企業におけるコンポーネント型開発の実践と課題

じょうご　たくみ

城後　　匠

［略　歴］

2001 年　サントリー株式会社入社

現在　　情報システム事業部　情報化推進部

システム経験年数　3 年

原稿量

本文　　12,016 字

要約　　1,199 字

図表　　15 枚

＜要　約＞

めまぐるしく変化する昨今の経営環境において「ビジネスの変化に柔軟に対応すること」は、情報システムに求められる最重要課題である。当社においても、酒類販売免許自由化に代表される流通形態の変化や事業の統合及び分社化、競合他社との厳しい競争など、ビジネスを取り巻く環境は急激に変化しており、情報システムには、開発生産性を更に高め、変化に強く高品質なサービスを素早く低コストで提供することが求められている。

当社では、開発生産性向上と変化に強いシステム構築への解決策として、２０００年にサーバーサイドでのＪＡＶＡ技術であるＪ２ＥＥを基盤としたコンポーネント型開発を導入した。以来、オブジェクト指向による分析設計の徹底、分析設計の共通言語としてのＵＭＬの導入、フレームワークの採用、システム部品の整備といった土台作りを積極的に行ってきた。そしてコンポーネント型開発は、２００２年末から始まった全社基幹システム再構築プロジェクトにおいて本格的に始動することとなった。各開発プロジェクトから切り出されたコンポーネントを平行開発して大量に蓄積し、別の開発プロジェクトで再利用していくことで、全社のアプリケーション全体の生産性向上を図るとともに、徹底したコンポーネント化によって、変化に強いシステムを目指してきた。コンポーネント型開発を実践していく上で、行った方策を以下に示す。

（１）コンポーネントアーキテクチャーの策定

開発するコンポーネントのアーキテクチャーと、それを利用するシステムのアーキテクチャーを明確にして標準化を行った。

（２）コンポーネント開発体制と開発プロセスの整備

再利用性を追求するコンポーネントの開発はコストがかかるため、開発プロジェクトの中で実施していくことは非常に難しい。そこで、全社アプリケーション全体としての生産性向上をコンポーネント型開発で目指す意志を持ったチームとして、コンポーネントの開発、運用、標準化の専任チームを組織した。またより多くのコンポーネントを生み出すため、コンポーネントの開発プロセスを確立した。

（３）コンポーネントの蓄積と情報の公開による再利用の促進

使いやすいコンポーネントを目指して公開情報を整備するとともに、ポータルサイト「コンポーネントビレッジ」を立ち上げ、コンポーネントに関するすべての情報をダウンロードできる環境を構築し、再利用の促進を行った。

以上の取り組みの結果、２００３末の時点で約９０個近くのコンポーネントを開発し、のべ３００個以上のコンポーネントをミッションクリティカルな基幹システムに適用することに成功した。２００５年には、コンポーネントを再利用することで、開発プロジェクトにおける生産性を４０％向上させることを目標としている。今までの活動をベースに、今後より多くのコンポーネントを創出し、効率よく再利用していくことで、目標は達成できると確信している。

目　次

1. はじめに

当社を取り巻く環境は、流通形態の変化や、事業の統合及び分社化、競合他社との厳しい競争など急激に変化しており経営にもスピードが求められている。また情報システムには、開発生産性を更に高め、変化に強く高品質なサービスを素早く低コストで提供することが求められている。

ソフトフェアを部品化し、その部品を用いて開発するいわゆる「部品化再利用」の概念は、生産性向上の手段として以前から提唱され続けてきた。近年、サーバーサイドのＪＡＶA技術であるＪ２ＥＥ（Java 2 Enterprise Edition）と、ＪＡＶAの部品化技術であるＥＪＢ（Enterprise Java Beans）の登場で、部品化再利用は大きく脚光を浴びている。

当社では、部品化再利用（コンポーネント化とその再利用）による生産性向上と、変化に強いシステム構築を現実のものとするべく、２０００年よりＪ２ＥＥ／ＥＪBベースでのコンポーネント型開発を導入してきた。そして、システム部品だけではなく、社内のデータや業務をコンポーネントとして開発し、各開発プロジェクトの中で再利用することで全社アプリケーション全体としての生産性を向上させ、同時に変化に強いアプリケーションを構築する取り組みを実施してきた。本論文では、当社のコンポーネント型開発への取り組みについて紹介していく。

2. コンポーネント型開発の背景と目的

2.1. コンポーネント型開発にいたる背景

当社では、１９９０年代中頃より、第４世代言語「ＮOMAＤ」を活用し、開発手法にはＲＡＤ（Rapid Application Development）を前面採用することで、ＣＯＢＯＬと比較して生産性を４～５倍に高めることに成功した。

しかし、ビジネスが激しく変化していく中、経営からは今まで以上のスピードで、且つ低コストでシステムを構築することが求められるようになってきており、従来のＮOMAＤとＲＡＤをベースとしたシステム開発では、これ以上の生産性向上は見込めない状態であった。

そこで当社では、サーバーサイドのＪＡＶA技術であるＪ２ＥＥ／ＥＪBをベースとしたシステム開発に着目し、ＮOMAＤ／ＲＡＤに変わる新たな開発手法として、コンポーネント型開発を導入したのである。Ｊ２ＥＥ／ＥＪBによるコンポーネント型開発を採用した理由を以下に示す。

（１）ＷＥBとの親和性とオブジェクト指向

ＷＥBとの相性がよく、継承、カプセル化が使えるオブジェクト指向言語であること。

（２）デファクトスタンダード、豊富な製品群

事実上のデファクトスタンダートであり、世界中に多くの開発者が存在し、アプリケーションサーバーをはじめとする多様なミドルウェア製品が整備されていること。

（３）独立性、移植性の高さ

高度な部品化仕様を持ち、部品の独立性、移植性が高く、分散オブジェクトとして実装されていること。

（４）コンポーネントの半製品市場の存在
　コンポーネントを調達可能な市場が存在していること。

以上の理由から、２０００年にＪ２ＥＥ／ＥＪＢによるコンポーネント型開発を導入し、基盤整備を行ってきた。そして２００２年末から始まった、物流、経理、マーケティング、流通といった全社基幹システムの１０００人月以上に及ぶ再構築プロジェクトの中で、コンポーネント型開発を全面的に導入したのである。

