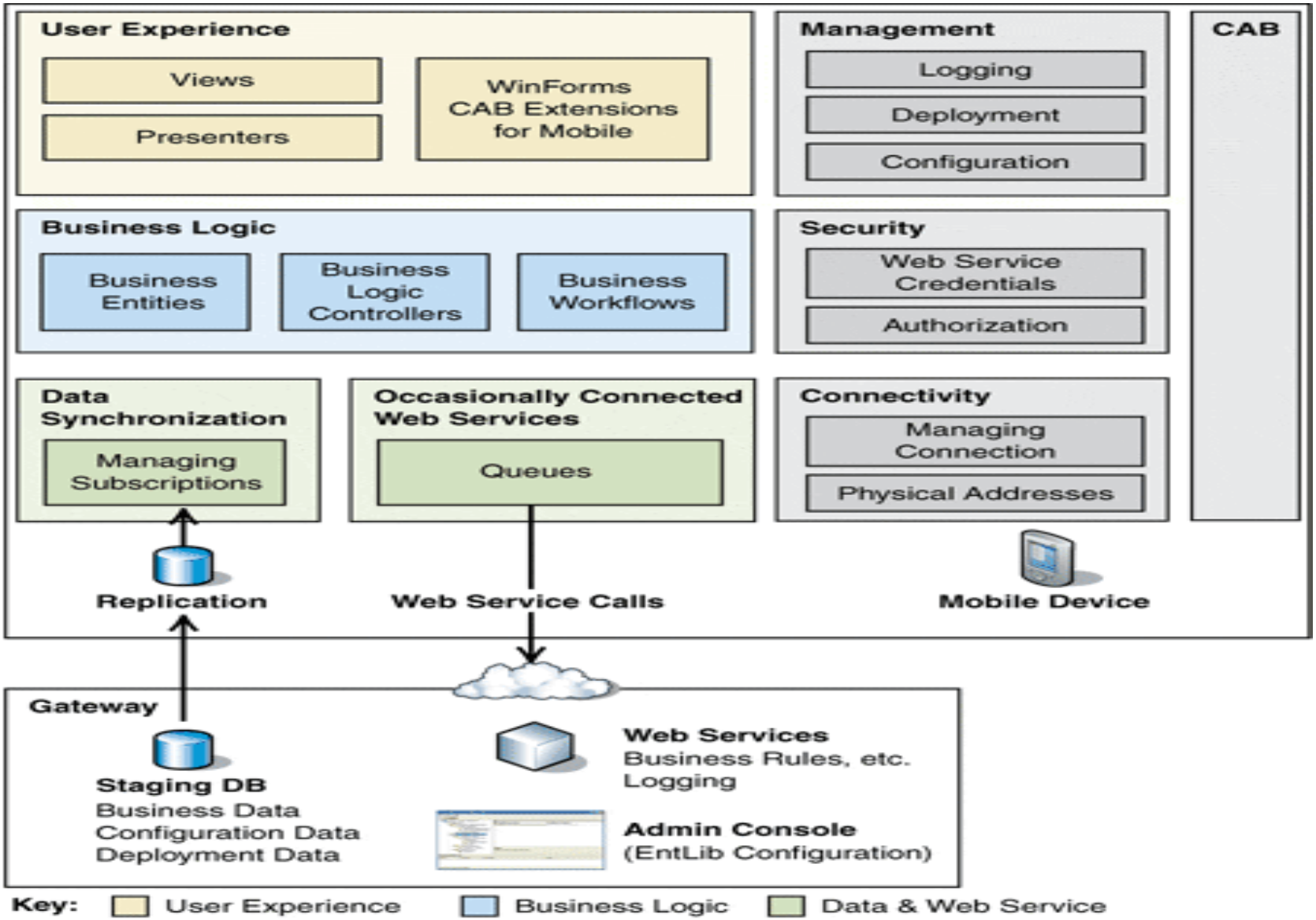
Mobile Client System Framework



http://msdn2.microsoft.com/en-us/library/aa480471.aspx

ドメイン・コンポーネント

Patterns and Practices Application Block



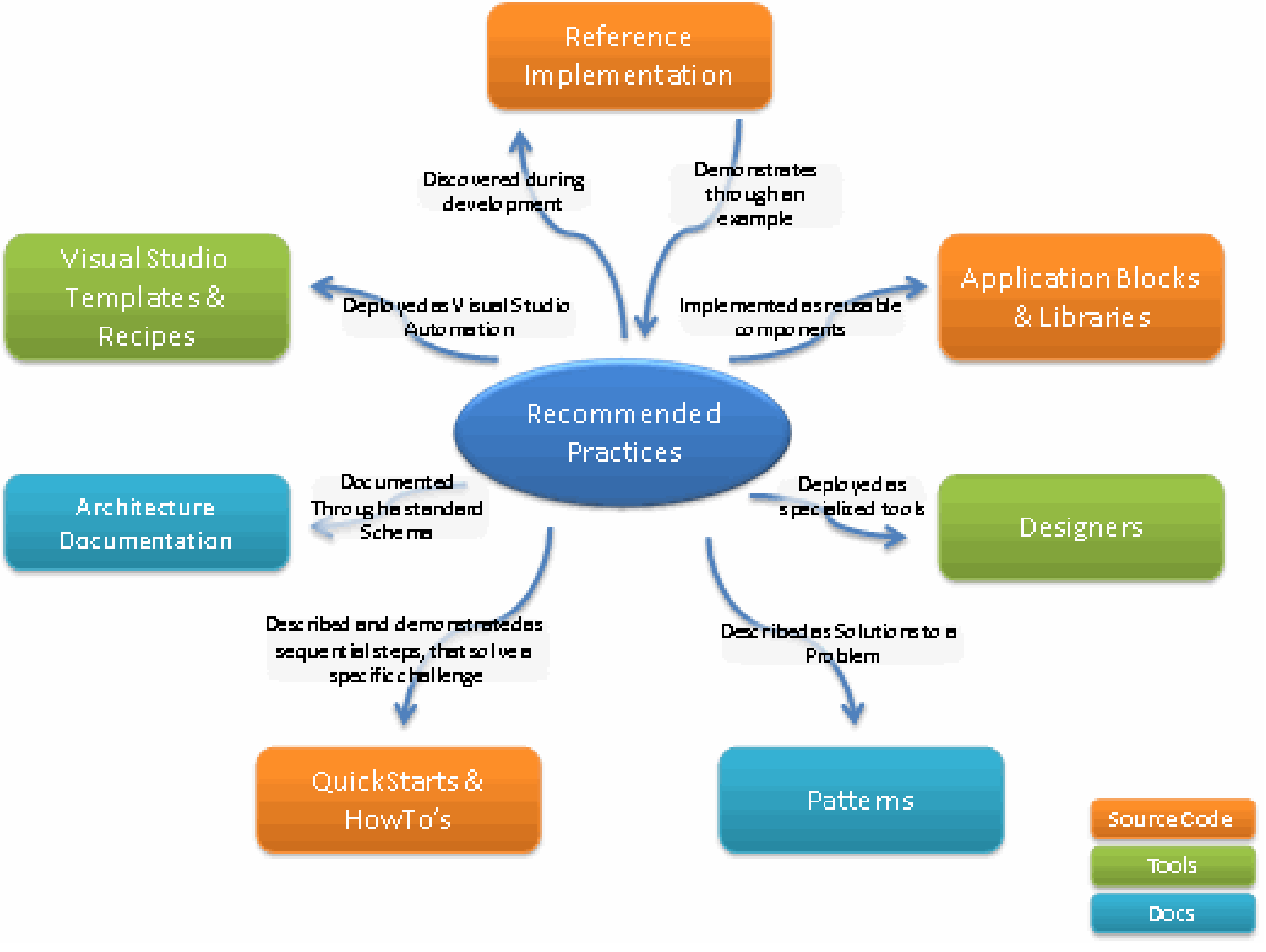
http://msdn2.microsoft.com/en-us/library/aa480471.aspx

Microsoft's Software Factoryが提供するもの (1)

- **Application blocks and libraries.** The Composite Web Application Block, Page Flow Application Block, ObjectContainerDataSourceControl control are included in the software factory. The software factory also uses Enterprise Library application blocks for security, exception management, logging, and data access.
- **Recipes.** The software factory includes the Add View (with presenter) recipe and Add Page Flow recipe. Recipes automate procedures in How-to topics, either entirely or selected steps. They help developers complete routine tasks with minimal input.
- **Templates.** The software factory includes the Solution template, Business Module template, Foundational Module template, and Page Flow template. Templates are prefabricated application elements with placeholders for concrete arguments. They can be used for many purposes, including creating initial solution structures to creating individual solution artifacts, such as project items.
- **Designers.** The software factory includes the page flow designer. Designers provide information that architects and developers can use to model applications at a higher level of abstraction. Designers can also generate code that is compatible with the architecture baseline.
- **Reference implementation.** The software factory includes the Global Bank Corporate e-Banking reference implementation. A reference implementation provides an example of a realistic, finished product that the software factory helps developers build.
- **Architecture guidance and patterns.** The software factory includes architecture guidance and patterns that help explain application design choices and the motivation for those choices.
- **How-to topics.** The software factory includes How-to topics; these are documented step-by-step instructions that describe how to implement recommended practices in a specific domain.

