レイヤリング戦略

Peter Eeles

Rational Software ホワイト・ペーパー

TP 199, 08/01

Rational®
the software development company

目次

要約

ソフトウェア・システムを分解するための技術は数多くあります。ここでは、その 1 つの例であるレイヤリングについて説明します。これらの技術では、2 つの主要な問題を扱っており、ほとんどのシステムは複雑すぎてその全体を理解できないことと、システムに対する別の視点が必要であることです。

レイヤリングはさまざまなソフトウェア・システムで採用され、多くのテキストで取り上げられています。 Rational Unified Process (RUP) でも、これを取り上げています。ただし、レイヤリングは誤解されたり、正しく適用されていない場合があります。このホワイト・ペーパーでは、レイヤリングが意味するものを明らかにし、さまざまなレイヤリング戦略を適用する影響について説明します。

「レイヤリング」とは

まず、「レイヤリング」の意味を定義します。レイヤーという用語は、一般的に Layers パターンとして知られるアーキテクチャー・パターンの応用を表します。このパターンは、多くのテキスト (参考資料 [Buschmann]、[Herzum]、[PloP2]) で説明され、RUP でも説明されています。パターンは、特定のコンテキストに存在する一般的な問題に対する解決策を表します。Layers パターンの概要を、表 1 に示します。

表 1:「Layers」パターンの概要

	Layers パターン
コンテキスト	分解を必要とするシステム
問題	システムが複雑すぎて、その全体像を理解できない。 システムの保守が難しい。 最小限の安定要素が分離されていない。 最も再利用可能な要素を識別するのが難しい。 システムが異なるチームによって (おそらく、異なるスキルで) 作成されている。
解決策	システムの構造にレイヤーを導入する。

レイヤリングの最も有名な例に、国際標準化機構（ISO）によって定義されている OSI 7 レイヤー・モデルがあります。このモデルは、図 1 に示すように、ネットワーク・プロトコルのセットを定義しています。各レイヤーは通信の特定の側面に注目し、下位レイヤーの機能の上に積み重なっています。OSI 7 レイヤー・モデルでは、責務に基づくレイヤリング戦略が使用されていて、各レイヤーには特定の責務があります。この戦略については、後で詳しく説明します。

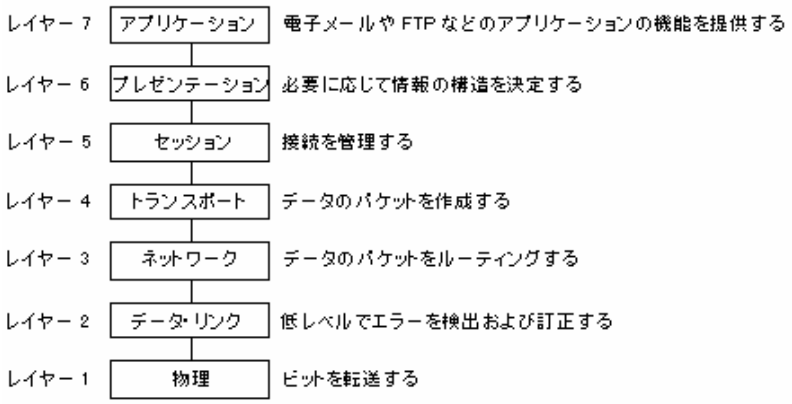

図 1:OSI 7 レイヤー・モデル (責務に基づくレイヤリング)

図 2 は、責務に基づくレイヤリングの別の例を示しています。

- 「プレゼンテーション論理」レイヤーには、ユーザー・インターフェースの要素など、ユーザーに対して何らかの表現を提供することに責任を持つ要素が含まれます。
- 「ビジネス論理」レイヤーには、ビジネス・プロセスの実行や、ビジネス・ルールの適用に責任を持つ要素が含まれます。
- 「データ・アクセス論理」レイヤーには、リレーショナル・データベースなどの情報ソースへのアクセスを提供することに責任を持つ要素が含まれます。

後で説明するように、レイヤーはさまざまな方法でモデリングできることに注意してください。ここでは、«layer» ステレオタイプの UML パッケージを使用して、レイヤーを明示的に示しています。

