オープン勘定系システム開発の事例紹介

Case Study of Open Banking System Development

中 原 直 紀, 三 島 敏 彦

要 約 オープン技術は急速に進歩しつづけており, ビジネスの世界においても様々な業界で, オープン技術を活用したIT化が推し進められている．金融機関においても，異業種から参入した銀行やインターネット専業銀行などオープン技術を活用した銀行が出始め，一般の銀行においてもオープン技術が主流となるのはそれほど遠くない未来と考えられる．

このような状況の中，日本ユニシスでは，勘定系システムのオープン化に本格的に着手した．

本開発に当たっては，現状の勘定系システムが抱える問題を解決するために部分的・段階的なシステム更改を可能とすべく，単純に勘定系システムのオープン化を行う作業のみならず, オープン技術の特性を活かして勘定系アプリケーション構造を再度見直す作業も行った．

Abstract Open technology is continuing to make rapid progress, and implementation of open technology based IT system is promoted in various business industries. Also in the financial industry, the banks using open technology, such as the Internet banks, new entrants from outside financial industry, begin to appear, and it is considered to be in the not so distant future that application of open technology becomes in mainstream also in traditional banks.

In such circumstances, Nihon Unisys got started the open technology conversion of the banking systems in earnest.

In this development, in order to enable partial/gradual system renewal for solving problems which the present banking system have, not only the works that performs simple technology conversion of a banking system but the work which improves banking application structure again taking advantage of open technology was done.

1. はじめに

オープン技術の進展に伴い，現在メインフレーム上に構築されている金融機関勘定系システムも，オープン技術を活用したシステムに変化して行くものと考えられる．

日本ユニシスでは，「オープン技術の成熟度に応じた段階的なオープン環境への移行」という基本方針に基づき，2000年度最初に本番稼働した勘定系システムパッケージSBI 21[*1]において，既に，各種統計帳票作成機能や営業店端末・ネットワーク環境のオープン化を実現しており，更に，プログラム開発から単体テスト工程までをPC上で行うことができる開発環境も提供している．

また, ミッションクリティカル性の高いシステム向けに, メインフレームと同等の高信頼性・可用性を備えたIAサーバであるES 7000を2000年3月に発表し，それと同時に勘定系システムオープン化の実証実験を開始した．

その実証実験結果に基づき，2001年1月にオープン環境上のミッションクリティカルシステムを支えるミドルウェアMIDMOST[*2]の開発に着手し，2002年に提供している．

このように，勘定系システム基盤としてのオープンシステム環境が整いつつあることを受けて，2002 年より，勘定系システムのオープン化に本格的に着手した．

本稿では，オープン勘定系システム開発にあたって，オープンの特性を活用したアプリケーション構造，データベース構造面について考慮・検討した結果について紹介する．

2. オンラインシステムのアプリケーション構造

2.1 新しいアプリケーション構造に求められるもの

一般的に，「勘定系システムは，度重なる制度改正，新商品・新サービス・新チャネル対応等により，年月を経るにしたがい，複雑化・肥大化し，保守性が著しく低下する」と言われている．

そのため，勘定系システムを現在の構造のまま単純にオープン化し再構築した場合，再構築直後は課題が解決されたとしても，年月の経過とともに現在と同様の結末を迎えるであろうことは想像に難くない．

したがって，勘定系システムのオープン化に際しては，現在とは異なる新しいアプリケーション構造の検討が必要である．

では，どのようなアプリケーション構造を実現すべきか．

過去の勘定系システムの再構築に際しては，スリム化が大きなテーマの一つであった．しかしながら，現実には，勘定系システムが対応すべき商品・サービス・チャネル等は，金融の自由化に伴う多様化に加え，急激に進化する IT 技術活用による多様化，異業種交流による多様化等により，もはや 10 年後の金融機関に求められる勘定系システムを思い描くことが困難な状況にある．そのような中で，将来にわたってスリムで有り続けることを保証できる勘定系システムの実現は非常に難しい．

一方で，現在の勘定系システムが複雑化・肥大化しているといっても，勘定系システム全体に当てはまるわけではない．普通預金，定期預金，証書貸付などの業務単位で見た場合，今のままでも保守性にはほとんど問題がないという業務も多い．ただ，現在の勘定系システムのアプリケーション構造は，問題がある部分だけを再構築することが難しい構造であるため，現実的には勘定系システム全体を指して複雑化・肥大化していると言われている場合が多い．

以上のことを考えると，勘定系システム再構築に際しては，スリム化は不変のテーマではあるものの，10 年後の金融機関の姿が不透明な現在，変化に追随することが難しくなった部分だけを容易に書き替えることができるアプリケーション構造の実現が最も重要なテーマと言える．

2.2 コンポーネントウェアに基づくオンラインアプリケーション構造

従来の勘定系システムの部分的書き替えを困難にしている要因は「アプリケーション構造が，取引を行うための処理とデータの流れを中心とした構造になっている」からである．

具体的には以下のような問題があげられる．

- データベース構造に依存したアプリケーション構造のため，データベース構造の変更を伴うような修正・変更が困難．
- 各プログラムは自身の処理に必要なデータベース全てを直接参照・更新できるため，結果として一つのデータベースが複数のプログラムから操作されており，データベースの

変更を行った場合の影響が広範囲.