- http://msdn2.microsoft.com/en-us/library/bb264518.aspx

Microsoft's Software Factoryが提供するもの (2)



- http://msdn2.microsoft.com/en-us/library/bb264518.aspx

ドメイン･システムフレームワークはソフトウェアファクトリで形成

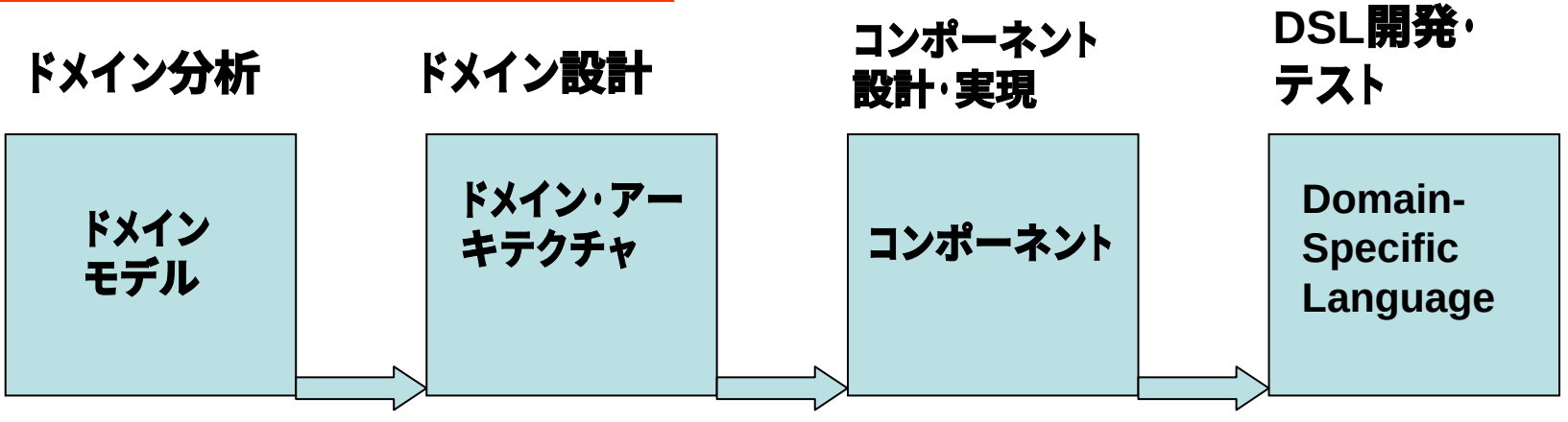
Mobile Client Software Factoryの場合には：

- **文書を読む：** Proven practices of mobile client development
- **ソフトウェア搭載：** Install the Mobile Client Software Factory.
- **端末画面：** Read the documentation for common patterns.
- **端末画面：** Review the Reference Implementation.
- **端末画面：** Review the Getting Started section in the documentation.
- **端末画面：** Click Start on the taskbar, click All Programs, click Microsoft patterns & practices, click Mobile Client Software Factory – July 2006, and then click Help.
- **端末画面：** Use the templates and recipes in the guidance package.
- **端末画面：** Read and follow the procedures in the Quickstart examples.
- **手作業：** Create a mobile client application by following the procedures in the Quickstart examples.

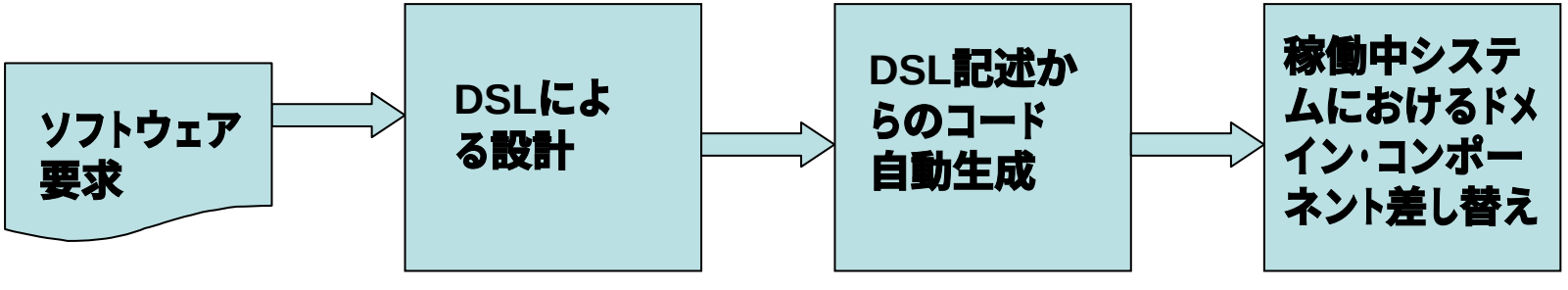
ドメイン･コンポーネントの自動生成と実装

プロシューマが、毎日変更したくなるドメイン･コンポーネントは、下記のようにして随時差し替え可能にする。例：　量販店における物品価格意思決定など

イノベーション･ステージ　(準備ステージ)



