変化対応を考慮したシステム構築の考察

Consideration on system development technology responding to changes in the business environment

八　木　達　矢

要　約　近年，ビジネス環境の変化に素早く対応する柔軟なシステムを構築する要求が強まっている．ソフトウェア再利用アプローチを取り入れることで変化対応を考慮したシステム構築が可能になる．ソフトウェア再利用の歴史を振り返ると 6 つの類型を経て進化していることがわかる．本稿では，業務アプリケーション分野におけるソフトウェアの部品化と再利用アプローチの歴史的変遷に注目し，変化対応を考慮したシステム構築の今後の展望について考察する．

Abstract　Recent years, the needs have increased to develop information systems which have flexibility to be able to respond to the changes in the business environment quickly. Reuse of software module is one of the approaches in order to respond to these environmental changes. From the viewpoint of software reuse, the information system architecture has evolved through six steps of transformation. This paper introduces the history of approaches to componentalization and reuse of software, and considers the future of the system development technology responding to changes in the business environment.

1. は じ め に

ビジネスの情報システムへの依存度が高まる中，情報システムが果たす役割範囲が大きくなっている．ビジネスの世界ではビジネスモデルの変更やビジネスプロセスの変更が短周期で発生している．変化対象は，M&A による基幹系システムの統合といった大きな変化から，現場での作業手順変更といった小さな変化まで様々である．

これまでに構築され運用されてきた情報システムは，変更の都度個別にシステム開発・改修を繰り返してきた結果，アプリケーション構造が非常に複雑になり素早く変化対応できなくなっていることが多い．複雑性が増し改修が難しくなると全面的にアプリケーションの再構築を行うことになり，変化対応に多くの時間とコストをかけることになる．

今，情報システムに求められていることは，ビジネスの変化に柔軟に対応する変化対応力をつけることである．変化に素早く対応できるということは，変更によるシステム改修に要する時間が短期間であること，変更による影響が及ばないあるいは影響範囲が明確に分かること，変更によって品質が下がらないことが必要条件となる．変更に伴う実開発を出来るだけ減らし，部品単位の変更によって他部分への影響を抑え，高品質な部品を使用して品質を保障する「ソフトウェア再利用アプローチ」を取り入れることで，この条件を満たすことができる．

本稿では，システムの変化対応の必要条件である業務アプリケーション分野におけるソフトウェアの部品化と再利用アプローチの歴史的変遷に注目し，変化対応を考慮したシステム構築の今後の展望について考察する．

2. 部品化と再利用による変化対応

一度作成したソフトウェア資産を再利用したいという要求は1960年代から存在した．ソフトウェア再利用という言葉が国際会議で初めて使用されたのは，1968年に開催されたNATO国際会議でのMcIlroyの講演である[1]．McIlroyはこの中で“Use of off-the-shelf components as building block in new large system”という言葉を使っており，ソフトウェアを部品化しそれを組み立てることでシステム構築を行うコンセプトが提唱されている．以降，ソフトウェアエンジニアリングでは「ソフトウェアの再利用」アプローチが発展した．

ソフトウェアを部品化し再利用することは，品質と生産性の向上に有効である．これにより短期間でアプリケーションの構築が可能になり，変化に強いアプリケーション構造をとることが出来る．

ソフトウェア再利用アプローチは一度下火になるが，1983年Freemanの提唱により再燃する[2]．この中では“Use of non-executable work products for developing new software”とされており，実行可能なソフトウェア部品以外のものを対象とした再利用が提唱されている．これらは，再利用情報レベルとして定義されており，再利用対象は，コード，論理構造，機能構成，外部知識，環境情報などになる．

再利用のアプローチは再利用対象範囲を広げて発展してきているが，本稿では実行可能なソフトウェアの部品化と再利用アプローチに限定して，再利用対象の分類と考察を行う．

部品化と再利用による変化対応を考えるときに重要なことは，再利用技術を利用する際に重要視されるポイントが時代とともに変遷しており，テクノロジの進化とともに部品化と再利用の方法そのものが変化していることである．

業務アプリケーションの部品化と再利用による変化対応の歴史を振り返る際，著者は再利用の形態と境界という2つの軸で整理することが可能であり，整理した結果から今後の変化対応について予測できる，との仮説を立てた．ここで言う形態とは，アプリケーション部品の再利用方法である．また境界とは，アプリケーション部品を再利用することができる範囲のことである．この仮説に従ってアプリケーションの部品化と再利用の歴史的変遷を整理すると，図1に示す6つの類型に分類できる．また，それぞれの時代で再利用技術を利用する際に重要視されるポイントは図2に示すように変遷している．

形態は，Copyでの再利用とShareでの再利用を繰り返している．Copy形態とは一度作成されたアプリケーション部品を別のアプリケーションにコピーすることで組み入れて再利用することである．これに対してShare形態とは，ある目的のために作成したアプリケーション部品を共用化し，他のアプリケーションから再利用する形態である．どちらの形態でも一度作成したモジュールを再利用する．

