オブジェクト指向技術の実践適用

An Application of Practice of Object-Oriented Technology

石 田 政 海

要 約 オブジェクト指向技術はソフトウエアの生産技術として現在最も有効な技術である．

オブジェクト指向がまだまだ特別な情報技術であると思われてはいるが，オブジェクト指向という考え方の本質は非常に単純であり，実世界をそのままオブジェクトという概念でモデル表現し，それをそのままコンピュータ世界に写像表現することである．

オブジェクト指向技術は，真の意味で，ソフトウエア部品によってアプリケーションを構築する部品による製造技術だと言える．

本稿では，オブジェクト指向技術の本質と技術導入するポイント，円滑に導入するための情報モデルアプローチを紹介する．

Abstract Object-oriented technology is the most effective technology as the production engineering of software at present time. Though the object-oriented technology is considered to be information technology for an expert still more, the essence of concept of“ object-oriented ”is very simple.

The“ object-oriented ”is the concept in which system or component in the real world is expressed as a model in terms of objects and connections between those object, and a model representing objects and their connections are mapped onto the computer system. It may be said that object-oriented technology is production technology by parts assembly where an application is built software component (that is, parts) in true meaning.

This report describes the essence of object-oriented technology and technological know-how of its introduction, and information model approach to smooth introduction.

1. は じ め に

現在，オブジェクト指向技術が急速に普及して注目されている．C＋＋や Java という開発言語だけではなくフレームワークやアーキテクチャにもオブジェクト指向技術が取り入れられている．急速に普及した Windows 環境で提供されているアプリケーションの機能が向上し，品質も安定している状況を見ることができるが，このようなアプリケーションの開発にオブジェクト指向技術が導入され寄与していることはあまり知られてはいない．一方，金融機関の勘定系をはじめとする基幹系システムにもオブジェクト指向技術が適用されつつある．

しかし多くの基幹系システムや大規模システムの開発現場では，オブジェクト指向技術の効果を認識してはいるものの，実際のシステム適用には躊躇している場合が多い．オブジェクト指向言語を用いて開発しているにもかかわらず期待される効果を得られないことから，オブジェクト指向技術に懐疑的な意見もある．

情報処理試験のテキストの一部では，オブジェクト指向技術が小規模システムに適合している，と指摘しているものもあるが，オブジェクト指向技術は規模の大小に左右されるものではなく，複雑な対象を整理できる特徴を備えているため，基幹系シス

テムまたは大規模システムにこそ効果が高いソフトウエア生産技術と言える．小規模対象に向いていると評価されている理由としては，オブジェクト指向言語による開発実績がミドルウエアやクライアントアプリケーションのようなシステムよりのソフトウエアに集中していて要員スキルがあらかじめ備わっていたか，または小規模システムの場合であれば万一リスクが顕在化しても回復が容易である事が考えられる．

IT 投資効果として期待されているのは，サーバーアプリケーションと呼ばれる基幹系システムの拡張や再構築における生産性，品質の向上や迅速な対応であり，それらを実現するためのキーテクノロジとしてのオブジェクト指向技術を位置づけることができる．

本稿では，ソフトウエア開発におけるオブジェクト指向技術の本質と有効性，技術導入の考え方，円滑に導入するために開発した「情報モデルアプローチ」の手順を紹介する．対象読者としては基幹系システムや大規模システムへオブジェクト指向技術の適用を考慮しているプロジェクトマネージャやアーキテクトを想定している．

2. 現状と課題

オブジェクト指向技術を実際の開発に適用しようとする際の課題を列挙する．

・業務モデルは作成できるが実装イメージがわからない
・既存の開発技法との効果の違いがわからない
・概念的で具体的イメージがつかめない
・継承や多態性が難しい
・オブジェクト導出の基準がわからない
・ソフトウエアの部品化ならすでに関数として実現している
・UML（Unified Model Language）など表記法が複雑で扱いづらい
・標準となる開発方法論がない
・スパイラルを止める基準がわからない
・プロジェクト管理方法がわからない

要するにオブジェクト指向技術が誤解され，適用を困難にしている主な理由は

・オブジェクト指向技術を実務適用するための方法論や標準が存在しないこと
・大規模または基幹系システムへの適用事例がないこと，要員が少ないこと

となる．

これはオブジェクト指向技術に特有なことではなく，新技術を導入する場合は共通する課題である．ソフトウエア生産技術としてオブジェクト指向技術を導入する場合には，技術を生産道具として使いこなすために技術の特徴や効果を理解しておかなければならない．その上で分析から設計に展開する手順や成果物，さらにプログラムとしての構造や動作の理解までが必須となるのである．

現在のオブジェクト指向技術が複雑だと思える主たる原因は，オブジェクト指向技術の本質を考察せずに，オブジェクト指向言語が提供する言語要素に振り回されているからだと考えることができる．

継承や多態性の実現に固執するよりは，オブジェクト指向分析によってモデルを整理し，役割と責任を明確にすることによって冗長性のないアプリケーション構造にす

るだけでも適用効果は期待できるのである．

新技術としてオブジェクト指向技術を導入する際には留意しておかなければならないポイントが三つある．

1）　オブジェクト指向技術の本質

オブジェクト指向技術は OOP（Object Oriented Programming：オブジェクト指向言語 Smalltalk，C＋＋，Java などが有名）が主導的な役割を果たしてきたため，言語の持つ機能がそのままオブジェクト指向概念として広く説明されている．オブジェクト指向言語の機能比較においても，カプセル化，継承，多態性，動的結合といった機構を提供しているかどうかでオブジェクト指向表現の充実度合いを主張している．しかしオブジェクト指向技術は新しいものの見方や考え方と説明されており，開発に使用される言語に影響を受けるものではない．オブジェクト指向の「心」すなわち本質的な特徴や効果を理解した上で，実装技術である言語機能を活用する必要がある．