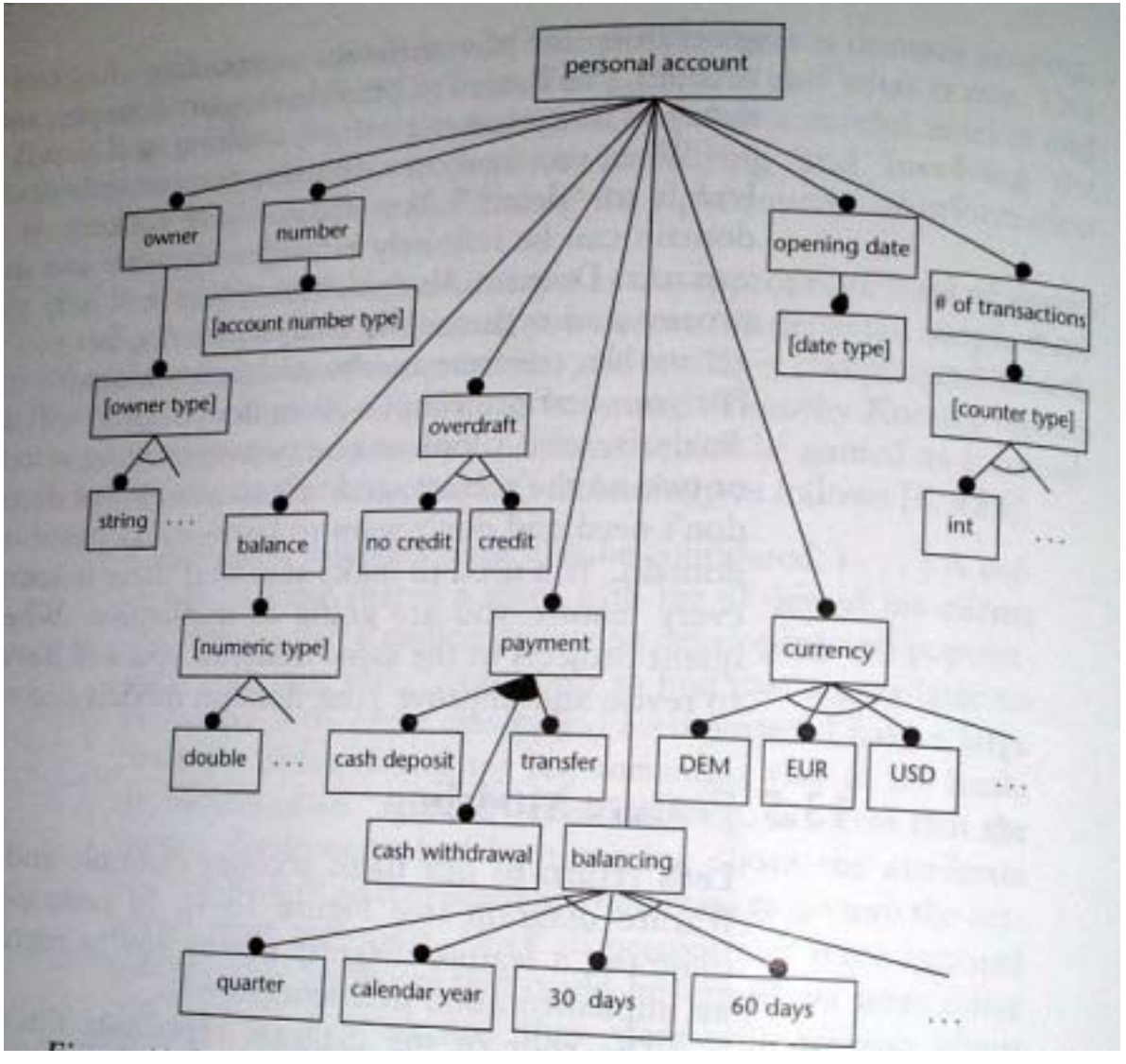
コモディティ･ステージ　(差し替え実行ステージ)



ドメイン・コンポーネントのエンジアリング例 (1)

ドメイン分析:

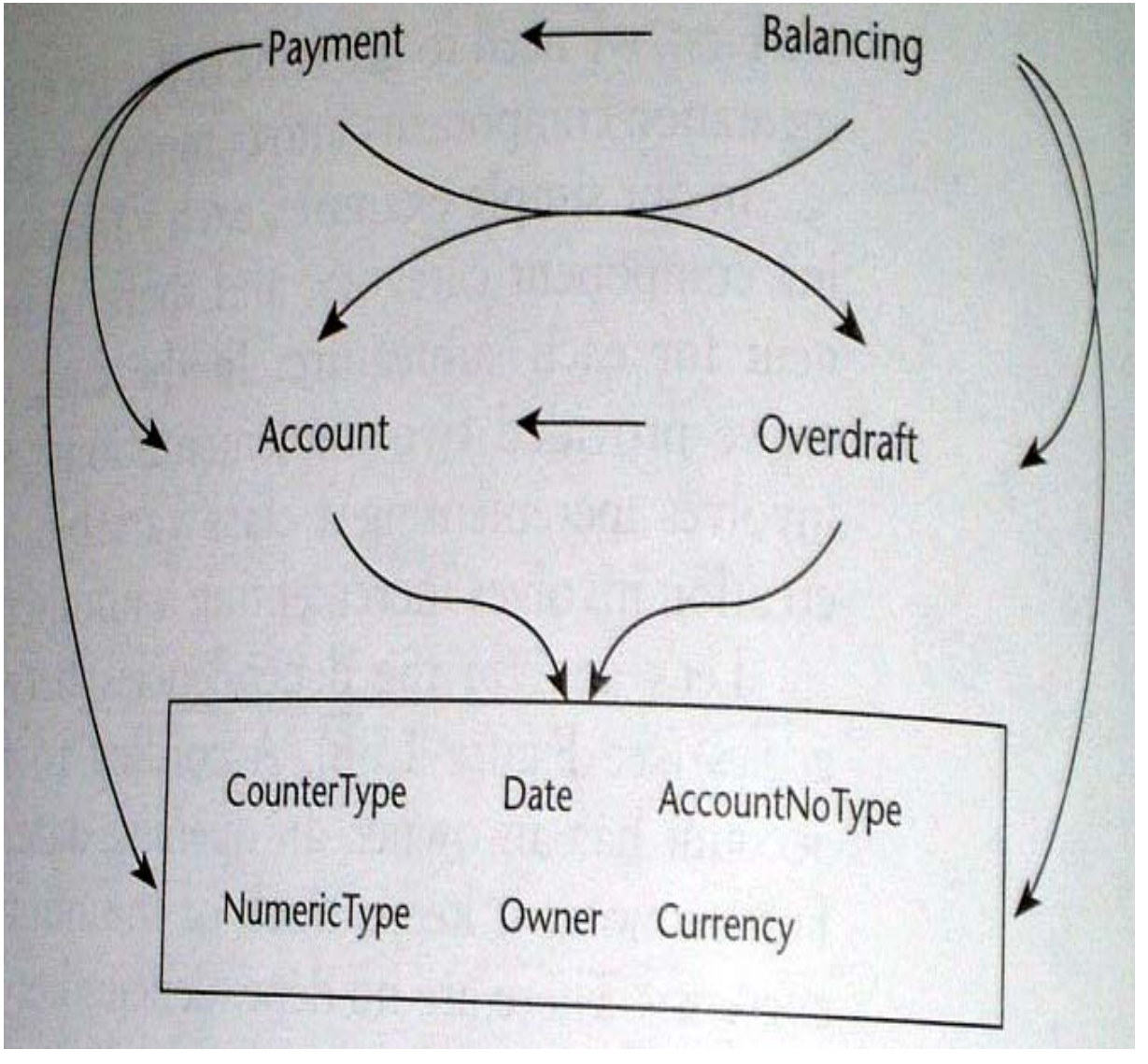
まず、「個人口座」というドメインに存在する概念(concepts)の特徴(features) を図で表す。



ドメイン・コンポーネントのエンジアリング例 (2)