・取引の流れを想定したプログラムとなっており，部品として他の処理で再利用することが困難.

・プログラムの再利用率を高めるために，共通サブルーチンを多用した場合，サブルーチン内に取引毎の判定箇所が多くなり，可読率の低下を引き起こし保守性が低下.

上記課題への対応として，オブジェクト指向技術の考え方を採用した．ただし，本アプリケーション構造は，オブジェクト指向技術の方法論を忠実に実現することを目標としているわけではなく，あくまで上記課題解決の手段としてカプセル化の考え方だけを採用しているにすぎない.

オブジェクトとは，一般論で言うとデータ（属性）とその振る舞い（操作）を一体化させたものである．本システムにおけるオブジェクトとは，勘定系システムを構成する要素（例：普通預金口座，取引明細，定期預金口座等）毎に，その保有する属性（データベース項目）とその属性に対する操作（例：入金，出金等）を一体化させ，プログラムとして実装したものを指している．図1は，従来の勘定系システムとオブジェクト指向技術を取り入れたアプリケーション構造の相違例である.

ここでは，従来の処理（→で示した横の処理の流れ）を顧客，普通預金，取引明細の三つのオブジェクトから形成されるシステムとして捉えている．このように，従来の勘定系システムをオブジェクトの集合体として捉えると，各オブジェクトの果たすべき役割は極めて簡素になる．各々のオブジェクトは他のオブジェクトの処理からは独立した構造となり，また，各々のオブジェクトが保有する属性はオブジェクトの外側には解放されていないため,自身の属性(すなわち，データベース）を変更しても他のオブジェクトに影響をおよぼすことはない.