2）　コンピュータ上でのデータ構造と動作

分析における成果物は最終成果物ではない．さらに設計を行い，コンピュータシステム上で稼働させなければならない．最終成果物であるプログラムを作成するだけではなく評価する観点からも，分析や設計工程で定義された成果物要素がコンピュータ上でのどの要素に対応するのか，データ構造と動作を理解する必要がある．このため分析設計技術者もオブジェクト指向言語を経験しておく必要がある．

3）　適用指針と効果目標

オブジェクト指向技術を導入すること自体が目的ではない．ビジネス要求の観点から重要なのは実装技術や生産技術ではなく，要求した機能が定められたコストと期間で実現できることである．基幹系システムや大規模システムにオブジェクト指向技術を適用する場合には，理想ではなく現実的な技術導入レベルを明確にすることが重要である．オブジェクト指向技術のどの技術要素によって生産性や品質の向上といった効果を実現するのかを確認する必要がある．

以上の三点をバランスよく組み合わせることが重要となる．オブジェクト指向技術は分析だけに効果があるとかプログラミングにのみ効果が期待されているわけではない．特にオブジェクト指向言語に偏重した考え方は技術適用の阻害要因となる．

3.　オブジェクト指向技術の本質

オブジェクト指向の考え方は，1967 年に開発された Simula 67 が最初だと言われている．その後 1980 年代初頭に Smalltalk，後期に C＋＋が開発されている．オブジェクト指向の定義として J. Rumbaugh は，「データ構造と振る舞いが一体となったオブジェクトの集まりとしてソフトウエアを組織化すること」としている．オブジェクト指向技術は，言語が先に提供されたためにオブジェクト指向言語の言語要素から特徴づけられている[1][2]．

・カプセル化

・クラス化

・継承（および多態性）

オブジェクト指向言語を議論する場合には，これらの特徴を実装しているかどうかが論点となっている．後発した開発方法論も，下流工程ではオブジェクト指向言語による実装を想定としているため，モデル構成要素や表記法がこれらの特徴を満たしている．さらにC＋＋やJavaはより表現力を強化するために言語仕様を進化させ複雑化している．例えば，多重継承やパラメタライズドクラス，独立したインタフェース概念などの言語要素である．その結果開発方法論や表記法も言語仕様にあらわれた言語要素をオブジェクト指向の新たな特徴と見なして対応している．

3.1 オブジェクト指向技術の捉え方

端的に言えば，継承や多態性，動的結合，インタフェースの独立という考え方はオブジェクト指向言語による拡張であって，オブジェクト指向の本質的な特徴ではない．Simula言語がシミュレーション言語として活用され，実世界の模倣をコンピュータシステムで実現することを目的としたが，この基本的な考え方がSmalltalkに影響を与えて，さらに現在のオブジェクト指向言語へ受け継がれている．

またオブジェクト指向のモデルは工程間でシームレスであるという特徴がある．意味は，実世界を表現するモデルとコンピュータ世界でのモデルの表現にオブジェクトという共通のモデル要素を導入することによって，モデル間の変換作業を最小限に抑えることができる，ということである．

つまりオブジェクト指向の本質とは，実世界とコンピュータ世界を写像関係にとらえることにあると言える（図1）．

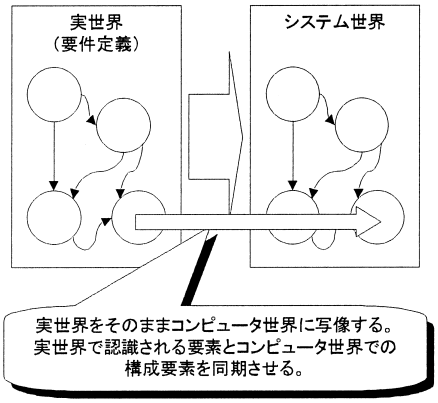


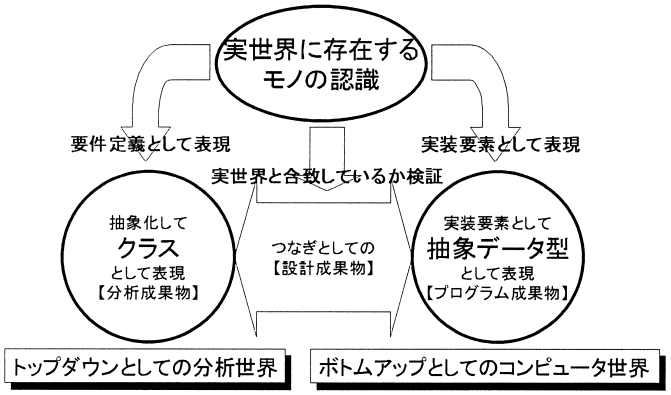
図 1　実世界とシステム世界との写像関係

プロセス指向やデータ指向のように，処理とデータの分離を前提としていない．前提としていないのだから処理とデータを再度カプセル化するという意味も本来は含んではいない．関係のない処理とデータをカプセル化することが目的ではなく，実世界に存在するものは元々カプセル化されていると捉える．実世界では認識されない，または存在しない仕掛けを作り出すのではなく，実世界をあるがまま抽象化してコンピュータの中に表現することを目的としている．

3.2 抽象化とクラス

抽象化は個（オブジェクト）と個を類型化した類（クラス）を単位とすることで実現する．現在のオブジェクト指向は，オブジェクト指向(＝オブジェクト単位で表現)ではなく意味的にはクラス指向（＝クラス単位で表現）にほかならない．個々のオブジェクトをグループ化・抽象化することで実世界をクラスとして表現している．実世界にはクラスに相当する実体は実在しないが，無数に存在する個々のオブジェクトをそのままコンピューター世界に表現する本来のオブジェクト指向ではなく，制限はあるものの一つの実現解としてクラス指向を採用している．