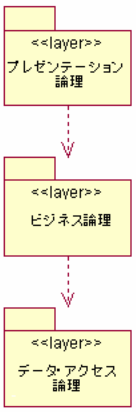

図 2:責務に基づくレイヤリング

この責務に基づくレイヤリングの例で示したレイヤーは、「層」と呼ばれる場合があります。これらは、分散型のシステム開発ではよく知られた概念であり、2 層、3 層、n 層のシステムがあります。

図 2 の重要な面は、示されている依存関係の方向です。これは、レイヤー・システムの特徴である特定のルールを意味していて、特定のレイヤーの要素は、同じレイヤーか下位レイヤーの要素にしかアクセスできません[1]。ここで示した例では、「ビジネス論理」レイヤーの要素は「プレゼンテーション論理」レイヤーの要素にアクセスできません。また、「データ・アクセス論理」レイヤーの要素は「ビジネス論理」レイヤーの要素にアクセスできません。この構造は、DAG (directed acyclic graph) と呼ばれます。directed は依存関係が一方向であることを表し、acyclic は依存関係の経路が循環していないことを表します。

また、レイヤリング戦略を定義する場合は、要素が適切なレイヤーに正しく配置されるように、各レイヤーの意味を正確に示すことが重要です。要素を適切なレイヤーに正しく配置しないと、第一に戦略を適用する価値が失われます。各レイヤリング戦略の説明において、各レイヤーの意味に関する一般的なガイダンスが説明されています。

レイヤーのモデリング

さまざまなレイヤリング戦略を調査すると、特定のモデル (と特定の UML 要素) を使用して、各戦略で情報を交換することが適切であることがわかります。モデルは、特定の視点から見た、システムの完全な説明を表しています。図 3 は、4 つのモデルの例を示しています。これらのモデルは、対象とするシステムの異なる視点を表しています。

- ユースケース・モデル:システムの要求を把握します。
- 分析モデル:システムの要求分析を把握します。
- 設計モデル:システム設計を把握します。
- 実装モデル:システムの実装を把握します。

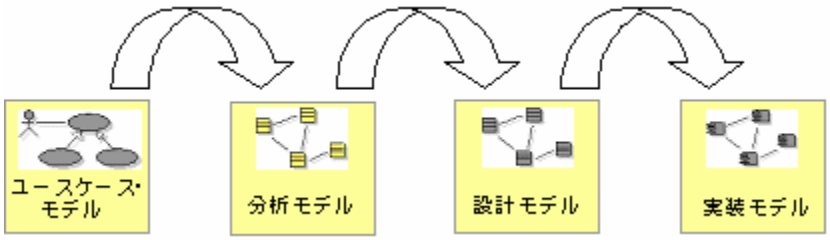

図 3:段階的な進展を表す 4 つのモデル

次のようなモデルもあります。

- *配置モデル:*システムの配置に関する側面を把握します。
- *データ・モデル:*システムの永続的な側面を把握します。

レイヤリング戦略

レイヤリングは、さまざまな特性に基づいて計画できます。ここでは、次の特性に基づくレイヤリングについて説明します。

- 責務
- 再利用

[1] イベント通知によって、送信元のレイヤーの要素から上位レイヤーの要素にメッセージが生成される場合がありますが、この方向には明示的な依存関係はありません。

各戦略の表現は、それぞれを詳しく説明するときに考慮します。

責務に基づくレイヤリング

最も一般的に使用されるレイヤリング戦略は、おそらく責務に基づく戦略です。この戦略は、システムの開発と保守を向上させることができます。これは、さまざまなシステムの責務が、お互いに独立しているためです。前の例 (図 2) に示したように、システムは次の責務に従ってレイヤー化できます。

- プレゼンテーション論理
- ビジネス論理
- データ・アクセス論理

これらの責務はそれぞれ、図 4 に示すようなレイヤーとして表すことができます。図では、各レイヤーのコンテンツのサンプルを示しています。ここでは、受注処理システムの 3 つの概念である顧客、注文、製品を考えています。たとえば、顧客の概念は、次のようになります。

