平成 14 年度

ビジネスオブジェクト検討プロジェクト

「ユーザが考える
ビジネスオブジェクト調査報告書」

平成 15 年 4 月

社団法人　日本情報システム・ユーザー協会

【目次】

1. はじめに

ビジネスオブジェクト検討プロジェクトが発足した目的は、本論の中で述べているように、オブジェクト指向技術をビジネスの分野で活用しようとする動きが活性化してきており、これがベンダからの提供にとどめることなく、ユーザが活用することで、コストの削減をはじめとして経営戦略を支援する情報化戦略に役立てようとするもので、その為にはユーザの立場からビジネスオブジェクトのあり方や利用方法について発言する必要があるとの認識から出発している。

一般論として、オブジェクト指向技術は難解とされている。書肆に揃えられている関連の書籍にしても、わかりやすさを謳った標題とは裏腹に、内容は難しい。執筆者がベンダの人であったり、大学の先生、洋書の翻訳であったりすることが一因となっているのだろう。まだ今のところユーザの立場から書かれたものはないようだ。この意味においては、当研究報告書がユーザの立場から書かれた最初の一冊になるのだろう。

ビジネスオブジェクトの役割は、経営工学と情報工学の融合にある。経営のモデルをオブジェクトで表現することにより、システム開発にシームレスに持ち込むことができる。さらに、業種や企業によってはモデルを共有化することで、コストの削減も期待できる。市場の変化に迅速に対応したシステム開発もできるようになる。さらには、レガシーシステムとの連携も技術的には可能性がある。このように、これまで企業の情報システム担当者が抱えていた問題を一気に解決できる可能性を秘めている。

ただし、オブジェクト思考技術は今のところ難解なので、広く普及するには時間を要すだろう。しかし、良いものは時間がかかっても広く受け入れられるもので、ビジネスオブジェクトがこれからの企業経営と情報システムに貢献できることがはっきりすれば、自ずとわかりやすいものに変化してゆくのだろう。また、それを促進するのが本委員会の本来の目的である。

本年度は初年度にあたるため、技術の調査と可能性について検討することに大半の時間を費やした。本報告書はユーザの立場から、ユーザの理解を深める為に作られているので、ビジネスオブジェクトについて綜合的に理解するには最適な書物になったと自負する。

最後に、このような研究の機会を与えてくれた日本情報システム・ユーザー協会に感謝申し上げる。

ビジネスオブジェクト検討プロジェクト
委員長　　福田　修

2.　調査目的と調査方法

2.1 背景と目的

2.1.1ビジネスオブジェクトの重要性の増大と課題

2.1.1.1　企業のかかえる問題点

オブジェクト指向技術が登場した 1980 年代初頭は、日本経済が高度成長を終え、安定成長期の中間に位置していた時期になる。

日本企業の多くは、積極的な情報化投資を行い、これまでの生産技術を向上させるためのコンピュータ投資から、事務の合理化と生産性の向上を目的とした情報化投資を超えて、経営戦略の為の情報化投資へと大きなうねりを生み出した時代となっていた。

この時代は、右肩上がりの経済成長に支えられて、都銀の第 3 次御ライに代表される大規模なシステム開発が行われ、企業経営の屋台骨を支えるシステム作りが多くの企業で実施された。都銀の第 3 次オンラインで 1200 万ステップの規模とされ、投資額は優に 1000 億円を超えていたと言われる。現在の平均的な情報化投資額からすれば桁外れの額と言える。このような巨額の投資が行えたのも、強い製造業と豊かな資金力を持った金融機関との連携があればこそ可能だったと言える。

しかし、1990 年代に入り、バブル経済の崩壊とともに、グローバル会計の導入が実施され、フリーキャッシュフロー中心の経営へと、これまでの含み資産型経営から大きく変化する。これは、長期的展望を前提とした様々な投資が許されなくなり、短期的な収益に直結する投資を求められるようになったことを意味する。

企業の情報化投資も、このような経営に引きずられる格好で投資対効果の明確なものがもとめられ、さらに大型投資の抑制、研究開発コストの削減が行われるようになった。

2000 年初頭の今、多くの企業はその情報システムのあり方について深刻な問題に直面している。その幾つかをあげてみると次のようなことになる。

- 弛まぬコスト圧縮の要求。
- これまで営々と作り上げてきたレガシーシステムの保守が次第に難しくなってきている。新規に再構築が出来なかった事と、長年の改訂作業によりソフトウェアの論理構造が弱くなってきているが主な原

因だが、COBOL や簡易言語の保守要員が少なくなってきていることも上げられる。

- インターネットに代表される新しい技術が社内に蓄積されていないため、5 年間の耐用年数を目標としたシステムでさえ安心して開発できない。
- 次々と出てくる新しい技術についてゆけない。
- 新しいシステムとレガシーシステムとの連携が難しい。
- 業務アプリケーションを ERP などのパッケージに置き換えられない。
- 市場の変化が激しいため、それに対応する為のシステム開発基幹が短期となっている。

このように、企業がその情報システムに抱えている課題は、多様で複雑なものとなっている。さらに、企業統合や M&A などが今後も多発すると考えられ、システム統合や他社のシステムとの連動が必要となってくる。企業が抱える問題は、小型のシステム開発の有り様から期間システムの今後の構想に至るまで、そう時間的には猶予できるものではない。

2.1.1.2 必要とされている技術

企業の抱えている情報システムに関する現在の問題は、先に述べたように実に複雑で深刻なものとなっている。これらの問題を如何に解決すべきか、またその為に要求されている技術とはどのようなものか、マクロ的な観点から考察してみる。

1980 年代の経営環境は右肩上がりであり、市場のニーズにしたがって組織も緩やかな変革で済んでいた。組織とは、市場のニーズに応えるために最大効率の経営を実現する装置だと考えられる。さらに情報システムは、その組織を有効に機能させるための仕組といえる。市場が安定していれば組織も比較的安定しており、企業活動を支えている IT も 2〜3 年がかりの比較的長期間で開発すれば良かった。

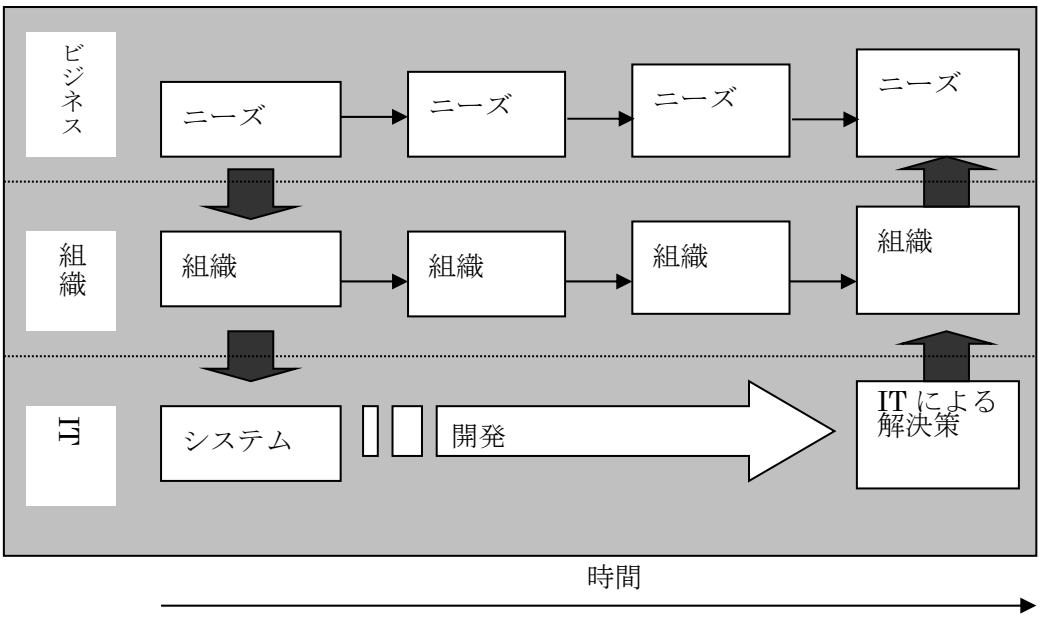


図 2-1 市場ニーズと組織のあり方

しかし、ビジネス領域が複雑に変化する時代には、2 年前のシステム化ニーズはシステムが完成した時には役に立たなくなっている。この問題は、企業の組織についても同様に発生する。

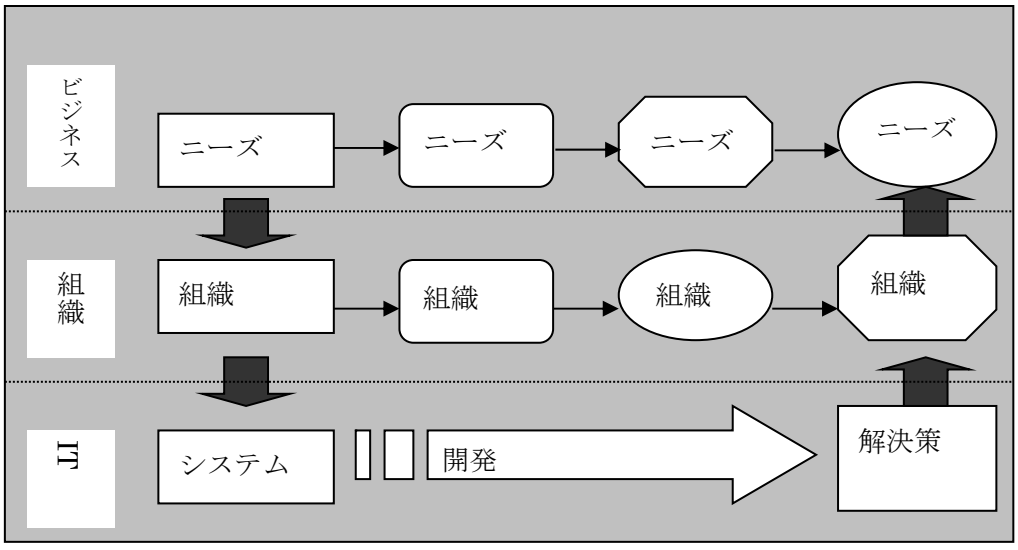


図 2-2　市場ニーズと組織のあり方 2

これからの組織とそれを支援する IT システムはその時その場で的確に市場ニーズに応えるものでなければならない。

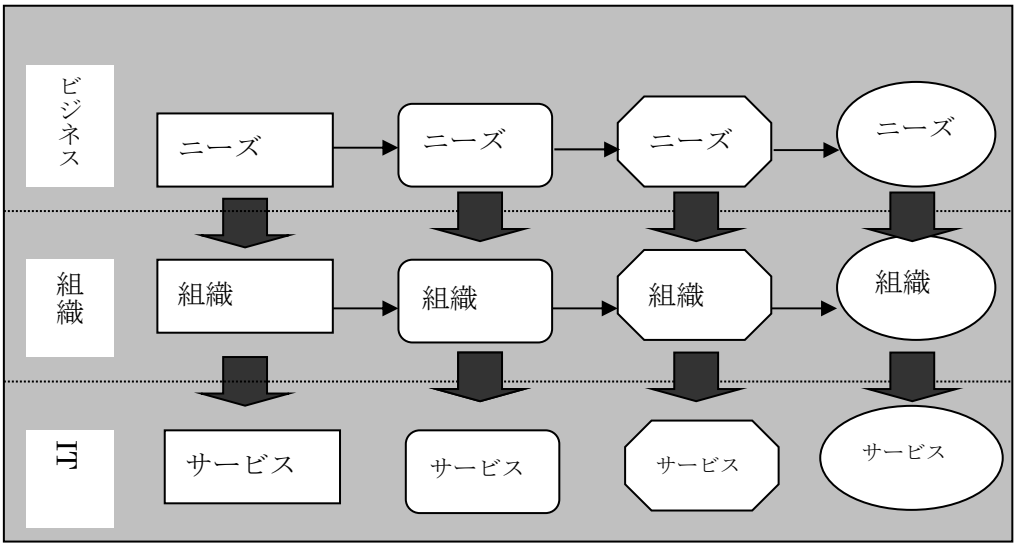


図 2-3　市場ニーズと組織のあり方 3

世界的規模での競争激化は、シェアの獲得と維持の為にもこれまでの図式と大きく異なってくる。

顧客要求の変化は、情報伝達の方法がインターネットなどによって著しく早くなっており、これに伴い新製品の市場への投入も他社より、あるいは自社の従来製品よりもさらに早く行わなければならなくなった。これは高品質の商品を早く廉価に提供できる企業でなければ生き延びることができない時代に突入したと言える。企業にとっては大変困難な課題となる。これを実現させる選択肢は限られている。商品を市場に投入する「仕組み」すなわちプロセスを改善させる事が求められている。これからの時代に求められている商品の要件は、「廉価」であり、「スピード」があり「品質が高い」ことだが、同様に商品を作りだすプロセスにおいても、廉価、スピード、高品質が求められ、商品の競争はプロセスの競争に置き換えて考えることができる。

これらの経営における三要素を科学的に行うために経営工学がある。

経営工学は今から約 90 年ほど前に、これまで経営者の経験や勘に頼っていた経営をシステムとして扱うことで企業の収益力を高めるための技術として誕生している。鉄鋼会社で適用されたテーラーシステムと、自動車会社のフォードシステムがその初期の事例となる。経営工学の目的は、安く、高品質のものを，納期どおりにおさめることにある。日本が 1980 年代に大きな成長を遂げることが出来たのも、経営工学のおかげといえる。そして、時代の変化と共に経営工学も製造現場から営業・流通も含むようになり、さらに現場から、より上位の管理者の仕事に経営工学を生かそうという観点へ移行してきた。それは経営が設備投資を含めて企業の将来を見つめ、中・長期的に計画を立てる必要性が高

くなった事にある。このためには、企業内外の膨大な情報を処理し、的確な方法論を駆使することで的確な経営判断が求められるようになった。情報システムはこれらの要件を満足させることの出来る重要な装置となる。いまや経営工学と情報システムは、切り離すことの出来ない関係になっている。

一方、この経営工学を支えるための情報システムに関する技術としてソフトウェア工学がある。ソフトウェア工学の目的は、その対象をソフトウェアの品質と生産性に置き、誰が開発しても同じ方法と技術を使えば同じ品質と生産性を得ることができるようにすることを主眼においている。歴史的には 50 年に近い学問となっている。

米国における生産性の高い企業の分析報告では、高生産性を実現した企業は、経営工学を情報技術と直接に結びつけていると結論している。特に、高生産性を実現した企業の特長として、ビジネスのやり方を革新し改良するために情報システムを活用している点を挙げている。この観点からすれば、経営工学と情報システムが適切に連動し合えば、企業の生産性は向上することになる。

しかし、ここで問題になるのは、経営工学が市場の変化に即応して、安く高品質の商品を早く提供するために継続的な活動を必要とするのに対して、経営を支援する情報システムの構築には、これまでの開発手法では 2 年から 5 年を要している。これでは変化に対応した情報システムにはなり得ない。

継続的な商品開発のプロセス改善およびプロセスの最適化には、これまでのソフトウェア工学は適さないものとなってしまっている。

これまでのことを要約すると、企業が激しい市場競争に対応できる力を持続させるためには、継続的な組織とプロセスの改革が必要であり、それを支える情報システムとの連動が不可欠となる。つまり、経営工学と情報工学を統合させる何かが必要となっている。

しかし、両者の統合はもともと別の目的で扱われてきた技術であるため、大変難しいものとなっている。もともとソフトウェア工学は、ビジネスを意識して進められてきたものではなかったことが大きな両者の乖離の原因と考えられる。組織が変化に対して反応が鈍い以上に、これまでのソフトウェア工学の考え方は、柔軟性よりも安定性を重視してきている。

この両者の乖離は、たとえば企業が市場の変化に対応して迅速にプロセスと組織を変更したとしても、情報システムの方では無数の変更を行わなければならない。変更によってこれまで安定してきた機能が不安定になることもあるし、変更に時間がかかる場合もある。このような悪材料は経営においては混乱の元となる危険性がある。結局、ほとんどの企業が思うように経営のプロセスを変えることができず情報システムに足を引きずられなければならなくなる。

今、企業が求めている技術は、経営工学とソフトウェア工学の乖離を解消できるものである。つまり、経営を支える情報システムを合目的的に構築することのできる技術が求められている。
これを技術要件として要約すると次の 3 点になる。

- ビジネス・プロセスの構造がソフトウェアの構造と整合性をもって調和していること。
- ソフトウェアの設計が分かりやすいもので、ビジネス・プロセスとの乖離が解消できるものであること。
- ビジネス・プロセスの変更に柔軟な対応ができるソフトウェア構造であること。

これらの要件が満たされるソフトウェアが実現できれば、企業は市場の変化に即応した柔軟なプロセスの編集と組織の改革を行うことができるようになる。

2.1.1.3 実現可能な技術

コンピュータの言語すなわちプログラミング言語には、これまで、手続き型言語や関数型言語、パラメータ型言語などがあった。いずれもシステム化要件を機能に分解し、さらに機能を手続きに分解したあとにその手続きを記述してコンピュータを動かすための言語といえる。
これまでのプログラミング言語は、手順化されたものをコンピュータに指示するのには適していたが、複雑なものや非連続に相互作用する事象を実現するには難しいものがあった。
これに対してオブジェクト指向技術は、複雑なシステムをコンピュータ上で実現するために開発されている。別の言い方をすれば、人間社会の複雑な仕組みをモデル化する技術ということになる。
また、オブジェクト指向技術はプログラミング言語から独立しており、プログラミング言語の持つ機能に依存することはない。

このようなオブジェクト指向技術とは具体的にどのようなものなのか、さらにオブジェクト指向技術がなぜこれからのソフトウェア開発技術として注目されているのかを簡単に紹介する。

オブジェクト指向技術の原点は 1960 年代にある。1961 年に SIMULA というシ

ミュレーション言語が開発されている。この言語はシミュレーション以外にも有効であるとのことから、汎用言語としての改訂を繰り返している。1967 年に SIMULA67 がリリースされ、この時点でオブジェクトの概念やクラスの考え方が初めてリリースされ現在のオブジェクト指向技術を支える原点となっている。オブジェクト指向技術は歴史上の経緯が示すよう、人間世界の実体をモデル化する技術として始まっている。

オブジェクト指向は難しいとされているが、その一つには「オブジェクト」に対する適切な日本語がないことが起因しているものと思われる。本論ではオブジェクト指向そのものを扱うのが目的ではないので、詳細の技術論は避けるが、最初にオブジェクトで躓くと全体が見えなくなる恐れがあるのを避けるために簡単にオブジェクトの考え方を紹介しておく。

OBJECT は OB〜が「〜に対して、〜の方向へ（towards)」という意味を持つ説頭辞であり、発音もこの部分にアクセントが置かれる。アクセントが置かれる意味は、それが重要だからであり、別の意味の OBJECT は真ん中の〜JE にアクセントが置かれ「反対する」という意味になる。そして、〜JECT は「投げる」の意味をもって、全体として「対象、物体、事物、目的」と日本語に訳される。これらの事を総合すると、オブジェクトの意味するところは、自分を中心に置いて、人間が五感を通じて認識できる外界のものごとと言える。

我々が何かを認識する場合、例えば車輪は回るといった「ものごと＝車輪」の持つ機能による認知、車輪は円いと言った「ものごと＝円い」などの分類を行うことによる認知、さらに車輪はタイヤとスポークと軸で構成されていると言うような「ものごと＝車輪＝属性」の構造による認知を行っている。つまり人間は、認識する対象物（ものごと）が、何ができるものか、何に似ているのか、何からできているのか、という 3 つの観点で「ものごと」を認識する。

オブジェクト指向では、何ができるものかはクラス、何に似ているのかはインヘリタンス、何からできているのかは属性というそれぞれの呼び方でモデリングすることができるようになっている。

オブジェクト指向では、人間が実社会を観察する方法に近い概念を使ってモデリングからプログラミングまで行うことができることが大きな特長となっている。この点からすれば、従来のソフトウェア開発としての構造化手法は、何ができるものかといった機能に中心を置いたものであり、データのモデリングで使われる E/R などは、何からできているのかといった属性に主眼をおいた方法論であることから、十分に複雑なモデルは扱うのが難しいものであったと言える。

オブジェクト指向技術では、モデリングによるシステムの分析から設計、そ

してプログラミングに至るまで統一された方法論で実施されるため、一貫性のあるソフトウェア開発が可能となった。

このように、オブジェクト指向技術は実世界のモデリングを得意としている。これはモデルベースのシステム構築と呼ばれている。

ビジネスの本質的な要素を忠実にオブジェクトとしてモデルに表現することによって、オブジェクトがビジネス上の振舞いを実現することができるようになる。そして、それぞれのビジネスのオブジェクトは、対応するソフトウェア・オブジェクトに引き継がれる。

これまで述べてきたように、複雑で大規模なモデルを設計するにはオブジェクト指向技術が適しており、上位のモデルにはビジネスのモデルが位置付けられ、下位のモデルにはソフトウェアのモデルが位置付けられる。上位のモデルと下位のモデルの間には、抽象度の異なる数階層のモデルを置くことができる。この階層の数は、ルールによって決められものではなく、人間の認識の出来る範囲で決めればよい。

つまり、あるビジネス領域をモデリングして、これをオブジェクトの集まりとして表現できれば、さらに一つひとつのオブジェクトを詳細に分解してゆくことを繰り返すことで、最終的にはソフトウェアの部品にまで展開されることになる。

しかもどのレベルのオブジェクトであっても、それ自体は適切に振る舞うための全ての情報を持っており、カプセル化という大切な内容を守る仕組みを持つことができる。さらに、オブジェクトは他のオブジェクトと組み合わせて、ひとつの機能を構成することができ、オブジェクトの部品化による品質の向上と生産性の向上をもたらすことができるとともに拡張性をも持たせることができる。

このように、オブジェクト指向技術を利用することによって、これまでのソフトウェア開発にはなかった経営工学と情報工学を融合することができるようになる。

特に、オブジェクトをビジネスのモデリングレベルで再利用する試みが生まれており、これをビジネスオブジェクトと呼んでいる。

ビジネスオブジェクトが実現できることによって、ソフトウェア部品の交換だけではなく、業務のコンポーネント化やコンポーネントの流通までもができるようになり、企業のコスト削減や情報化投資の効率化を推進することができる可能性がある。

2.1.1.4　予想される課題

オブジェクト指向技術は、変化の激しい市場に対して、情報システムで戦ってゆく上で非常に有望な方法論と言える。また、経営工学と情報工学を融合する技術としても期待される。

このような将来性のあるオブジェクト指向技術であるが、使いこなす為にはいくつかの課題も予想される。

ひとつには、オブジェクト指向技術はこれからの技術とされている点で、この課題には次のようなものがある。

- 現在のノイマン型コンピュータでは、情報の質は量に還元される。言い方を代えれば、人間に近い技術であればあるほどコンピュータに負荷を与えてしまう。この事は、オブジェクト指向は拡張性が高いが実行性能は遅くなる事を意味している。
- 次の課題は、オブジェクト指向開発を効率的に行うための開発体制にある。オブジェクト指向開発では、プロトタイピングでのアプローチや、何度も反復しながら次第に機能を増やしてゆくやり方、さらに部品を切り出して再利用しなければ効率があがらないといった要因をうまく取扱うことができる開発体制が必要となる。
- 現時点では幾つかの開発方法や組織が考案されて実施されているが、時間的に有効性が確認され、誰もが共有できるまでには至っていない。
- 最後の課題としては、オブジェクト指向技術は非常に難しいとされている点にある。

大手のSI会社ではここ数年新入社員にオブジェクト指向言語の教育を行っているが、500人の新入社員で3ヶ月のオブジェクト指向言語の教育を行って、実際にプログラミングできるようになったのは50人程度とされている。約10%の成功率だが、これでもまだ高い数字だとする評価もさらにある。

オブジェクト指向技術の難しさは、オブジェクトという概念の難しさが本質ではなく、この技術がソフトウェア工学の歴史50年の知識と体系が凝縮されたものである点に本当の理由がある。

さらにビジネスオブジェクトの課題となるとさらに深刻で複雑な問題がある。オブジェクト指向技術が経営工学と情報工学を融合するものだとしているが、この事は、ITの分かる経営者、経営の分かるIT技術者が必要だと言うことを意味している。しかもこのIT技術者はオブジェクト指向技術を理解していなければならない。また、ビジネスオブジェクトの場合には、企業内・企業間でのコ

ンポーネント流通を前提としているから、業界業種全般に関する知識、さらに自社と業界との違いなどを知った上で設計を行わなければならず、その専門性は特殊中の特殊となる。果たして、このような人物を確保することができるのか、あるいは育成が可能なのか、また何某かの仕組みがこれを解決するのか、今のところ未知数と言わざるを得ない。

最後に、ビジネスオブジェクトを含んだ意味でのオブジェクト部品の再利用と流通問題を取り上げる。

これまでにシグマ・プロジェクトやサンフランシスコ・プロジェクトなどに代表されるソフトウェア部品の再利用と流通に関する幾つかの試みがなされてきたが、いずれも成功までには至っていない。

また、銀行の第 3 次オンラインにおいても、都銀のシステムをパッケージ化して他の銀行に導入することが行われたが、当初の見積よりもはるかに多額の投資となってしまった事がある。これらの失敗の原因は、部品切り出しのルールにあったとされている。特定の範囲でのルールの適用はできたとしても、広範囲の再利用と流通を促す為のルールにはなりえなかったようだ。

厳密なルールを適用すれば柔軟性に欠け、柔軟性を持たせようとすれば厳密さに欠けるというパラドックスがこの課題には存在する。

ましてやビジネスオブジェクトの再利用となれば、ルール化の対象は業務全般に及ぶ。果たしてそのようなルールが設定できるものだろうかという課題が浮かびあがる。

2.1.1.5　課題解決に必要となる事項

オブジェクト指向技術で実装されたソフトウェアの実行速度が遅い点については、ムーアの法則（注 1）が今後もしばらく有効であるとの前提をとるならばそれほど問題視することはないかもしれない。しかし、オブジェクト指向技術を用いない場合との比較では、相変わらず劣性であることに違いはなく、結局は総合的な機能面と速度とのトレードオフにならざるを得ないと思われる。

（注 1）米国の Intel 社社長であった共同設立者のゴードン・ムーア（Gordon Moore/現在は名誉会長/1929～）が「一定のシリコン上にエッチングできるトランジスタの数は 18 カ月ごとに倍になる」という、1965 年にプロセッサの発達スピードを予測した法則の名称。つまり、マイクロ・プロセッサのコンピューティング・パワーも当然、18 カ月ごとに倍になり、同じコンピューティング・パワーを買うために要する費用は 18 カ月で半分になる。

次にオブジェクト指向技術の難しさに関する解決策については、いくつから

の施策が考えられる。まず、フレームワーク技術あるいはアーキテクチャパターンを用いることによる解決がある。フレームワークとは、オブジェクトがひとつの仕組みの中で互いに協調しながら動作をするための仕組みを意味している。

アーキテクチャパターンとは、ソフトウェア設計を行う際の論理構造体で参照モデルの事を意味している。

オブジェクト指向技術は難しいものであっても、優秀な人材を集中させてフレームワークやアーキテクチャパターンを開発し、短い教育期間で大量の開発者を育成し実装に配備することにより解決できる。

オブジェクト指向技術向けの開発体制については、いまだ試行錯誤の状態にある。しかし、その中でも「統一プロセス（ユニファイド・プロセス)」が次第に定着する傾向にある。今後、幾つかの選ばれた方法で開発が何度か繰り返され、その実効が評価されれば結果的に最適解としての開発方法論が定着することになるものと考えられる。

2.1.2　ハードウェアとネットワークの推進

2.1.2.1　ハードウェアの進歩

オブジェクト指向技術がハードウェアの性能に依存する割合は決して少なくない。その理由は、オブジェクト指向技術が拡張性に高く柔軟性に富んでおり、その分処理速度が遅くなる傾向にある。

技術的な観点からは次のようなことが言える。

拡張性が高く柔軟性に富んでいるということは、オブジェクト指向技術を用いた実装では、内部での動作として各種の処理を行うための準備をコンパイル時ではなく実行時に行う場合がある。クラスやメソッドの名前解決をダイナミックに行うなどの処理はその一例だが、これだけで処理速度は遅くなる。さらに JAVA 言語の場合には、どのコンピュータでも動くようにするために、コンパイルされたバイナリを JVM という仕組みで動かす為に、JAVA 以外の言語と比較すると 20%程度速度が落ちるようだ。さらに JAVA ではキャストと呼ばれるデータ型の変換や、動的コンパイル、ガベージコレクションといった処理も行われるため、コンピュータの性能に依存する場合が多い。

このような観点からハードウェアの進歩がどのようにオブジェクト指向技術に関わっているのかを考察する。

● 高速化

MPU（CPU）の高速化を意味している。それ以外にもボードなどの設計技術によっても全体の処理速度は変化する。

高速化のもたらすオブジェクト指向技術にとっての恩恵は、ハードウェアの制限によるオブジェクト設計の制約から解放され、より論理的な設計ができるようになる事にあるだろう。また、実際に開発されたプログラムが稼働する環境がそれほど高速ではなくても、コンパイルとテストの為のコンピュータが高速であることは、開発の生産性向上に貢献する。オブジェクト指向言語のコンパイルは他の言語と比較しても MPU のパワーを大量に消耗する。

また、オブジェクト指向言語では、GUI 操作やそれにともなうベクトル線を取り扱いことがあるため、MPU の消費量が多くなる傾向にある。この点でも MPU の高速化はオブジェクト指向技術に大いに貢献する。

● 大容量化

大容量の記憶媒体としてはメモリーとハードディスクが主に挙げられる。メモリーの大容量化は、ハードディスクとの入出力処理を少なくするため、処理効率向上に貢献する。また、十分な容量のメモリーが無い場合には、バーチャルメモリー機能が働いて、極端な処理速度の低下を招くことがある。

ハードディスクの容量に関しては、オブジェクト指向言語で開発したプログラムが他の言語と比較して、極端に多くのディスク空間を消耗することは少ない。しかし、最近のオブジェクト指向言語は、音声や画像などのメディアなどを扱うため、どうしてもディスク容量は大きくなりがちとなる。

● 信頼性の向上

稼働率の向上が信頼性の主要な観点となる。最近のプログラムは、スタンドアローンではなく、ネットワークで機能が連携している場合が多い。特にリモートで接続されたコンピュータ同士が、それぞれのオブジェクトを利用し合う場合があり、安定した稼働は必須の条件となってきている。

● 小型化

究極の小型化はユビキタス・コンピューティングにつながっている。オブジェクト指向技術に関するこれまでに述べてきた特長以外のもう一つのハイライトは、各種のコンピュータ同士がネットワークを通じて相互に作用し合うことにあり、そこにオブジェクトが重要な役割を果たすことにある。

例えば、家庭内のサーバから冷蔵庫に組み込まれたコンピュータに温度設定の操作を行う場合には、サーバから冷蔵庫の温度を調べるオブジェクトが

メッセージを発信し、それを受け取る冷蔵のコンピュータ内のオブジェクトが温度センサーから温度データを獲得してサーバに返信するような例では、様々な家電製品に組み込まれたコンピュータの中で、それぞれの機能を果たすオブジェクトが相互に関連し合っている。

コンピュータの小型化は、このようなコンピュータ社会の実現をもたらし、それを実装するための設計と開発技法にオブジェクト指向技術が役立つ。

- 低価格化

低価格化によって、誰もが家庭でオブジェクト指向技術の勉強ができるようになった。

1980 年代には、オブジェクト指向でのソフトウェア開発をおこなうために高額な UNIX ワークステーションが必要とされていた。これは企業の研究施設や大学など限られた場所でしかさわることのできないものだった。それが今では、当時の 100 倍近い機能のコンピュータが 200 分の 1 程度の価格で入手できるようになった。

さらにインターネット上からは、豊富な教材が入手でき、技術情報も開示され、FAQ などのサポート情報やサービスも無料で手に入る時代となった。特に、オープンソースコードの代表格である LINUX では、C++や JAVA などのコンパイラが無料で提供されている。

誰もがオブジェクト指向技術を勉強できる時代には、絶対数が不足しているオブジェクト指向開発の技術者を解消するひとつの福音になるかもしれない。

2.1.2.2 ネットワークの進歩

ネットワークの進歩には、ブロードバンド技術、IPV6、インターネット技術、無線 LAN 技術と常時接続、ブルーツースなどめざましいものがある。オブジェクト指向技術にとって、ネットワークが進歩することの意義は、大規模な分散システムの実現にある。

1980 年代の後半には、ワークステーションや PC の性能が飛躍的に向上し、価格が急激に安くなった。これにより、これまでのメインフレーム中心から、ダウンサイジングとオープン化が始まっている。

これまでのメインフレームによる集中処理型からクライアント・サーバシステムによる分散処理へと発展し、現在では分散オブジェクト技術へ移行している。

これまでのクライアント・サーバシステムでは、従来よりも開発コストを押さえることが可能となり、システムの拡張性も向上したが大規模なシステムの適用ではその限界が指摘されている。

クライアント・サーバシステムの問題点を解決する手段として注目されているのが、分散オブジェクト技術である。

分散オブジェクト環境を実現するミドルウェアの仕様である CORBA が、オブジェクト指向技術の標準団体である OMG によって策定されている。CORBA 仕様のアプリケーションは、そのアプリケーションが稼動しているアーキテクチャ、言語の相違に関係なく連携できる。

分散オブジェクト技術では、次の 3 点に着目している。

- システムの機能分割と分散配置
- コンポーネント化と再利用
- 標準技術の採用と相互運用性

分散オブジェクト技術では、全体のシステムを独立性の高い幾つかの機能単位に分割して開発する手法を前提としている。これは 3 層構造アーキテクチャと呼ばれ、プレゼンテーション層、ビジネスロジック層、データ層に分かれている。これらの層は機能単位ごとに最適なマシンに分散させて配置される。また、分散処理においては、アプリケーションにお互いの相手の物理的な位置を意識させずに通信させることが、メンテナンス性や移植性の観点から重要となる。分散オブジェクト技術はこのような要求を満たす技術といえる。

分散オブジェクト技術を使うことで、ネットワーク・プログラミングやセキュリティ・メカニズムなどの詳細を気にせずに、あたかもローカルの JAVA や C++ のオブジェクトのメソッドを呼び出すのと同じ感覚で、リモート・マシンに存在するオブジェクトを呼び出すことが可能になる。

さらに、分散オブジェクト技術はオブジェクトの再利用性を向上させるための仕組み、つまりコンポーネント化の仕組みを提供しているため、利用する側がそのコンポーネントの内部実装を意識しなくてもよいのと、バイナリ・レベルでコンポーネントを再利用できるという特徴を持っている。

分散オブジェクト技術は今のところ実用段階に入ったばかりで、使用されているネットワークも社内 LAN など高速のものだが、今後のネットワーク技術の発展によって、インターネット上でもストレスのない分散オブジェクト技術が実現できる時代になると、コンポーネントの流通や再利用が飛躍的に向上することが期待できる。

2.1.3　ルールとしてのEJB

オブジェクト指向技術によって生み出されるコンポーネントやソフトウェア部品が再利用され流通するようになるためには統一されたルールが必要であることは先に述べた。

ルールの適用には 2 段階があると考えられる。ひとつには、分析と設計段階での適用であり、二つ目には実装段階での適用がある。現在の技術的な動向を観察すると、分析と設計段階においては UML が主流であり、実装段階の適用にはEJBが普及の兆しを見せている。

2.1.3.1　UMLとは何か

オブジェクト指向技術による分析と設計の方法論は、表記法とプロセスの 2 つの要素があり、ノーテーションはオブジェクト指向分析設計による作成されたモデルを表現するための設計図であり UML は ORG によって制定されたノーテーションの標準規格となっている。

一方、プロセスはオブジェクト指向分析/設計を行うための手順やガイドラインを指している。これまでのオブジェクト指向分析と設計法にはSHLAER/MELLOR法やCOAD/YORAN法、CATALYSISなどがあり、これらの手法においては独自にノーテーションとプロセスを規定しており、基本的に相互の運用性は考慮されていなかった。UMLからはノーテーションとしてこれを使用することが前提となっている。

UMLでは以下に示す9種類の図が用意されている。これらの図を用途に応じて使い分けることになる。

- クラス図:クラス間の静的な構造を表現するための図でオブジェクト指向開発の中心となる。クラス図はクラス、インタフェースとそれらの関係から構成される。
- オブジェクト図:クラス図で表現されたクラスが実際にオブジェクトとして実体化された場面の関係を表す。オブジェクト図はシステム実行時のオブジェクトのスナップショットになっている。
- ユースケース図:システムの要求仕様表現するために使用される図。システムの提供する機能を示すユースケースとシステムのユーザを示すアクタ、そしてそれらの関係から構成される。
- コラボレーション図:オブジェクト間のインタラクションを空間的な観点で表現するための図。

- シーケンス図:オブジェクト間を流れるメッセージのシーケンスを表現する図。オブジェクト間のインタラクションを時間的な観点で表現する。
- ステートチャート図:状態遷移図とも呼ばれる図で、オブジェクトの状態の遷移を表現するために使用する。
- アクティビティ図:アクティビティのフローを表現するための図。システムの動的側面をより具体的に記述するときに使用する。フローチャートやデータフロー図といった構造化手法で使われていた図はこのアクティビティ図で表現できる。
- コンポーネント図:システムの物理的なインプリメンテーションを表現するための図。EJB などのソフトウェアコンポーネントによって構成されたアプリケーションの構成図は代表的な例。
- ディプロイメント図:システム動作時の動作環境を表現するための図。

※各図については、概要をここに紹介するが、事例を含めての解説は、「4.7 モデリングの実際」以降を参考。

2.1.3.2 ユニファイド・プロセス（UP）とはなにか

ユニファイド・プロセスはオブジェクト指向分析/設計の 3 大方法論であった BOOCH、OOSE、OMT を統合したものであり、オブジェクト指向分析設計についての参照モデルとして扱われる。

ユニファイド・プロセスのアーキテクチャは以下の 3 つにされる。

- ユースケースドリブン
- アーキテクチャセントリック
- イテレーティブ/インクリメンタル

ユースケースドリブンとは、ユースケースを起点にして開発プロセスを進めて行くということで、まずユースケースから分析フェーズの生産物が生成される。次に分析フェーズの生産物から設計フェーズの生産物が生成される。分析フェーズと設計フェーズの生産物はクラス図が中心となり、動的な振舞いはシーケンス図やステートチャート図などを使って表現される。

次に実装フェーズでは JAVA などによるプログラム生産物が生成される。さらにコンポーネント図やディプロイメント図といったアプリケーションの物理的な構造を表現するための図も作成される。

テストフェーズで必要となるテストケースを生成するための情報としてユー

スケースが使用される。ユースケースはシステムが外部に対して公開する機能を表現している。

アーキテクチャとは、開発に際してチーム全体に共有化されるビジョンの事を指す。ここで言うビジョンとは、開発プロジェクトの方向付けを行うもので、このビジョンに基づいて、プロジェクト参加者全員が自らの役割を認識して、具体的な開発作業を遂行する。

大規模なソフトウェアの開発では、明確で堅牢なアーキテクチャは必須の要件となる。

ユニファイド・プロセスではアーキテクチャを表現する枠組みとして 5 つのビューを用意している。

- ユースケースビュー
- デザインビュー
- インプリメンテーションビュー
- プロセスビュー
- デプロイメントビュー

ユースケースビュー：システムのエンドユーザやアナリスト、テスターの目からみたシステムの振舞いを記述するためのビューとなる。このビューはユースケース図が中心的な図となり、動的な側面をインタラクション図(シーケンス図、コラボレーション図)、ステートチャート図、アクティビティ図で記述する。

デザインビュー：システムの機能と構成について記述するためのビュー。このビューはクラス図、オブジェクト図が中心的な図となり、動的な側面をインタラクション図、ステートチャート図、アクティビティ図で記述する。

プロセスビュー：システムの並行処理、同期処理の側面を把握するためのビュー。システムの静的な側面はデザインビューと同様にクラス図やオブジェクト図を用い、動的な側面はインタラクション図、ステートチャート図、アクティビティ図を用いて記述する。

インプリメンテーションビュー：システムを組み立てるための物理的な構成を記述するビュー。システムの静的な側面はインプリメンテーション図を用いて記述する。このインプリメンテーション図がインプリメンテーション図の中心となる。システムの動的な側面はインタラクション図、ステートチャート図、

アクティビティ図を用いて記述する。

デプロイメントビュー：システムを構成するハードウェアの構成を記述するビュー。システムの静的な側面はディプロイメント図を用いて記述する。このディプロイメント図がインプリメンテーション図の中心となる。システムの動的な側面はインタラクション図、ステートチャート図、アクティビティ図を用いて記述する。

2.1.3.3 EJB とはなにか

エンタープライズ JAVABeans とは、アプリケーションのビジネス ロジックをカプセル化するサーバ側のコンポーネントの事を指している。ビジネス ロジックは、アプリケーションにインフラストラクチャを提供するコードとは対照的に、アプリケーションの目的を実行するコードのことを指している。たとえば、ある在庫管理アプリケーションで、EJB が CHECKINVENTORYLEVEL と ORDERPRODUCT というメソッドでビジネス メソッドを実装しているとすると、リモート クライアントはこれらのメソッドを呼び出すことで、アプリケーションによって提供される在庫管理サービスにアクセスすることができる。

EJB は、ホーム インタフェースとビジネス インタフェースという 2 種類のインタフェースを公開している。EJB2. 0 以前では、ビジネス インタフェースは一般に、Beans のリモート インタフェースと呼ばれていた。

クライアントは、ホーム インタフェースを使用して、通信する EJB のインスタンスを取得する。ホーム インタフェースのメソッドは、EJB インスタンスを作成または検索するものに限定されている。クライアントは、EJB インスタンスを取得すると、EJB のビジネス インタフェースを呼び出して処理を実行する。ビジネス インタフェースは、EJB にカプセル化されたビジネス ロジックに直接アクセスする。

EJB を表す EJB コントロールを作成するには、ホーム インタフェースとビジネス インタフェースの名前を知っておく必要がある。通常、ホーム インタフェースの名前は Com.MYcompany.MYBeannamehome という形式を取り、ビジネスインタフェースは Com.MYcompany.MYBeanname という形式を取る

EJB の種類として、セッション Bean、エンティティ Bean、およびメッセージ

駆動型 Bean の 3 種類の EJB がある。

EJB コントロールを使用すると、WEB サービスはセッション EJB およびエンティティ EJB のクライアントとして機能できる。リクエストは JMS コントロールを使用してメッセージ駆動型 Bean に間接的に送信されるので、EJB コントロールではメッセージ駆動型 Bean はサポートしていない。

セッション EJB は、アプリケーション サーバ内の単一のクライアントを表す。アプリケーションのクライアントは、セッション Bean のメソッドを呼び出すことでアプリケーションにアクセスする。セッション Bean は、サーバ内のビジネス タスクを他の EJB を呼び出すことで実行し、複雑な処理をクライアントから見えないようにしている。

エンティティ Bean は、永続ストレージ メカニズム内のビジネスオブジェクトを表している。ビジネスオブジェクトの例としては、顧客、注文、製品などがある。永続ストレージ メカニズムはリレーショナル データベースとなっている。一般に、各エンティティ Bean はリレーショナル データベース内の基底のテーブルを持ち、Bean の各インスタンスはそのテーブル内の行に対応している。

メッセージ駆動型 EJB は、JMS（JAVA MESSAGE SERVICE）メッセージをリスンする機能を持つエンタープライズ Bean を指している。メッセージは、JMS に準拠したコンポーネントまたはアプリケーションによって送信される。メッセージ駆動型 EJB は、J2EE アプリケーションがメッセージ ベースのレガシー アプリケーションとの関係に参加するためのメカニズムとなっている。

多少専門的な説明になったが、このような EJB の構造とルールに則って開発されたオブジェクト指向のソフトウェア部品やコンポーネントは、ネットワーク上で有機的に連携することができる。
ビジネスオブジェクトを考える場合に必要なルールは、EJB を踏襲することで、世界規模の適用に参加することができ、原理的にはグローバルなコンポーネントや部品の獲得が可能となる。また、提供もできるようになる。

2.1.4　オブジェクト指向技術の発展

これまでのコンピュータの世界は、現実世界の仕組みを機能の観点で把握し、機能を手続きに変換させ、それをプログラムとしてコンピュータに指示させていた。

オブジェクト指向技術は、このような手続き型ではなく、コンピュータに人間の観点を与えて、あたかも現実世界でのオブジェクトの振舞いをコンピュー

タに振る舞わせる。

このパラダイムの変換は、人間がコンピュータに歩み寄っていた立場から、コンピュータを人間に近づかせるという画期的なものとなった。このような人間とコンピュータとの関係が大逆転する事象がどのように起きたのか、その歴史的流れを把握して、これからのオブジェクト指向技術の発展について考察する。

ちなみに、オブジェクト指向技術は、次の 4 つのコンセプトが基本となっている。

- カプセル化
- クラス/インスタンス
- インヘリタンス(継承)
- メッセージパッシング(アクセス方法の抽象化)

2.1.4.1 オブジェクト指向技術発展の歴史

- 基本概念創世期(1960 年代後半から 1970 年代半ば)

1960 年代後半にデータの抽象化、クラスからのインスタンス生成、クラスの階層と継承、などの機能を持ったシミュレーション言語 SIMULA がノルウェー計算センターで開発された。

1970 年頃、MIT (マサチューセッツ工科大学) の C.HEWITT が、メッセージパッシングを主体とした計算モデルである ACTOR モデルを提案している。

1974 年には、同じく MIT の B.LISKOV が本格的な抽象データ型機能を持つ CLU を開発した。

この 10 年間で、オブジェクトとクラス、メッセージパッシング、抽象データという考え方が揃い、オブジェクト指向技術の基本概念が整うことになる。

- 言語開発期(1970 年代後半から 1980 年代半ば)

オブジェクト指向技術の基礎ができあがると、これを実現するための言語が必要になる。最初に登場したオブジェクト指向言語として、XEROX 社の Smalltalk がある。1970 年頃から人間の創造的活動をコンピュータが支援する方法を研究する過程で開発された。初代 Smalltalk に改良を加えた次のバージョンが 1980 年にリリースされた Smalltalk-80 であり、当時は画期的な言語として世界から評価された。

その後、オブジェクト指向への関心が急速に高まり、本格的なオブジェクト指向言語の開発研究が活発になった。ほとんどは、既存の言語をベースにオブジェクト指向機能を導入したハイブリッド型またはマルチパラダイム型と呼ばれる言語である。ESP、S-LONLI、CLOS、TAO、ABCL、C++、OBJECTIV-C などがある。

一方、知識工学の分野でも、知識表現にオブジェクト指向概念を取り入れたエキスパートシステム構築ツールが開発された。

● 応用拡大期(1980 年代後半～1990 年前半)

1990 年前後から、GUI とその構築ツール、マルチメディアデータを扱えるデータベース、分散システムに対応する OS やネットワークなど、オブジェクト指向の応用が急速に拡大している。さらに、それらの開発支援のための設計技法や部品化再利用などのソフトウェア生産技術、あるいは、アプリケーションソフトウェアの共通のプラットフォームとしてのオブジェクト管理システムの開発などが積極的に行われており、オブジェクト指向ベースのソフトウェア開発環境が充実し始めている。

● 実用化時代（ 1990 年代後半以降)

現在の家電製品の多くはその制御のためにコンピュータが組込まれている。しかし、それぞれのコンピュータを動作させるための言語や規格がまちまちなので、使用者は非常に不便さを感じていた。ここに目を付けたアメリカのワークステーションメーカであるサンマイクロシステムズは、家庭内のすべての家電製品を誰でも簡単にコントロールできる装置の開発プロジェクトを始めた。

このプロジェクトでは、家電製品に組込むための小さなコンピュータを設計したが、その開発に使用するプログラム言語は、既存のものでは問題点が多く、使用できるものがなかった。

そこで、これまでに開発されたプログラム言語の良いところだけを取込み、使用頻度の低い機能やわかりにくい機能を排除したシンプルなプログラム言語を開発した。これが、JAVA の前身である OAK である。

このプロジェクトでは、1992 年に試作機を完成させたが、商品化までは至らずに解散した。

そして、インターネットが一般社会へ急速に普及し始めた 1994 年に、インターネット上で利用できるように改良され、JAVA と改名された新しいプログラミング言語として登場した。

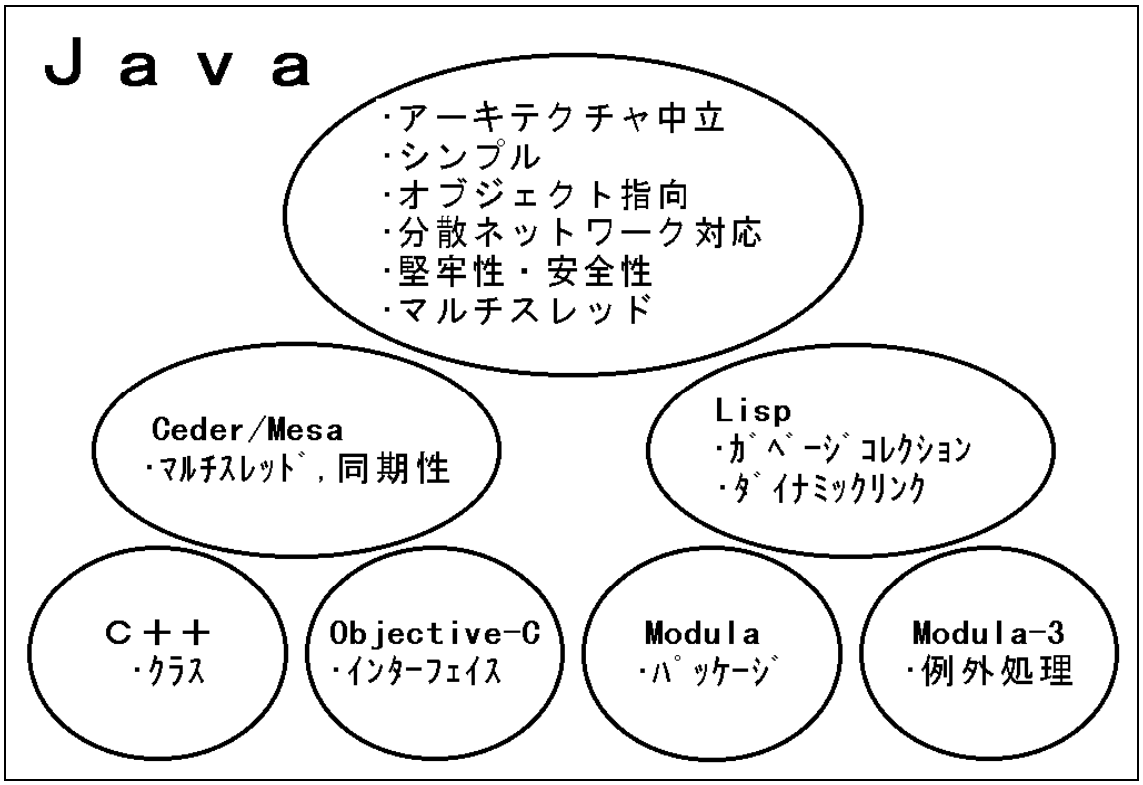


図 2-4 JAVA とは

新しいオブジェクト指向言語としての JAVA が、これまでに述べてきたオブジェクト指向言語であることの利点以外に、具体的にユーザにどのようなメリットをもたらすのか簡単に述べる。

- クロス・プラットフォーム性

  一度コンパイルするとプラットフォームを選ばないことを意味している。従来、開発者はコンピュータを囲む環境が変わる度に移植のための苦痛を強いられてきた。

  ユーザもその度に新しいソフトウェアのリリースを待たなければならなかった。JAVA が普及するとどのような OS でも動くようになるため移植に要する工数を新規の開発に振り向けることができる。
- 先進的な言語仕様
- JAVA は GUI やネットワークをシームレスに言語仕様に取り入れているため、JAVA 以前の GUI ツールのようにプラットフォームや OS に依存することがない。また、従来のように新しい OS が発表されるたびにそれをサポートする GUI モデルに習熟する必要がないため、「一度コードを記述した

らどこででも動く」というJAVAのメリットを最大限に享受することができる。

- JAVAのオープン性

  JAVAは非常に中立的で仕様がインターネットで公開されているため、多くの開発者から注目されている言語でもある。このようなオープン性のため、特定のベンダの製品戦略に影響を受けることが少なく、独自に開発戦略を維持することができる。

- インターネット時代の言語

  JAVAアプレットは作業が簡単で使用実績も多く、アプリケーションとブラウザ上のアプレットを統一した言語仕様で扱うことができる。また、インターネットに限らず、サーブレット（サーバーアプリケーション）やネットワーク関連のクラスが豊富に用意されているため、ネットワーク対応がやりやすい言語となっている。

- 現在の動向

JAVA 言語が登場して以来、オブジェクト指向技術は劇的な変化を示している。プログラミングレベルでは共通部品化や部品の流通ビジネスが出始めている。また、開発方法論においては、UMLの拡張やユニファイド・プロセス（統一プロセス）、フレームワーク、デザインパターンが実際のシステム開発において効果を上げている。また、当報告書が主眼においているビジネスオブジェクトについても、米国をはじめ我が国でも団体活動が行われている。ビジネスオブジェクトの動向については、第 2 章で将来を述べている。

今後のオブジェクト指向技術に関する大きな流れは、ITの進化に伴って、オブジェクト指向技術もその適用範囲が拡大される可能性がある点に着目したい。

例えば、ブルーツースやユビキタス・コンピューティングは、家庭のサーバや家電製品、携帯電話、個人が装着する腕時計、心拍計測装置などの実社会におけるオブジェクト群を取り扱う必要性を求めてくる可能性がある。

これまでのオブジェクト指向技術は、もっぱら企業や教育機関内での活用が主な対象であったが、今後は人間の生活にも適用される可能性が出てくる。

2.1.5　わが国の状況と本調査の目的

オブジェクト指向技術がソフトウェア開発においてどのような可能性を秘めているのかについては、これまでの報告でおおよそ理解して頂けたものとする。

米国はもちろんのこと、インターネットの普及が急速に拡大していることもあり、オブジェクト指向技術の可能性についての認識は、韓国、中国、インドなどでは確実にこれからの技術であるとの評価をおこなっている。

すでにこれらの国における情報工学関連の学部では、JAVA と UML に的を絞ったカリキュラムを実施している。

では、わが国におけるオブジェクト指向技術への取り組みは、どのようになっているのか、について、おおよその外観をここで紹介するとともに、本調査の目的について述べる。

2.1.5.1　わが国のオブジェクト指向技術に関する状況

日本国内におけるオブジェクト指向技術に関する状況を調査した資料は非常に少ない。それだけ技術として難しいのか、あるいは関心がないのか、いずれにしても定量的に状況を把握することが難しく、定性的な評価にならざるを得ない。