ドメイン設計：

つぎに、特徴の構造、特徴間に存在する関係を基にして、ドメイン・アーキテクチャを設計する。

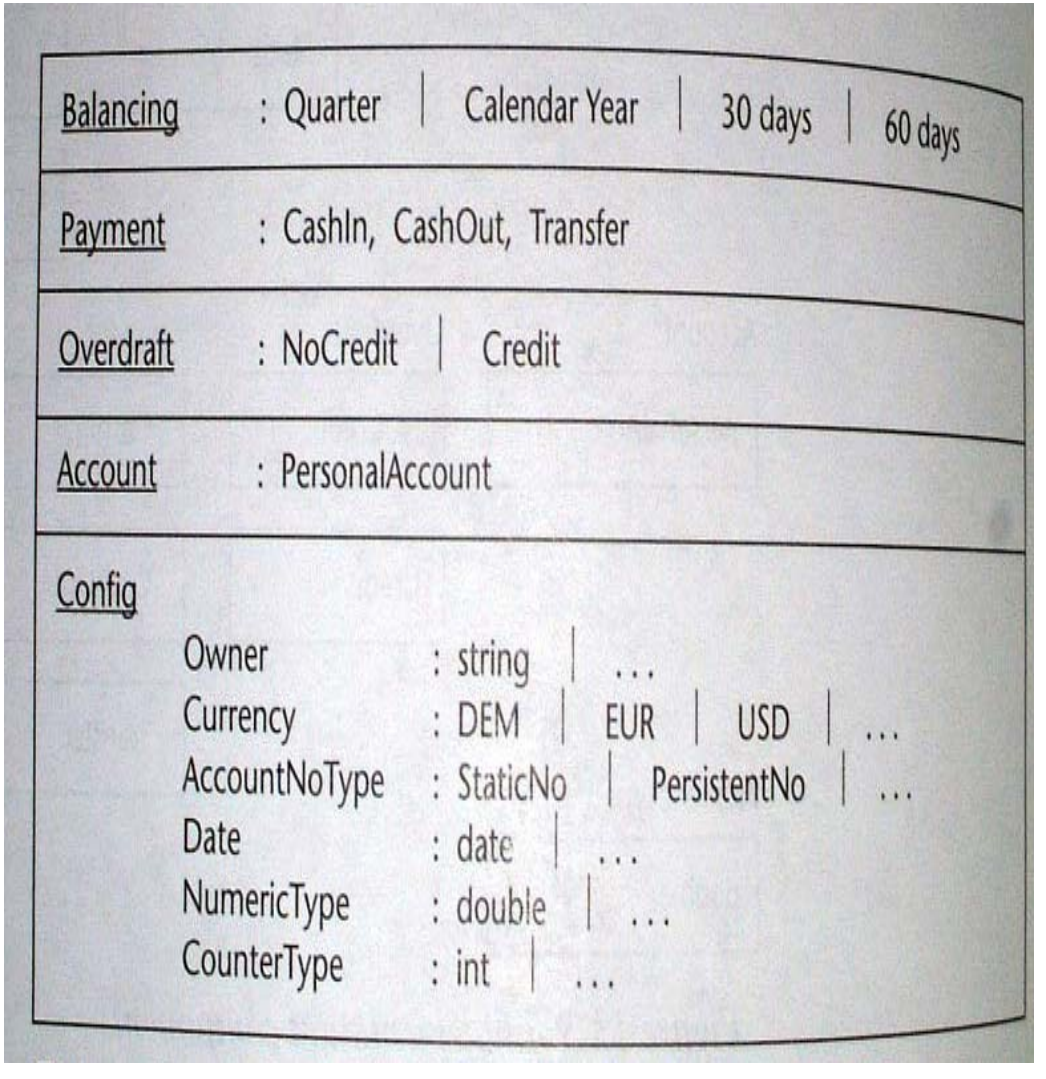


ドメイン・コンポーネントのエンジアリング例 (3)