- *CustomerView クラス*:顧客に関連付けられるプレゼンテーション論理に責任を持ちます。たとえば、ユーザー・インターフェースでの顧客とのやり取りです。
- *Customer クラス*:顧客に関連付けられるビジネス論理に対して責任を持ちます。たとえば、顧客の詳細の確認などです。
- *CustomerData クラス*:顧客に関連付けられるデータ・アクセス論理に責任を持ちます。たとえば、顧客の状態を永続にします。

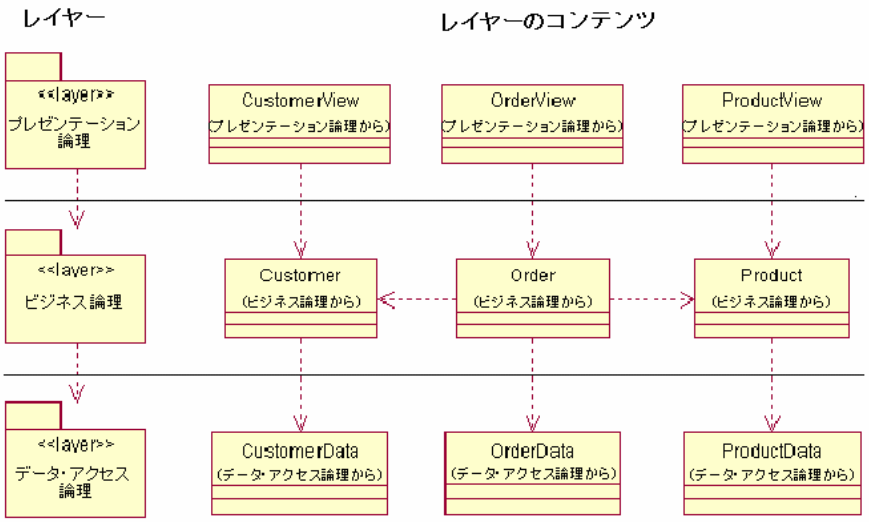

図 4:責務に基づくレイヤリングのレイヤーとコンテンツ

続いて、このレイヤリング戦略に関する「概念」について考察します。

概念 1:レイヤーと層は異なる。

この概念は、一般的な混乱の元になります。実際には、レイヤーが特定の戦略 (たとえば、責務) に基づいていても、層はレイヤーです。表 2 に示すように、層の概念はさまざまな方法で適用できることから、混乱がさらに複雑になります。

表 2:層の定義

適用	レイヤー (層)
2 層	プレゼンテーション論理と ビジネス論理の組み合わせ データ・アクセス論理
3 層	プレゼンテーション論理 ビジネス論理 データ・アクセス論理
n 層	プレゼンテーション論理 ビジネス論理: 分散 データ・アクセス論理

概念 2:レイヤー (層) は物理的な分散を意味する。

別の一般的な誤解として、論理的なレイヤリングは物理的な分散を意味するというものがあります。ここで、3 層のレイヤリングを考えます。さまざまな要素が 1 つのレイヤーに存在しますが、各レイヤー自体にはさまざまな適用方法があります。表 3 では、特定の物理的な分散を特徴付けるための名前 (「シン・クライアント」など) が使用されます。

表 3:3 層レイヤリングの適用

適用	レイヤー	
	クライアント側	サーバー側
単体システム	プレゼンテーション論理 ビジネス論理 データ・アクセス論理	
シン・クライアント	プレゼンテーション論理	ビジネス論理 データ・アクセス論理
ファット・クライアント	プレゼンテーション論理 ビジネス論理	データ・アクセス論理

また、単体システムで複数の物理的な分散戦略を採用することもできます。この場合、特定の要素は「シン・クライアント」を構成し、別の要素は「ファット・クライアント」を構成するように分類されます。一般的に、これらの選択は性能などの機能外要求に基づいて行われます。