境界は，アプリケーションアーキテクチャの進歩と，再利用技術を利用する際に重要視されるポイントに従って3つに分けられる．

最初の境界は，プログラム空間である．1970年代から1980年代が相当する．まず重複するコードを一箇所にまとめ，プログラム内から再利用することが可能になった．次に一度作成したプログラムをプログラム間で共用することが可能になった．この時代では機能中心のシステム構築が行われていた．機能中心であることから，再利用の際には手続きが重要視されていた．

次の境界は，メモリ空間である．1980年代後半が相当する．まずプログラムを実行時に組み合わせて再利用することが出来るようになった．次に異なるメモリ空間（異なるマシン）の

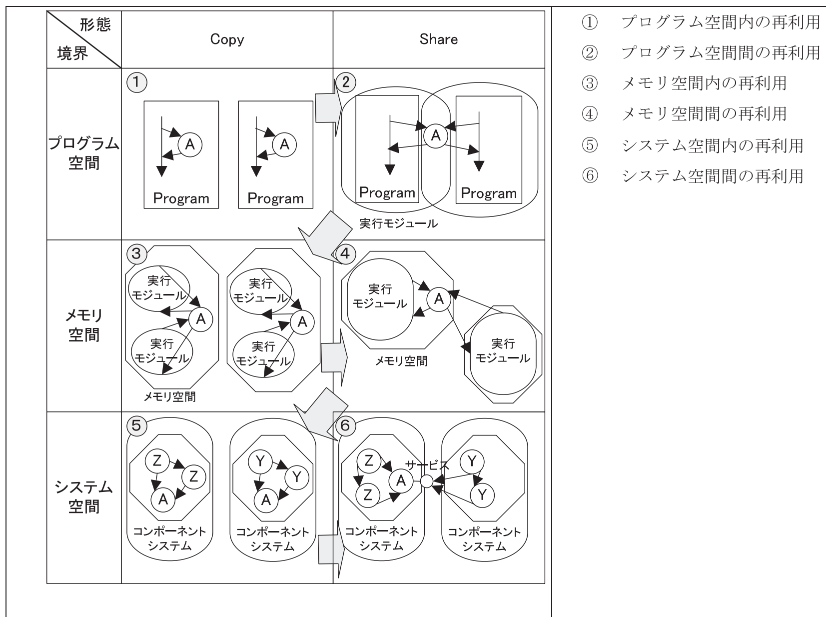

図 1　アプリケーションの部品化と再利用の変遷

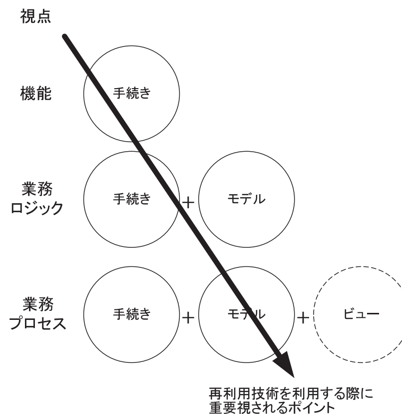

図 2　再利用技術を利用する際に重要視されるポイントの変遷

プログラムを共用して再利用することが出来るようになった．この時代では，機能の定義を中心としたシステム構築から,実世界のモデル化を中心としたシステム構築へシフトしつつあり，業務に対するモデリング手法が発達した．業務モデルはデータに依存する．そのためデータとデータに対する操作の集合をセットにして取り扱うデータ抽象化の概念が適用されるようになった．よって，再利用の際には手続きとデータを同時に取り扱うモデルが重要視されていた．

次の境界は，システム空間である．1990 年代後半から 2000 年代が相当する．まず再利用対象となるプログラムを独立アプリケーション化し，異なるシステムで再利用できるようになった．次に，別システムが提供するアプリケーションを呼び出して組み合わせることで，アプリケーションを共用化して再利用できるようになった．この時代では，業務ロジックや業務プロセスの再利用が始まり，異なるシステムで同様の業務ロジックや業務プロセスを再利用することが試みられている．まず業務ロジックの再利用が行われた．業務ロジックを独立したソフト

ウェア製品であるコンポーネントという形式で実装し，コンポーネントを再利用することが進められた．次に，コンポーネントを共有化し，共有したコンポーネントに対する呼び出しを組み合わせることで業務プロセスの再利用を実現することが試みられている．業務プロセスの共有を実現するため，業務ロジックを組み合わせて実行するビューが重要視されつつある．ここでいうビューとは,サービスを組み合わせて業務プロセスを実行するモジュールのことである．業務要求の変化を吸収し業務プロセスを実現するソフトウェアのことを指す．