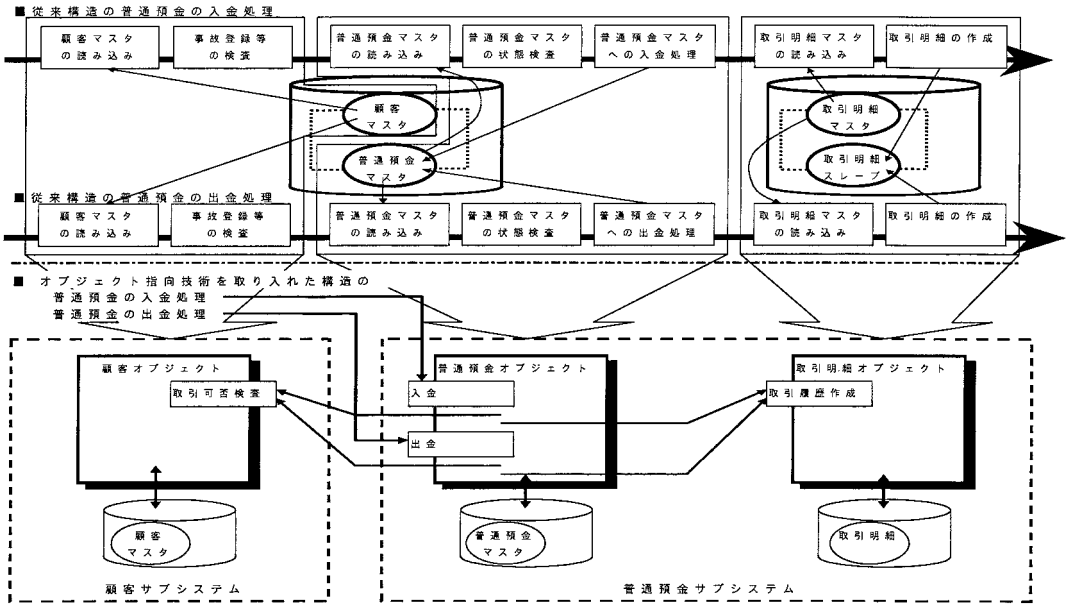

図1　従来のAP構造とオブジェクト指向技術を取り入れたAP構造の相違

しかし，単純にオブジェクト指向技術を適用しただけでは，以下の課題が新たに発生することになる．

1）非常に密接に関係したオブジェクト間においては，データの転記処理が増える．

2）オブジェクトの数が増えるとオブジェクト間の関連を管理することが難しくなる．

上記 1）に関し，普通預金と取引明細の二つのオブジェクトの関係で説明すると，取引明細は通帳記帳や各種照会処理のために，普通預金に対する処理（例：入出金）の履歴を管理するオブジェクトであるため，取引明細オブジェクトの属性には普通預金の属性が多く含まれることになる．しかしながら，取引明細オブジェクトには普通預金オブジェクトの属性は開示されていないため，普通預金オブジェクトが取引明細オブジェクトを呼び出す際に，①普通預金オブジェクト内で自身の属性→取引明細の入力パラメタ，②取引明細オブジェクトの中で入力パラメタ→取引明細の属性というように 2 回のデータ転記処理が必要となる．

また，2）に関して言えば，取引明細オブジェクトは，多くの場合，普通預金オブジェクトからのみ呼び出されるオブジェクトであるため，他のオブジェクトとの関連までを管理する必要はない．

したがって，前述した二つの課題を解決するために，各オブジェクトをそれぞれ単体で扱うのではなく，複数のオブジェクトを統合して取り扱うこととした．その統合した単位を本システムのアプリケーション構造では，「サブシステム」と呼ぶ．図 1 の場合は，「顧客サブシステム」と「普通預金サブシステム」の二つのサブシステムから形成されることになる．すなわち，一つのサブシステムは一つ以上のオブジェクトから形成された集合体ということになり，これがコンポーネントウェアでいうところのコンポーネントの単位となる．

コンポーネントウェアにおいては，コンポーネント内部の構造は特に規定していないため，普通預金・取引明細の両オブジェクト間では，各々の属性を直接受渡しすることが可能となり，それによりデータ転記処理を削減できる．また，各オブジェクト間の関連はサブシステム内だけの管理ですみ，オブジェクトが増加していっても，オブジェクト間の関連を管理する負荷は余り増大しない．

前述したサブシステム単位にコンポーネント化したオンラインアプリケーション全体の構造は図 2 に示す通りである．

この図におけるサブシステム A,B という単位は，顧客，普通預金，定期預金，証書貸付という業務単位である．サブシステム内の処理ロジック簡素化のために，各々のサブシステムは，そのサブシステムの役割の範囲に限定されたサービスのみを提供する．例えば，定期預金を解約し，普通預金口座へ入金するという業務処理において，定期預金サブシステムの持つ「解約」というサービスは，定期預金の「解約」処理のみを行い，その元金および利息額を普通預金に入金するというような処理は行わない．なぜならば，普通預金を操作するという処理は，明らかに定期預金サブシステムの役割の範囲外だからである．

したがって，定期預金を解約し，普通預金口座へ入金するという業務処理を遂行する場合，各サブシステムが提供するサービスの呼び出し順序等を制御するコンポーネントが必要となる．本システムでは，そのコンポーネントを「AP コントローラ」と名づけた．

前述した例においては，AP コントローラが定期預金サブシステムの「解約」サービスを呼び出し，その後，普通預金サブシステムの「入金」サービスを呼び出すという制御を行うことになる．