責務に基づくレイヤーのモデリング

この戦略の適用は設計モデル、実装モデル、配置モデルに影響を与えます。一般的に、設計モデルの構築には 2 つの手法があります。

1 つ目の手法では、レイヤーに「含まれている」要素が示されます。結果は図 5 のようになります。これは Rational Rose ブラウザーのスクリーンショットで、次のことが示されています。

- プレゼンテーション・クラス (CustomerView、OrderView、ProductView) は、「プレゼンテーション論理」パッケージに存在しています。
- ビジネス論理クラス (Customer、Order、Product) は、「ビジネス論理」パッケージに存在しています。
- データ・アクセス論理クラス (CustomerData、OrderData、ProductData) は、「データ・アクセス論理」パッケージに存在しています。

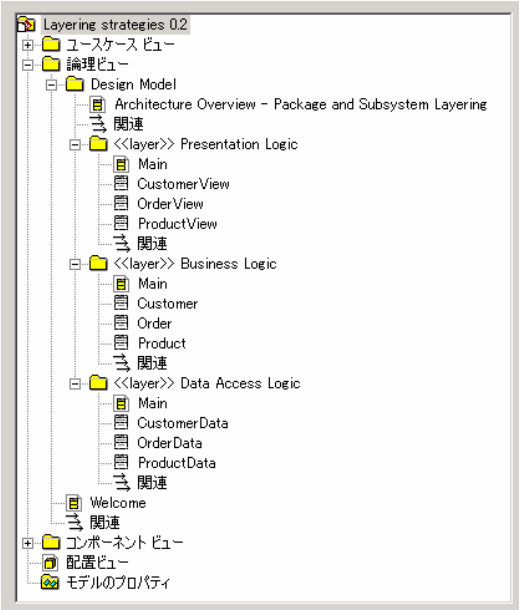

図 5:レイヤーに含まれる要素

2 つ目の手法は、「ビジネス・コンポーネント (この場合は、Customer、Order、Product)」の概念を最上位のメンバーとして組み込む方法です。この場合、主要な要素は、システムでサポートされる領域に関係する概念になります。たとえば、「Customer」の概念には、プレゼンテーション論理、ビジネス論理、データ・アクセス論理の要素が関連付けられます。このビジネス・コンポーネントの概念は、参考資料 [Eeles] と [Herzum] で説明されています。この考え方から、図 6 に示すモデル構造が得られます。この例では、要素名によってレイヤリングが表されています。たとえば、View クラス (CustomerView など) はすべて「プレゼンテーション論理」レイヤーを表し、すべての Data クラス (CustomerData など) は「データ・アクセス論理」レイヤーを表します。何も付かないクラス名 (Customer など) は、「ビジネス論理」レイヤーを表します。

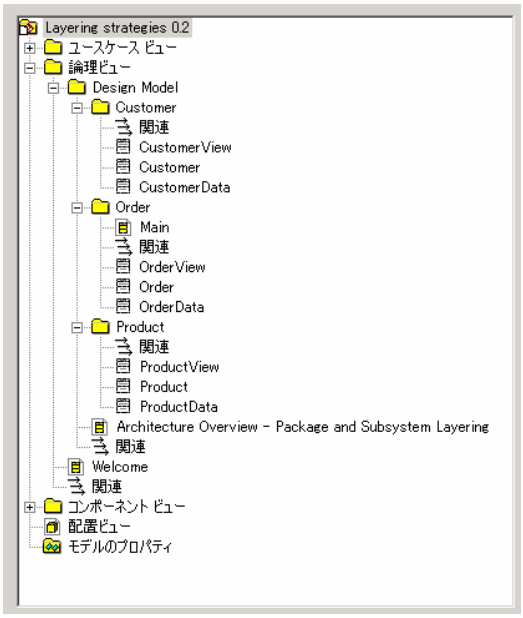

図 6:各ビジネス・コンポーネント・パッケージ内の暗黙的なレイヤリング

レイヤリングは、図 7 に示すように、ビジネス・コンポーネントを表す各パッケージ内で明示的に表すこともできます。この構造化は、ビジネス・コンポーネントの各レイヤーに多数の要素が含まれる場合に適しています。この例では、Customer ビジネス・コンポーネント・パッケージだけが展開されていますが、Order と Product パッケージも同様の構造になります。

