メール討論――オブジェクト・モデリングと実装

これは，1998 年夏から秋にかけて行った首記メール討論会での発言内容をもとに，サブテーマ別に編集したものである．「オブジェクト指向によるモデリングと実装はシームレスか？」について，まったくのフリーディスカッションが行われ，現場からの意見，方法論としての考察，実務における感触などが議論された．実際のメール討論では，さらに広範かつ詳細な質疑応答や意見交換が行われたが，本討論会の主旨に即して内容を選別し，また本稿の公開性を考慮して，全体をかなり圧縮している．初めての試みであったが，これを契機に，実際の現場での具体的な問題を中心に，互いに技術向上できる社内コミュニケーションの場が広がることを期待したい．

出席者：
会田信弘　（ビジネスソリューション三部）
妻木俊彦　（情報技術部）
大石光太郎　（新事業企画開発部）
羽田昭裕　（エンタプライズソリューション一部）
古村哲也　（ソリューションビジネス部）
藤井伸之　（ビジネスソリューション四部）
司　会：
岩田裕道　（情報技術部）
（所属は 1998 年 9 月時点のもの）

司会　テーマの中心は「モデリングと実装がシームレスにいくのか，あるいはいったのか」です．参加者各位の積極的かつ建設的な討論を期待します．

1. オブジェクト指向―現場からの意見

DOA（Data Oriented Approach）との関係

会田　「流通業務の簡単なシステムを作るのに，一々要求定義が必要とメーカは言うのですか？　お互い時間の無駄ではないですか！」　ある生協のお客さんから上記の様な疑問を投げかけられたことがあります．「なぜシステム開発をするときに要求定義をするか」ですが，初心者が SE として要求定義を勉強でする場合と，単に要求定義が顧客とベンダの通過儀式になっている場合があります．いずれにしろ①システム基盤要件，②画面・帳票の操作要件や表示要件，③業務運用ルール等が整理されないまま，客先ごとに要求定義の名のもとに何度も同じ議論を繰り返しているケースが多いと個人的に感じていました．

小売システムを構築するに当たってお客さんに主張してきたことは，本部と店舗のマスタの整合性でした．しかし，現実には店舗では本部に無い商品が納入されて販売したり，本部で廃番した商品が在庫で残っているなど，実運用としてマスタの整合性が取れていないことが度々でした．それまで，どちらかといえばデータ中心設計を得意としてきた私ですが，データ矛盾を許すモデルを知りませんでした．そういうモデルを ER（Entity Relationship）で描くことに抵抗があったと言った方が正しいかも知れません．

妻木　要求定義というプロセスで，具体的に何が議論されていたのかは分かりませんが，文脈から想像するに，多分，システム系と人間系の役割が不明確なままシステムの機能についての議論が行われているのだろうと想像されます．つまり，システム化領域の定義がされ

ていないのだと言うことなのでしょうか．問題領域の中からシステム化領域をどうやって切り出すかという問題は，確かに一筋縄には行かないところがあると思います．ジャクソンは「モデルからはじめて，その後で機能を定義する」と言っています．彼はその理由を機能よりモデルの方が頑健であるからだと言っていますが，ここでは，問題領域を人間系とシステム系に分割するには，まず実世界の抽象化/モデル化が必要であると理解したいと思います．個人的なことですが，常々，何故モデルが必要なのかという問いに直面している私としては，会田さんの問題定義を逆手にとって，だからモデル化が必要なのだと言いたいのです．これはオブジェクト指向に限った問題ではないのですから．

会田　特に DOA では，「概念データベース」「論理データベース」「物理データベース」とブレークダウンして，ビジネスルールを極力データ間の関係として明示することによりビジブルに管理対象を把握することが可能です．つまり管理するデータ群と業務ルールが同一であれば，システムアーキテクチャが異なっても論理データベースレベルでは同じ ER 図を描くことが期待できます．

大石　私の経験では，十人十色でした．あるものを ERA の Entity，Relationship，Attribute で捉えるかみな考え方が違う．また RDB（Relational Database）にマッピングするときも，正規化の程度，View の導入の程度などでさらに違ってきます．

会田　OOA（Object-Oriented　Analysis）では，“ まず実世界の事象をあるがままに捉える ” とあります．これは，DOA における概念データベースと似ていますが，DOA ではデータの固まりとして認識できるものを分析対象（の中心）としているのに対し OOA では，“ 名詞的 ” および “ 動詞的 ” なものをオブジェクトとして分析することになります．これは，OOA が非常に柔軟性を持つことを可能としている反面，設計者のスキルや経験，並びに分析する視点の違いからオブジェクトをブレークダウンまたはロールアップ（汎化）する上で個人差が出やすいことを意味します．管理対象のオブジェクトの機能とデータを分析する上で，ユースケースが使われるようですが，その振る舞いの抽出は帰納的ですのでオブジェクトのライフサイクルを網羅する上では不充分です．「ビジネスは変化する」のが今日当たり前になっていますが，DOA では対象データの関係性が変わらないのであれば，その変化は「機能」（すなわちプログラム）に押し込めることができます．OOA では，想像力不足のオブジェクトモデルは，ビジネス変化がそのモデル全体に大きく影響することが考えられます．

妻木　データ・モデルがデータ構造でビジネスを表現し，オブジェクト・モデルがオブジェクト構造でビジネスを表現しているという議論に，若干，引っかかるものがあります．つまり，データ間の関係で本当にビジネス・ルールが表現できるのでしょうか．多分，ER 図よりも表現力の高いクラス図を使っても，十分に表現しきれないビジネス・ルールを ER 図で表現できるというのは，どこかにごまかしがあるのではないかと思います．

会田　妻木さんの指摘のように ER 図ですべてを表現するのは困難で理解しづらいことは明らかなのですが，データの関連性を明示するだけでも建築の構造設計位には相当するのではないでしょうか．流通のシステムの中で，もっとも重要なデータ項目として “ 売価 ” と言うものがありますが，“ 売価 ” を管理する論理データベースを，“ 商品 ”“ 商品・店 ”“ 商品・地域 ”“ 商品・取引先 ” のいずれに設定するかで，システムの顔が変わってしまいます．