技術の進歩に伴って変化対応方法そのものが変化している．それは改良の積み重ねによって実現されている．また，重要視するポイントは移り変わってきており，再利用の対象は拡大している．以下の節では類型ごとに，類型分けの基準としている再利用の境界と構造，積み重ねられた変化対応方法と再利用技術を利用する際に重要視されるポイントの移り変わりについて解説する．

2.1　プログラム空間内の再利用

同一のプログラム内で重複するロジックを一箇所にまとめ，それをプログラム内であればどこからでも呼び出すことが可能となった類型である（図 3）．COBOL の PERFORM 句などが該当する．

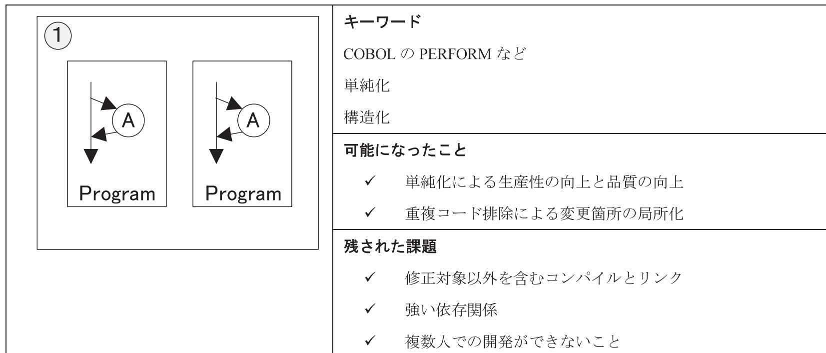

図 3　プログラム空間内の再利用

再利用が可能な境界はプログラム内に限定されており，別のプログラムで再利用したい場合にはソースコードをコピーして使用することが必要になる．呼び出し側と呼び出される側で相互に依存関係があり，変更が発生した場合には双方向に影響範囲を調査する必要がある．

この類型では,プログラムの再利用ではなくプログラムコードの再利用が焦点となっている．一度記述したプログラムコードを重複して記述しないことで，生産性の向上，品質の向上，保守性の向上を実現している．厳密には，プログラムコードの再利用はソフトウェアの再利用とはいえないが，再利用の試みはここから始まったため類型に加えた．

プログラム内でソースコードの再利用が可能となったことで，構造化プログラミング*1 が可能となった[3]．構造化プログラミングでは，トップダウン的に複雑なソフトウェアを分割し順次処理，分岐，繰り返しといった単純な構造に分割してプログラムを組み立てる．単純化された構造は，サブルーチンとしてプログラム内のどこからでも使用することが可能となる．単純化することで生産性と品質が向上する．また，重複コードを排除することで，変更が発生した

ときに同様のコードを複数箇所変更する必要がなくなった．課題としては，修正の都度プログラム全体をコンパイル・リンクする必要があること，呼び出す側と呼び出される側に強い依存関係があり，修正が発生する都度影響調査を双方で行う必要があること，基本的にプログラムの分割が出来ないため，プログラムサイズが大きくなること，プログラムコードはプログラマが単独で作成する必要があること，などが挙げられる．

なお，単純な構造に分割することで，規模が大きくてそのままでは内容を把握することが難しいシステムであっても内容把握が容易になるという考え方は，システム分析・設計に応用された．これは構造化アプローチとしてソフトウェアエンジニアリングの中で体系立てられ，後に大きな影響を残した．

構造化アプローチでは，データの流れからデータに対する手続きを機能として定義することを試みている．このため，機能が重要視され，再利用の対象は手続きが中心となる．

2.2　プログラム空間間の再利用

共通的に使用できる部分をプログラムとして独立させ，複数のプログラムから呼び出して再利用が可能となった類型である（図 4）．COBOL のサブルーチンなどが該当する．

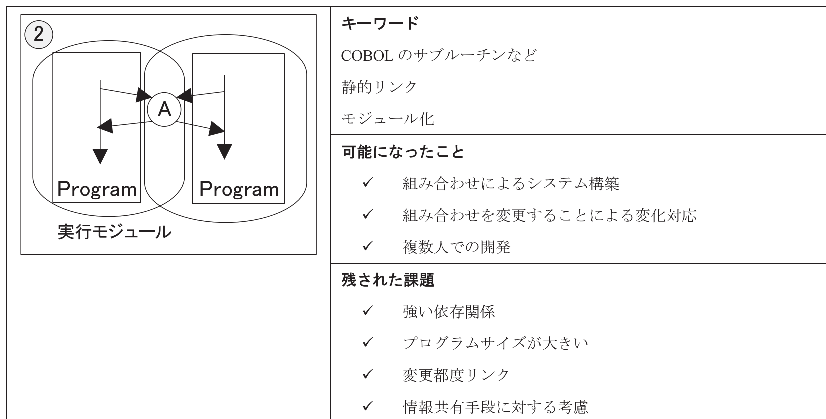

図 4　プログラム空間間の再利用

異なるプログラム間で同一ロジックを使用したい場合であってもソースコードのコピーを行う必要はなくなった．実行モジュールを作成する際には，静的リンクによって共通部分のプログラムとそれを使用するプログラムを結びつけ一つの実行モジュールを作成する．実行基盤からプログラムを見た場合には，プログラムは一つであるように見える．プログラム間で再利用する場合でもプログラムをコピーする必要はなく，静的リンクさえすればよい．