2.2. コンポーネント型開発の目的

今回コンポーネント型開発を実践していくことで、以下に示す目標を達成することを目的とした。
（１）アプリケーション開発の生産性向上
　全社的にコンポーネントを整備蓄積し、再利用して開発を行うことで開発プロジェクトでの開発生産性を４０%向上させる。
（２）品質の向上
　コンポーネントを利用することでコーディング量の削減し、品質保証されているコンポーネントを組み込むことで品質を向上させる。
（３）変化に強いアプリケーションの構築
　アプリケーションにコンポーネントを組み込むことで、仕様の変化に柔軟に対応する。業務やデータの変化はコンポーネントの層で吸収し、アプリケーションへの影響を最小限にするアーキテクチャーを確立する。
（４）運用効率の向上
　コンポーネント化によってプログラム資源を一元化し、運用効率を向上させる。

2.3. コンポーネント型開発の目指す姿

コンポーネント型開発によって生産性を高めていくには、まず再利用可能なコンポーネントを蓄積することが必要になる。業務やコストの視点から、当社では市場からのコンポーネントの調達を検討外とし、社内でコンポーネントを開発することにした。しかし、コンポーネントの開発と蓄積は、短期間で実現することはできない。そこで図１のように、比較的長いスパンで、コンポーネント型開発の計画と目標を策定し目指す姿を明確化した。（図１）２００４年以降に、開発生産性を４０%以上向上させることが目標である。その実施ステップについて以下に示す。

（１）コンポーネント型開発に向けての基盤整備（～２００２以前）
　２０００年のＪ２ＥＥ導入以降、オブジェクト指向による分析設計を取り入れ、分析設計の共通言語としてUMLを採用し、開発者へ教育を実施してきた、またログや文字変換やメールなどのシステムレベルのユーティリティクラスを整備し、Ｊ２ＥＥベースのフレームワークを導入することで、コンポーネント型開発を行う基盤の整備を実施した。

（２）コンポーネント開発開始（２００２年）

コンポーネントアーキテクチャーの策定を行い、全社共通で利用されるデータや手続きをコンポーネントとして開発し、一部のアプリケーションに適用した。

（３）コンポーネント大量創出（２００３年）

全社基幹システム再構築プロジェクトの中で、切り出されたコンポーネントを大量に平行開発し、蓄積した。また開発したコンポーネントの別の開発プロジェクトでの再利用促進を行った。

（４）コンポーネントの普及と再利用による生産性向上（２００４年～）

コンポーネントの開発量も増え、蓄積したコンポーネントの再利用が本格化することで、開発プロジェクトでの開発生産性の４０%向上を目指す。

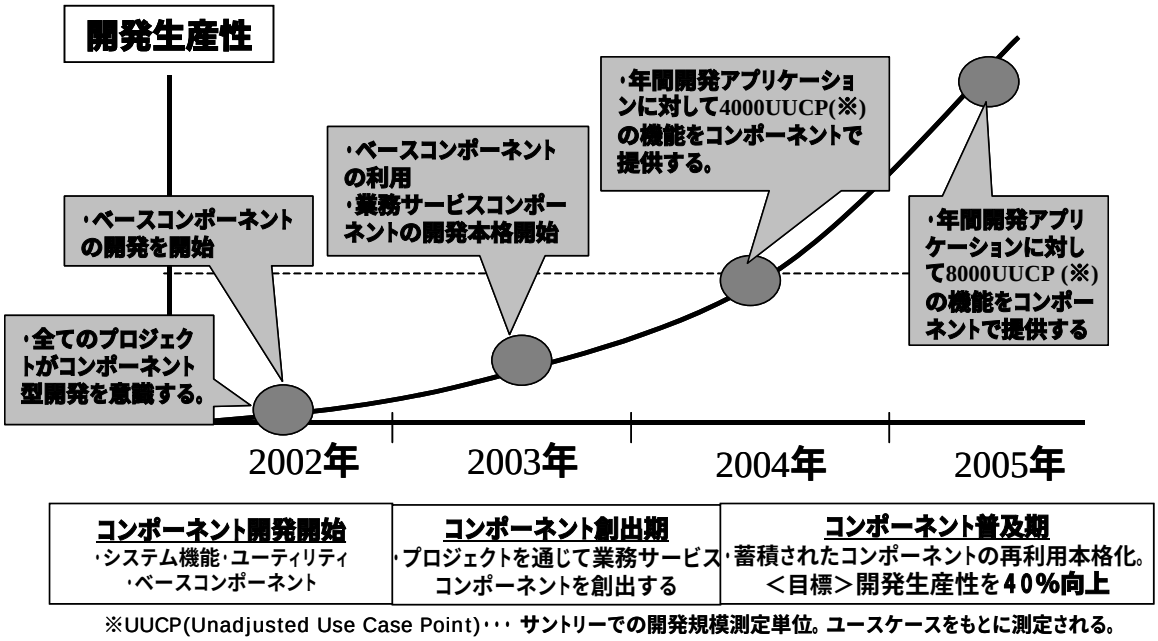

図１．コンポーネント型開発の目指す姿と実施ステップ

3. コンポーネントの分類とアーキテクチャー

3.1. コンポーネントの分類

今回開発対象となるのは、社内の業務やデータをカプセル化したインテリジェントなコンポーネントである。そしてコンポーネントを開発する上で、大きさ（粒度）は再利用の単位でもあり重要な意味を持ってくる。そこで今回は、コンポーネントを、粒度が小さくマスターの情報や単機能のサービスを提供する「ベースコンポーネント」と、比較的粒度が大きくビジネスユースケースの単位で開発され、複数のコンポーネントやロジックでよって業務機能を提供する「業務サービスコンポーネント」の２つに分類して開発を行った。（表１）ベースコンポーネントの例としては、人組織や相手先といったマスターデータを扱うものや、認証などの単機能を実現するものが挙げられる。業務サービスコンポーネントは、予算管理や支払など比較的粒度が大きく、ユースケースに該当す

るようなものが挙げられる。また、以前から進めてきたユーティリティクラスの整備や共通システムのパッケージ化も平行して実施した。

表１.コンポーネントの分類

分類	概要	例
ベース コンポーネント	単一のデータモデルの操作するマスター参照系と、単一の機能を提供する機能系の2種類に分かれ、比較的粒度の小さなサービスを提供する。	・人組織の情報を扱うコンポーネント ・相手先の情報を扱うコンポーネント ・承認を行うコンポーネント
業務サービス コンポーネント	業務機能単位のサービスを提供するコンポーネント。ビジネスユースケースごとに作成され、業務機能を提供する。	・予実管理を行う業務サービス ・請求書を作成する業務サービス ・販売管理を行う業務サービス