羽田　DOA では，ビジネスロジックをプログラムの側からではなく，データベースの側から捉えることを強調しているようです．ある種のビジネスロジックをデータベースに括り付けることにより，従来いろいろなプログラムに散在していた “ チェック ” を一箇所に纏められる，そのことで保守性が向上する，という主張です．ビジネスロジックにはいくつかの分類方法があるようですが，例えば Usoft では，明示的（叙述的：declarative）ルールと暗黙的（構造的：structured）ルールに分けています．このうち，構造的ルールが ER 図で表現され，DOA で自然に出てくるルールです．Usoft では，制約（restrictive）ルールと修正

(corrective) ルールに分けています．A の値は B を超えてはいけない（制約）とか，条件 C が成り立つ場合は値引きをする（修正）とかです．完全性（integrity）ルールと加工ルールという呼び方もあるようです．実装レベルの言葉で言えば，データの作成・更新・削除のルールです．

ER 図で表現できないのが，「テーブル間の関係で表現できない」ロジックでしょう．叙述的なルールです．AI（Artificial Intelligence）のルールベースで書こうとしていたような分野だと思います．複雑な経路でデータをアクセスする必要があったり，データ項目の関係だけでは表現できない，ビジネスの慣習や規則です．いわゆる“運用”です．多分この辺の議論は，DOA ユーザの間では常識なのだと思います．基本的には，ビジネスロジックはオブジェクトの関係で表現されますが，オブジェクト指向では，上記の二つの分類のルールにどう対応するのかを問われているのだと思います．

大石　DOA の権威である堀内一さんが『ビジネスルールとオブジェクト』のテーマで講演し，ビジネスルールの表現手段には，以下の四つがあると整理されていました．

① データベースの参照制約で表現できるもの
② オブジェクトの型制約で表現できるもの
③ オブジェクト間の関連の制約で表現できるもの
④ プロダクションルール（プロセスで if〜then）で表現できるもの

妻木　ビジネスロジックというのをビジネスルールと読み替えると，とても良く分かります．そこで，叙述的なロジックを独立したオブジェクトで表現した方が良いのか，エンティティ間の相互作用で表現した方が良いのかという問題が発生してきます．基本的には，どちらでも表現可能なので，それはスタイルの問題かとも思うのですが，それを「あるべき」論で大上段に振りかぶられると，ちょっと辟易してしまいます．

OOA のモデルが大きく影響を受けるというのは，モデリングの問題と，ビジネス変化の大きさという二つの問題があります．メイヤーも言っているように，機能よりもデータ構造の方が不変であり，会田さんの“名詞的”オブジェクトというのも，そういう主旨だと思います．しかし，どんなモデルでもビジネスの変化に無関係であるとは言えないのではないでしょうか．ビジネス変化が大きければ，当然モデルも変化するわけで，その変更を如何に局所化するかが重要な点といえるでしょう．基本構造の頑健さとか，構成部品の取り替えの容易さとか，ホットスポットの局所化などが言われていますが，その顧客のビジネスの特徴を正しく理解しているかどうかに依存する問題だと思います．

“名詞的”なオブジェクトとか“動詞的”なオブジェクトという議論ですが，個人的には，モデルと言うよりアプリケーション・アーキテクチャの問題と考えています．ランボーは，属性の無いオブジェクトはオブジェクトではないといっており，ワーフスブロックは，オブジェクトは問題領域の中で果たすべき責務を担っているといっており，実体をオブジェクトとして識別するというのが世の大勢と思われます．そういう意味では“名詞的”なのでしょうか．“動詞的”なオブジェクトというのは，ビジネスを制御するような機能をオブジェクト化したもの，トランザクション・データをオブジェクト化したもの，オブジェクトの集合を操作するためのオブジェクトなどがありますが，これらをシステム上の機能で分類するとアプリケーション・システムの基本的な構造が見えてくると思います．これをアプリケーション・アーキテクチャとして定義すれば，何がオブジェクトかという議論はしないで済むと思います．

大石　機能だけのオブジェクトは，属性のないオブジェクトです．ランボーはもともとデータベースが専門なので，初期のころは，属性のないオブジェクトはオブジェクトではないと思っていたのかも知れません．

羽田　ところで，機能だけのオブジェクト，データだけのオブジェクトでもいいかという

問いかけですが，インタフェースが定義できれば，オブジェクトと見てよいのでは，ないでしょうか．データだけのオブジェクトは，きっとデータアクセスという機能があるので，当然オブジェクトです．ただ，本来実装上の内部データまで見せるとオブジェクト的ではない，ということだと思います．機能だけのオブジェクトは，本当に機能だけなら単なる関数です．普通は（暗黙に仮定していたり，メッセージとして引き回したりしていますが）状態を持っているのではないでしょうか．それならオブジェクトでしょう．実装上出てくるクラスについては，業務システム開発上は，あまり神経質になる必要は無いと考えています．

会田　オブジェクトモデルをビジネスに実装するときに暗黙に期待しているのは，プログラムの部品化であることは間違いありません．オブジェクトの継承機能や多様性を用いれば，新たなオブジェクトを作るときに差分プログラムの技術が使えるかも知れません．しかし，ビジネスは複数のオブジェクトが時間軸の中で，相互関連をもちながらビジネスを進めていくわけですから，オブジェクトの数が増えていくとその関連性を見極めるのは困難です．また，継承機能を用いた場合に階層が深くなると親の機能が何だったのかを正確に把握することが難しくなることが予想されます．オブジェクト化を進めれば進めるほど，再利用することを阻害するというジレンマをこの方法論は抱えているのではないでしょうか．