開発時には共有するプログラムを独立したプログラムモジュールとして開発することが可能となった．アプリケーションは複数の部品に分割され，それを組み合わせて一つのソフトウェアにすることが可能となる．これによって組み合わせによるシステムの変化対応が可能になった．モジュールは独立した単位で開発が可能であるため，複数人での開発が進むようになった．一方，変更の都度コンパイルの必要はなくなったが，静的リンク作業を実施する必要は残っている．また，情報共有手段に関する考慮を新たに行う必要が出てきた．

構造化アプローチは機能仕様を作成することを中心としているが，このころから機能は変更

が発生しやすく，変更に対応するためには業務のモデル化を行った上で機能を詳細化する必要があると考え始められた．JSD 法[4][5]は実世界のモデル化を中心にモデリング作業を進めていくシステム開発方法である．この中では「実世界のモデルは開発のための安定した基盤」であり，「モデルはその上に組み立てられる機能よりも変更されることが少ない」からだと紹介されている．この考え方は，後に続くモデル中心のアプローチにつながっていく．

2.3　メモリ空間内の再利用

再利用対象モジュールは独立したプログラムとして作成され，メモリ空間にロードされる．再利用対象モジュールを使用するモジュールも同一のメモリ空間にロードされており，実行時にリンクが張られる類型である（図 5）．

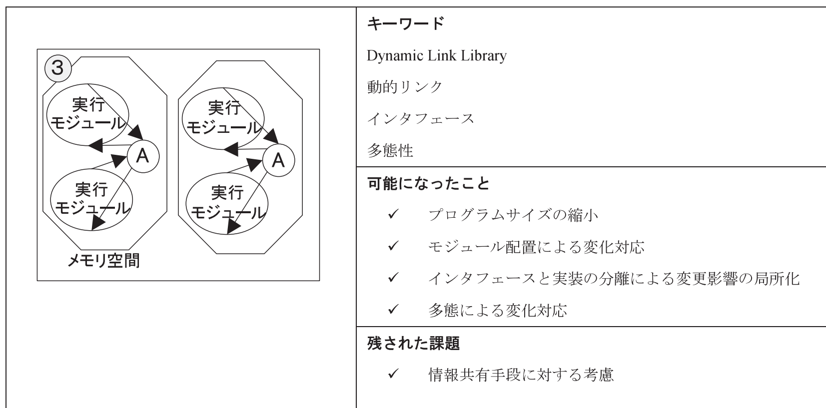

図 5　メモリ空間内の再利用

実行時に動的リンクをするため，プログラム変更時には全体の再コンパイル・静的リンクなしに再利用対象モジュールを入れ替えることが可能になる．また，実行時にリンクすることでプログラムサイズを縮小化することが出来た．

変更が発生した場合には，変更対象部分のみ再コンパイルし再ロードするだけですむようになる．開発時や実行モジュールの作成時にはインタフェースのみがあればよい．モジュール間は直接の依存関係を持たずメッセージをやり取りすることで連携する．インタフェースが同じであれば実行前に実装を入れ替えることで変化対応することが出来る．これは多態性の一種である．多態性とは，同じインタフェースを使用したプログラムが実行時の条件によって異なる動作をすることである．

この類型では呼び出す側から呼び出される側への一方的な依存関係は存在するが，インタフェースと実装を分離することにより，実装部分に変更が生じた場合でも使用する側は影響を受けなくなった．

このころ，機能の定義を中心としたシステム構築から，実世界のモデル化を中心としたシステム構築へシフトしつつあり，業務に対するモデリング手法が発達した．業務はデータに依存するため，構造化手法のように手続きとデータを分けて考えることが出来なくなる．手続きとデータ構造は一緒に取り扱う必要があり，データ抽象化の概念が必要となった．このため，データのライフサイクルをプログラムで考慮する必要がでてきた．ここからオブジェクト指向ア

プローチが発達する．

オブジェクト指向は 1980 年代以降に本格的に利用されるようになった．オブジェクト指向では，抽象データ型，多態性，継承の三つの概念でシステムを表現する．継承は元になるモジュールを再利用して，部分の変更や追加を行うことを可能とした考え方である．