つぎに、特徴の品質 (汎性と特化性、独立性、基礎性) を基にして、特徴仕様を作成する。

右の例は、GenVocaによる使用記述例。

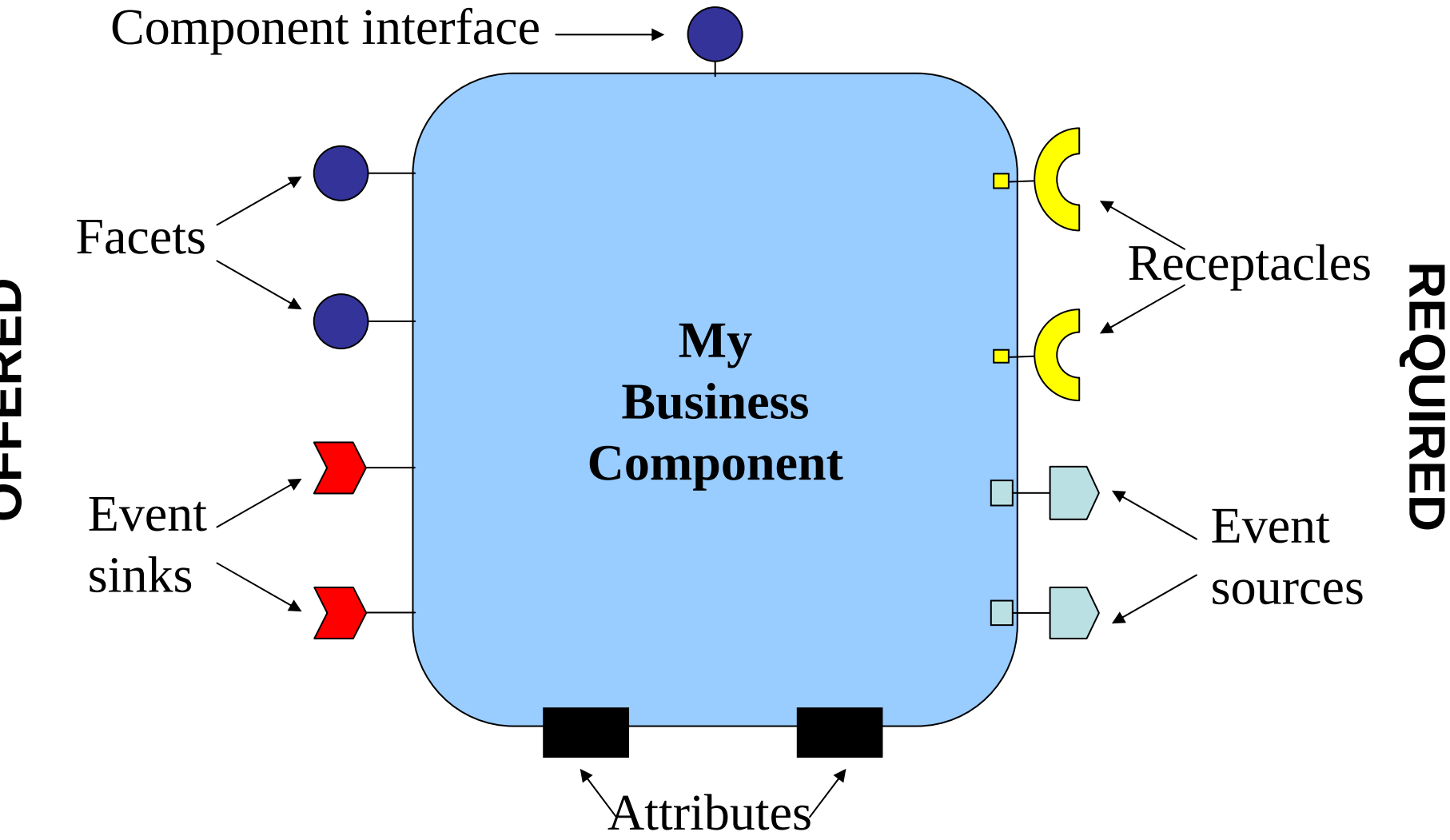
つぎに、特徴を構成するソリューション・コンポーネントを抽出し、各ソリューション・コンポーネントを抽象データ型 (クラス) として実現する。



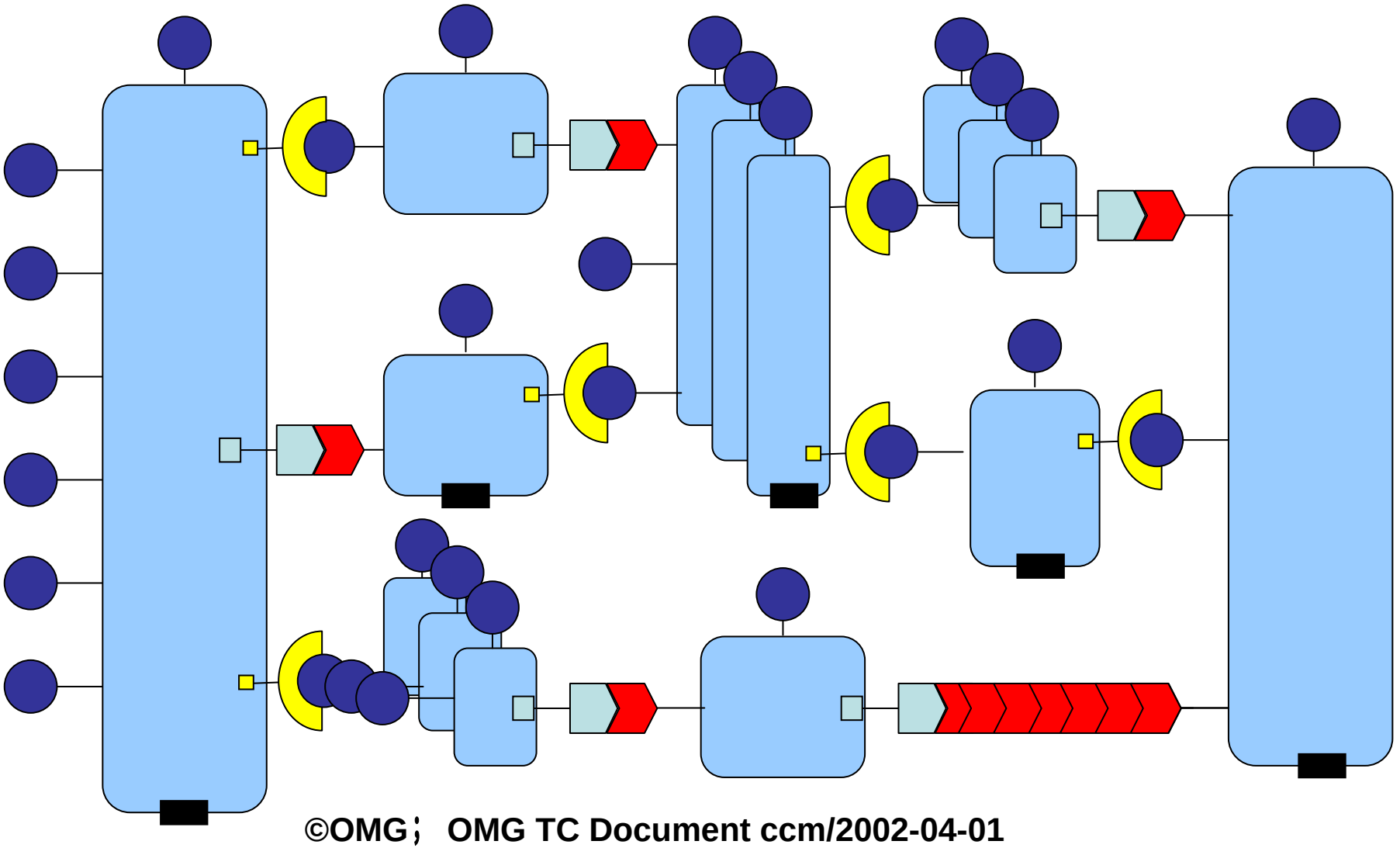
ドメイン・コンポーネントのエンジアリング例 (4)