一方，プログラム言語の型理論では抽象データ型概念が導出されている．簡単に言えば利用者定義型である．一般的には，抽象データ型はプログラミング言語の視点から考えて最高のものといえる，と評価されている．抽象データ型は，オブジェクト指向言語以外でも実装可能な概念である．



・実世界を抽象化したものが【クラス】
・実装するための技術要素が【抽象データ型】

図 2　クラスと抽象データ型の関係

このクラス概念と抽象データ型が合成・揚棄することによって，分析成果物であるオブジェクトモデルが設計や実装におけるオブジェクトモデルへとシームレスに写像されることを保証している．

クラス概念と抽象データ型を結びつける視点や媒介は，データ構造やインタフェースではなく，実世界で認識される概念が重要となる．整数を例にすると，整数の内部フォーマットや加算するという意味の＋というインタフェースが重要なのではなく，整数という概念を表現したクラスと，データ型としての整数型とを同一のものであると認識することが重要である（図 2）[3][4]．

3.3 カプセル化と情報隠蔽

カプセル化とは，処理とデータ構造を一体化した場合に実装にあたるデータ構造を隠蔽することである．この考え方も手続き型言語のモジュールとして実現可能なため，オブジェクト指向に固有な特徴ではない．データ指向と区別するための特徴としては，メソッドすなわち振る舞いが重要であり，実装であるデータ構造を考慮する必要は無

いことを示している．

3.4 継　　承

継承は，抽象化されたクラスをさらにグループ化することで，汎化・特化関係にある下位のクラスが上位クラスのデータ構造や振る舞いを受け継ぐ機構である．この考えもオブジェクト指向の本質ではなく，オブジェクト指向言語の機構であり，開発環境においてクラス定義の負荷を軽減する差分プログラミングの特徴である．ビジネス世界には，継承に相当する概念が存在しないことが多い．ある金融機関向けに開発された普通預金クラスを再利用して，新たに普通預金サブクラスを導出してもそのまま利用できるのは希であり，カスタマイズする必要が生じる．現在のオブジェクト指向言語では，下位クラスにおいて上位クラスのメソッドをカスタマイズするためには同一名称のメソッドを再定義する遮蔽定義しか方法がなく，効果は期待できない．

3.5 多態性と動的結合

多態性とは，設計時に上位クラスとして定義しておき，実行時に下位クラスを確定させて処理をする機構である．これも継承構造を前提としており，オブジェクト指向言語の機構だと言える．特定のサブクラスに依存しない抽象化された手続きを記述することが可能となる．この機構を利用するためには，サブクラスの振る舞いを共通化させる必要がある．しかし普通預金と定期預金の入出金という動作を共通化できても，必要な引数情報は異なっているため単純には共通化はできない．普通預金と定期預金は独立関係にあり，普通預金の入金に関わる引数情報の変更は，他の金融商品には影響を及ぼさないからである．

3.6 インタフェースの抽象化

GUI やコレクションのような汎用部品などのように，新規に考案するオブジェクトであればインタフェースをあらかじめ共通化しておくことができる．しかしビジネス領域におけるオブジェクトの場合には，工業製品のように規格化されるものではないし，ビジネス世界にはインタフェースを共通化するという考え方がない．普通預金の入金と定期預金の入金は操作名（インタフェース名）が同じではあるが詳細には全く異なるものである．実世界では別物であるインタフェースを共通化するのは，オブジェクト指向の本質つまり実世界をそのまま写像する考え方に合致しない．またビジネス世界におけるオブジェクトのインタフェースは時間とともに変化するものであり，その変化をそのままコンピュータ世界に写像・実現することが重要である．

インタフェースを共通化させるという意味での抽象化作業は，オブジェクト指向設計に高コストを要求し，インタフェースの変更要求に対して影響範囲を局所化できないという弊害を生み出す．

3.7 状　　態

通常の手続き型言語や関数には状態を保持する機構が基本的には備わっていないのに対して，オブジェクト内部には状態を保持する機能が本質的に備わっている．実世界に存在するモノは必ず状態を備えているため，コンピューターの中に写像表現するため状態をモデルとして表現する必要がある．既存の勘定系システムでは処理ルーチンに対して共通引数として顧客番号と口座番号を渡しているが，オブジェクト指向技術ではインスタンス生成時に一度だけ渡しておけば，顧客番号と口座番号はインスタ

ンスの状態として保持されているため外部から再度引数として与える必要はない．オブジェクトが状態を保持しているため，メソッドにおける引数設計には注意が必要である．オブジェクトの状態を変更させるために必要な外部データを引数とする．

3.8 ま と め

現在のオブジェクト指向技術はオブジェクト指向言語から多くの影響を受けている．継承や多態性といったオブジェクト指向概念が難解で理解しにくいものであるならば無理に利用する必要はない．

オブジェクト指向技術の本質的意義は，生産性や保守性を向上させる技術として位置づけ，実世界のシステム化対象領域をコンピューターの中へ，オブジェクトという概念を用いて写像する事にある．単純で明快な生産技術だと評価できるのである．

4. オブジェクト指向技術の効果

オブジェクト指向技術は，上流工程（分析から論理設計）では整理術，下流工程（プログラム設計から実装）では利用者定義型による実装技術ととらえることができる．導入することによる期待効果は次の二つとなる．

1) ソフトウエアの構造が実世界と写像関係

ドメイン専門家やエンドユーザの要求をそのままモデル化することによって評価可能な成果物として表現でき，その成果物がソフトウエアの機能として実現されるため，要求と実現機能との間のギャップが少なくなる．実世界では，役割と責任の所在が重複しないように整理されていると考えることができ，ソフトウエアに写像した場合もデータや処理の冗長度を削減することができる．