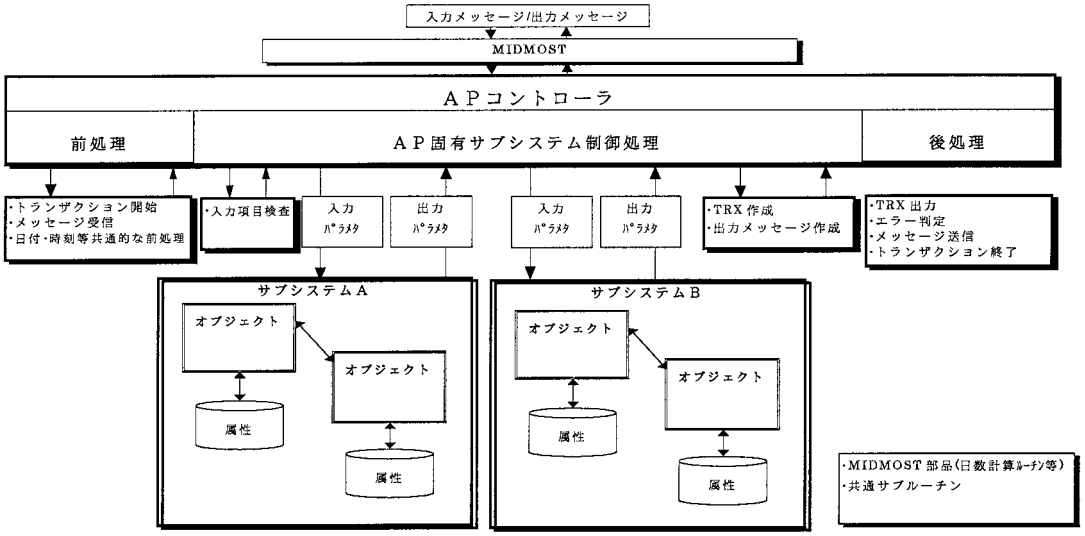

図 2　オンラインアプリケーション全体構造

この構造により，各サブシステム，および，AP コントローラは入出力パラメタだけを接点とした疎結合状態で結合されることになる．したがって，入出力パラメタさえ保証できれば，サブシステム内部を書き替えてもシステム全体に大きな影響をおよぼすことはなくなり，サブシステム単位に部分的に書き替えが可能なアプリケーション構造となる．

2.3　処理性能重視から保守性重視へ

2.3.1　デリバリチャネルの多様化への対応を事例として

本アプリケーション構造においては，個々のオブジェクトが保有する属性（データベース）は，そのオブジェクトが帰属するサブシステム内にカプセル化される．したがって，他のオブジェクトが保有するデータベースを更新・参照する場合は当該オブジェクトが帰属するサブシステム経由で行うことになる．

この場合，サブシステムの単位を小さくすればするほど，一つの業務処理で呼び出すサブシステムのサービスの数は多くなり，処理性能面においてサブシステムの呼び出し負荷や I/O 回数増加に伴う負荷が問題となる．

しかしながら 2000 年 3 月より実施した勘定系システムオープン化のための実証実験から，価格性能比においては，メインフレームを凌駕するという結果を得ていた．

この実証実験により処理性能面での制約が大幅に緩和できることがわかった．このことを受け，従来の処理性能重視の考え方から，処理ロジックのわかりやすさ，変化に対する対応のしやすさへと重点を移すことが可能となった．以降にその事例としてチャネルの多様化への対応例について記述する．

金融機関のチャネルは，営業店の金融端末や自動機に始まり，専用端末（ファームバンキング・ホームバンキング），電話（テレホンバンキング），汎用 PC やモバイル端末・携帯電話（インターネット接続）などますます多様化してきており，チャネル多様化への柔軟な対応は勘定系システムの重要な要件の一つになってきている．

図 3 は従来の勘定系システムにおけるアプリケーション処理を単純にサブシステム化した際の普通預金の自動機出金取引のイメージを表したものである．普通預金に対する出金処理の中で，チャネルが自動機であることを考慮し，手数料徴収の判断を行い，その結果，必要があれば出金処理と同時に手数料の支払処理も行っている．これはプログラムの呼び出し負荷軽減や I/O 回数削減といった処理効率確保のための考慮であるが，結果として，自動機取引で発生する手数料徴収処理というチャネル固有の処理を普通預金サブシステム内に抱え込む構造となってしまっている．

そのため，新規チャネル追加に伴う手数料徴収処理ロジックの追加・変更の発生，あるいは，既存チャネルにおける手数料徴収の考え方の変化が起こると，普通預金サブシステム内の処理ロジックにまで影響が波及することになり，チャネルの多様化に対して柔軟な構造とは言い難い．

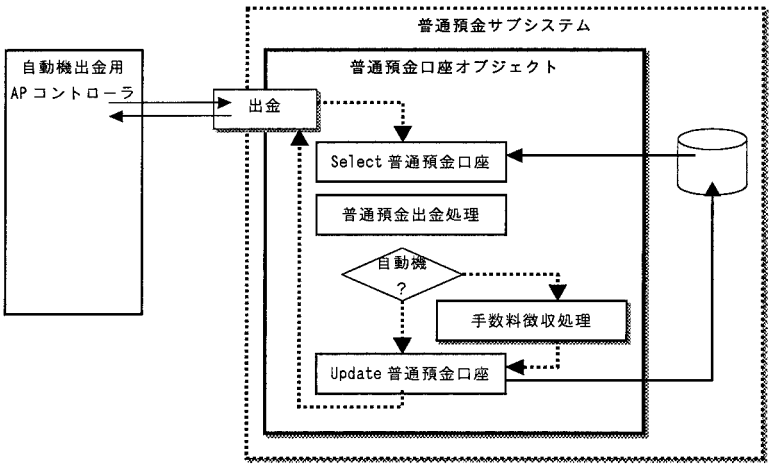

図 3　普通預金自動機出金取引イメージ

この場合，手数料徴収処理を普通預金サブシステムから外し，普通預金出金処理→手数料算出処理→普通預金出金処理（手数料分の出金処理）という流れにすると，普通預金サブシステム内の処理は，チャネルが自動機であるかないかに関わらず，単純に出金という業務処理だけを行えばよいことになる．すなわち，普通預金サブシステムはチャネルの影響を受けない部品となり，新チャネル追加時に普通預金サブシステム内のロジックを見直す必要はなくなるため，チャネルの多様化に対応しやすい構造が実現できる．構造変更後の普通預金の自動機出金取引イメージについて図 4 に示す．