図 7:ビジネス・コンポーネント・パッケージ内の明示的なレイヤリング

一般的に、各責務を実装する要素を物理的に分割する必要がある場合、責務に基づくレイヤリング戦略は、設計モデルに加えて実装モデルにも影響を与えます。たとえば、「シン・クライアント」の物理的な配置を示すシステムを考えます。この場合、クライアントでの実行をサポートするために必要な実装単位と、サーバーでの実行をサポートするために必要な実装単位を識別すると有効です。この例では、「プレゼンテーション論理」レイヤーの要素はクライアントに配置されるアプ

リケーションに存在し、「ビジネス論理」レイヤーと「データ論理」レイヤーのすべての要素は、サーバーに配置される別のアプリケーションに存在します。

このシナリオは、図 8 に示す実装モデルを表しています。この図は Rational Rose のブラウザー画像と、クライアントに配置されるアプリケーションの要素を表すコンポーネント図です。この例では、設計モデルのクラスと実装モデルの UML コンポーネント間の対応が 1 対 1 になっています。一般的には、この対応は使用される実装の技術によって変わります。

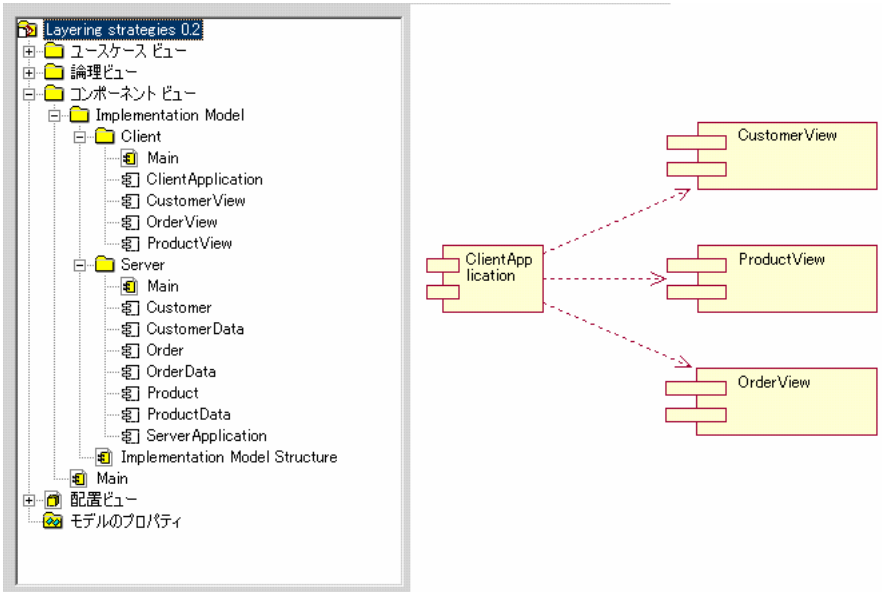

図 8:実装モデル内の暗黙的なレイヤリング

同様に、責務の物理的な分散を記述する必要がある場合、責務に基づくレイヤリング戦略は配置モデルにも影響を与える場合があります。上の例を使用すると、図 9 では 6 つのノードが定義されていることを確認できます。3 つの *Client ノード*には、それぞれ ClientApplication プロセスがあります。FrontEndServer ノードには、クライアントの要求を 1 つまたは 2 つの Server ノードに配分する責任を持つ LoadBalancer プロセスがあります。各 *Server ノード*には、ServerApplication プロセスがあります。