保守・拡張する際に影響を及ぼすシステム範囲が予測可能となり，SE の恣意的な構造や機能を排除することで余分な頭脳労働を減らすといった効果も期待できる．

2) 抽象データ型によるソフトウエア部品化

整数型や実数型，文字列型などの開発言語が提供している原始データ型だけを組み合わせることでのみアプリケーション構築するのではなく，利用者定義型を自由に作成して組み合わせることで，生産性や品質の向上を実現することができる．状態を持たない関数やサブルーチンでは利用者が状態を意識しなくてはならないが，状態を持っている利用者定義型を利用することで，部品の状態を意識した処理を記述する必要がなくなる．

関数部品との違いは，日付処理に関して日付処理関数と日付型という部品において，日付処理関数の利用者は引数として与える日付値が日付として妥当であることをあらかじめ検査しておくか，処理結果としてエラーが返された場合に相当する処理を記述する必要がある．日付値が日付型変数として保存される場合には初期化時点で日付として妥当であるかどうかの検査は行われるが，その後の演算においては日付が不当であるというエラーが返らないため検査する処理記述が不要となる効果がある．

ソフトウエア部品化は，機能を追加する場合や不具合が発生した場合でも影響範囲を局所化できる．

・要求を分析オブジェクトモデルとして抽象化する…【実世界部品：要件定義】
・分析オブジェクトモデルをコンピューター世界の設計オブジェクトモデルに写像する…【設計】
・設計オブジェクトモデルをオブジェクト指向言語によって実装する…【ソフトウエア部品】

実世界部品である要件定義をそのまま受理する開発言語が存在すれば両者の変換作業は最小となるが，現在のオブジェクト指向言語では変換作業が必要となる．この役割が設計であり，要件定義の表現であるクラス定義とプログラミングレベルでの抽象データ型を結びつける作業となる．

5. オブジェクト指向技術による分析・設計成果物

オブジェクト指向技術における分析とは，要件定義をオブジェクトモデルとして形式化することである．要件定義の成果物として必要なのは，クラス定義とシーケンス図のみである．簡単に言えば，クラス定義とは部品の仕様書であり，シーケンス図とは部品の代表的な利用方法を現している．

5.1 クラス定義

クラス定義では，クラスの役割と責任を明確にする．特にメソッド定義を優先する．DOA（Data Oriented Approach）やデータベース設計とは異なり，主属性や永続属性の設計を優先させない．カプセル化における情報隠蔽の考えによればデータ構造の表現は重要ではないし，ソフトウエア部品としてとらえて見た場合にも内部実装は部品の利用者から見て透過である必要がある．たとえば，実数型の利用者は実数型の使い方，つまり部品の仕様である，有効精度情報や四則演算ができること，左辺値や右辺値として操作できることがわかれば十分であり，実数型の内部構造である仮数部や指数部，正負情報などのビット仕様を知りたいとは考えない．業務用に導出されるソフトウエア部品であるクラスも同様に定義と操作仕様だけが明らかであれば十分である．

5.2 シーケンス図

別名インタラクション図．ソフトウエア部品を組み合わせて要求仕様通りに呼び出し連鎖が動作するか確認するための役割を担う．シーケンス図の目的は次の三つである．

1） すでに導出されているクラスによって業務シナリオを実現できるかどうかの検証

漏れているクラスを発見する契機となる．他のクラスとの呼出連鎖関係からクラスの役割と責任を明確にすることができる．呼出連鎖のパターン化も決定する．

2） クラス定義のうちメソッドを導出して引数を定義する

メソッドの妥当性検証と詳細化としての引数設計を行う．シーケンス図に現れるメッセージはメソッドに対応する．引数は値の発生源からの伝搬を保証しなければならない．またメソッドはクラスに対するメソッドとインスタンスに対するメソッドを区別する必要があるが，分析段階では個を特定した処理と特定しない処理との区別でよい．

3)　ソフトウエア部品同士を結合させた場合のテスト仕様

シーケンス図は完成したソフトウエア部品を結合させることによって予定通り業務機能が実現できるかどうかを検証するための設計図であり，結合テスト仕様を作成する際の入力成果物として位置づけることができる．

シーケンス図は，例えば金融業務における入金とか出金といった業務取引パターンに対応するものである．引数の主なものとしては整数や実数，文字列型といった原始データ型の変数や構造体型変数，可変要素を格納できるコレクションクラスとしてのリスト型変数および利用者定義型としてのオブジェクトがある．特に実世界では要素数を固定しない場合が普通であるため，固定要素しか表現できない配列は使用せずにリスト型を導入する．要件定義に相当する分析段階においては，リスト型という概念は存在しないが，可変要素を明確に表現する概念が無いこと，および次工程である設計段階でプロトタイプ実装を円滑にすすめる点から導入している．

クラス定義は成果物を静的に把握するものであり，シーケンス図は動的に把握するのに便利である．両者を相互に検証しながら設計成果物の精度をあげていくことになる．

5.3　実践的適用するためのポイント

ソフトウエア部品としてのクラスの導出方法には三つのアプローチ視点がある．

・企業が再利用するために保存しておかなければならない情報という視点（トップダウン）
・企業の組織設計と同じ視点（トップダウン）
・プログラミングにおいて効果のあるソフトウエア部品としての視点（ボトムアップ）

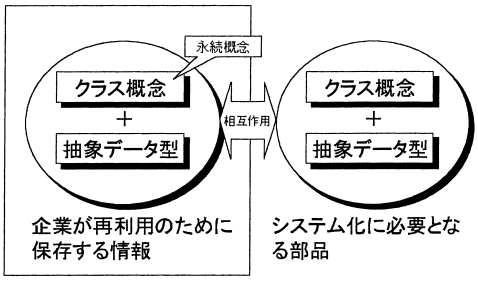


図 3　情報，クラス概念，抽象データ型の関係

1)　オブジェクト指向の方法論ではオブジェクトを導出することが困難であるとか，まず名詞を抽出するという記述があるが，ビジネス領域に適用する場合には，企業が再利用のために保存しておく情報，に着眼する．情報と言っても文書バインダーやキャビネットという情報そのものを現しているのではなく，情報を主管する役割と責任を意味している．情報に関わるビジネスロジックは情報そのものが保持していると考える．5.1 章で述べた通り，具体的なデータベースをイメー