変更前の構造においては，普通預金サブシステムの呼び出しは 1 回で済んでおり，普通預金のデータベースへの Update の I/O 回数も 1 回で済んでいる．それに対し，構造を変更することによって，普通預金サブシステムの呼び出し，普通預金のデータベースへの Update の I/O 回数とも 1 回ずつ増加することになり，確かに処理効率は悪化する．しかしながら，効率の悪化に関しては，前述したようにオープン化に伴う処理性能向上で補うものとし，チャネル多様化に対する保守の容易性に重点を置くものとする．

ここに記載した事例は一例であるが，性能重視から保守性重視に視点を転じて見直すことに

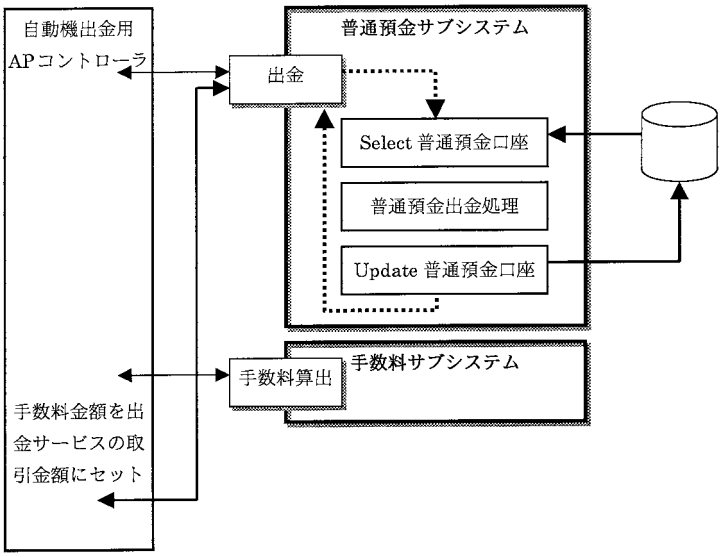

図 4　構造変更後の普通預金自動機出金取引イメージ

よって，勘定系システムの構成要素であるサブシステムを外部環境の変化に対し堅牢な部品にすることができ，勘定系システム全体の保守性を向上することができるようになる．

3.　バッチシステムのアプリケーション構造

コンポーネントウェアの採用により，オンラインシステムのアプリケーション構造については，部分的・段階的なシステム更改が可能となった．

しかし，オンラインシステムとバッチシステムの関係に着目すると，バッチ処理は複数のサブシステムが保有するデータを横断的に扱うという性格をもっているため，これまでオンライン側のサブシステムの変更に対して大きな影響を受けてきたことに気づく．

つまり，バッチシステムが従来通りオンラインシステムとの関係を保ったままでは，依然として部分的・段階的なシステム更改はできない．

したがって，本章では部分的・段階的なシステム更改を可能とするために，オンライン処理とバッチ処理の処理特性からの分析とそれに対する考慮点について述べる．

3.1　オンライン処理とバッチ処理のデータベースに求める特性の相違

前述したように，勘定系システムはオンライントランザクション処理効率が要求され，オープン環境上といえども，データベース設計において，オンライン処理の処理効率を完全に度外視して考えることはできない．したがって，物理的なデータベースの構造は個々のオンライン処理に対して最適化された構造とならざるを得ない．

一方バッチ処理は，帳票作成やデータ累積等の大量データに対して一括して処理することを目的としているため，各々のオンライン処理にとって最適化された多種多様のデータ構造を相手にするのでは処理が複雑になってしまう．

また，現在の勘定系システムにおいて，24 時間 365 日稼働は避けては通れないテーマであり，オンライン側のデータベースは静止することができない．しかし，バッチ処理においては，

1日の処理確定時点，あるいは，月末時点のデータベースが求められる場合が多い．

このように，オンライン処理とバッチ処理ではデータベースに求める特性は大きく異なっている．

表1にその相違点を示す．

表1　オンライン処理とバッチ処理のデータベースに求める特性の相違

オンライン処理	バッチ処理
最も処理効率を要求されるオンライン処理にとって最適な構造のデータベース	データの切り出し等が容易なフラットで単純な構造のデータベース
24時間365日静止しないデータベース	前日末，前月末で静止しているデータベース