3.2. コンポーネントのアーキテクチャー

ベースコンポーネントは、単一のＥＪＢで実装され、ＪＡＶＡのデータベースアクセス機能であるＪＤＢＣ（Java Database Connectivity）経由でデータの操作を行う。コンポーネント型開発の目的に１つである変化に強いアーキテクチャーを構築するため、データやロジックの変化に柔軟に対応する４層からなるアーキテクチャーを採用した。（図２）各層の役割を以下に示す。

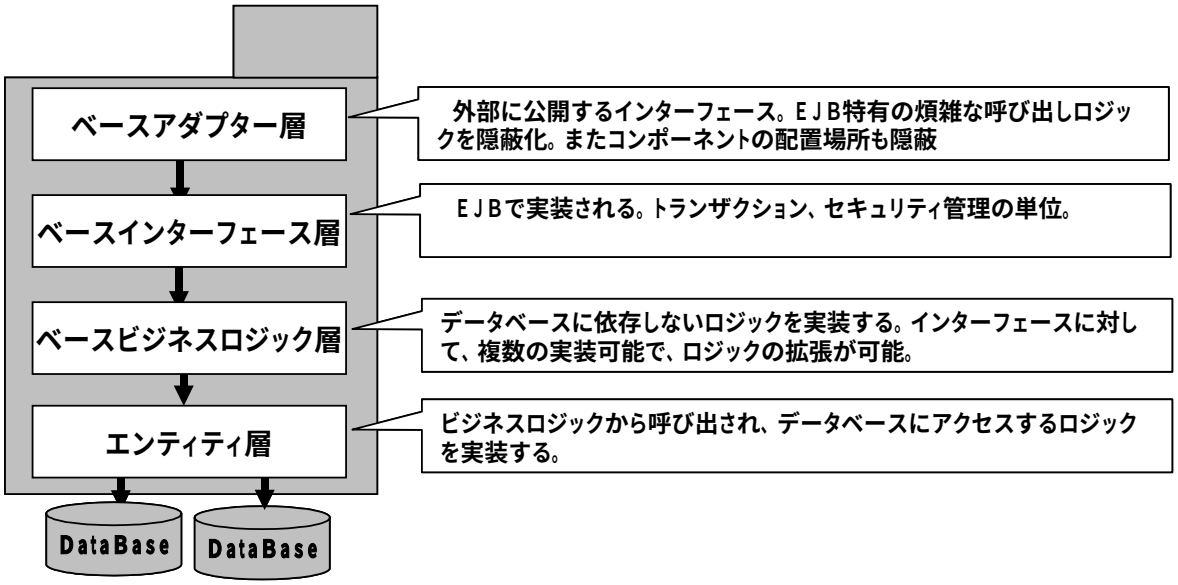

図２.ベースコンポーネントのアーキテクチャー

（1）ベースアダプター層

アプリケーションに公開するインターフェースを定義する。ＥＪＢ呼び出しの煩雑な手続きと、コンポーネントの配置場所を隠蔽化する。クライアントは通常のクラスと同じ方法で、コンポーネントの操作が可能になる。

（2）ベースインターフェース層

ＥＪＢで実装されベースアダプター層から呼び出される。トランザクションや、セキュリティは管理の単位。パラメータにより、ビジネスロジックの振り分けを行う。

（3）ベースビジネスロジック層

エンティティ層から取得したデータを使ってのデータチェック、計算、シーケンスコントロールなどのデータベースに依存しないロジックを実装する。ロジックの付け替えが可能。

（４）エンティティ層

データベースに依存したアクセスロジックを実装する。ＪＤＢＣを利用した最適なＳＱＬ文を発行し、データを操作する。

業務サービスコンポーネントは、複数のコンポーネントと、個別のデータベースやロジックで構成される。内部の詳細なアーキテクチャーは定義せず、インターフェースがＥＪＢで実装されていることと、再利用可能なインターフェースを定義することのみを必須条件として開発を行った。

3.3. アプリケーションアーキテクチャーの中でのコンポーネントの位置付け

コンポーネントを利用して開発するアプリケーションのアーキテクチャーを図３に示す。アプリケーションはすべてＪ２ＥＥをベースに導入したフレームワーク上で開発される。クライアントは基本的にブラウザで、WEB 層の受付とシーケンスコントロールはサーブレットが行う。ここまでは、導入したフレームワークで大部分を用意している。フレームワークに依存しない WEB 層から呼びされる業務ロジックを、業務サービスコンポーネントとベースコンポーネントの組み合わせによって実現し、各コンポーネントは、他システムから再利用、共有されるという位置付けである。他システムからの利用をじゅうぶんに考慮して汎用的なコンポーネントインターフェースを設計することが、変化に強いアプリケーションとして開発するための重要なポイントである。

