保険金融商品のための契約管理アプリケーション設計

欧州各国や米国などの保険先進国では、保険料の支払いと補償付加の自在性を狙って、保険商品の開発が進められてきました。現在での最高到達点は、「ユニバーサル・ライフ」です。その進化は、従来の責任準備金を中心とした蓄積機能から個人勘定の蓄積機能に移ることによって実現されました。しかし、この商品は、保険の補償と個人勘定（銀行の口座のようなもの）という、まったく性格の異なった商品の組み合わせです。このため、複数の契約サブシステムの連携管理が必要です。このような複数サブシステムの管理を行い、契約や取引の過去にさかのぼった自在な変更を可能にするための中核技術が、UNDO/REDO機能です。本論文は、UNDO/REDO機能の設計を提示するものです。

日本アイ・ビー・エム株式会社
金融ソリューション・センター 保険第4システム部
主任ITアーキテクト
Advisory IT Architect
Development Insurance No.4, Services Derivery-Finacial Services
IBM Japan, Ltd.

竹内 孝明 Takaaki Takeuchi

［プロフィール］
1984年日本アイ・ビー・エム入社。以来、システムズ・エンジニアとして、生命保険／損害保険会社の個人保険、団体保険の契約管理システム開発などのプロジェクトに数多く参加した。また、この知識・経験を基に、保険のビジネス・モデルであるIAAをITモデルにまでに拡大し、関連するソリューションの開発を多数行ってきた。現在は、上記のITモデルやソリューションに基づいた契約管理システムを損害保険会社で構築するプロジェクトで全体設計を担当している。

日本アイ・ビー・エム株式会社
金融ソリューション・センター 技術推進部
ICPシニア・コンサルティングITアーキテクト
ICP Senior IT Architect
Technical Promotion, Services Derivery-Finacial Services
IBM Japan, Ltd.

大橋 敏男 Toshio Ohhashi

［プロフィール］
1975年、日本アイ・ビー・エム入社。以来、システムズ・エンジニアとして、銀行システム・流通管理システムなどの開発を経て、1980年以降、生命保険／損害保険会社の基盤アプリケーション・システムの開発に携わる。1993年以降、IAAを基に、保険商品分析・サブシステム定義・契約管理システム設計・保険料請求計上システム設計・金融商品対応など、保険会社のITソリューションの開発を主導してきた。現在は、金融工学を基にし、保険を包含した複合商品の契約管理システムを構想している。

Design of Policy Management Control Applications for Insurance Finance Products

In Europe, the United States and other countries at the forefront in the world of insurance, efforts have been made to develop insurance products with the aim of achieving more freedom for the payment of insurance premiums and for adding compensation. The highest point reached at the present time is "Universal Life." Evolution in this area has been achieved by moving from accumulation functions centering on existing liability reserves to functions involving the accumulation of personal accounts. However, this product is actually a combination of products with entirely different characteristics in the form of insurance compensation and personal accounts (along the lines of bank accounts). This means that there is a need for linked management of several contract subsystems. Undo/Redo functions constitute the core technology for enabling management of several subsystems and free changes in contracts and transactions that go back into the past. This paper offers proposals for the design of Undo/Redo functions.

1. はじめに

本論文で扱うUNDO/REDO機能を組み込んだ契約管理アプリケーションの設計は大変大掛かりなものであり、残念ながら国内では、変額保険を一部で対応しているにすぎません。まして、ユニバーサル・ライフを実現するための契約管理アプリケーションを構築しようとしても、ノウハウもないのが実情です。

幸い、筆者たちは2年前からある損害保険会社のお客様において、新しいタイプの契約管理システムを構築する機会を得ました。そして、本論文の記述対象となったUNDO/REDO機能を組み込んだ契約管理アプリケーションを完成することができました。

本論文の設計は、複数サブシステムや契約管理の本質を見据えた、非常に一般化したものとなっています。欧米のパッケージのUNDO/REDO機能は、単一サブシステムにしか適用できません。それに対して、本論文で提示した設計は、サブシステムの追加も容易にできる柔軟な設計となっており、保険の金融商品以外にも適用可能なものだと自負しています。