ジした主属性定義を指してはいない．抽象データ型の観点から，主属性は実装における内部表現でしかないため重要ではない．しかし実世界をモデル化したクラス概念には永続概念が本質的に内在しているため，システム化においては抽象データ型の内部に永続化処理を実装する必要性が生じる（図 3）．

2） CRM（Customer Relationship Management）などでは，通常の取引情報やコールセンターなどからの接触情報を採取・保存しておかなければ有効な顧客情報として再利用できない．通常のコンピューター処理空間では取引，商品，決済，顧客情報がある意味づけのもとに集合して，処理空間が消失するときにそれらの情報も散逸してしまう．情報そのものと情報同士の関係付けは，採取しておかなければ情報欠損してしまう．採取しておくべき情報，という判断がオブジェクト導出の判断指標となる．

情報という判断指標がわかりにくければ，企業の組織設計という視点が有効である．ビジネスの世界をシステム化するということは，コンピュータが存在しないと仮定した場合，求められるミッションを遂行するための組織が必要となる．顧客クラスというのは顧客情報を主管する部署とみなすことができる．どのような役割や責任も企業の組織として設計することができる．この考え方によって，組織をクラスに，その組織の役割や業務をメソッドに対応づける．役割と責任を持つものをオブジェクトとして導出することができるのである．

3） ボトムアップ的なアプローチとしては，ソフトウエア部品の視点から考察することも効果がある．導出されるオブジェクトは最終成果物としてプログラムとして表現されなければならない．企業の情報システム部門はエンドユーザーから機能の追加，新制度や新商品への対応，画面や帳票の作成要求を受けることとなる．情報システム部門がいろいろな要求に対して即座に対応できることが求められている．つまり情報システム部門であるプログラム設計者にとって画面や帳票を容易に作成することができるソフトウエア部品という視点でオブジェクトを導出することができる．

4） その他

① 結合リスクをヘッジするためのプロトタイプ

オブジェクト指向技術によってソフトウエア開発を行う場合，同じ抽象度でクラス定義とシーケンス図が作成されていれば，結合動作を先行して検証することができる．確認するポイントは，オブジェクトとメソッドの呼び出し連鎖と引数の伝搬である．検証のため各メソッドの開始点と終了点に通過イベントと引数の内容を証跡情報として採取するための処理ロジックを実装するだけでよく，メソッドの詳細処理を実装する必要はない．クラス定義とシーケンス図が仕様通りに動作することを確認した後に，メソッドの処理仕様を実装する進化型プロトタイプが効果的である．技術リスクヘッジも目的とするため，本番システムが想定する基盤技術を組み入れておくことも可能である．

② 要員教育と開発体制

オブジェクト指向技術で開発するためには上流モデリングから下流のプログラミングまでの成果物と開発プロセスを理解しておくことが必要である．特に

オブジェクト指向言語を経験してオブジェクト指向言語の言語的な感覚やコンピュータのデータ構造と振る舞いを理解しておく必要がある．

オブジェクト指向技術を未導入である場合には，初期学習コストがかかる．導出したオブジェクトが妥当であるかの評価も重要であるため，オブジェクト指向技術による開発経験者の参加が必要である．大規模開発でチーム分割した場合，チームごとにクラス定義の抽象度や完成度合いにばらつきが出る可能性が高いため，クラス設計成果物を検証・評価するチームの準備も必須である．

6. 情報モデルアプローチの手順

生産技術として非常に効果が期待できるオブジェクト指向技術であるが，本稿の最初で提示した課題にもあるように情報システム部門などの現場ではオブジェクト指向技術に対して理解しにくいというイメージがあり，技術導入に対する抵抗値となっている．一つの試みとしてオブジェクト指向技術を前面に出さずに，企業が保存しておく情報と情報構造に着眼した，「情報モデルアプローチ」を開発して，分析・設計・プロトタイプ作成に適用してみたところ，その分析・設計・実装の作業を経てオブジェクト指向技術の本質を抵抗感無く理解するという実績を得た．

情報モデルアプローチでは，クラスを情報，クラス図を情報構造に置き換えている．基本的な考え方はオブジェクト指向技術に基づいている．実装はオブジェクト指向言語が理想ではあるが，非オブジェクト指向言語であっても考え方は有効である．表記法としては UML のクラス定義とシーケンス図を使用するが，継承関係はモデルの説明のためにだけ導出している．

情報モデルアプローチでは成果物を定義して，その成果物から検証用プロトタイプを作成することにより具体的に実証することを目的としている．複雑さを軽減するため使用するオブジェクト指向概念や表記法を，最終成果物として必須となるものにおさえている．

6.1 情報モデルアプローチ概要

1) システム化対象領域にある要求を，企業の再利用のために保存しておく情報に着目して，要件定義モデルとして整理・表現する．
2) 要件定義モデルは二つの成果物から成り立つ．
   - 情報と情報構造
   - 業務フローシーケンス図
3) 要求は要件定義として写像表現する
4) 実装は要件定義を写像実装する

情報モデルアプローチの目的は，要件定義モデルにオブジェクト指向を適用することによって要求とシステム構造を写像関係として実現することである．高生産性言語や自動生成による生産性の向上を図るのではなく，変化する要求とそれにともなうシステム構造への変更の見通しを明確にできることにある（図 4）．