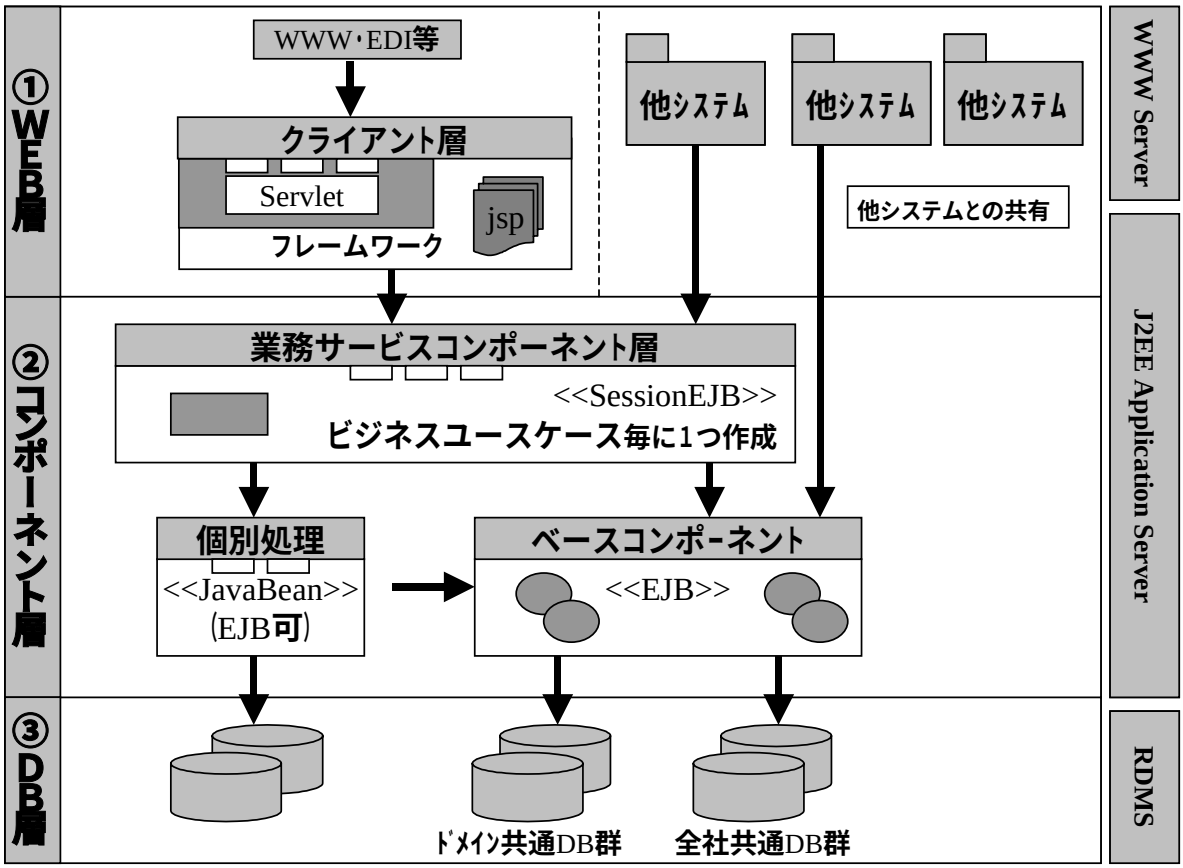

図３.アプリケーションアーキテクチャーとコンポーネントの位置付け

4. コンポーネントの開発

4.1. 開発体制の確立

今回の取り組みは、大規模な基幹システム再構築に伴って複数のアプリケーション開発プロジェクトが進んでいる中で、その開発プロジェクトから切り出されたコンポーネントを蓄積し、各開発プロジェクト同士もしくは、今後の開発プロジェクトで再利用することを目的としている。つまり、開発プロジェクトと平行して、コンポーネントを開発し再利用を促進していかなくてはならないのである。

その解決策として、各開発プロジェクトとは別にコンポーネントの開発、運用に特化したコンポーネント専任チーム（以後コンポーネントチーム）を組織した。各開発プロジェクトでのコンポーネント化支援と、切り出されたコンポーネントの開発が主な役割である。（図４）

コンポーネントは再利用されることや変化を吸収することを考慮して開発する必要があるため、必然的に通常の開発に比べコストが高くなる。よって各開発プロジェクト内で、大量にコンポーネントを開発することは非常に難しい。そこでコンポーネントチームという各開発プロジェクトに対し、横断的に活動する組織を作ることで、各開発プロジェクトから切り出されたベースコンポーネントを大量生産することが可能となった。ただし、より粒度の大きい業務サービスコンポーネントについては、アプリケーションの粒度に近いので、各開発プロジェクトでの開発を原則とし、コン

ポーネントチームは業務サービスコンポーネントの開発支援を行う体制とした。

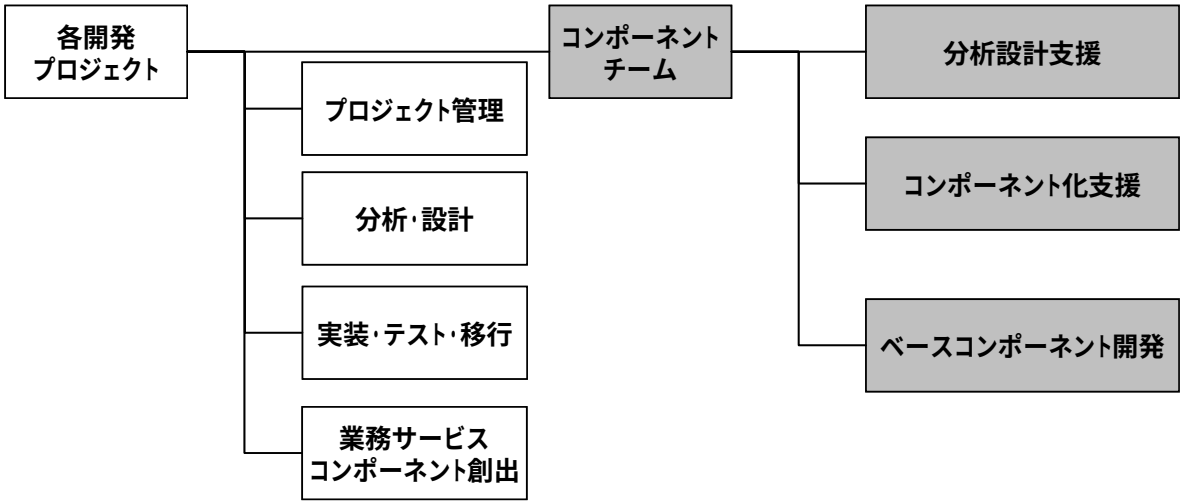

図４.コンポーネント型開発の役割分担

4.2. コンポーネント開発プロセス

コンポーネントチームを設置することで、各開発プロジェクトからベースコンポーネントを開発するためにかかる負荷は取り除かれた。ただし開発プロジェクトと平行して開発を行うため、各開発プロジェクトと、コンポーネントチームとの間の円滑なコミュニケーションが必要となる。開発プロジェクトから見た場合に、他のチームにある部分の開発を任せた時点で、納期に間に合わないなどのリスクを背負うことになるからである。

そこで、そのリスクを少しでも減らすために、コンポーネント開発における役割分担と開発プロセスを明確に定義することにした。（図５）各工程での成果物を定義し、成果物に対する役割を明確にすることで、円滑にコンポーネント型開発を進めていくことが目的である。定義した開発プロセスは以下のとおりである。

（１）要求分析

開発プロジェクトで開発の対象となる業務やデータの中から、コンポーネント化できる部分を切り出し、ウィッシュリスト（コンポーネント化要望書）を作成する。コンポーネントチームはウィッシュリストの内容を精査し、コンポーネントとして開発するべきかを開発プロジェクトと協議し、要求を洗練させた上で、開発計画を立てる。

（２）設計

コンポーネントチームで、コンポーネントのインターフェースの設計を行う。提供するサービスが決定した時点で、インターフェース概要を定義して開発プロジェクトとレビューをおこない仕様を確定させる。

（３）実装

コンポーネントチームが、コンポーネントのプログラム仕様書を作成する。開発プロジェクト

はプログラム仕様をもとにアプリケーションに組み込んで実装を行う。

（４）組み込み・テスト

コンポーネントチームが実装したテスト前のコンポーネントを一度引き渡し、アプリケーションで結合テストを行う。コンポーネントチームは引き続きテストを行い、品質を上げていく。