上述したような相違があるにも関わらず，従来の勘定系システムでは，バッチプログラムもオンライン処理で使用するデータベースをそのまま入力とし処理する業務フローとなっている場合が多い．

また，データベースを共有していることにより，システム全体が難解で複雑になってしまうという弊害が発生することがある．勘定系システムが難解で複雑になっていく一つのプロセスとして図5に示すケースがあげられる．

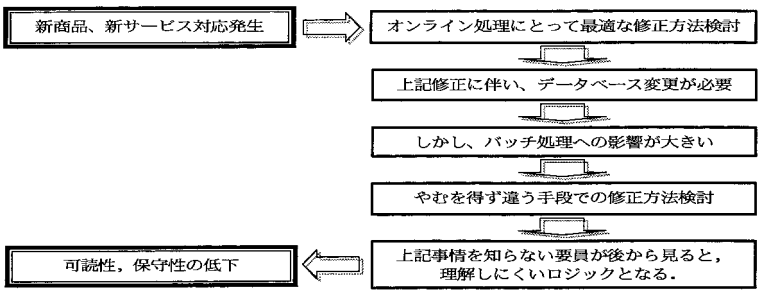

図5　システムが難解で複雑になっていく一つのプロセス

3.2　バッチ処理の役割による分類

一方，バッチ処理をその役割によって分類すると，大きく以下の四つに分けられる．

① 外部報告資料・事務用帳票・顧客宛ダイレクトメール等の各種還元帳票の作成

② 各種元帳作成や取引ログ累積等の保存データ作成と渉外支援・自己査定等の他システムや情報系システムへのデータ配布

③ 勘定日付繰越やデータベースガーベッジ処理等のオンライン運用処理

④ 自動振替データ抽出・実行・結果管理や為替取引終了後処理等のオンライン連携処理

①，②の処理については，ある時点の静止データベースを必要とするバッチ処理である．これらの処理を行う上では，オンライン処理で使用しているデータベースと同じものを使用する必要はなく，むしろある時点での静止状態を確保したデータベースを使用できる方が望ましい．

一方，③，④の処理は，勘定系システムにおけるオンライン処理補完を行うバッチ処理であり，使用するデータベースは，当然オンライン処理で使用しているデータベースと同じでなけ

ればならない．

また，この分類におけるバッチプログラムの本数比率では，①，②の方の比率が高く，かつ，①，②から出力される帳票の多くは，そのデータの抽出条件や加工条件がエンドユーザ側からの要求により，頻繁に変更になるという特性も持っている．

したがってバッチ処理は，オンライン処理を補完するためにオンラインデータベースを入出力とするバッチ処理と，ある時点での静止データベースを入力とし，エンドユーザニーズの変化に対して容易な対応が可能となる処理構造が要求されるバッチ処理の 2 種類に大別することができる．

3.3　バッチ専用データベースの必要性

3.1 節に記載したデータベースに求める特性の相違と 3.2 節のバッチ処理の役割による分類結果から，オンラインデータベースとは別にバッチ専用データベースを設け，オンライン環境とバッチ環境を切り離すこととした．つまり，オンラインデータベースからバッチ処理に必要なデータを抽出し，バッチ処理が取り扱いやすいフラットで単純な構造の静止データベースを専用に構築することとした．このバッチ専用データベースを「大福帳 DB」と呼び，これをオンライン処理との境界面とする．

このような構成は，オープン系システムにおける，ディスク装置の大容量化，低価格化により可能となる．

オンライン処理とバッチ処理でデータベースを共有している従来の勘定系システムとバッチ専用データベースを保有するオープン勘定系システムの比較イメージを図 6 に示す．