オブジェクト指向における抽象データ型はクラスと呼ばれる．オブジェクト指向ではクラス間のメッセージのやり取りでシステムを動作させ，モジュール間を結合する．インタフェースに対する依存はあるが，実装とインスタンスに対する依存がない．このためインタフェースの変更を伴わない程度の変化であれば，実装のみ変更することで変化対応することが出来る構造になっている．

モデル中心のアプローチのため，再利用の際に重要視されるポイントはモデルになる．また，再利用の単位はクラスになる．手続きのみの再利用と比べて，クラスでは現実世界に近い単位で再利用が可能となる．再利用粒度は，手続きのみの再利用と比べて大きくなり，より業務アプリケーション寄りの範囲での再利用が可能となる．

2.4　メモリ空間間の再利用

呼び出し対象となるプログラムが，呼び出し側プログラムと同じメモリ空間内にあっても別のメモリ空間にあっても区別なく呼び出すことが可能な類型である．CORBA に代表される分散オブジェクトが該当する（図 6）．

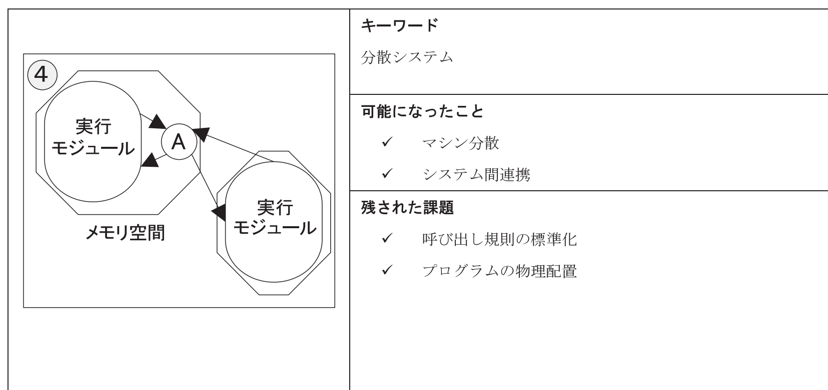

図 6　メモリ空間間の再利用

使用する側の実行モジュールは自メモリ空間に呼び出し先がある場合でも，他メモリ空間に呼び出し先がある場合でも区別なく呼び出し結果を受け取ることが出来る．他メモリ空間に呼び出し先が存在する場合には通信を行うことで呼び出しをかける．通信による依存関係しか存在しないため，呼び出し先の呼び出し規則が変更されなければ呼び出す側はプログラム変更の必要はない．

プログラムは呼び出し先で実行される．呼び出し側の業務ロジック内では，メモリ内のモジュールであっても，メモリ外のモジュールであっても呼び出し方式は変わらないため，開発者は実行モジュールの配置場所を気にすることなく使用できる．

分散オブジェクト技術では，データと手続きをオブジェクトにカプセル化するとともに，オ

ブジェクトの配置場所もカプセル化して隠蔽する．この技術によって，ユーザは実装の詳細やネットワーク上の位置の詳細から自由にオブジェクトに仕事を依頼するという形で情報システムを構築することが可能になる．

分散環境に適応させるため，呼び出し規則の標準化が重要な課題として存在する．呼び出し規則は単純で標準化されていなければならない．また，分散環境下でプログラムを物理配置する場所の考慮が重要な課題となる．

分散環境では，トランザクションや実行基盤となるミドルウェアの制約条件に従って事実上の再利用粒度が決まった．

2.5　システム空間内の再利用

再利用の対象が API のレベルから，独立したコンポーネントと呼ぶソフトウェア部品へと拡大された．コンポーネントはある業務ロジックを実行する機能をもった独立したソフトウェア部品であり，コンポーネントの組み合わせによってアプリケーション構築を行う類型である（図 7）．

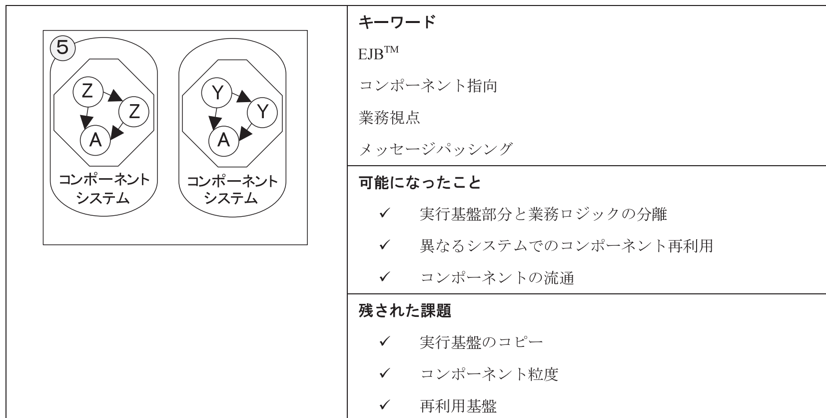

図 7　システム空間内の再利用

コンポーネントは再利用を前提としたソフトウェア部品である．コンポーネントを使用する際には，コンポーネントをコピーして自システムに取り込むことになる．変更が生じた場合には，コンポーネントの組み合わせを変更することで変化対応することが出来る．