しかしながら、全く異なるアプローチとしてインターネットを利用した調査を試みることができる。

この調査方法の妥当性に関する仮説は次の通り。

- Web 上に掲載されている情報は、情報の共有を目的としているもの、商品の販売を目的としている物、情報発信を目的にしているもの、などであり少なくとも話題性があるからこそ WEB に掲載されるものとする。
- Web 上でキーワードをもとにヒット件数を調べることによって、関心の深さがキーワードごとに把握することができる。
- Web 上の検索エンジンが数える単位、すなわち今回の分析で指標とするデータとしての計数は、HTML のページ単位が基本となっており、キーワードの意味領域はページごとに集約される傾向にある。従って二つのキーワードの論理和で検索しても、片方のみがカウントされるといった冗長性は誤差の範囲と考えられる。

この仮説を前提に、国内の WEB サイトに対して、次のキーワードで検索を行った。

- 一般的な技術に関するキーワード

情報工学
ネットワーク技術
経営工学
オブジェクト指向技術
データベース技術
ビジネスオブジェクト
ソフトウェア工学

- 言語に関するキーワード

C++
SMALLTALK
COBOL
JAVA
Delphi

- 「オブジェクト指向技術」というキーワードの中で、普及度とか関心度に関するキーワード

教育
技術者
商品
書籍
論文
研修
技術情報
勉強会

- オブジェクト指向言語に関するキーワード

C++
SMALLTALK
COBOL
JAVA
Delphi
C#

これのキーワードをもとに検索した結果は次の通りとなった。

キーワード	検索結果
情報工学	1,151,000
ネットワーク技術	60,800
経営工学	15,200
オブジェクト指向技術	5,660
データベース技術	5,330
ビジネスオブジェクト	2,470
ソフトウェア工学	1,260

この結果を見ると、情報工学の意味が広いためか、他のキーワードと比較するにはあまりにもデータ量が多い。経営工学という歴史のある学問と比較しても、如何に情報工学すなわち IT に感心が高いかが伺える。さて、オブジェクト指向技術の位置は、ネットワーク技術の約 9%程度、これはデータベース技術とほぼ同程度であり、まだ良くは知られていない言葉と判断できる。また、オブジェクト指向技術はネットワーク技術と同程度に感心がもたれている事については、言葉の定着度からすれば、健闘していると思われる。

今回調査対象のビジネスオブジェクトは、2470 件のヒットとなり、オブジェクト指向技術の 50%を占めている。関心が高いと考えられる。

次にオブジェクト指向言語について検索を行った。

キーワード	検索結果	%
JAVA	734	31%
C++	493	21%
SMALLTALK	404	17%
COBOL	344	14%
DELPHI	263	11%
C#	154	6%

このデータで注意しておく必要があるのは、キーワードの片方に「オブジェクト指向」とする場合と、「オブジェクト指向技術」とする場合では検索結果が大きく異なる点になる。

例えば、C＃はオブジェクト指向で検索した場合、数千件ヒットする。これは、オブジェクト指向に技術を付けるか付けないかといった単純な話ではなく、「技術」をつけることで「オブジェクト指向」の意味領域を限定する行為が、文章を書く側の意識として、歴史的なオブジェクト指向の深さを知っているかどうかと言った認識の差につながっている。また、オブジェクト指向は広義では、オブジェクトを中心においた実世界の観察方法を示すが、オブジェクト指向技術は、エンジニアリングを意味している。この意味では、オブジェクト指向は、科学か技術であり、オブジェクト指向技術は工学の意味になる。これははっきり区別しておく必要がある。

過半数を JAVA が占める結果となった。オブジェクト指向技術においては、JAVA が最も興味を持たれているといえるのではなかろうか。

最後にオブジェクト指向技術のキーワードと動向に関連したキーワードで検索した結果を示す。

キーワード	検索結果
教育	1330
技術者	1130
商品	1080
書籍	817
論文	538
研修	467
技術情報	323
勉強会	97

オブジェクト指向技術と同じ価値を持った他の技術が見あたらないため、これらの数字がどのような位置付けになるのかは明確にできない。しかし、教育に関する関心の高さ、同様に人材と商品が他のキーワードに比較して高い数字を示しているのが分かる。

オブジェクト指向技術は、日本において修得段階にあると推測される。

3. ビジネスオブジェクトを取り巻く状況

これまで述べてきたように、オブジェクト指向技術の将来性と可能性は、企業が抱える問題解決に対して有効な手段となるだろう。

実際に米国はもとより、日本国内においてもオブジェクト指向技術に関する多くの取り組みが行われている。

これらの取り組みの中でも、とりわけビジネスオブジェクトに関する取り組みについては、単なるソフトウェア生産の効率化に資するばかりではなく、これまでの企業が行ってきた情報システムの開発を根幹から変革する可能性を持っている。

ここでは、わが国におけるビジネスオブジェクトの取り組みについて現状を報告する。

3.1 オブジェクト指向技術の動向

3.1.1 ビジネスオブジェクトの歴史

ビジネスオブジェクトという言葉が最初に使われたのは、OMG（Object Management Group）に BOMSIG（Special Interest Group for Business Object Managenemt）が設置された 1993 年末となっている。

BOMSIG の議長だったロバート・シェルトン氏のレポートでは、ビジネスオブジェクトのコンセプトが必要となった背景について、急激な市場変化に対応するため、ビジネス・プロセス・リエンジニアリング（BPR）が企業の中で活発に進められる中で、企業間競争に勝ち残るためには、商品開発やサービスの迅速な提供を実現しなければならず、そのためには、ビジネス・プロセスそのものまでも市場から調達してきてもよいと考えている。もちろんその企業のコア・ビジネスは戦略システムであるから、外部調達することにはならない。

オブジェクト指向技術では、この実世界を人間が認識している通りに、オブジェクトでモデル化する。このモデル化の対象がソフトウェアの世界ばかりではなく、実際のビジネスの世界に適用するのがビジネスオブジェクトということになる。ビジネスの世界で扱うオブジェクトには、人や物、概念、場所などがあり、これらのオブジェクトを用いて、内包性はクラス、外延性はインヘリタンス、属性はそのまま属性という形でモデリングすることによって複雑な実社会を表現することができる。

ビジネスオブジェクトを用いることによって、企業のビジネスモデルを共有し、再利用することが可能になる。

ロバート・シェルトン氏の提唱するオブジェクトの定義には、以下の 3 つがある。

- アプリケーション・オブジェクト
- ビジネスオブジェクト
- テクノロジー・オブジェクト

アプリケーション・オブジェクトとは、ビジネスオブジェクトとテクノロジー・オブジェクトとの組み合わせ方を記述したものであり、ビジネスオブジェクトとは、顧客とか注文、製品などをオブジェクトとして取り扱うもの、テクノロジー・オブジェクトとは GUI、ネットワーク、データベースなどのオブジェクトを扱う。

ビジネスオブジェクトとは、人、物、場所、概念などをオブジェクトの形で表したものになる。

これまでのオブジェクト指向技術に基づくプログラミングでは、オブジェクトとは、上記の内のテクノロジー・オブジェクトに該当している。テクノロジー・オブジェクトの具体例としては、物理的機能としての、オペレーティング・システムやファイル、入出力、ディスプレイ処理等オフジェクト、クラスライブラリとしての、ディストリビューション・サービス、ディレクトリ・サービス、メッセージング・サービスなどがあり、テキスト処理やグラフィック処理、マルチメディア処理、分散コンピューティングなどを実現する。

アプリケーション・オブジェクトは、個別アプリケーションのことを意味している。例えば普通預金への入金や注文入力などがこれにあたる。

ロバート・シェルトン氏によると、アプリケーション・オブジェクトとは、GUI の操作部分、クライアント・ソフトあるいはプレゼンテーション機能をもつソフトとしている。これは特定の業務領域をカバーする企業内のオブジェクト・クラスの集まりとして認識され、ほとんど完結した、すぐ動くアプリケーションに近いものといわれている。

3.1.2　ビジネスオブジェクトがもたらすもの

1960 年代から企業はコンピュータによって設備の自動化、事務の自動化、ネットワーク技術による時間と空間の短縮、さらに戦略情報システムによる他企業との差別化を追求してきた。

これまでに企業が行ってきた情報化投資は、1960 年代、1970 年代、1980 年代とおおよそ 10 年ごとに大規模な再構築と新機能実現を果たしてきた。しかし、1990 年代初頭のバブル経済崩壊によって、これまでのような情報システムに対する大型投資は事実上実現できなくなった。

しかし、経営が求める情報システムの機能は、日増しに高度化しシステムの構築に要求される期間は益々短くなってきている。さらにこれまでに構築してきた既存のシステムに対する保守の工数も増えることはあっても減る傾向にはない。現在の企業が抱える問題は、天井知らずの予算による理想的なシステムづくりではなく、短期間で安く機能も優れたものを作らなければならないと言った厳しい要求環境にある。さらに保守

のコストもこれまで以上に手のかからない安いものでなければならない。

情報システムには経年劣化がある。長い間保守作業を続けていると、プログラムの論理構造が崩れ、いわゆるスパゲティプログラムになるのがその原因で、工業製品のライフサイクルとして表現されるバスタブ曲線がこれにあたる。

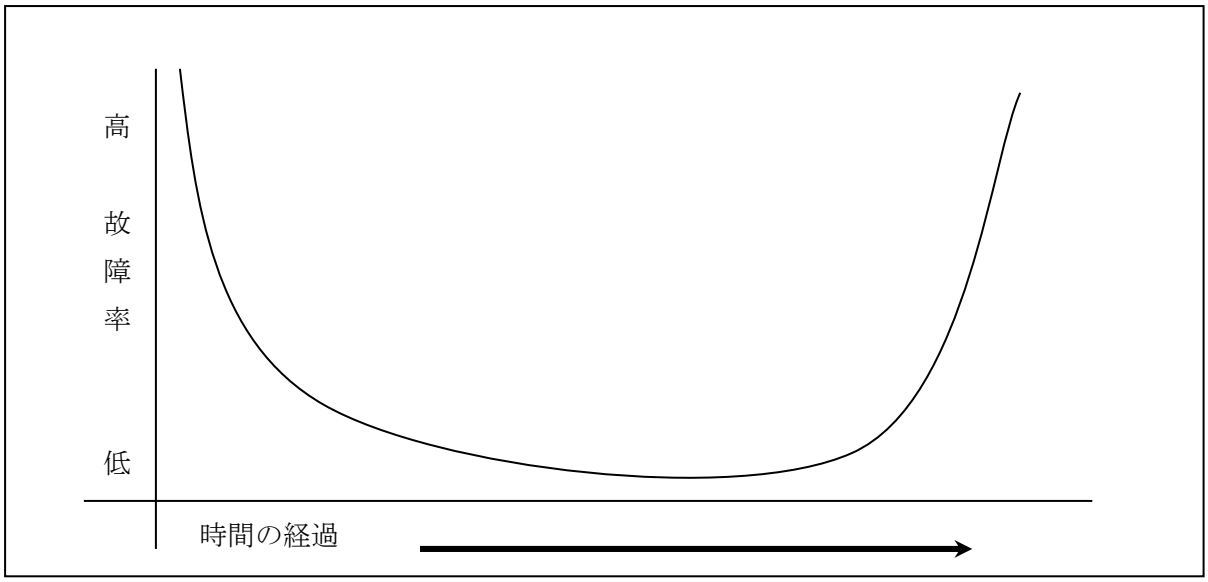


図 3-1　バスタブ曲線

このために、情報システムはおおよそ 10 年ごとに再構築されてきた。再構築の目的には、システムの経年劣化だけではなく、新しい情報化技術の導入や設備の老朽化問題もある。しかし、新しく構築したシステムが年を重ねるごとに保守が複雑になり、それに伴って保守のコストが増大し続けるのは、企業にとって頭痛の種となる。

システムの再構築に関しては、次のようなことが明らかになっている。業務システムの場合、再構築に必要な業務要件は、70%が従来通りであり、30%が不要なもの、新規に追加するのが 30%という数字がある。(R･E Shelton “Business Objects White Paper”: Http://www.opening.com/bnsobj.hmtl)

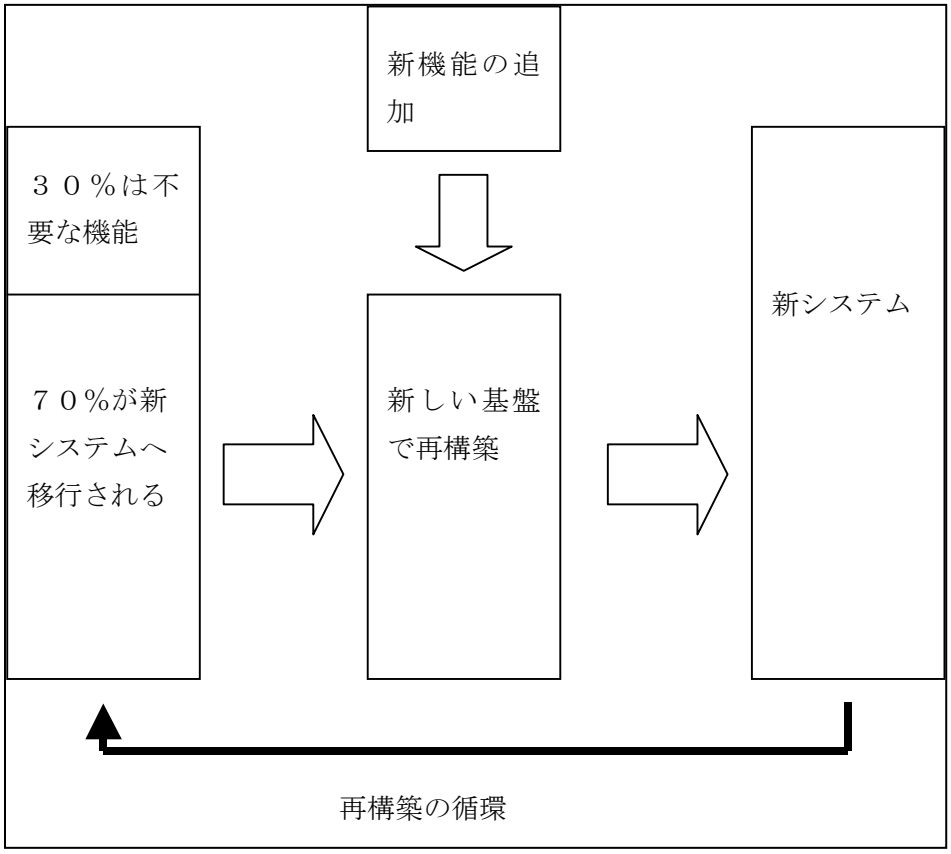


図 3-2 情報システム再構築に繋がる流れ

システムの再構築に関するこの経験則は、企業の業務システムの内、70%は普遍的、あるいは安定したものであり、基盤システムに依存しなければ長期間にわたって使い続けることができる可能性を示唆している。

またこの 70%の企業内で普遍性をもった機能は、企業間でも共有できる可能性がある。

特に日本国内の企業では、情報化投資コストの内、70%が保守のコストだとされている。ちなみに、企業の情報化投資額は売上額の 1%程度とされているから、売上 1 千億円の企業であれば 10 億円が情報化投資にあてられ、この内の 70%にあたる 7 億円が保守に使われていることになる。残りの 3 億円が新規投資に割り当てられるのだが、情報システムが経営を支援するものである限り、新しい取り組みが情報化投資予算の 30%しかない状態で本来の戦略性をもったシステムの実現は、窮屈なものになってしまう。

保守コストの削減こそが一義的に解決されるべき情報システムの課題といえるだろう。

わが国の情報化投資に対する経済効果、すなわち情報化投資が企業収益にどれだけ貢献したのかを示す数字について、経済産業省（旧通産省）から資料が提示されている。

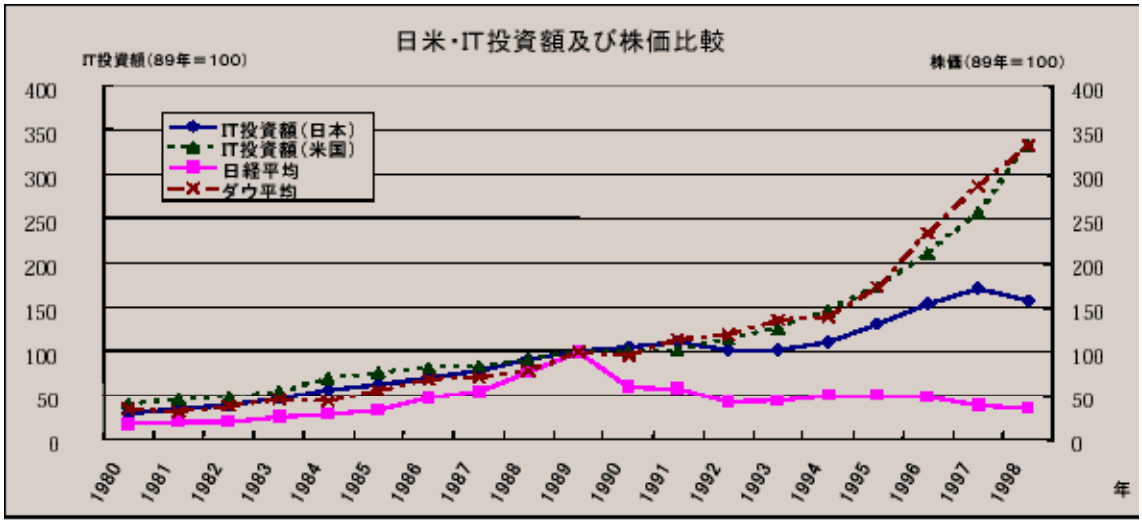


図 3-3　日米・IT 投資額及び株価比較

企業収益は株価に反映するので、日米の情報化投資額とダウ平均および日経平均を見ると、1989 年を境に情報化投資に対する経済効果が日米で大きく乖離しているのが分かる。

投資額に関しては日米とも同様の伸びを示しているが、株価平均は大きく差がでている。これは日本の情報化投資が企業の業績に結びついていないことも一因である。

日本独自の企業文化やコンピュータの活用技術に差があるのはもちろんだが、日米での情報化コストは、日本が約 70%を保守に費やし、米国は約 30%が保守コストという状況であるようで、効果的な投資が日本においては難しく、その原因が保守コストにあるといえるだろう。

このように、企業のおいて情報システムのコストを削減することが、戦略的な投資との関係からも喫緊の課題となっている。

ビジネスオブジェクトがもたらす福音は、新規の開発コスト削減はもちろんのこと、より保守のコスト削減に効果が期待されている点にある。

米国では、石油業界及び半導体業界において、オブジェクト指向技術をベースにしたシステム開発が多数試みられた。この開発実績から結論されたことは、システム構築に必要なオブジェクトの 80%はすべての企業で共通であり、残りの 15%は同業界であれば共通化できるとしている。（R･E Shelton “Business Objects White Paper”:

Http://www.opening.com/bnsobj.hmtl）

例えば、業界にビジネスオブジェクトが整備されていれば、情報システムの構築・運用・保守に必要なコストの 95%がきわめて安いコストで実現できることになり、これまでの開発費の 20 分の 1 に相当する 5%の投資でシステムの構築ができるようになる。

ビジネスオブジェクトが注目を浴びている理由の幾つかに、オブジェクト指向技術の発展が、共通化ルールとしての EJB の登場と、インターネットの発達、そしてハードやソフトウェアの高性能化と低価格化によってもたらされたものであり、そのビジネスへの応用が劇的なコスト削減をもたらす可能性がおおきいことにあることを第 1 章で述べた。

今後、ビジネスオブジェクトの標準化が進めば、取引の多くがその標準化されたネットワーク上の商品オブジェクトによって行われるようになるだろう。

3.1.3 ビジネスオブジェクトに関する最近の動向

ビジネスオブジェクトに関する最近の動向について、大きく 3 つのカテゴリに分類して報告する。

- ビジネスオブジェクトの生産技術に関するもの
- 企業の情報システム構築に関するビジネスオブジェクト分析設計の方法論
- ビジネスオブジェクトの再利用・流通に関するもの

まず、ビジネスオブジェクトの生産技術に関しては、フレームワークとコンポーネントベースの考え方がでてきた。オブジェクト指向技術におけるコンポーネントとは、再利用あるいは置き換えができるように構築されたモジュール群と定義できる。

コンポーネントの実装技術としては各団体から次のものが提供されている。

EJB コンポーネント	サンマイクロシステムズ
COM コンポーネント	マイクロソフト
. NET コンポーネント	マイクロソフト
CORBA コンポーネント	OMG
製品特有のコンポーネント	Coldfusion Coolplex など

図 3-4 コンポーネント提供状況

コンポーネントの種類も、部品としてどの業界でも使えるものとしてのユーティリティ・コンポーネントから、業務で共通に利用できる業務共通コンポーネント、さらに業務に依存したコンポーネントに分類できる。

ユーティリティ・コンポーネントとしては西暦和暦の変換部品や入力されたテキストのチェック部品などがある。業務共通コンポーネントとしては、金融機関の接続部分のコンポーネントやカード決済コンポーネントなどがある。業務に依存したコンポーネントとしては、会計や販売などがある。

コンポーネントを開発する場合、アプリケーションを構成するコンポーネントの青写真が必要となる。これをアーキテクチャと呼んでいる。アーキテクチャを実現するためには、フレームワークが必要となる。

フレームワークとは、各アプリケーションで共通に必要となる機能を提供する開発実行環境であり、アプリケーションの骨格を提供する。このフレームワークによってアプリケーション開発者は、アプリケーション固有の機能を実現する為のロジックを開発し、フレームワークにはめ込む。いわばフレームワークに穴があいており、それに開発したロジックを入れる格好になる。

コンポーネントとフレームワーク技術は、オブジェクト指向技術の難しさを緩和することができる。これらを開発するためには、高度な技術を使いこなせる優秀な技術者を集め、少数精鋭でチームを編成することによって成果が達成できる。このチームによって開発されたコンポーネントとフレームワークを利用して、短期間の教育を受けた一般的な技術者が、ソフトウェア部品を大量生産できるようになる。

ビジネスオブジェクトの生産技術に関する技術として、デザインパターンを最後にあげる。デザインパターンとは、オブジェクト指向技術を用いたソフトウェアの再利用のために、過去の開発経験の中から有効な設計パターンを集め、カタログ化したものを意味している。デザインパターンは Web 上で公開されており、だれでも参照することができる。これは、設計上の問題を解決するための有効な手段として利用される場合が多い。コンポーネントやフレームワークを構築する場合には、デザインパターンを組み合わせて行う。デザインパターンは、ユーティリティ・コンポーネントに近く粒度が小さい為に、柔軟性があり再利用性の高い設計ができる。

企業の情報システム構築に関するビジネスオブジェクト分析設計の方法論の動向としては、統一プロセスと UML が挙げられる。

統一プロセスとは、ラショナル社によって開発されたシステム開発方法論でであり、RUP(Rational Unified Process)と呼ばれている。しかし、この開発論がオブジェクト指向技術によるソフトウェア開発方法論の標準となりつつあることもあって、一般的には統一プロセス(ユニファイド・プロセス)と呼ばれている。

統一プロセスには、3 つの基本概念がある。

● ユースケース駆動
● アーキテクチャ中心
● 反復、インクリメンタルの活用

また、フェーズ毎にモデルを作成するため、成果物＝モデルという構図をもっている。

反復型の開発方法については下図を参照。

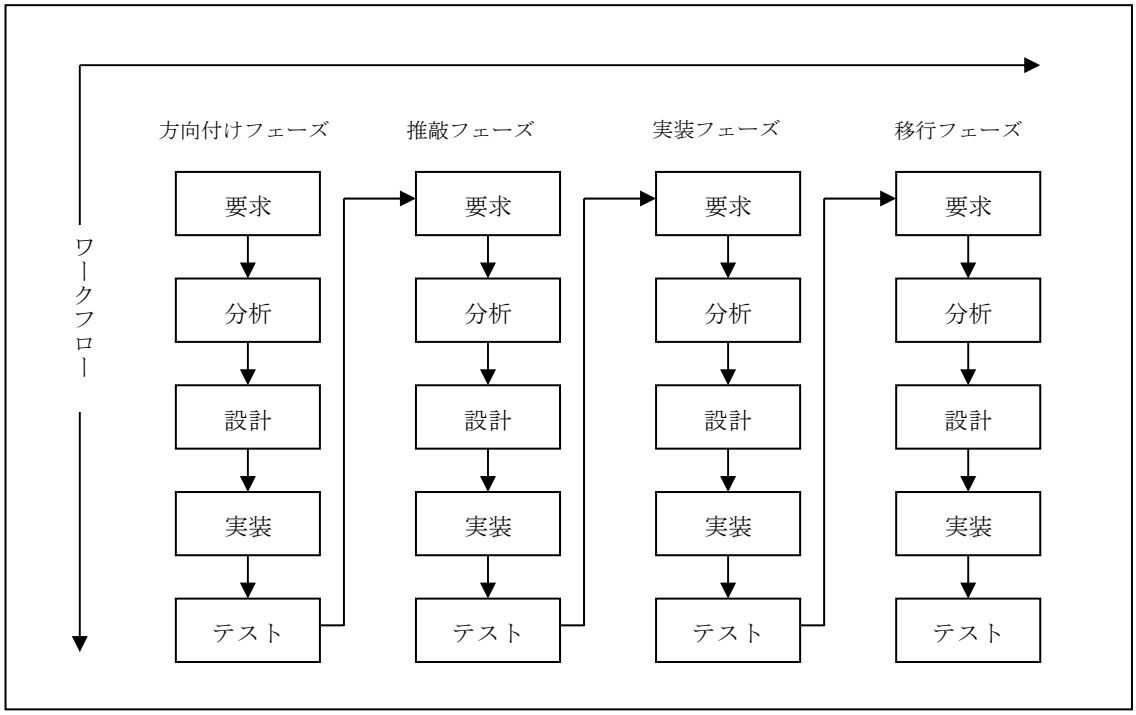


図 3-5　反復型の開発方法

統一プロセスのメリットは次の通り。

- ユースケースドリブンのアーキテクチャによって要求仕様を早い段階で獲得することができ、後続の工程を効率良く運用することができる。
- アーキテクチャセントリックのアーキテクチャによって理解しやすく、柔軟性が高く、堅牢で、拡張性の高いアーキテクチャをベースに長期間にわたる製品開発を進めていくことができる。
- イテレーティブ/インクリメンタルのアーキテクチャによって開発者とステークホルダ双方が開発中の製品に対して充分な経験を積むことを可能にする。
- リスクの高い機能を先に開発することにより、開発作業全体に対するリスクを最小限に押さえることができる。
- コンポーネントベースの開発によって開発スピードの向上、開発コストの

低減、品質の向上が得られる。

リスクの高い機能を先に開発することを統一プロセスではリスクドリブンと呼んでいる。統一プロセスがリスクドリブンな開発を行えるのは、ユースケースドリブンによりシステム機能を早い段階で獲得していること、アーキテクチャセントリックによってリスクの高い機能を早い段階で発見することができること、イテレーティブでインクリメンタルな工程によりリスクの高い機能の実装を早い段階でできること、といったユニファイド・プロセスのアーキテクチャの効果となっている。

さらに、統一プロセスの大きな特長として、開発チームやステークホルダ、顧客といったソフトウェア開発に関係するすべての人々の役割を明確に定義していることにある。このことを統一プロセスでは「工学化されたプロセス」としている。ビジネスの現場に適用するための開発方法論は、高い完成度を求められている。
以下にイテレーションとインクリメンタルな統一プロセスの構成図を示す。

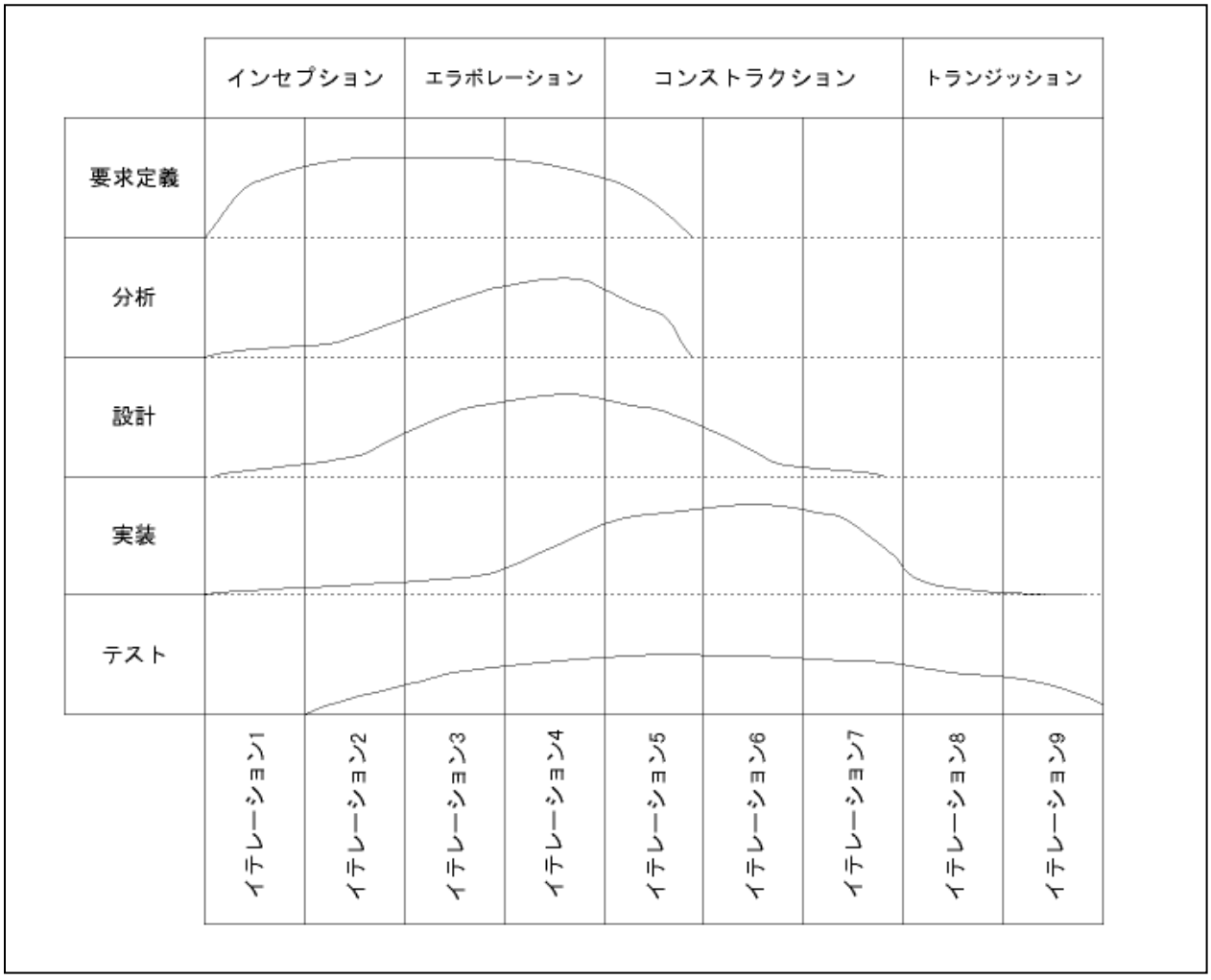


図 3-6　イテレーションとインクリメンタルな統一プロセスの構成図

3.2 オブジェクト指向の国内と国外の取組状況

3.2.1　海外の取り組み状況

ビジネスオブジェクトに関する海外での取り組みには、次のような団体がある。

- OMG(Object Management Group)

1989 年に設立された標準化団体で、ソフトウェア開発の生産性を向上させるオブジェクト指向モデリング、高い柔軟性を持った分散システム、新旧のソフトウェア資産を連携させる相互運用性、データリポジトリのメタデータ技術といった基盤技術、および各産業別の標準フレームワークの策定を行なっている。

OMG は、特定のソフトウェア企業に依存しない中立の非営利団体であり、オープンなプロセスによって各種標準を策定している。

NIIIP(National Industrial Information Infrastructure Protocols)

NIIIP 協会は、オープン・インダストリ・ソフトウェア・プロトコルの開発に関して、米国政府との共同開発契約に参加した組織の共同体であり、IBM をリーダとして結集した 18 組織からなる集団である。各組織は NIIIP アーキテクチャの定義付け、あるいは適用に必要な最新技術や最終ユーザとしての経験を提供する。

協会の構成は、Exective Board、Operational Management Borad、Technical Advisory Board および Development、Test & Interation、Deployment ならびに Operation の各委員会からなっている。

NIIIP は、ビジネスオブジェクトに関して、人が資源を共有し何らかのシステムを使用するという一般的な状況に現れる総称的なオブジェクト, すなわちグループ, ユーザ, 役割, ワークプレース, タスク, リソース・アダプターといった CBO（共通ビジネスオブジェクト）とそれらの関連を規定している。

STEP(Standard for the Exchange of Product Model Data)

製品モデルデータ交換規約(ISO 10303 製品データ交換規格)。CAD の製品データ交換のための国際規格製品のライフサイクル全体を通じて,CAD/CAM/CAE データを異なるシステム間でも交換可能にするファイル構造を規定している。

STEP では、規格の検討開始から 10 年以上の年月が経過して共通のリソース規格と,一部のアプリケーションプロトコル（AP）規格の ISO 標準化ができて、現在約 30 個が提案されている。しかし、次のような事情があって、標準化の進捗は思わしくない。

- 複数の AP 規格案の対象範囲に重なり合う部分がある場合,互いに整合性をはかる必要があるが,その調整が容易ではない。
- リソース規格と AP 規格の間では,当初から階層構造が定義されている

が,AP 規格のレベルでは,モジュール化や階層構造化が,トップダウンで行われていないため,数が増加するとともに,異なる性格の規格案が入り混じって雑然とした感じになり,応用分野全体あるいは産業界全体から見た場合に,過不足を判断しようとしても見通しがよくない。

- 10 年以上が経過する内に,情報技術と産業界ニーズの両方の面で大きな変化があり,当初の STEP の基本構想には納まりきらない動きが出てきている。
- CALS 標準を国際標準に合わせようとする動きの中で,STEP にも CALS 側からの要求が次々と出されている。これらは,これまで先行してきた形状データや,部品の構成管理データを中心とするものに対して,製品のライフサイクル 全体を支援するための,管理データを中心とするものに重点をおいたものとなっている。

以上のような状況を打開するために、STEP では新たな規格体系として適応させる以下の活動を行っている。

- STEP の AP 規格のモジュール化や階層構造化の方法を検討する。
- ビジネスオブジェクトの概念に合わせた,STEP の体系を検討する。
- CALS 側からの要望を受けて, 製品のライフサイクル支援 のための,あるべき規格体系を検討する。
- STEP と SGML を連携させて,適用範囲を拡充する方法を検討する。
- 以上のような要望に必要な,リソース規格の改定を検討する。

ビジネスオブジェクトと STEP との関わりは、2 番目のビジネスオブジェクトの考え方を STEP に取り入れようとする試みにある。

以上見てきたように、ビジネスオブジェクトを全面に掲げて活動している海外の団体は OMG が最も活発と思われる。

従って、OMG の活動について理解を深めることは、ビジネスオブジェクトの動向を把握する上でも重要な観点と考えられるので、次に OMG についての全体像を追ってみる。

3.2.2 OMG とビジネスオブジェクト

ビジネスオブジェクトの標準化は、OMG の BODTF(Business Object Domain Task Force、以前は BOMSIG と呼ばれていた)を中心としてすすめられている。

1996 年に BODTF は CBOs(Common Business Objects)と BOF(Business Object Facility)に関して、それぞれの定義を行い、これらがどのようにあるべきかの

提案を求めた。

CBOs とは、ほとんどのビジネスにおいて共通であるビジネスセマンティクスを表すオブジェクトであり、BOF とはビジネスオブジェクトの操作をサポートすることが要求される内部構造を表している。OMG と BODTF とによるビジネスオブジェクトの共通理解は次の通り。

- ビジネスオブジェクト
  - 実世界のビジネスにおける情報の概念，およびその概念におけるオペレーション，それらオペレーションによるモデル表現
  - 実世界のビジネスにおける情報の概念と他ビジネスコンセプトの間の関係
  - ビジネスアプリケーションはビジネスオブジェクトの実装を介して相互作用する

この定義に従えば、ビジネスオブジェクトは経営工学とソフトウェア工学でそれぞれ次のような意味で使われる。

- ビジネスオブジェクトのモデル化は，ビジネスモデルにおけるビジネスオブジェクトの使用範囲を，「ビジネスの概念を捉えること」と「ビジネスの抽象的なビューを表現すること」だと明示する.
- システムビジネスオブジェクトは，ソフトウェアシステムの設計やプログラムコードの基盤であり，ビジネスの概念の実現を反映する.

さらに、ビジネスオブジェクトは以下の 3 種類に分類される。

- Entity Business Object(製品，注文，商品，etc)
- Process Business Object(受注，仕入れ，支払い，etc)
- Event Business Object(会計年度の終了，契約の期日，etc)

OMG の CBO における主目標は以下の通り。

- ビジネスオブジェクトの相互運用は特別な統合の可能性を含んでいる。
- 設計や実装，構成や配置が平均的な開発者にも可能なように簡易であること。

ここで言う、「特別な統合」とはビジネスオブジェクトコンポーネントがプラグアンドプレイでなければならないことを意味している。CBO のビジョンは市場におけるビジネスのモデル化あるいはビジネス・プロセスのモデル化、さらにシステムビジネスオブジェクトの標準化を目指している。

OMG においてこれらの活動がどのような背景をもって行われているのかを整理する。

ビジネスオブジェクトよるアプリケーション構築という考え方が浮上してきた背景は 2. で述べているが、情報技術の観点から整理するとつぎのようになる。

- 三層アーキテクチャや Web 三層アーキテクチャなど、オープンな世界が広がり、それとともに分散オブジェクトという考え方が出ていている。
- ビジネスのモデリング技法としてオブジェクト指向技術が普及しつつある.
- インターネットとの親和性をもったJAVA言語とそのアプリケーション開発環境が, プラットフォームから独立性、言語の簡便性, などの特性から急激に普及している。
- 分散オブジェクトの標準化活動が, ビジネスに視点をおいた標準化に軸足を移している。

特に、分散オブジェクトの標準化が次第にビジネスの世界に視点を移している点は、共通ビジネスオブジェクトおよびビジネスオブジェクト・ファシリティの標準化に直接結びついている。

ビジネスオブジェクトが標準化されると、従来のシステム開発手法がソフトウェア部品としてのコンポーネントを組み立てる方式に取って代わる可能性は高い。

2.で述べたように、R. E. Shelton が紹介している「特定のビジネス概念や構造の 80% が複数の業界で共通であり, 特定の業界のビジネス間では 95% が共通である」という認識を前提とするなら、ERP パッケージなどのベンダが, 標準のビジネスオブジェクトを再利用可能なコンポーネントとして実装し, それらの組み立て品としてパッケージが構成されることが可能となってくる。

このようなシステム開発の方法論が大きくパラダイムを変化させることによって、顧客は異なるベンダ製品を部品単位で利用・置換できるようになる。さらに、分野ごとに特化した質の良い部品の流通も実現できるようになる。

また、ビジネスオブジェクトの相互に関連し合った構造を整理し標準化することによって、ビジネス・パターンを求めることができる可能性がある。

オブジェクト指向技術を用いたシステム開発では, ビジネス領域のモデル化をまず行うが、この作業は、オブジェクト指向の難しさが最も反映される。仮にビジネス・パターンが標準化されれば、この複雑で難解な作業の簡素化と品質の向上が実現されることになる。

つぎに、ビジネスオブジェクトとビジネスオブジェクト・ファシリティの関係について紹介する。

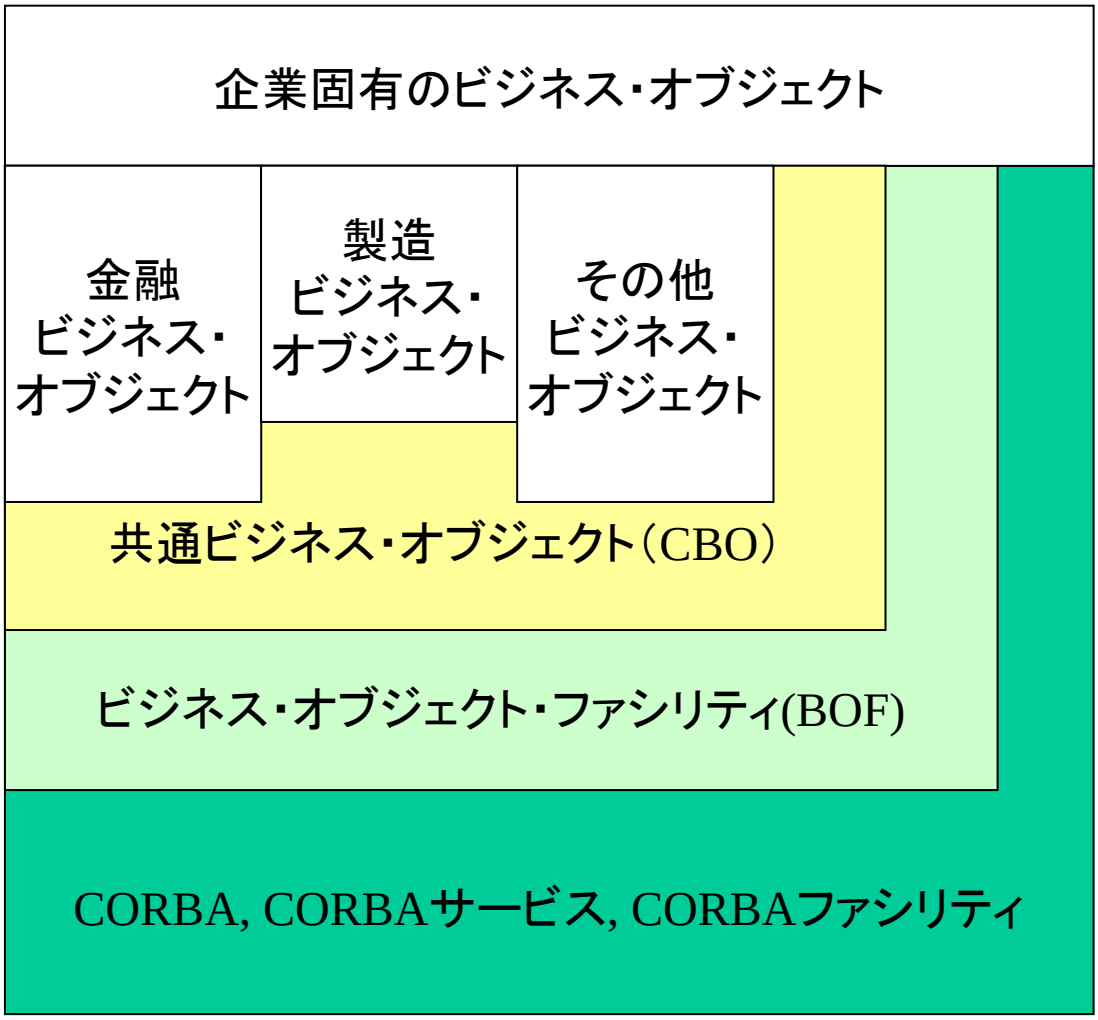


図 3-7　OMG が示すビジネスオブジェクト全体の構造

OMG が示すビジネスオブジェクト全体の構造は図のようになっている。一番上層に位置するのは、企業固有のビジネスオブジェクトであり、これを支える形で、業種ごとのビジネスオブジェクトがある。これをドメイン・ビジネスオブジェクトと呼んでいる。さらに、ドメイン・ビジネスオブジェクトを支えるものとして、共通ビジネスオブジェクト（CBO）がある。これは上位のユーザ・アプリケーションによって再利用される。これらの上層オブジェクトを支援するのが、ビジネスオブジェクト・ファシリティ（BOF）の役割であり、ドメイン・ビジネスオブジェクトと共通ビジネスオブジェクトに対して、最下位に位置付けられる分散基盤との間で、相互運用の役割を担う。

最下部はソフトウェア・プラットフォームを表し，分散基盤としての CORBA や共通オブジェクト・サービス（COS）群などから構成されている。

OMG におけるビジネスオブジェクトに関する活動のひとつとして、オブジェクト分

析・設計タスクフォース（OA & DTF）がある。また、これ以外にもいくつかのドメイン・タスクフォース（DT）が実質的に進められている。

OA&DTFでは、UMLとメタオブジェクト・ファシリティ標準（MOF）の確定を行う作業を進めている。メタオブジェクトとは、オブジェクトの上位置に位置する概念的なオブジェクトで、オブジェクト構造と属性に関して、抽象度の高いレベルでのオブジェクトを表現する。さらに、UML はオブジェクトモデリングを行うための標準図式記法を提供する。MOF は，リポジトリのメタモデルを表すと同時に，UML のメタモデルなど他の各種メタモデルを統制する上位のメタモデルを提供する。

ドメイン・タスクフォースは，縦割りのビジネス領域に対応している。製造，金融，通信、医療などの各タスクフォースと，これらに共通するビジネスオブジェクト・ドメイン・タスクフォース（BODTF）があり、各ビジネス・ドメイン固有のビジネスオブジェクトについて標準化作業を担当している。さらに、BODTF は CBO と BOF の標準化，およびワークフローの標準化も担当している。

1996 年初頭に CBO と BOF に関する提案要求（略称 CFRFP−4 現在は BODTF RFP−1 ）が発行され，1997 年 1 月にいくつかの具体的な提案仕様が提出された。

初回に提案された仕様は、以下の 8 項目となっている。

- 相互運用性
  コンポーネントは，ビジネスセマンティクスの実装，エンジニアリング技法，ソース言語，OS，各ビジネス・ドメインなどから独立して互いに協調できなければならない。

- 拡張性
  ビジネスオブジェクトもコンポーネントも任意のユーザ企業で使えるように拡張できなければならない。そのための手段を備えていること。

- 再利用性
  多様なアプリケーションで再利用できなければならない。ビジネスオブジェクトは大きな市場性を持つ必要がある。

- 開発と導入の容易性
  ビジネスオブジェクトの開発や利用，情報システムへのそれらの配備が一般のシステム開発者やユーザの観点から容易に行えるものでなければならない。

- セキュリティ
  オブジェクトに関する情報の機密が保たれ，セキュリティ・ポリシが励行できるよ

うになっていること。

- レガシー連携
  レガシー・アプリケーションとの統合が容易であること。

- メタデータ
  データを規定するメタ情報すなわちメタデータのサポートが行われていること。

- 多言語サポート
  国際化を前提として他言語のサポートが行われていること。次に、略称 RFP−4（BODTF RFP−1）について述べる。RFP−4 に対する BOF 標準は現在次の二つの仕様に集約されている。これは現在も改訂が続行され同時に評価作業も行われている。
  - ビジネスオブジェクト・コンポーネント・アーキテクチャ（BOCA）
  - 相互運用フレームワーク（Interoperability Specification）

  その二つの仕様は相互に補完しあっており、二つを合わせて Combined Business Object Facility（CBOF）と呼んでいる。CBOF の中でも特徴的な機能について幾つかを述べる。

- ビジネスオブジェクト・コンポーネント・アーキテクチャ

  これは、ビジネスオブジェクト・コンポーネント・アーキテクチャ（BOCA）とコンポーネント定義言語（CDL）のふたつによって構成されているもので、BOCA は CBOF 全体を統制する。CBOF はメタモデルであり、すなわち、CBOF で用いられる多様な概念を規定するメタレベルのアーキテクチャに位置づけられる。CDL は，インタフェース定義言語であり、ビジネスオブジェクトをコンポーネントとして記述する役割を持っている。

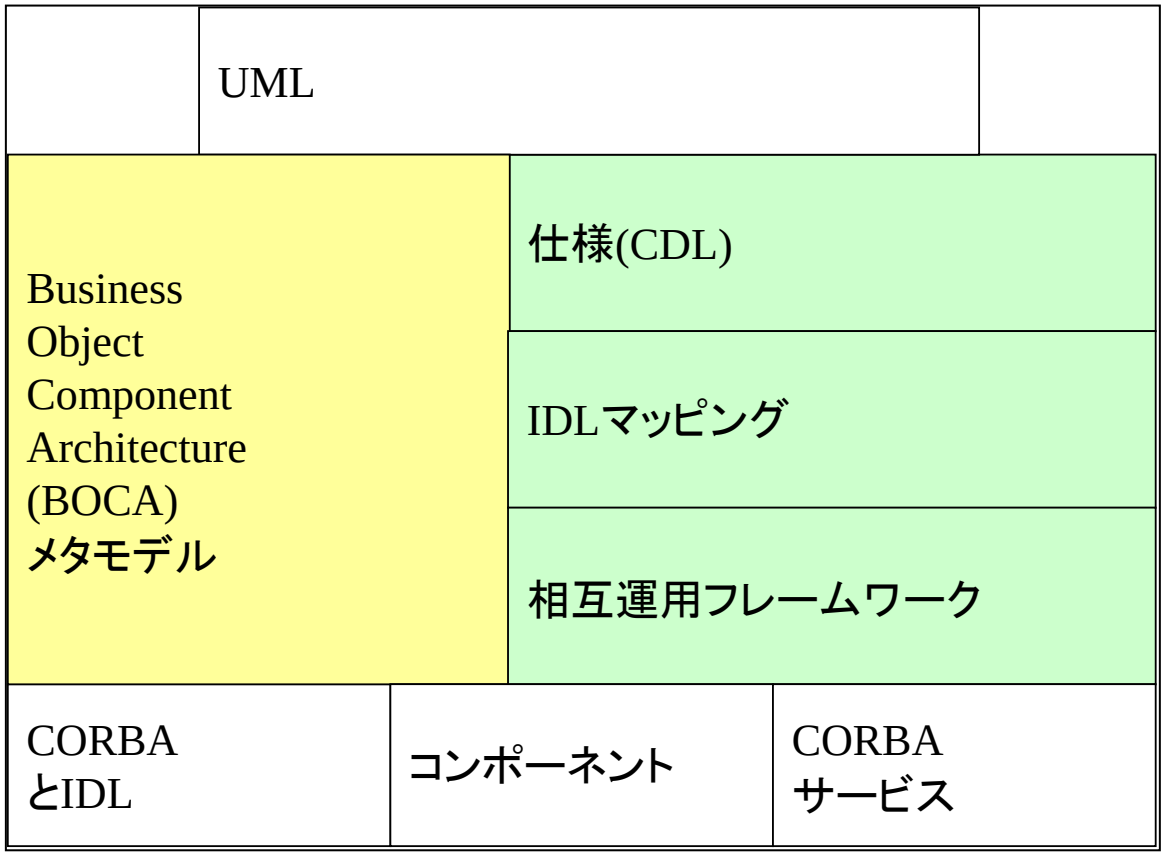


図 3-8　ビジネスオブジェクト・コンポーネント・アーキテクチャ

● 相互運用フレームワーク

このフレームワークは実行時のファシリティとして位置付けられている。CDL で記述されたコンポーネント群が異なった分散環境内で効果的に相互運用するためのアーキテクチャとサービスから構成されている。基盤部分に CORBA と共通オブジェクト・サービス（COS）があり、このフレームワークはその上の層を形成している。

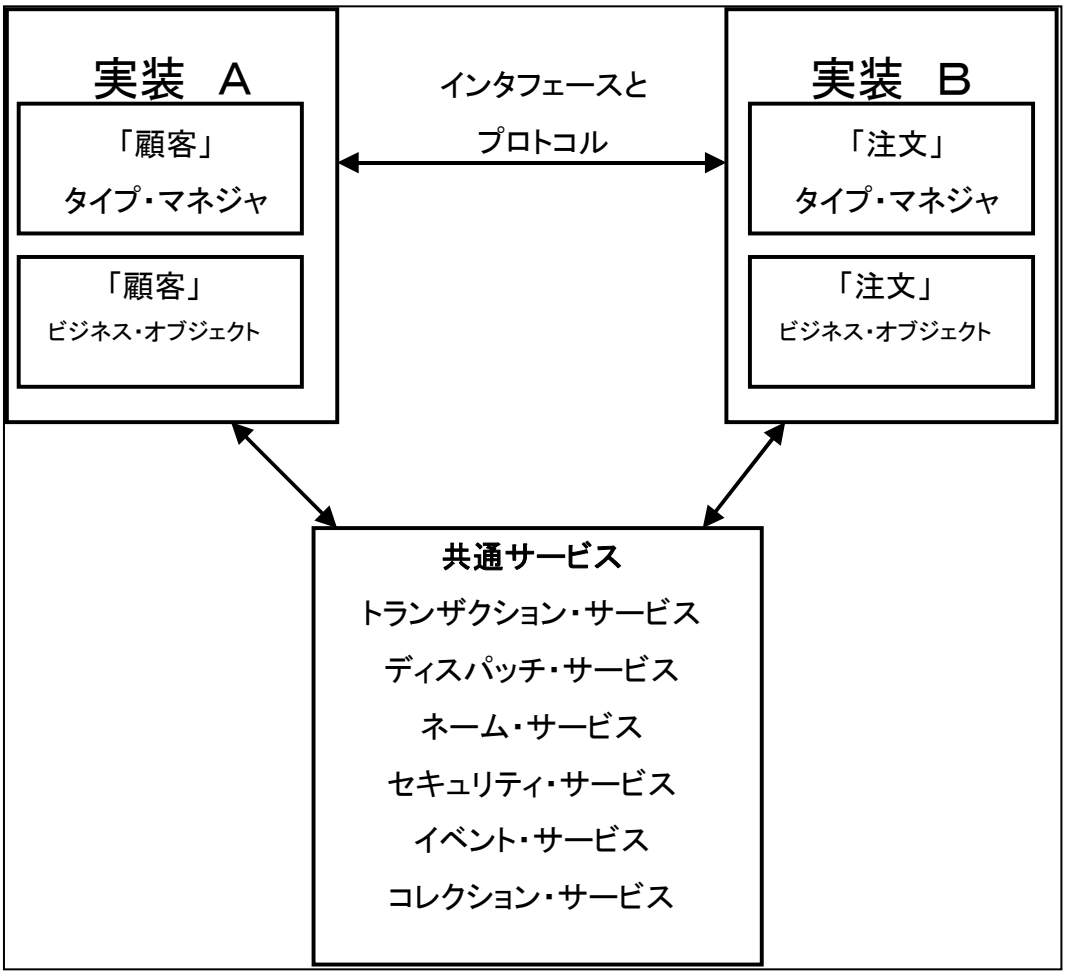


図 3-9 相互運用アーキテクチャ

タイプマネジャは，ビジネスシステムドメイン内に位置する各々ビジネスオブジェクトのタイプごとに存在する，タイプあるいはサブタイプのすべてのエクステントやサブエクステントと言った、いわゆるインスタンスを統制するテクノロジオブジェクトとして存在する。タイプマネジャは分散オブジェクト環境において、オブジェクト・ファクトリ機能、クラス属性，クラス操作などを動的に扱うためのアーキテクチャとなっている。タイプマネジ

ャの主要機能を以下に示す。

- ライフサイクルの管理
- 問い合わせ
- メタデータへのアクセス
- イベント通知

共用サービスは，実装されたフレームワークの幾つかの群が共有するサービスの集まりとなっている。ビジネスオブジェクトはトランザクションに参画し，かつ共用サービスと連携し続けているオブジェクトでなければならない。