妻木　クラス階層構造の深さは，プログラムの読み難さにつながり，Smalltalk が優れた言語であるにもかかわらず一般化しない理由の一つに挙げられると思います．そんなこともあって，ガンマなどは，デザイン・パターンの中でインタフェースに対してプログラムすべきであると言っています．どこまでの階層構造がよいのかを計数的に述べることはできませんが，後工程の作業者が負担にならない程度にしておくべきでしょう．

バッチ処理

会田　OOA が，実装上の仕組みとして“リアル”，“バッチ”を区別している訳では無いのは自明ではありますが，皆さんも OOA ではバッチ処理をどうオブジェクトとして表現したら良いか悩んだことはないでしょうか．誤解を覚悟で言うのであれば，小売の発注業務をバッチとリアルで処理するのでは 10 倍程の処理効率の差がでてきてしまい，データ処理コスト上そのモデルが受け入れられないと考えてしまいます．

妻木

・バッチ処理によってオブジェクトのカプセル化が破られる．

・オブジェクトの特定の属性の集合をクラス図で表現できない．

などの問題があり，一般にはトランザクション・バッチが推奨されていますが，効率問題が発生する場合があります．ここで，視点を変えてみると，ビジネス・ルールとバッチ処理というのはレベルの違う問題であることに気が付かれると思います．つまりバッチ処理というのはコンピュータシステムの処理モードのことであり，問題領域のモデルとは次元の異なる議論と言うことになります．ですから，バッチ処理の仕方をモデル化しようという議論そのものが間違っているのではないでしょうか．バッチ処理をモデル化したいのであれば，それは，問題領域のモデルではなく，システム上の処理手順としてモデル化するのが適当だと思います．UML（Unified Modeling Language）にはそのための動的なモデル記述法が用意されています．どこかにバッチ処理のためのデザインパターンがあったら，教えて下さい．

羽田　バッチ処理をデザインパターンとして特許出願中という記事が，日経ソフトウェア［四倉幹夫：アプリケーション・フレームワーク「NSObject」，1998. 10，日経ソフトウェア］に載っていました．処理モードのロジックのパターン化という位置づけです．ビジネスレベルでのオブジェクトの構造とは関係ないところに位置づけられているようです．

永続オブジェクトの扱い

会田　詳細は省きますが，さきほどの 3 層データベースに模した形で流通業のビジネスオブジェクトを前提にした場合，「概念オブジェクト」「論理オブジェクト」「物理オブジェク

ト」および「ビューオブジェクト」という形で造出するのがDOA的には自然だと考えています．これをクライアント/サーバ・システムでは，ユーザインタフェース・オブジェクト，データアクセス・オブジェクト，ビジネスロジック・オブジェクトのように分類しますが，関係データベースをバックボーンにするとき，後の2種類のオブジェクトでの永続化の問題があります．また，分散オブジェクトモデルであれば，オブジェクト間の通信の信頼性を保証するために，通信インタフェース部分を永続化できる仕組みが必要なのだと思います．CORBA仕様に関して一般的知識しか持ち合わせていませんが，ORBの永続化サービスとはそのような要請から存在するのでしょう．永続化を自前で作ることは困難であり，System *v*［nju:］などの基盤が必要と思います．

オブジェクトの永続化をしなくても良い分散モデルでも業務システムは作成できるのでそこは割り切り方でしょう．オブジェクトのインスタンスに対してユニークな識別子を与えなければ分散型の場合は矛盾が生じるはずですが，CORBAのネーミングサービスとは内部的にそのメカニズムをサポートしているということなのでしょうか？

羽田　CORBAでは，Object Referenceを通じて，Objectにメッセージを送るという仕組みになっています．このObject Referenceは，ORBによって，自動的にユニークな識別子が振られます．通常，この識別子の振り方，識別子の値を知る必要はありません．このObject Referenceの引き方の一つが，Naming Serviceです．Naming ServiceはLDAP（Lightweight Directory Access Protocol）のようなDirectory Serviceと考えるべきで，データベースアクセスのような形態での利用を目指したものではありません．

会田さんのご指摘の通り，分散システムでの名前の一意性と，階層性を保障するための仕組みですが，オブジェクトの識別性とは，区別されます．名前が無くても，オブジェクトはアイデンティティを持ちます．Naming Serviceで探すオブジェクトは，おそらく永続化されているものです．永続化する方法の一つがPersistent Object Serviceです．

妻木　今のところ，ユーザインタフェース・オブジェクトやデータアクセス・オブジェクトを最も洗練した形で実装しているツールは，FORTEだと思います．私は，FORTEのアーキテクチャを知ることによって，何度か目から鱗が落ちた経験をしています．

羽田　“Building Scalable Database Application”という本［C. P. M. Heinckiens, Addison-Wesley, 1997］に，MVC（Model-View-Controller）を拡張して，データベースのインピーダンスミスマッチを解消しようという方法が提示されています．これは，

- データベースは実際には，複数のアプリケーションから利用されるので，一つのビジネスモデルから導かれる構造にするのは現実的ではない．
- 効率向上のチューニングのためには，データベース構造は，（ビジネスモデルと関係なく）変更される．この変更のアプリケーションへの反映は困難である．

という認識に出発しています．

ビジネスモデルのクラス構造をmodelとし，実際のデータベース構造をViewと位置付ける．実際のデータベース構造にはインピーダンスミスマッチがある．それは，属性と項目との型の対応と，テーブルとクラスの多対多対応であるから，これをcontrollerが調整する．というアプローチです．ただ，このアプローチだと，どこからデータベース構造が導き出されるのだ？　という疑問が湧きます．この本では，オブジェクトモデルと独立にデータモデルは導き出され，データモデルをデータベース構造に反映させるという道筋です．二つのモデルには関係があり対応づけられるが，独立に導けるというのです．確かにそうなのですが，ただ，二つのモデルの関係がはっきりしない．すっきりした説明がありますか？　システムの分散配置の方法が，オブジェクト構造からは直接導けないが，分散単位とオブジェクトの単位が関係するように，データベース構造は，モデルから直接導けず，設計・実装上の産物であることは，確かですが．