（５）共有・公開

開発したコンポーネントを、ポータルサイトに登録し公開を行う。

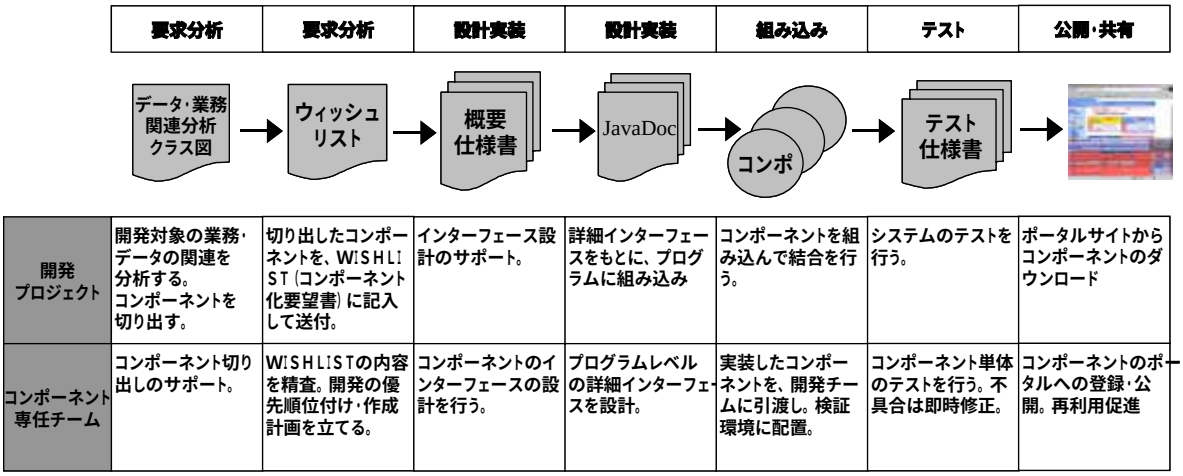

図５.コンポ―ネント開発プロセス

コンポーネント開発体制の整備と、開発プロセスを定義することにより、役割分担が明確になり、複数の開発プロジェクトと平行してコンポーネント納期どおりに開発し、蓄積していくことが可能となった。

4.3. コンポーネント型開発における情報公開の重要性

コンポーネントとコンポーネントを利用するアプリケーションのアーキテクチャーを策定し、開発体制及び開発プロセスを整備していくことで、開発プロジェクトから大量のコンポーネントを生み出す仕組みを構築してきた。これは基幹システムの再構築のタイミングで、全社的に複数の開発プロジェクトが立ち上がっている中で、質の高いコンポーネントを大量に切り出して開発し、蓄積することに主眼を置いている。しかしコンポーネント型開発が本来目的としているものは、蓄積されたコンポーネントの再利用による生産性の向上である。ただコンポーネントを蓄積するのではなく、開発したコンポーネントの再利用を促進していかなくてはならない。

コンポーネント自体の使いやすさは、アーキテクチャーの中には既に盛り込んである。そこでコンポーネントを利用するためのマニュアルにあたる情報の整備と公開が重要にある。仕様も使い方もわからないコンポーネントは再利用できない。そして再利用できないコンポーネントは、生産性の向上にも貢献することができないのである。

そこで再利用を促進するために、コンポーネントの公開情報の整備と、コンポーネント公開サイ

トの構築による情報の公開を行った。

4.4. コンポーネント公開情報の整備

コンポーネントの再利用を促進するには、とにかく使いやすいコンポーネントを作って、組み込むために必要となる業務を減らし、コストを下げる必要がある。コンポーネントの仕様がどんなに使いやすく作られていても、その仕様を的確に説明するドキュメントがなければ、導入コストがかかり、結果として生産性向上に寄与することはできない。

今回は、コンポーネントの仕様、利用マニュアル、利用ツールの３つの観点から公開情報の整備を行った。公開に必要な情報として定義したものについて以下に示す。（表２）

（１）機能概要

コンポーネントの概要を説明したもの。コンポーネント名、バージョン、概要、適用分野、稼動環境、サービスレベル、連絡先などの利用する際のカタログに当たる情報

（２）インターフェース情報

コンポーネントが持つメソッドの一覧とその説明を記述したサービス仕様書と、プログラムレベルでのインターフェースであるコンポーネントのＪＡＶＡＤＯＣを公開。

（３）データベース設計情報

データベースを必要とするコンポーネントについては、対象となるデータベースのレイアウトや、関連図を公開。

（４）テスト仕様書

単体テストにおけるテストケースと確認事項をまとめたドキュメント。パフォーマンステストを行った結果についても同様に公開。

（５）コンポーネント利用ガイド

コンポーネント全体で一つ提供。コンポーネントを利用するために必要なライブラリや設定を定義。開発環境を整備するためのガイド。

（６）インストールガイド

コンポーネントを開発環境に組み込む手順を定義したドキュメント。

（７）データベース構築ツール

コンポーネントが利用しているデータベーステーブルを生成し、サンプルデータをロードするツールを作成して公開。

（８）サンプルプログラム

コンポーネントの各サービスを呼び出すサンプルプログラム。開発環境にインストールした際の稼動確認用に公開。

仕様	マニュアル	ツール
機能概要	コンポーネント利用ガイド	データーベース構築ツール
インターフェース情報	インストールガイド	サンプルプログラム
データベース設計情報		
テスト仕様書		

表２.コンポーネント公開仕様

上記の公開情報を整備していくことで、コンポーネントはより使いやすくなり、導入にかかるコストも大幅に削減することができた。そして、再利用するハードルが下がったことで、更に活発にコンポーネントの再利用が行われることになったのである。

4.5. コンポーネント公開サイト「ＣＯＭＰＯＮＥＮＴ　ＶＩＬＬＡＧＥ」

整備した公開情報を共有し、コンポーネントの再利用をより円滑にするため、２００２年に、コンポーネント公開サイト「ＣＯＭＰＯＮＥＮＴ　ＶＩＬＬＡＧＥ」を立ち上げた。開発したすべてのコンポーネントは、カタログ形式でこのサイトに公開されている。（図６）公開されたコンポーネントについては、公開情報もコンポーネント本体もすべてダウンロード可能な状態で登録されており、ブラウザから直接コンポーネントの動きを確認できるサンプルアプリケーションも公開している。そして、実際の開発者の方々にダイレクトに使っていただくため、エクストラネット上にサイトを構築し、社外の協力会社の方々からも利用可能な環境とした。

またコミュニケーションツールとしても活用している。メーリングリストを開設して、コンポーネント利用者に対してバグ情報やバージョンアップなどの情報を発信し、逆に利用者からは質問を受け付け、よくある質問についてはＦＡＱ形式でサイトに掲載している。