6.2 対象領域の決定

分析対象となる領域（ドメイン）を仮置きする．分析作業が発散しないため，どこから始めるかを決定する．業務の中核部分や複雑な部分，分析作業参加者が一番関心

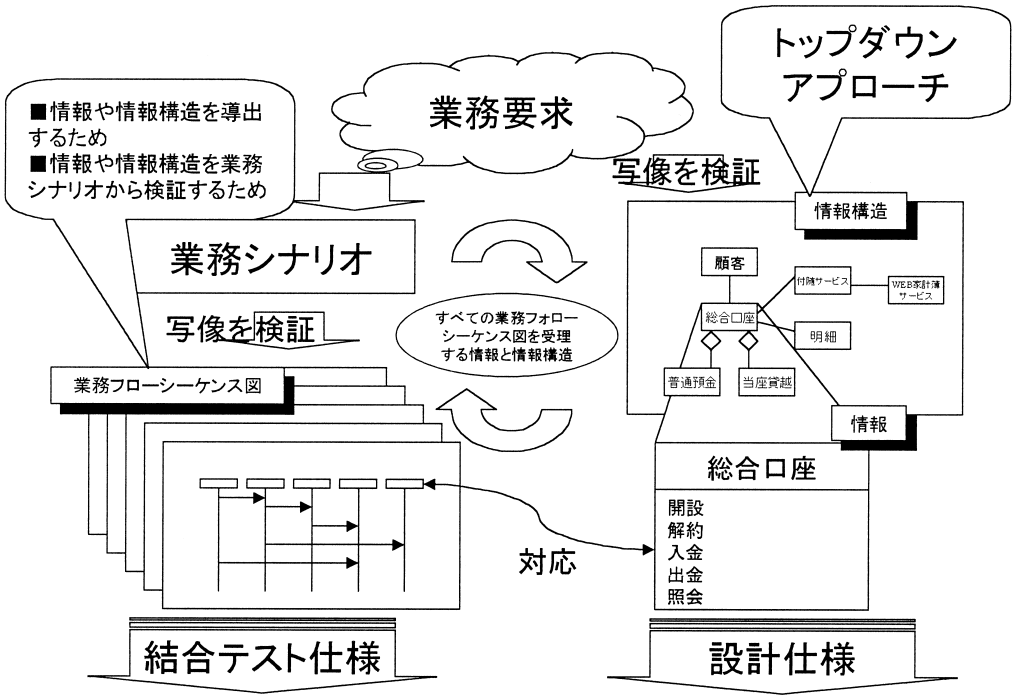


図 4　情報モデルアプローチの要件定義モデル

を持っているところを開始点にする．

6.3　業務フロー候補の抽出

対象領域が決定したら，その領域の主要な業務フローを抽出する．例えば銀行業務での総合口座開設から入金処理．目的は業務フロー遂行に必要な情報候補を洗い出すためである．情報抽出が第一目標であるため業務処理の詳細には立ち入らない．

6.4　情報候補の抽出

抽出された業務フローから情報候補を抽出する．再利用を目的として保存しておく価値のあるものが情報候補となる．単に名詞としてあらわれるものは情報でない場合が多い．また既存システムのデータベースを参考にしても良いが必ず再評価する．たとえば物理データベースや論理データベースには処理効率を実現するために非正規化されている場合があるためである．導出するときはシステム的なデータモデルを導出するというよりはビジネスからみた情報という観点が重要である．対象領域世界で認識できる概念と名称をそのまま情報に対応させる．

ビジネス領域においては上記情報という抽出基準が有効である．その後シーケンス図の作成において情報を補完するオブジェクトが導出される．継承や多態性概念は分析レベルでは必須ではない．

6.5　情報の役割と責任の明確化

抽出された情報候補の役割と責任を明確にする．役割としては，情報に識別子となる名称をつけ定義付けを行う．その際，業務領域に現れる用語や単語を選ぶ．情報が持つべきサービスを明らかにする．情報の責任範囲を判断できる基準を記述する．

6.6 シーケンス図の作成

業務要件の機能の動的構造を記述する．これは情報と情報構造を導出するための入力成果物となる．業務シナリオからシーケンス図を作成して，その業務シナリオを実現するための情報と情報の持つサービスを決定する．データの流れと制御の流れを明確にする．情報の生成や消滅のタイミングや責任を明確にする．

シーケンス図はプロトタイプ検証シナリオや結合テスト仕様の元ネタとなる．

シーケンス図の記述精度は情報や情報構造の抽象度に合わせる．

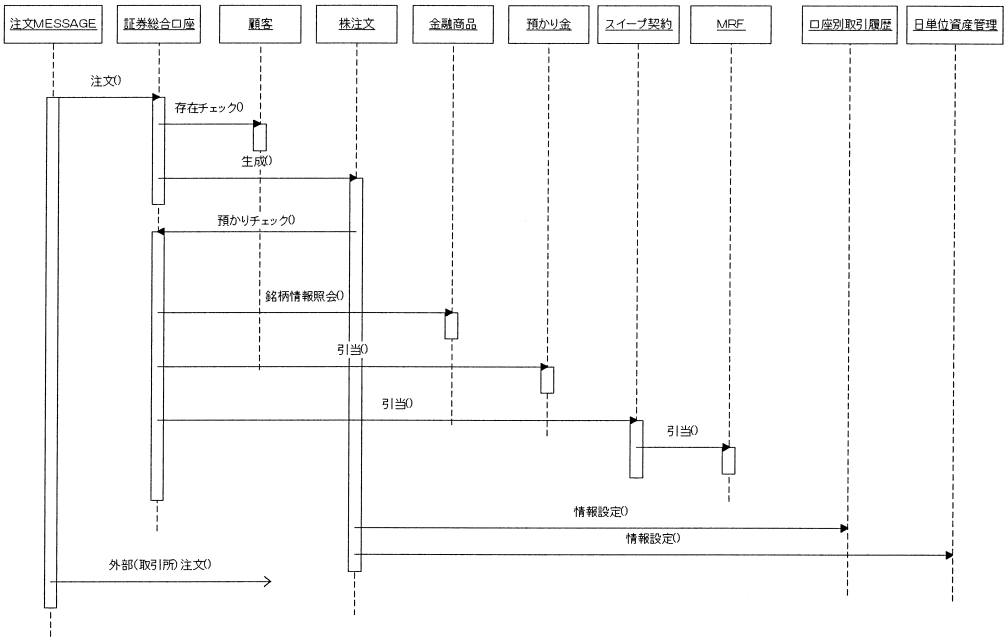


図 5　シーケンス図（国内株式注文）

実装するためには，システム制御構造，アプリケーション制御構造およびアプリケーション構造を定義しなければならない．業務モデルはアプリケーション構造に対応している．業務部品としてのソフトウエア部品は，それを再利用するプログラムが存在することと暗示している．図 5 中の注文 MESSAGE がアプリケーション制御に相当するものであり，入力および応答のユーザーインタフェース処理，出力帳票，トランザクション成立可否判断を行う．