共用サービスには次のものが含まれている。

- トランザクション・サービス

  ディスパッチ・サースと協調して，トランザクションの一貫性や唯一無二の単位であることを保証するメカニズムをもつ。分散環境における複数のフレームワークに所在するビジネスオブジェクトを横断的に扱う。
- ディスパッチ・サービス

  ここれはトランザクション内での操作の実行タイミングをきめ細かく制御するためのメカニズムを持つ。例えば、更新を伴わない操作のコミット処理を、あらかじめ影響範囲を調べるために、すべてのオブジェクトの最終状態を検証する作業などをおこなう。
- ネーム・サービス…COS のサービスの拡張機能になる。
- セキュリティ・サービス…基本的に COS のサービスに従う。
- イベント・サービス…コンシューマによるイベントの登録とイベント発生時でのイベント通知を行う。
- コレクション・サービス…集合体を扱うサービス。相互運用アーキテクチャに参加するオブジェクト全体を扱うサービス。

3.2.3　ビジネスオブジェクト標準化動向

これまで述べてきた OMG によるビジネスオブジェクトの標準化作業は、基本的にコンポーネント間の相互運用に着目している。しかし、実際にコンポーネントを実装するレベルになれば、異なる BOF 上で稼働するビジネスオブジェクトのバイナリあるいはソースコードの移植性が問題となってくるが、ここまではまだ標準化が進んでいない。

いずれにしても、実装を前提としたアーキテクチャが今後提案されるものと思われる。

OMG はあくまでも標準化の団体であり、自らが提案することはなく、会員企業から

の提案を受けて、標準化に取り入れるかどうか決め、SIG を発足させる、単純に言えばそれだけの機構にすぎない。OMG の会員企業は、大半がベンダであり、ベンダの提案は顧客企業のニーズに根ざしている。

これまで述べてきたように、ビジネスオブジェクトの標準化は必然の流れであり，ユーザ企業を支援する各ベンダが、統一されたビジネスオブジェクトの標準化に熱心なのは、企業利益と顧客利益が一致することにある。ユーザがビジネスオブジェクトの標準化を望む限り、OGM のビジネスオブジェクトに対する標準化作業は、今後も積極的に行われることになるだろう。

3.2.4　ビジネスオブジェクト標準化の課題

共通ビジネスオブジェクト（CBO）は概念として理解することは可能となっているが、具体的に「共通」とは何を意味して何を指し示たものかをきめるのは、意外と難しい。オブジェクトとして扱う対象には、日付、時間、通貨、などといった具体的なオブジェクトもあれば、ビジネスの世界であつかう人や組織、顧客、商品、在庫などといった非常に抽象度の高いオブジェクトもある。

これらを共通に認識するとは、企業内、業界内、国内、海外と領域がひろくなればなるほど、共通に認識する事が難しくなる。どの領域で合意をとって共通とするかは、技術論よりもむしろ意味論の世界に近い。これを一元的に取り扱えるようにするにはさらに時間が必要となるだろう。

3.2.5　ビジネスオブジェクトの国内の取り組み

日本国内におけるビジネスオブジェクトに関する取り組みを調査した。国内といっても非営利団体から私企業におけるまで広範囲に及ぶので、今回は、公的機関を中心とした活動に焦点を絞る。

3.2.5.1　情報処理進行事業協会（IPA）の取り組み

IPA（情報処理進行事業協会）では、ビジネスオブジェクトに関する取り組みを 1999 年から行っている。これは、「ビジネスオブジェクト関連システム開発事業」として、ビジネスオブジェクトの具体的なプロジェクト提案を求める形となっている。この結果は、2001 年度に成果として報告されており、その一覧を次に示す。

ビジネスオブジェクト関連システム開発事業

ASP による CRM に必要なソフトウェア技術の開発および実地検証 井上亨、垣岡淳（日本総合研究所）

B2B サービスプロキシーによる OPENEDI システム開発 徐利、滝川好夫（ウッドランド）杉本浩（スキル・インフォメーションズ）吉川洋（翼システム）
EJB のための拡張再利用の技術開発 樽井俊行（経調）大島一（情報技術開発）
JAVA(EJB)を活用した SFA パッケージの開発 琴井啓文（サン・ジャパン）坂下一幸（サイバービーンズ）田村俊明（住商エレクトロニクス）飯泉浩明（新日鉄情報通信システム）
SLCP を基準にしたプロジェクト管理支援システム 土屋雅紀（日本電子計算）掛本喜嗣（日本総合研究所）
Wordl Wide Web を利用した部品・コンポーネント流通システム 古屋義人（富士ソフト ABC）
オープンソースをベースにしたデジタルコンテンツ流通システム 伊東直幸（キーウェアソリューションズ）西中芳幸（SRA）
「オブジェクト指向設計およびプロトタイピング統合開発環境」の開発経緯と成果 佐野元之、岩田成康、古木良子、鈴木重徳（オープンテクノロジーズ）山田正樹（メタボリックス）小野哲男（小野商店）佐藤賢司（マルチパラダイムシステムズ）高木明（エイビスシステムソリューション）永松貴久代（メランジ）鈴木信寛（インフォ・ウエイブ）加藤俊嗣（アステック・イーコマース）具志堅隆児、垣花一成、岸本克巳、新垣将史（琉球大）
ソフトベンダ連携のコンポーネント活用型アプリケーション開発 岡本泰次、太田昭一（富士通）
ビジネスオブジェクト仮想統合プラットフォームの開発 沓沢直樹、岡村隆雄（インテリジェントウェイブ）
ビジネスオブジェクト標準化によるソフトベンダ間協調基盤の構築 奥悦史、橘昌宏、湯浦克彦、石田厚子（日立製作所）
ビジネス機能オブジェクトパターン標準化によるコンポーネントベンダ・ビジネス基盤の確立 内田弘之（マトリックス・システムズ）江畑英郷（シービーシー）
リアルタイム・マーケティングプラットフォーム開発事業 加藤修一（日本電算機）
分散高速情報ストレッジ実装リアルタイムプッシュ型システムの開発 小川浩二、衛藤敏夫、中村洋一（数理技研）花崎雄二（アルビス）

また、2002 年度の成果としては、次のプロジェクトが成果を報告している。

既存システム再構築のための BFOP 支援ツールの開発と検証 堀内一(ビジネスオブジェクト推進協議会)、藤野博之(NEC ネクサソリューションズ)、大林正晴(管理工学研究所)、長瀬嘉秀(テクノロジックアート)、 高橋まゆみ(日立システムアンドサービス)、辻徹(日本コンピューター・システム)、白谷勇人(コンポーネント・ベース・ソリューション)
ビジネスアプリケーション開発用共通フレームワークの整備 清水俊一郎(三井情報開発)
EC ビジネスシステムの生成ソフトウェアの開発と実地検証 安西正浩(エスシーシー)

今のところ、IPA におけるビジネスオブジェクトに関する取り組みは、この 2 件だけとなっている。しかし、日本において、ビジネスオブジェクトに関する公募に民間企業なり、研究期間が積極的に参画し、プロジェクトを実施している点は、ビジネスオブジェクトに対する国内の関心の高さを示していると言える。

3.2.5.2 CBOP の活動について

CBOP(Consortium for Business Object Promortion)とは、ビジネスオブジェクト推進協議会の略称で、設立趣旨は、「企業間あるいは異なるコンピュータネットワークの上で、オブジェクト指向技術を使った業務レベルの再利用可能なソフトウェア部品を、複数企業間で流通させるための基盤の構築に関する研究、実験、開発、普及、維持を産学協同で行う事を目的として設立された」としている。

CBOP の活動に関する概要は下図の通り。

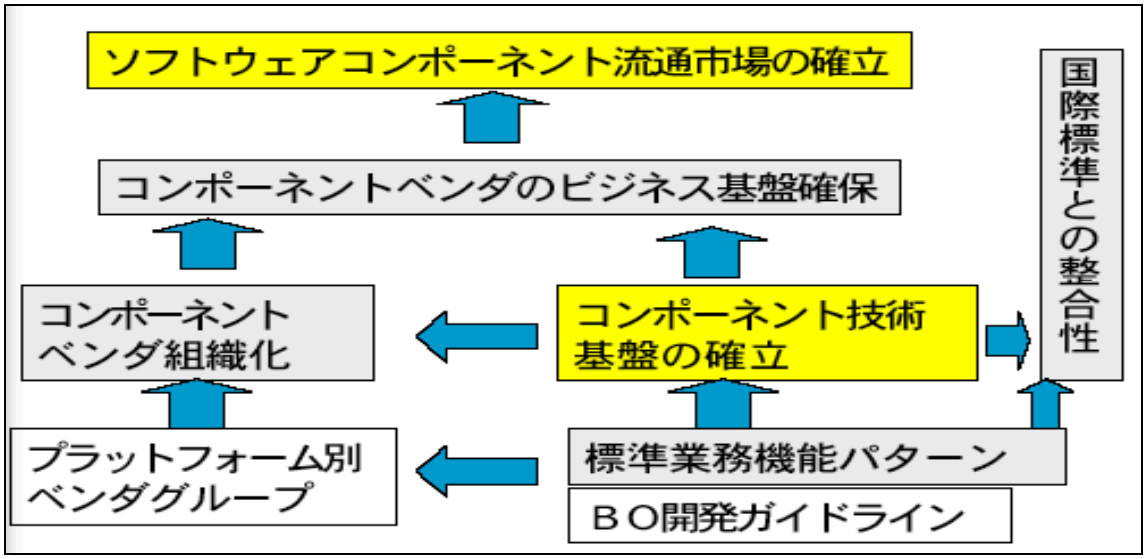


図 3-10 CBOP の活動概要

CBOP のビジネスオブジェクトに対する取り組みは、もともとビジネスオブジェクトの流通の実現を前提としているため、他の派生的にビジネスオブジェクトを取り扱う団体とは活動を異にしている。

1997 年に設立され、現在の会員数は 100 社程度となっている。CBOP では、設立の 99 年から始まって、その活動を大きくフェーズ分けしている。

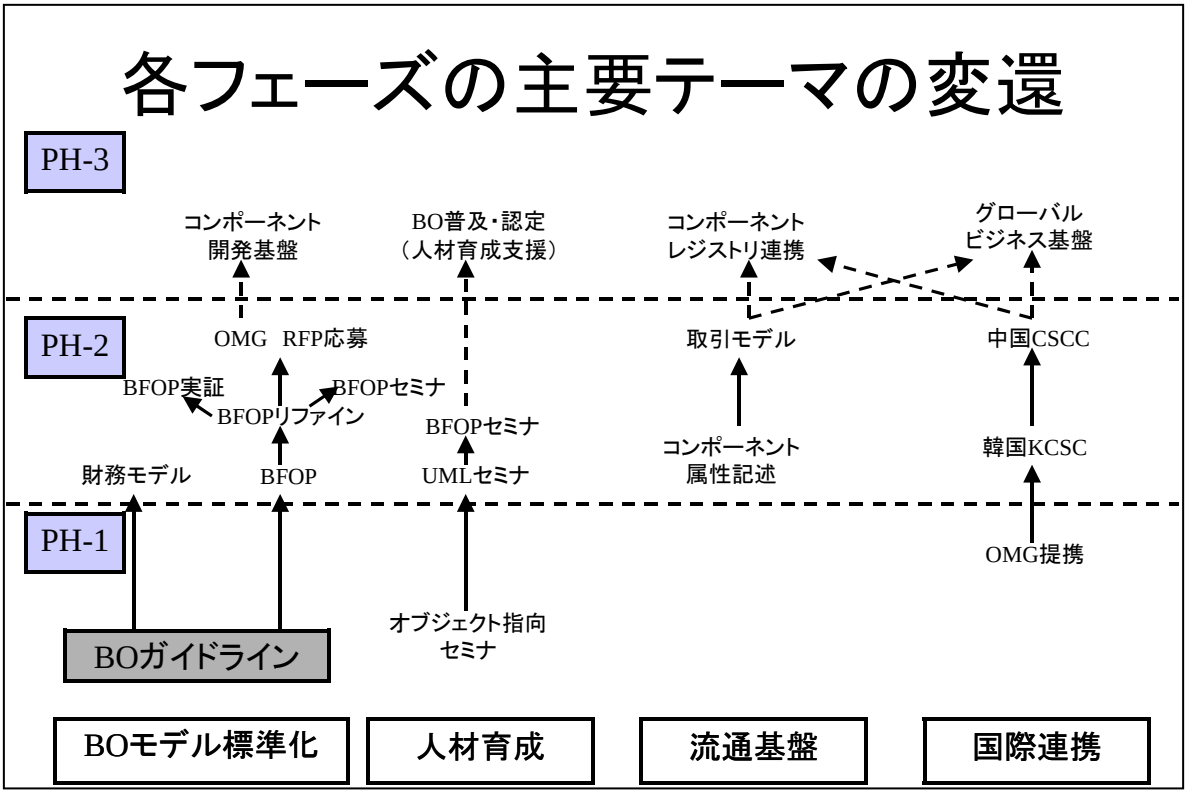


図 3-11　CBOP が考える各フェーズの主要テーマ変遷

特に、OMG との連携はもとより、アジアにおいては韓国、中国との連携を積極的に行い、グローバルなコンポーネントの流通を目指している点が新しい。

3.2.5.3　団体活動についての調査

ビジネスオブジェクトに関する団体活動としては、OMG と CBOP が双璧だが、ビジネスオブジェクトをコンポーネントという概念に絞り込むと他の団体活動が幾つか散見できる。

まず、国外におけるコンポーネント技術の普及促進団体として、米国での活動を中心とした、Reusable Asset Specification Consortium と、ヨーロッパの企業・大学を中心とする CBSEnet がある。

Reusable Asset Specification Consortium は、ソフトウェアを再利用可能資産として流通させる枠組みを確立するための業界団体であり、米 Rational Software 社・IBM 社・Microsoft 社によって運営されている。

ソフトウェア開発資産としての、コンポーネント、パターン、フレームワーク、及び、文書などを一定の形式でまとめあげて、組織間の流通促進をねらっている。このための標準仕様として Reusable Asset Specification（RAS）を策定・提案している。コンポーネント、パターン、フレームワーク、及び、文書などのソフトウェア開発資産を同コンソーシアムではアセット（ASSET）と呼と呼んでいる。アセットは、概要・利用方法・解決・分類から構成される。概要とは、アセットが提供する実装が解決する元の問題や、動機に関する人間が読むことのできる文書を示している。

分類とは、分類と検索に用いるためのメタタグ（記述子）、および、文脈（実装を用いることのできる問題領域など）に関する記述を示す。解決は、プログラムコードや設計書といった実装そのものを指している。さらに、利用方法には、実装の利用方法と、変更可能な箇所に関する記述がある。

また、RAS は、様々なアセットの仕様を記述するためのモデルを規定すると同時に、具体的な対象物に関する記述形式（プロファイル）を規定しており、今後デザインパターンなどの様々な対象物に特化したプロファイルの提案が期待される。

同コンソーシアムは、2003 年中に OMG へ RAS を標準提案することを予定している。

CBSEnet（Component Based Software Engineering Network）は、主にヨーロッパにおけるコンポーネント指向ソフトウェア開発の普及と発展を目的として、ヨーロッパの複数の企業と大学が参画するプロジェクトとなっている。 コンポーネント技術に関する調査資料の作成や、議論する場として会議を開催するなどの活動を行っている。将来的には、コンポーネント指向開発のモデルの提案を目指している。

一方、日本国内のコンポーネント関連組織としては、EJB コンポーネントに関するコンソーシアム、ビジネスオブジェクト推進協議会、分散オブジェクト推進協議会が挙げられる。

EJB コンポーネントに関するコンソーシアムは、EJB の普及と流通の促進を目指す非営利な業界団体で、2000 年に発足して以来、エンタープライズ Bean の普及を実現するために幾つかの規約を独自に提案し、また、主要な複数の EJB コンテナ製品におけるエンタープライズ Bean の可搬性の検証実験などを行っている。

ビジネスオブジェクト推進協議会については、先に紹介しているので、ここでは省略する。

分散オブジェクト推進協議会は、CORBA に代表される分散オブジェクト技術を国内において普及させることを目的とした非営利な業界団体であり、異なる分散オブジェ

クトブローカ（ORB）製品間の相互接続実験や、 SOAP 製品間の相互接続実験（Webサービスの相互運用性確認実験）、および、 J2EEサーバ製品間のエンタープライズ Bean の相互接続実験などを行い、分散オブジェクトに関連する製品間の相互運用性の検証と検証結果の公開を行っている。

情報技術コンソーシアムは、ソフトウェア産業の基盤の強化、技術水準の向上を図るため、ソフトウェア業界の共同出資によって設立された会社であり、コンポーネントウェア技術や、ソフトウェア部品流通などに関する様々な調査研究を外部と連携する形で行い、調査結果を広く一般に公開している。特に、 1998 年から 1999 年の間に実施された次世代コンポーネントウェア技術に関する研究開発の報告書は、 2000 年以前の国内外におけるコンポーネント技術に関する調査資料として高い評価がある。

情報サービス産業協会は、経済産業省の認可のもとで、国内の主要な情報サービス企業で構成される社団法人である。同協会は、 1996 年から 2000 年の間に、ネットワーク時代のソフトビジネスに関する調査研究や、ソフトウェア部品流通に関する調査研究などを行っており、調査結果を広く一般に公開している。

パターン関連組織は、J2EE 開発を主としたコンポーネント指向開発に関するパターンの提案と利用を中心とした活動をおこなう。パターンに関する国内外の団体として、以下に Hillside Group と JapanPLoP を取り上げる。

The Hillside Group スは、ソフトウェア開発に纏わる様々な知見をソフトウェアパターンとして共有・発展させることで、ソフトウェア開発をより良いものとすることを目的とする。定期的なパターンに関する国際会議（Pattern Languages of Programs:PLoP）の開催やパターンに関する書籍の発行を通じて、パターンに関わる人々（パターンコミュニティ）の支援を行っている。The Hillside Group の幾らかの協力を受ける形で、日本国内で初めての PLoP（MensorePLoP2001）が 2001 年に開催された。

JapanPLoP は国内における代表的なパターンコミュニティであるが、 2002 年 12 月現在、事実上活動を停止している。

さらに国内においては、情報処理学会のソフトウェア工学研究会（IPSJ/SIGSE）のもとでパターンに関するワーキンググループが 2003 年 4 月に発足する予定となっている。

4. オブジェクト指向による実験

4.1 モデルによる設計とは

4.1.1 プロセスとは

従来の工業化社会では、大量生産・規格品生産の為に組織は、機能毎にくくられ、それをコントロールするために上位に組織を積み上げる階層型の組織であった。

すなわち、工業化時代の階層的組織とは、

- 機能に合わせて業務を組織化し、
- 教育を受けていない労働力で
- 従業員の監督と研修を単純化し、
- 管理の統制の範囲を最大化し、
- 情報の自由な流れにほとんど依存しない形態である。

当然、この時代のシステム設計というのは、上記の組織の省力化ということで、その組織に合わせてシステムを作ると言うことになる。システムを機能単位に分割して、それの連動で全体が構成されるという機能分割による設計となる。

これが情報化社会となると、利益の源泉が、

- 製品中心のコストダウンからプロセスコストダウンの要請の高まり
- プロセス自体が競争力の源泉という認識の高まり
- 事業価値の変化(製品とサービスの総合的供給など)
- 商品の変化(製品のコモディティ→プロセスのスリム化)

等々の要求が高まり、業務は、非連結の機能の集合というよりも、外部ユーザにサービスを提供するプロセス(利用可能な製品サービスの生産に必要な活動の集合であったり、顧客を持ち、従来の組織とは独立して組織をまたがって存在する機能であったりする)で組織化されマネジされなければならないという、プロセス革新の時代に入ってきた。

今後、この一気通貫のプロセス全体をコントロールすることが求められる。また、このプロセスを再構築する BPR などが求められてきた。

この様な時代には、上記の機能別に分割してソフトを組み立てていくという考え方でなく、プロセスという物を 1 つのシステム化の対象物として取り扱っていく必要がある。

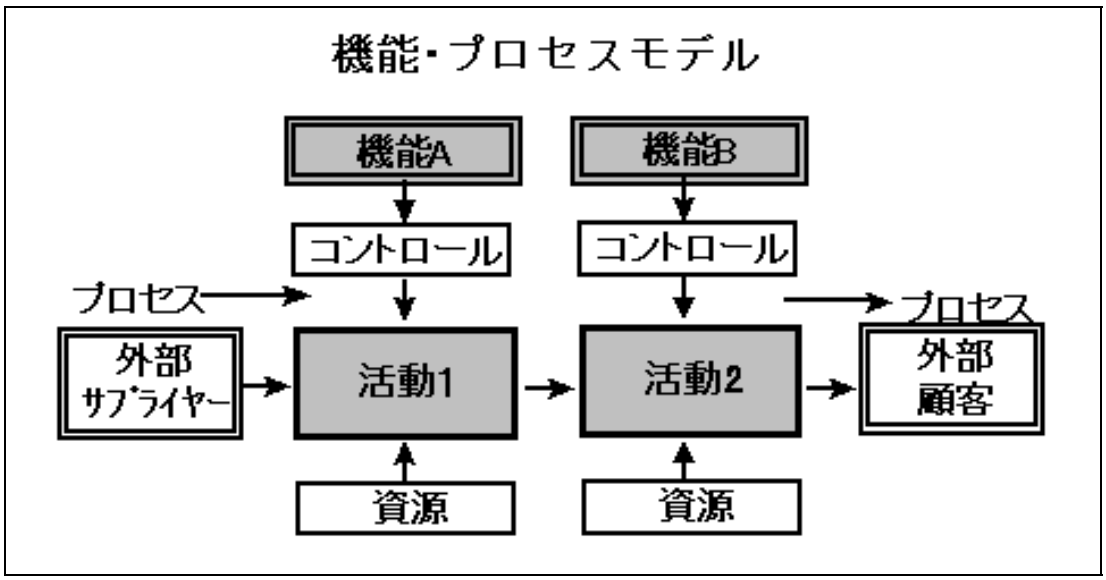


図 4-1　機能・プロセスモデル

4.1.2　ビジネスモデルとビジネス・プロセスモデル

ビジネスモデルとは「価値を創造するビジネスの仕組み」であり、それを実現するために仕事のやり方を定義したものがビジネス・プロセスである。
先程の、ダベンポートのプロセスの定義は、簡単に言えば仕事のやり方を定義したものである。ただしその定義は抽象的であり、実際の仕事進めるためには「プラクティス」が決められる。

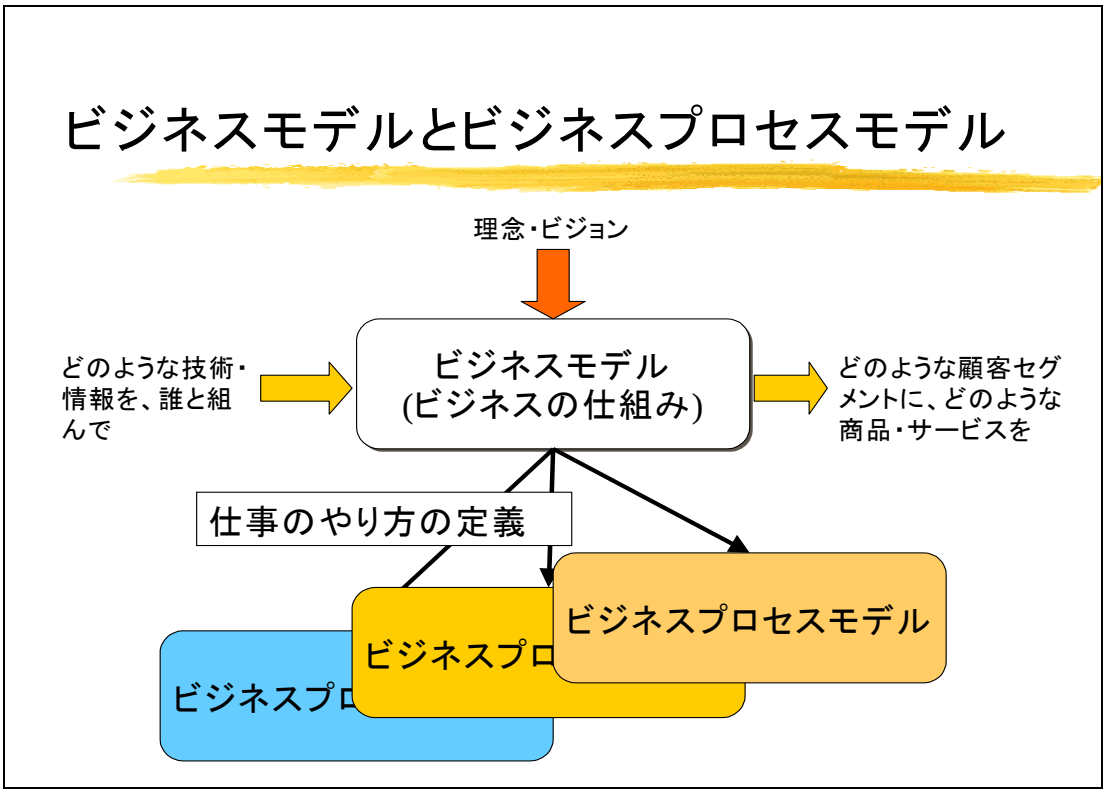


図 4-2　ビジネスモデルとプロセスモデル

ビジネス・プロセスという粒度の抽象度で、ビジネス・プロセス自体も類型化する事が出来る。その中で、良いプロセスとは、「プロセス自体の処理が速いこと、コンパクトなこと」であり、それらの良いプロセスを集大成したベストプラクティスが作られることになる。そのベストプラクティスが 1 つの枠組み（フレームワーク）として提供されることになる。

4.1.3　ビジネス・プロセスモデルを使用することの利点

この様に、ビジネス・プロセスモデルを抽象的な形で定義し、そのフレームワークを用いて開発を実施することで、プロセス指向に基づいた一連の業務を提供し、ワークフローや情報フローを企業横断で整理・統合することができる。

また、既に作られたプロセスを参照することは、システム構築では、次のような利点を持っている。

- 業務システム段階の業務分析コストの削減
- 業務機能と業務ルールの明確化

- 業務モデル部品のパターン化によるシステム開発工数の削減
- 情報技術を位置付ける型紙
- 国際競争力を維持

また、企業間では、

- 企業間での共通言語
- 企業間での共通データ
- システムの統合を可能とする。
- Ｅコマースへの展望

等の利点をもたらす。

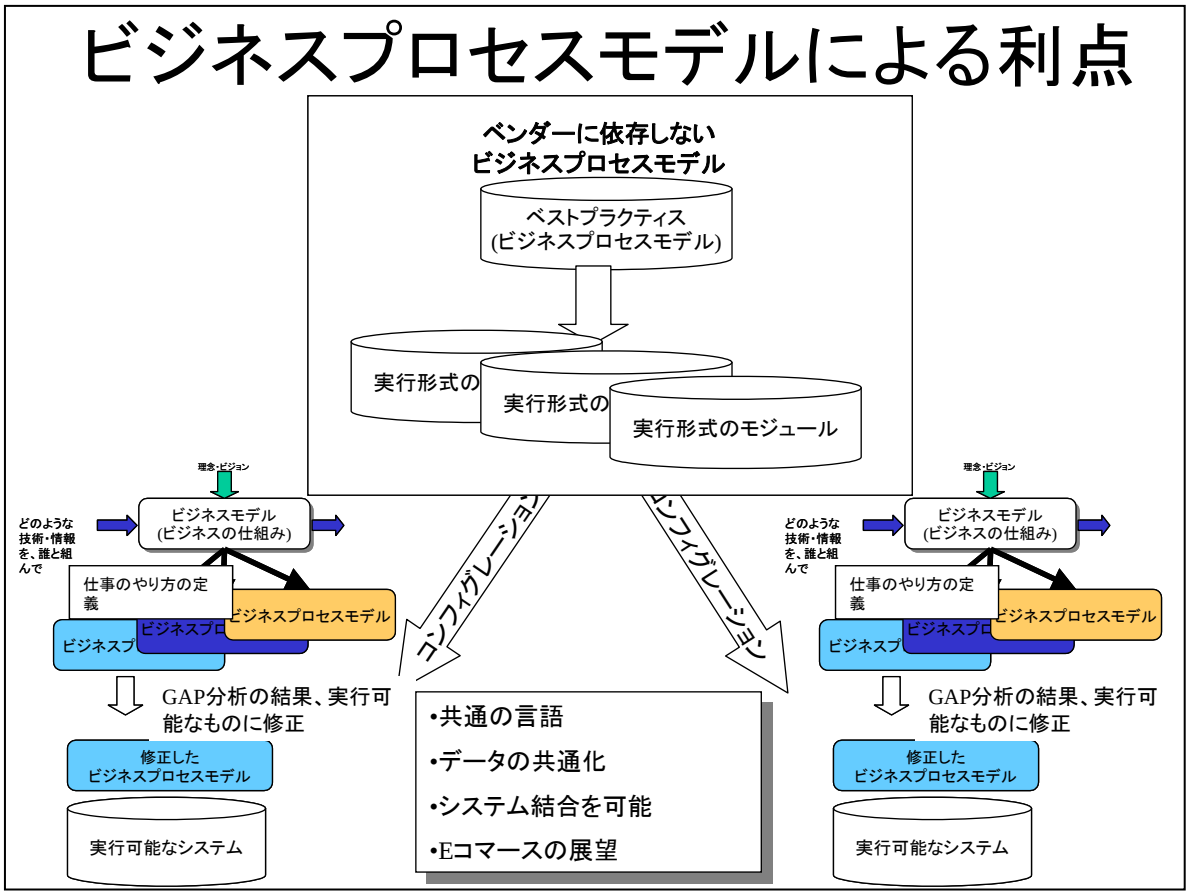


図 4-3　ビジネス・プロセスモデルによる利点

4.1.4　モデルから一気に実行可能なソフトの世界へ

（モデルベースの設計）

この様なビジネスモデルを実現するためのビジネス・プロセスの集合から、一番良いもの（ベストプラクティス）を型紙として、派生させたカスタマイズされたビジネス・プロセスが出来上がる。もしも、このベストプラクティスで記述されたビジネス・プロセスモデル

が、実際に稼働可能なソフトと対応していれば、ソフトの世界は、モデルから一気に実行可能なシステムへ移行できることになる。ERP(エンタープライズ・リソース・プランニング)では、この型紙を極力変更せずに適用するように、ビジネス・プロセス自体の変革を求めるものである。一方、モデルから実行可能なソフトまでが、オブジェクト指向技術で、フレームワーク化とコンポーネント技術などでビジネス・プロセス自体がビジネス・プロセスオブジェクト整備されておれば、それを使ってのカスタマイズされた自社向けのシステムを構築することが出来る。

要は、ビジネスモデル→ビジネス・プロセスモデル→オブジェクト指向技術での実行可能なソフトへのブリッジが架けられれば、ビジネスの変更が即座にシステムに反映することが出来る様になる。

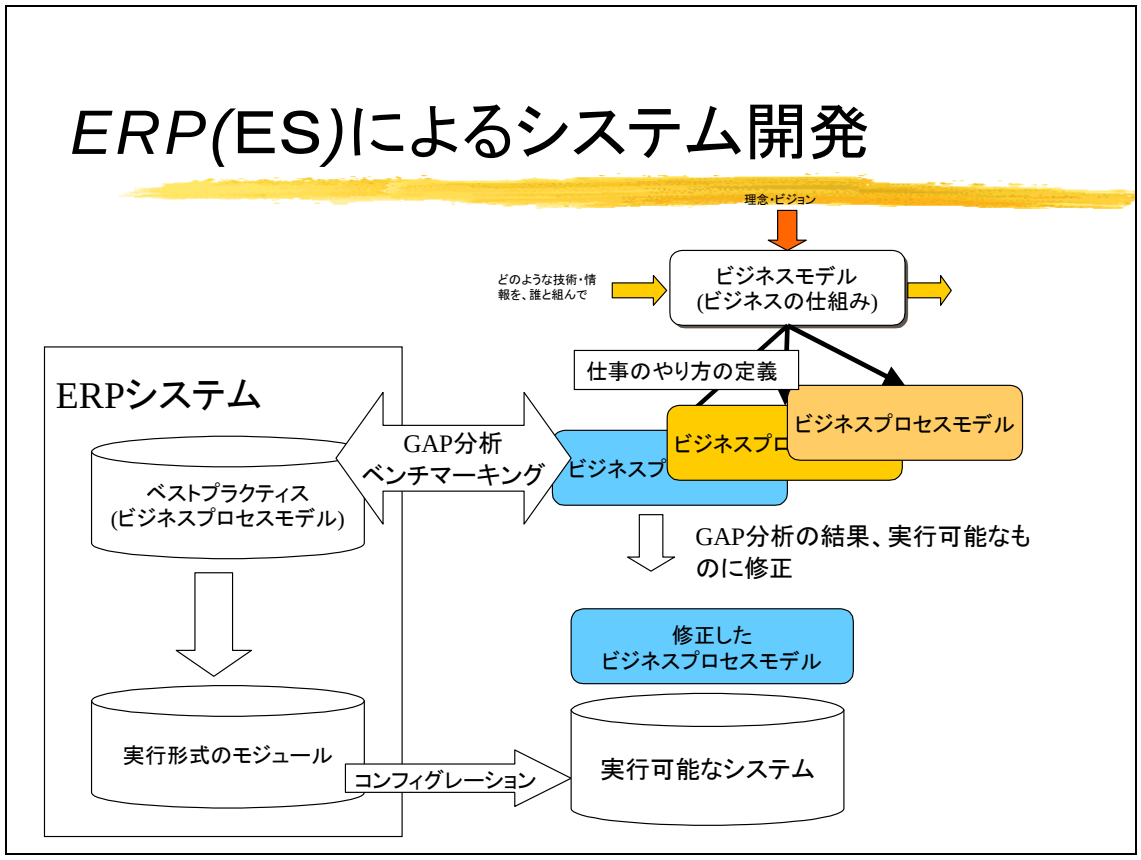


図 4-4　ERP(ES)によるシステム開発

4.1.5　経営工学から情報工学・ソフトウェアエンジニアリングへ

ビジネスモデルからビジネス・プロセスモデルを経由して、既にあるビジネス・プロセスとそれを実行可能としているモジュールへマッピングすることは、従来のシステム開発での大きな課題であった、経営課題と IT そのものを直接に

結びつけることになり、それのパフォーマンスを測定可能とすることが出来る。
(経営に役に立つシステム、また、それの測定を可能とする。)
これは、経営自体が工学化されて、それを情報工学、ソフトウェアエンジニアリングへとつなぐものである。

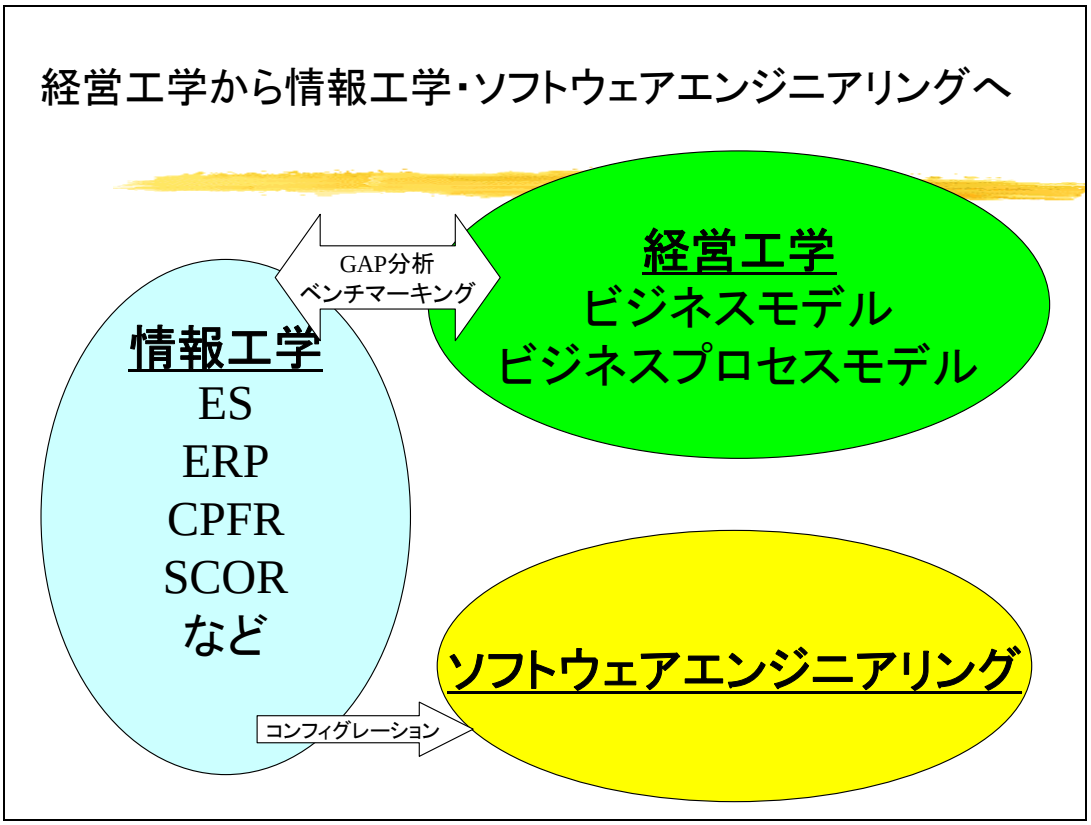


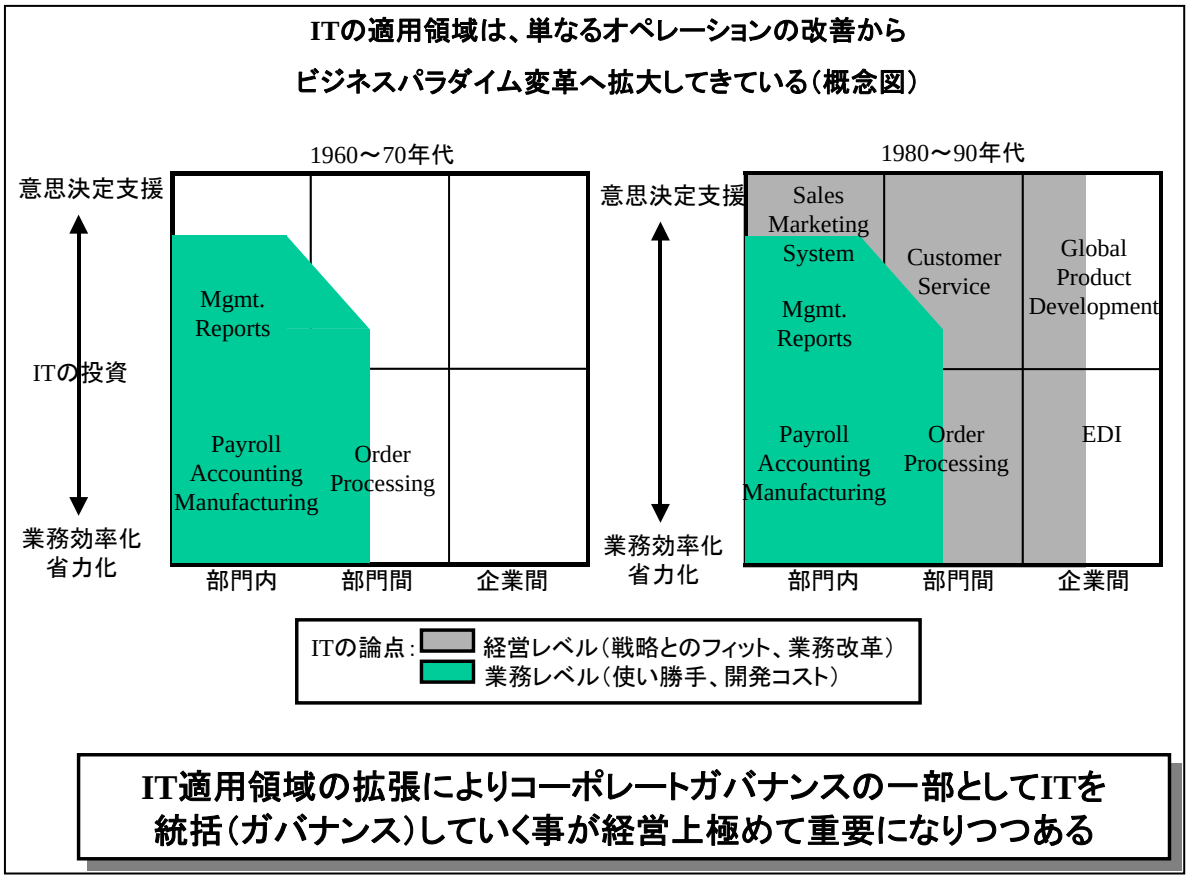
図 4-5　経営工学から情報工学・ソフトウェアエンジニアリングへ

4.2 オブジェクトによる設計

～　モデルからシステムへの架け橋として　～

4.2.1　情報システムの変遷

従来のシステム設計は、前章で述べたように、企業の一機能の省力化から始まり、現在では、単にオペレーションの概念から、IT を如何に使いこなすということが企業利益の源泉となりつつある。



（参考）「企業の IT ガバナンス向上に向けて」～情報化レベル自己診断スコアカードの活用～

平成 11 年 3 月　通商産業省　より

図 4-6「企業の IT ガバナンス向上に向けて」

これらに対応した言葉として、IT ガバナンスということが提唱されつつある。IT ガバナンスとは、「企業が競争優位性構築を目的に、IT（情報技術）戦略の策定・実行を コントロールし、あるべき方向へ導く組織能力」ということであろう。従って、IT システムを構築すると言うことは、

- 「経営者の想い（ビジネスモデル)」から
- 何故作らなければならないかの動機を明らかにして（WHY）
- それが要求するものを明確にし（WHAT）
- 分析を通じてシステムに実装する方法を明らかにする（HOW）

ということになる。

4.2.2　従来の構造化設計

構造化設計は、システム化すべき対象を機能で分割して、それをトップダウンで、段階的詳細化していくものである。その分割は機能として多くの人が理解できるところで止めることになる。システムは、これらの機能の集まりとして実現されることになる。物理学のたとえで言えば、物質そのものでは理解が出来ないので、それを分子、原子、電子､素粒子などに単位に分割することで全体を理解しようとする考えに通じるものがある。

それに応じた開発過程として、ウォータフォールモデルが提唱された。

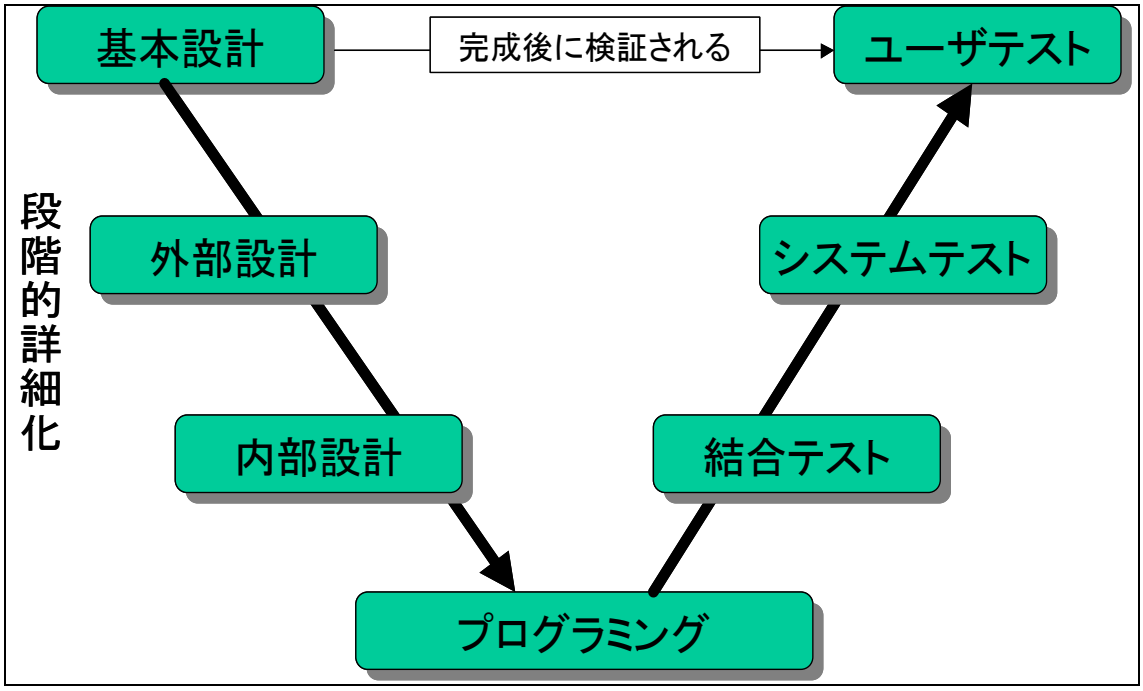


図 4-7　構造化設計

しかし、要求自体の検証は、システムがほぼ完成した時点で、実施されるこ

ととなり、後戻りが出来ないなどの欠点がある。それらをカバーするためにプロトタイプ開発、スパイラル開発などが提唱された。

しかし、基本的には分割単位としては機能を中心に分割された物である。従って、ビジネスモデルの変更に対するシステムの変更は、全体を見直すことになる。

4.2.3　データ中心設計

構造化設計が機能中心であるために、システムの強度が弱いと言うことで、ビジネスが如何に変化しようとも、その変化を受けない物として、データに着目して設計をする方式が、データ中心設計である。80 年代全般にはデータフローモデルを中心にしたシステム記述(DFD モデル)や、情報資源管理(IRM:Information Resource Management)概念の登場で、情報やデータは共有物であり企業の経営資源の一つであると言う概念が普及し始めた。先に共有物としてデータベースを決定すれば、ソフトウェア開発はデータに従属したものとなるということで、情報システム開発方法論にデータ中心アプローチ(DOA)が普及した。

データ中心設計では、まずデータ分析を行い、システム化対象業務で使われているデータ項目を意味のある単位にグループ化し、その相互関係を、データモデルを用いて視覚的に表現する。

データモデルとは、あるルールを適用することで、物事（業務内容）に関連する複雑に絡み合ったデータの関係を分かりやすく解き明かそうというもので、ER モデル（エンティティ・リレーションシップ・モデル）を利用する。

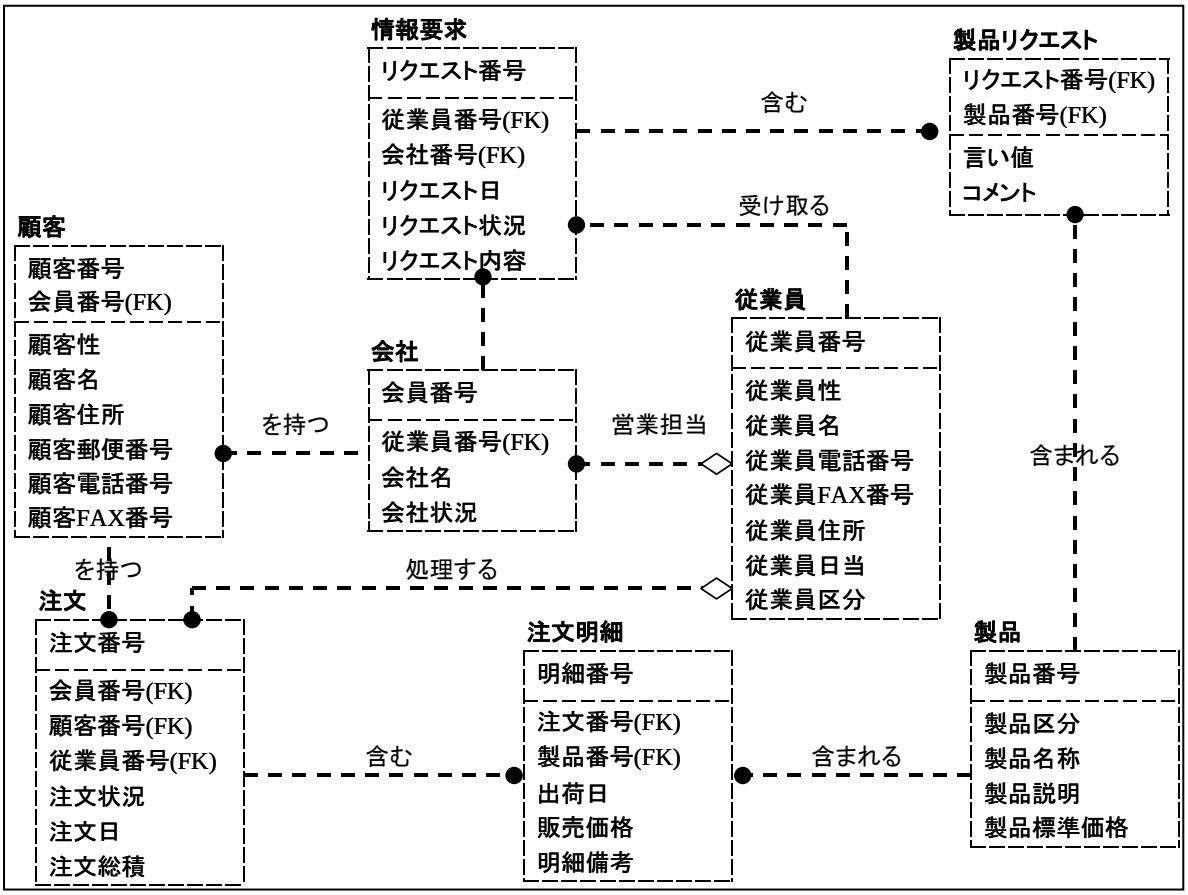


図 4-8　ER 図例

90 年代には、データ主導の発想とオブジェクト指向とは一体になり、オブジェクトモデル化手法として Runbaugh らが提案した OMT(Object Modeling Technique)として、データモデルにオブジェクト指向としてのクラス概念や継承などを追加された。その後に OMT と Booch, Jacobson らがそれぞれ唱えていた手法を統合した UML(Unified Modeling Language)などが提唱された。

4.2.4　オブジェクト設計

そもそも、ソフトウェアとは、

- 問題領域の複雑さ
- 開発プロセス管理の複雑さ
- ソフトウェアの柔軟性の確保
- 個々のシステムの、振舞いの癖の問題

という性質を持っており、その複雑さは本質的な性質である。その複雑さを取扱うためには分解(Decomposition)が必要で必要であった。**この分解の観点を、機能やデータという観点でなく、オブジェクトという単位で行うこと**で、よりよくシステム全体が理解できるようになった。

たとえば、在庫管理システムということを考えるとき、従来の手法であれば、

在庫マスタがあり、それを検索することでA商品の手持在庫を知ることが出来る。

という記述より、

ここにA商品があり、そこを見れば今の在庫量がわかる。

という記述のほうが、イメージし易いし、判り易いに決まっている。

これは、また従来手法のように、プロセスモデリングとデータモデリングを別々に設計することではなく、認識された物(オブジェクト)で行うことができることが大きな利点となっている。すわわち、認識できる実態としてオブジェクトを考えることである。

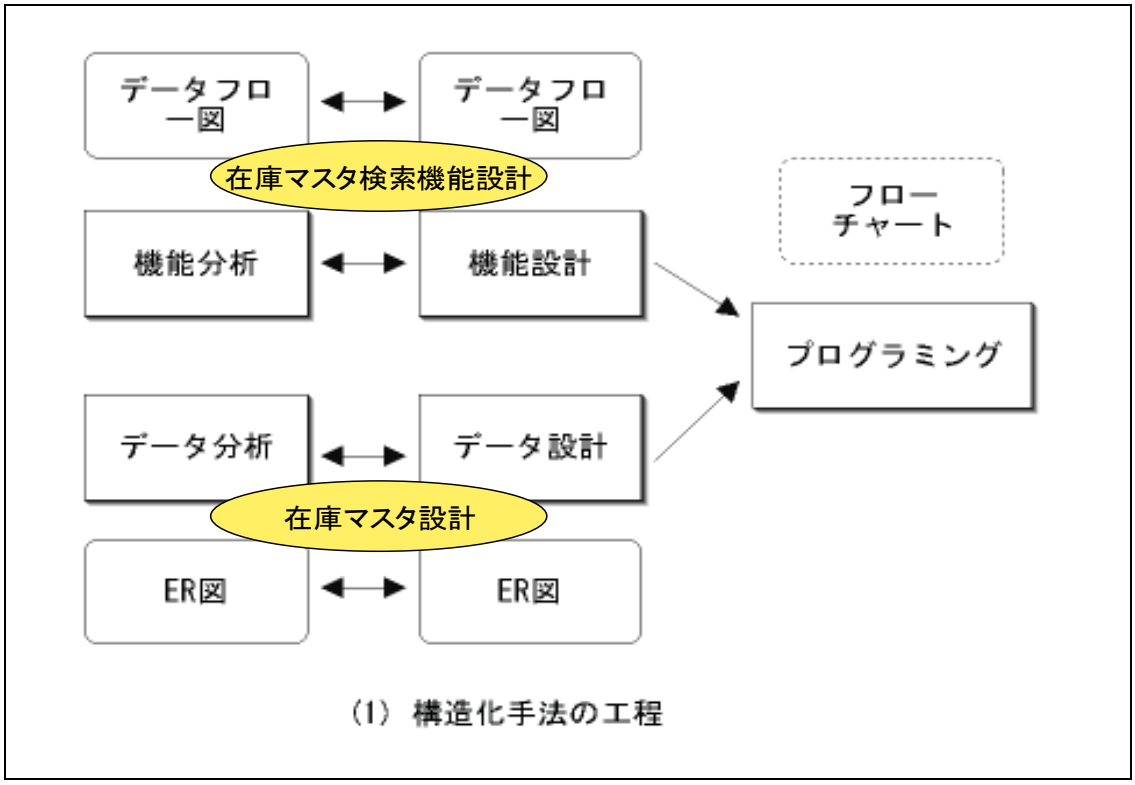


図 4-9　構造化手法の工程

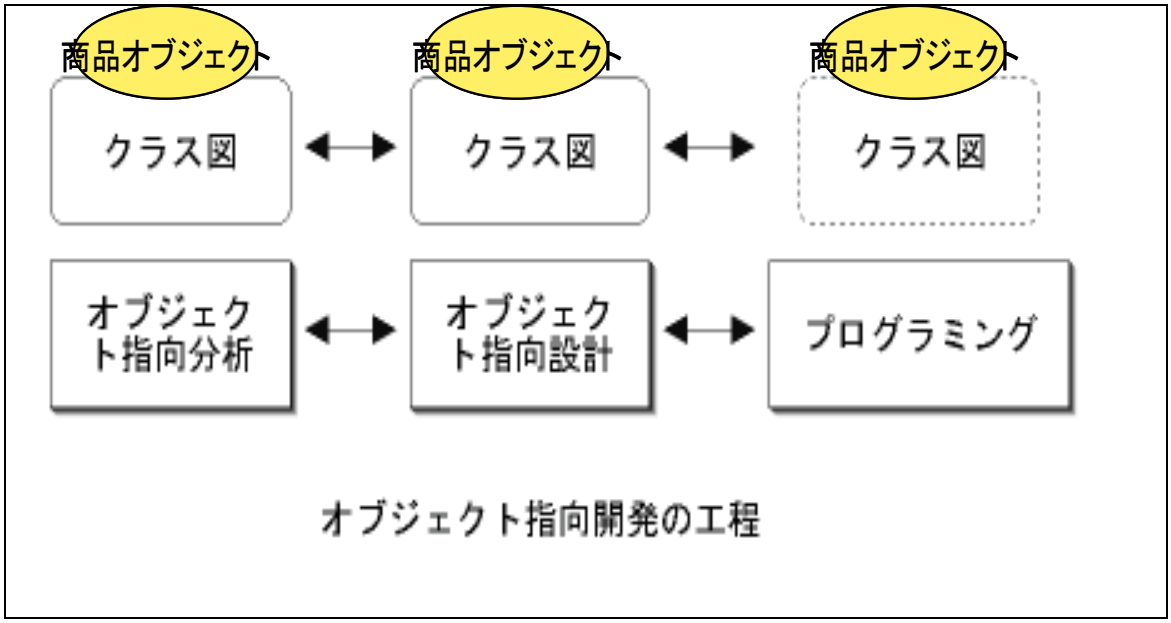


図 4-10　オブジェクト指向開発の工程

すなわち、

「我々の経験によればオブジェクト指向の観点を最初に(アルゴリズムの観点より前に)適用することが望ましい。なぜならこのアプローチはソフトウェアシステムが本来もっている複雑さを組織化することを助けるのに適しているからである。」