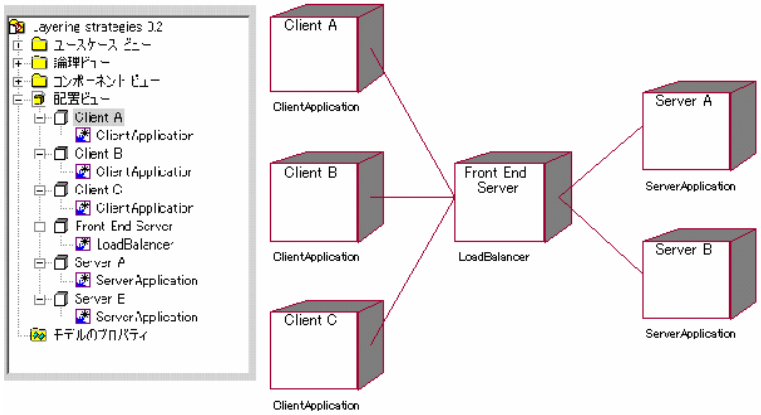

図 9:責務の物理的な分散を表す配置モデル

再利用に基づくモデリング

一般的に使用される別のレイヤリングとして、再利用に基づく方法があります。この戦略は、特に、組織全体でコンポーネントを再利用するための明確な目標を持っている組織に関係があります。このレイヤリング戦略を使用する効果は、コンポーネントの再利用性が明確に視覚化されることです。コンポーネントは、再利用のレベルに従って明示的にグループ化されます。図 10 は、参考資料 [Jacobson] で説明されている戦略を利用した例です。Base、Business-Specific、Application-Specific の 3 つのレイヤーがあります。

- *Base レイヤー*には、組織全体に適用される要素である計算などが含まれます。これらの要素は、広い範囲で再利用されます。
- *Business-Specific レイヤー*には、特定の組織に適用される、アプリケーションに依存しない要素である住所録などが含まれます。これらの要素は、同じ組織のアプリケーションで再利用されます。
- *Application-Specific レイヤー*には、特定のアプリケーションまたはプロジェクトに適用される要素である整理記録ツールなどが含まれます。これらの要素は、ほとんど再利用できません。

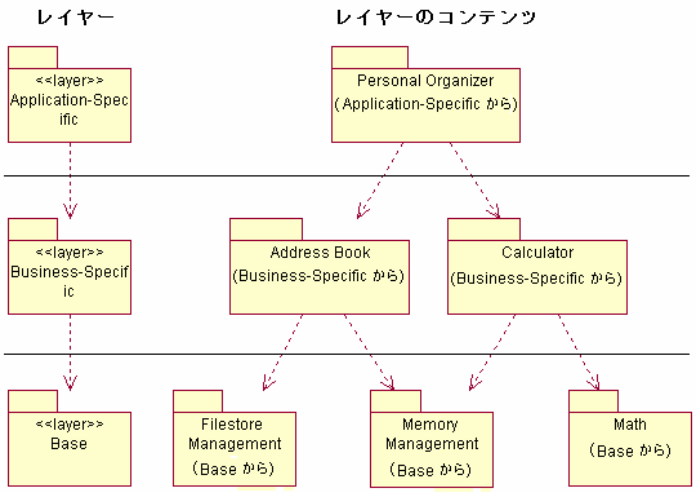

図 10:再利用に基づくレイヤリングの例

Base レイヤーの要素が最も再利用性が高く、Application-Specific レイヤーの要素はプロジェクトにより固有であるため、再利用性が低いことがわかります。

再利用に基づくレイヤーのモデリング

再利用戦略の適用は、主に*設計モデル*に影響を与えます。図 11 に示すように、再利用に基づくレイヤリングを組み込んだ設計モデルの構造は考え方が率直です。この図は、図 10 の例に基づいています。

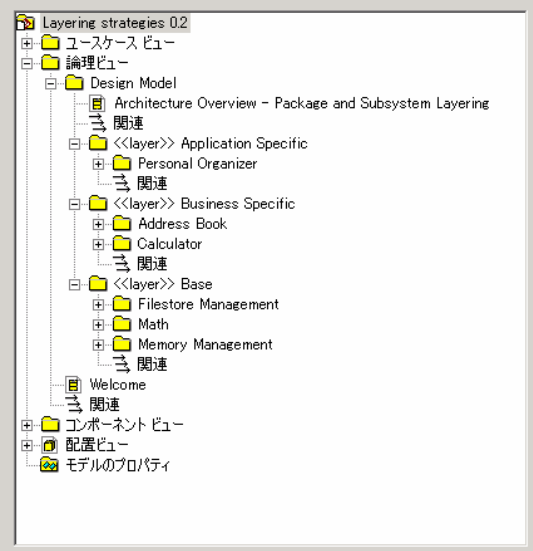

図 11:再利用に基づくレイヤリングを組み込んだ設計モデル

その他のレイヤリング戦略

このホワイト・ペーパーの目的は、最も幅広く使用されている 2 つの戦略を例として、レイヤリング戦略ごとの特色の違いを示すことです。セキュリティー、所有権、スキルなどの特徴を持つ同様の手法でも、戦略として利用できます。