6.7 情報定義の洗練化

仮決定されている情報の仕様を，主属性レベルを目安に詳細化する．詳細化するのは

- 主属性
- サービス
- 引数

である．

主属性とは，情報が保存するデータ項目で，永続化対象項目が有力な候補となる．情報の定義（役割と責任）から連想できるデータ項目も候補となる．

サービスは，情報の生成と消滅，生成から消滅までの間で大きな状態変化を引き起こす契機に着眼する．状態変化だけではなく状態照会も対象となる．

引数は，サービスを遂行するために情報の外部から与えられるものと考える．引数から主属性を連想することができるので，主属性，引数，サービスの抽象度を合わせることが重要である．主属性の導出よりはサービスと引数の洗練化を優先させる．

6.8 情報と情報構造図の洗練

鳥瞰図としての情報構造図はクラス図に相当する．情報構造図の目的は業務対象領域の概要を表現することである．構成する情報の抽象度や粒度をそろえる．主要な情報は中央に配置し，関連線は主要な情報を起点にして重要なものに限定して記述する（図 6）．

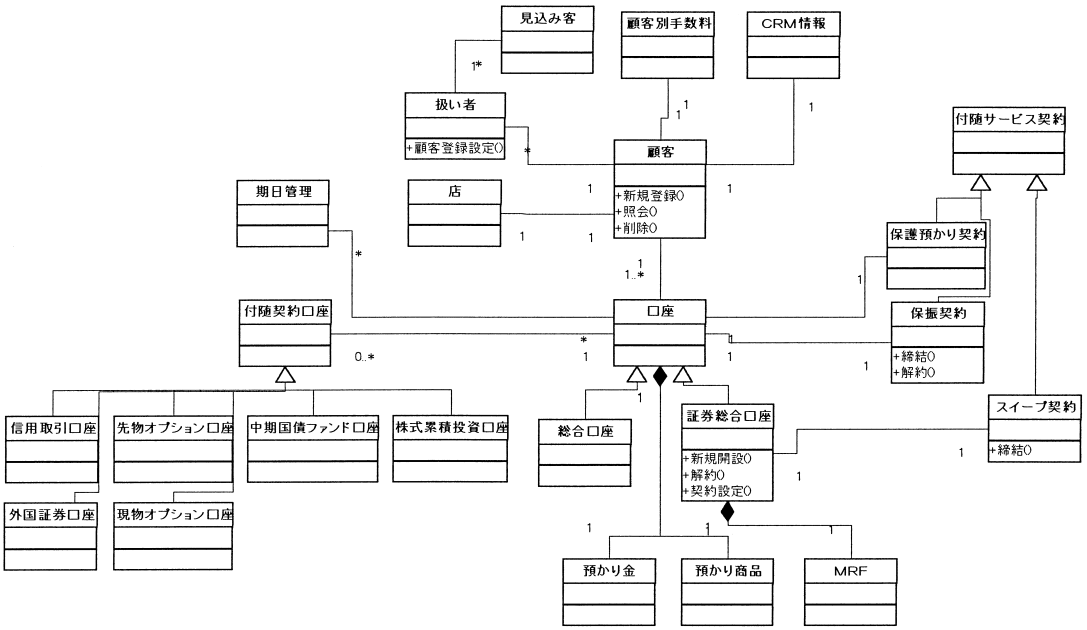


図 6　情報構造図［クラス図］（証券総合口座）

6.9 プロトタイプ

プロトタイプ対象となるコアシナリオを選定する．Inquiry/Answer 電文処理のように入力から出力までに及ぶトランザクション処理や業務領域で頻度の高い業務フローを選定する．

プロトタイプの目的は

- 情報モデルアプローチで作成された成果物によってソフトウエアが設計通り稼働するかの検証
- 情報と情報のメッセージ連鎖を検証
- 選定された IT 技術を検証して，技術的課題の早期洗いだし

コアシナリオの実装方針としては，情報の外形のみを実装しビジネスロジックの実

装は目的ではない．情報のサービスには検証するためのトレース機能を実装する．トレース対象は，サービスの呼び出しイベント，入力引数の名前と値，出力引数の名前と値および戻り値である．トレースの結果から業務フローシーケンス図の呼び出しシーケンス図を実現しているか検証する．先行的にIT技術要素を組み込み，課題が発生した場合には対応策の検討を行う．

6.10 ま と め

1) 1情報（クラス定義）と情報構造（クラス図）が重要

1情報と情報構造とは，実世界とシステムとの1対1写像関係を重視した表現方法である．

2) 業務要求を形式化するため要件定義モデルを導出

要件定義モデルとは要求の表現形式で，情報，情報構造，業務フローシーケンス図からなる．要件定義モデルはオブジェクト指向言語を導入することでモデル変換を最小限におさえて実現することができる．