G. Booch の「Object-Oriented Analysis And Design with Applications (2ND)

ということになる。

オブジェクト指向の観点を導入することによって、

- 人間指向のモデリング
- シームレスな開発工程の連携
- 異分野の技術の統合
- 再利用性の向上
- 生産性の向上
- 品質の向上
- 分散技術との相性

を、可能にすることができる。

4.3 オブジェクト開発方法論

開発方法論とは、記法（Notation）と開発プロセスから構成される。

- 記法とは、オブジェクト指向分析・設計に使用するクラス図などのダイアグラムの文法を指す。
- プロセス(Process)とは、オブジェクト指向設計を行うための方法論そのものである。

構造化設計であれば、DFD とウォータフォールモデルなどの組み合わせです。オブジェクト指向開発では、記法や開発プロセス対して多くの提唱がなされた。百家争鳴の時代が続いた後に、記法についてはほぼ UML（Unified Modeling Language）として、統一されていった。

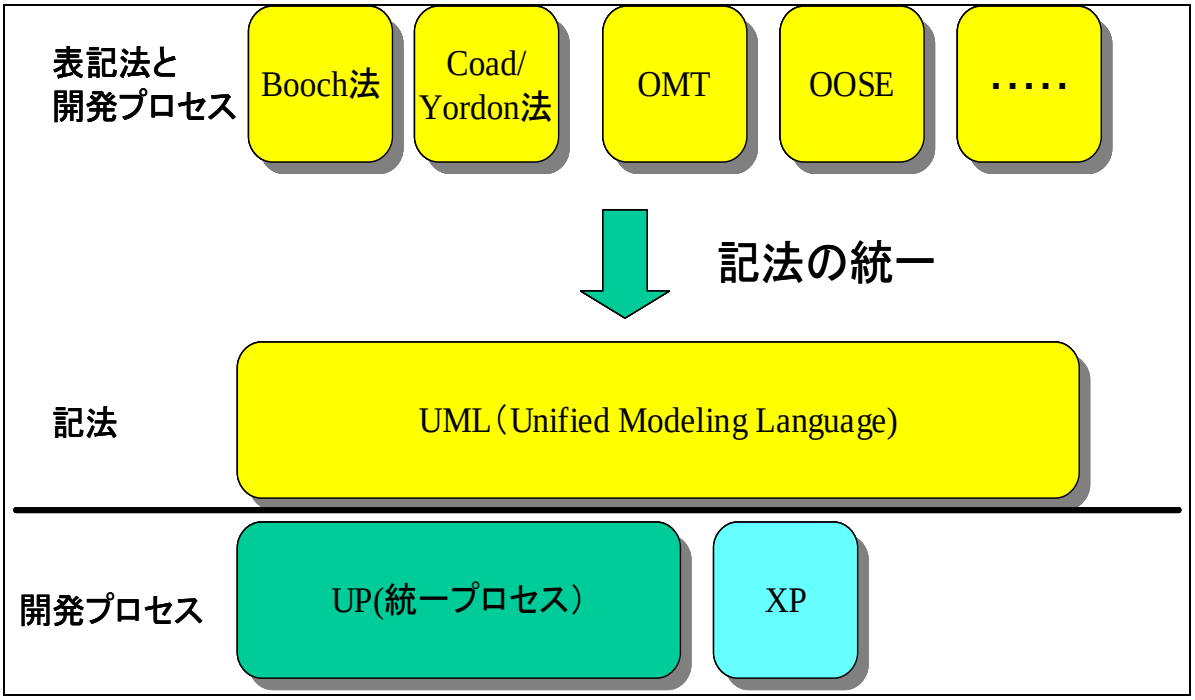


図 4-11　表記法統一の流れ

この統一の際に UML に 4 つの目標が設定された。(『UML サマリ』、日本ラショナルソフトウェア株式会社より)

- オブジェクト指向の概念を使ってシステム（ソフトウェアとは限らない）をモデル化すること
- 上流工程の開発成果物や実行可能な成果物との明示的な結び付きを確

立すること
- 複雑かつミッションクリティカルなシステムでは避けられない、システム規模の問題に対応すること
- 人間とコンピュータの両方に利用可能なモデリング言語を開発すること

4.4 UML について

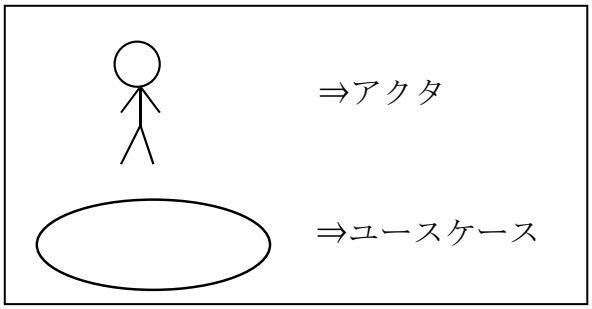
UML は下記に示す 9 種類のダイアグラムからなる。

	視点	ダイアグラム名
要求図	要求を表す	ユースケース図
構造図	静的な構造を表す	クラス図 オブジェクト図
振舞い図	動的な構造を表す	シーケンス図 コラボレーション図 アクティビティ図 ステートチャート図
実装図	物理的な実装を表す	コンポーネント図 配置図

4.4.1ユースケース図(Use Case Diagram)

システムの要求仕様、つまりシステムが実現しなければならない機能を表現するために使用される図である。システムの提供する機能を示すユースケースとシステムのユーザを示すアクタ、そしてそれらの関係から構成される。

＜判例＞



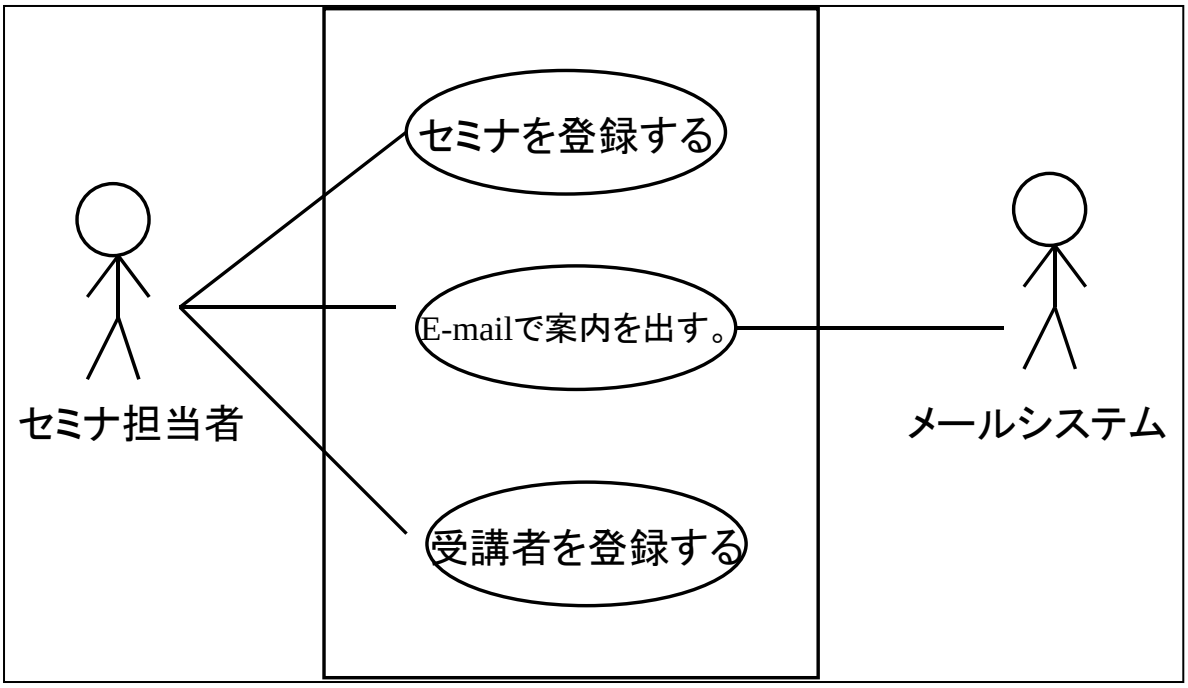


図 4-12　ユースケース図例

4.4.2クラス図(Class Diagram)

クラス間の静的な構造を表現する図で、オブジェクト指向での中心となる図です。クラス図はクラス(Class)、インタフェース(Interface)とそれらの関係(Relationship)を表す。

- 具体的には問題領域における概念間の関係を示す構造とか、システムの構造をしている。非常に難しい部分をモデリングするために使われる。
- 問題領域にどのような概念があり、それらがどう関係しているのかを表している。
- ER 図と違うのは、データ以外に操作の配置も含まれていること、全てのクラスが DB にマッピングされるわけでは無いこと

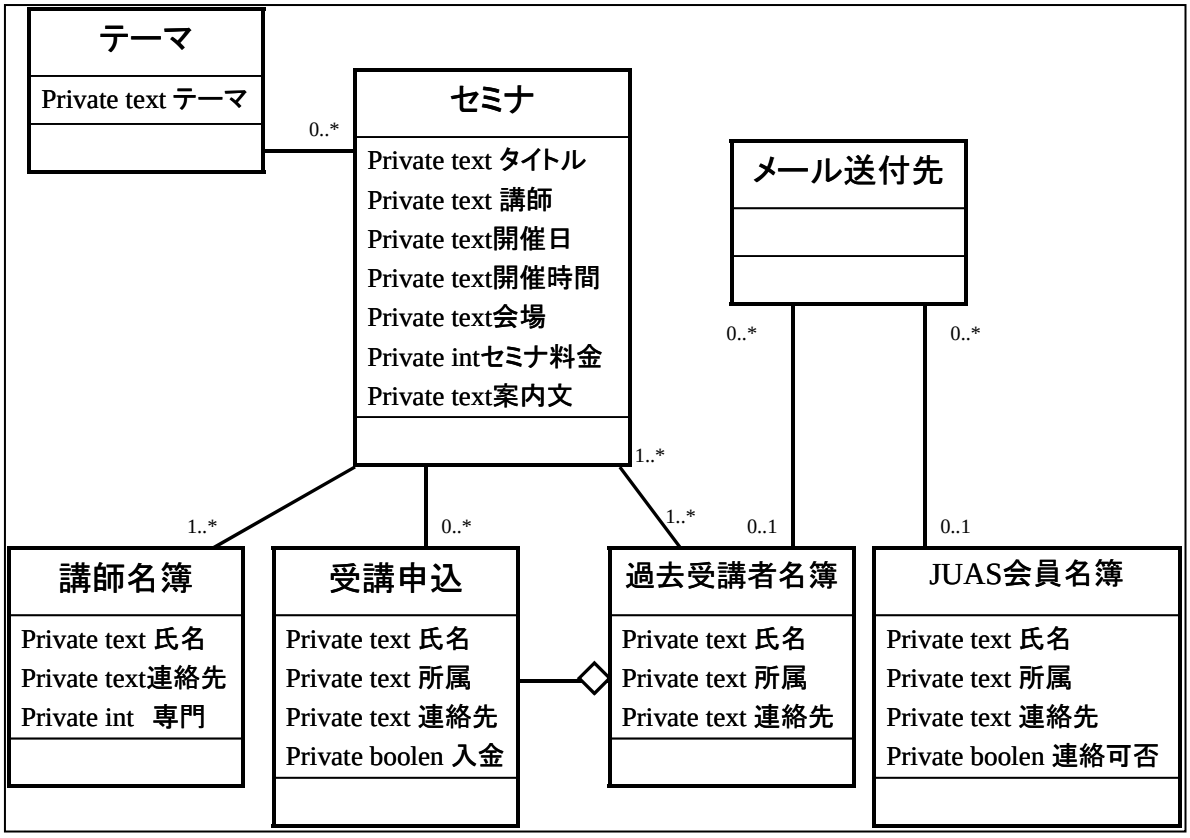
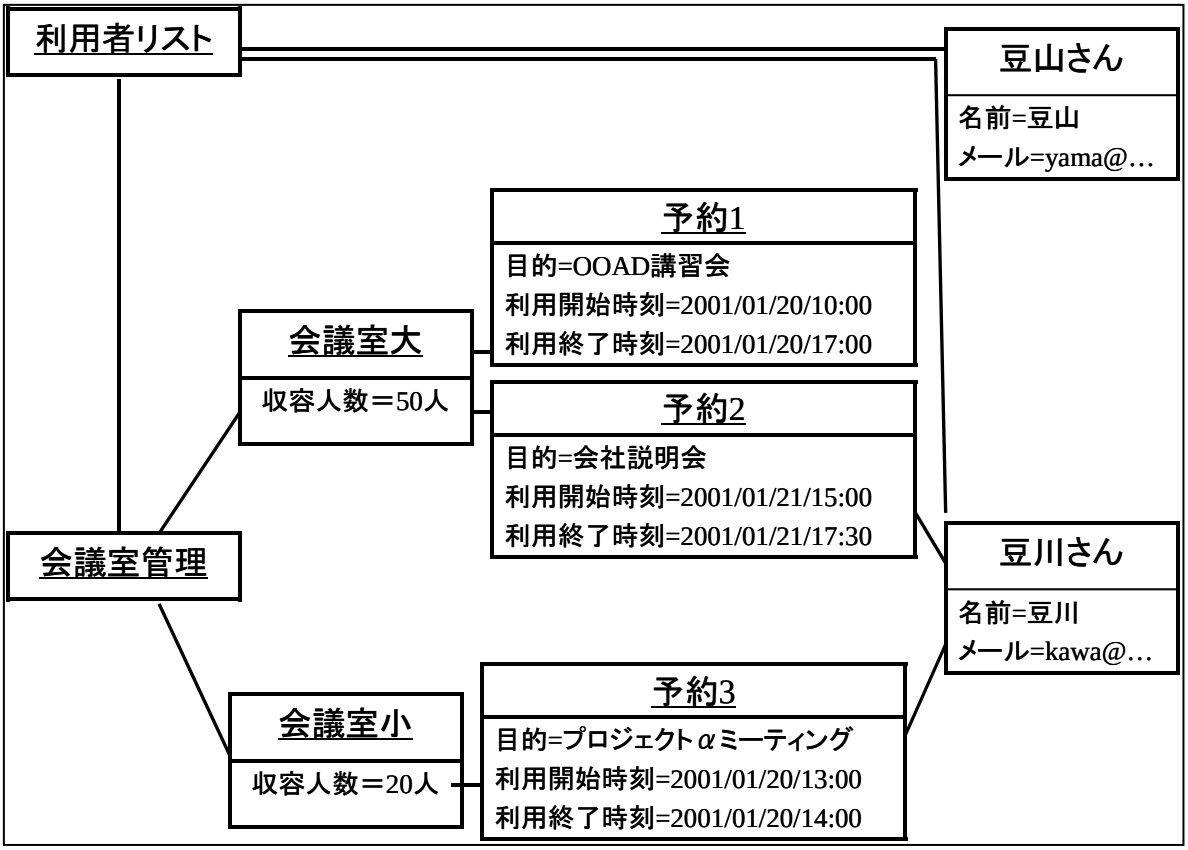


図 4-13　クラス図例

4.4.3オブジェクト図(Object Diagram)

クラス図で表されたクラスがオブジェクトとして実体化された場合の関係を表している。オブジェクト図はシステムの実行時のある瞬間のオブジェクトの実体を表したものである。



出典：株式会社　豆蔵　UML 入門<基礎編>HTTP://WWW.MAMEZOU.COM/TEC/TIPS/UMLFORBEGINNER/INDEX.HTML　より

図 4-14　オブジェクト図例

4.4.4インタラクション図

コラボレーション図とシーケンス図はまとめてインタラクション図(Interaction Diagram)と呼ばれることもある。オブジェクト間のメッセージを表している。

4.4.4.1シーケンス図(Sequence Diagram)

オブジェクト間を流れるメッセージのシーケンスを表現する図である。オブジェクト間のインタラクションを時間的な観点で表現する。

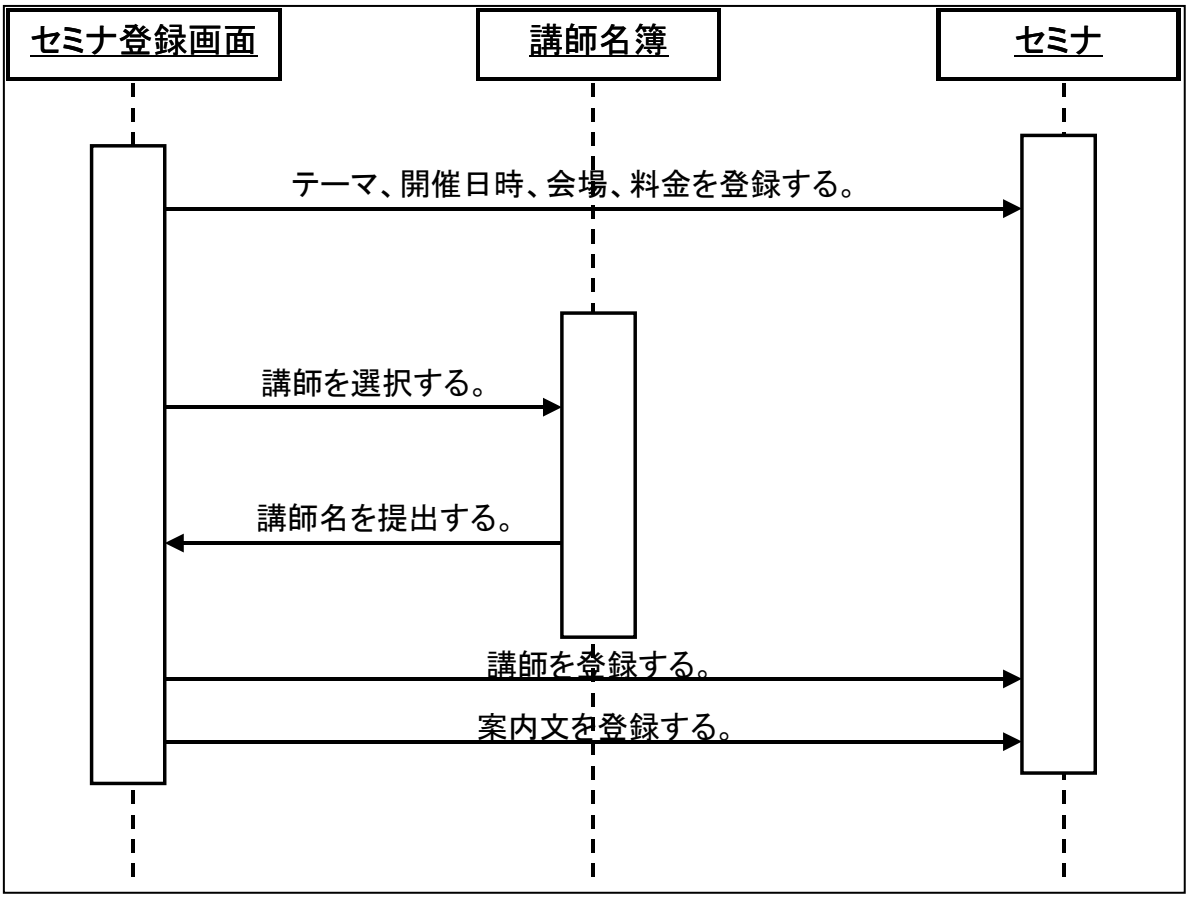
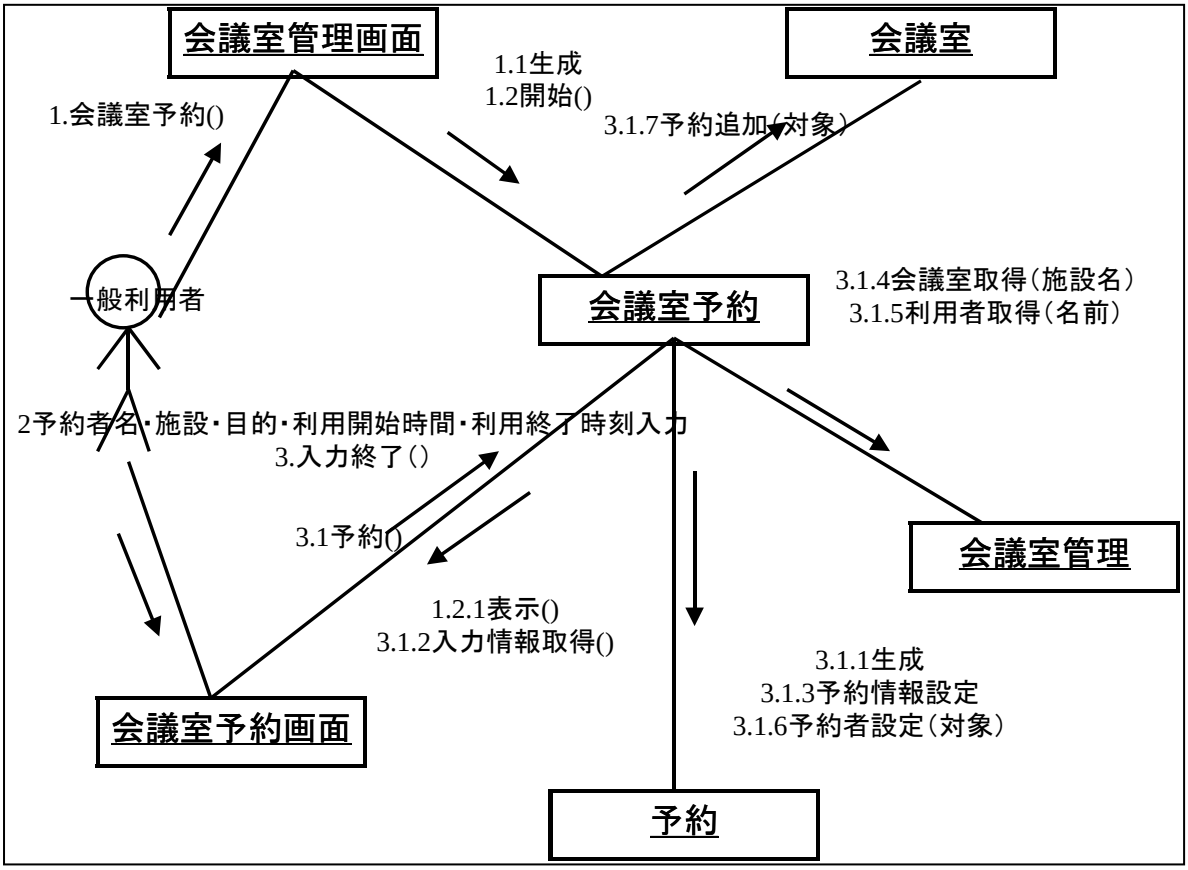


図 4-15　シーケンス図例

4.4.4.2コラボレーション図(Cooaboration Diagram)

オブジェクト間の相互作用を空間的な観点で表現するための図。

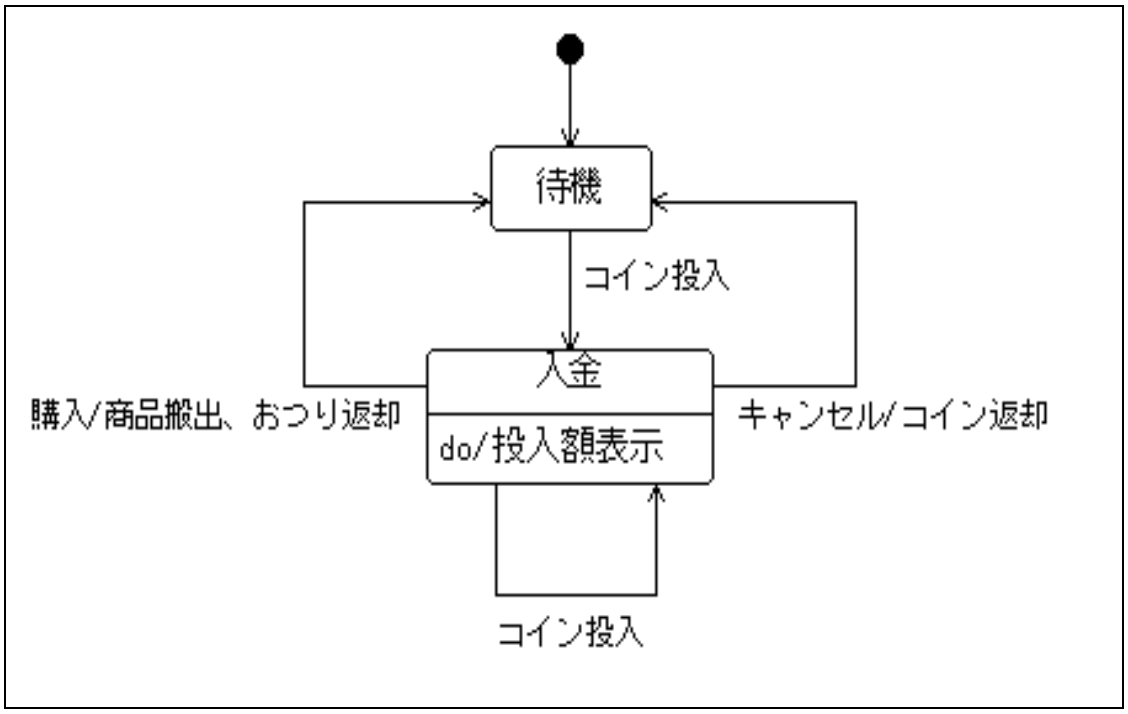


出典：株式会社　豆蔵　UML 入門<基礎編>HTTP://WWW.MAMEZOU.COM/TEC/TIPS/UMLFORBEGINNER/INDEX.HTML　より

図 4-16　コラボレーション図例

4.4.5ステートチャート図(State Chart Diagram)

状態遷移図とも呼ばれる図で、オブジェクトの状態の遷移を表現するために用いる。

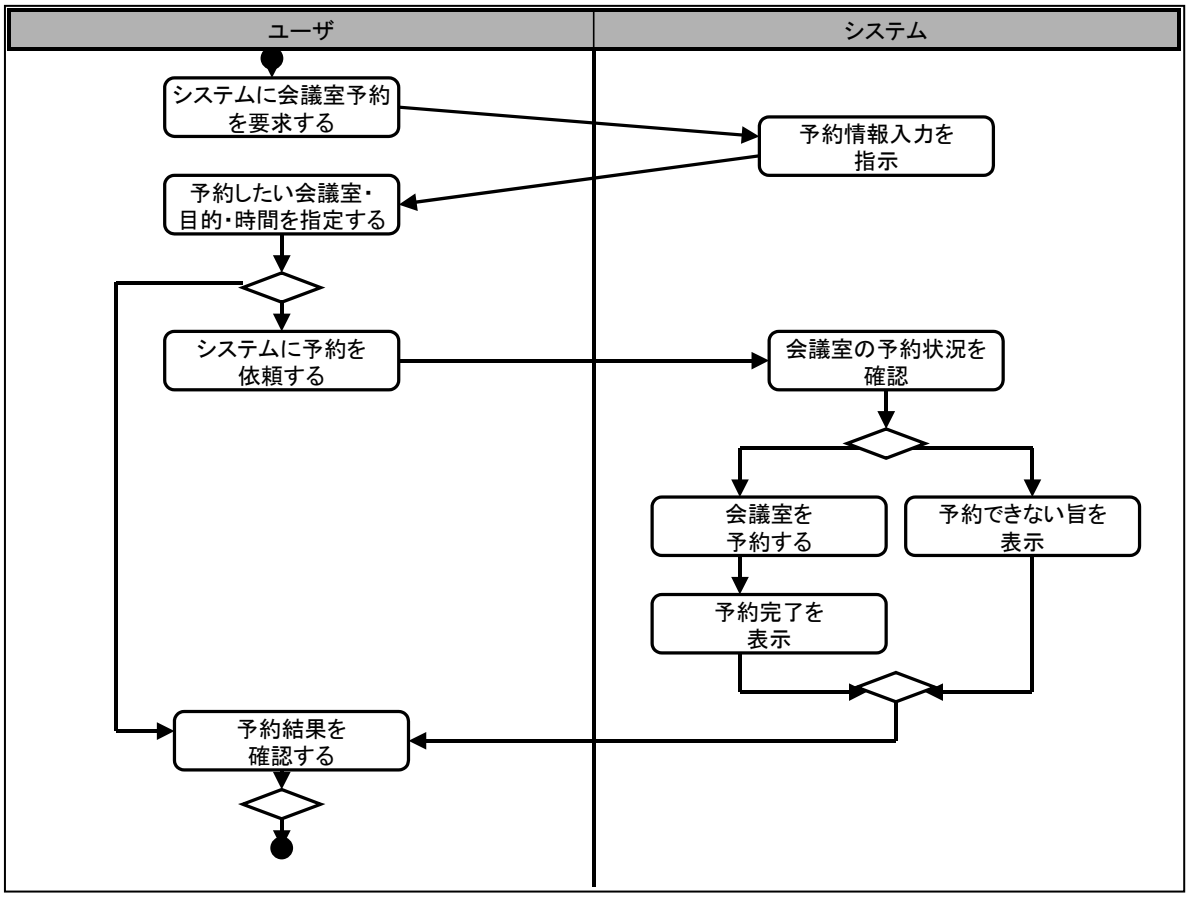


出典：じゃばじゃば　HTTP://WWW.ASAHI-NET.OR.JP/~DP8T-ASM/JAVA/HOME.HTML　より

図 4-17　ステートチャート図

4.4.6アクティビティ図(Activitiy Diagram)

アクティビティのフローを表現するための図です。システムの動的側面をより具体的に記述するときに使用する。フローチャートやデータフロー図といった機能詳細化手法で使われていた図はこのアクティビティ図で表現できる。

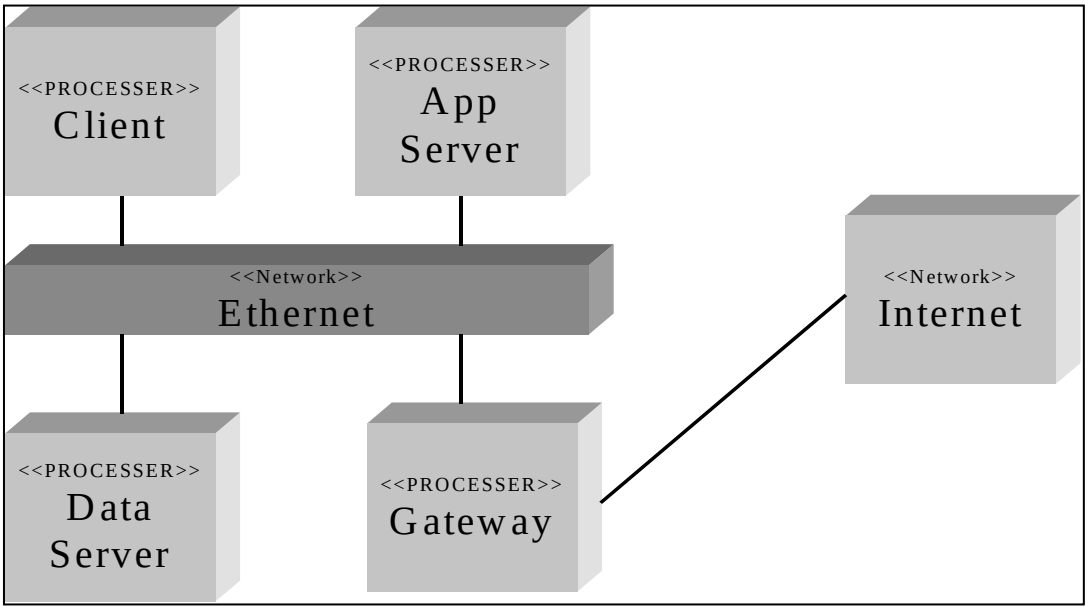


出典：株式会社　豆蔵　UML 入門<基礎編>HTTP://WWW.MAMEZOU.COM/TEC/TIPS/UMLFORBEGINNER/INDEX.HTML　より

図 4-18　アクティビティ図例

4.4.7コンポーネント図(Component Diagram)

システムの物理的なインプリメンテーションを表現するための図である。ここでいうコンポーネントはJAVABeansやActive Xなど、いわゆる再利用のためのモジュールとしてのコンポーネントも含むが、それよりも対象とする範囲が広くなっている。

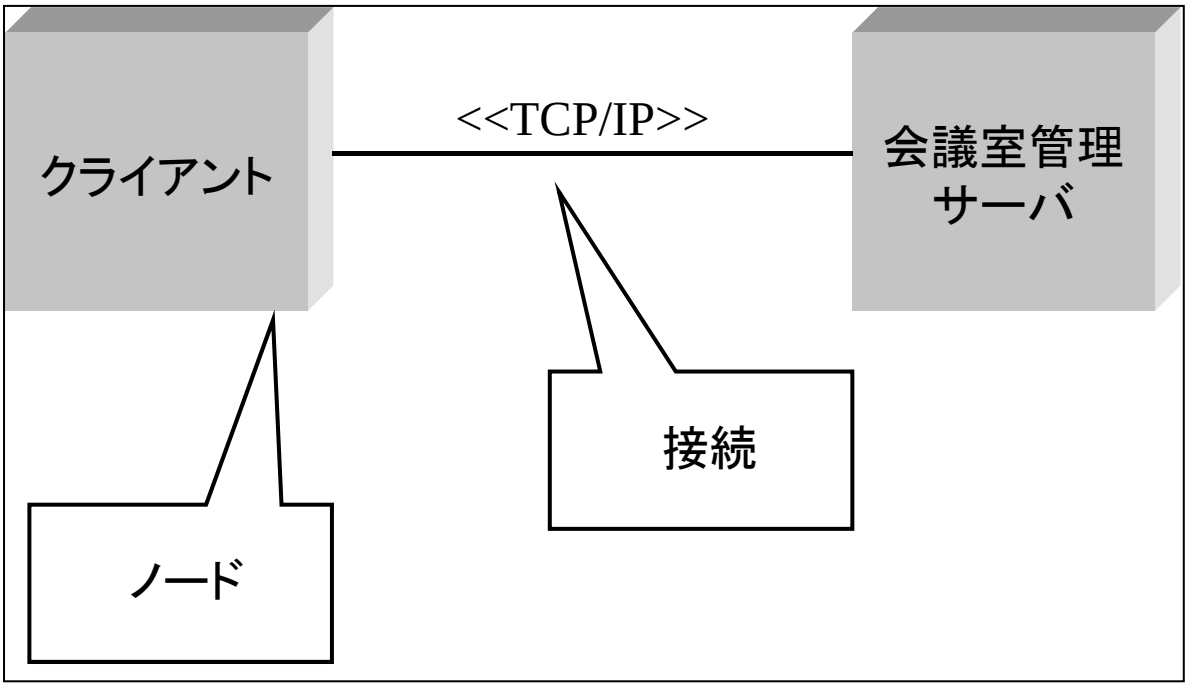


出典：JAVAプログラマのためのUML事始め　HTTP://WWW.NETPASSPORT.OR.JP/~WTASAMI/より

図 4-19　コンポーネント図例

4.4.8ディプロイメント図(Deployment Diagram)

システム動作時のコンフィグレーションを表現するための図。



出典：株式会社　豆蔵　UML 入門<基礎編>HTTP://WWW.MAMEZOU.COM/TEC/TIPS/UMLFORBEGINNER/INDEX.HTML　より

図 4-20　ディプロイメント図例

4.5 開発プロセス

4.5.1　近年のソフトウェア開発の特徴

近年のソフトウェア開発の特徴としては、

- 開発工期が極端に相対的に短い
- 要求仕様が固まっていない状態で開発を始めるケースが多い
- 途中で、要求の追加・変更が発生する
- 新しい技術や製品が次々に登場・改訂される中で開発しなければならない
- アプリケーション固有画面、Web、携帯など多くのインタフェースに対応しなければならない

などがあげられる。また、一方、開発時にヘッジしなければならないリスクには以下のようなものがある。

- 要求を誤解するというリスク
- 要求された速度が出ないなどの性能面でのリスク
- 適切な技術を持った人材が確保出来ないリスクなど
- 社内での力関係での圧力や干渉などのリスク

プロジェクト管理とは、これらの想定されるリスクをうまく回避することが望まれる。しかし、従来のウォータフォールモデルでの開発では、これらのリスクを回避するようにマネジすることは困難である。そもそも、前提として要求が完全に定義されていることを条件とした開発手法である。

これらを回避するためには、システム全体を一気に構築するのではなく、システムを機能ではなく、ある場面で活用されるシステム毎に構築して、検証しながら開発する方法論が求められる。このためにユースケースを単位として、これ毎に開発を行うような方法が用いられる。また、このユースケース毎に、リスク度の高い順にプライオリティをつけて、リスクの高いものから開発し検証することで、全体のリスク度合を低減する方法が用いられたりする。

4.5.2　RUP(Rational Unified Process)とは

米国ラショナルソフトウェア社が Booch、Rumbaugh、Jacobson の 3 人（スリ

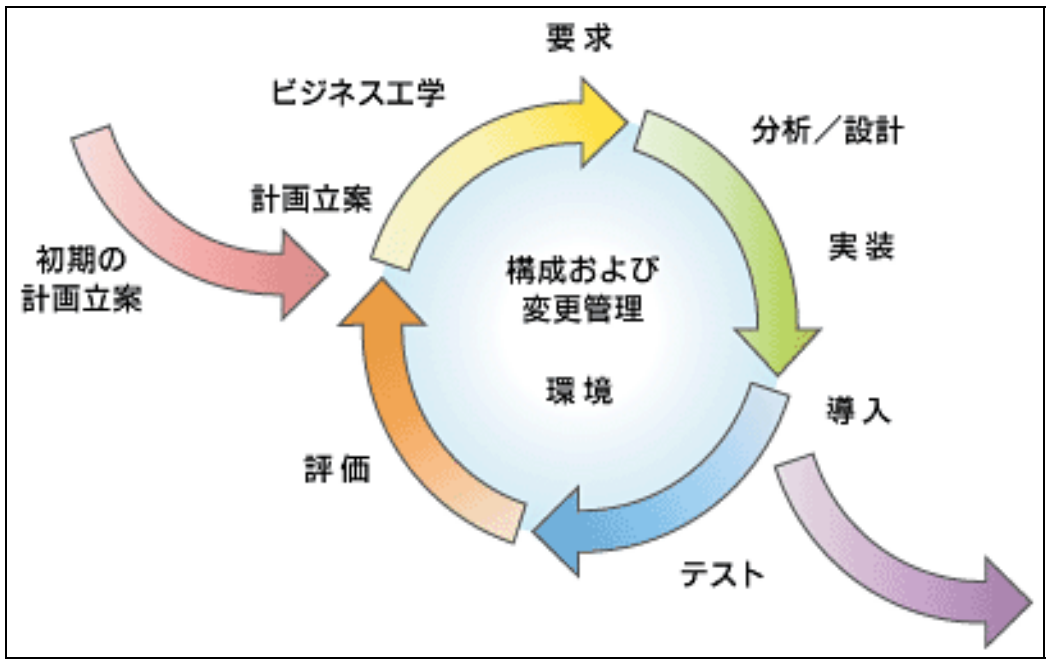
ーアミーゴという愛称）のそれぞれの手法に加えて、自社で培われてきた経験をはじめとした業界におけるさまざまなプラクティスを反映し、オブジェクト指向の分析/設計に**とどまらない包括的な体系として構築**したオブジェクト指向開発プロセスとしてまとめたのが、RUP である。これは、以下の根本的な考え方を持つ。

- 反復型開発（Interative Development）
- ユースケース駆動（Use Case Driven）
- アーキテクチャ中心（Architecture Centric）
- リスク駆動（Risk Driven）
- カスタマイズ可能（Configurable Process）

4.5.2.1　反復型開発（Interative Development）

小さなリリースを繰り返しながら全体的なシステムを構築していくというアプローチである。リリースのたびに統合作業が繰り返され何度も検証されるため、最後になってから致命的な問題が発覚するという問題などない。このことで、要求を誤解するというリスクを回避する。

このリリースのことを反復（Interetion：イテレーション）と呼ぶ。



(参考)出典：いまなぜ開発プロセスを注目するのか？（後編）株式会社豆蔵
HTTP://WWW.ATMARKIT.CO.JP/FJAVA/DEVS/PROCESS02/PROCESS02_1.HTML　より

図 4-21　反復型開発の流れ

4.5.2.2　ユースケース駆動（Use Case Driven）

ユースケースは、ユーザから見たシステムの使い方であり、それを、ユーザが何のためにどのように使うのかを表現し記述したものである。このユースケースをベースに開発を進める考え方が、ユースケース駆動という考え方である。システムのユーザ視点からみた部分毎に作り上げていくことで、要求内容を検証しながら作り上げていくことになる。

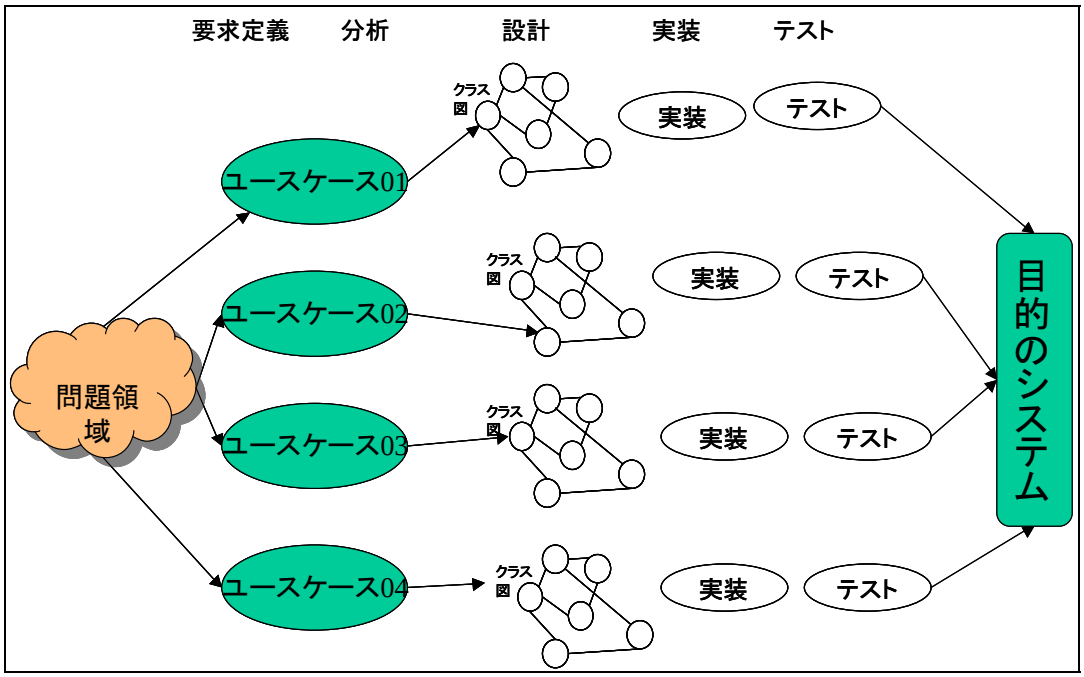


図 4-22　ユースケース駆動

4.5.2.3　アーキテクチャ中心（Architecture Centric）

ユースケースはユーザからの使い方を中心としたとらえ方であったが、一方、機能以外の要求、例えば性能や信頼性、拡張性などを考慮する必要がある。この性能等を保証するシステムの骨組みが、アーキテクチャである。このシステムの骨組みを中心とする考え方が、アーキテクチャ中心の考え方である。

4.5.2.4　リスク駆動（Risk Driven）

プロジェクト管理というのは、ある意味でのリスク管理である。開発ライフサイクル当初に、リスクの大きい物を対処することで、初期段階でリスク

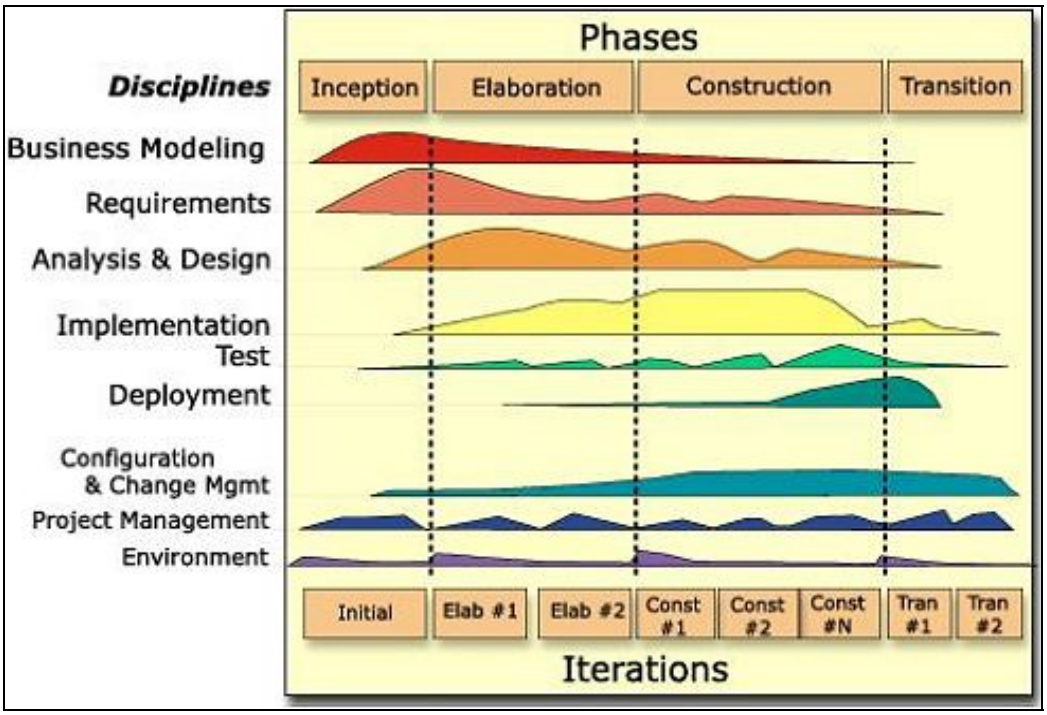
を解消しておくという考え方をとる。このことで、最終局面で、致命的な欠陥が明らかになることを防ぐ。また、リスクというものは開発が進むに従って変化していくため、反復開発で、日々変化するリスクを管理していくことが開発を成功に導くポイントである。

4.5.2.5　カスタマイズ可能（Configurable Process）

従来の開発手法の問題点として、実際はさまざまな組織や人材の違いがあるにもかかわらず、1つの手法で賄おうとしたことと、日進月歩で進化する新しい技術に対応できなかったことがある。これらを、解消するために、オブジェクト指向技術を用いた開発を RUP のフレームワークとして構成されており、それを、それぞれの組織、プロジェクトにフィットした形にカスタマイズして利用するという使い方をすることになる。

また、開発プロセス自体のバージョンアップも、ベンダであるラショナルソフトウェア社が行うことで、常に新しい技術に対応することが可能となる。

4.5.3　RUP としての全体像



出典：いまなぜ開発プロセスを注目するのか？（後編）株式会社豆蔵

HTTP://WWW.ATMARKIT.CO.JP/FJAVA/DEVS/PROCESS02/PROCESS02_1.HTML　より

図 4-23　4つのフェーズ

4.5.3.1　4つのフェーズ

- 方向づけ(Inception)
  システムの提供する機能を明確にし、システム開発におけるリスクなどを検証するフェーズで、重要なユースケース（全体の 10～20%）を抽出し、主要なシナリオに対して少なくとも１つのアーキテクチャを提示して、デモが出来るようにすること。また、潜在的なリスクもここで洗い出す必要がある。

- 推敲(Elaboration)
  システムの骨格を構築するフェーズで、このフェーズの重要な生産物はアーキテクチャベースライン。ユースケースの 80%がこの段階で検証される。また、使い易さやパフォーマンスといった、機能外の要求を分析するための補足要求定義書の作成と、実行可能なアーキテクチャプロトタイプの作成を行う。

- 作成(Construction)
  文字通りシステム構築を行うフェーズで、このフェーズが完了した時点で開発作業の 90%が完了したと考えてよい。すなわち、ユーザに引き渡す準備が出来た製品（βリリースやユーザマニュアル）が出来上がっていることになる。

- 移行(Transition)
  システムをユーザ環境に適用していくフェーズ。わかりやすい例で言うとシステムのベータリリースによりトランジッションフェーズが始まる。ユーザからのフィードバックとそれに対応する作業という形でトランジッションフェーズは推移していくことになる。

各フェーズの割合

フェーズ	スケジュール	作業量
方向づけ	10%	5%
推敲	30%	20%
作成	50%	65%
移行	10%	10%

4.5.3.2 コアワークフロー

- 要求定義(Requirement Capture)
  RUP では、要求を「システムが準拠しなければならない条件又は機能」と定義している。**「顧客の言語」でシステムが何をするのかを記述する。**
  成果物としては、
  - ✧ ユースケースモデル
  - ✧ 補足仕様書
  - ✧ ユーザインタフェースのプロトタイプ

  が作成される。

- 分析(Analysis)
  理想的な条件の中でのシステムの構造を定義する作業
  より正確な要求定義、開発者の言語への変換、アーキテクチャといった目的もある。成果物としては、
  - ✧ ユースケース図
  - ✧ ユースケースビューでのパッケージ階層
  - ✧ クラス図

  が作成される。

- 設計(Design)
  分析で得られたシステムの構造をより現実的なファクタを加味して具体化していくこと。成果物としては、
  - ✧ ユースケース図
  - ✧ ユースケースビューでのパッケージ階層
  - ✧ クラス図

  が作成される。

- 実装(Implemntation)
  主にプログラミング作業になるが、プログラムの物理的な構成を表現するコンポーネント図、プログラムのシステム上の配置を表現するディプロイメント図の作成もこのフェーズで行う。成果物としては、
  - ✧ 統合ビルド計画書
  - ✧ 実装モデル
  - ✧ コンポーネント

  が作成される。

- テスト(Test)
  テストケースを実行してテストを行うフェーズで以下のようなテストを実行する。
  - ✧単体テスト
  - ✧統合テスト
  - ✧システムテスト
  - ✧受け入れテスト

4.6 要求収集でのユースケース

ここでは､UML の中でも特異的な位置を占めるユースケースについて記載する。ユースケースが要求定義の手法として他の方法に取って代わることを示す｡また､ビジネス･プロセスモデル自体の記述にも今後使われていくと予想される。4.6 の節は、以下の書籍による。

ユースケース導入ガイド　成功する要求収集テクニック

ダリル・クラク、イーモン・ギニー　著　株　ピアソン・エデュケーション

4.6.1　要求とは

> 要求定義とは、コンピュータプログラミングの力によって、コンピュータがある問題領域において発揮することを期待されている効果である。
> By Benjamin L. Kozvitz

左記はすなわち､ユーザのためにコンピュータが実行しなければならないことであり、システムが、その存在に値するものになるために､提供しなくてはならないある特定の機能､特徴、あるいは原則である。また、その要求を満たすことが､ソフトウェア開発プロジェクトの提供範囲である。

一方、要求を見失う典型的なパターンとしては次のような事柄があげられる。

- 要求定義時に、設計に関する考察が入り込むことで、ユーザにわかりにくくなる。
- 曖昧さがあること。
- コンピュータ業界特有の用語の使用でユーザの理解を得られない。

4.6.2　2 つの機能

また、要求自体には次の 2 つのものがある。本来､システムがユーザに提供すべき、**機能的要求**そのものと、システムが満たさなければならない「～～ility」という単語で示される**非機能要求**（Scalability など）がある。

4.6.3　要求収集の困難性

ユーザが必要としているものを知るということの難しさは、Steve Mcconnel の“Rapid Development”によると、以下のような要因が挙げられる。

- ユーザ自身が何を望んでいるのかを理解していない。
- ユーザが記述された要求セットに対して意見を明らかにしない
- 費用とスケジュールが決定された後に、ユーザが新しい要求を強く主張する。
- ユーザとのコミュニケーションが迅速に行われない。
- ユーザがレビューに参加しないことが多い。あるいはレビューに参加できない。
- ユーザが専門知識を十分に持っていない。
- ユーザがソフトウェア開発プロセスを理解していない。

等々がある。

4.6.4　要求収集に求められるもの

ユーザの時間を尊重して、ユーザとのリレーションシップを確立することで本当にユーザの必要とするものを知り、それのトップ･コミットメントを得ることが必要である。また後工程との連携では、後工程に役に立つような冗長でないドキュメントを作成することである。また、その際に、要求分析者と設計者・開発者の信頼を確立して、設計に関する時期尚早な推測を回避する事が肝要である。（ともすれば、要求定義段階で設計のことまで考える事が有るが、これは排除されるべきである。）それと、今後、絶対に必要とされるものは、
要求のトレーサビリティである。要求と、システム（クラス）との対応が、今後、保守段階での工数を減らすこととなり、開発側での冗長なシステム構築などを排除できる。

4.6.5　伝統的要求定義手法の問題

- 要求仕様一覧
  各機能を個別に抽出して扱うことが出来るかのように単に様々な機能を

項目として明細化したもの。

- DFD(Data Flow Diagram)
  要求の中に、この時点では必要のない数多くの技術的要素を持ち込む。プロセス→プロセス→データストア（内部の事柄）
- ERD(Entitiy and Relation Diagram)
  ユーザが理解するためにはDFDとの組み合わせが必要となる
- プロトタイプ
  画面デザインなどの細部へユーザの関心を移行する。既に稼働している画面と勘違いさせる恐れがある。

など、従来の手法は問題を抱えている。

4.6.6　ユースケースへの移行

要求定義で、重要なのはユーザとコンピュータの相互作用であり、ユーザにとってコンピュータはブラックボックスであり、「中に入っていくもの」「外に出てくるもの」というコンテキストで記述することが必要である。そのためには、ユーザとシステムの相互作用だけを示してくれる、ユースケースが必要である。ユースケースは、「What」だけを示してくれるツールであり、DFD、ERD は、What,Howの両方となり混乱を招く。