コンポーネントを作成する際には，システム制御部分を実行基盤に委譲し，業務ロジック部分のみに注目すればよくなった．また，規定の実行基盤上であればどのようなシステムでも動作することが義務付けられるため，異なるシステムでもコンポーネントをそのまま再利用することが出来る．コンポーネントを市場に流通させることも試みられている．

一方，システムは規定の実行基盤上でコンポーネントを使用する必要があり，実行基盤の制約がシステムの制約となる．コンポーネントを提供する側では，再利用可能なコンポーネントの粒度が決めにくいという課題がある．コンポーネントを使用する立場とコンポーネントを提供する立場をつなぐ再利用基盤を整備するという課題が発生する．

独立したソフトウェア部品を組み立てることでシステム構築を行うため，再利用の際にはコンポーネントを組み立て，それを使用するビューが重要なポイントに加わってくる．

2.6　システム空間間の再利用

コンポーネント実行基盤を自前で持たず，他の実行基盤上のモジュールを組み合わせることでシステムを実現する．システム外から使用されるコンポーネントはサービスとして公開される．サービスはサービスを提供する側のサービス基盤上で実行される．2000 年代後半から本格化すると予想される類型である（図 8）．

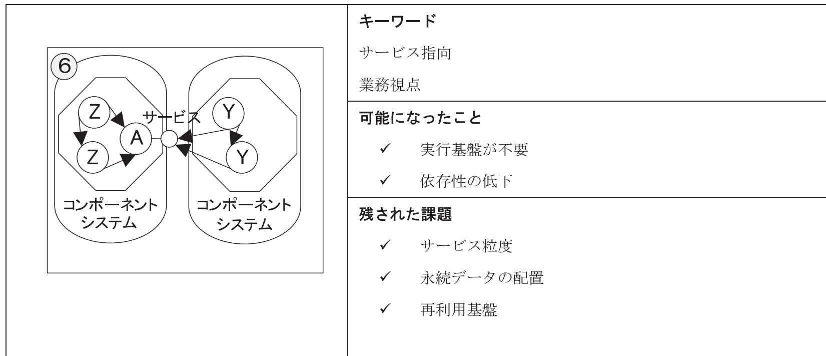

図 8　システム空間間の再利用

サービス指向では，業務プロセスとソフトウェアをマッチングする単位としてサービスという概念を持ち出している．サービスの組み合わせによってアプリケーションの結合を実現するため，変更の必要が生じた場合には該当するサービスを交換することで変化対応することが出来る．

サービスを使用する立場に立った場合，サービス基盤は用意する必要はなく，サービス基盤にアクセスできれば良い．サービスに対するアクセス方法は標準化されている．サービスを提供する立場に立った場合，サービス基盤を提供する必要がある．

標準化されたサービス呼び出しによる連係によってアプリケーションを構築するため，これまでの類型と比較して依存性が低く抑えられている．

業務アプリケーションの部品化と再利用はここまで進んできたが，現在残っている課題も存在する．課題とは，サービス粒度，永続データの配置，追跡性，再利用基盤である．

- サービス粒度

  提供するサービスの粒度が問題となる．一般的に，再利用可能な粒度が大きくなれば再利用価値が高くなるといわれている．一方で変化対応を考慮した場合，小さな粒度にした方がモジュール単体の再利用性は高まる．サービスごとにどの粒度で提供するのが良いかを考慮する必要がある．

- 永続データの配置

  サービス実行にはデータが必要となる．サービス固有の永続データはサービス内に隠蔽される．一方，共通的に使用されるような永続データについては，どこに配置するのか，どのように受け渡すのかが問題となる．

- 再利用基盤

  システム間でサービス情報を共有する仕組みが必要である．単独システム内では統一したルールに従ってサービスを公開することが難しくないが，組織間で再利用する場合には

情報を共有することが難しい．

3.　今後の展望

アプリケーションの部品化と再利用による変化対応の歴史から，Copy 形態と Share 形態が繰り返されていることが分かった．また，再利用は 6 つの類型に分類できることが分かった．それぞれの累計ごとに変化対応方法は改良されており，それは過去の再利用技術の積み重ねによって実現されている．再利用の類型の移り変わりとともに，再利用技術を利用する際に重要視されるポイントも移り変わってきている．ここで示したサイクルは今後についても繰り返されると考えられる．このサイクルに沿って今後の展望を示すことで，変化対応方法の準備をしていきたい．