- つぎに、「個人口座」にとって必要と考えられる典型的な型（タイプ）を選定（たとえば、下記）し、特徴仕様、およびソリューション・コンポーネントを作成する。
  - 総合口座
    - 普通預金
    - 定期預金
    - 特典つき積立預金
    - 国際等公社債
  - 決済用普通預金
  - その他、将来予想される個人口座の型
- これら「個人口座」型に対する設計を行い、そのなかから固定できる共通要素と、固定できない可変要素を分析し、新しい要求によって可変要素をカスタマイズし、要求を実現できるようなDSL (domain-specific language)を開発する。

ドメイン･コンポーネントの実装は、CORBA Componentsによる



Building CCM Applications = Assembling CORBA Component Instances



参考文献（1）

- [Aaen97] Aaen, I, et al., The Software Factory: Contributions and Illusions, in Proceedings of the Twentieth Information Systems Research Seminar in Scandinavia, Oslo (1997)
- [Baldwin00] Baldwin, C.Y. and K.B. Clark, Design Rules, The MIT Press (2000)
- [Basili89] Basili, V. R., The Experience Factory: Packaging Software Experience,
- Proceedings of the 14th Annual Software Engineering Workshop, NASA Goddard Space Flight Center, Greenbelt MD. (1989)
- [Clements02] Clements, P., et al., Software Product Lines, Addison-Wesley (2002)
- [Cusumano91] Cusumano, M.A., Japan's Software Factories, Oxford University Press (1991)
- [Eppinger91] Eppinger, S.D., Model-based Approaches to Managing Concurrent Engineering, Journal of Engineering Design 2(4) (1991)
- [Greefield04] Greenfield, J. et al, Software Factories, Wiley Publishing (2004)
- [INCOSE04] INCOSE/Systems Engineering Handbook, International Council on Systems Engineering, INCOSE-TP-2003-016-02, Version 2a, 1 June 2004

参考文献(2)

- [Matsumoto81] Matsumoto, Y., SWB System: A Software Factory, in H. Hunke (Ed.): Software-Engineering Environments, North-Holland, Amsterdam (1981)
- [Matsumoto87] Matsumoto, Y., A software factory: An overall approach to software production, in "Software Reusability" ed. by P. Freeman, pp.155- 178, IEEE Computer Society (March 1987)
- [Matsumoto92a] Matsumoto, Y., Toshiba Software Factory, in "Modern Software Engineering, P.A. Ng and R.T. Yeh (eds.), pp.479-501, Van Nostrand Reinhold (1990)
- [Matsumoto92b] Matsumoto, Y., Japanese Software Factory, Advances in Software Science and Technology, Vol.4, Japan Society for Software Science and Technology, pp.21-42, Iwanami Shoten, Tokyo (1992)
- [Matsumoto94] Matsumoto, Y., Japanese Software Factory, in “The Encyclopedia of Software Engineering”, J.J.Marciniak (ed.), 1st Edition, pp.593-605, John Wiley & Sons, New York (1994)
- [McIloy69] McIlroy, M. D., Mass-Produced Software Components, in “Software Engineering Reports” on a Conference Sponsored by NATO Science Committee, Brussels (1969)
- [Parnas83] Parnas, D.L., A Technique for Software Module Specification with Examples, Comm. ACM, 26(1) (1983)
- [清水83] 清水御代明、 概念的思考、坂本昂(編)、現代基礎心理学7、pp.71-105 (1983)
- [Smith81] Smith, E.E., and D.L. Medin, Categories and Concepts, Harvard University Press (1981)