筆者たちは、IBMがワールドワイドで展開しつつある保険のビジネス・モデルであるIAA（Insurance Application Architecture）に基づいてIT（Information Technology：情報技術）モデルを構築してきました。IAAの非常に汎化されたビジネス・モデルを生かすために、ITモデルも可能な限り汎用的なものにしています。こうした長年の検討過程で得られたソリューション群を、CAFI（IAA Based Core-Insurance Application Design Framework and Inplementation）と名付けています。

2. 保険金融商品のための必要機能

2.1. 保険金融商品とは何か

保険の本質として、保険料の収入と保険金の支出との間にはタイム・ラグがあります。このタイム・ラグの間の資産運用は、とても重要な課題です。特に、契約期間の長い生命保険においては、さらに重要な課題となります。また、顧客利便性の観点から、平準保険料を払う方法が導入されています。このため、必然的に次の二つの課題が保険、特に生命保険に課せられることとなりました。

(1)保険料の収入から保険金の支出まで長いタイム・ラグがあるため、この間の資産運用が重要になる。

(2)平準化された保険料のうち、死亡に対する危険保険料に充当される金額を除いた残りの金額は、責任準備金として一般勘定で運用される。しかし、この責任準備金があるために、保険料や保険金などの契約内容の変更を契約者は簡単にはできない。

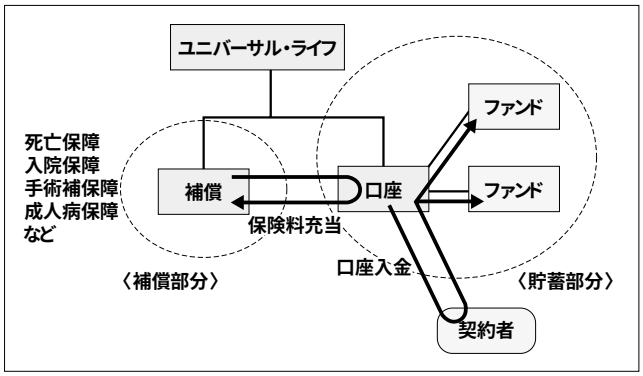

図1. ユニバーサル・ライフの構造

上記の問題を解決するために、新しい金融技術を利用した個人勘定を多用し、運用を個人に任せるという新しい発想で出てきたのが、保険金融商品です。代表的なものとして挙げられるユニバーサル・ライフは、保険料の蓄積部分を個人資産とし、補償部分と完全に切り離した商品です。資産運用と保険料の平準化を、完全に顧客責任とし、上記の問題を解決しているのが特長です。

図1は、ユニバーサル・ライフの構造です。補償部分と貯蓄部分が完全に分離しています。図の左側が補償部分で、補償に必要な保険料は保険料の請求時に口座を窓口にしてファンドから引き出され、各補償に充当されます。補償側は責任準備金を持たず、補償側の保険料は死亡率をベースにした自然保険料※です。

図の右側が貯蓄部分で、契約者からの保険料は口座に入金され、口座は保険料の管理・運用をファンドに依頼します。一つの口座には、複数のファンドを付与することが可能です。貯蓄側の保険料は契約者の利便性を考え、一定額の平準保険料※となっています。

※自然保険料とは死亡率ベースの危険保険料のことで、金額が変動する。契約者の利便性から、終身にわたって保険料を一定金額にしたのが平準保険料。

2.2. 保険金融商品に対応するための機能

ここで、補償部分に対して保険料の変更を伴う異動を行ったときの動きを考えてみます。特に、過去にさかのぼって既存の異動よりも前に異動を行った場合（割り込み異動）はどうなるでしょうか。保険料の変更を伴う割り込み異動を行った時点（過去）から現在までの間に口座側に対する補償の保険料の請求が行われていた場合は、補償側は変更後の保険料で口座側から保険料の引き去りをやり直す必要があります。それによって、口座側は、口座残高やファンドへの投入資金などが変わることになるため、利息計算・利率計算・手数料計算などを

やり直す必要があります。

ファンドの利率は時期によって変動します。また、割り込み異動は常時発生します。割り込み異動による利率への対応を含めた各種計算の再計算の対応は、システム・サポートなしでは不可能なのです。