図 1 では，形態の遷移がサイクルしており，サイクルごとに再利用の対象範囲は広がっていくことが分かった．次のサイクルでの再利用対象はサービス空間だと考えている．現在のサービス空間は，企業内，あるいは業界内など限られた空間で提供されるが，再利用基盤の進化により更に広い範囲に対してサービス提供すると予想している（図 9）．

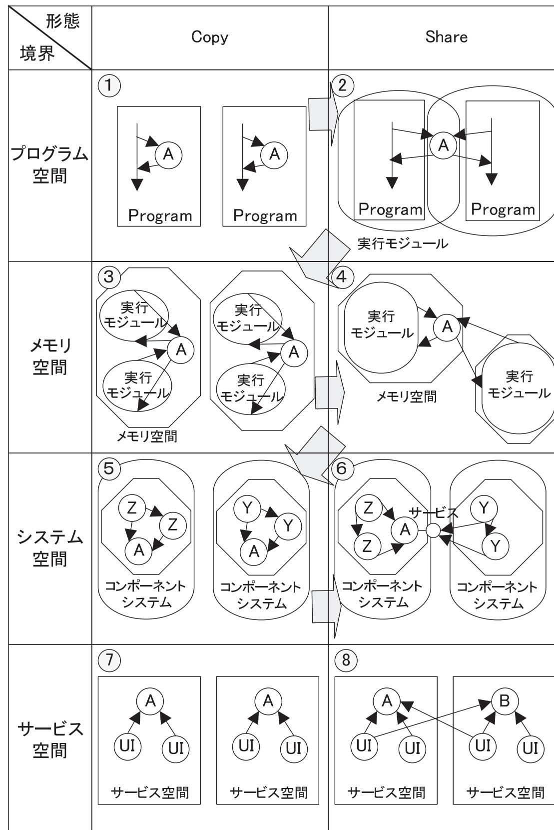

図 9　再利用形態サイクルと再利用対象の展望

図 2 では，過去から現在にかけてシステム構築において再利用技術を利用する際に重要視されるポイントの遍歴を示したが，ポイントが「手続き」から「手続き＋モデル」の視点を経て今後は「手続き＋モデル＋ビュー」に遷移していくと考えている（図 10）．

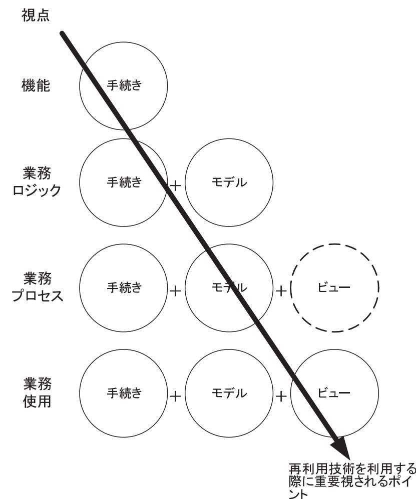

図 10　再利用技術を利用する際に重要視されるポイントの変遷展望

以上より，今後の「サービス空間の再利用」に対する対応と「ビューの多様化」に対する対応をとる必要があると予想している．

3.1　サービス空間の再利用

サービス空間の再利用とは，あるサービス空間で提供するサービスをコピーして別のサービス空間に組み入れる，あるいはサービス空間を共用化して他のサービス空間から使用することである．サービスを使用する立場で考えると，サービス空間の拡大と複雑化ということになる．

アプリケーションを構築するために組み合わせるサービスの選択範囲は広くなり，手動で最適なサービス探索を行うことは時間・コスト的な制限から難しくなることが予想される．サービス探索の自動化が期待され，サービス再利用基盤の役割が重要になってくる．

再利用基盤でサービス探索の自動化を行うためには，サービス空間内外で「意味」の統一をはかる必要がある．意味が統一されていなければ目的と似て非なるサービスが探索されてしまうことになるだろう．よって，サービス探索の自動化に対する準備として，意味を事前に定義し公開することが必要になる．

意味を定義し公開するための有望な技術としてオントロジーがある．オントロジーとは，「人間が対象世界をどのように見ているかという根源的な問題意識を持って物事をその成り立ちから解きあかし，コンピュータと人間が理解を共有できるように書き記したもの」と定義付けられる．オントロジーを用いることによって，「意味」を厳密に表記することが可能である[6][7]．

オントロジーなどのテクノロジによって，対象領域の意味が定義されれば，サービスの組み合わせを自動化する道が拓ける．予めオントロジーによって対象領域の意味定義が実施していれば，サービス空間上に存在するサービスからシステム構築に必要なサービスを抽出し，最適な組み合わせを自動的に選択することが可能になる．理想的には，システム要求とそのシステムのゴールを定義するだけで，ゴールを実現するために必要なサービスを探索し，サービスの組み合わせを自動化するような仕組みを作成することが可能になる．

ただし，オントロジー作成は非常にコストと時間がかかる作業であり，長時間をかけて業界内で定義していく必要がある．今のうちからオントロジーによる「意味」定義を進める必要が