ユースケースは、その単純さ故に、偉大なコミュニケーション・ツールである。

4.6.7ユースケースの目標

- アクタに価値を提供する相互作用を導き出す。
  ユースケースは、システムとシステム外部にある実体との間の相互作用を見せること。
- 実装に固有な言葉の排除　Context Free
  特定の人々、特定の組織
  ユーザインタフェースのウィヂット
- 詳細レベルをユーザにとって適切なものにする。
- 分量をユーザにとって適切なものにする。

非常に大きいシステムでも 70~80 を越えるべきでない。
通常、20~50 のケース[1]

という、ユースケースを描く際の注意がある。

[1] Ivar Jacobson 1996 Ooplsa(Object-Oriented Programming System,Languase System,Language,And Application

4.7 モデリングの実際

4.7.1　問題領域の把握

今回、UML を理解する為に、手軽で誰にでも理解しやすいモデルとして、JUAS にて行っている教育事業を取り上げてみた。教育事業の中でも、色々な企業向けに開催しているオープンセミナ（ここではすべてセミナと表記）にターゲットをあてていく。既に事業を行っているうえでの業務フローがあることから、まず事例として検証を行った。

課題：JUAS セミナ管理システム

全体の業務の流れ

① セミナの企画
セミナ担当者は、セミナの企画を行う。企画内容は、セミナ目的・内容（テーマ、タイトル）、講師の選定、開催日時、開催場所、参加予定数、予算・収益を企画書としてまとめて、企画会議にかけて承認を得る。

② セミナの募集
企画書実施の承認を得た段階で、セミナの募集を行う。募集は、E-MAIL で行い、対象者は、JUAS 会員で E-MAIL での連絡許可をもらっている方と、過去のセミナ受講者で同系統のセミナテーマに参加した人を抽出した人に送付する。送付方法は、既存のメールシステムにアドレスと案内内容を渡すことで送付される。

③ 受講者の登録
受講者からの連絡は、E-MAIL またはファックス、郵送等になる。これら申込みは、システムに申込者として登録される。

④ 経理処理
締め切った段階で、申込者名簿は経理に渡され、経理から請求書が送付される。また、開催日が近づいた段階で、申込者には再度案内のメールを送付する。

⑤ 開催前準備
開催前準備としては、テキストの印刷、受講者名簿の印刷など資料を取りそろ

える。開催時には、出欠をとる。

⑥ 開催後の後始末

開催終了後は、講師許可が出た資料は、JUAS ホームページにダウンロードできる様に掲示する。また、セミナ妙録をホームページに掲載する。

⑦ 経理処理

セミナ料金未納者には、経理より催促をする。

4.7.2　ユースケースによる要求定義

前段の業務の流れから、システムが外から見てどのように動作して、どのような反応を返しているかを明確にするためにユースケース図を記述する。

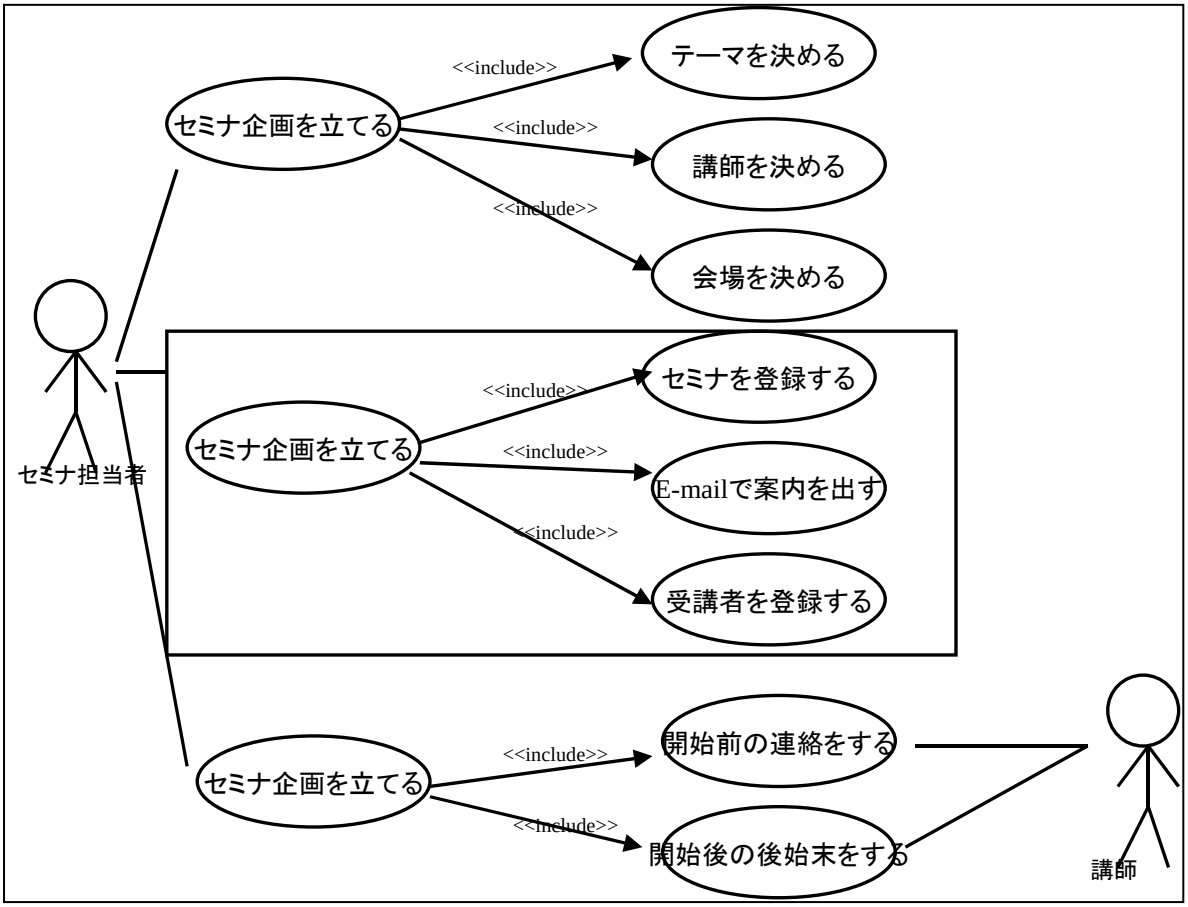


図 4-24　ユースケース図 1

このユースケース図を元に、IT 化すべき領域を、上図の「セミナを募集する」というユースケースと判断し、そのユースケース図を詳細に記載した。

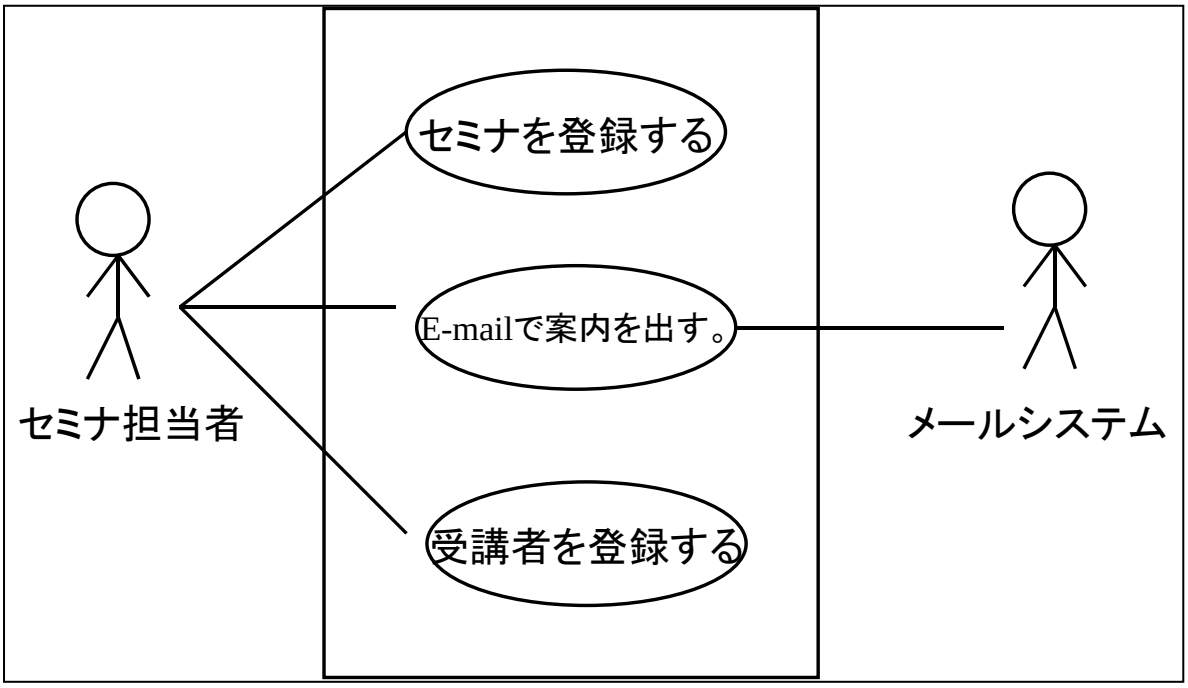


図 4-25　ユースケース図 2

4.7.3　シナリオ作成

ユースケース名	U01:セミナを登録する
アクタ	セミナ担当者
概要	計画されたセミナをシステムに登録する。
要件	
事前条件	セミナのテーマ、タイトル、講師名、開催日、開催時間、セミナ料金、会場、案内文が事前に決められていること。
事後条件	該当セミナが登録されていること。
イベントフロー	1. セミナ担当者は、すでに決まっているセミナテーマ、タイトル、開催日、開催時間、セミナ料金、受講料、会場、案内文を登録する。 2. セミナ担当者は、講師名簿から講師名を検索して、登録する。
例外フロー	
備考	

ユースケース名	U02:E-MAIL で案内を出す。
アクタ	セミナ担当者
概要	JUAS 会員と過去のセミナ受講者名簿から案内送付対象者を、E-MAIL 送付先として抽出する。
要件	
事前条件	セミナタイトル、講師、開催日、開催時間、セミナ料金、案内文が事前に決められていること。
事後条件	E-MAIL 送付先一覧ができあがっていること。
イベントフロー	1. セミナ担当者は、JUAS 会員からメール連絡許可をもらっている会員を抽出する 2. セミナ担当者は、今回セミナテーマと類似の過去セミナをセミナ台帳から検索して、その過去受講者一覧を抽出する。 3. セミナ担当者は、上記のメール許可された会員と、類似テーマを受講している過去受講者を、E-MAIL 送付先一覧とする。 4. セミナ担当者は、E-MAIL システムに上記宛先にメール送付を依頼する。
例外フロー	
備考	

ユースケース名	U03:受講者を登録する。
アクタ	セミナ担当者
3　概要	4　メールにて送られてきた受講申込書をもとに、セミナ申込者受講者をシステムに登録する。
要件	
事前条件	受講申込書がメール、ファックス等で送られてきていること。
事後条件	受講申込み者がシステムに登録されていること。
イベントフロー	1. セミナ担当者は、メール、等で送られてきたセミナ申込者をシステムに登録する。 2. セミナ担当者は、メール送付者に該当者があれば、その情報を流用する。
例外フロー	
備考	

以下、これらのユースケースについて要求ワークフローを進めていくことにする。

4.7.4　オブジェクト抽出

ユースケースから名詞を取り出す。これが、オブジェクトの候補ないしは、オブジェクトの属性となる。

名詞	対象
セミナ担当者	システムの利用者(アクタ)
セミナ	オブジェクトの候補
テーマ	オブジェクトの候補
タイトル	セミナクラスの属性
講師名簿	オブジェクトの候補
講師	講師オブジェクトの属性
開催日	セミナクラスの属性
開催時間	セミナクラスの属性
セミナ料金	セミナクラスの属性
案内文	セミナクラスの属性
会場	セミナクラスの属性
セミナ台帳	オブジェクトの候補
JUAS 会員	オブジェクトの候補

過去受講者	オブジェクトの候補
E-MAIL 送付先一覧	オブジェクトの候補
セミナ申込者	オブジェクトの候補
E-MAIL システム	外部システム(アクタ)

ここでは、JUAS 会員、過去受講者と有るが、実際には JUAS の会員から、メール連絡を許可されている(複数の)会員を示している。従って、JUAS 会員のある特定の人を指し示すのではなく、ある集合から条件を満たす集合を選び出すことである。

従って、これらは、正確には JUAS 会員名簿から連絡可能な会員名簿を抜き出すという事になる。同様なことは、過去受講者にも言えることであり、ある特定の過去受講者(A さんということではなくて)、同一テーマを受講している過去受講者の集団である。従って、過去受講者名簿から、同一テーマを受講している過去受講者名簿を抜き出すということになる。

従って、(日本語では曖昧である)JUAS 会員、過去受講者ではなくて JUAS 会員名簿、過去受講者名簿と表現するのがオブジェクトとしては正確に表していると考えられるので、JUAS 会員を JUAS 会員名簿、過去受講者を過去受講者名簿という命名し、それをオブジェクトする。

4.7.5 オブジェクト間の相互作用

ユースケースから導出されたオブジェクト間のメッセージの流れをとらえることで、オブジェクト同士で行う振舞い(機能)を理解することが出来る。この目的で、シーケンス図やコラボレーション図を作成する。

ここでは、アクタであるセミナ担当者が、システムに介入して、登録や検索などの作業を実施することになる。コラボレーション図等でアクタそのものを記述してもそれとの相互作用と言うことでの記述もあるが、実際にはシステムとアクタとの間には何らかのインタフェイスが考えられる。

ドメイン分析から、システム分析という橋渡しということでロバストネス分析という方法では、これらのユースケースを下記のような記号で、途中で分析することが出来る。これを用いることで、画面やデータベースのあり方を明確にして MVC モデルとして明確に実装のあり方を明示することが出来る。但しここでは、アクタがある画面(セミナ登録など)を操作することが、容易に想像出来るので、この様な分析過程を経ずして、その様な画面というインタフェイスがあることが自明なこととして、シーケンス図やコラボレーション図に××画面として記載する。

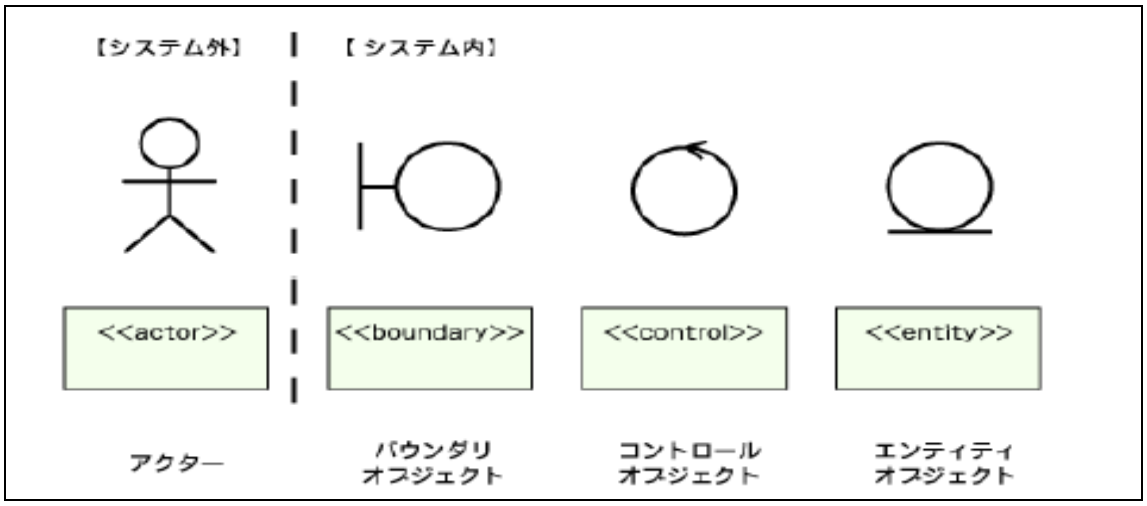


図 4-26　オブジェクト間の相互作用

ユースケース:U01 からは、

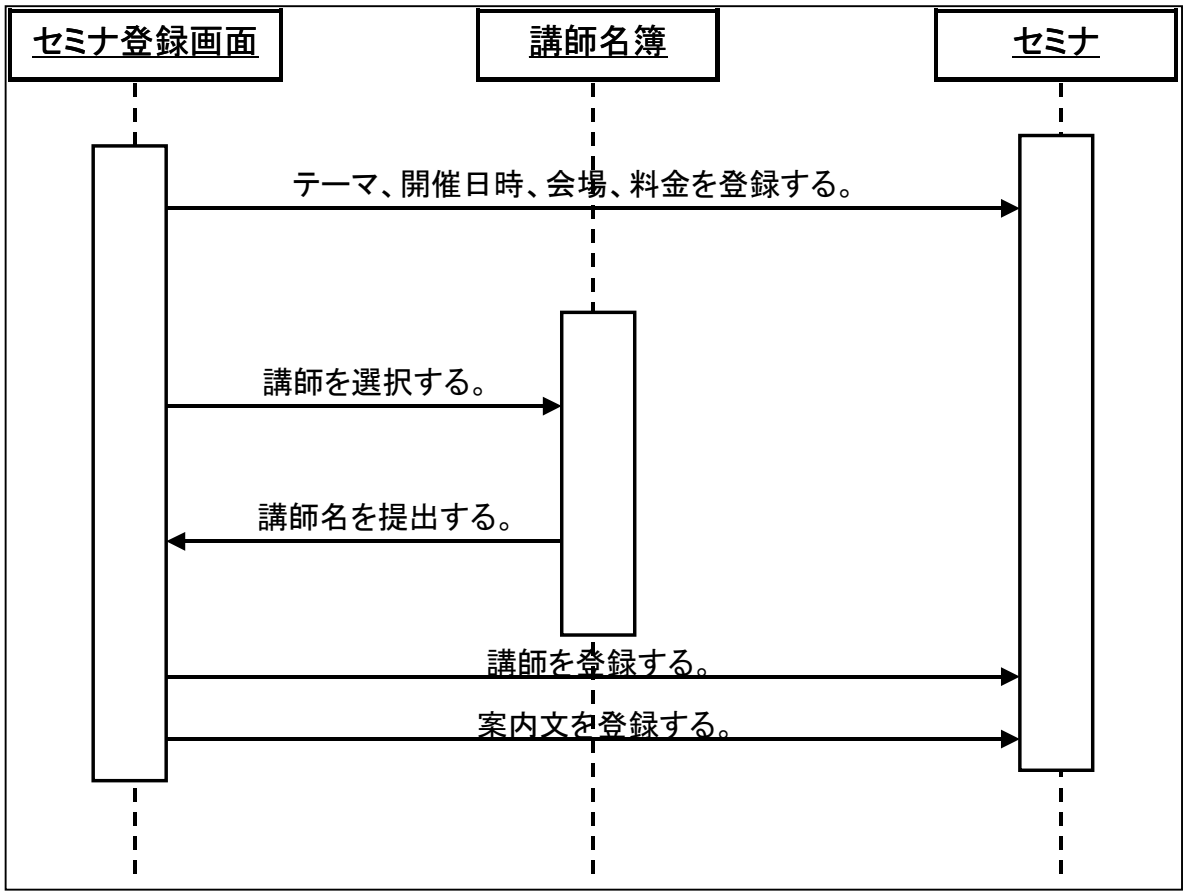


図 4-27 シーケンス図 1

ユースケース:U02 からは、

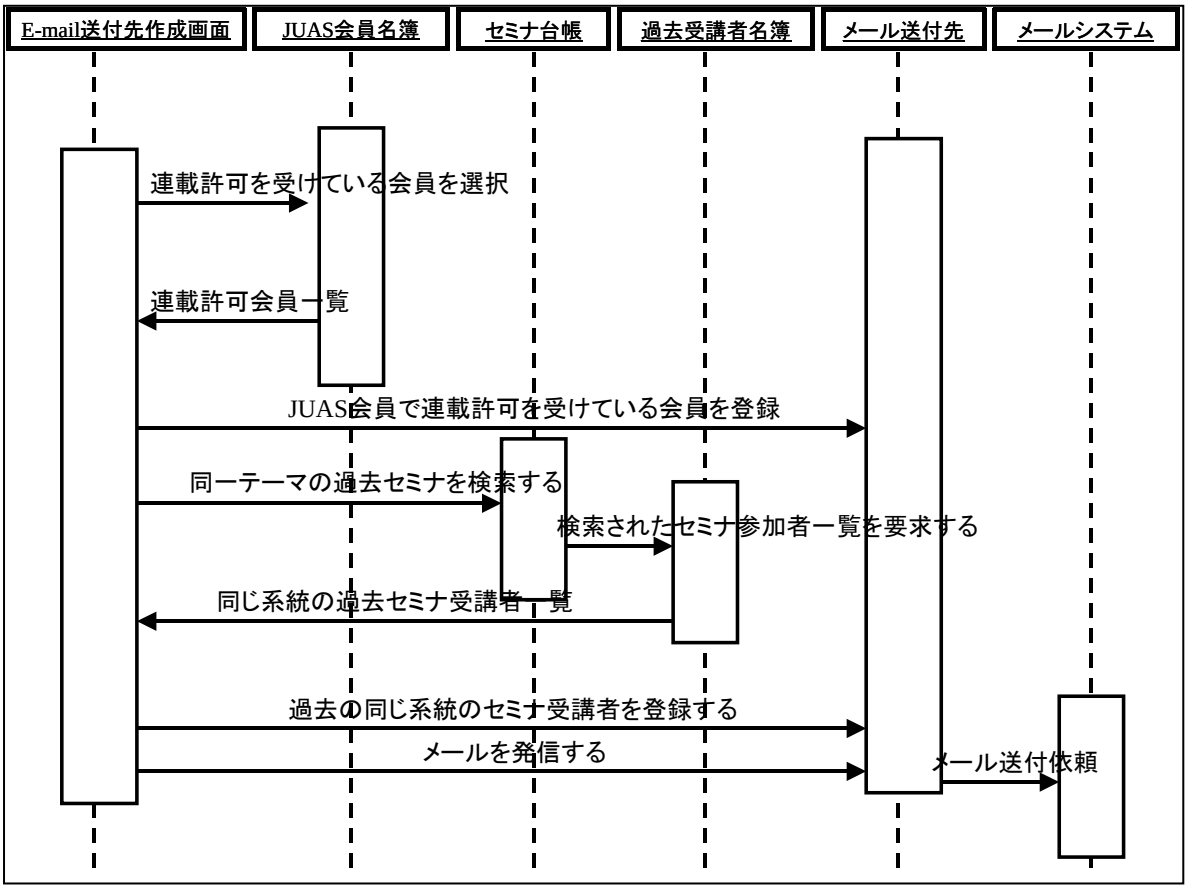


図 4-28　シーケンス図 2

4.7.6　クラス設計

シーケンス図やオブジェクト図から、下記の様なクラス図を導出する。

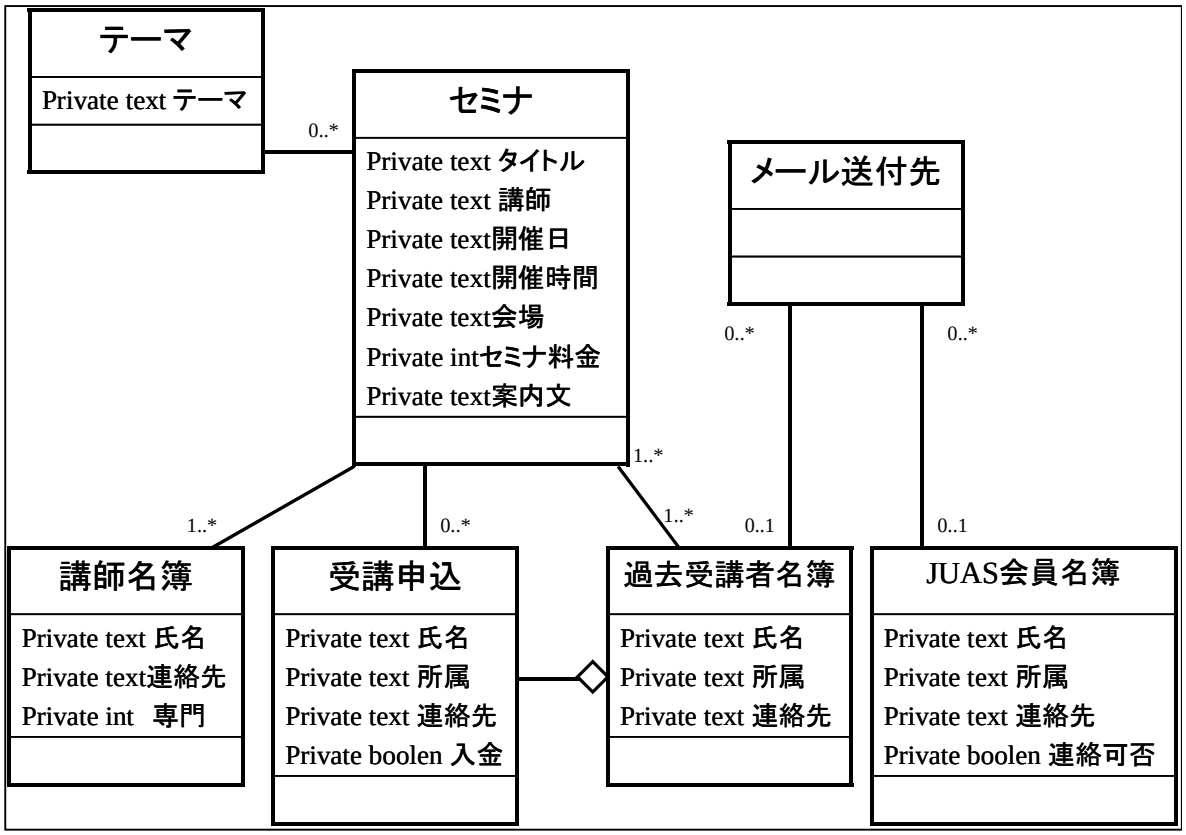


図 4-29　クラス図

以上によって要求ワークフロークラス図までの作成は出来た。次は、これらを分析・設計ワークフローへ進めて、実装まで進むことになる。

5. ビジネスオブジェクト開発事例分析報告

5.1 目的と背景

各企業のビジネスオブジェクト指向開発の状況を、会員へのアンケートと企業訪問により調査分析を行う。

5.2 調査の要点

下記の項目でアンケートを行った。

Q1. オブジェクト指向開発の貴社における定着度合について。

＜IT 部門の方々＞

( )JAVA を使用している。
( )フレームワークを持っている。
( )UML を一部使用している。
( )UML を全面的に企業として採用している。
( )オブジェクト指向の普及技術サイドを備え、人材育成に利用している。
( )ビジネスオブジェクトとしての再利用を実行している。
( )再利用を目的としたビジネスオブジェクトを開発し利用普及している。

Q2. ソフトウェア資産の内、オブジェクト言語（JAVA、C++、C#）で作成したプログラムの割合は。

・JAVA　3 年前（　）％　現在(　)％　3 年後（　）％
・C++　3 年前（　）％　現在(　)％　3 年後（　）％
・C#　3 年前（　）％　現在(　)％　3 年後（　）％

Q3. オブジェクト指向言語が扱える SE、PG は、現在何人/割か。

（　）人/（　）割

Q4. オブジェクト指向言語開発により、どのような面でその効果がでているか。

( )生産性・価格
( )品質
( )工期
( )ユーザとシステム部門の理解の促進

( )その他（内容　　）

Q5. 御社の新規システム開発に際して、UML の活用の割合はどのくらいか。
3 年前（　　）％　現在(　　)％　3 年後（　　）％

Q6. UML を活用して、どのような面でその効果がでているか。
( )生産性・価格
( )品質
( )工期
( )ユーザとシステム部門の理解の促進
( )その他（内容　　）

Q7. UML を活用してオブジェクト指向設計・開発を実施する場合の、一番の問題は何か？（複数回答可・最大 3 つ）
( )良い教材・指導者がいない。
( )理解できる SE が少ない。
( )投資できる人材・期間が必要。
( )既存システムのしがらみが多い。
( )既存の技術に比較して、それほど優れた技術ではない。
( )利用する事によってのメリットが分からない。
( )積極的に活用する社の方針が無い。

Q8. オブジェクト指向設計が出来る技術者数は何人/割くらいか。
（　　）人/（　　）割

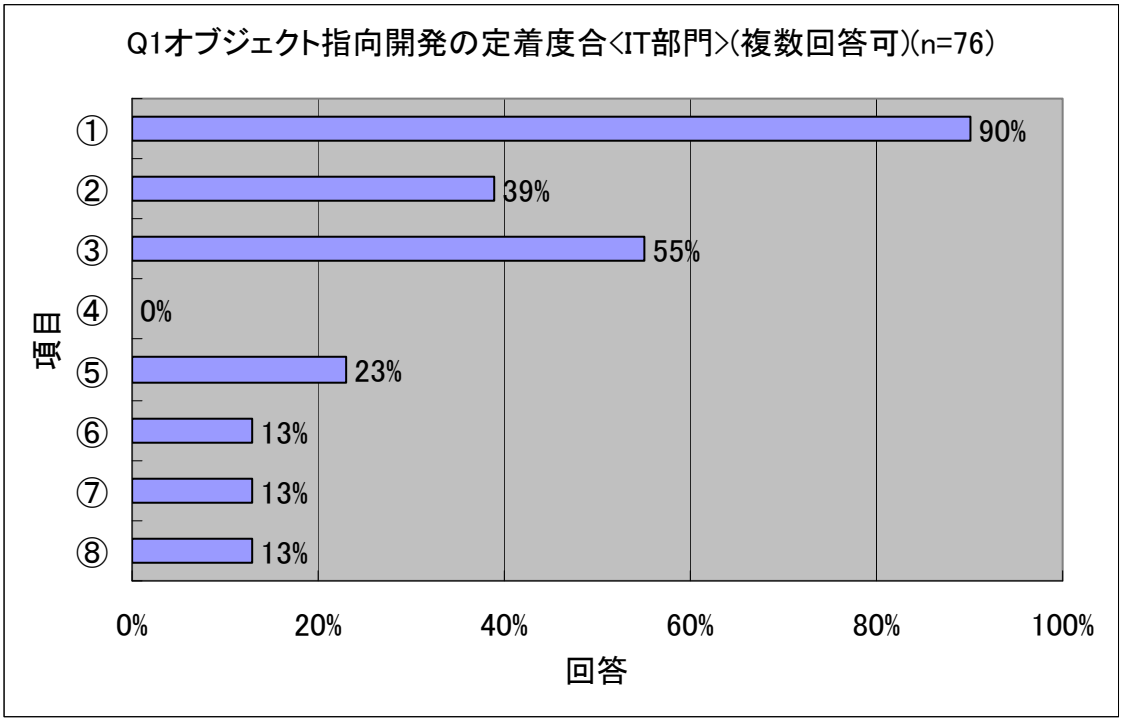
Q9. オブジェクト指向開発の教育は行っているか？
行っている場合、どのようなカリキュラムか？
( )行っている。
( )行っていない。
カリキュラム内容（　　）

5.3 事例の分析

Q1. オブジェクト指向開発の貴社における定着度合について。

＜IT 部門の方々＞

① JAVA を使用している。
② フレームワークを持っている。
③ UML を一部使用している。
④ UML を全面的に企業として採用している。
⑤ オブジェクト指向の普及技術サイドを備え、人材育成に利用している。
⑥ ビジネスオブジェクトとしての再利用を実行している。
⑦ 再利用を目的としたビジネスオブジェクトを開発し利用普及している。
⑧ その他



＜その他の回答＞

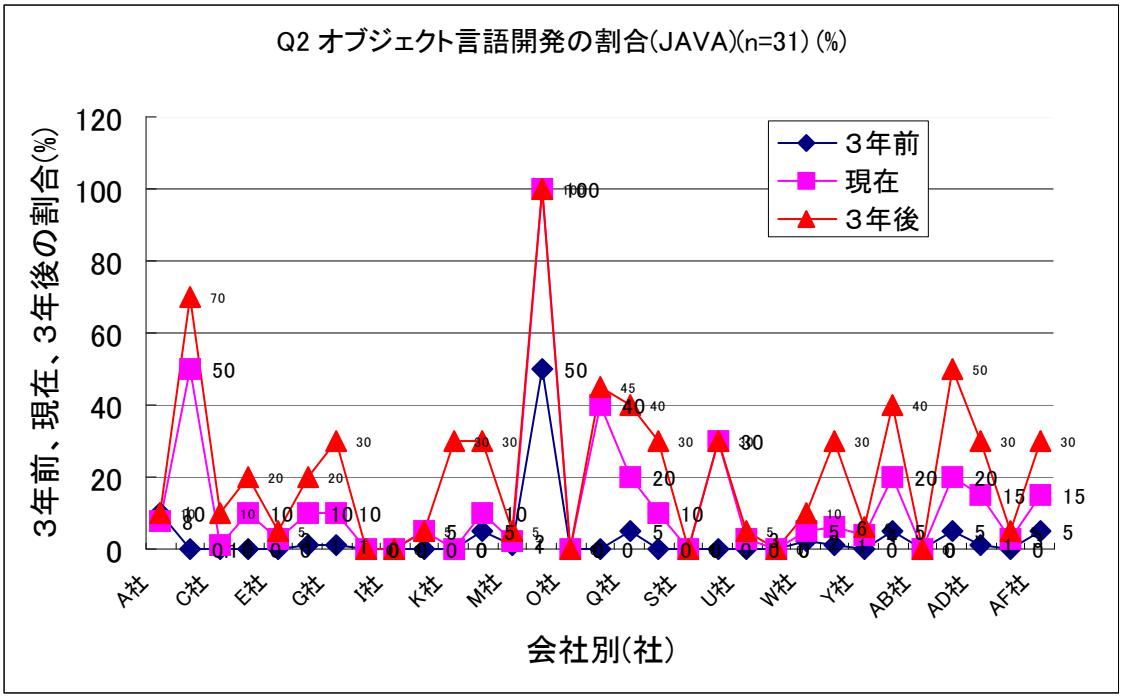
- 開発委託先で使用している。基盤オブジェクトのみ行っている。
- JAVA は使用しているが、クラス設計はしていない。COBOL 等と同じレベルでの利用。
- UML モデリングツールを一部で採用。
- UML に近い手法で実施している。

＜考察＞

各企業で、プログラミング言語の一つとして、JAVA を利用している（90%）が、UML は一部利用しているのみ（55%）で、全面的に企業として採用できていない（0%）。オブジェクト指向により、システムを部品化することで再利用・再構築しやすいシステムを開発する為の手段として期待しているところではあるが、実際はまだまだ普及していない状況が読み取れる。また、JAVA を使用しつつも、実際のロジックは、COBOL と同じようなプログラミングレベルであり、オブジェクト指向化しているシステムは、構築中といったところである。

また、まだまだオブジェクト指向での開発までいっていないが、人材育成に力を入れている（23%）という回答があり、将来性を期待している状況にある。

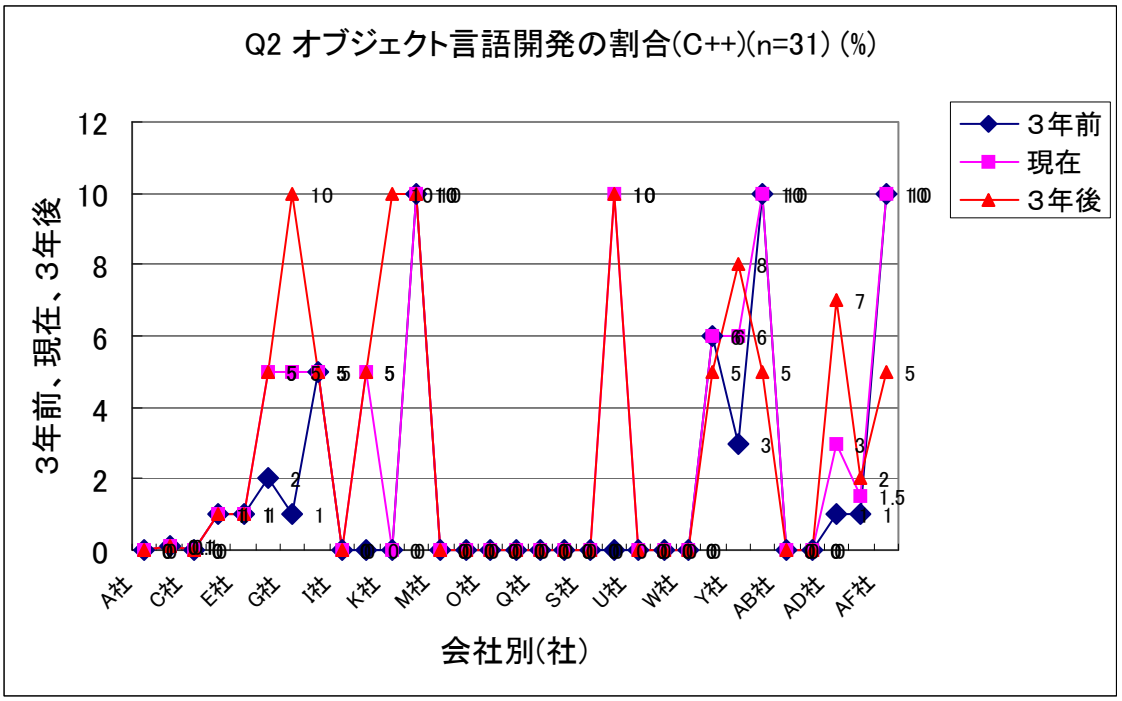
Q2－1．ソフトウェア資産の内、オブジェクト言語（JAVA、C++、C#）で作成したプログラムの割合は。　JAVA　3 年前（　　）％　現在(　　)％　3 年後（　　）％



＜考察＞

JAVA に対しての期待が伺える結果となった。3 年前はまだまだ開発言語としては定着していなかったものの、現在まず利用という事で、開発言語として取り入れている企業が増えている事が読み取れる。一企業に限っては、現在 100%開発言語として取り入れている企業もあった。他の企業も現在は、10%程度の企業がほとんどだが、3 年後は 20%～50%の割合で、JAVA によるオブジェクト指向型開発に移行するという予測結果になった。

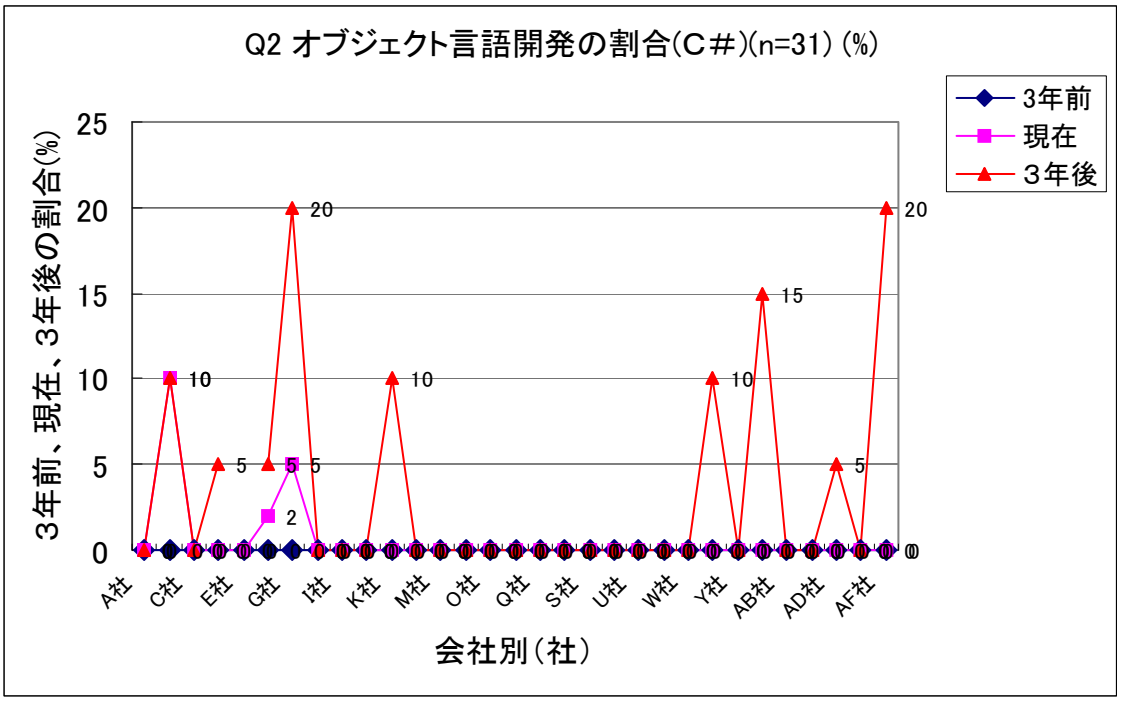
Q2－2. ソフトウェア資産の内、オブジェクト言語（JAVA、C++、C#）で作成したプログラムの割合は。　C++　　3 年前（　　）％　現在(　　)％　3 年後（　　）％



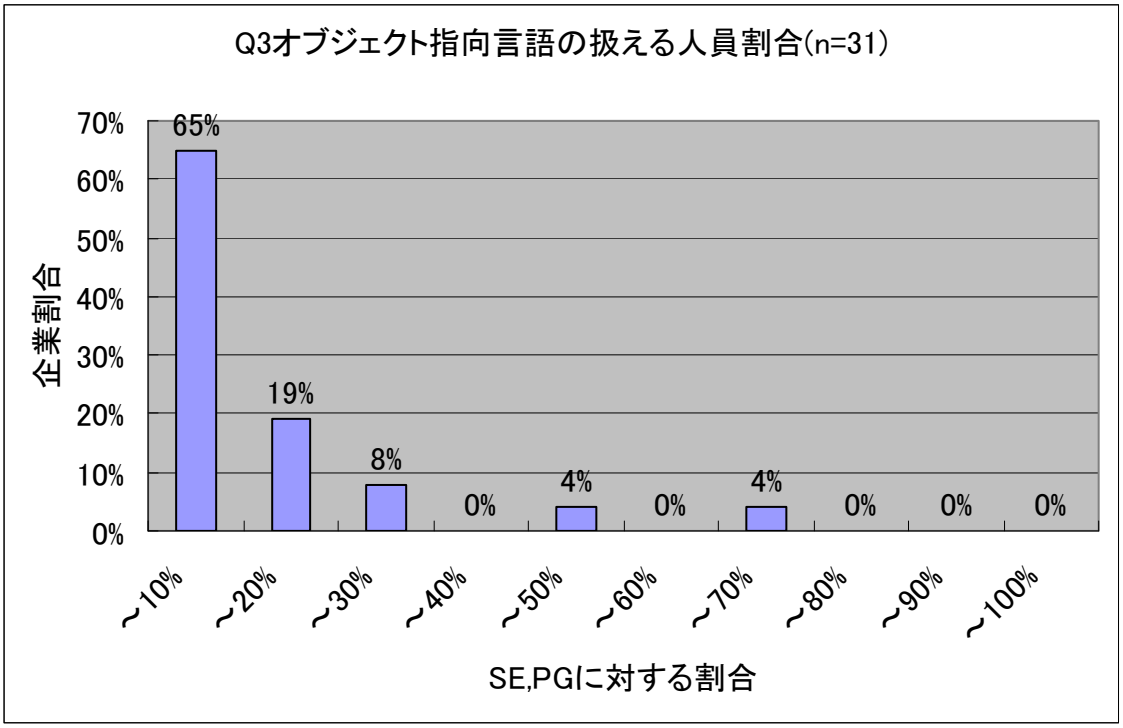
＜考察＞

回答にあまり統一性はないが、C++に関しては、3 年後大幅に割合が増えるという期待は無いようである。

Q2－3．ソフトウェア資産の内、オブジェクト言語（JAVA、C++、C#）で作成したプログラムの割合は。　C#　　3 年前（　　）％　現在(　　)％　3 年後（　　）％



Q3．オブジェクト指向言語が扱える SE，PG は、現在何人/割か。



＜その他回答＞

- 言語が扱えるレベル。
- 社員に対して SE や PG という明確な区分けをしていない。

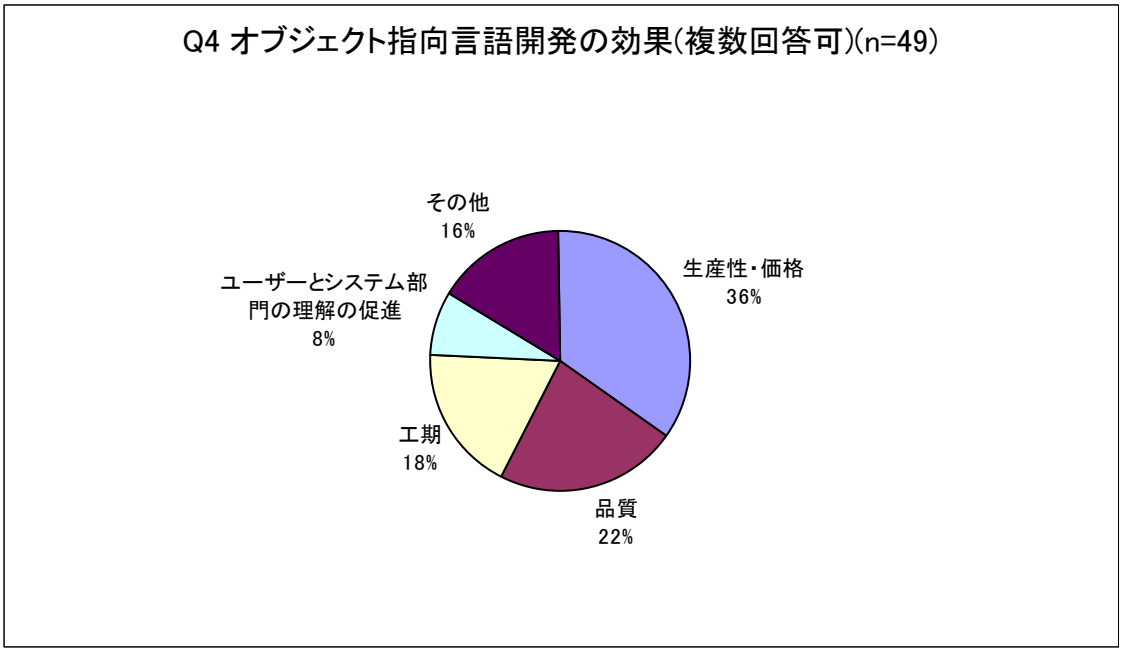
＜考察＞

オブジェクト指向言語が扱える社員が、10%の企業が 65%（17 社）であった。先のグラフ結果からも、期待してはいるが、現実開発できる人員を確保している企業は少ない。今後教育していく事で、オブジェクト指向言語が扱える社員を増やしてく企業の方針があることは、先のグラフ結果から読み取れる。

唯一、70%の社員が扱えると答えた企業では、現在 100%JAVA による開発を行っている企業で、オブジェクト指向型開発に企業を上げて取り組んでいる企業もある。今後、システムを再構築・再利用しやすいとされるオブジェクト指向型開発をユーザから求められる可能性は高いので、そのような企業が増えていくのではないか。

Q4. オブジェクト指向言語開発により、どのような面でその効果がでているか。

① ユーザとシステム部門の理解の促進
② 工期
③ 品質
④ 生産性・価格



＜その他回答＞

- プロジェクトによる。
- 先進技術。
- 効果殆どゼロ。現時点ではマイナス。まだ目に見える成果はない。
- 保守性の向上。
- いまだ試行錯誤状態。効果は？
- 効果が出ていない。
- WEB 関連のアプリ開発。

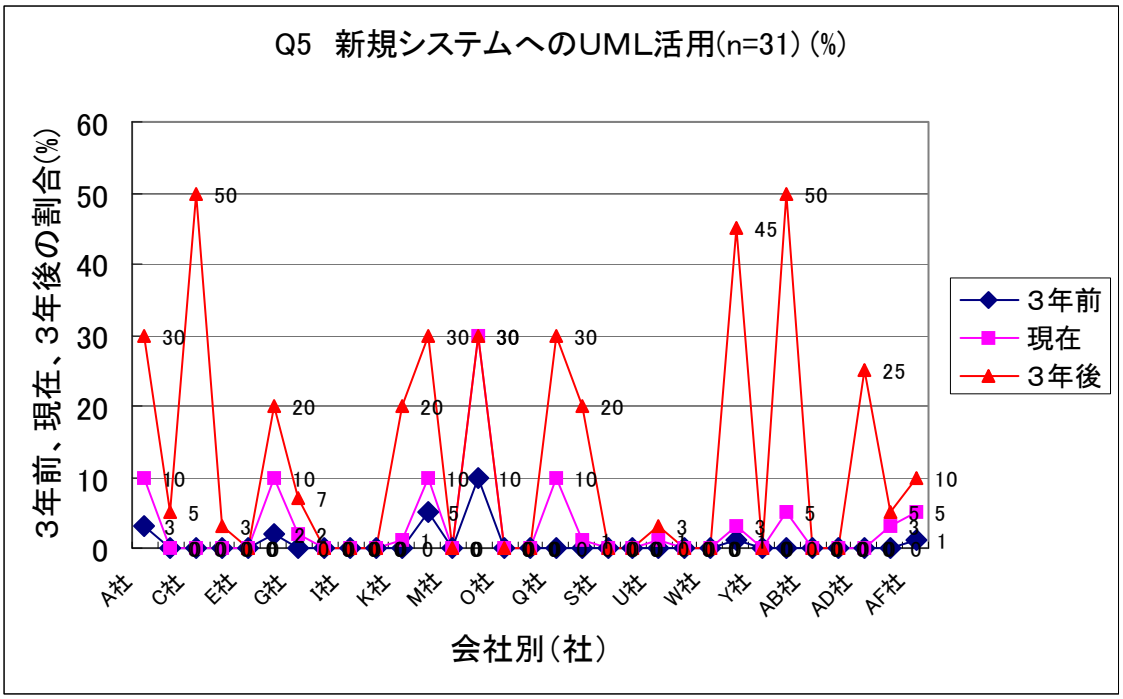
＜考察＞

オブジェクト指向開発に取り組んでいる企業は、どういう効果を求めているかという問に対して、第一位が生産性・価格（36%）であった。システムを部品化・オブジェクト化していくことで、同じ機能を別のシステムで使う際に、再度同じ機能を作るのではなく、利用できる事から、生産性に効果を表している・期待している企業が多いと読み取れる。

また、プロジェクトや開発するシステムによって、効果がまだまだ把握しき

れない状況も読み取れる。大規模プロジェクトになると、オブジェクト化するだけで時間がかかる可能性があり、生産性が逆に落ちているように見える時期もあるか。

Q5. 御社の新規システム開発に際して、UML の活用の割合はどのくらいか。

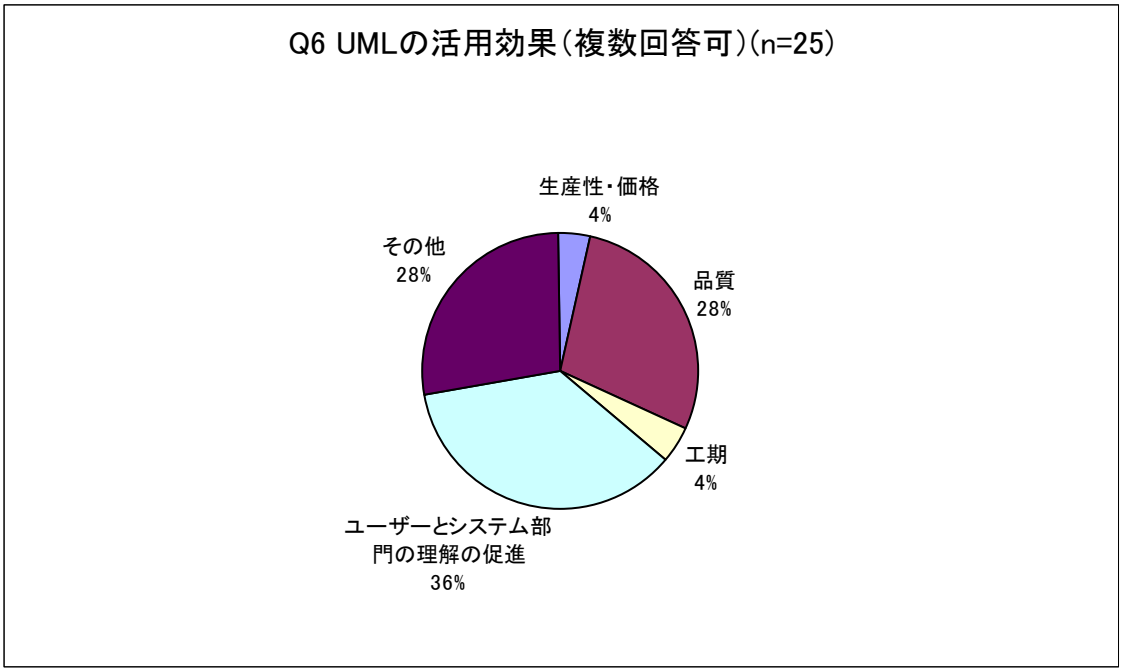


＜考察＞

Q1 にもあったように、まだまだ企業で UML を利用している企業でも、一部利用という回答が圧倒的であったが、この結果を見る限り、各企業で UML に対しての将来性を期待していることが分かる。3 年後には、新規システムを構築する際、UML を活用する場面が増えると答えた企業がほとんどであった。

Q6. UML を活用して、どのような面でその効果がでているか。

① ユーザとシステム部門の理解の促進
② 工期
③ 品質
④ 生産性・価格



＜その他の回答＞

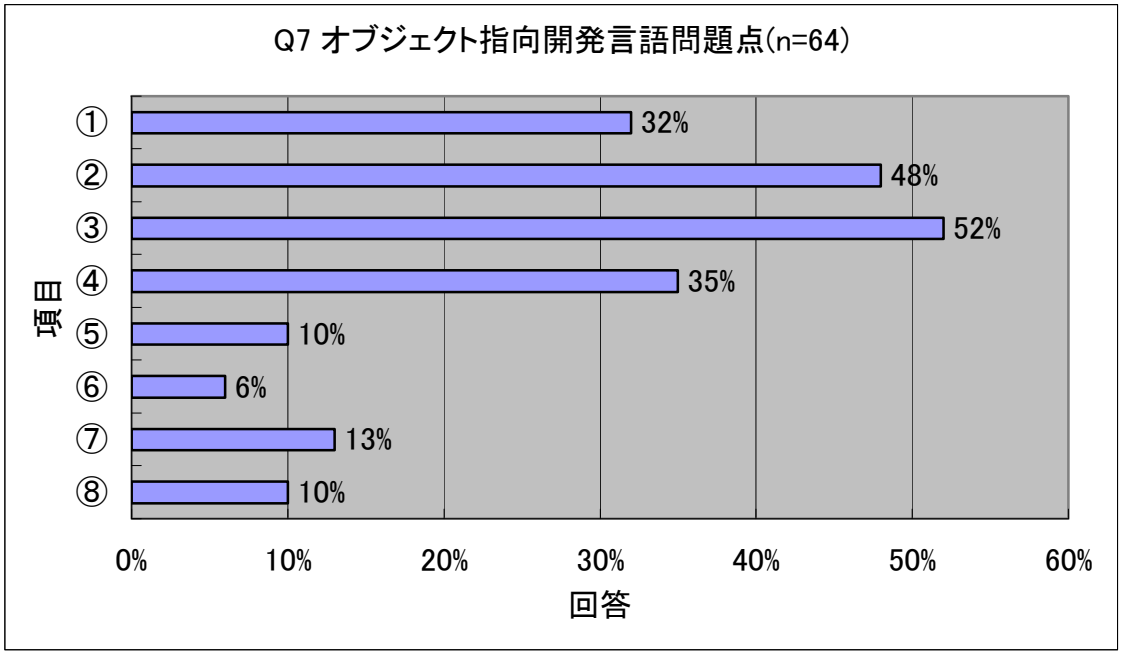
- 未活用のため、0。
- 明確に効果が出ているかは評価しておりません。
- 内部設計については生産性、品質、工期　。
- 未使用
- 実例が少なく、効果把握に至らず。
- いまだ試行錯誤状態。効果は？