大石　データモデルの導出について，私は，オブジェクトモデルの中の永続データを持つオブジェクトをベースにデータモデルを考えます．

2.　シームレス論
方法論として

羽田　ちょっと古い記事になりますが，羽生田栄一氏が日経エレクトロニクスに「統一モデリング言語 UML の現状と課題」という記事を書いていました［1997. 9. 8, No. 698/1997. 9. 22, No. 699］．そこでランボーらとシュレイア＆メラーの「推敲派 vs. 翻訳派」の論戦について言及されていました．分析モデルを洗練していき，実装上の判断を入れていくと設計モデルになるというのが前者で，分析モデルと設計モデルは別物で，設計モデルの一部に分析モデルが埋め込まれるというのが後者です．電力中央研究所（電中研）のシステム[*1]は，後者の発想で作りましたが，基本的には，JSD（Jackson System Development）から着想を得ました．

大石　ランボーらは，推敲派なんですか!?　推敲派は，オブジェクト指向プログラミング言語屋さんに多いように思います．Eiffel を作った Bertrand Meyer さんもセミナーで，そのようなことを言っていました．そのとき彼は，オブジェクト指向分析ツール（お絵描きツール）などはいらない．そんなものに時間をかけている暇があったら，いきなり Eiffel の言語でクラスを定義していけばいいのだ．あとはそれを詳細化，肉付けしていけばいいのだと．“ シームレス ” は，よく日本訳では，“ 継ぎ目のないこと ” と訳されますが，これだとあまりよく分かりません．Webster では，“ 1：having no seams　2：having no awkward transitions or indications of disparity ” とあります．2 を拙訳すると，“ ぶざまな（やっかいな）変化や不均衡な徴候がないこと ” となります．“ ぶざまな変化や不均衡な徴候 ” とは，具体的にどのような状況を言うのかある程度考えておいたほうがよいのかも知れません．たとえば，オブジェクト指向分析してできたオブジェクトモデルを実装するのに非オブジェクト指向プログラミング言語（COBOL，C）や，非オブジェクト指向データベース（RDB）でやって，モデル変換をせざるを得ず，そのための工数が多くかかったとか．オブジェクトモデルの永続クラスのために，オブジェクト指向データベースを使ったが，効率が上がらないので，オブジェクトモデルを非正規化やビューの作成などしたとか．これらは “ ぶざまな……徴候 ” なのか，そうではないのか．

古村　1996 年に行われた社内の最初の OO ワークショップで，モデル作成者がモデルの原案をターゲット・マシンと独立に設計し，実装は別人がやった結果，実装モデルは当初のモデルとは様変わりしたものになった，と言う報告がありました．OOA 担当者が実装するアーキテクチャを視野に入れないと，このような状況は避けられません．システムの保守は実装モデルに頼るので，優れた OOA であっても役にたたなくなる．こういう点でもシームレスではない気がします．

妻木　アプリケーション・システムと言っても，実はその中にはいろいろなものが含まれていると思います．メインフレーム上でアプリケーション・システムを作っていたときは，プログラムの作り方というようなものが決まっていて，システム・プロセッサーが提供してくれる統一的な API を使ってプログラムを作成していれば良かったので，その雑多性が問題にならなかったのだと思います．それはメインフレームが独自の単一アーキテクチャを持っていたからで，それがオープン・システム時代になると異なったアーキテクチャ上にアプリケーション・システムを構築しなければならなくなった．アーキテクチャ・フリーになってしまったのです．そうするとどうやってアプリケーション・システムを構築すればよいか誰も教えてくれない．下手をすれば，A 社の通信システムに合わせて作ったアプリケーション・プログラム（AP）が，B 社のデータベースでは上手く動かないということが起こる

わけです．メインフレーム時代には考えられないことが平気で起きるわけで，そのことはオブジェクト指向と直接的な関係はないでしょう．むしろ，ドメイン・モデルとシステム・アーキテクチャの問題として捉えるべきことと思います．指摘された問題というのは，ドメイン・オブジェクトがユーザインタフェースのための属性や操作を抱え込んでしまった点にあり，DFD（Data Flow Digram）に比べオブジェクト・モデルがアーキテクチャルな分，その構造の悪さが目に付いたのだと思っています．システム・アーキテクチャの違いを埋めるという観点からこの問題を考えてみても良いかも知れません．ドメイン・モデルがコンピュータ・システムから独立すべきであるというのは，テーゼだと思いますから．

岩田　Jacobson の著書（OOSE など）ではシームレスの具体的な意味付けとして分析・設計・実装の段階で洗練されていく各モデル同士の双方向のトレーサビリティをあげています．実際には，シームレスはもっと広い意味でいう場合が多いようですが．実務的な立場からはシームレスのこの意味付けは具体的な気がします．

羽田　トレーサビリティも，推敲派・翻訳派，いずれに組するかで，違って見えてくると思います．前に紹介した羽生田氏のペーパーでは，

- 推敲派の立場で進めると，純粋にモデルのレベルでは，トレーサビリティをいえなくなる．そもそも，分析レベルと設計レベルの境界が不明確．
- 翻訳派の立場だと設計モデルのどの部分が，分析モデルのどの部分を前提にしているか，というトレース情報が生まれる．モデル要素間の対応関係が明確になる．

という整理がされていました．

岩田　これらはどういう意味でしょうか？

羽田　詳細化を行なっていく中で，「モデルの一貫性を保つ機能」をトレーサビリティと位置づけています．分析モデルと設計モデルの関係は，純方法論的な議論ですが，「モデル中に新たなモデル要素を追加する場合に，元のモデルとどう関連付けるか」という問題に還元しています．