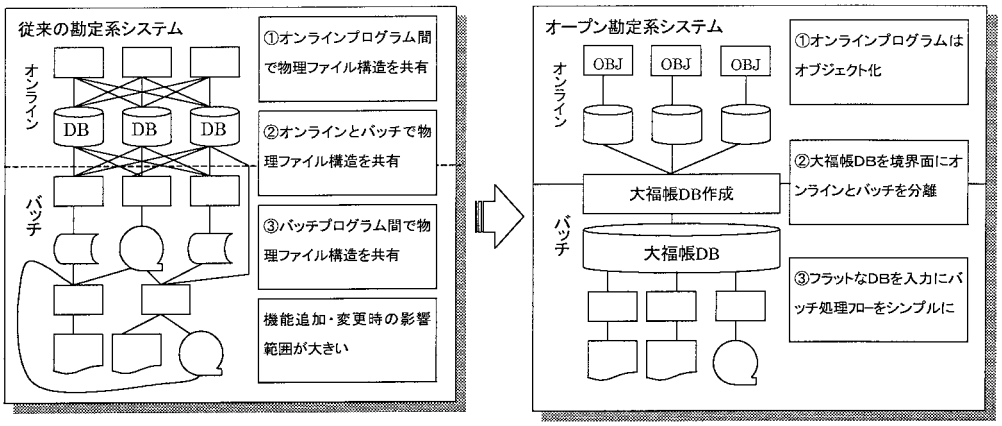

図 6　従来の勘定系システムとオープン勘定系システム

大福帳 DB を構築することにより以下のような利点を得ることができる．

①　静止データの確保

大福帳 DB 作成バッチにより前日末/前月末の大福帳 DB を作成することで，静止データの確保が可能となり，大福帳 DB のみを入力とすることで前日末/前月末基準日処理が可能となる．

② オンライン側/バッチ側の変更を局所化

大多数のバッチプログラムを大福帳 DB 入力とすることにより，オンライン側のサブシステムに対し，その保有するデータベースの変更を伴うような大幅な書き替えを行ったとしても，その影響は，大福帳 DB 作成バッチの変更だけで吸収することが可能となる．大福帳 DB を使用するバッチプログラムは，RDB（Relational Data Base）の特性を活用し，自身が必要とする列項目だけを処理対象としており，メインフレームのようなレコードという概念がないため，自身が必要とする大福帳 DB の列項目に影響がない限り，影響を受けることはない．

③ バッチプログラム生産性の向上

大福帳 DB はバッチ一括処理に適したフラットな構造のデータベースであるため，オンラインデータベースを入力とする場合に比べ，入力データの取扱がシンプルになる．なぜなら，複数のデータベースを入力として行っていた処理を，条件指定によるクエリの発行に置き換えることが可能となるからである．

④ 統計帳票の EUC 促進

大福帳 DB を対象とした OLAP（オンライン分析処理）を行うことにより，統計帳票作成などの様々な切り口で分析を行う作業の EUC 化（エンドユーザ対応）を促進することにつながる．

3.4 大福帳 DB の内容と作成

大福帳 DB の内容は，以下の通りである．

1） データ種類

① 大きくは，顧客情報，口座情報，トランザクション，日計情報，その他（商品・金利管理データ，個別データ，二次加工データ）とし，取引明細はトランザクションにて代替するものとし，原則として保有しない．

② 口座情報は，複数科目を対象とする各種統計帳票作成処理を考慮して流動性，定期性，融資のカテゴリで複数科目をまとめて保有する．統計帳票作成処理以外のバッチ処理では科目単位の処理も多くあるが，複数科目を対象とする統計帳票作成時に，同一テーブルから必要科目を Select で簡単に条件抽出できることと，科目別にテーブル化された複数テーブルの操作をしなければならなくなることを比較した場合，前者の方が操作性と処理効率に勝っていると考え，複数科目をまとめた 1 テーブルとする形態とした．

2） 口座のデータ項目

原則として【オンラインの口座データベース保有項目】＋【バッチ処理に必要な切り口項目】＋【バッチ用加工項目】を保有する．

① 【オンラインの口座データベース保有項目】

明らかにバッチ処理では不要と考えられる項目を除いて，オンラインで保有している全データベース項目を保有する．

② 【バッチ処理に必要な切り口項目】

バッチ処理に必要な切り口項目とは，「人格コード（預金者区分コード）」，「担当者コード」，「業種コード」といった各種統計帳票を作成する際に集計キーとなるような属性項目を指す．これらの属性項目は顧客情報にも保有するため，データの正規化という観

点から見ると好ましい保有形態ではないが，口座データに保有しない場合，統計帳票を作成する際に，都度，大量データの表結合が発生し，処理効率の低下を招くため，口座情報にも重複して保有する．

③ 【バッチ用加工項目】

口座データの中には，オンライン処理には関係がないためオンラインデータベースには保有していないが，各種バッチ処理のためには必要なデータ項目というものがある．それらのデータに関しては，大福帳 DB 作成バッチが作り出すことにする．そうすることにより，大福帳 DB を入力とするバッチプログラムは複雑な計算処理はほとんど不要となり，SQL 文による一括処理を最大限に活用することができる．

また，大福帳 DB 作成に関しては，以下のように行う．

- 日付繰越のタイミングにおいて，レプリケーションしているオンラインデータベースのミラーボリュームを切り離す．
- それにより静的状態となったデータベースを入力として大福帳 DB 作成バッチを実行する．
- 月末日においては，月中に勘定異動が発生しなかった口座の月末残高積数の算出を行い，大福帳 DB の月末確定状態を作成する．