＜考察＞

ユーザとシステム部門の理解の促進に効果があると答えた企業が 36%。ユースケース図を利用する事で、ユーザの業務プロセスを理解し、要求事項を明確にする手法として、UML を評価している事が読み取れる。先程の、「オブジェクト指向言語開発の効果」の際、一位だった生産性・価格に対しては、効果が出ている・期待している企業はほとんどない状況。UML は、儲ける為の効果より、品質やより戻り開発が少なくする為の手段として期待していることが分かる。

Q7. UML を活用してオブジェクト指向設計・開発を実施する場合の、一番の問題は何か？（複数回答可・最大 3 つ）

① 良い教材・指導者がいない。
② 理解できる SE が少ない。
③ 投資できる人材・期間が必要。
④ 既存システムのしがらみが多い。
⑤ 既存の技術に比較して、それほど優れた技術ではない。
⑥ 利用する事によってのメリットが分からない。
⑦ 積極的に活用する社の方針が無い。
⑧ その他



＜その他回答＞

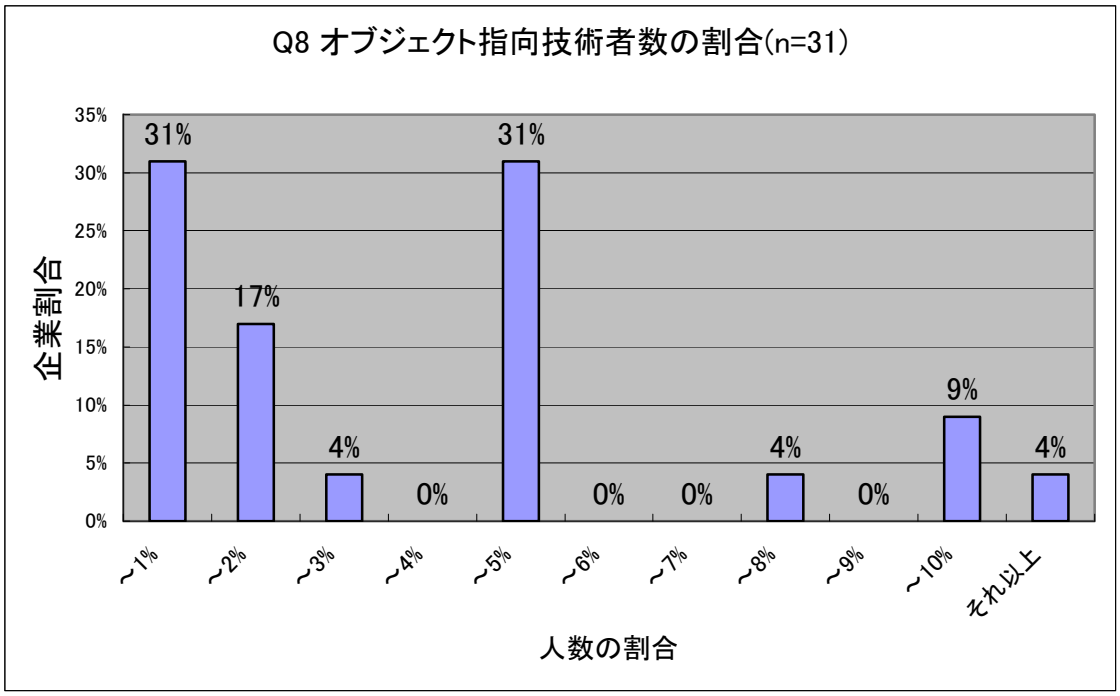
- 利用することによるデメリットのほうが大きいため、利用していない。
- 未使用
- CASE ツールが無ければ実用的な利用はできないと思う。現在手法の勉強程度に現在とどまっている。

＜考察＞

投資できる人材・期間が必要（52%）が第一位。理解できる SE が少ない（48%）が第二位の状況から、まだまだオブジェクト指向型開発は、理解・利用する事が難しい状況。将来性は感じているものの、実際に行ってみると、難しいという実感が各企業にあるようである。企業によっては、有効性が理解できず、デ

メリットの方がおおい、もしくは検討もしていないという回答もあった。教育するにも教育できる人がいないという現実。

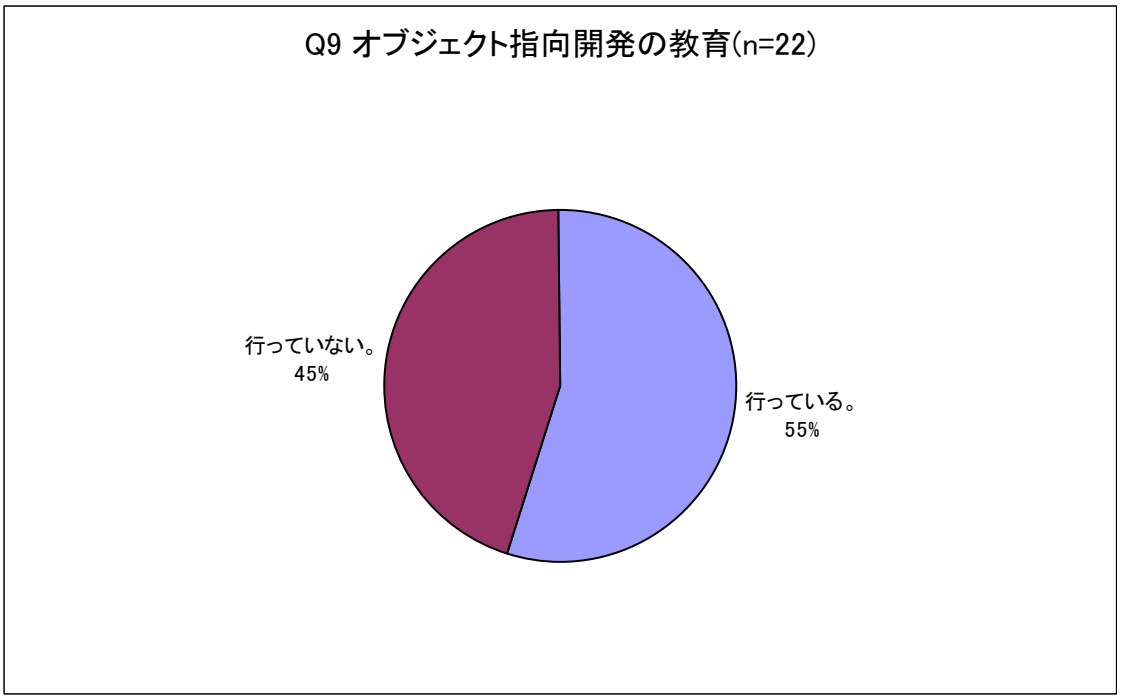
Q8．オブジェクト指向設計が出来る技術者数は何人/何割くらいか。



＜考察＞

1%～5%と答えた企業が 8 割。将来的には期待しているものの、まだまだ Q7 の結果からも、実際に出来る社員がいないのが現状。

Q9．オブジェクト指向開発の教育は行っているか？行っている場合、どのようなカリキュラムか？



＜カリキュラム内容＞

- プロジェクト固有の研修が多い。
- 基礎レベルの教育。
- オブジェクト指向設計、JAVA 入門。
- 自社開発のフレームワーク講座が中心です。
- 外部教育機関。
- サーブレット・JSP 演習。
- 新人向 JAVA プログラミング教育。
- 富士通等の講演会テキスト。
- 新入社員研修・WEB による教育資材。
- 概論レベルのみ，今後予定。
- DOA-RAD FOR JAVA。
- 社内研修。
- 「オブジェクト指向とは～」といった概要的なもの。

＜考察＞

Q7、Q8 と現状はオブジェクト指向開発が出来る社員はまだまだ少ないが、教育を行っている企業が 55%。教育といっても、入門レベルの企業がほとんどだ

が、ここでも将来性を期待している状況が把握できる。企業によっては、フレームワークを作成して、そのフレームワークの操作の教育なども含んでおり、今後どのように教育をしていくか、企業で検討中という状況である。

5.4 分析結果のまとめ

今回のアンケート全体から、下記のことが読み取れるのではないか。

- 将来的に、オブジェクト指向型開発は有効であると期待している。
- 各企業でシステムの再構築・再利用をするための開発手法として有効性は認識している。
- 教育したいが、まだまだ現状では理解できる社員が少ない。
- 新規システムの開発手法というより、フレームワークを作成し、それを利用していく方向で動いている。
- オブジェクト指向開発・UML 使用することのメリットをまだまだ感じられていない企業が多い。

6. 個別事例調査報告

6.1 A社

6.1.1 オブジェクト指向適用の取り組み 概要

A社では、1995年より工場における出荷管理、生産管理のシステムに対してオブジェクト指向を適用している。

1995年の適用開始当初のシステムは、約30KLS(キロライン・ステップ=1000ステップ)の規模で試行的な意味合いの強いものであった。1996 年以降の開発では、この試行で得られたノウハウを元に、プログラムの部品化、デザインパターン、フレームワークなどを積極的に取り入れ、約500KLS～約1500KLSという規模のシステム開発を成功させている。

6.1.2 オブジェクト指向適用例

6.1.2.1 オブジェクト指向適用の契機

- システム開発費用及びシステム運用費用の削減
- 生産工程管理のレベルアップ
- システム技術者人材育成

上記3点を契機にオブジェクト指向を適用。

6.1.2.2 オブジェクト指向の導入時の考慮点

- プロトタイプで試行し評価をする。
- Globalstandardを取り入れる。
- 有識者の協力を得る。
- 最初の1、2年は投資と考える。長期的な視野で投資を回収するという考えを持つ。

6.1.2.3 制御系（プロセスコントロール系)、操業系へのオブジェクト指向設計の適用可能性

制御系のシステムは、10年前くらいから近代化がされてきており、現在は、32ビットマシン程度の性能を持った設備になっている。制御系のシステムは、進化のスピードが遅くメーカーも力を入れない傾向にある。特殊なマシンと

なるためコストも大きく、容易に変更することは難しい。今後も大きな問題として残されていくと考える。

操業系のシステムは、30 年前から動いているシステムで、改修したいと考えている。投資額が大きいが、システム構造がスパゲッティ構造化している等の問題もあり必ず行いたい。

6.1.2.4 オブジェクト指向技術の適用範囲

A 社全体で分散システム（クライアントサーバ型システムと N 層型システムの総称）の割合は約 4 割。一般管理系システムは Visualbasic が中心。操業系システムはオブジェクト指向が約 50%。生産管理系システムは 50%以上がオブジェクト指向。

A 社の約 80%の社員が JAVA で実装ができる。

6.1.2.5 設計思想の変遷

オブジェクト指向導入時は、OMT をベース。最近は、UML を活用しているが、UML も OMT も難しい。

ヒュージョン 4 というものを適用しようと検討した時期もあったが、クラスズにこだわって、肝心な機能に力をいれられなかった。その後 UML が出たときにユースケース主導で設計を進めていくことになった。

CRC も導入した時期があったが、クラスが 50 個以上あるシステムに CRC を適用するのは難しい。現在は、教育時にのみ適用している。コラボレーションとレスポンシビリティを学ぶには良い。

6.1.2.6 UML の適用

オブジェクト指向でない部分でも使用される場合が、最近多くなってきている。システムの種類に関係なく、基本的な設計言語として使用されている。

ユーザとの意思疎通に使用するのは、ユースケースまで。クラス図などを見せても理解は難しい。形式にはこだわらず、ユーザが欲しいものは何かということをはっきり言ってもらうことに注力している。

UML の中では、ユースケース、クラス図、シーケンス図、オブジェクト図を使用している。

6.1.2.7 設計ツールの変遷

Rational rose くらいで、あまり使えるものは無かった。実際には、Rational rose も使用せず VISIO を使用した。

6.1.2.8　ODB の適用

オブジェクト指向導入当初は、ODB を導入した。オブジェクト指向導入当初の試行期間では、オブジェクト指向技術の獲得が優先課題であったため賢い選択であったと考える。

ODB には、Objectstore を使用した。

しかし、パフォーマンス等の問題があり、実用的では無かった。RDB と ODB の併用という策もあったが問題も多く最終的には RDB で一本化した。

6.1.2.9　その他 ODB

POET という ODB を使用したことがある。

SQL のように何か 1 つ知っておけばよいというものではなかった。また、ロックの粒度が大きくコントロールがし難くい上、挙動も RDB と比較し予測し難い。

6.1.2.10　開発推進体制

人材育成のため人を流動化させたが、キーマンは不動。

レガシーシステムのベテランをキーマンとして、キーマンが開発メンバーを決定する。

開発メンバーは、別の部署や外注会社からも登用し、新しい技術への対応と既存メンバーへの技術力の拡大を狙った。

プロジェクトのメンバーは、開発期間中は基本的に同一メンバーとしている。

6.1.2.11　ドキュメントの標準化

独自の開発作業標準がある。

企画、開発、運用、保守などのフェーズ別に項目が定義されている。

すべての項目をピックアップすると膨大な作業となってしまうため、プロジェクト開始時にテーラリングを行い、作業項目の中からプロジェクトにとって必要な項目をピックアップする。

6.1.2.12　プロジェクト管理

プロジェクト管理は、ウォーターフォール型が基本。GUI 関係など機能によっては、一部スパイラル型を使用している。

6.1.2.13　フレームワーク

「あらまほ」という自社開発のフレームワークをメインで使用している。

このフレームワークは、GUI とビジネスロジック間の処理の体系化と、共通コンポーネント、ソース生成のジェネレータ等を提供している。

6.1.2.14　部品の設計と変更管理

部品は、プロジェクト開始時点で体系も含めて決める。設計段階の変更管理は、メールや口頭、管理はドキュメントで行う。実装段階では、システムに入力する。

6.1.2.15　外販部品の使用

有効なライブラリは購入した。購入したライブラリは、内部仕様変更に対応できるように独自のクラスライブラリとして修正した。

6.1.2.16　部品の外販化

A 社ではしていない。

6.1.2.17　プログラム部品の再利用性

同系列のシステム開発を行った場合のプログラム部品の再利用性は、「ネジクギ」の基本部品およびデータ項目部品では再利用率が 50%、GUI 表示具品、GUI 部品制御部品では 75%、MF との通信や障害監視部品では 95%の部品が再利用できている。

6.1.2.18　オブジェクト指向適用による保守性の向上

オブジェクト指向適用前のシステムと比較し、プログラムの修正範囲や修正ステップ数が減少。これに伴い修正時間も減少傾向にある。

6.1.2.19　工期短縮の評価

比較対象が現時点では、メインフレームのシステムとなってしまう。工期という考え方が違ってきてしまう。具体的に数値化しているわけではないが、品質も含めて工期短縮になっていると考えている。

6.1.2.20　オブジェクト指向教育

まず基礎として、C++を勉強し、オブジェクト指向→JAVA を教育。期間は 3 ヶ月～5 ヶ月。教育は、社内のシステム研究開発部門が行う。講師は特定の 1 人。

複数人の講師に教育を依頼してしまうと、プログラムの作法にバリエーションが出てしまうので、1 人に限定している。

6.1.3総評

今回の調査で以下の議論が交わされた。

「UML で実装できるまで詳細に内容を書くとなると、OCM など色々なものを駆使して書かなければならなくなる。仮にそこまで書いた場合と、JAVA のプログラムを直ぐに書く場合の手間を比較すると、やはりプログラムを書いた方が早いという判断を取ってしまうと考える。

UML とプログラムの間に境界線があるかというと疑問が残る。日本語自体が、コンピュータの仕様やユースケースの記述に適合していないのではないかと疑問が残る。

英語では、動詞と名詞、複数の内の全部を選択や複数の内の一つを選択などが普通に書くだけで明確である。日本語ではかなり詳細に文書を書きこまないと、システム開発に使える設計書にはならない。」

当日、結論は出なかったが、システム設計をした経験のある技術者であれば、どこまで記述したら良いかという漠然とした考えをもっているはずである。これを体系化し整理すれば、システムの品質の安定化に繋がると考えられる。このような考察をする良い契機となった。

6.2 B社

6.2.1　ビジネスオブジェクト適用の取り組み　概要

B 社では、情報システムに関する企画～開発・保守に関するサービスを自社、関連会社を中心に提供している。

平成 14 年 10 月現在、自社向けシステム開発は、以下の考え方で推進している。

- 受注、設計、製造、外注、出荷、原価管理に関する業務システムは、品種別に展開された事業部毎にシステムを開発。
- 購買、物流、経理、人事、総務に関する業務システムは、事業部を横断してシステムを開発。

上記の購買及び物流システムには、自社開発したフレームワークと独自の開発プロセスを積極的に活用し、開発の高効率化を達成している。
このフレームワーク適用に至るまでに、1970 年代から以下のような取り組みをしている。

1970 年代 システム開発技法の標準化
1980 年代～1990 年初め パッケージソフト・4GL 言語の活用
1991 年～ プログラムのパターン化
1994 年～ データ中心設計の技法適用
1997 年～ 生産性把握、見積りに FP 法適用

1990 年代後半以降は、オブジェクト指向言語の JAVA に着目し、上記の独自フレームワークを開発さらに、データ中心設計とオブジェクト指向を融合した独自開発プロセスを確立している。

現在も、フレームワークのバージョンアップを行っており、ビジネスロジックの一層の部品化を進めている。

6.2.2　独自フレームワークの設計・製作

6.2.2.1　JAVA 導入時の問題点

- オブジェクト指向プログラミングは難しい技術。
- 標準化や部品作成には時間がかかり、オブジェクト指向で生産性を上げるのは難しい。
- 難しい技術であるため、優秀な SE・プログラマが必要となるが、確保が難しい。
- 自社で優秀な SE・プログラマを育てるにも教育に時間がかかる。

これら問題により JAVA での開発が組織的に立ち上がり難い。

6.2.2.2　対してフレームワークに求めるもの

- JAVA 言語に習熟していなくても開発が可能なこと
- 高度な基幹系業務プログラムを開発できる部品の提供すること
- コードの記述は最小化し高生産性、高品質開発を実現できること
- 低コストで維持できること

6.2.2.3　データ中心設計の適用について

自社内で体系化されているデータ中心設計技法（T 字型 E/R データベース設計技法）があるため、開発プロセスの上流工程へのオブジェクト指向設計の適用は、初めから考慮していなかった。

オブジェクト指向設計にて、カプセル化により変更箇所を 1 箇所に限定するような設計が出来ればよいが、そのような設計は出来ないのではないかと考えている。

6.2.2.4　フレームワークの製作

- 設計・製作は、優秀なキーマンに権限を集中し、精鋭部隊（10 名程度）による製作を進めた。
- 製作期間は、1999 年から最初のバージョン完成まで 1 年半。
- 1994 年に手順と内容が確立されたデータ中心設計技法を用いて分析・設計。
- 実装フェーズはオブジェクト指向。言語は JAVA で開発。
- 製作は、関連会社と共同で行っている。

関連会社任せでは、親会社のシステムを考慮しない設計となってしまう。

親会社に保持して外部に出さな技術も必要。親会社と関連会社との業務分担は明確化している。

6.2.2.5 EJBについて

技術者の評価は低い。しかし、最近は、良くなっていると聞いており、EJB の中で良いものがあればフレームワークに取り込むことを考えている。

6.2.3 独自フレームワークの特徴とこれを利用した開発実態

6.2.3.1 独自開発フレームワークの特徴

- 基本構成

  クラスライブラリ（部品)、各業務共通で必要なアプリケーション（ユーザ管理、権限管理、帳票管理、ワークフロー)、開発ツール、開発標準、設計マニュアル、教育コースで構成。

- クラスライブラリの特徴

  ネジクギ部品及び画面構成部品をパターン化しクラスライブラリ化。

- 業務ロジックの部品化

  販売・生産管理、購買等の業務ロジックを部品化。

画面構成部品や業務ロジックの部品については、異業種、業態によって処理が異なってくるが、協力会社に対してソースを公開することで、パターンの拡張を目指している。

6.2.3.2 ドキュメントについて

現在、CMM(Capability Maturity Model)の体系で作成している。T/ER(T型 E/R 図)があればほとんどの開発はできると考える。UML については使用していないが、ユースケースについては、必要性を感じており使用を促している。ただし、今まで使っていた要求仕様書と何が違うのかという反発がある。

6.2.3.3 独自開発フレームワークの利用者と教育

- JAVA を知らない技術者でも、4 日間の講習でフレームワークを利用したプログラム作成が可能。（以下箇条書き）内訳･･･0.5 日：JAVA

言語の基本講習。3.5 日：部品・フレームワークの使い方。

- フレームワーク利用者の経験は千差万別。かつて COBOL でシステム開発を経験した技術者もいる。

業務ロジックの部品化、画面部品のパターン化、開発プロセスのパターン化が進み技術者のテクニックに依存する部分が少なくなったが、技術者のスキルアップに対する欲求を満たすことが難しくなった。フレームワーク利用者の中には、仕事をやめたいと希望するものも出てきた。

6.2.3.4 フレームワーク適用による開発生産性（1 ファンクション・ポイント当たりの生産性）

自社内 COBOL で開発した場合と比較し 2.6 倍。
自社内 VB で開発した場合と比較し 1.8 倍。

6.2.3.5 カットオーバー後のバグ

プログラムのバグはかなり少ない。

6.2.3.6 カットオーバー後のメンテナンス

標準化によりメンテナンス効率も向上している。

6.2.3.7 社外に販売した時の評判

納入先の生産性が 5 倍程良くなったと聞いている。

6.2.4 取り組みに対する評価

6.2.4.1 システム構築費用の削減

- LINUX 等を利用することで OS 及びハードウェアに関する費用を 1/2～1/3 に削減できた。
- 自社製作のフレームワークを利用することで、ソフトウェア開発効率が 30%以 上向上した。

6.2.4.2 TCO 削減

LINUX、JAVA による WEB システム構築で、クライアント管理コスト及びサーバシステムの安定性が向上した。

6.2.4.3　操作性の統一

WEB システムで統一し、自社フレームワークに搭載されている標準画面パターンを利用することで操作性の統一が図れた。

6.2.4.4　今後の取り組みについて

ビジネスロジックの一層の部品化を進める。

6.2.5　総評

今回ヒヤリングした企業の中で、業務領域まで部品化を進めた唯一の企業である。教育期間の短さやフレームワーク利用者のスキル依存を最小限にしている点など学ぶべきところが多い。

プログラムの部品は、ハードウェアや OS の領域に近づくほど汎用性が高まり、業務領域に近づくほど、汎用性が低下し用途の独自性が強まる傾向にある。プログラムの部品化を推進するほど生産性や拡張性、メンテナンス性の向上が図れるため、現在多くの企業で、ハードウェアや OS の領域に近い、いわゆる「ネジクギ」と呼ばれるプログラムの部品化を推進している。しかし、業務領域のプログラムを部品化する企業は少ない。これは、業務領域の部品化をしても、部品化を進めた企業の業務モデルが反映されてしまい、異業種、異業態への適用が難しくなるためである。B 社では、この点を考慮し購買など異業種・異業態でも同様と考えられる業務モデルに限定し部品化を進め、フレームワークと共に社外販売をしている。それでも、異業種・異業態への適用時には、部品のカスタマイズ化が必要と考えられており、適用方法や業務モデルの更なる部品化への取り組み等今後の動向に注目したい。

6.3 C社

6.3.1 概要

当社は、通信事業向けを中心として各種システムの企画、保守、運用に関する一切を実施している。
事業内容としては、以下のシステムの企画、開発、保守を行っている。

- 電気通信システム
- 通信機器の保守、運用、料金に使用するシステム（OSS）
- バックオフィス系のシステム（人事、経理、会計、総務系など）
- SCM、CRM系の各システム
- KM、EAI、EIPなどの各システム
- 政府系、金融系、流通系、インフラ系などの各業界向けのシステム
- IPネットワーク、VOIP、PBXなどのNWシステム構築
- その他情報システム、ネットワークに関する諸ビジネス

当社社内には1000人を超えるJAVA技術者がシステム開発を行っている。また、オブジェクト指向開発手法には、ラショナルソフトウェア社の開発技法である、RUP(Rational Unified Process)を採用しており、米国ラショナルソフトウェア社が認定するRUPインストラクターの資格を持つ社員数名が中心となって育成指導を行っている。また、JAVA開発支援の専門の組織を設け、JAVAでの開発を強力にサポートしている。

JAVAベースのシステム開発時には、JAVAセンタのメンバが開発内容のアセスメントを行い、コンポーネント利用の教育及びサポートを実施している。

また、開発したシステムはJAVAセンタのメンバが内容を精査したうえで、社内のコンポーネント流通システムに登録し、ビジネスオブジェクトたるJAVAコンポーネントの会社内での普及を図っている。このような営みにより、ビジネスオブジェクトたるJAVAコンポーネントの作成、利用が図られている。

6.3.2 ビジネスオブジェクトへの取り組み

6.3.2.1　電気通信業におけるビジネスオブジェクト

事業内容にもあるように、もともと通信の世界から始まったため、国際団体での活動が盛んであり、ITU(国際電気通信連合)、TMF（Telecommunication Management Forum）などの団体において、ビジネスオブジェクトに取り組んできた。

電気通信の業界では、ヨーロッパが ITU を主体として標準化活動を進めており、北米は業界団体である TMF を主体として標準化活動を進めている。

両団体の進め方の違いは、以下の通りとなっている。

- ITU・・・標準を決めてから製品化を進める
- TMF・・・業界で最も普及しているものを、デファクトスタンダードを標準として採用する

など、欧米の仕事のやり方がそのまま活動に反映している。業界の進歩が早い現在では、ITU で勧告がまとめるのを待っていられないため、各ベンダが独自に実用化したもののうち、デファクトとなったものを追認する TMF でのやりかたが一般的になりつつある。

日本も世界の流れにあわせ、いままで、ITU を中心に活動してきたが、最近は、TMF を重視する傾向での活動となっている。

TMF では、通信事業を営むために必要な機能を「eTOM」として取りまとめ、公開している。

「ETOM」では、各機能が大きな意味でのビジネスオブジェクトとなっており、その組み合わせにより、通信事業を提供していこうというものであり、各ビジネスオブジェクトは、いろいろなベンダが開発し販売しており、ユーザ企業（電気通信事業者）は、それらを組み合わせて OSS として組み上げて導入する。当社としては、当初から TMF 活動に取り組んでおり、技術動向や商品について発表している。

6.3.2.2　C 社におけるビジネスオブジェクト

C 社でのオブジェクト指向への取り組みの歴史は以下の通りである。

1980 年代・・・オブジェクト指向での通信機器の開発
1990 年代・・・オブジェクト指向での各情報システムの開発
2000 年代・・・JAVA におけるコンポーネント流通の促進

当初は、システム間（通信機器～OSS 間）において、オブジェクトの流通が始

まったことから各システムは、C++言語によりオブジェクト指向で開発しながらも、オブジェクトの流用は同一システム内での流通が多く、異なるOS間でのオブジェクトの流通は少なかった。流用する場合も、オブジェクト単位というより、プログラム単位に大きなまとまりでの流用が多く、ソフトウェアの部品化という面では、取り組みが停滞していた。

1990年代後半から、JAVAのコンポーネントということが叫ばれてから、当社においてもビジネスオブジェクトという観点での開発、検討が始まった。

JAVAというOSにディペンドしない言語の登場により、オブジェクトの流通はしやすくなったが、それぞれのシステムが全く独自に設計したままでは、オブジェクトの流通は難しいことから、フレームワークの必要性を痛感し、独自のフレームワークの検討に入り、2003年の「JAVA　ONE」で発表した。現在は、このツールの上で構築することにより、コンポーネントの流通を図っている。

6.3.3当社フレームワークによる、ビジネスオブジェクト化の促進

6.3.3.1　異システム間でのビジネスオブジェクトの流通

たとえ JAVA で作られたコンポーネントであっても、異なるシステム間で JAVA コンポーネントを流用することは容易ではないし、流用できたとしても、必ずしも本来の目的である、開発のスピードアップ、信頼性の向上、開発コストの削減などにはつながらない。

このため、当社では、これらを実現するため、自前のフレームワークを用意し、現在は、このフレームワーク上で構築することにより、コンポーネントをビジネスオブジェクトとして異システム間で流用できるようにした。また、社内にコンポーネント流通のためのしくみを作り、生産性の向上に努めている。

6.3.3.1.1　当社ツールの特徴

当社ツールは、J2EE 準拠の各種アプリケーションサーバ上で動作するフレームワークであり、以下の構成をとした。

6.3.3.1.2　アーキテクチャ

プレゼンテーション層とビジネス層を完全に分離し、柔軟性・拡張性の高いシステムを構築可能としている。

アプリケーションサーバを特定しないので、自由な構成が可能である。

（1）ビューフローツール

プレゼンテーション層の仕様確定を支援し、画像遷移をシミュレートする。

- 仕様確定までの期間を短縮
- WEB デザイナーと開発者の作業分離により並行開発を可能とする

（2）ビジネスシナリオツール

ビジネス層におけるコンポーネントベースの構築を支援する。

- コンポーネントで再利用性を高め、開発期間の短縮
- プレゼンテーション層とビジネス層をノンプログラミングで連結

（3）開発技法

当社ツールに最適化された開発技法により、業務分析から試験までをス

ムーズにサポートする。

- 業務分析から試験までシームレスに開発
- ユースケース駆動開発により、より確実な要求事項のキャッチアップと要求リスクの解消に対応
- 成果物テンプレートの利用により、成果物作成稼動を削減

（4）基本部品、API 群

開発をサポートする機能、保守・運用をサポートする機能、サービス開始後のマーケティングを支援する機能などの部品があらかじめ用意されており、開発を強力サポートする。

過去に作成したシステムにてコンポーネントが図られたソフトウェアを部品として利用することが可能である。

6.3.4　社外へのビジネスオブジェクトの流通

当社では、ビジネスオブジェクトたる JAVA コンポーネントの社外への展開、社外からの調達を実施しており、社内外でのビジネスオブジェクトの普及に努めている。

6.4 D社

6.4.1「MDAによるビジネスモデルベースJ2EE開発」

ビジネスの視点をしっかり持った、ソフトウェア開発のご紹介。

6.4.1.1　プレゼンテーション内容：要約

ソフトウェアのモデルとは、万人が一見するだけで理解出来るように視覚化する事である。また、それがツールによって実際のソースコードと対応がつくこと必要である。保守党で引き継ぐ場合は、モデル化して渡すことが必要である。

また、今後の開発では、モデルを作ることで発注仕様書などをわかりやすくすることが出来る。モデルからは、単にスケルトンなどのソースコードが生成されるだけでなく、「振舞い」自体もモデル化でき、動的な側面の理解を容易にする。モデルとソフトウェア開発の歴史は、先端はDavid A Taylorなどによる、ビジネス系の人とシステム系の人の共通理解を行うために「オブジェクト技術によるBPR」などの研究が行われた。

また、Gang of Fourなどからは、ソフトウェア分析設計のためのノウハウを集大成した「デザインパターン」などが提唱されてきている。最近では、ソフトウェアが稼働するプラットフォームに依存しない形式でのモデルで、なおかつそれらのプラットフォームへマッピングできるモデル作成と言うことで「MDA」(Model Driven Architecture) がOMGを中心に提案されてきている。すなわち、技術的変化に対応できる、プラットフォームに依存しないソフトウェアを開発していく事が目標である。ビジネスモデルには、ビジネス・プロセスモデル（UMLアクティビティ図、EPC)、システム操作モデル（UMLアクティビティ図)、経営概念の静的な特徴を表すもの（UMLで言えば、クラス図）などがある。これらを使うことで、効率的な理解、正確な仕様伝達、高速開発などを実現する事が出来る。

現状の技術では、時間がたった後。「ビジネスオブジェクトモデル」と「ソースコード」の乖離（かいり）が出てしまう。いままでの技術は、ここをフォローし切れていない。

MDAは、ソフトウェアとそのモデルとを高度に同期させる方法論として、OMGから提唱されている新しい技術である。ポイントとしては、「プラットフォーム独立モデル」(PIM) と「プラットフォーム依存型モデル」(PSM) からなる。このPIMからPSMへの変換し、それをソースコードに変換し、PIMから直接ソースコ

ードに変換するなどの「モデル変換」行う。

これらを行うためのツールとして、ARCS TYLER が開発された。これは、MDA 対応の統合開発環境で、プラットフォームに依存しないモデル(PIM)を扱うことが出来る。また、この PIM を、実行するプラットフォームへマッピングするための機構もある。これに、対応プラットフォームへ変換する MDA 変換カートリッジを使うことで、対応するプラットフォームへの変換が出来る。

また、今後、多くのプラットフォームが開発されてくることに対しては、その変換カートリッジを作るための機構（アーキテクチャ IDE というメタプログラミング環境）を提供している。この後､MVC モデルについて実際の開発事例の紹介があった。

その中で、既に出来上がっているものをモデル化する MDA イネーブリングを実践してみたところ、可能だったがかなり時間がかかる事が分かった。大規模システムだと大体 3 ヶ月かかる。問題としては、「ソースコードがない」、「ライブラリがない」などがおきる為時間がかかる。大規模なシステムのリバース作業には、これくらいの時間がかかる。

6.4.1.2 質疑応答

A 氏：自然言語に近いものが普及しない理由は何か。問題は何か。

D 社：難しい問題。このツールをドイツの会社と契約したばかり。

A 氏：ブロードバンド、インターネット、大量容量がそろったから出来る時代でないかと思うが、どうか。

D 社：ビジネスオブジェクトは自然な要求の書き方モデル。実際にそのモデルからソフトウェアから開発できてしまうツールなので、利用する価値があるのではないかと思っているが。

B 氏：ビジネス・プロセスモデルを書けば、ソースまで落とせるという事だが、ビジネスモデルをどうかくかが大変なのではないかと思うが。その、前の関係をどうしたいのかが今後の課題かと思うが。

D 社：このツールは、大変現実的。普及しているツールを使いながら実装につなげるツール。

C 氏：現実的に使いこなすには、どうなのかと思ったが。ボタン一つとおっしゃったが…。

D 社：MVC モデルと、PIM モデルを書く事が、時間がかかるか。それ以降は、本当にボタン一つで行える。そこから先の技術としては、EJB の知識がある程度必要になる。しかし、ツールなしの EJB に比べたら、まったく難しさが違う。

C 氏：弊社も使える人材、部隊が本当に一部。
D 社：EJB を使える部隊が、社内全体の専門家集団となっていただいて、広める事が必要なのではないか。それ以外の部隊は、EJB をある程度分かればいいという状況でツールを利用すればいいか。
B 氏：EPC は？
D 社：EPC は、持っていない。意味的に深く拡張している部分はない。こういったツールが何故普及しないかという問題ですが。モデリングの難しさの件ですが、分析者と開発者のレベルがまったく違うから問題なのではないか。分析者は、何もないところからモデリングする人達。開発者（技術者）は、モデリングされたものをソース化する人達。実際は、開発者がモデリングされていない、はっきりしない情報からソース化する状況。PIM を PSM にする人達が技術者の仕事であるべき。そのためには、そのモデルをたくさん見せてあげる事が大事。
B 氏：再利用を考えると、概念もハンドリングする技術が、分析者にも必要なのではないか。もともとある世界を、抽象化する技術も必要なのではないかと思う。ハンドリングする機能が欲しい。
D 社：そうですね。PIM の世界でモデルをたくさん持つ事で、PSM の世界で持つよりかなり良いのではないかと思っている。
A 氏：分析は人が？
D 社：そうです。

6.4.1.3 「SYNERGY METHOD」…再利用の仕組み。

- 技術依存性軸…MDA。
- 多様性軸…オブジェクト指向のスーパークラス、サブクラスの考え方。フレームワーク。
- 進化軸…一度作ってしまったシステムをどう発展させていくか。

部品も進化が必要。オブジェクト指向のフレームワークを使った進化の仕方などを標記。
このようなシステムの作り方を行いたいと思っている。

6.4.1.4 実績

「SYNERGY METHOD」の考え方に基づき、開発。30 社くらい実践済み。
プラットフォームを簡単に変えられることが可能。

A 氏：開発の生産性や納期は短くなっているなどのデータはないか？

D 社：現在、整理中だが、生産性は大変高い。使いやすさのための試行錯誤は、かなり時間がかかる。しかし、そこにターゲットをあてた事も売り。

E 氏：RUP は、こういうものの開発に向いているか。

D 社：向いている。リリースしてユーザを聞いての繰り返しだった。サイクルも長かった。(6 ヶ月サイクル) ウォーターフォールでは出来なかった。

E 氏：技術者は、モデルから考えるという流れを作らないと駄目なのではないかと思った。

D 社：その通り。PIM から PSM へのケースをたくさん見せてあげる事が大事。そうする事でいきなりソースを組んでいた事に間違えを感じる。

B 氏：実装とビジネス・プロセスの対応を考えて、モデル化する必要がある。

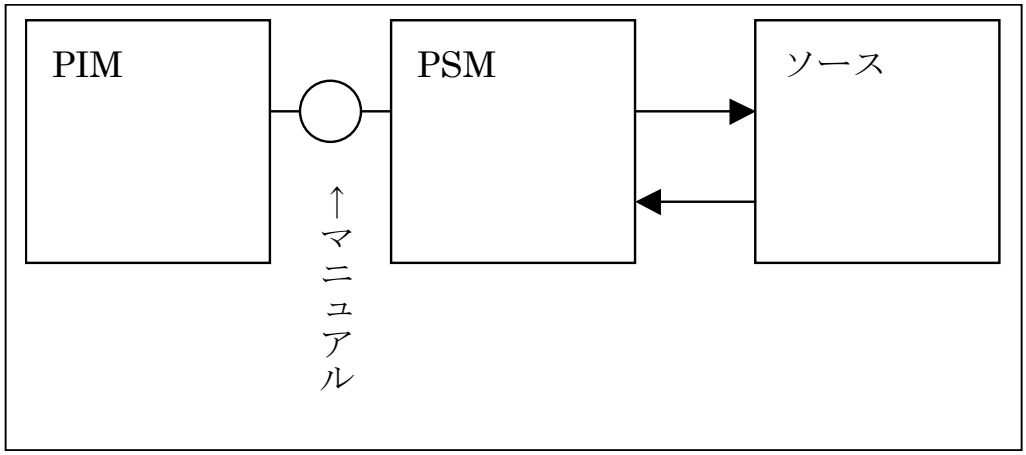
6.4.1.5 MDA モデル変換の分類

経営とずれていない IT を作っていく技術が MDA。経営の視点と技術の視点の間にあるのが MDA。モデル化にもレベルがあると思う。

- レベル 1：アットワークの例。
- レベル 2：コンピュータが自動的にモデル化を作成できるようにする。
- レベル 3：小さい PIM を使って、大きな PIM を作成できるようにする。コンポーネント開発と一致している。
- シナジーメソッド：
- フレームワーク：

F 氏：ツールベースのお話ですが、ツールを使わないで J2EE を開発する場合はどういった方法があるか。

D 社：PIM から PSM に書く際に、マニュアル作業になる。



6.5 E社

6.5.1　概要

E社では日本の中にN事業部・M工場・L研究所で、多数の製品毎のシステムを開発してきたが、以下のシステム開発的特徴を持っている。

- 部品を組み合わせての製品が多様である。→何とかソフトウェア部品を組み合わせて簡単にシステムを開発することができないか？
- 製品寿命が短い。かつて10年あった製品寿命が最近は半分になってきた。→短工期開発の極限は化が狙えないか？

従来は別々に30システムを開発して来たが、上記課題に応えるために、オブジェクト指向設計技術を追求している。
オブジェクト指向設計は「将来有効なもの」と認識しいくつかの試行を重ねてきた。その結果をお聞きした。

6.5.2　プロジェクト1：生産管理システム

期間　分析2001,10月12日～12月11日、実装01年12月3日～01月31日
作業メンバー　E社3名　X社3名

6.5.2.1　分析フェーズの作業と評価

- 製造計画作成から製造実績把握までの業務範囲を対象とし、分析フェーズでは概念モデル、アクティビティ図、コンテキスト図、ユースケース図、ステートチャート図、非機能要件定義書を作成した。この分析フェーズにおいて概念モデリングを3回に分けて行った。
- クラス図の作成
  - ✧ 1回目…ブレーンストーミング形式で、メンバー全員でクラス候補を洗い出した後画面や帳票を参照しながらクラス図を作成した。
  - ✧ 2回目….一つの製品について、再利用性を考慮しながら抽象度を上げた。
  - ✧ 3回目…他製品について簡単な分析を行った後、それとマージし抽象度を調整した。

- 分析フェーズの評価
  - ✧ メリット　表記法が統一されているので個人個人が作成した成果物が理解しやすかった。従来の手続き処理とオブジェクトとの相違を再確認した。ドメインレベルでのフレームワーク化のめどが立った。
  - ✧ デメリット　全て UML 化することは非効率になる恐れがある。オブジェクトの切り出しについてはスキルと経験を要する。

6.5.2.2　実装フェーズの作業

- プレゼンテーション層、ビジネスロジック層、DAO 層に分けて整理を進めた。
- シーケンス図作成のために CRC を採用した。
- クラス図は概念モデル→類型クラス図→実装モデル図へと展開した。
- RDBMS を使用し E/R 図を作成した。JAVADOC によるメソッド仕様書、テスト仕様書を利用した。DB アクセスには DAO を使用した。
- マネジャークラスのフレームリレー化は伝票番号採取、赤黒処理、DB 書き込みなど基本的なロジックをマネジャークラスにおいて実装した。差分はパラメータークラスなどのヘルパークラスを用いて実現。マネジャークラスは差分に影響しない構造にする。

6.5.2.3　実装フェーズの評価

- メリット
  - ✧ モデリング技術の有用性に対する認識が定着してきた。
  - ✧ 反復開発手法の採用によりスケジュール通りプロジェクトが遂行できた。
- デメリット
  - ✧ ペアプログラムを常時実施したために効率低下が生じた。適用方法を工夫すべき。

6.5.3　プロジェクト 2：経理の原価管理システム

（説明者 IT 推進室　Y 氏　）

6.5.3.1　検討時期

事前整理 2002/6～…2 人×3 月
モデリング 11～12 末…3 人×2 月

6.5.3.2　対象システム

標準原価計算、月次総合原価計算、人件費間接費予実計算、収支予実計算の四つの主要コンポーネントからなる、ものづくりに関わるコスト管理と決算の一部を目的とした経理システム。

6.5.3.3　モデリング手法

E 社の親会社のコンサルを受け「カタリシス」を採用した。UML に準拠しているが一部オリジナル図表もある。オブジェクト指向を用いた。

6.5.3.4　分析、基本設計フェーズで作成したドキュメント

- マインドマップ…システムの構築範囲と大きな枠組み。
- アクティビティ図…システム内の概略ロジックを表した.業務フロー。ユーザには従来方式の別の業務フロー図を作成して説明した。
- ユースケース図…機能図（ユーザとシステムの役割の関係表現…DB も一つのアクタ）原価の取り方を工場間で統一したために時間がかかった。数は 1 つか 2 つ。アクティビティ図とユースケース図は対応しているように作った。やはり、事業所によっては使ったり使わなかったりしている状況はある。しかし、考え方をあわせるという視点では良かったのではないかと思う。
- タイプモデル…コンポーネント内のクラスの関係を表現。クラス図を一つのコンポーネントに絞って作っているもの。→関係というのはどういった関係かというと、1 対 1 の関係を持つ。

6.5.3.5　詳細設計フェーズで作成したドキュメント

- （論理）クラス図…オブジェクトの関係を表現。KEY+プロパティに整理した。一番多くなクラス図は、「製造原価」というクラス図。クラス図とクラス図の関係は、(1 対 1 の関係かどうか）などは UML の標記の仕方で整理した。
- フロー図…内部処理の流れや分岐条件を表したもの（シナリオの補助、フローのような表現も活用した）シナリオは、文章で書かれているので分かりにくくなるので、その際の補助解説資料として作成した。従来のフローチャートに近いもの。
- シナリオ…ユースケース等を使用して、内部処理の内容をエクセルにて文章で記述したもの（実装者への作業指示用）本シナリオ 100 個+代替シナリオ（例えば実行時の初期値など）→シナリオはユーザさんの為ではなく、実装する際の資料として作成した事が目的。プログラム

を文章にしたような形のもの。

- 画面遷移図…別画面の呼び出しの関係を図示したもの
- 画面定義…EXCEL にて、データをつけた画面を設計した。内部処理などは触れていない。

全体的な流れとしては、

- マインドマップ：ビジネスドメイン・コンポーネントごとに整理。
- アクティビティ図・ユースケース図：どういう種類が。
- クラス図・フロー図で整理するが、工程を戻ったりする。

実際に作っている時に、実装の部分まで検討してしまったりしてしまう場合も多々ある。オブジェクト指向、UML は、長くやっていれば出来るようになるものではない様な気がする。今のレベルは、人のレベル・センスによって左右されるというのは問題であると認識している。

6.5.4 総評

企業では、事業部別に製品を開発、製造し、販売する組織構造を一般に持っている。

製品を開発、販売する時期が別々であるために、あるいはその製品を扱う責任者が各事業部長で異なっているために、あるいはシステム開発部隊の準備期間がかかり過ぎるために、小さい別々のシステムを開発し間に合わせてきたのが一般的である。

基本構造を定め、コードなどの統一を図り、企業として「標準化した基本システム」を作成し、後々発生してくる新商品のシステムは、この基本システムを活用して開発したい要望は相当幅広く存在するが、実例としてこのような仕組みで上手くシステムを標準化し、ユーザの要望に応えている例はほとんど見かけない。

この事例がその代表になって「このオブジェクト指向設計開発を利用して、システムが早く、安く、でき良くなった」と成功評価を受けることを期待したい。

このような仕組みへのトライをするためには、「技術に理解がありかつ業務の標準化の推進できる指導力のある CIO が必要」を痛感した企業訪問であった。

6.6 F社

6.6.1　ビジネスオブジェクト適用の取り組み　概要

F社G氏は、様々な企業に対して、オブジェクト指向に関する技術コンサル（上流フェーズにおけるモデリング、UML、開発プロセス、フレームワーク構築）を経験している。コンポーネントの流通や技術共有、標準化にも力をいれており、その取り組みや動向についてお聞きした。

6.6.2　コンポーネントとフレームワークについて

6.6.2.1　コンポーネントの分類

再利用、置き換えが可能なように製作したモジュール群

実装技術（EJB、COM、.NET、CORBA など）による分類と役割（①受発注、在庫、販売などの業務依存型、②カード決済、外部接続、汎用検索などの業務共有型、③日付、入力チェック、DB 接続などのユーティリティ型）による分類ができる。

6.6.2.2　アーキテクチャについて

アーキテクチャとは、アプリケーションの構造やインタフェース、通信メカニズムなどを規定し、アプリケーションを構成するコンポーネントとなるもの。効果的にコンポーネント開発を進めるために必要。

以下の3つに分類している。

- エンタープライズアーキテクチャ･･･ハードウェア、ミドルウェア、ネットワーク構成などのインフラのことを示す。企業システム全体のアーキテクチャコンセプト。
- ソフトウェアキテクチャ･･･ソフトウェアの基本構造。最近ここを補足するフレームワーク製品などが多くなってきている。コンポーネント化に最も適した領域。
- ビジネスアーキテクチャ･･･業務領域に登場する概念とプロセスの構造。コンポーネント化が難しい領域。

6.6.2.3　フレームワークについて

アーキテクチャを維持するための標準を強制するもの。開発手順を画一化して品質のばらつきを軽減する。

6.6.2.4　フレームワークとコンポーネント分類について

現在、フレームワークは様々な団体により提供されている。各団体によりコンポーネントの分類方法や呼び名はまちまちであるが、先に述べた業務依存型、業務共有型、ユーティリティ型の 3 種類にほぼ分類することができる。

6.6.2.5　フレームワーク製品について

現在の代表的なフレーム製品を以下に挙げる。

- オープンソースフレームワーク･･･Struts、Turbine、Velocity
- 商用フレームワーク･･･cframework(EC-ONE)、オブジェクトワークス（野村総研）
- SI やコンサルタントとセットで提供･･･JAG(オージス総研)、TOCCATA（日本総合研究所）San-Franciscoframework は失敗例

6.6.2.6　フレームワークの限界について

現在の多くのフレームワークは、業務共有機能やユーティリティ機能に対して分割化と部品化を推進するアーキテクチャを提供している。しかし、業務機能の分割に関しては、各業務アプリケーション設計者が分割しなければならない状況である。どのような基準で分割するかについて、何らかの手順やガイドラインが必要である。

6.6.3　コンポーネント開発の実情

6.6.3.1　ユーザ企業の望むこと

コンポーネント適用により生産性があがっているのかいないのか、その結果及び生産性の指標がほしい。

様々なベンダに対しシステムの企画・設計・開発を依頼した際、成果物としてのドキュメントも様々な形式で提出されるため統一してほしい。

6.6.3.2　フレームワーク、コンポーネント適用時の生産性指標

外部に出せる明確な指標はない。

オブジェクト指向の経験のない技術者がフレームワークを使用して開発して生産性が上がったという事例はある。

ただし、コンポーネントを適用の初期段階は、必要なコンポーネントを作ることから始めるため生産性が上がらない。いくつかのプロジェクトを経てコンポーネントの数がそろった時に生産性向上のメリットを享受できる。

6.6.3.3　コンポーネント化の向き不向き

ビジネス・プロセスは、開発毎にロジックが変わってしまうため、使い捨てのプログラムでよいと考える。エンティティレベルのプロセスは、コンポーネント化すべきであると考える。

6.6.3.4　E/R エンティティとオブジェクト指向エンティティの違い

ER のエンティティは、正規化という考え方。

オブジェクト指向のエンティティは、実態をそのままという考え方。

6.6.3.5　再利用性の検討フェーズ

ユースケースモデリングの際に検討すべきか？　検討フェーズを明確化したガイドラインが必要と考える。

6.6.3.6　オブジェクト指向は難しい

オブジェクト指向の教育をしてきちんと理解する人が教育を受けた人の 2 割程度という情報がある。オブジェクト指向は難しいということを受け止める。その上で、全員が理解せずともシステム開発が出来るように、フレームワークとしてのアプリケーション基盤を専任の技術者（オブジェクト指向を理解した）が開発してしまうのが良い。これは、昔から取り組まれている方法であり明示的にしたものが統一プロセスである。

6.6.4　開発プロセスについて

6.6.4.1　開発プロセスとは･･･

開発の工程、手順、成果物の規定とガイドライン。

- UML･･･ISO の標準になる。今後利用が拡大される(*1)と考えられる。あくまで表記法と意味を定義したものであり、方法論ではない。
- 統一プロセス･･･開発プロセスの雛型である。

考え方を具体化することが必要。

(*1) UML の動向

UML に関する情報は大幅に増加している。最近は、UML に興味を示す経営層も増えてきている。また、ベンダ企業の中には、UML 認定試験の

合格を昇格条件の一つにするなど普及の傾向にある。

6.6.4.2　普通のオブジェクト指向開発とコンポーネント開発の違いと注意点

- コンポーネント開発とはオブジェクト指向開発の前の段階。
- コンポーネント識別のタイミングが重要。
- 再利用タスク・アクティビティの明確化が必要。
- 役割分担･･･レイヤ毎の役割分担が一番良いと考える。

6.6.4.3　開発ガイドラインの標準化

開発ガイドラインも体系化できる。

例えば、開発チュートリアル、フレームワーク利用ガイド、共通規約など。

6.6.4.4　ドキュメントの標準化

ユースケース図など､ユーザ企業毎に各社各様で作っており独自性が強い。上流過程のドキュメントについては､ユーザ企業にて形式が規定されている場合もあり､ベンダ企業が標準化するべきではない。下流過程のドキュメントは､ユーザ企業には必要のないものであり､ベンダ側で自由に製作が許されるため標準化するべきである。

6.6.5　コンポーネント開発と公開についての取り組み事例

6.6.5.1　X社コンポーネントセンター

- 作成したコンポーネントを会員企業に対して提供するサービス
- 支援サービス
- 見積、受注、出荷などをコンポーネント化
- ユーティリティ系のコンポーネントも提供

6.6.5.2　CBANK

- 会員制度でコンポーネントを公開。しかし、外向けのコンポーネントはあまりない。EC1 と友好関係。
- ビジネスとしてあまり力を入れていない。

6.6.5.3　コンポーネントスクエア

自分たちで製作するのではなく、会員企業からコンポーネントを登録して

もらい、コンポーネントを必要とする企業に販売するという方法を取っている。

6.6.6　コンポーネント流通の課題

- コンポーネントの流通は厳しい状況にある。
- 知的所有権などは曖昧
- 大きいコンポーネント（企業特有のノウハウが入ったもの）はノウハウの流出が懸念されるため流通しない可能性が高い。
- オープンソースかブラックボックスかで揺れている。

6.6.7　今後期待するアーキテクチャ

MDA（Model driven Architecture）･･･モデル駆動型アーキテクチャ。プラットフォームに依存したモデルの記述方式を規定して、プラットフォームに依存しないモデルから連携させてシステムを構築していく。

モデルレベルでポータビリティの高いものを開発していくという概念のアーキテクチャ。

6.6.8　総評

業務共有型コンポーネントやユーティリティ型コンポーネントの開発により高い生産性を目指している企業は多い。

F社 G氏には、こうして開発されたコンポーネントを流通するというポイントについてもご講演頂いた。これが実現できると、自前でコンポーネント開発の出来ない中小のソフトハウスも、高い生産性を維持しつつ、高品質なソフトウェアを供給できそうである。しかし、知的所有権や、「企業独自のノウハウが入ったコンポーネントは流通したくない」などの問題があり、現状はまだ課題が多い。

一方、業務依存型コンポーネントについても課題がある。「どのような基準でモジュールを分割し作成したらよいのか」というガイドラインの規定や、「ノウハウの詰まったコンポーネントをどのように再利用するのか」という点である。

この講演の聴講により、これら課題の再認識と共に、「流通コンポーネントを利用するときに、何を基準に利用するようになるか？」という点を考えるきっかけとなった。

7.　今後の動向予測(事例を振り返って)

7.1 オブジェクト指向の難しさについて

7.1.1今後の動向を踏まえたオブジェクト指向技術への取り組みの必要性

今回の調査は、必ずしも広範囲のものではなかったが、調査を通じて、我が国では米国等に比べて、企業が経営に活用する情報技術に関してオブジェクト指向技術への関心及び取り組みが遅れている印象を強く受けた。現在のままでは、米国ばかりでなく、インド、中国に比べても格差が大きくなる危険性をもっている。