「ＣＯＭＰＯＮＥＮＴ　ＶＩＬＬＡＧＥ」は、単なる公開サイト、ダウンロードサイトではなく、コンポーネント型開発におけるサントリーのナレッジを集結したポータルサイトとして機能している。コンポーネントの開発から再利用まで、コンポーネント型開発におけるあらゆるシーンで情報のハブとして活用されている。

図６.コンポーネント公開サイト「ＣＯＭＰＯＮＥＮＴ　ＶＩＬＬＡＧＥ」

5. コンポーネント型開発の成果

5.1. コンポーネントの蓄積

以上の取り組みの結果、コンポーネントの数は、２００２年末の２２個から、２００３年末には約４倍の９１個まで蓄積することができた。（図７）２００３年に開発したコンポーネントの内訳は、ベースコンポーネントが５５個、業務サービスコンポーネントが１０個、共通システムをパッケージ化したものが４個で、これらのコンポーネントはすべて「ＣＯＭＰＯＮＥＮＴ　ＶＩＬＬＡＧＥ」に公開した。特に粒度の小さいベースコンポーネントについては、２００３年の開発ステップ数が約６０万ステップにものぼり、会社内のほぼすべてのマスターデータをカバーしている。大量にコンポーネントを開発し、蓄積するという目的はじゅうぶんに達成することができた。

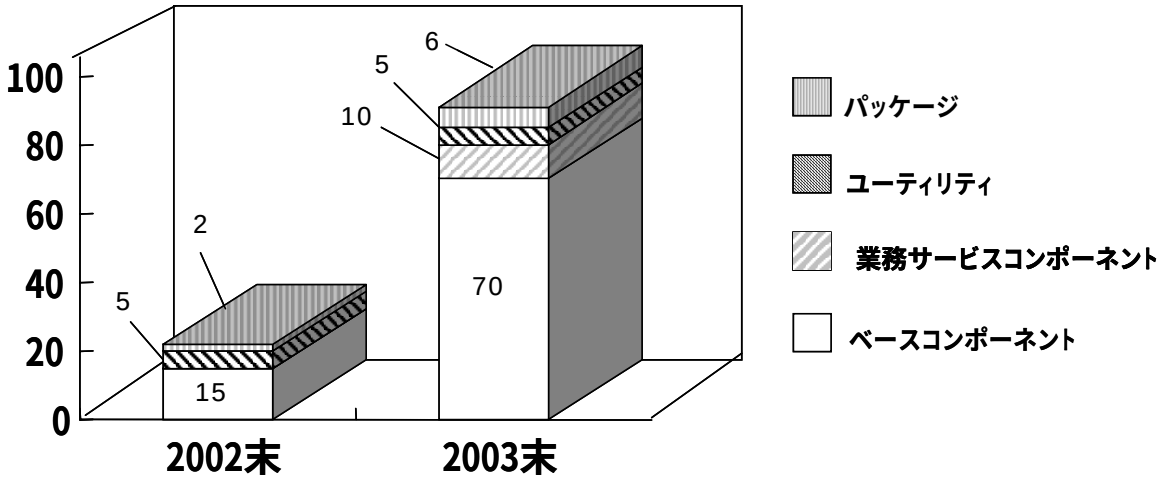

図７.コンポーネント開発数実績

5.2. コンポーネントの適用と効果

２００４年１月の時点で、５２個のアプリケーションが、のべ約３００個以上のコンポーネントを利用しており、トランザクション数でみると、１日に約７０万回コンポーネントが呼び出されている。その中でも今回再構築した基幹システムでの利用状況を表３にまとめた。今回、開発プロジェクトと平行してコンポーネントを開発し、再利用を推進した結果である。

表３.基幹システムにおけるコンポーネント利用状況

分類	ベース コンポーネント	業務サービス コンポーネント	ユーティリティ	パッケージ	合計
物流系	17	0	4	1	23
経理系	25	3	2	1	31
マーケティング系	22	6	2	1	31
流通系	10	2	1	1	14

ベースコンポーネント１つあたりのステップ数が平均して約３０００ステップと考えると、コンポーネントチームが開発したベースコンポーネントによって、各プロジェクトで約３万～７万のステップ数が削減できたことになる。この規模のコンポーネントが、今後の開発で再利用されていくことで、生産性がより向上していくことになる。また、コンポーネント化によって資源が極小化されることで、運用効率も向上していくと考えられる。

次に、基幹システムである物流システムで、コンポーネントの適用率の測定を行った。（表４）結果として、システムの機能の約１５％がコンポーネントによって提供されていることが分かる。今後再利用を促進していくことで、この適用率引き上げていき、ほとんどコンポーネントの組み合わせでアプリケーションが構築できる世界を目指さなくてはならない。

表４.物流システムにおけるコンポーネントの適用率

システム名	総ステップ数	利用 コンポーネント数	コンポーネント 総ステップ数	コンポーネントステップ 数／総ステップ数
物流システム	56,8195ＳＴＥＰ	16	8,8066ＳＴＥＰ	0.15

現時点では、まだコンポーネントの組み合わせだけで、アプリケーションが構築できるところま目標は実現できると考えている。

5.3. コンポーネント型開発スタイルの確立

コンポーネント型開発の目的は、コンポーネントの再利用による生産性の向上にある。そのためには大きく分けて、コンポーネントを整備することと、整備されたコンポーネントを再利用することが重要になる。この２つのサイクルをうまくまわすことで、生産性が向上していくのである。今回の取り組みをとおして、図８に示すコンポーネント型開発スタイルを確立することができた。そのポイントを以下に示す