多層レイヤリング

ここで説明した戦略を組み合わせて、新しいレイヤリング戦略を作成することもできます。図 12 に例を示します。

- 前の例で使用した、再利用に基づく 2 つのレイヤー
  - アプリケーション固有
  - ビジネス固有
- 責務に基づく 3 つのレイヤー (層)
  - プレゼンテーション論理
  - ビジネス論理
  - データ・アクセス論理

図 12 に示すように、再利用に基づくレイヤリング戦略の依存性は、一般的に「ビジネス論理」レイヤーの要素間の依存性によるものです。図では、整理記録ツールと住所録間の依存性を確認できます。

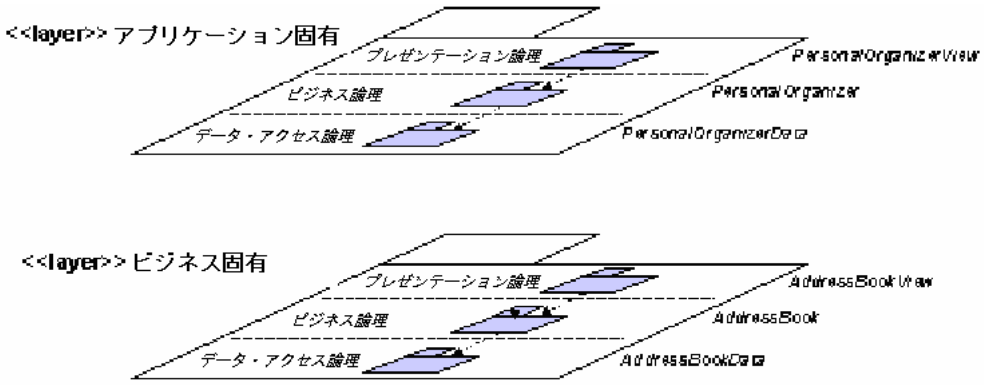

図 12:多層レイヤリング

多層レイヤーのモデリング

ここでは、2 層の設計モデル内で、レイヤリングの多層性がどのように表現されるかを考えます。また、*ビジネス・コンポーネント* の概念を組み込んだ構造についても考えます。