- 推敲　「分析モデルを詳細化し拡張した結果が設計モデル」「分析モデルが徐々に改訂を受け，推敲された結果として設計モデルが出来上がる」

これは，分析モデルと設計モデルを含むオブジェクトモデル空間を想定した場合，その中の 1 点が分析モデルで，もう一点が設計モデルである，というイメージです．詳細化の過程が，2 点を結ぶ軌跡となるのですが，これが連続で滑らかであれば，シームレスという感じがします．ただ，この軌跡の中で，どこまでが分析モデルでどこからが設計モデルかを客観的に決める基準がありません．そもそも，どこの位置にあるのが分析モデルで，どこの位置にあるのが設計モデルかも分かりません．だから，「純粋にモデルのレベルだけでトレーサビリティをいうことができない」

- 翻訳　「分析モデルと設計モデルは互いに独立したモデル」「設計モデルのアーキテクチャの一部に分析モデルは埋め込まれる」

これは分析モデル空間と設計モデル空間があり，それぞれの空間中に分析モデルの点と設計モデルの点がある．その間の変換ができれば，シームレスというイメージです（多分，分析モデル空間と設計モデル空間には，“ オブジェクト性 ” というような何らかの共通な性質を持つ必要があると思いますが）．だから「設計モデルを変更しても分析モデルは不変」「設計モデルのどの部分が分析モデルのどの部分を前提に作られたのか，という縦のトレース情報を保持する」．JSD もこのような枠組みで，モデルの段階と実装の段階を区別していると認識しています．

第 3 の立場として，分析モデルも設計モデルも，同一のモデルの異なるビューに過ぎない，というのもあるそうです．この共通モデルが，リポジトリに含まれ，分析ビューや設計ビューが取り出せる，という構造になります．

岩田　共通モデルとはどのようなもののことを言っているのでしょうか？

羽田

- 共通モデル　「分析も設計も同一の共通モデルの二つの異なるビューにすぎない」「実現手段に関する情報を捨象し，より抽象度の高い視点で問題ドメインに関する部分のみ見せるのが分析ビュー」「実現手段とその前提となるモデル情報を見せるのが設計ビュー」　これは，オブジェクトモデル空間に一つの点がある，これが共通モデル．これを設計空間に射影すると設計ビューが分析空間に射影すると分析ビューが見える，というイメージです．

妻木　こう言っては身も蓋も無いのですが・・・実際のシステム開発では，推敲することも翻訳することも共通することもあると思います．色々な方法論も，そういう観点で見て行けばみんな含まれています．ただ，方法論の作成者達にすれば，旗幟を鮮明にする必要があるので，自分の基本スタンスはこうだ！ということになるのでしょう．しかし，我々は方法論の作成者ではないので，ちょっと別の視点からシームレスという問題を取り上げてはどうでしょうか．

例えば，今までのシステム開発に較べてシームレスだと感じたかどうかとか，実際のシステム開発で上手くいったかどうかということです．何故そうなのかという理由は人それぞれだと思います．私の場合，モデリングまではやるのですが，実装の段になると人任せで，側で見ているという状態になってしまうのですが，モデルがきちんとできていれば，実装にそんなに苦労している風には見えませんでした．モデルがきちんとできていない所は，当然ながら苦労されていましたが…．もう一つ気が付いたのは，実装言語が高級言語になればなるほど，モデルとの乖離が大きくなるということです．これは，仕様とはなにかという問題提起だと思うのですが，現在のオブジェクト指向モデリングは状態遷移図を除けば，極めてプリミティブな OO 言語を想定しているためではないかと思われます．

藤井　ここ数年，FBA　Navigator という銀行，証券会社の営業店システムのパッケージを開発していますが，このパッケージのサーバ部分のアプリケーションを OMT で開発してきました．経験面で意見していきたいと思います．「シームレスか？」ということに関してですが，ずっと，OMT（Object　Modeling　Technique）で開発したためか，シームレスであると思い込んで，分析/設計を行ってきました．その結果，モデルとしては，1 枚（設計モデル）しか残らず，逆に保守がしやすくなりました．

分析モデルを残す場合は，どこで区切りをつけて残せばいいのでしょう？　実装上の判断が入ってないものだとすると，FBA　Navigator の場合は，シームレスではなく，設計モデルの一部に分析モデルが埋め込まれたのだと思います．分析時（初期）のモデルと最終的なモデルでは，あるオブジェクトは引き継がれているが，モデル（クラス図）は，全く違っているものができています．

羽田　推敲か翻訳か，とトレーサビリティの議論は，純理論的な問題だと思っています．ただし，CASE ツールをどう作るかを考えると，この分岐は大きく方向性を定めるものだと考えます．

ところで，実際のシステム作りについてです．オブジェクト指向で，分析し，設計し，プログラムに落とすという考え方で，シームレスということを考えたくない，というのが私の立場です．そのための議論の枠組みが，推敲か翻訳か，なのです．分析モデルは，なるべく直接実行できる仕様となるべきだと考えています．そのためには，いきなりプログラミング言語で書くのではなく,そのモデルが動くインフラなりミドルを作るべきだと考えています．そのようなインフラやミドルを作る，またはモデルを分類するためにフレームワークやパターンを定めるときに，オブジェクト指向は役立ちます．また，コンポーネントなどの部品や部品の組み合わせを実現するのに，オブジェクト指向言語は便利です．こういう流れで，シ

ームレスを考える方が現実的ではないでしょうか？　分析モデルを設計モデルにはめ込み，設計モデルはインフラの上で“そのまま”動く，というイメージです．実際のデータベースにRDMSを選んでも，ミドルがそれをオブジェクト風にラッピングしてしまえば，いいのではないでしょうか．言語とモデルの乖離はこのようにして解決すべきだと思います．プログラミングをさせないのです．

実際の開発事例ではどうか

岩田　実装（製品の適用やプログラミング）やテストまで含んでの，開発現場での現実的な感触はどうなんでしょうか？

藤井　以下に，今までのOO開発について，簡単に紹介します．

1）　FBA　Navigatorの場合は，要求定義→OO分析→OO設計→C＋＋実装→テスト（データベース（D/B）および，CASEツール使用せず）という工程で開発してきました．