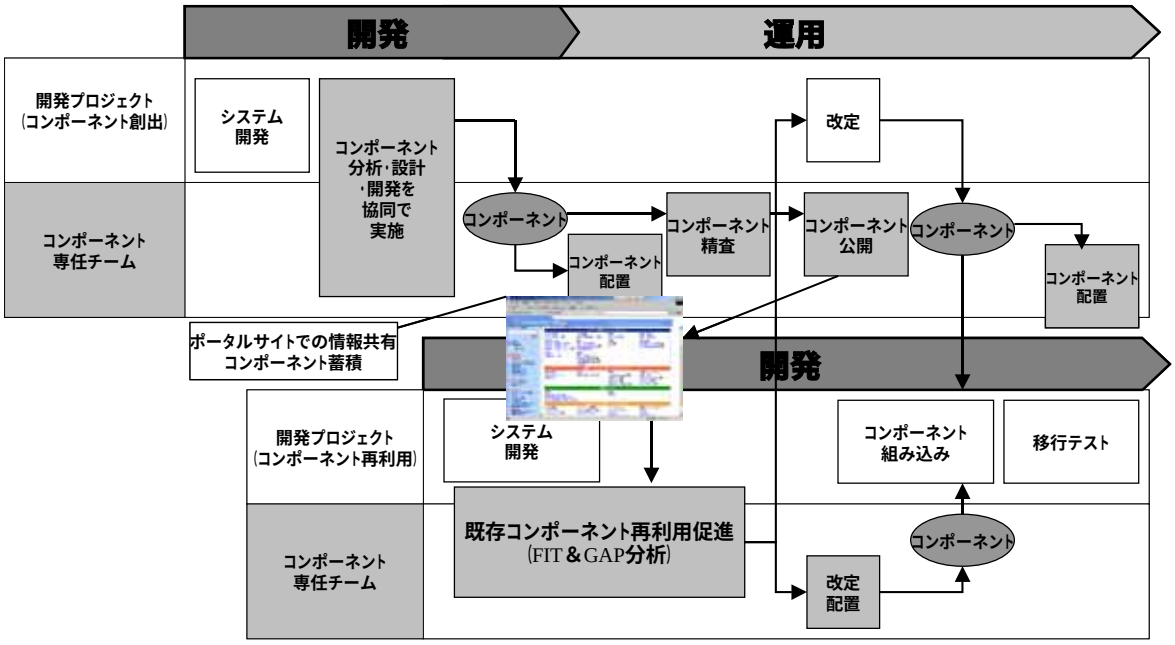

図８.コンポーネント型開発スタイル

（１）開発プロジェクトとコンポーネント専任チームの分離

コンポーネント専任チームを組織し、開発プロジェクトとの分業体制でコンポーネントを開発する。コンポーネント専任チームは、コンポーネントの設計開発運用について、責任を持ち、開発プロジェクトを支援する。また、開発されたコンポーネントの管理も行う。

（２）コンポーネントの開発と再利用の２つの側面で開発プロジェクトをとらえる。

開発プロジェクトを、コンポーネントを創出する側面と、コンポ―ネントを再利用する側面の２つに分けてとらえると、前者では、コンポーネント専任チームが各プロジェクトと並

行して、開発プロジェクト内のコンポーネント化される部分を開発していくことになる。後者の場合は、開発プロジェクトに対して既存のコンポーネントの適用検討（ＦＩＴ＆ＧＡＰ分析）を共同して行い、分析の結果仕様が足りない場合は、改定によるバージョンアップを行って対応していく活動になる。今回の開発では既に整備されているコンポーネントが少なかったため、必然的にコンポーネント創出の側面がほとんどであったが、今後開発するプロジェクトについては、じゅうぶんにコンポーネントが整備された状態からスタートするため、再利用をメインに進めていくことになるはずである。

今後の開発プロジェクトからは、まず再利用を検討し、既存のコンポーネントを適用しつつ、まだコンポーネント化されていない要素がある場合は、平行してコンポーネント専任チームとともに、コンポーネントの開発を行うというプロセスになる。

この開発スタイルが、再利用しながらコンポーネントが蓄積されていく「天使のサイクル」となるかどうかは、今後の再利用の進め方次第である。この開発スタイルを生かしてコンポーネントを再利用しつつ蓄積させていき、「２００４年以降にプロジェクトの開発生産性を４０％向上させる」という最終的な目標を達成していきたい。

5.4. 変化に強いアプリケーションアーキテクチャーの構築

コンポーネント型開発のもう１つの目的として、「変化に強いシステムを作る」ことが挙げられる。コンポーネントベースのアーキテクチャーを基幹システムに採用することで、業務サービスコンポーネントの層と、ベースコンポーネントの層で、それぞれアプリケーションのインターフェースと実装を明確に分離させている。その結果、データの変化はベースコンポーネントが、業務の変化は、業務サービスコンポーネントがそれぞれ吸収する変化に強いアーキテクチャーでアプリケーションを実現することができた。

6. 今後の課題

6.1. コンポーネントの配置と運用

コンポーネントは全システムから「共有」されることを前提に今回開発を行ってきた。各システムにコピーするのではなく、物理的に１つのコンポーネントを全システムから呼び出して利用する方法である。（図９当初の予定）ＥＪＢが分散オブジェクトとして実装されている以上、サーバー間での分散呼び出しは技術的には可能なはずであった。しかし実際のところ、アプリケーションサーバーの実装にもよるが、サーバー間のＥＪＢ呼び出しには制約が多数存在しており、またネットワーク経由で呼び出すことの負荷やパフォーマンスの劣化かなり大きいことが判明した。結果として、各アプリケーションが稼動しているサーバーごとにコンポーネントを配置する方法をとることにした。（図９現実解）

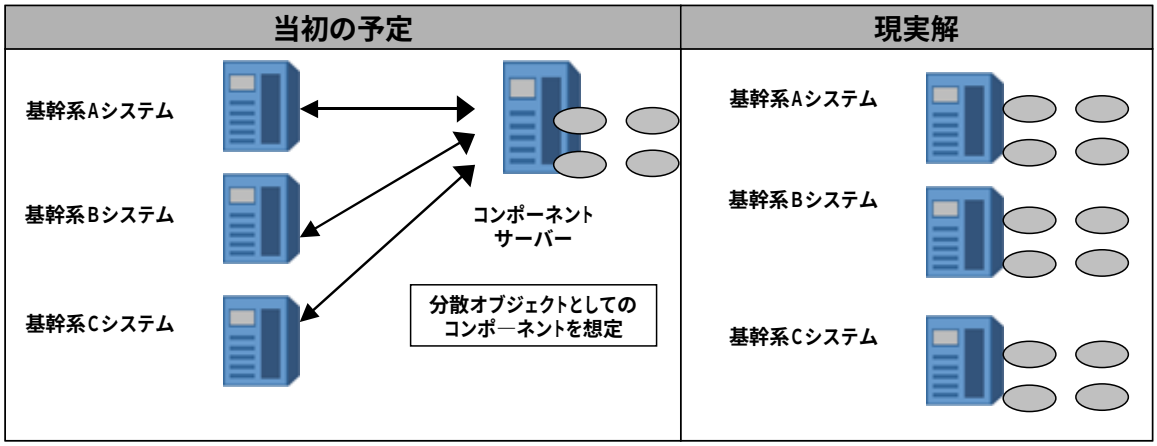

図９.コンポーネントの配置における当初案と現実解

コンポーネント化の目的の一つに、資源の一元化による運用効率の向上を挙げていたが、プログラムは一元化できても、実行モジュールを各サーバーに配置するという作業が必要となり、運用効率の大幅な改善は現時点では見込めない。配置ツールの整備などで、現在の構成で業務を軽減する方法を検討していく必要がある。