その恐ろしさは何処にあるかと言えば、従来の情報技術では、もはや、激しい経営環境変化に、時間の面でも、コストの面でも対応できなくなるからである。企業が経営に活用する情報技術に対する考え方は、今後、所有から利用へと劇的に変化して行く[2]。新しい潮流は、ハードウェアやソフトウェアを自ら所有し、保守する従来型の情報システム構築を戦略的に厳選し、インターネットを通して活用できるサービス（ソウトウェアの時間貸し）の活用で済ませれれるものは済ませてしまうと言う方法である。その背景になる技術が、7.3.3 で詳しく述べる「Web サービス」である。

「Web サービス」は、人間がオペレータとして直接介在することなく、インターネット上で提供されるサービスを利用できる仕組みである。この仕組みを利用することによって、必ずしも自社で開発するメリットがないソフトウェアについてその時間貸しをインターネット上で提供しているところがあればそれを利用できるようになる。また、「Web サービス」では、企業競争力を高める必要上、どうしても自社で開発しなけければならないソフトウェアと時間貸しのソフトウェアとをインターネットを通じて、リアルタイムで自動的に連動できる。このようなことが出来るためには、「Web サービスアーキテクチャー」に従った形でソフトウェアが作られている必要がある。その基本になるのが、「オブジェクト指向技術である。

確かに「Web サービス」は、技術面、制度的なインフラ整備面及び「Web サービス」プロバイダー市場が未成熟等多くの課題があり、現状では試行段階である。しかし、米国においては、官民を問わず、業務における IT 活用において、将来「Web サービス」が、相当活用されるようになると予測して、備えに入っている。この必ずしも表面に顕在化していない準備に、官民で、我が国は、こ

[2] John Hagel Ⅲ & Jhon Steely Brown：ウェブ・サービス・アーキテクチャーの可能性「所有から利用」への IT マネジメント　ダイアモンド・ハーバード　ビジネス　レヴュー　２００１年１２月号　pp.115-127

の潮流の恐ろしさに気が付いていないように思われる。

「Web サービス」が将来の動向であると見なされるもう一つの理由は、基盤環境（現在勢力を二分しているマイクロソフトの COM/.NET 系と、サンを源流とするJavabeans/EJB系)に関係なくインターネット上でソフトウェア部品(コンポーネント）を再利用できる技術体系にある。コンポーネントを再利用することに対する期待は、今まで裏切りの連続であったが、今度こそはという思いがあることと、再利用の諸基盤技術、開発に時間が掛けられないという厳しい経済環境からくる深刻なニーズの高まりから、コンポーネント再利用の実用化は進むと考えれれる。「Web サービス」では、インターネットを介して、このようなコンポーネントを、オンラインで組み合わせて、実業務に使用する情報システムを構成するような事が可能になる。このような画期的な方法によらなければ、激しい経営変化に対応することは出来なくなる。先に述べた通り、先見性のある組織は「Web サービス」化を想定して準備を進めている。試行システムの報告を見る限り、オブジェクト指向技術を用いて、システム構築を行なってきた所は、「Web サービス」化がきわめて短時間にできると報告されている。将来に備えてオブジェクト指向技術を活用してシステムを構築することは、きわめて重要なことである。

7.1.2 オブジェクト指向導入の最大の難しさ

調査を通じて、どの企業もオブジェクト指向導入の最大の問題は、何を「オブジェクト（クラス)」に取り上げるかの難しさである。さらに掘り下げてみると難しさの本質は、本来データ変換が基本である業務処理情報システムを「オブジェクト（クラス)」を基盤にしてモデル化することであると考えれれる。実用になる業務処理情報システム全体が、抽出・特定された「オブジェクト（クラス)」で過不足なく機能させられるか読みきり、システムの完成により実証できるオブジェクト指向技術者（特に、実装設計者）の育成が困難な点が更にオブジェクト指向の実用化を阻んでいる。

オブジェクト指向の実用化を成功させている企業では、中核となる実務的な実装設計技術者がいて、リーダーシップを発揮していることを、改めて痛感した。一方で、残念ながらこのようなリーダとなるオブジェクト指向技術者の育成が非常に難しい現実がある。教育の機会が豊富にあり、教材、教科書も豊富な現在においても難しさは、数年来あまり変化がないようである。

オブジェクト指向の難しさは、何処にあるのか。当該当プロジェクトでは、問題は 3 点あるのではないかと考える。

困難 1：オブジェクト指向は、物事の理解の仕方を変えなければならない。
困難 2：適切なオブジェクト指向の指導が行なわれにくい。
困難 3:オブジェクト指向を用いて作成されるモデルは人によって一様にはならない。以下、これらを説明する。

7.1.3　モデル化の理解のステップ

まず「オブジェクト指向は、物事の理解の仕方を変えなければならない」という問題について検討する。
情報システムを構築するためには、実世界のモデルをコンピュータ内部に作ることが必要である。別の言葉で言えば、実世界を情報システム構築の観点から理解する事である。
人は、行動（活動、アクション）を中心に物事を理解することは、持って生まれた能力と言って良いほど自然に行なうことができる。小さな子供に、話をさせたり、作文を書かせたりすると、まず何々をして、次に何々をしたというように、行動の事実を羅列する。このことからも、行動（活動、アクション）を中心に物事を理解することは、かなり自然にできる。
情報システムの機能を考えるモデル化の際に、まず何々を情報システムにさせ、次に何々をさせるという思考は容易である。情報システム（データ処理システムのほうが正確な呼称ではあるが）の揺籃期には、このような書き下し文のようにプログラムが作られる事が主流であった。書き下し文的なプログラムを作成するツールとしては、フローチャートが有効である。
次に、情報（データ）の入力から、求める情報（データ）の出力までの過程を、情報（データ）の流れと、情報（データ）変換させる機能との連鎖（図 7－1）をベースに考える考え方が出てきた。いわゆるデータフロー(DFD：Data Flow Diagram)によって、物事を理解する考え方である。
ところで、現在では、信じられない事であるかもしれないが、DFD が世に出た頃は、それまでフローチャートを用いてプログラムを作成したプログラマは、なかなか DFD を書くことができなかったことを、我々は経験している。現在でも、まともな DFD を書くことができない人がいるくらいに、行動（活動、アクション）の手続きを中心に物事を理解することから、データの変換の連続によって、原データから、求めるデータや情報を得るという機能中心に物事を考えることには、発想を変えなければならなかったのである。しかも、当時のすぐれたプログラマは、手続き中心の考え方でプログラムを作ることに長じていた

ので、その考え方を変えることは至難に近いことだった。

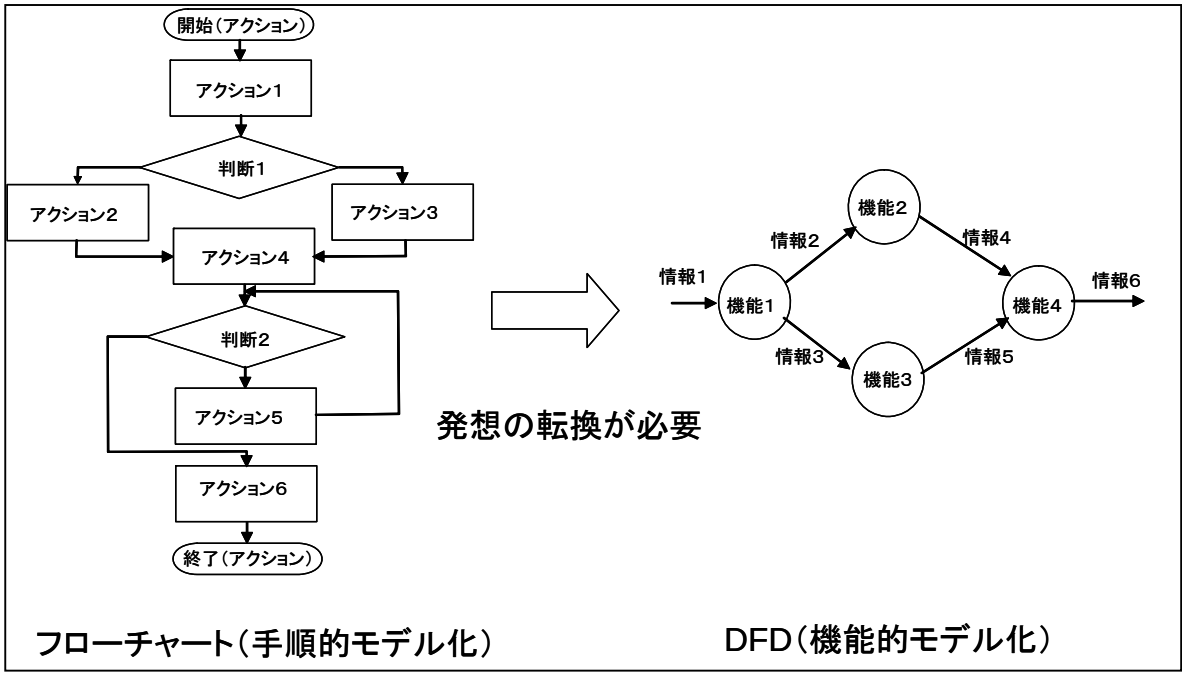


図 7-1 モデル化の発想転換の第 1 ステップ

発達心理学[3]の教えるところによれば、事物を理解するためには、頭の中に、思考の枠組み（シェマ）ができ、知覚した事物を、この枠組みに照らして、必要なら枠組みを調整（同化）して、はじめて理解が可能であるとのことである。発想の転換には、理解のための枠組みがまず頭の中にできなければならない。上記の場合には、実世界をデータフローで捉えるという枠組みが頭の中にできなければ、いくら教育しても、実世界をデータフローで捉えることはできず、したがって頭の中に、活動の手順で物事を捉える枠組み（フローチャート的枠組み）しか持たない者は、データフローを描けないことになる。

情報システム化の目的で実世界をモデル化する次ぎの段階として、データフローにおける情報と機能とのうち、データフローの機能側を重視した視点ではなく、情報に視点を置き、情報に対する操作の纏まりを考え、情報に対する操作要求と操作結果の受け渡しで、情報システムの機能を考えるモデル化を行う考えがでてきた。これが、データ中心思考である。この考え方がでてきた背景には、実世界の変化に対して、比較的変化の少ない安定したものに立脚して、情報システムを構築すれば、情報システムが、変化に適応しやすくなるという思想である。一般的には、情報（正確には情報の構造）が変化しなくとも、機

[3]ジャン・ピアジェ：発生的認識論（白水社　文庫クセジュ 519)、ジャン・ピアジェ&ベルベル・イネルデ：新しい児童心理学（白水社　文庫クセジュ 461）

能は変化する事ができるが、情報（正確には情報の構造）が変化すれば、その情報に対する操作（機能）は変化せざるを得ないという関係がある。実世界の変化に対して、情報構造のほうが、機能より安定している。情報システムを構築するにあたって、まず、データベースを設計し、データベースへのアクセスを考える方法が、この考え方である。データ中心の考え方を発展させると、情報に対する操作の纏まりを考え、情報構造に対する操作要求と操作結果の受け渡しで、情報システムの機能を考えるモデル化を行なうという考え方に至る。これがオブジェクト指向の考え方である。

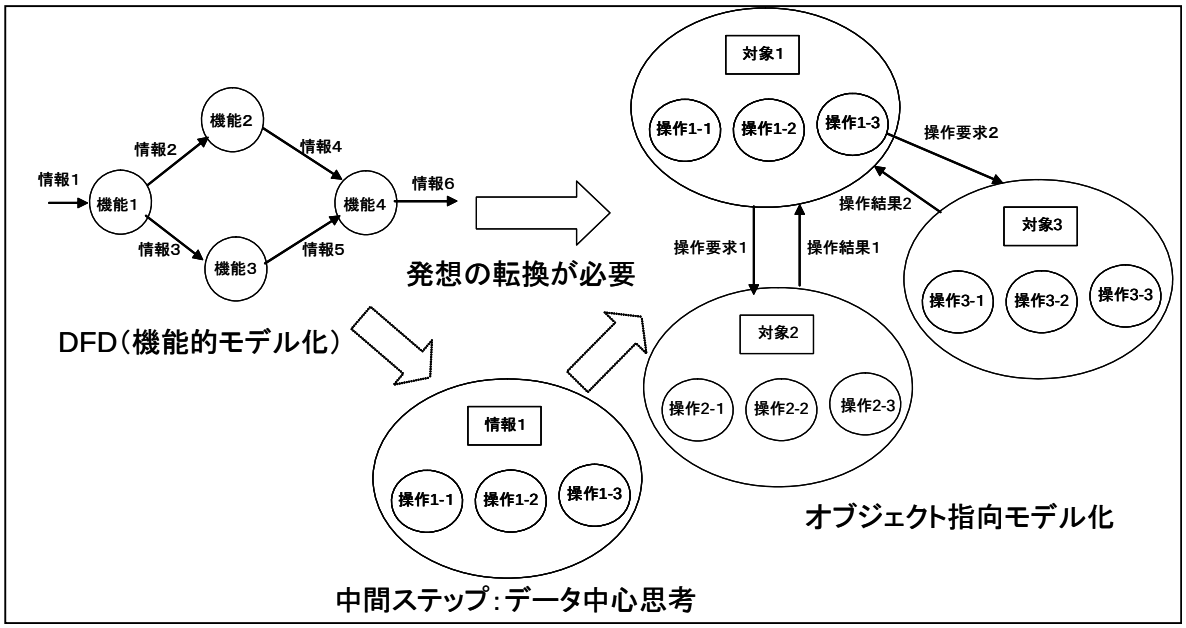


図 7-2　モデル化の発想転換の第 2 ステップ

オブジェクト指向の考え方は、更に、情報に対する操作をサービスと考える考え方に発展した。情報を操作の対象と考え、情報に対する操作のサービス要求とその結果の取得をメッセージの交換として情報処理を行う考えである。以上をまとめて見ると、モデル化には、次ぎのステップがある。

- ステップ 0：活動（アクション）経験を列挙する。（もともと人間が出来る能力）
- ステップ 1：目的達成のための活動（アクション）を列挙する（手順的発想）。
- ステップ 2：目的達成に至る過程（プロセス）を情報の変換の系列としてとらえる（機能的発想）。

- ステップ 3：操作と操作の対象を考え、操作の対象の視点から操作をまとめた纏まりを作り、その纏まりに対して操作の要求を出す事を繰り返して目的を達成する（オブジェクト指向発想）。

当然、ステップ 0 から、ステップ 1 は、容易に移行出来る。ステップ 1 から、ステップ 2 への転換は、出来ない人が出てくる。ステップ 2 から、ステップ 3 への転換は、相当難しい。

ステップ 1 から、ステップ 3 への直接の移行は、ほとんど不可能である。これは、人間が発想する立脚点を

活動（アクション）→機能→機能の対象（情報等）

に移すという発想の転換が難しいためである。

再び、発達心理学を引用すれば、頭の中にオブジェクト指向の発想の枠組みができていないかぎり、オブジェクト指向によるモデル化は不可能である。

オブジェクト指向の難しさは、この発想の転換ができる、頭の中の、枠組み形成の難しさによっている（図 7-3）。

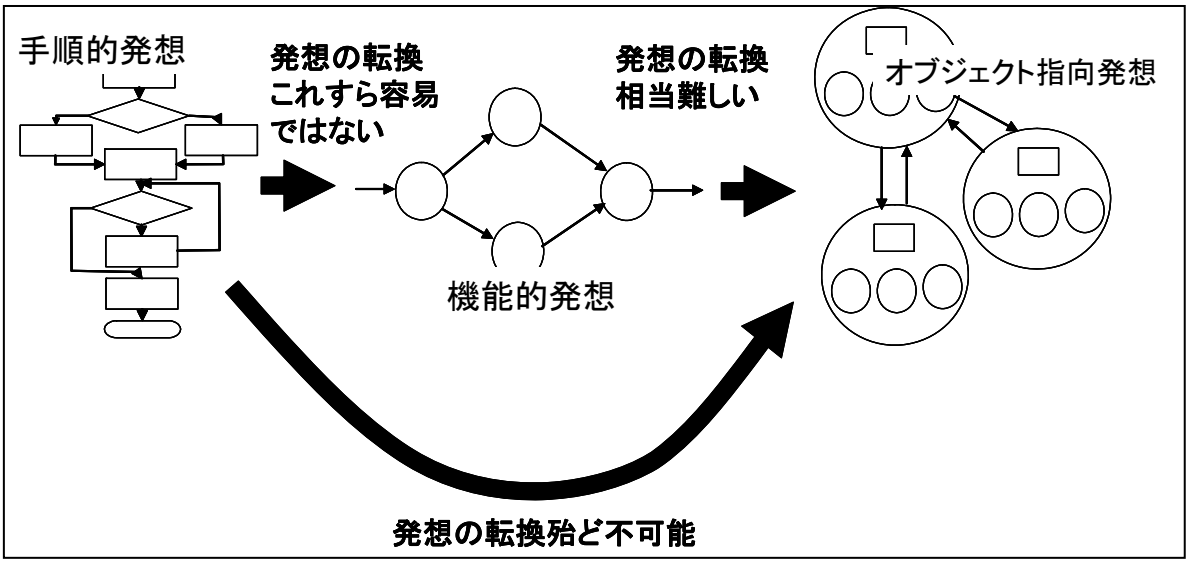


図 7-3　モデル化の発想転換の難しさ

以上に述べた困難さには、従来技術に精通したもの、従来技術の熟練者ほど、新しい考え方に転換できないという極めて厄介な問題が絡んでいる。情報システムの実装設計や実装には、過去の情報システム構築経験が重要な要素を占めている。過去に豊富な情報システム構築経験を持つものが、新しい考え方をマスターして、仲間や後進を指導する必要がある。ここの所が、オブジェクト指向の普及の隘路になっている。

ところで、以上に述べた思考の発達過程の難しさは、処理が手順的に行なわれる情報システムという領域での問題であり、一般的な事物の理解の仕方としては、実在する事物に即して物事を理解するということ自体は、むしろ容易な面もある。従って、情報システムの新人は、比較的人間が発想する立脚点を機能の対象（情報等）に移すという発想を理解することが比較的容易である。これは、福音である。良い指導者により良い教育方法をとる事によって、成功体験をつめば、急速に新技術へ転換できる道があることを示唆している。

7.1.4　オブジェクト指向の教育法

オブジェクト指向技術者の育成を困難にしている問題に、オブジェクト指向技術を理解させるための適切な指導方法がないという点があげられる。

その問題は、概略次のようにまとめられる。

- そもそも頭の中にオブジェクト指向的理解の枠組みの出来ていない者の頭の中に、オブジェクト指向的理解の枠組みを形成させる教育は難しいことが認識されていない。
- 比較的容易な手順的や、機能的理解から入って、オブジェクト指向的理解へ至る手順を示す良い教育方法を示す教科書がない。
- 発祥の事情から、人工知能や知識工学の耳慣れない用語が必要以上に使われているが、教育の最初からこのような用語がどんどん使われる。

7.1.4.1　オブジェクト指向的理解の枠組みを作ることの難しさ

何事によらず、頭の中に理解の枠組みを形成させる教育は難しい。それは、自ら、強い関心（動機付け、問題意識等）があり、反復、再認を繰り返してはじめて理解のための枠組みが出来、理解への備えができるためである。良く「オブジェクト指向ができるようになるのは難しい。しかしもっと問題なのは、オブジェクト指向ができるようになると何故、他人が、オブジェクト指向ができないのか分からなくなってしまう」と言われている。これは、発達心理学の考え方の傍証になっているように思われる。頭の中にオブジェクト指向的理解の枠組みが出来、オブジェクト指向的思考をマスターしたものが、手順的や、機能的理解の枠組みしか持たないないものに対して、いかに頭の中にオブジェクト指向的理解の枠組みを形成させるかという適切な教育を施せないためである。

そもそも、思考に飛躍的な発展段階（7.1.3 項で述べた、手順→活動（アクション）→機能→機能の対象（情報等）の４段階のような発展段階）がある場合

には、その発展段階を辿って理解を深めさせる事が大切である。

オブジェクト指向の提唱者達は、手順的や、機能的理解と、オブジェクト指向による理解がいかに異なるかという点に重点をおいて、オブジェクト、クラスとインスタンス、関連、関係、対応付けのような概念を最初に解説することから始め、構造化分析、構造化設計との違いを主張する教科書を書いている。業務処理情報システムでは、データ変換したがって（業務）機能が最大の関心事であるにもかかわらず、オブジェクト指向分析・設計では、機能を重視すると従来型のモデル化に陥る危険性を必要以上に警戒するあまり、データ変換アプローチを軽視する傾向が強い。これは、やむを得ない事ではあるが、問題は、このような教科書を使って、オブジェクト指向技術の教育が行なわれている点である。これが、業務処理情報システムにオブジェクト指向技術の普及を阻害している。

オブジェクト指向の提唱者達が記述した教科書、その他オブジェクト指向の啓蒙書の記述は、

- オブジェクト
- クラスとインスタンス
- 関連
- 関係
- 対応付け
- 汎化
- 継承
- 多相性

のような書き出しになっていて、まず対象間の関連や関係（UML ではクラス図）から教育される事が多い。このため、実際のソフトウェア開発にあたって、オブジェクト（クラス）から設計するように思われることも多い。その場合、ソフトウェアの開発経験が豊富な者（ほとんど、手順的や、機能的理解で仕事をしてきた者）にとっては、断片的にソフトウェアを開発するような感じになり、こんなことで本当に情報システムが構築できるのだろうかと心配になる。手順的や、機能的理解に慣れ親しんだものは、順序であるとか、情報の流れが気になるものである。バラバラなオブジェクト間の関係分析をやっていても機能が見えてこないように感じるものである。そのため、手順的や、機能的理解に基づいてソフトウェアを作成してきた技術者は、必要以上に難しさを感じてしまったように思われる。

オブジェクト指向分析、設計の最終形態は、操作の対象（情報も対象である）の側に視点を置かれたモデル化になるのであるが、入り口は、必ずしもオブジェクト（クラス）から出発点にする必要はない。設計結果がオブジェクト（ク

ラス）に視点が置かれたモデル化になれば良いのである。

7.1.4.2　適切なオブジェクト指向の教育法

UML のユースケースは、今回調査した企業のほとんどで、ソフトウェア開発の、最初の段階の要求定義に有用であり、活用していると報告している。しかし、純粋なオブジェクト指向の専門家は、ユースケースは、本来オブジェクト指向の考え方からは邪道であり、クラスから教育すべきであるという主張も未だに多い。ところで、ユースケースは、4.7.2 と 4.7.3 項で紹介したように、ユースケース図とシナリオ記述（ユースケース記述）とからなるが、ユースケース図から入るのは、なかなか難しく、シナリオ記述から、入るのが入りやすい。シナリオ（4.7.3 項のイベントフロー）は、手順的に記述されるのが普通である。シナリオから、名詞を抽出してオブジェクトの候補ないしは、オブジェクトの属性を抽出する方法(4.7.4 項)が、容易にオブジェクト指向に入りやすいが、このような教育指導が行なわれる事は少ない。

さらに、この方法でも、適切にオブジェクトを切り出すことは難しい。これを補う方法に、CRC（Class Responsibilties And Collaboration Card）法[4]がある。この方法は、我々が日常の仕事を行なう際に、自分自身では出来ない仕事や、人にやってもらったほうが効率の良い場合に仕事をその人に依頼する方法と同じ方法である。すなわち、仕事を依頼できる人に相当するものをオブジェクトにする方法である。次のような考え方に基づく方法である。

（１）オブジェクト自体が何らかのサービスを行う責務をもった主体であるか、あるいはそのオブジェクトに対して管理者を置くほど重要なものをオブジェクト（対象）にする。

（２）このようにして抽出されたオブジェクトは、他のオブジェクトからの求めに応じて、定められたサービスを提供する責務（あるいはオブジェクトの管理者が行う責務）を持つ。このような考え方は、責務駆動型の設計法[5](Responsibilty-Driven Design)とも呼ばれている。このように教育を行えば、オブジェクト指向をより容易に理解させることができるが、我が国ではあまり未だ普及していない。

メッセージのやりとりで仕事をするというモデルを理解させた上で、オブジ

---

[4] DAVIDA,TAYLOR：BUSINESS ENGINEERING WITH OBJECT TECHNOLOGY WILEY1995 及び
DAVIDBELLIN 他著：実践 CRC カード（翻訳　ピアソン・エデュケーション 2002/9）
JUAS に、研修講座も用意されている。

[5] Rebecca Wirfs-Brock et.al：Object Design, Addison-Wesley 2003

ェクト間の関係、関連それから、複雑性を解消する手立てとして、一般化（汎化)、や抽象化いうような概念を教育するのが、ソフトウェアの開発経験が豊富な者（ほとんど、手順的や、機能的理解で仕事をしてきた者）にとっては理解しやすい。

繰り返しになるが、オブジェクト指向技術の導入にあったって、把握しなければならない重要な概念を整理すると、

- オブジェクト（対象）は、何らかのサービスを行う（対象に対して操作を施す）責務をもった主体であって、そのようなオブジェクトに仕事を依頼するメッセージを渡し、仕事の結果を得るようなモデル化。（これをメッセージパッシング機構という。）
- オブジェクト（対象）が責務として他に提供するサービスの具体的実施方法。（これを Information Hiding という。）

の 2 点であって、その外は、主としてオブジェクト指向技術を用いてソフトウェアを作成する際に必要になる技術である。

7.1.4.3　耳慣れない用語が多いことによる取り付き難さ

次に、発祥の事情から、人工知能や知識工学の耳慣れない用語が必要以上に使われている問題がある。オブジェクト、クラス、インスタンス及びメッセージパッシングは、最小限の知識としても、汎化[6]、継承[7]など非常に難しい言葉が使われる。新しい概念を定義するためには、新しい言葉を使う必要があるとは言っても、オブジェクト指向を学ぶ者にとっては、敷居の高さを感じるのも事実である。

しかし、メッセージのやりとりで仕事をするというモデルが重要であって、このことをまず理解させてから、システムの複雑性をとりのぞくための、手法として汎化、継承等があることを教育することで、かなり難しさが改善できるのではないかと思っている。

7.1.4.4　オブジェクト（クラス）の抽出は一様にならない難しさ

第 3 の難しさは、オブジェクト指向を用いて作成されるモデルは人によって一様にはならないという事である。オブジェクト指向を用いて作成されるモデルは人によって一様になれば、オブジェクト指向を理解することは、比較的容

[6] 汎化(Generalization)：複数のクラス間で共通性を抽出してより汎用的なクラス（スーパークラスという）を作ること。例えば、リンゴ、みかん、バナナというクラスから、「果物」という一般化したクラスをつくること。

[7] 継承(Inheritance)： 汎化により「果物」という一般化したクラスに、リンゴ、みかん、バナナというクラスが含まれている形ができるが、リンゴ、みかん、バナナはすべて「果物」の性質を継承すると言うように使用する。

易になる。しかしながら、残念なことに、同じ要求課題に対して何をオブジェクトとするかは、情報システム化の目的や、モデル化を行う者の着眼点によって異なる部分が生じる。これは、経営に活用される情報システムについては、究極的には、経営の価値観及び設計者の価値観に結びつくもので、多様性はある程度避けられない性質のものである。

例えば、

- あるオブジェクトを情報システムの基本となるオブジェクトとして独立にとらえるべきか、あるいは基本となるオブジェクトの属性とすべきかについては見解が分かれることが多い
- オブジェクトとして独立にとらえるべきか、あるいはあるオブジェクトの役割（ロール）と見るか見解が分かれる

などの差異が生じる。ただし、福音は、E.Gamma 等が、実装設計の「パターン」[8]を提唱したことが引き金となって、要求分析段階のパターン化（オブジェクト指向分析のパターン[9]、オブジェクト指向によるビジネスモデリング[10]など）の優れた教科書が出版されており、オブジェクト抽出の「作法」を学ぶことができるようになってきた。

[8] E. Gamma 他「デザインパターン」（日本語訳　ソフトバンク 1995）
[9] M. Fowler 他「アナリシスパターン」（日本語訳　アジソン・ウェスレー 1998）
[10] H. K. Eriksson 他：「UML によるビジネスモデリング」（日本語訳　ソフトバンク 2002）

7.2 企業がオブジェクト指向に取り組む為の指針

企業がオブジェクト指向に取り組むためは、次の諸点を考慮する事が望まれる。

① オブジェクト指向が、業務プロセス設計、情報処理システム設計及び実装について将来、重要性がますことを認識し、将来にそなえること。
② オブジェクト指向を理解することの出来る者と、出来ない者がいることを認識して育成をはかること。
③ オブジェクト指向を理解することの出来ない者も、情報処理システムの実装では、活躍の場があるので、その仕組みを整備する。
④ ビジネスオブジェクトコンポーネントを活用、創生・蓄積する文化を築くこと。

まず①について検討する。近い将来のネットワーク中心の IT インフラは、ネットワークに単にプロセッサが接続された形態から、次図に示すように、情報を提供するサービスを実施する単位がネットワークに接続される形態になると考えられる。その情報提供サービスの単位がプロセッサに割り当てられ、しかもその割り当ては動的に変化することもある。すなわち、プロセッサは二義的なものになり、どのようなサービスが提供されるかということが主となる。

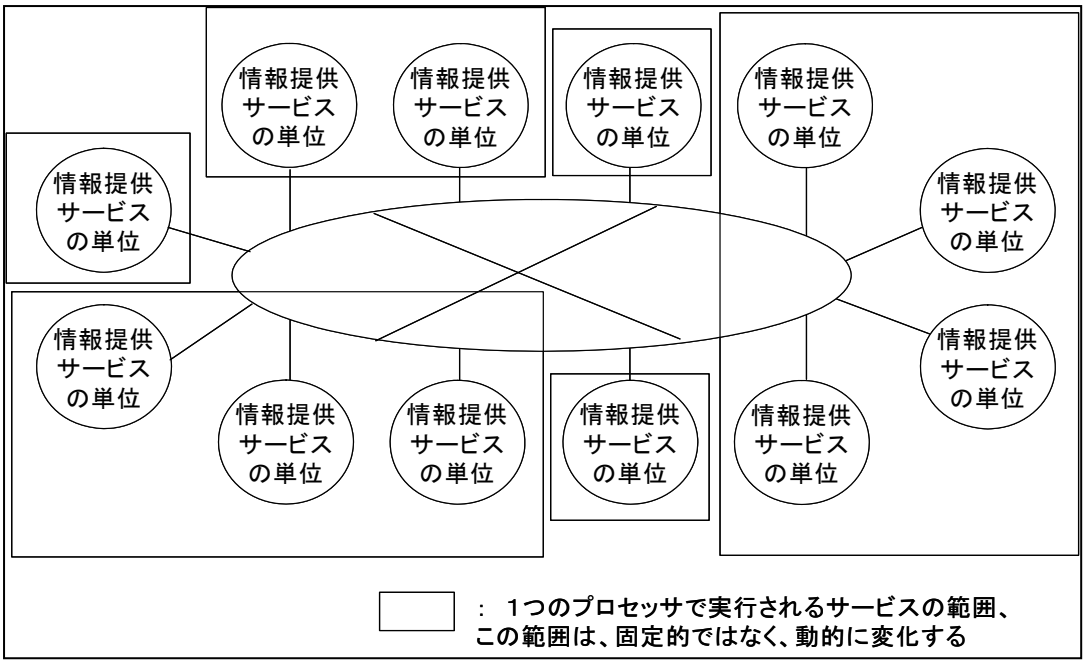


図 7-4 将来のネットワーク中心 IT インフラの姿

上記の形態は、プロセッサを人間と見て、ネットワークを LAN のようなメカニックなものではなく会話や、書類の受け渡し等のコミュニケーションと考えれば、人間の組織におけるフラットな協業の形態と同一である。

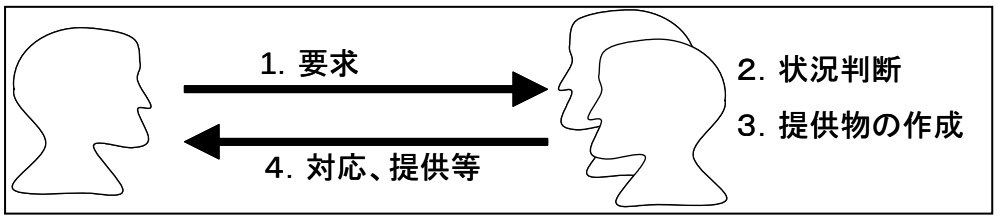


図 7-5　人間間の協業のモデル

（これは、オブジェクト指向のメッセージパッシングのモデルと同一である）

情報提供サービスの単位で行なわれるサービスが、機械的なものであればコンピュータプロセッサに行なわせれば良く、創造的なものであれば人間が行なうという状況になる。IT も、まちがいなく、図 7-4 の形態の実現に向かっている。情報提供サービスの単位は、オブジェクトである。このような形態では、業務のプロセスを設計することと、IT のアーキテクチャを設計することとの間に差が無くなって行くことを示している。IT 化を行なう、行なわないに係わらず、業務のプロセスをオブジェクト指向技法でモデル化を行う。経営環境変化に適応力のある、業務遂行や、組織編制を得ることを目的として、業務をオブジェクト指向でモデル化する。出来上がったモデルにおける業務オブジェクトのうち、機械的な部分を IT 化する。このようにすれば、ビジネス・プロセスリエンジニアリングと IT 化検討を同時に行なうことができる。変化への即応がますます求められる経営環境では、きわめて有効な手法と考えられる。

次に②を検討する。オブジェクト指向技法の適用については、次の(A)、(B)を区別する必要がある。

(A)オブジェクト指向技法による業務プロセス設計
(B)IT 化する業務の実装へのオブジェクト指向技法の採用

前節で、オブジェクト指向の技術者の育成には、根本的な難しさがある事を述べたが、それは、主として上記(B)についてである。(A)の場合は、適切な教育、例えば CRC（Class Responsibilty Collaboration）カード法等を学ぶ事によって、マスターすることが可能であることは前節で述べた。(B)におけるオブジェ

クト指向の技術者を教育する事は難しい。それは、実装では、業務のオブジェクト化に加えて、ソフトウェアの複雑性を低減し、高品質（機能性、信頼性、使用性、効率性、保守性、移植性等の観点）ソフトウェアを構築するという観点が重畳されるためである。従来のソフトウェア開発においてソフトウェアの高品質化を行なってきた経験がかえって、オブジェクト指向の考え方を理解する足枷になるからである。実装技術としては、クラスの階層をどうするか、継承をどうするかなどを完全にマスターする必要がある。

例えば、継承等の抽象化の主たる目的は、ソフトウェアの記述量の削減、従って開発用を削減するものである。今回の調査で、オブジェクト指向技術を導入しても、機能拡張、機能改良のいわゆるメンテナンスコストの削減効果をあげられないという話を聞いたがその理由の一つは、オブジェクト指向技術が未熟なため、継承などを必要以上に使用したため、プログラム間の独立性をそこなっていることが原因になっていることが考えられる。ちなみに、優れたオブジェクト指向実装技術者は、あまり継承を使用しないで、オブジェクト指向技術としては、「面白みに欠ける」が、プログラム間の独立性を維持できる集約[11]を多く使用する傾向にある。

また、現実のソフトウェア開発では、やはり最後に性能という問題が起こり、当初の「美しい」ソフトウェアキテクチャが崩れる事がしばしばあるが、多態性の多用による動的結合[12]のオーバーヘッドも留意すべき問題がある。

更に、適当なコントロールオブジェクト（全体のプログラムの動きを司るオブジェクトであるが、大きなコントロールオブジェクトを作ってしまうと従来の手順型のプログラムに陥ってしまう危険性を含んでいる）を作ってダイナミックなプログラムの実行を管理する仕組みを作らないと、実行時の障害解析が非常に困難になる場合があるなど、オブジェクト指向の実装に関する技術力が非常に重要になってくる。

実装技術者として、重要な事は、従来の情報技術の優れた知見を活かしながら、効果的にオブジェクト指向技術を活用して実装を行えることである。設計したプログラムに対し、実行時の性能、障害時の情報収集、機能拡張・改良のし易さ等の異なる視点から評価できる、すなわち、作る視点とは異なる視点を行きつ戻りつできる複眼の視点を有することであり、これは、一朝一夕には身につけることが難しい。

いずれにしても、本人の適正を良く考慮した教育が必要であるとともに、良

11　集約(Agrigation)：全体とその構成部品との関係、シャーシ、車輪その他から自動車ができているというような関係をあらわす。

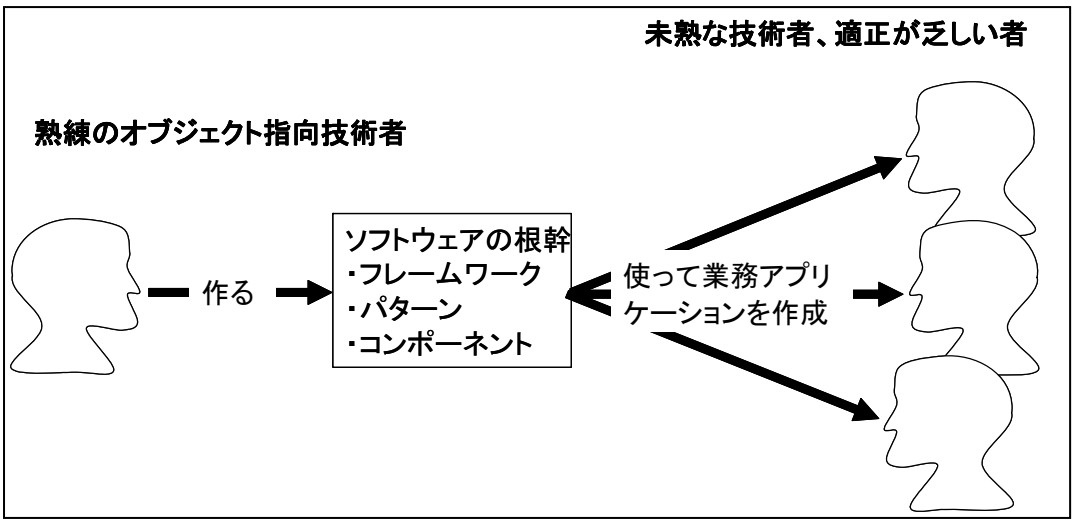
12　多態性(Polymorphism)と動的結合(Dynamic Binding)：例えば、面積計算と言っても、図形が円の場合、三角形の場合、矩形の場合で求め方がことなるが、
すべてについて、面積計算という言葉でプログラミングをしておき、プログラムの実行時に対象の図形が決まったときに、その図形に応じたプログラムを動的に呼び出して面積計算を実行するような方法。

い技術者の指導の下で、オブジェクト指向技術を用いたソフトウェア開発経験を積ませる必要がある。前節で述べたとおり、教育を施しても、オブジェクト指向を理解する事ができない者もいるので、無理はしないほうが良い。また、オブジェクト指向を育成するには時間がかかるので、適正があると思われるものは、気長に育成する必要がある。

ただし、福音は、E.GOMMA 等が、提唱した「デザインパターン」[13]が核となって優れたオブジェクト指向活用のパターンが出版されており、優れた設計を学ぶことができるようになっていることである。更に、コンポーネント活用の基盤として、マイクロソフト社が提唱する COM/.NET、サン社外がすすめている Javabeans/EJB が普及してきており、これらを活用することによって確実に技術力を高められる土壌ができてきている。

次に③について検討する。

企業がオブジェクト指向に取り組むためには、B 社、A 社で実施されている、優秀なオブジェクト指向技術者が、ソウトウェアの根幹となるフレームワーク、パターン、コンポーネントを作成し、オブジェクト指向に未熟なもの、あるいはオブジェクト指向の適正が乏しい従来型のソウトウェア技術者は、フレームワーク、パターン、コンポーネントを使用して業務アプリケーション（一過性のアプリケーションの中には、手続き型でプログラムを作成しても問題が少ないものもある。）を作成するような分業化をはかる。すなわち、オブジェクト指向によるモデル化を行う者と、その成果を利用するものとを分ける方法が最も



現実的と思われる。

図 7-6　オブジェクト指向技術力のレベルを考慮した開発体制

[13] E.GOMMA 他「デザインパターン」（日本語訳　ソフトバンク 1995）

なお、これと並行させて、実際的ではあるが、小規模のソフトウェアを、オブジェクト指向技法を用いて開発するプロジェクトを推進し、将来、ソウトウェアの根幹となるフレームワーク、パターン、コンポーネントを作成に従事できる技術者をオンザジョブトレーニングにより育成する事も重要である。

次に④について検討する。重要なことは、ビジネスオブジェクトコンポーネントの開発と、ビジネスオブジェクトコンポーネントの活用を促進するためのリーダーシップのある定常組織が機能する必要である。この組織のミッションは、ビジネスオブジェクトコンポーネント開発の指針を示し、その指針に基づいて作成されたコンポーネントの活用を推奨し、仮に、コンポーネントを活用できる状況下で、活用されないケースが発生した場合には、その原因を徹底的に追求し、必要があれば、広く活用できるように、コンポーネントを改良することにある。良いビジネスオブジェクトコンポーネントが揃い、そコンポーネント活用が普及するようになると、組織としてオブジェクト指向技術力は急速に向上するものである。

7.3 技術の動向

7.3.1　オブジェクト指向ビジネスコンポーネント

今回の調査では、ビジネスコンポーネントの開発、流通は、残念ながら低調であったと考えざるを得ない。しかも、1 企業内にあっても、社内でビジネスコンポーネントを活用しているところはほとんどない状態であった。これに対して、フレームワークを整備し、Web 用のアプリケーションを効率的に開発できるようにしているのが、A 社をはじめ、ほとんどの企業で実用化している方法であった。汎用的なビジネスコンポーネント作成プロジェクトとしてのサンフランシスコ・プロジェクト(図 7-7)が成功しなかったように、企業内、企業間をこえた、ビジネスコンポーネントの流通には、相当時間がかかるようである。

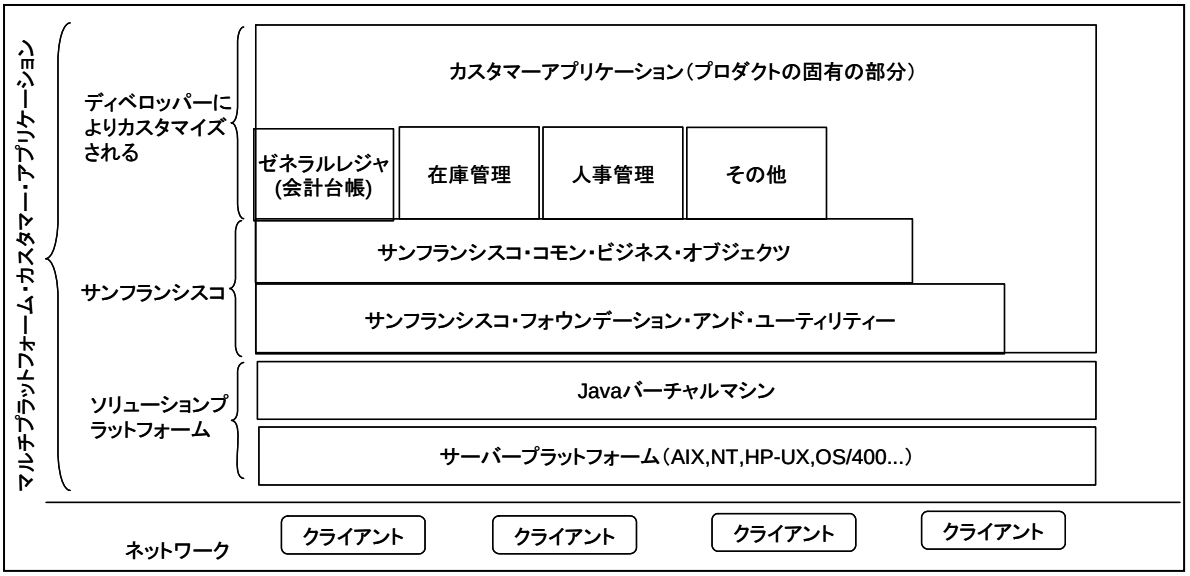


図 7-7　サンフランシスコアーキテクチャー

（Internet Magazine 1997/10 P.368）

A 社での経験では、ソフトウェアの再利用において、JAVA Beans のような細かいオブジェクト指向部品レベルと、ERP パッケージのような大きな単位のレベルが実用化されて、ビジネスコンポーネントのような中位レベルの部品はなかなか実用にならないとのことであるが、興味深いことである。ここ数年の傾

向では、JAVA Beans のような細かいレベルで、極めて汎用性の高いオブジェクト指向部品と、MVC(Model/View/Controller)アーキテクチャに基づくフレームワークの実用化が進むと見るのが常識的ではあるが、標準的な見解である。

フレームワークについては、各社各様に作られたものと JAKARTA プロジェクトで提案されている STRUTS のような汎用的なものとがあるが、今後は、汎用的なものが充実すると思われる。

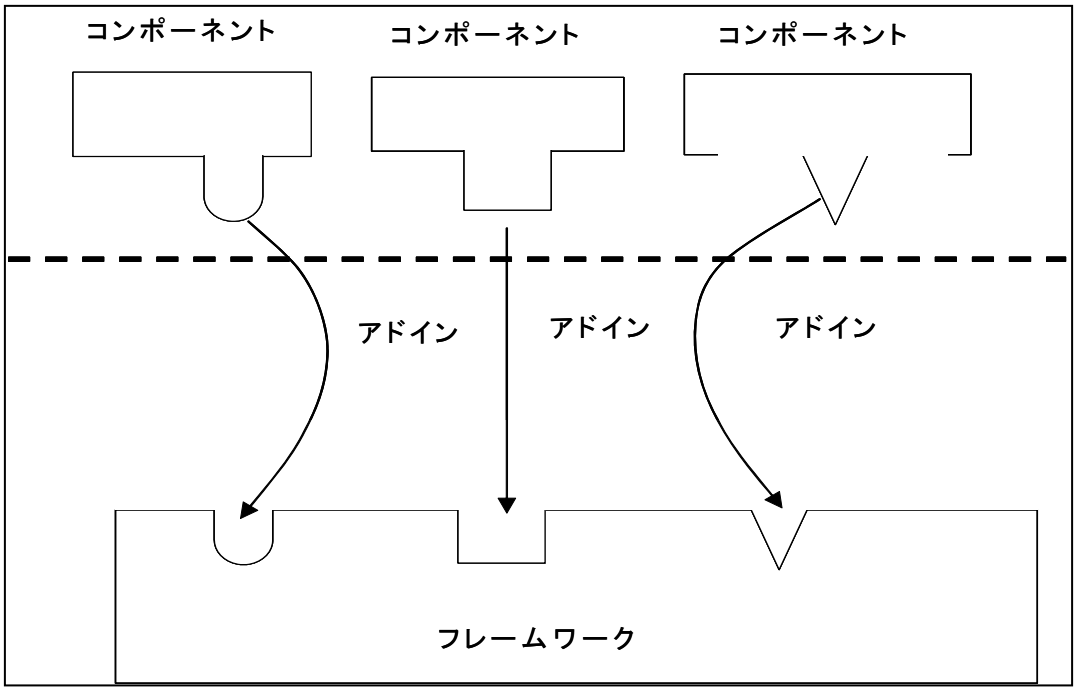


図 7-8　フレームワークとコンポーネントとの関係イメージ

（JAKARTA プロジェクト徹底攻略　P.11　技術評論社 2002/10/6）

7.3.2　企業間ビジネス連携へのオブジェクト指向技術の活用

ブロードバンド・インターネット時代を迎え、ますます企業間取引にインターネットを活用することが多くなると考えられる。SCM(Supply Chain Management)等が有効に機能するためには、企業間の取引における情報の伝達（企業間でやりとりするデータ（伝票に代表される文書データ)、実際にデータをやりとりする手順（ビジネス・プロセスの手順)、伝票データを送るための通信機能（通信プロトコル））標準化が進まなければならない。

1998 年に設立された Rosettanet 等のインターネットを利用したサプライチェーン構築のための標準設定と実用化の試みは、着実に IT 機器業界、電子部品

業界、半導体製造業界、自動車部品業界(Rosettanet ではなく、Covisint が有力)等の実業界に浸透しているように思われる。しかしこれらは、特定の取引先との決まった取引を行う仕組みである。

これに対し、インターネットを利用して、不特定多数の相手と自由に取引をする試みが行なわれている。その中で、有力かつ将来性がある考えられている試みが、1999 年に設立され、18 ヶ月の期限付きで、2001 年 5 月に終了した EBXML(Electronic Business Using Exetensible Mark Up Language) Initiative で行なわれた XML をベースとした企業間取引システムの枠組みである。

インターネットを利用して、不特定多数の相手と自由に取引（企業間でサービスを交換する事とみなす事ができる）を行なうためには、

① 企業間の取引において伝達される情報の標準化が一層徹底される必要がある。

- 企業間でやりとりするデータの標準化…Rosettanet の伝票情報の標準化である PIP(Partner Interface Process)など。

- 実際にデータをやりとりする手順の標準化…EBXML INITIATIVE から提案されているビジネス・プロセス手順の標準化である BPSS(Business Process Specification Schema)など。

- 伝票データを送るための通信機能の標準化…信頼性が高く、非同期通信の出来るオープンな通信プロトコルとして、SOAP(Simple Object Access Protocol)をベースとした機能拡張・機能強化が、OASIS(The Organization for the Advancement Of Structured Information Standards)等で進められている。

② サービスの登録と検索機能の標準化

- 電話帳、サービス検索ガイドブックに相当するビジネスレジストリの標準化…OASIS で標準化が進められている UDDI(Universal Description, Discovery And Integration)ある。ネットワーク上のサービスにアクセスするためのインターフェースを定義する XML フォーマットである WDSL(Web Service Description

Language)が W3C(The World Wide Web Consortiumonsortium)で標準化されている。

③ 取引の合意を形成する手順の標準化

- 電子的な合意形成手順…EBXML Initiative で、
  CPP(Collaboration-Partnerprofile)と
  CPA(Collaboration Partner Agreement)とが標準化された。

以上の技術の基盤となっているのが、オブジェクト指向技術である。

ただし、インターネットを利用して不特定多数の相手と安全な自由取引が可能になるためには、各種のバリデーション機能が強化される必要がある。そのような機能に、

- リソース管理
- 運用管理
- 負荷分散
- フォールトトレランシー
- 構成管理
- 各種ペイメント処理

等がある。

さらに進めて、人手によるオペレーションを省き、自動化をはかる WEB サービスの活用も将来の方向と思われる。

7.3.3 WEB サービス

将来インターネットの活用では、インターネット Web サービス（図 7.9）化が進むと予想されている。

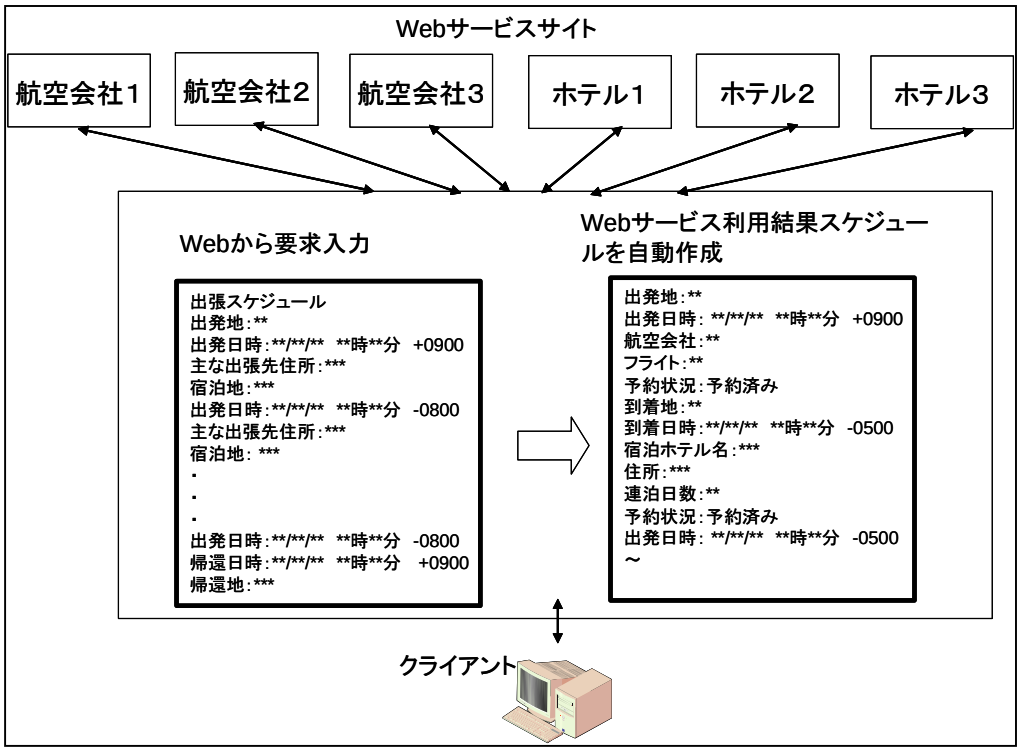


図 7-9　WEB サービス利用を利用した出張スケジュールの作成イメージ

図 7-9 に使用した例では、利用者が Web から要求情報（出発情報と帰還情報）の入力、クライアントアプリケーションが Web サービスのメカニズムを用いて、WEB サービスを提供するサイトと自動交信を行い、航空機、ホテルの予約を自動的に行い、出張スケジュールを自動作成する。Web サービスのメカニズムを、図 7-10 に示す。

インターネットを活用した B to B、B to C、G to B、G to G、G to C 等を効率的かつ高信頼に行なうためには、クライアントとサーバ間のやりとりにおいて、人手を出来る限り省く事が必要になると思われるので、Web サービスへの期待が大きい。

Web サービスでは、クライアントとサーバ、サーバ間の文書情報の交換には XML が使用されるのが標準的である。Web サービスに用いられる通信プロトコルとしては、下位レイアに HTTP をサポートした SOAP が本命である。それは、インターネットの標準化機関の W3C[14]が提案した点と、最近セキュリティ対策として用いられるファイアウォールに特別な設定をすることなく、ファイアウォール内のコンピュータ間の情報交換が出来る、いわゆる相互運用性(Interoperability)が高い点が評価されるためである。どのようなサービスが提

[14] http://www.w3c.org/tr/soap

供されているか探索するためには、レジストリ（サービスの電話帳）が必要であるが、検索のためのレジストリ仕様としては、UDDI(Universal Description, Discovery And Integration)[15]が本命である。また、レジストリにサービス内容、アクセスのためのインターフェース等の情報を登録する際に用いられる情報記述用言語として WSDL(Web Service Description Language)[16]が用いられる。