この割り込み異動を、システム側で整合性を取って実行する機能が、UNDO/REDO機能です。UNDO/REDO機能は、保険金融商品を実現する上では必須です。

3. アプリケーション処理構造の一般形

UNDO/REDO機能を検討する前に、前提となる契約管理アプリケーション処理の一般構造を分析しておきます。ここでは、構造化設計を徹底させて、アプリケーションの種類に依存しない形で、アプリケーション処理構造を構成しています。

3.1. サブシステム化

システムは、幾つかのサブシステムに分解できます。大きなシステムを幾つかのサブシステムに分割することにより、保守開発が容易になります。サブシステムとは、データと機能がまとまっており、そのほかのサブシステムから分離されるものをいいます。通常は、機能とデータに対してCRUD（Create, Read, Update, Delete）分析を行い、アクセスのある部分（CRUDで記述されている）を対角化することにより、サブシステムを抽出することができます。

IAAのデータ・モデルと機能モデルを用いてこのような対角化分析を行うことにより、保険業務全体のサブシステムを抽出したのが、図2です。

本論文では、説明のために次に挙げる四つのサブシステムを取り上げます。

(1)補償管理：保険契約の補償部分を管理する。図2では、新契約と保全がこれに当たる。

(2)契約口座管理：保険契約のお金の蓄積部分についての入出金を管理する。

(3)請求収入管理：保険契約において、その保険契約の収入に当たる保険料についての請求依頼と、それに対する収入結果の管理を行う。

(4)案内収納管理：各保険契約ごとの請求依頼を請求先単位に合算し、その合算結果の案内とそれに対する収納の管理を行う。案内収納管理は業務共通契約管理に属し、保険契約とは別の体系で管理する。