6.2. 業務サービスコンポーネントの創出

「２００４年以降に開発生産性を４０%向上させる」ことが、コンポーネント型開発の最終目標である。その最終目標を達成するためには、既存のコンポーネントの再利用でアプリケーションの大部分が開発される必要がある。そのために必要なコンポーネントの目標数を２００個に設定した。（図１０）２００個あれば、コンポーネントを組み合わせるだけで、アプリケーションがほとんど開発できてしまうであろうということである。現時点で９１個なので、およそ約２倍のコンポーネントが必要となる。そこでポイントとなるのが業務サービスコンポーネントである。ベースコンポーネントは２００３年に１年で５０個以上開発したという事実に対して、業務サービスコンポーネントは、１０個しか開発することができなかった。業務サービスコンポーネントについては、粒度が大きくアプリケーションに近いため、各開発プロジェクトで開発を実施したことが原因である。結果として、コンポーネント型開発を意識している開発プロジェクトからのみ、コンポーネントが創出されるという状態であった。２００４年末には、約７０個の業務サービスコンポーネントが創出される必要がある。粒度が大きくて汎用的なコンポーネントという一見相反する内容をどうやって実現するかが大きな課題である。業務サービスコンポーネント創出のための仕組みを整備していかなくてはならない。

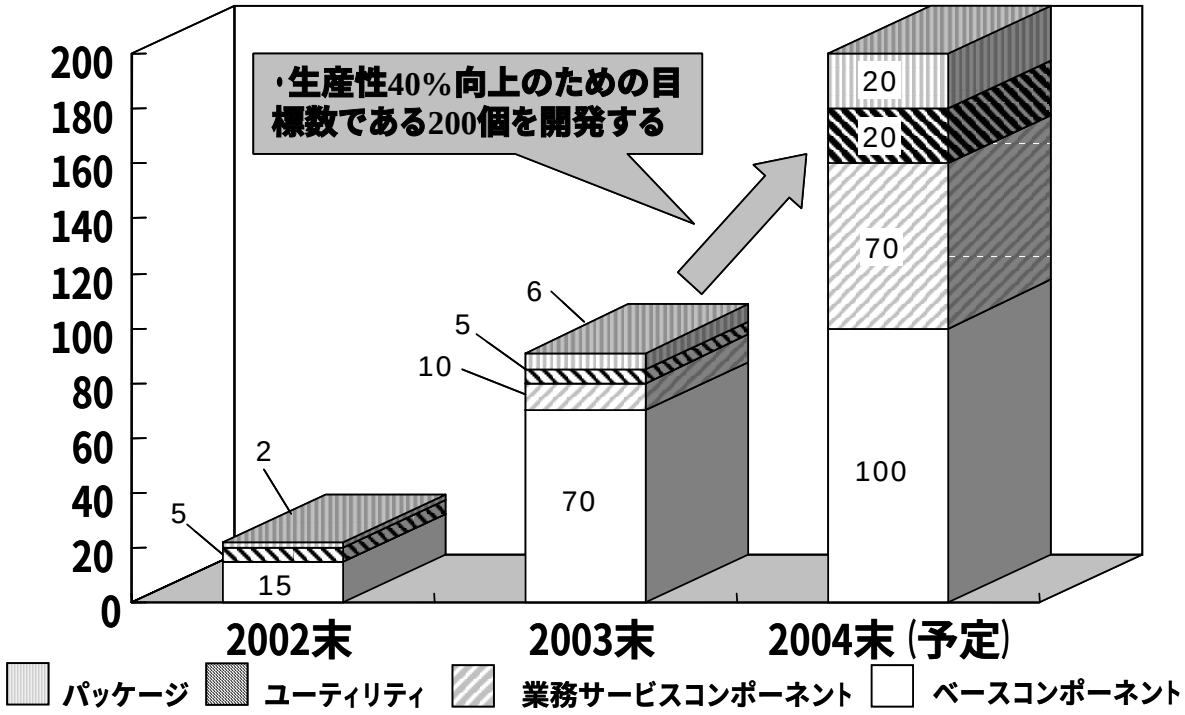

図１０.コンポーネント開発の目標数

6.3. 再利用による生産性向上

これまで、コンポーネントを蓄積することに重点を置いて取り組んできた。今後はコンポーネントを引き続き開発しつつも、効率よく再利用することに軸を移していくこととなる。そこで、再利用による効果の視点から、開発したコンポーネントを表５にまとめた。ここで分かる事実は、ベースコンポーネントは粒度が小さいため、適用は簡単にできるが、生産性にはあまり寄与しないことと、逆に業務サービスコンポーネントは、生産性には大幅に寄与するが、適用するパターンが限られているという事実である。（表５）

表５.コンポーネントの分類と再利用による効果

分類	コンポーネントの例	Fit率	生産性寄与	変化対応
ベース コンポーネント	相手先 人組織 部署 勘定 販売先 地名 商品 承認	○	△	○
業務サービス コンポーネント	EDI 原単位分解 受払計算 予算管理 実績管理 実績検索	△	◎	○
ユーティリティ	カレンダー	○	△	△
パッケージ	掲示板 データアップロード	○	○	△

今後は、この粒度の異なるコンポーネントの組み合わせをパターン化し、再利用パターンのようなものを作成していく必要がある。ただ情報を公開するだけではなく、必要なコンポーネントを選

択して組み合わせる作業をいかに効率化できるというところまで、踏み込んで支援をしていくことが、再利用促進と生産性向上のポイントとなる。開発プロジェクトに対して、コンポーネントをコンサルティングしていくことが重要になってくる。

7. おわりに

「２００４年以降のプロジェクトの開発生産性を４０%向上させること」が、今回のコンポーネント型開発における最終目的である。その意味において、現時点ではまだ折り返し地点を迎えたに過ぎない。ただし、コンポーネント型開発による効果は既に出始めているのは確かである。２０００年にＪＡＶＡを導入してから丸４年、基盤を整備し、開発体制を整備し、蓄積してきたコンポーネントがついに花開こうとしている。これまでの効果をみても、今後豊かな実りを迎えるのは間違いないと確信している。今後とも、ＦＯＲ　ＴＨＥ　ＴＥＡＭの精神で、全社アプリケーション全体の開発生産性向上を目指して邁進していきたい。

本論文が企業でコンポーネント型開発に取り組もうとされている方々のお役に少しでも立つことができれば幸いである。