なお，大福帳 DB の更新は大福帳 DB 作成バッチのみ可能とし，一般の業務バッチでの更新は認めない．

また，前日対比，前年同月対比の計算を可能にするため大福帳 DB の保有世代数は，日次 2 世代，月次 13 世代としている．

3.5 大福帳 DB を入力とするバッチプログラムの注意点

オープン環境上では，オンラインデータベースも，大福帳 DB も RDB である．RDB の場合，SQL 文によりかなりのアプリケーションロジックを記載することができる．したがって，現在のバッチプログラムロジックの大半を SQL 文だけで記述するということも可能である．特に，大福帳 DB を入力とした場合，バッチで必要とする加工データ項目のほとんどについて事前に大福帳 DB 作成バッチプログラムが計算して，結果を大福帳 DB に設定しているため，ほとんどのロジックはデータの条件抽出，SORT，四則演算といった SQL 文の範囲で対応できる．

しかしながら，以下の理由により，SQL 文の使用範囲に関しては，ある程度の制約を設ける必要がある．

- SQL 文は，RDBMS（Relational Data Base Management System）によって非互換部分が少なからず存在し，SQL 文への依存度を高めてしまうと，将来的に RDBMS を変更することが難しくなってしまう．
- あまりに複雑な SQL 文は可読性および保守性に難点があり，同様の処理を開発言語にてプログラミングした方が良いケースもある．

また，大福帳 DB 作成の弊害として，バッチプログラムが使用できるオンラインの共通サブルーチンが限定されるという問題がある．大福帳 DB を入力とするバッチプログラムは，極力オンライン処理側の変更の影響を受けないようにするため，オンラインデータベースの使用を認めていない．これにより，オンラインシステム側のデータベースを自身の中で使用している

共通サブルーチンは，バッチプログラムでは使用できないこととなる．同様の理由により，各サブシステムが提供しているサービスに関しても，大福帳 DB を入力とするバッチプログラムでは使用することができない．

これらに関しては，そのサブルーチン，あるいは，サービスの処理ロジック自体が必要なのか，その処理結果だけが必要なのかを見極め，後者であれば，あらかじめその結果を大福帳 DB にデータ項目として保有することを検討する必要がある．

4. お わ り に

本システムは 2003 年 1 月現在，本稿で記載したアプリケーション構造に則り，実際のアプリケーションの一部書替を終え，WEB システムをフロントとしたデモンストレーション取引が可能となっている．

2000 年 3 月に開始した実証実験から，既に 3 年近くが経過したことになる．このような長期間を費やしたのは，「金融機関の勘定系システムというミッションクリティカル度が極めて高く，公共性の高いシステムのオープン化というテーマに対しては，慎重の上にも慎重を期して技術を検証し，ノウハウを積み重ねていくべき」と考えたからである．また，現行勘定系システムのオープン環境への単なる移植ではなく，オープンの特性を活用することによって，現在の勘定系システムが抱えている課題をどのように解決できるかという点に関しても検討を重ねてきたからである．

これまで述べてきた考慮点や検討結果はその成果物の一部である．

勘定系システムのオープン化に関しては，24 時間 365 日稼働時のセンタカット処理やシステム運用について，若干，技術検証が必要な部分が残されてはいるが，今までの実証実験/開発作業を通して，勘定系システムを再構築する際の選択肢の一つとして提案できる時期に至ったと考えている．

残された技術検証を含め，今後とも開発作業を継続していく予定である．

本稿が勘定系システムのオープン化を検討している方々の参考になれば幸いである．

＊ 1　SBI 21：Strategic Banking Integrated-System for 21 century．地域金融機関向け勘定系パッケージ

＊ 2　MIDMOST：MIDdle-ware for Mission critical Open SysTem．オープン系ミッションクリティカル業務向けミドルウェア．本特集号別稿の「オープン系ミッションクリティカル業務構築を支援する MIDMOST 技術基盤」参照．

執筆者紹介　中　原　直　紀　（Naoki Nakahara）

1989 年東海大学工学部卒業．同年日本ユニシス(株)入社．勘定系ソリューション・パッケージ TRITON の開発，SBI 21 のインフラ適用業務を経て，オープン勘定系システム開発業務に従事．現在，システムサービス本部金融第二システム統括部開発一部に所属．

三　島　敏　彦（Toshihiko Mishima）
1989 年北九州大学商学部卒業．同年日本ユニシス（株）入社．勘定系ソリューション・パッケージ TRITON のアプリケーション開発および適用，SBI 21 の適用業務を経て，オープン勘定系システムのアプリケーション開発業務に従事．システムサービス本部金融第二システム統括部開発一部に所属．