あると思われる．

3.2　ビューの多様化

今後は，重要視されるポイントが「手続き＋モデル＋ビュー」に遷移していくと予想している．これは．ビジネスプロセスの変更サイクルがますます短周期化し，これに追随してシステム変更を行う必要があるためである．また，ビジネスプロセスが多様化し，同一企業内で部署ごとにビジネスプロセスが微妙に異なるケースも考えられる．このような多様性を吸収し，柔軟なシステムを構築するためには，サービスを組み合わせるビューも多様化させる必要がある．

コスト・時間的な制約から情報システム部門が多様なビュー全てを提供することは難しいので，現場のシステム担当者が自らサービスを組み合わせて使用する時代が到来するのではないかと思われる．情報システム部門は不変性の高いシステムや多くの人が共通的に利用するサービスについて予め基盤サービスとして提供し，基盤サービスのみでは足りない部分や個人的にしか使用しないような部分に関しては現場部署で専用のサービスを作成する．そのような一種の EUC（End User Computing）形態が発生すると予想している．

EUC 化を可能とするためには，サービスの組み替えが容易にできる必要がある．実現する手段としては，目的としているサービスを自動的に見つけてくるエージェントや，プログラミングレスでサービスの組み立てが可能なツールが考えられる．また，サービスの組み換えでは実現できないようなある程度の複雑性を持った業務プロセスについては，スクリプティング言語による補完が有望な技術になる．一般的にスクリプティング言語はコンパイラ言語で作成するプログラムよりも実現できる機能は少ないが，習得が容易である．近年では，スクリプティング言語の機能が大幅にあがり，コンパイラ言語並みの機能実装が実現できるようになってきた．

EUC によって作成されるビューはユーザのクライアントマシン，あるいは部門サーバ上に物理配置されると考えられる．

4.　お わ り に

1968 年当時にソフトウェアエンジニアリングは提唱された．当時からソフトウェアに対する生産性向上，品質性向上，再利用性の向上の要求は存在しており，ソフトウェアエンジニアリングは時代の要求に応えるべく発展を遂げてきた．ソフトウェアの再利用の概念とモジュール化はソフトウェアエンジニアリングが扱う大きなテーマである．

アプリケーションの部品化と再利用による変化対応の歴史を振り返ると，6 つの類型に分類できることが分かった．それぞれの類型で工夫されてきた再利用の技術があり，それを積み重ねることで現在の再利用技術は成り立っている．

過去の類型から次のサイクルを予想した場合，サービス空間の再利用とビューの多様化があると思われる．これは実現に時間のかかる作業であり，準備は今のうちから進めておく必要がある．準備にはアプリケーションのユーザ，サービスを提供するベンダ，サービスを組み合わせる SI ベンダなど業界全体で準備作業に取り組む必要がある．日本ユニシスはメインフレーム時代から現在まで蓄積してきたノウハウを持っており，そのノウハウを活用してソフトウェア再利用技術のさらなる発展に貢献できれば幸いである．

＊１　構造化プログラミングを提唱した Edsger W. Dijkstra は米国バロース社（後の米国 Unisys 社）でフェローとして従事し，構造化手法について大きな影響を与えた．

参考文献

[1]　McIlroy M. D.,“ Mass produced software components ”, In Proceedings of Report on a Software Engineering Conference Sponsored by the NATO Science Committee, NATO Science Committee, 1968.
[2]　P. Freeman. Reusable Software Engineering : Concepts and Research Directions. Proceedings Of the Workshop on Reusability In Programming 1983.
[3]　E.W. ダイクストラ，C.A.R. ホーア，O.－J. ダール共著；野下浩平，川合慧，武市正人共訳，「構造化プログラミング」，サイエンス社，1975.5
[4]　M. Jackson/大野徇郎・山崎利治監訳「システム開発 JSD 法」，共立出版，1989
[5]　妻木俊彦,“ オブジェクト指向のための文献紹介 ”, UNISYS TECHNOLOGY REVIEW 第 60 号，1999.2
[6]　Deborah L. McGuinness and Frank van Harmelen，上綱秀治訳，“ OWL Web Ontology Language Overview ”, W 3 C Recommendation 10 2004.2
[7]　溝口理一郎，「オントロジー工学」，オーム社，2005.1

執筆者紹介

八　木　達　矢　（Tatsuya Yagi）
1998 年大阪工業大学大学院博士前期課程修了（経営工学）．同年日本ユニシス(株)入社．Java を用いた WEB アプリケーションの開発および，アプリケーションフレームワーク製品「LUCINA Web Foundation」の技術主管業務および適用業務に従事．現在，IT ソリューション部サービスビジネスイニシアティブに所属．

All Rights Reserved. Copyright® 2006 Nihon Unisys, Ltd.