3) プロトタイプ技法

主要な業務取引の正常処理から開始するメインラインアプローチ手法によって，情報間の呼び出し制御動作を確認する．またIT技術リスクヘッジも検証する事が可能となる．

4) アプリケーション層との分離

中核となる情報の処理とその情報の表現である画面や帳票，データ変換，デー

表 1　開発手順と成果物

作業番号	作業	目的および作業	成果物
1	対象領域決定	モデル作業が発散しないためにおおまかな範囲を決定する	対象領域
2	業務取引候補の抽出	実際の業務取引に着眼して情報を導出するため	業務取引の概要
3	情報候補の抽出	企業が再利用するために保存しておく情報を明確にする	情報の候補 （コンポーネント、クラス、エンティティに相当） 概念DBのイメージに近い
4	情報の役割と責任の明確化	情報を正規化するために、その情報の役割と責任を定義する	情報の役割と責任 情報の定義（説明書き）
5	情報間の関係を明確化	情報が他の情報を参照したり依存したりしていないか、包含していないかとか、ある情報から別の情報にをたぐり寄せることができる構造になっているかどうかを計画にする Information Schema図は全体領域の鳥瞰図としての意味がある ★企業が情報をどのように整理しているかという情報構造も表現している	Information Schema図 (=情報構造図)
6	シーケンス図の作成	業務取引1つ1つからそれぞれ複数の情報への作用連鎖関係を表すシーケンス図を作成する。 ★このシーケンス図がプロトタイプのシナリオとなり、また結合テスト仕様となる。情報の実装が要求を満たしているかどうかのレビューにも使用する。	シーケンス図
7	主属性レベルへの洗練化	情報から情報への呼出連鎖に必要な主要な引数と、情報が保存すべき主属性の詳細化 プロトタイプで実装する精度となるため、検証したい属性を導出しておく	情報の定義 シーケンス図
8	シーケンス図から情報定義の検証	業務要求は業務シナリオのシーケンス図に変換されている。ここまでの工程で作成してきたすべての情報がすべてのシーケンス図を受理すれば、情報の定義は要求を満たしていることとなる。	情報の定義 シーケンス図
9	Information Schemaの洗練	8までの作業で情報に追加があったり情報間の関係に変更があった場合にはInformation Schema図を修正する この作業は並行で行ってもよい	Information Schema図 (=情報構造図)
10	プロトタイプ（コアシナリオの選定および実装）	プロトタイプはトンネルエ法に似ている。最初は細く穴を掘りその後徐々に広げる。このためエラーケースを含まないメインシナリオを選定する、そのメインシナリオが使用する（＝通過する）情報を抽出して通過する処理のみを実装、シーケンス図通りに動作するか検証する ★コアシナリオは重要な業務取引が情報の定義を実装するだけで稼働するかどうかを確認することが目的となる	プロトタイプ（コアシナリオ呼出連鎖確認） 情報の実装 コアシナリオを駆動させるメインプログラム
11	情報の実装	コアシナリオで実装されなかった情報を定義に基づいて実装する	情報の実装（すべて）
12	プロトタイプとシナリオの検証	残されたすべてのシナリオに対応するシーケンス図通りに動作するか検証する すべてが検証できた段階で、このプロトタイプは業務要求を主属性という抽象度で満たしていることが証明宇できることとなる	ある抽象度でのプロトタイプ実装完成（すべて）
13	本番システムへ	業務シナリオが詳細化される（エラーケースなどや特殊な取引パターンなど）、主属性という制限が無くなり全属性が対象となる 作成したプロトタイプを詳細化することが本番システムを作成することとなる	

タ入出力，二次加工処理とを分離することによって保守性・拡張性の向上が可能となる．

現在のプロセス指向開発や手続型言語を前提とする開発現場においても規模や特性に応じた開発手順と成果物が定義されている．オブジェクト指向技術は従来の開発方法やプロジェクト管理技術を否定するものではないため，オブジェクト指向技術の本質と効果を正しく捉えて実務適用することは難しい事ではない．情報モデルアプローチは一つの参考事例である（表 1）．

7. おわりに

オブジェクト指向技術を実現するための環境は整いつつある．ISO では C＋＋や JAVA の成果を受けて次世代 COBOL の標準化もオブジェクト指向言語要素を取り入れ OOCOBOL として標準化作業も終了間近である．認知度の高い Microsoft 社の Visual Basic の次期バージョンもオブジェクト指向言語となる．さらに基幹系システムへの適用期待も高く，技術検証はもちろん金融機関の勘定系システムにも適用が開始されている．

従来の構造化技法や手続き型言語によるシステム開発には限界が来ている．ビジネス構造の変革に即応する情報システムは劇的な生産性や品質の向上が求められており，オブジェクト指向技術はキーテクノロジーとして認知されていると言って良い．オブジェクト指向技術は小規模システムではなく大規模，基幹系システムにこそ適用効果を最大限に発揮する生産技術である．オブジェクト指向技術の本質は単純かつ明瞭であり難しいものではない．実世界をそのままオブジェクトで表現し，オブジェクト指向言語を採用することで利用者定義型と呼ばれるソフトウエア部品を実装して，それらを組み合わせることでアプリケーションを構築するだけなのである．

参考文献
［1］ James Rumbaugh, 羽生田栄一監訳, オブジェクト指向方法論 OMT, 1992 年 7 月.
［2］ B. Henderson-Sellers, 大森健児訳, オブジェクト指向ソフトウエア工学―分析・設計・実現―, 海文堂, 1993 年 1 月.
［3］ Timothy A. Buddt, 羽部正義訳, オブジェクト指向プログラミング入門, アジソン・ウェスレイ・トッパン, 1992 年 10 月.
［4］ J. C. Cleaveland, 小林光夫訳, データ型序説, 共立出版株式会社, 1990 年 4 月.

執筆者紹介 石 田 政 海（Masami Ishida）
1985 年立命館大学法学部卒業．同年日本ユニシス(株)入社．統合 OA システム開発，SWIFT CBT，SYSTEM-F 制御系開発，オブジェクト指向技術適用の企画推進，SBI 21 PC 開発環境の開発に従事．現在，E ビジネス技術部コンポーネント技術室所属．