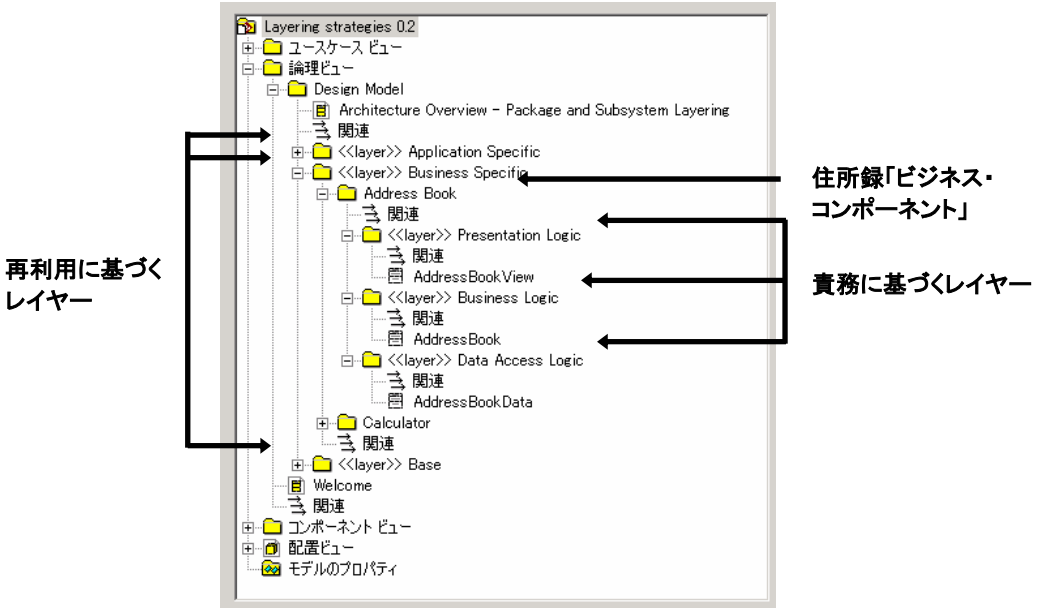

図 13:多層レイヤリングを組み込んだ設計モデル

多層レイヤリング戦略を採用する場合は、主要な戦略が確認されている必要があります。この例では、主要なレイヤリング戦略は再利用に基づいています。設計モデルはまずこの戦略に基づいて整理され、*Application-Specific*、*Business-Specific*、*Base* の各レイヤーが作成されます。これらの各レイヤーは、それぞれのレイヤーに存在する要素に従ってさらに整理されます。たとえば、図 13 は、Business Specific レイヤーに *Address Book (住所録)* と *Calculator (計算機)* が含まれていることを示しています。さらに、これらの要素は、第 2 の戦略 (責務に基づくレイヤリング) に基づいて整理されます。たとえば、*Address Book* パッケージには、*Presentation Logic*、*Business Logic*、*Data Access Logic* の 3 つのレイヤーが含まれます。

これらの各レイヤーには、このレイヤーに存在する要素が含まれます。

- *Presentation Logic レイヤー*は、AddressBookView クラスを含みます。
- *Business Logic レイヤー*は、AddressBook クラスを含みます。
- *Data Access Logic レイヤー*は、AddressBookData クラスを含みます。

結論

適切なレイヤリング戦略を選択することは、アーキテクトが行う最も重要な決定の 1 つです。レイヤリング戦略は、生成されるモデルの構造に大きな影響を与えます。さらに重要なことは、ビジネスの利益 (保守性や再利用) を、選択したレイヤリング戦略によって直接サポートできるということです。たとえば、より保守性のあるシステムは、責務に基づくレイヤリング戦略を通して、システムの各責務が相互に分離するように開発されます。また、再利用可能なシステムの要素は、再利用に基づくレイヤリング戦略を使用して明確に識別できます。

謝辞

草稿の段階で洞察に満ちたコメントを与えてくれた、Rational の Kelli Houston、Wojtek Kozaczynski、Philippe Kruchten、Bran Selic、Catherine Southwood に感謝します。

参考資料

[Buschmann] Buschmann, Frank, et al.*A System of Patterns*.1996. New York:John Wiley & Sons. ISBN 0-471-95869-7.

[Edwards] Edwards, Jeri.*3-Tier Client/Server at Work*.1999. New York:John Wiley & Sons. ISBN 0-471-31502-8.

[Eeles] Eeles, Peter, and Oliver Sims.*Building Business Objects*.1998. New York:John Wiley & Sons.ISBN 0-471-19176-0.

[Herzum] Herzum, Peter, and Oliver Sims.*The Business Component Factory*.2000.New York: John Wiley & Sons.

[Jacobson] Jacobson, Ivar, et al.*Software Reuse*.1997. Reading, Massachusetts:Addison-Wesley. ISBN 0-201-92476-5.

[PLoP2] Vlissides, John, James Coplien, and Norman Kerth. *Pattern Languages of Program Design 2*. 1996. Reading, Massachusetts:Addison-Wesley. ISBN 0-201-89527-7.

Rational®
the software development company

Dual Headquarters:

Rational Software
18880 Homestead Road
Cupertino, CA 95014
Tel: (408) 863-9900

Rational Software
20 Maguire Road
Lexington, MA 02421
Tel: (781) 676-2400

Toll-free: (800) 728-1212
E-mail:info@rational.com
Web:www.rational.com
International Locations:www.rational.com/worldwide