要求定義は，従来どおり日本語で，機能を記述し，そこからOO分析を行いました．実際には，要求定義以上のことも考慮しているので，ここからがスタートかもしれません．

OO設計では，特にCASEツール，等使わずにモデリングし，VISIO/WORDで，クラス図，状態遷移図，メッセージシーケンス図，データ辞書を作成しました．

実装は，NT上で，VC＋＋で開発を行いましたが，モデルからC＋＋への翻訳＋推敲が大変でした．

テストでは，クラス単位でのテスト，モジュール単位のテスト，実行モジュール単位でのテストを実施しています．クラス単位でのテストは，モデルの状態遷移，メッセージシーケンスが正しいかの確認を，モジュール単位および実行モジュール単位でのテストは，要求定義での機能確認をしました．

- 状態遷移

  状態遷移ももつクラスは，そう多くはありませんが，通信系のクラスについては，状態遷移がありました．この実装方法について，現在では，デザインパターンの中に何種類かありますが，初期開発時は，まったくシームレスではなく，いろいろ方法を考え，最初のほうに開発したクラスとあとで開発したクラスでは，実装の方法が異なってしまいました．（当然，後のほうがきれいな実装になりましたが）

- 多重継承

  C＋＋で実装できないことはないですが，同一属性をもってしまう，等の問題が発生し，継承の階層を変更しました．これについては，モデルにもどって修正しています．

- 各種ライブラリの利用

  ストリングクラス，リストクラス，等の利用による属性の修正

- ISAM

  D/Bは使用していませんが，ISAMファイルを使用しています．

2）　N証券における口座管理システムをダウンサイジングするためのプロトタイプ開発

プレート分析（仮称）[*2]→OO分析については，証券業務パターンを作成し，翻訳ルールを作成しようとしたが，プロトタイプのため中断．OO分析からはROSE/C＋＋を使用し，クラス図，メッセージシーケンス図を作成，pure　AP部分と，ミドル部分（ここでは，APとTUXEDO間のミドル），D/B部分をあらかじめ，分けて設計．AP部分，ミドル部分については，C＋＋のヘダー，スケルトンまでROSEで作成できるので，非常にシームレスです．あとは，C＋＋でロジックを書いていくだけでプログラムは完成です．属性，操作が変更される場合は，ROSEで修正します．D/B部分については，D/Bの種類によりROSEのスケルトンを一部変更します

3）　SWEETS（VB部品管理ツール）開発

リポジトリ（UREP）で管理したいものをRATIONAL/ROSE上で，クラス図に落とし，それをUREPのRSMから継承させるだけでOO設計まで終了です．UREPの機能により，ROSEモデル→ODL→C++ヘダー，および関数の一部までできてしまいます．あとは，インタフェース関数を少し作成するだけで完成です．非常にシームレスです．

羽田　電中研のシステムでは，分析用のモデル図と設計用のモデル図をわけ，その変換規則を定めました．そして，設計モデルを入力すれば，モデルが動き出す仕組みを作りました．このシステムは，離散シミュレーションのモデルです．離散シミュレーションの世界では，simula，simscriptなどの歴史を持ったモデル言語があり，それらはそのモデルを入力して，それが“動く”のです．数理計画法などORの分野では，普通のアプローチだと思っています．このシステムでは，それをOMT流にアレンジした道具を用意しました．具体的には，状態遷移機械，クラスメカニズム，タイマーです．基本的には，シュレイアー&メラーのlife-cycleの教科書［S. Shlaer and J. Mellor，本位田・伊藤監訳，続・オブジェクト指向分析―オブジェクト・ライフサイクル―，1992，啓学出版］に書いてある通りに作りました．但し，メソッドの中身は，C++で書きました．メソッドの中身は，実装に関わらない範囲で書くと言う道徳的な規律しかありませんでしたが，もう少し時間があれば，この点は詰めたかったところです．どうやって動かしたかは，技報の41号に書きました．興味がありましたら，ご覧ください．

分析モデルは，なるべく直接実行できる仕様となるべきだと考えています．従来は，インフラがあって，その上で記述してきた．それなのに何故OOとなると，素手で戦わなければいけないのかが理解できないのです．無ければ，作れば良い，と思います．世界のどこでも通用する，フレームワークを想定するのは，極めて困難だと思いますので，パターンや事例が蓄積されていくのだと，思っています．

大石　分析モデルや設計モデルをそのまま動かすようにしたいという純粋な気持ちは理解できますが，そのためには，モデル内のクラスの各メソッド（オペレーション）の中身を実行可能なくらい十分に定義（疑似コーディング）しないといけないのではないでしょうか．ほとんどがモデル内の他のクラスやミドルとして用意されるクラスの呼び出しだけだとしても．

羽田　Shlaer & Mellor法でも，この部分を記述する言語が提案されています．そのような言語を想定するまたは，規定するかどうかはともかくとして，分析と実装では，実装環境を意識するかしないか，という差があるわけですから，ここまでは仕様，ここからは実装依存と分けられると思います．ここをグチャグチャにしたら，OO以前の昔の世界へ戻ってしまう気がします．分析モデルがきちんと書ければ，翻訳すると言う道は開けると思っています．ただユーザの要求をそのレベルに落とし込むところに，シームレスでない部分があります．ユーザやOOに不慣れな作業者が理解できるレベルで，問題領域を記述する必要があります．これと，OO分析モデルの間が一番大きな隙間があるのではないでしょうか．

岩田　まったく新規にAPを作るのと，既存のものとの共存が要件になる場合とでは，事情が違うと思われます．翻訳派だけに頼るわけにはいかない現実があり，モデリングから実装，保守まで含む全ライフサイクルで考える必要があるのも事実です．モデルの保守の問題も指摘されています．当然関係DBとのマッピングは避けて通れませんし・・・．

羽田　新規に作る場合も，旧来の資産を引き継がなければなりませんから，移行というのが難物です．オブジェクトが持つ識別性と，業務的な識別性，従来のDBがもつ識別性は，単純に対応できないのが現実だと思っています．翻訳はできるのですが，OOからかけ離れた技術を採用すれば，翻訳の規則は難しくなり，やり方によっては非効率になる．OOと親和性がある技術を使えば，“枯れた”技術が無いので，信頼性に欠ける，というのが悩ましいところです．OOに向いた業務というのは，業務の性格だけでなく，必要な信頼性や市販

の道具との相性による部分が大きいと思います.