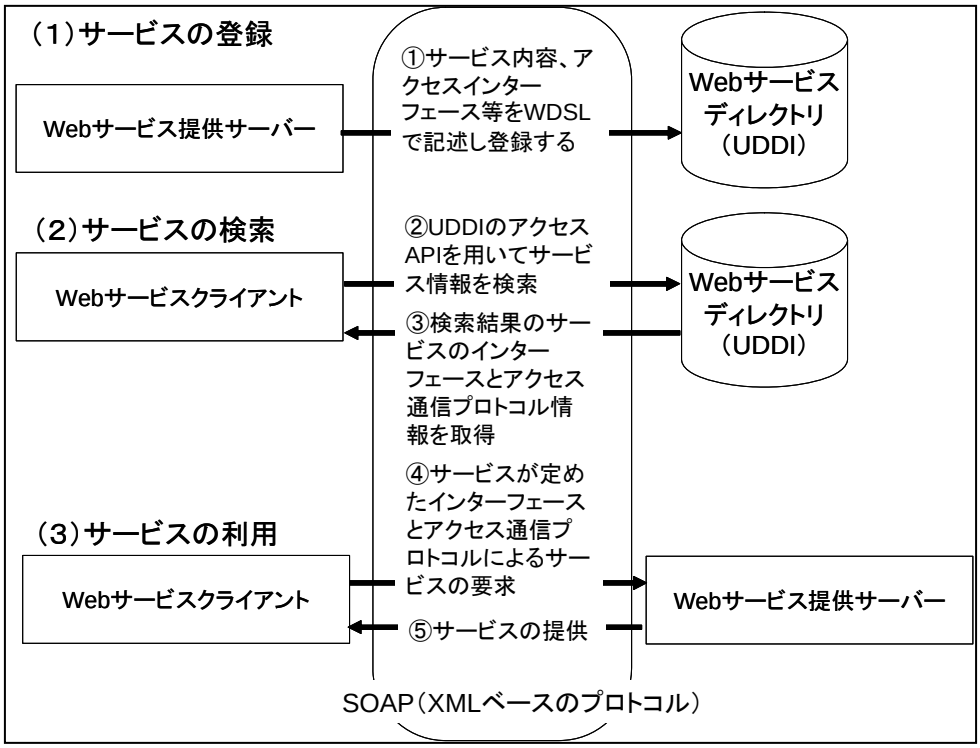


図 7-10 WEB サービスの概要

7.3.4 情報オブジェクトの利用

前節で Web サービスについて述べたが、Web サービスは、新しい形態のクライアント/サーバシステムである。Web サービスの場合、サーバ側にサービスを提供するオブジェクトが存在する形態である。更に、将来を考えると、情報オブジェクト（データとその意味を定義するプログラムとがカプセル化されたもの）が、ネット中心に運用されるサイバー空間において、自律性(Autonomy)をもって自由に動き回り、利用者はそれを捕らえて情報処理を行うようなモデルも考えられる。

[15] http://www.uddi.org/specification.html
[16] http://www.w3c.org/tr/wsdl

このようなことが実現して初めて、ネットワークが、真の意味の「情報処理」の道具となる。これが実現しない内は、どのような高度の機能があっても、「データ処理」に留まっていると言っても過言ではないような気がする。

もともと、意味の記述には、

- 代数的意味論（形式論理を用いて、手順を考慮せずに数学的に意味を与える方法)、
- 関数的意味論（関数で意味を与える方法、LISP 言語の定義に用いられた)、
- 操作的意味論（手順的な言語で意味を記述する方法・・・計算機言語で意味を記述する方法も含まれる。）のうち、応用範囲、実用性が高いのは、操作的意味論である。

このことを考えると、情報オブジェクトの意味の記述として、JAVA アプレットを使うのが、良いかもしれない。JAVA アプレットは、レスポンスが悪く、現在、思うようには実用化が行なわれていないようである。関心はサーブレット JAVA に移ってしまった感じが強いが、ブロードバンドネットワーク時代には、JAVA アプレット及びサーブレットともにますます活用されるようになる。

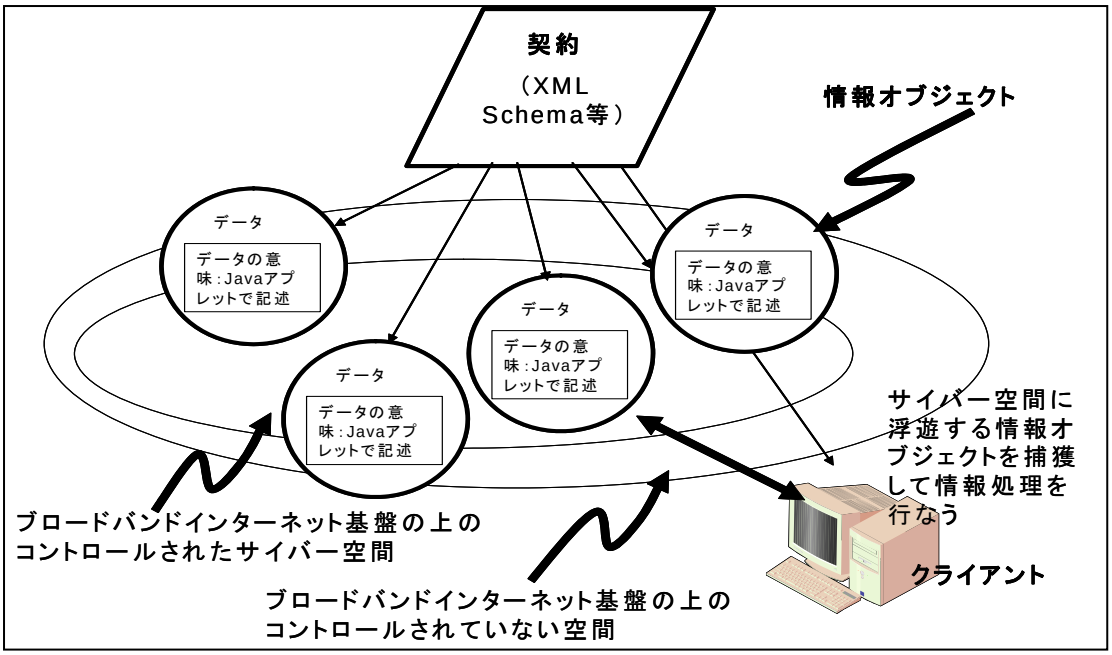


図 7-11 究極のネット中心インフラ

7.3.5 XML

オブジェクト指向技術と親和性がある技術に XML(Extensible Markup Language)がある。

XML は、

（１）データに開始と終了のタグをつけて、可変長のデータを表現し、プレインテキスト形式で、システム間で交換できる。

例. 〈住所〉東京都中央区小伝馬町 15-17〈／住所〉

（２）データの階層構造を入れ子のタグを使って表現できる。

例. 〈全国中核市一覧〉

〈都道府県〉北海道〈市〉旭川市〈／市〉〈／都道府県〉

〈都道府県〉秋田〈市〉秋田市〈／市〉〈／都道府県〉

〈都道府県〉福島〈市〉郡山市〈／市〉

〈市〉いわき市〈／市〉〈／都道府県〉

・・・

〈都道府県〉鹿児島〈市〉鹿児島市〈／市〉〈／都道府県＞

〈／全国中核市一覧〉

（３）表示、印刷等のフォーマットは、別途自由に定義[17]できる。

という３大機能を有する、情報交換用の言語である。

XML は、多くの異なる種類の情報システム間で情報交換の手段が必要になり始めた 1960 年代に IBM で開発された GML(Generalized Markup Language)に源流がある。その後、1978 年から 1986 年の間に SGML(Standared Generalized Markup Language)として、文書情報を交換する標準として活用されるようになった。考え方が世界中に広く使用されるようになったのは、1989 年に、Tim Berners-Lee が、インターネットのホームページ記述言語として HTML(Hypertext Markup Language)を採用してからである。XML は、ユーザが独自のタグを使って、データの属性や構造を定義できる（これは HTML では不可能[18]であった）情報交換用の言語として 1998 年に W3C で正式の標準が発表された。

XML が、オブジェクト指向技術と親和性がある例は、DOM(Document Object

---

[17] 報告書や、Web ページのレイアウトフォーマッテイングは、XML とは別に XSL(Extensible Stylesheet Language)で記述する。文書情報を、(1)文書の内容・構造、(2)レイアウトフォーマッテイング記述、(3)具体的な印刷、表示のページ記述を分けて扱う思想は、SGML の言語体系で既に確立していた。(1)SGML(2)DSSSL(Document Style Semantics and Specification Language)、(3)SPDL(Standared Page Description Language)がそれぞれ対応している。ネットワーク中心の時代を反映して、SGML が、XML に、DSSSL が、XSL に、新しい装いで登場している。

[18] HTML のタグは、どちらかと言えば、タグに囲まれたデータを Web ブラウザー上でどのように表示するかを定めたもので、Web ブラウザーの領域で、ユーザが使いやすさと、処理系のシンプルさを同時に実現したものである。このような設計は、実装の技術力と、センスがあってはじめて可能である。

Model)と呼ばれる XML 文書処理用の API(Application Program Interface)に見られる（図 7.12 参照)。

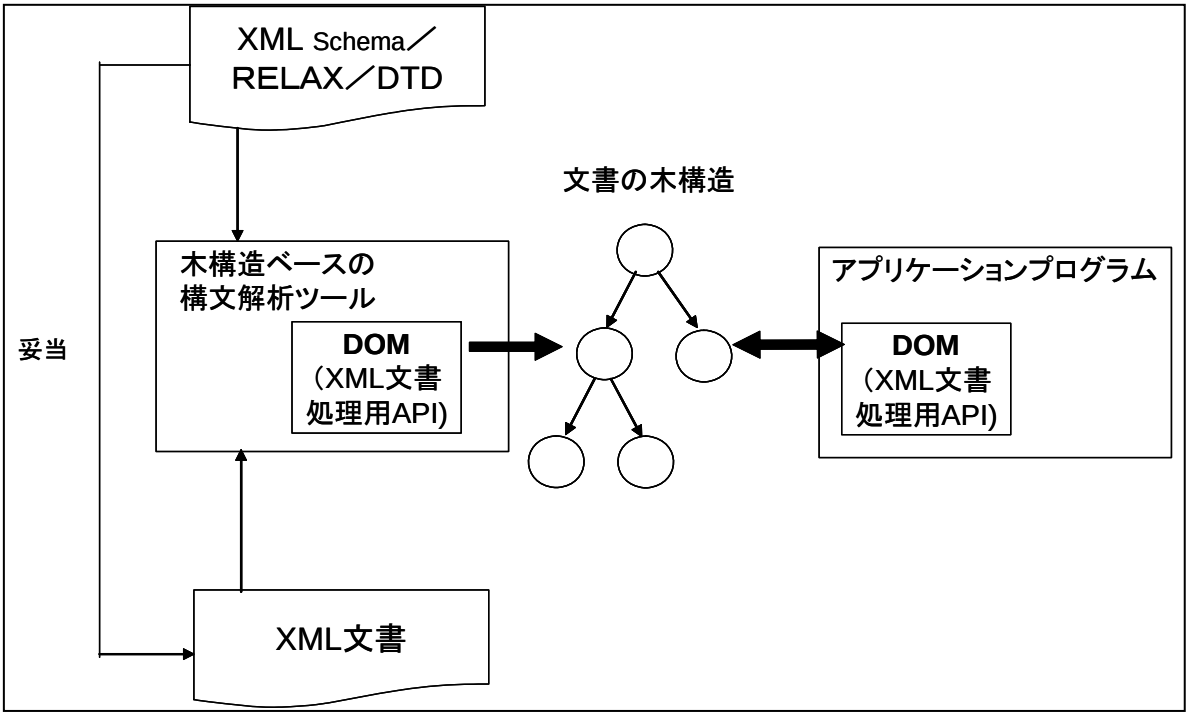


図 7-12 DOM の概要

XML 文書と、XML 文書が文法的に妥当なものかどうかを判定する基準（DTD(Document Type Definition)、XML Schema、RELAX[19]という言語で記述される）とを DOM を包含する構文解析ツールの入力とすると、構文解析ツールは XML 文書の木構造表現を作り出す。この木構造の、各ノードは、データ構造を表現するのではなく、オブジェクトになっていて、次に示すの情報を持っている。

（１）文書を表現・操作するインタフェースやオブジェクト
（２）これらのインタフェースやオブジェクトの意味･･･挙動とアトリビュートとの両方を含む
（３）これらのインタフェースやオブジェクトの間の関係や協調関係

この木構造化された XML 文書にアプリケーションプログラムから、DOM の機

[19] XML Schema は、XML の記述のルールを定める言語であり、DTD(Document Type Definition)の機能を強化した言語である。 主な強化点は、(1) DTD は特殊な構文が用いられていたのを、XML の構文に統一したこと、(2)XML の要素や属性のデータ型を指定出来るようにしたことである。2001 年 5 月 2 日に、W3C から勧告として発行された。ところが、XML Schema は、仕様が複雑で、使いにくいという批判があり、村田真氏が XML Schema に先立って開発した RELAX(Regular Language description for XML)を推す声も多く、XML Schema で世界中の足並みが揃っているわけではない。

能を使用してアクセスすることにより、各ノードの情報を取り込むことが出来る。

DOM を紹介したが、DOM を持ち出すまでもなく、XML 自体でクラスや、ユースケース記述等の内容を記述できるので、クロスプラットホームでのオブジェクト連携のためのインタフェース情報の記述が可能である。XML と XML の要素や属性データの型の、記述能力のあると組み合わせて、CORBA のオブジェクト間のインタフェース定義言語(IDL：Interface Description Language)相当の記述力を持つようになってきた。次項で述べる、SOAP は、オブジェクト呼び出しに必要な XML メッセージの交換を行うためのプロトコルである。

更には、オブジェクトのデータベースとしても活用できる。データベースの検索手段は、タグが付けられたデータを、タグを指定して、データのパターンマッチングによる方法が一般的である。

7.3.6オブジェクト連携（SOAP）

7.3.6.1オブジェクト連携（SOAP）

今までのオブジェクト間連携については以下のような技術が提唱されてきた。

- COM(Component Object Model)では、RPC (Remote Procedure Call)
- CORBA (Common Object Request Broker Architecture)、IIOP(Internet Inter-ORB Protocol)

これらは、それぞれの世界では、それなりの有効な手段である。が、クロスプラットホームでのオブジェクト連携が必要なってきている。そのために、XML のシンタックスを利用して、システム間連携を実現出来るような仕組みが検討され、その結果、SOAP(Simple Object Access Protocol)が提唱された SOAP は、単純にオブジェクトへアクセスするためのプロトコルの仕様である。

SOAP は、HTTP や SMTP といったプロトコルの上位に位置し、そのパケットの中で、オブジェクト呼び出しに必要な XML メッセージの交換を行うためのプロトコルです。どんなプロトコルであっても、SOAP のメッセージを送信、もしくは受け取ることができ、そしてその内容をシステムが解釈できれば、問題ないとされている。

SOAP は、XML 文書に対してエンベロープと呼ばれる付加情報を追加して、オブジェクトへの宛先、あるいは必要に応じてメッセージ ID などの付帯情報を自由に付加できる点にある。

仕組みを説明すると、他の場所にあるオブジェクトの実行を要求するクライアントは、オブジェクトが提供されているサイトに対して SOAP メッセージを送信し、受け取ったサイトでは、それを解釈して、どのオブジェクトにマッピングするかを判断して、その実行結果を XML 文書（SOAP メッセージ）としてクライアント側に返信する。そのためには、クラアント側では SOAP メッセージを生成するエンジン、受け取り側ではそれを解釈するためのエンジンが必要となる。

実際にクラアント側では、実際のオブジェクト呼び出しが実行されたときに、その命令が SOAP プロキシといわれる SOAP メッセージのシリアライザに受け渡され SOAP メッセージが生成され、オブジェクトを提供するサイトへ送信する。

また、受け取り側では、SOAP リスナが、受け取ったメッセージを解釈し、どのオブジェクトを呼び出せばいいのかを判断、そして実行した結果を送信元に返す。

このようにして、送受信されている XML ベースの SOAP メッセージの仕様が標準化されていることで、異種プラットフォーム間でもオブジェクト間連携ができるようになった。

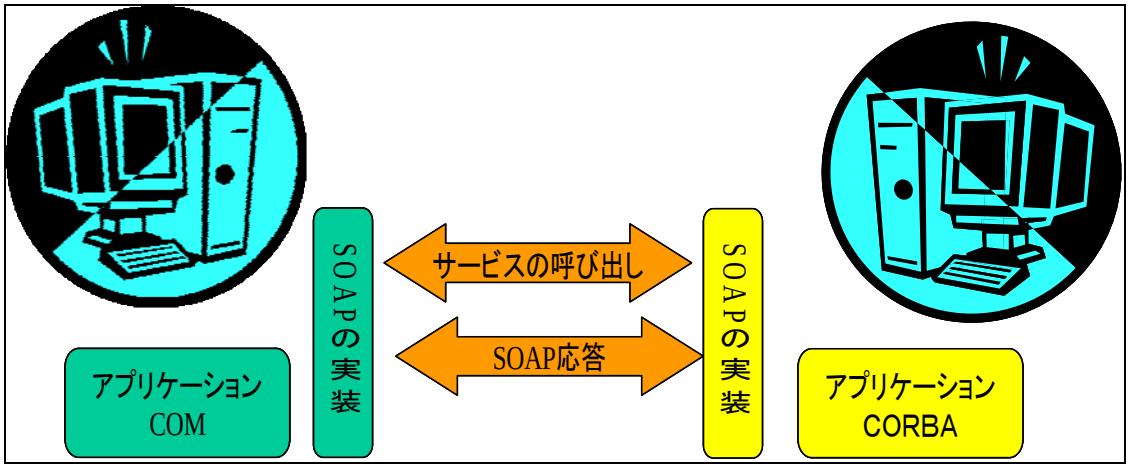


図 7-13 SOAP

7.3.6.2SOAP の構造

SOAP で規定しているのは、郵便でいうところの封筒の仕様であり、封筒には、あて名送り主を書き、中身を入れなさいというものである。SOAP メッセージは、次のような構造をしている

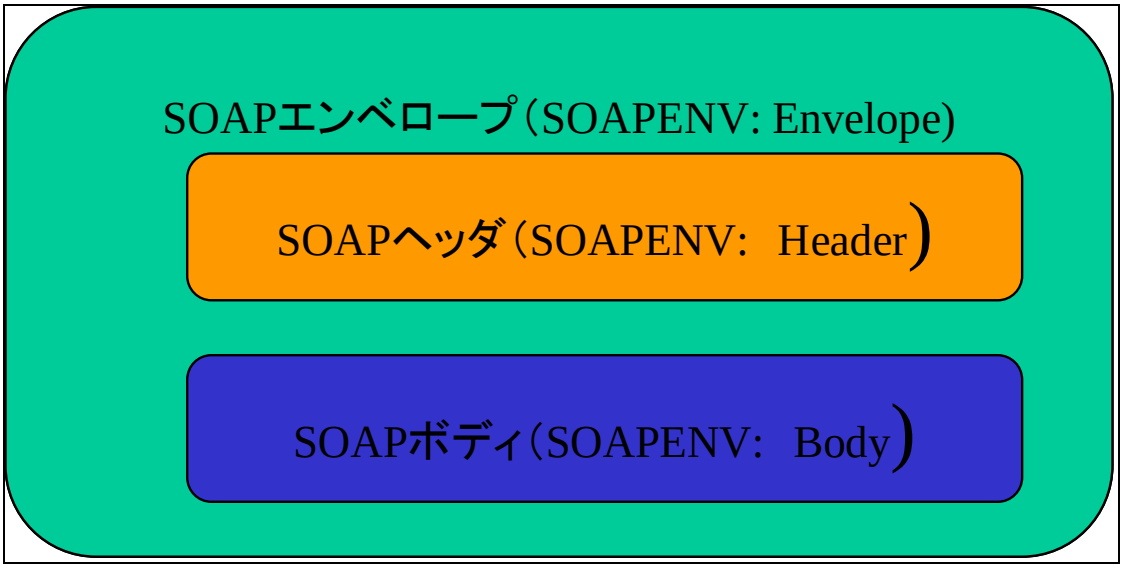


図 7-14　SOAP の構造

7.3.6.3　システム間の疎結合

このことで、コンポーネント化されたソリューションの持つ制限事項（技術をまたがって相互に関係のある動作ができないという欠点）を克服することが出来、さまざまな用途で構築されたコンポーネントは、他のアプリケーションからも再利用することができるようなる。すなわち、インターネットを通じて、これらのコンポーネント間が連携することができるようになったら、システムが自由につながることになる。その結果、インターネット上に存在するさまざまなコンポーネントやオブジェクトを結びつけることで、ソリューションを提供するという Web サービスが実現することにある。

また、SOAP の仕組みは、XML 文書の受け渡しということなので、仕組みとしては簡単で標準化されている。従って、システム間の結合は動的にでき、かつ、疎結合となる。

7.3.6.4　UDDI(Universal Description, Discovery, And Integration)

このプロジェクトは、マイクロソフト、IBM といった企業が中心になって活動を行っており、Web サービスがインターネットで公開された際の、動的な利用を支えるためのレジストリ機構を提供する。いわゆる、ディレクトリサービスである。

利用者は SOAP メッセージを利用して UDDI レジストリへアクセスし、必要な情報を取り出し、Web サービスの利用を進めていくことができるようになる。

UDDI は、サービスの利用者に対して自分に適した Web サービス発見するための機能を与え、Web サービス提供者には、レジストリへのサービス登録をすることでビジネスチャンス広げることとなる。

UDDI のホームページ（http://www.uddi.org/）には、ホワイトペーパーと仕様には、次のものがある。

- Exective White Paper
- Technical White Paper
- Programmer's API Specification
- UDDI Data Structure Reference

仕様の方を概観すると、UDDI のビジネスレジストリには、Web サービスを検索し利用する際に使われる情報を 4 つのカテゴリに分けて、それぞれが XML の要素として定義されている。これらの情報がレジストリに登録され、検索に使われることになる。

- ビジネス情報（Business InformationN）
  〈Businessentity
- 業務情報（Service Information）
  〈Businesservice〉
- バインド情報（Binding Information）
  〈Bindingtemplate〉
- サービス記述（Specification About Services）
  〈t model〉

7.3.6.5　ビジネスレジストリの構造

- 〈Businessentity〉という要素では、企業名、連絡先、企業を特定するためのものを記載する。企業を検索する場合、この要素を使って検索することになる。〈Businessentity〉要素での検索のことを「ホワイトページ」と呼んでいる。

- 〈Businesssrvice〉という要素では、企業がどのような業務を提供しているか表している。3 つの分類標準をサポートしている。

  - 業種： NAICS （Industry Codes - US GOVT.）

- 製品/サービス： UN/SPSC （ECMA）
- 場所： GEOGRAPHICAL TAXONOMY

〈Businesssrvice〉要素単位での検索を「イエローページ」と呼んでいる。

- 次に Businesssrvice がサポートする入出力の条件や、接続のプロトコル、接続先の URL などの技術的な情報が必要で、これらは〈Bindingtemplate〉という要素で記述されます。〈Bindingtemplate〉要素単位での検索を「グリーンページ」と呼んでいる。

<TMODEL>具体的な Web サービスの仕様を記述するための要素であるが、Web サービスの仕様を記述するためのメタデータである。Web サービスの仕様は、UDDI の中では定義されないことになっている。<TMODEL>には、名前、仕様を策定した団体、仕様への URL などが記述される。

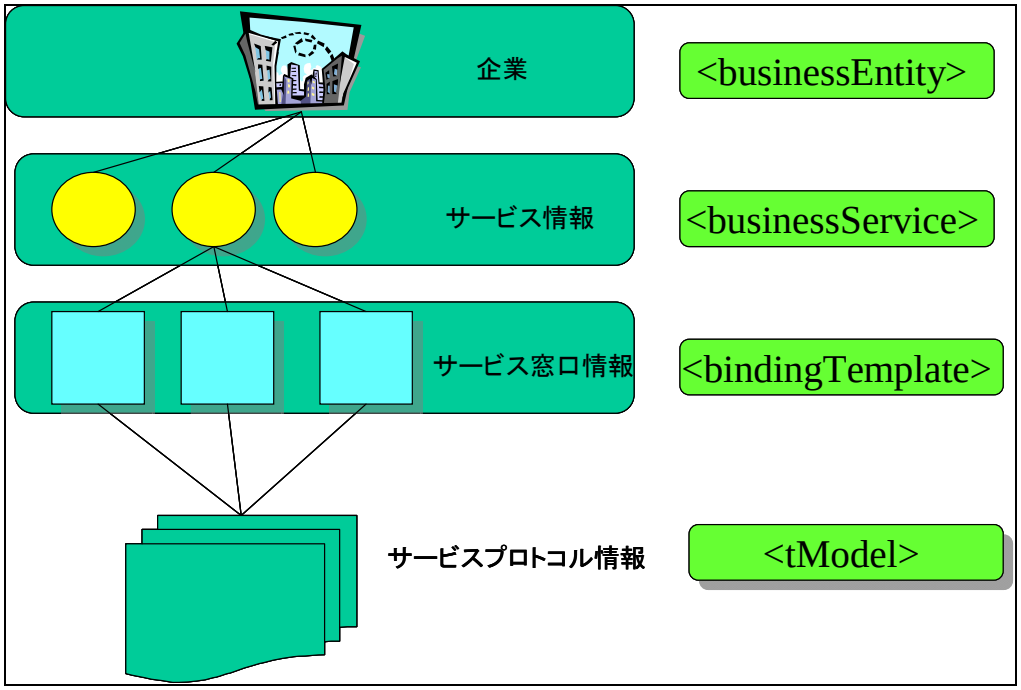


図 7-15　ビジネスレジストリの構造

7.3.6.6　UDDI での公開される WEB サービスの相互に運用基盤

UDDI を使うと、他の Web サービスを動的に発見し、接続することができる。このことは、Web サービスの世界で、完全にプラットフォーム、OS が違

う独立したプログラムをソフトウェア部品として使うことができるということを意味している。

UDDI によって、ある仕様が与えられたときに、その仕様を満足するための部品（Web サービス）の組み合わせを動的に検索して、組み立てる、いわゆる Just-In-Time Integration という考えが可能になる。例えば、利用者にあわせたツアーを組み立てる場合、航空会社の Web サービス、ホテルの Web サービス等の Web サービスを発見し、統合し、それらをアクセスしながら、最適な組み合わせを見つける必要がある。UDDI と Web サービスに基づく Just-In-Time Integration はこのようなシナリオを可能にすることとなる。

このように、Web サービスを必要に応じて統合して使うことで、新たな Web サービスを組み立てることが出来ることになる。すなわち、散らばっている Web サービスをソフト部品として使うことが出来ることになる。

7.3.7 MDA (Model Driven Architecture)

7.3.7.1 MDA（モデル駆動型アーキテクチャ、Model-Driven Architecture）

MDA は、モデリング主導のシステム開発、ライフサイクル管理を実現するアーキテクチャである。ビジネス・プロセスモデルから、コードまでがマッピングされることになる。

MDA とは、OMG のソーリー氏によると、
「いま求められていることは、アーキテクチャと設計の分離。既存のソフトウェアという資産を守りつつ、インフラに新技術を適用していく。それを実現するのが MDA だ」
ということになる。MDA は、“20 年持続するソフトウェアキテクチャ”を目標としたシステムの構築方法であり、UML でのモデル記述をコアとし、分析、設計、実装、保守、進化、統合というライフサイクル全体をサポートするものである。

「目標は、開発者が“モデル”と“コード”が完全に分離され、2 つのウィンドウでプログラミングできるようにすること。そして片方の変更がもう片方に反映されることだ」とソーリー氏は語る。「ソフトウェアでは 90%の作業が保守に割かれている。これまで構築したもの、これから構築するものを無駄にすることなく統合することこそ、業界の発展につながる」（ソーリー氏）。
MDA のメリットとしては、システム統合による投資の保護、あらゆるアプリケーションの統合が実現することなどがあることになる。

MDA は、すでに標準化されている UML、MOF（Meta-Object Facility）、XMI（XML Metadata Interchange）といった仕様でモデルからミドルウェアへのマッピングなど構成技術を持つと言うことです。

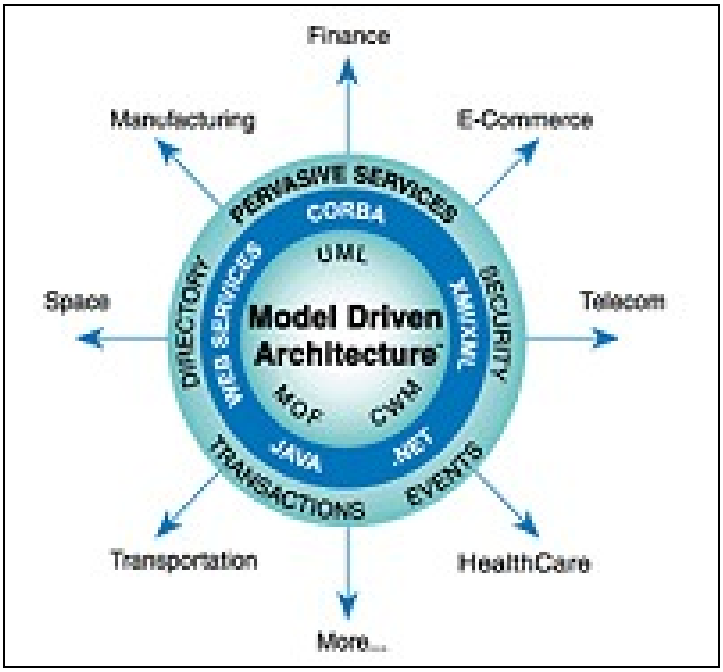


図 7-16　MDA の概観を表現した図

上図は OMG で紹介している MDA の概観を表現した図で、中心には，UML（Unified Modeling Language)，MOF（Meta Object Facility)，CWM（Common Warehouse Metamodel）といったモデルの表記技術があり、その外側には CORBA，JAVA，.NET などの実装技術，さらにその外側のセキュリテイ，イベントなどの共通サービスを経て，E-Commerce，Telecom，Healthcare などの各種ドメイン（業務分野）に向かう矢印が出ています。すなわち、実装技術によらないモデルがあり、それをモデル表記技術で記述することで、実装技術にマッピングして実行可能なシステムを作り上げることです。
では、どのような原理かというと、UML などのモデリング言語には，それ自身の構造を表現した「メタモデル」というものがあり、同じように EJB や CORBA などの実装技術にも，それ自身の構造を表現したメタモデルがある。そのモデル側と実装側, 2 つのメタモデルを対応させる標準化されたマッピングテーブルを用いて，モデルから実装へと落としていくのです。

MDA を用いてアプリケーションを構築する手順としては、まず PIM（Platform Independent Model）と呼ばれるプラットフォーム非依存のビジネスサイドのモデルを記述する。次に、この PIM を実装技術にあわせて自動変換した PSM（Platform Specific Model）を作成する。この間の自動変換ツールが「MDA ツール」と呼ばれている。UML を使って PIM を作成して、MDA ツールを使って PSM やソースコードを生成することになる。

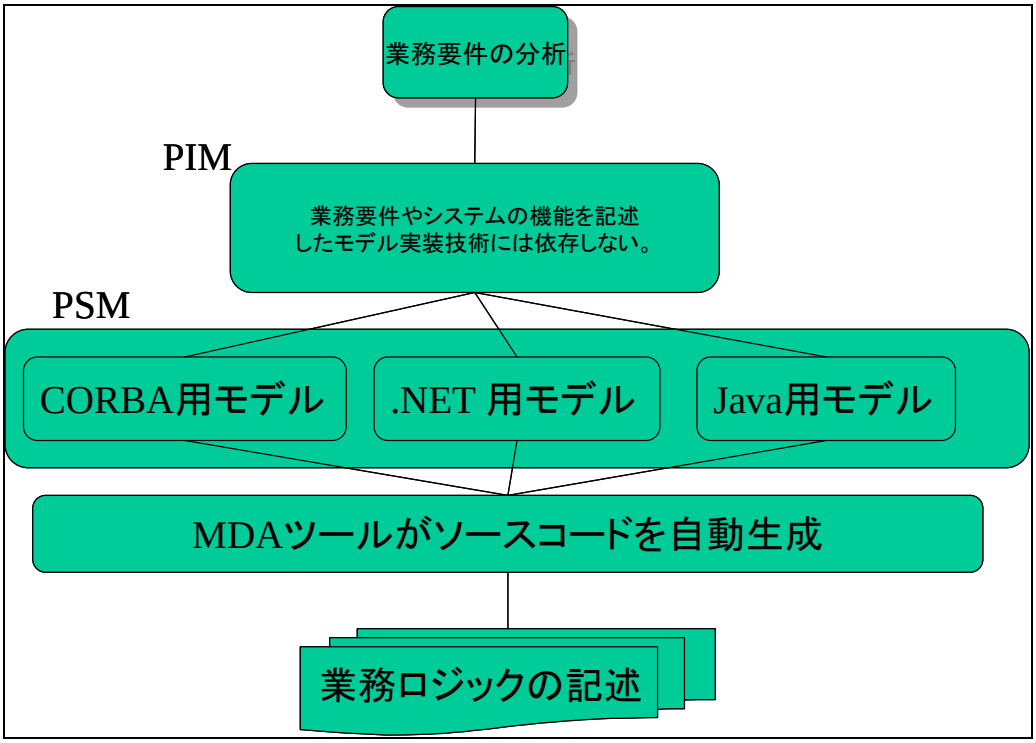


図 7-17　MDA を用いてアプリケーションを構築する流れ

MDA による開発を実際にサポートした技術が，OMG で策定されている。それが「UML Profile for EDOC」である。EDOC とは，「Enterprise Distributed Object Computing」の略で、企業レベルの分散オブジェクトコンピュータシステムのことです.

EDOC は，ビジネス・プロセスを中心に動作する。このビジネス・プロセスを中心とした，EDOC の要素には次の 4 つがある。

- ビジネスルール
- ビジネスイベント
- ビジネス工ンティテイ
- ビジネス・プロセス

ビジネス・プロセスは，ビジネスイベントによって起動される。どのビジネス・プロセスを起動するかは，ビジネスルールによって決定される。またビジネスイベントが発生すると，ビジネスエンティティは状態を変化することになる。

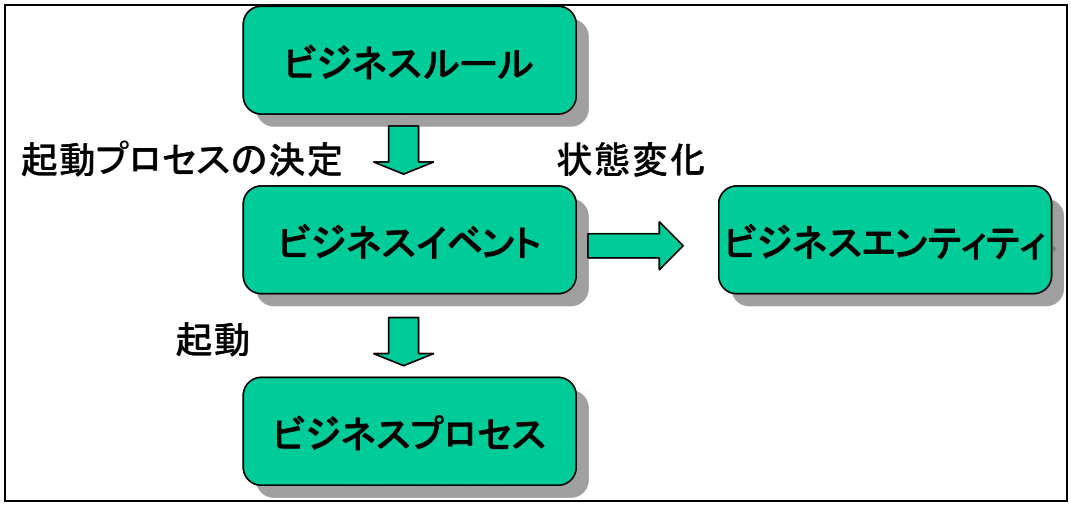


図 7-18　EDOC

この EDOC 環境をモデリングするために、OMG では UML の拡張を行っており、それが UML Profile for EDOC です。UML Profile for EDOC は、EDOC 環境の標準モデリング言語であり、今後エンタープライズシステムを EJB や CORBA、XML などの分散オブジェクト技術で開発する際の必須の技術になります。

7.3.8　フレームワーク

7.3.8.1　アプリケーションフレームワーク

ここでのフレームワークとは、ソフトウェアの基本的な枠組みが出来上がっており、一部を変更するだけで目的のアプリケーションが出来るソフトウェアを指す。これらをアプリケーションフレームワークと呼ぶ。これは、特定の業務に依存しないものと、特定業務向けのものがある。前者は、.NET Framework, J2EE Framework などがある。

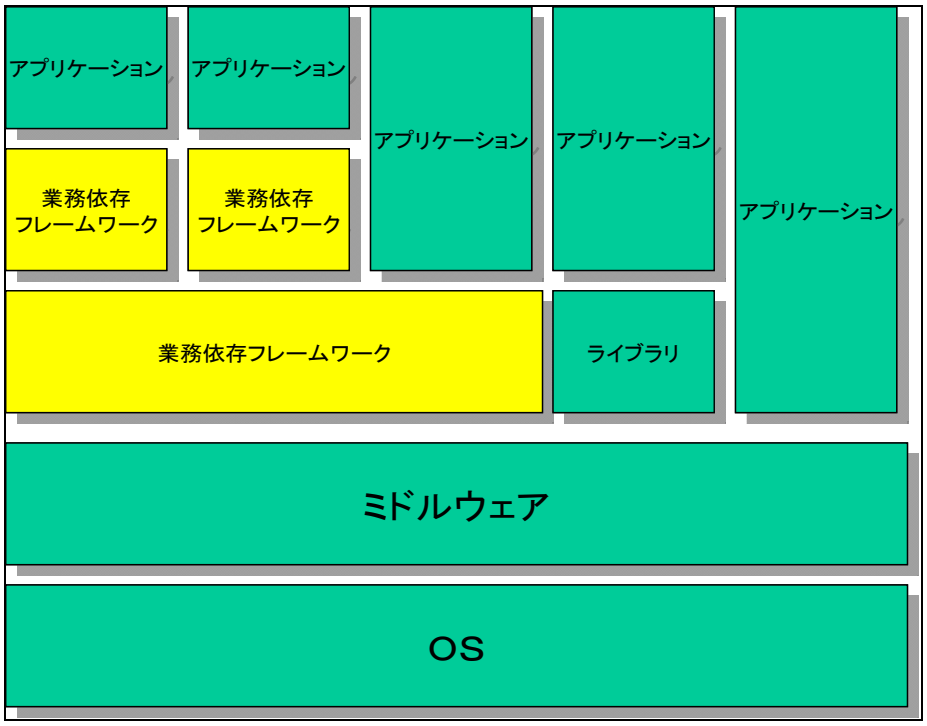


図 7-19　アプリケーションフレームワーク

すでにいくつかのフレームワーク製品が登場しており、これらフレームワーク製品を用いることで、より柔軟で再利用性の高いシステムを構築することが可能となっている。フレームワーク製品の一覧は、コンポーネントスクエアのホームページに詳細が掲載されている。http://www.c-sqq.com/

たとえば、MVC モデルを実現した、フレームワーク製品の概要は、以下のようになっています。

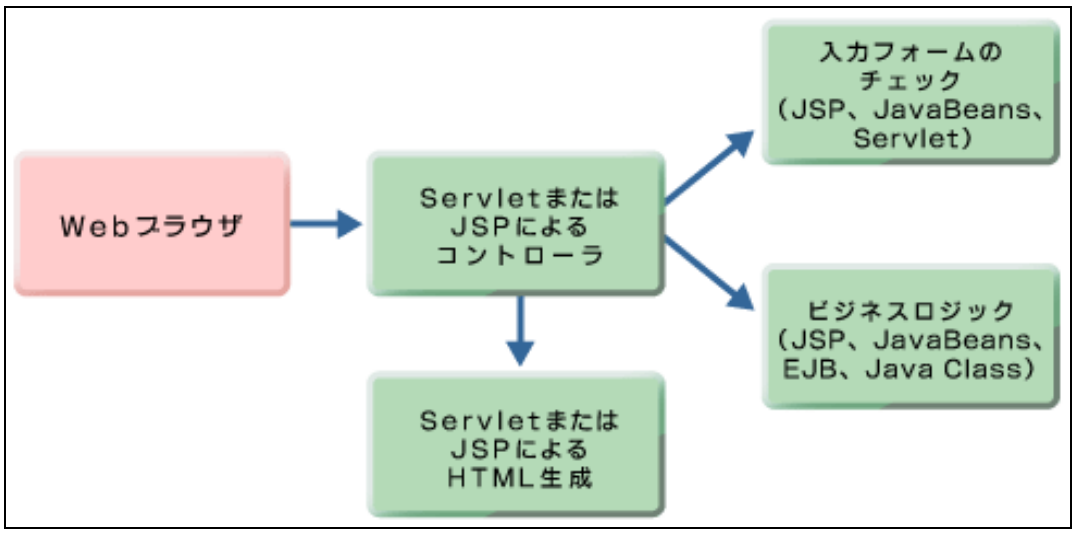


図 7-20　MVC モデルを実現したフレームワーク製品概要

具体的には以下のようになっている。

- 送信されてきたWEBから内容をリクエストとして、まず入力内容が正しいかをチェックするコンポーネントを呼び出すためのサーブレットとして提供されている。
- 入力チェックのコンポーネントでは、入力のタイプや書式などのチェックを行う。
- 入力が正しい場合ければ、ビジネスロジックを呼び出します。このビジネスロジックは、JAVABeans、EJBなどからなる。
- 最後にビジネスロジックから受け取ったデータを、JSP等でHTML化し、クライアント側へ返す。

フレームワーク製品を用いるメリットとして、以下のことが挙げられます。フレームワークを使うことで、サーバソフトの変化などをそこで吸収することで、その他他の業務に依存した部品を、より独立したものとして提供することが出来る。

Apacheの JAKARTAプロジェクトなどでは、Strutsなどがフレームワークとして無料で提供されている。

フレームワーク名	開発元	ライセンス
JAKARTA Struts	JAKARTA Project	無償
JAKARTA Turbine	JAKARTA Project	無償
cFramework	イーシワン	有償
ECOSS WAVE3 Framework	エコス	有償
INTRA-MART フレームワーク	NTT データイントラマート	有償
Logincletbox	シーマーク	有償

などがある。

8. まとめ

8.1 オブジェクト指向設計への期待と効果

「何に期待してオブジェクト指向設計を採用するのか？」は各企業の扱う商品の特徴､組織などによって期待している部分が異なることがわかった｡商品、組織を意識し、オブジェクト指向設計の持つさまざまな特性と期待感、効果について考察してみる。

8.1.1 X社の期待

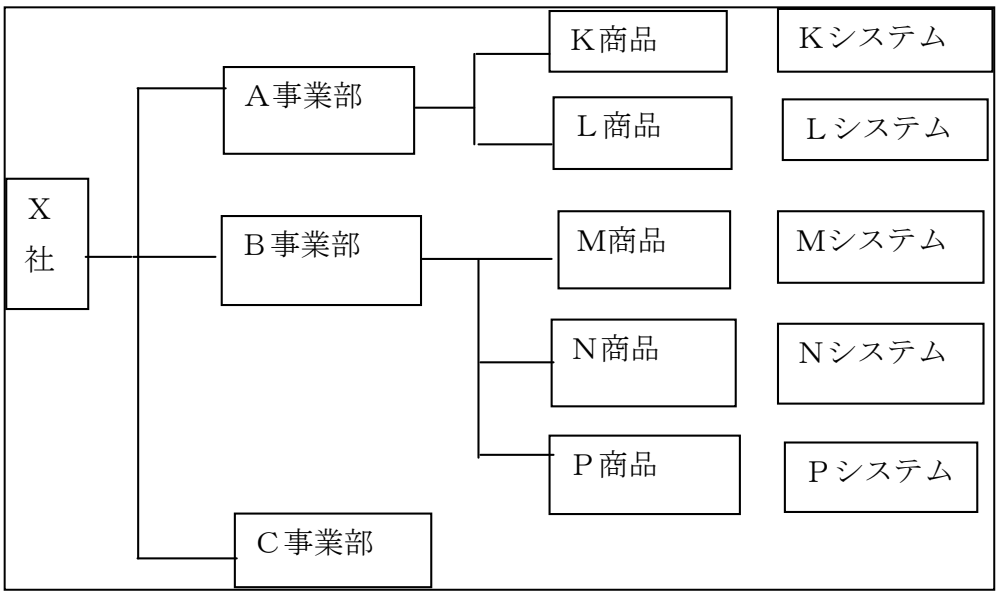


図 8-1 X社

X 社は急速な事業拡大を目指してA事業部のK商品システム、L商品システム、B事業部のM商品システム、N商品システムのように「事業部別商品システム」を個別開発してきた。
他の事業部、他の商品のシステムに惑わされること無く自商品の扱いができる点では、便利であったが各々別々のシステムを開発したための開発費、保守運用費は増加したばかりでなく、経理システムの業務統一や法規制への対応などの場合は、数多くのシステムを変更せねばならず、そのプログラム修正負荷やシステム間の調整負荷は大きな負担になってきた。

そこで CIO はこの問題を解決すべく次の案を考えた。

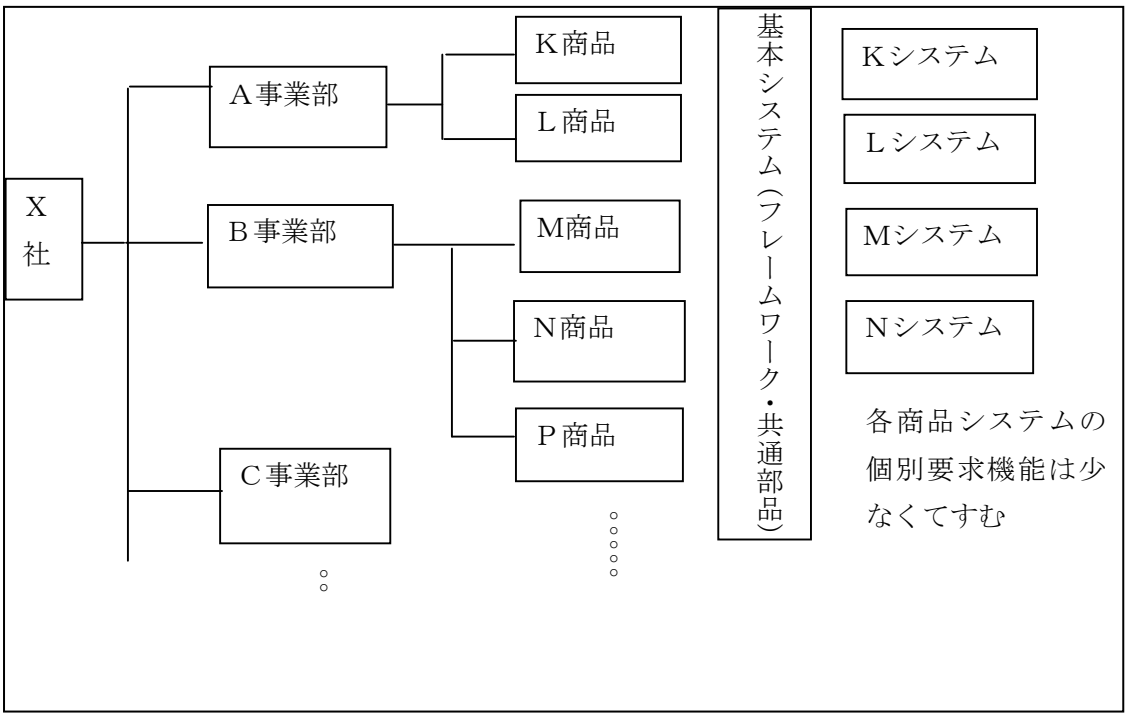


図 8-2　X 社展開

新しい商品や事業部ができるたびに、その都度システムを開発することをやめて EA(Enterprise Architecture)を元に基本機能システムとシステム部品を準備しそれを活用することにより開発負荷を軽減し、工期短縮を図ることを期待してオブジェクト指向設計を採用した。

もちろん旧来のシステムデザイン方法でもこのような設計は可能ではあるが、効率、使用性を配慮するとオブジェクト指向設計方法が有利である。

8.1.2Y 社の場合

Y 会社は、開発を専門にする企業で、学校事務システムの開発チームを作ってさまざまな学校の要求に対応しようとしている。

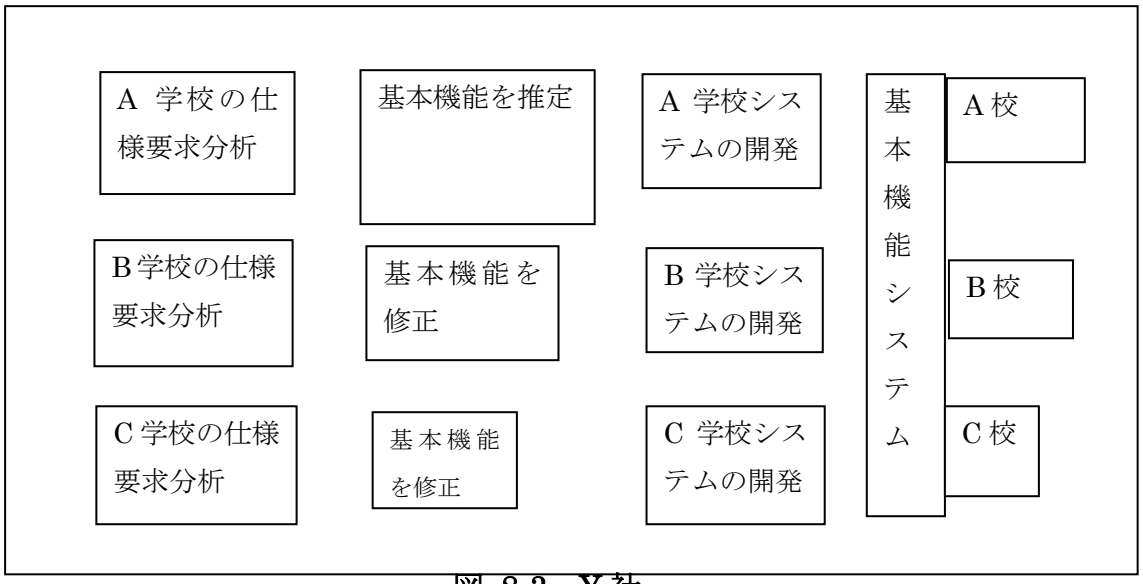


図 8-3　Y 社

まず、A 学校の仕様に合わせてシステム開発を行い、次に B 学校の仕様の追加を行い、内容を充実する。更に C 学校の仕様を少し追加する。このようにしてケースが増えていく事に従い整合性を持って機能追加が行われるので、ケースを重ねるたびにパッケージに近づく。

このように基本機能を修正しつつ、個別学校機能を追加することにより開発付加を減少し、品質を向上させるためにオブジェクト指向設計・開発を活用する。

ビジネスが好調な売れ行きを示し顧客が増えれば増えるほど、効果は大きくなる。

8.1.3　Z 社の場合

この Z 社は、金融商品を扱う企業であり、類似商品の販売を次から次へとタイムリーに出す必要に迫られている。起案から新商品発売にいたる期間はきわめて短くシステム開発期間は十分に取れない。

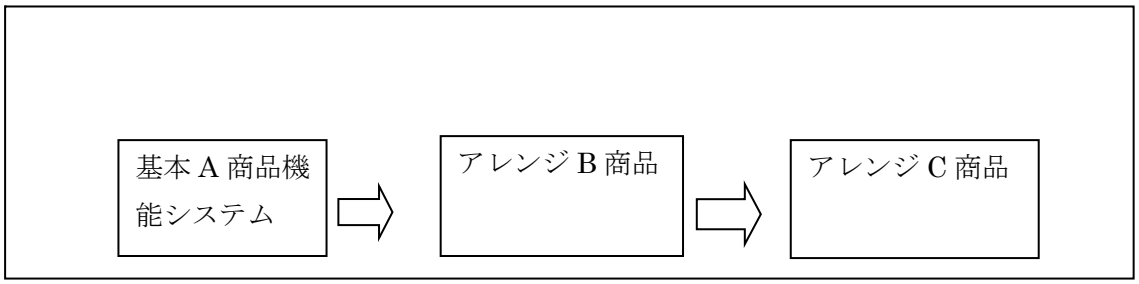


図 8-4　Z 社

A 商品を元に開発した基本機能システムを持っているので、B 商品の追加要求に合わせて機能を付加する。このようにしてC商品にも独特の機能付加を短期間で実施する。このようなケースにはオブジェクト指向設計開発の継承機能が有効である。

8.1.4　まとめ

以上述べてきたように、オブジェクト指向設計、さらにはビジネスオブジェクトについて具体的な効果を如何に生み出し利用するかについて事例中心に見てきた。

その中でも、この調査報告書で紹介した UML についてはさまざまな見解があることを補足しておきたい。

一例を挙げると、次のような意見である。

「他のシステム開発手法がたどったように、この UML もやがて消えてなくなるのではないか？」

「いや、他の手法とは奥の深さが異なる。徐々にではあっても急速に普及してゆくのではないのか？」

「UML を活用して効果が上がった事例を示せないと普及はおぼつかない」

「萬人に理解しやすいものにならないと普及しにくい」など多様な意見である。

私は、まだこの手法についての理解は未熟ではあるが、あれこれ批判するレベルまでに工夫されていない段階、使いこなされていない段階であると思っている。例えば、9 種類の UML ドキュメントは提示されているが、その作成分担者、作成順序、作成内容への工夫などがまだまだ発展段階である。

UML を設計局面において全面的に採用しようとしている企業も見つけにくかったし、今後活用が広まることは期待や予想をしながらも、全面的に採用することへの壁を破れていないことも現実である。まだ、さまざまな工夫や改良を試み、発展させてゆかねばならない有望な手法である。

JUAS でも、このような優秀なコンセプトを何とか発展させたいと考え、「CRC を実装段階ではなく、仕様決定段階で活用する」セミナを開催してみたところ非常に効果的であることがわかった。

さらに仕様の漏れをなくし、仕様を確実にユーザ自身が作成する手法を UML の中に盛り込むことを検討中である。このように使いこなす智恵を皆さんから集めて紹介したいものである。

今回この報告書を忙しい中でまとめていただいたベテランの方々が、「どうすればオブジェクト指向設計を判りやすくできるか？」と言う難題の解決への糸口を見つけてくれたことも収穫であった。

来期はこの手法の活用を広めるために「ユーザに理解しやすいオブジェクト指向設計ガイド」を作成することを検討したい。

なおオブジェクト指向設計を用いて設計されたシステムが、保守段階ではど

のような成果を出しているのか？その中でビジネスオブジェクトがどのような効果を発揮してゆくのか？ユーザがこれらの手法からメリットをうるためには、どのような戦略や対策が必要なのかを追求してみたい。

事例の X、Y、Z 社はオブジェクト指向設計の顕著な効果を期待し、取り組み始めた例であるが、この UML、JAVA の特性を根本的に生かしたシステム設計・開発・保守の採用をする企業が数多く誕生できることを願っている。

(社)日本情報システム・ユーザー協会
専務理事　細川泰秀

平成 14 年度　ビジネスオブジェクト検討プロジェクト

「ユーザが考えるビジネスオブジェクト調査報告書」

発行日：平成 15 年 4 月
発行所：社団法人　日本情報システム・ユーザー協会
〒103-0001　中央区日本橋小伝馬町 15-17 ASK 日本橋ビル 5 階
℡03-3249-4101　Fax03-5645-8493
URL http://www.juas.or.jp/

この事業は、オートレースの補助金を受けて実施したものです。