岩田　APについては，実際にはどのような感触なのでしょうか？

羽田　電中研のシステムでは，分析用のモデル図と設計用のモデル図をわけ，その変換規則を定めました．そして，設計モデルを入力すれば，モデルが動き出す仕組みを作りました．正に翻訳です．道具作りは若干手間取りましたが，モデルの変更には容易に対応できました．

3. OO周辺の話題

デザインパターン

妻木　ある会議で某大学の先生が，「設計でデザインパターンを使ったが，デザインパターンはオブジェクト指向パラダイムにはあってないのではないか？」といってました．確かに，デザインパターンは，オブジェクト指向独自の計算方式に基づいてある種のアルゴリズムを表現しようとしていますが，それは，オブジェクト指向計算モデルの基本である概念のカプセル化には程遠い手法といえます．設計フェーズでは，このようなアルゴリスミックな手法を取り入れても良いのでしょうか．それとも，最後までオブジェクト指向計算モデルを守るべきなのでしょうか.

藤井　デザインパターンを意識して分析/設計するか，というとそうではないと思います．デザインパターンは，物理設計〜実装にかけて使うものだと思います．物理設計では，実装するインフラの制約により，パフォーマンスの向上や，保守性向上，等の目的をもって設計を行うと思いますが，その時の保守性の向上に役立つものだと思います．デザインパターンを使うことにより保守性は向上すると思います．オブジェクト指向の良さはカプセル化であり，何らかの修正が入る場合は，そのクラスだけ修正すればよいということだとすると，デザインパターンを使うことによりその修正個所がさらに局所化されるという良さはあると思います.

羽田　オブジェクト指向パラダイムとは次のようなものだと考えています.

(A)「もの」から共通な性質を「抽象」して，「クラス」に分類する．ここが狭義の「カプセル化」でしょう.

(B)「クラス」から共通な部分を「捨象」して別の「クラス」を導く.(「汎化ー特化」関係を築く.)

ここから次のような主張が生じます.

1) (A)から現実世界に足場を持つことになり，モデルがロバストになる.

2) (A)(B)を組み合わせることにより，差分プログラムが可能になり，変更の局所化が実現される．また部品化・再利用の道が開ける.

3)「抽象」「捨象」により動的な振る舞い・関係が静的に表現される．このためシステムが取り扱いやすくなり，計算モデルも定まる.

4) (A)(B)の仕組みは，そのままの形で，計算モデルとして実装可能であり，ここから分析から実装までシームレスになる可能性がある.

さて，パターンは，このような主張を破るものではないので，オブジェクト指向パラダイムに反しているとは思えません．ただ，(A)の抽象化の部分はなく，(B)の「捨象」の部分をクローズアップしています.(それゆえパターンとしてしか表現できない，と考えます.)ですから，オブジェクト指向パラダイムから見れば，非常に偏ったものと思えます．しかし，これは仕方の無いことだと思います．以下その理由です.

パターンのうち，設計パターン(design pattern)に関して言えば，その名の通り設計段階で用いるパターンです．分析段階では，「もの」が出てきて，シーケンス図だとかユースケースだとか具象的に表現される．これにより，客先と合意しやすい道具となっている．一方，合意を収拾させ，プログラム化するには，静的な(＝抽象的な)表現に変更される．設

計過程では，この抽象化を推し進めて，具象的な部分を拭い去る．そのことで，実装可能なモデルになっていく．したがって，（B）の「捨象」が主役になっていくでしょう．

「振る舞い」「構造」「生成」といった，直接「もの」に根拠を持たない概念を「捨象」することに違和感はあります．パターン自体は，オブジェクト指向パラダイムを現実のものとするための，道具の一つであり，それを補強するものと捉えるのが自然だと思います．

UMLについて

岩田　シームレスに関連してUMLの機能を一部紹介します．

1）　モデルあるいはモデルの一部の間で「refinement」とか「trace」という関係づけを与える概念がある．これはモデルの詳細化過程を図式の中で明示的に指定することで，トレーサビリテイを保証しようとする意志が見られる．

2）　クラス図の中で，クラスとその実装クラスとを陽に関係付けることができる．これは分析・設計段階の図式要素と実装とのトレーサビリテイをビジブルにする．例えば，Setの集合体クラスをその実装クラスであるHashTableクラスと関係づけるなど．

これらの2例は，複数のモデル間の関係や，開発工程全体の中の異なる工程で作成される（したがって従来なら別々の独立な図式に現れる）要素との関係を陽に記述できるようにしています．

UMLの制定は，ユーザが直接利用してモデリングを行うという面でのメリットは当然として，一方CASEツール開発者がUMLの提供する諸概念を実装する製品を作るのを促進するという面があります．これらの2例は，CASEツールがこれらの概念をサポートし，したがってシームレスを技術的にビジブルにすることを促進するという意味で非常に良いことだと思います．UMLが開発プロセス全体を一貫した表記法で支援することから可能になる機能であり，私は画期的な概念だと思っているのですが．

大石　9月4日（金）PMに全日空ホテルで開かれたRationalROSE 98の発表会に行ってきました．そのとき，同じような感想を持ちました．ROSE 98は，コンポーネントモデリングとコード生成，コードからコンポーネントインタフェースのリバース，モデル内にインタフェースと実装の関係（具体化）を定義できるなど，分析設計と実装が随分近づいていると思いました．

コード生成はまだ中身のないスケルトンですが，OMGなどで検討されているメソッドの中身を記述するAction　LanguageがUMLに追加されれば，さらにシームレスに近づくと思います．

またトレーサビリティについては，これはシームレスを計る一つの尺度，あるいはシームレスを具体的に記述する一つの要素だと思います．トレーサビリティだけでシームレスを100%語れるものではないと思います．トレーサビリティは，ソフトウェア開発プロセスのある時点からある時点に辿っていける容易さです．分析・設計プロセス間だけでなく，実装も含まれます．要求からコードを辿る．コードから要求を辿る．要するに要求通り実装されているかを容易に辿れることだと思います[参考：Essays on Object-Oriented Software Engineering, volume 1, Edward V. Berard, 1993]．

現在，EJB（Enterprise Java Beans）コンポーネントモデルをサポートするアプリケーションサーバ製品が一斉に開発，リリースされようとしています．これも，特定インフラ（トランザクション，データベース，RPCなど）よりのコードをユーザが一切しないで，ビジネスロジックだけに専念できるようにする仕掛け（＝コンテナ）で，画期的なものだと思っています．

岩田　トレーサビリティという尺度だけではシームレスは語れないのも事実でしょう．技術的には他にどんな尺度を持つべきでしょうか？　またトレーサビリテイを保つ技術も，まだビジブルではないようです．UMLなどにはこれを支援しようとする姿勢が見られますが，

われわれは何をすればよいのか？（もっとも，従来の開発では，そもそもシームレスなどは議論の対象にもあがらなかった要件ですが….）

羽田　洗練は，抽象度の異なる二つのモデルの対応関係を示す記法で，トレーサビリティに関連するのは確かだと，思います．さきほどの日経エレクトロニクスの記事の中では，UMLにもある“洗練”がモデルに対するどのような見方に基づくかを明らかにするのは，今後の課題である，といっています．

4. モデラの立場

会田　モデル化についての話ですが，私は一番興味を持つのは実はこの分野です．現実の社会をそのまま EDP で実現できるわけではないですから，システムデザインは，データとプロセスを表現するためにある程度の制約をはめて，実世界の写像をつくることだと思っています．自分の発想は DOA 的といいましたが，経験上論理レベルでのテーブルが一つ増えると，システムの複雑さは加算的というよりも乗数的に複雑さが増すと思っています．その為，ビジネス分析の取っ掛かりとして，まずデータの関係でどのくらいビジネスルールが吸収されるか，そして真に必要な論理テーブルは何なのかに注力してきました．当然それでも吸収できないビジネスルールはプロセスとして表現せざるを得ません．

大石　私も昔，DOA（ER）と RDB を武器にモデル屋気取りで仕事をしていた時期がありました．OO が出てきて，どう消化してよいか戸惑った経験があります．OO は，DOA に継承（is_a）関係と N 項関係，そしてメソッドを追加した拡張 ER モデルだと納得しようとしたこともありました．

古村　一般に，情報処理技術者は実装人間で通っており，モデル作成者としてはほとんど評価されていない．先鋭的に特化された一部を除いては，業務知識に乏しく仕方が無い面がある．といって，そのままで将来も良いという展望には繋がらない．システム作成のプロとして，情報処理の専門家であるためには，モデル作成から実装までを引き受ける集団になるべきだと思う．モデルの立場から見れば，業務知識はメソッドに置き換えられるから，必ずしもモデル作成者が詳細を知っている必要はない．分析者が設計をやるとは限らないし，分析を極めれば設計に行き着くとも考えられない．マシンと独立な核となる部分と，それを運用するマシン依存部分も含んだモデルを，分析段階で考慮しなければならない．このようなモデル作成を行えれば，モデル作成者の仕事が成り立つのでは？

妻木　オブジェクト指向システムの特徴として

・再利用性

・拡張性

の二つがよく挙げられていますが，実際のシステム開発で，どちらの特徴を優先させるべきなのでしょうか．私は，拡張性だと思っています．将来の変化を先取りしたシステムを構築してこそ，技術者としての誇りが持てると思うからです．再利用性というのは，モデリング・ノウハウや独立したソフトウェアの再利用と言うことなら分かりますが，オブジェクト指向で言われている部品の再利用というのは，何か姑息なイメージがつきまとっていて好きになれません．

会田　システム設計の最大の課題は対象システムのスコープを明確にすることと考えていますが，全体の思想の前に，すぐ機能階層図に陥ってしまう自分が情けありません．昨今，モデラーとかシステムアーキテクトとか言われ始めていますが，彼らのミッションが何なのか良い教科書があれば教えてください．

妻木　モデラーと問題領域の専門知識との関係については，考えさせられることが沢山あります．確かに，我々は問題領域の専門家ではありません．それでは，何者かと問われて，情報処理技術者だといって，それが何を意味しているのかよく分かりません．単に，道具の

使い方を多少知っているという程度の意味なのか，問題のこなし方を知っているという意味なのか．モデルにも，オブジェクト・モデルだけでなく統計モデルや数理計画モデル，数値計算モデル，ルール・モデル，述語論理モデル，ニューラルネット・モデルなどいろいろモデルがあります．ある程度のモデリング手法を使いこなせるようになって，はじめて一人前のモデラーと言えるのではないかと思っています．

モデラーのミッションということではありませんが，モデリングとは何かということを考えるのに，先にもあがったJSD法の本［M. Jackson/山﨑利治ほか監訳「システム開発JSD法」，1989，共立出版］は役に立ちました．

司会　多様な論点からの議論が続いていますが，このメール討論をここで閉じたいと思います．この討論でのさまざまな問題提起や意見はオブジェクト指向を取り巻く課題の一端にすぎませんが，これが発端となってさらに情報交換が行われることを期待します．本討論会はその契機としても意義があったと思います．

＊1　原子炉運転員モデルをOOでモデル化し実装した実例で，詳細は技報41号に掲載されている．
＊2　業務イベント単位に業務シナリオを抽象化するビジネスモデリング．