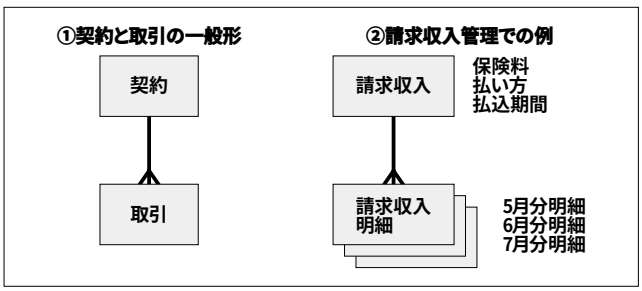

図3. 契約と取引の関係

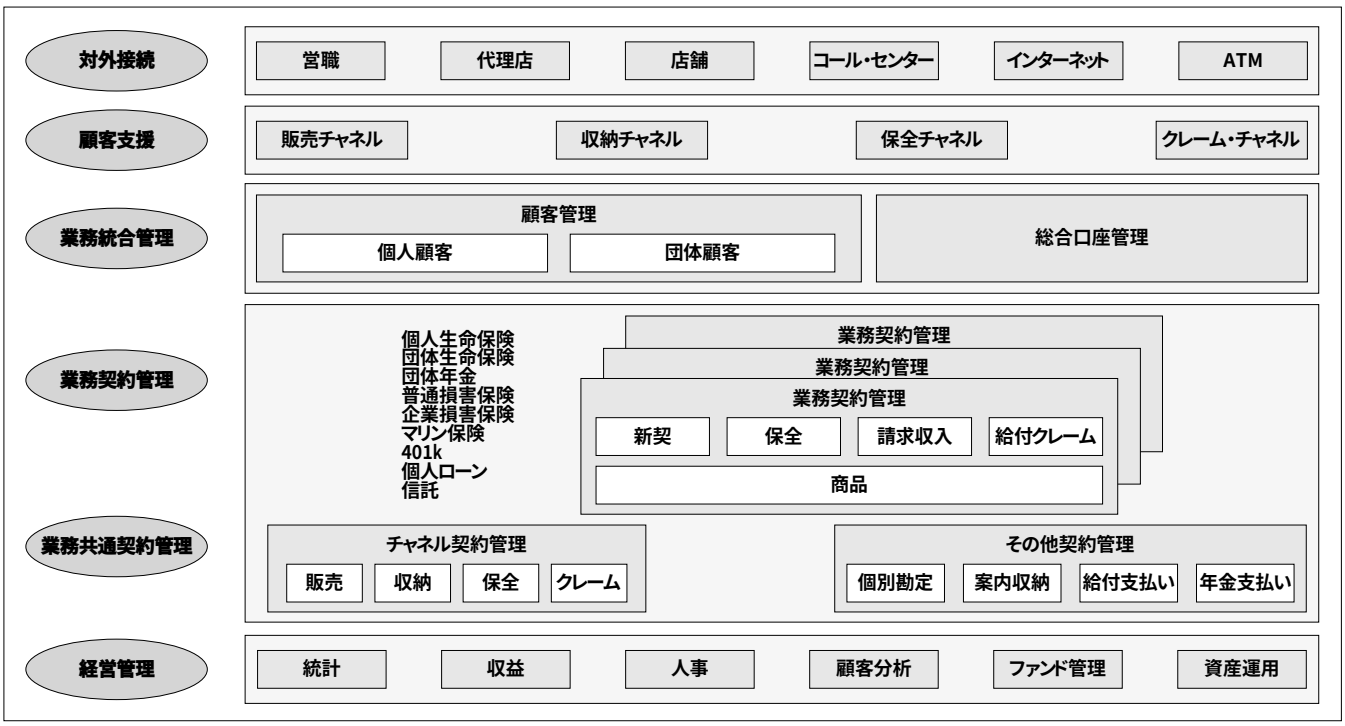

図2. IAAに基づく保険のサブシステム抽出

3.2. サブシステム内の契約管理の構造

サブシステム内の契約管理の構造は、契約と取引に分けられます。契約とは、二者間（保険会社と契約者）で結ばれた、それぞれの権利と義務を明記したものです。また、取引は契約に基づいて発生した処理の履歴です。

例えば、請求収入管理では、契約として保険料の額、支払い方法、払込期間などが定められています。また、取引としては、支払い方法が月払いだとすれば、毎月の保険料請求額、収納額などの明細が、契約に基づいた取引処理の履歴として保持されます（図3）。

取引は、契約の内容に基づいて発生します。従って、契約に対して変更（異動）が行われると、契約の変更に付随して取引の変更が行われます。例えば、請求収入管理の例では、保険料の変更に伴う、毎月の保険料請求額の変更がこれに当たります。

また、契約に対して割り込み異動が行われた場合は、取引についてもその前提条件の契約の内容が変わるので、過去にさかのぼっての取引の再処理（遡及取引処理）を行う必要があります。図3の請求収入管理の例では、6月にさかのぼって保険料変更の割り込み異動を行った場合、6月分・7月分の明細についても、再処理を行う必要があります。

3.3. サブシステム内の処理構造

以上のようなサブシステムが定義されると、その中の処理構造はどのようになるのでしょうか。

各サブシステムは、データとそれを扱うファンクションとに分解されます。3.1節で対角化分析に使用したデータと機能がこれに当たります。ファンクションは、外部から使用されるAPI（Application Programming Interface）とサブシステム内部でのみ使われる機能とに分けられます。前者を、「システム的に動くトランザクションを構成する基本部品的なTR（トランザクション）」という意味で、「基本TR」と呼びます。サブシステムの独立性を高めるために、外部からサブシステムのデータや機能を使用するには、必ず基本TRを経由します。対して、サブシステム内部でのみ使われる汎用ルーチンが、サブシステム内ファンクションに当たります。

複数のサブシステムにまたがって各サブシステムの基本TRをつないで動かすものを、ファンクション・フローと呼びます。ファンクション・フローは、次のようなレイヤーに分かれます。

(1) **プロセス・フロー**：一つの筐体内で、複数の基本TRをつないで処理する。

(2) **アプリケーション・ハブ**：複数の筐体にまたがって、複数のプロセス・フローをつないで処理するものをいう。システム的なトランザクションの受け口となる。

(3) **ワークフロー**：事務処理などで、システム的なトランザクションをつないで処理を行うことをいう。ワークフロー・サポート・システムを使うことも多い。

3.4. バッチ処理のトランザクション化

バッチ処理についても、入力データをトランザクション化しておけば、3.3節の処理構造をそのまま使用することが可能です。プロセス・フロー以下は、オンライン処理とバッチ処理とでは共用することができます。考慮点としては、バッチ処理は一括処理であるため、入力TRをためておく機能が必要です。図5は、

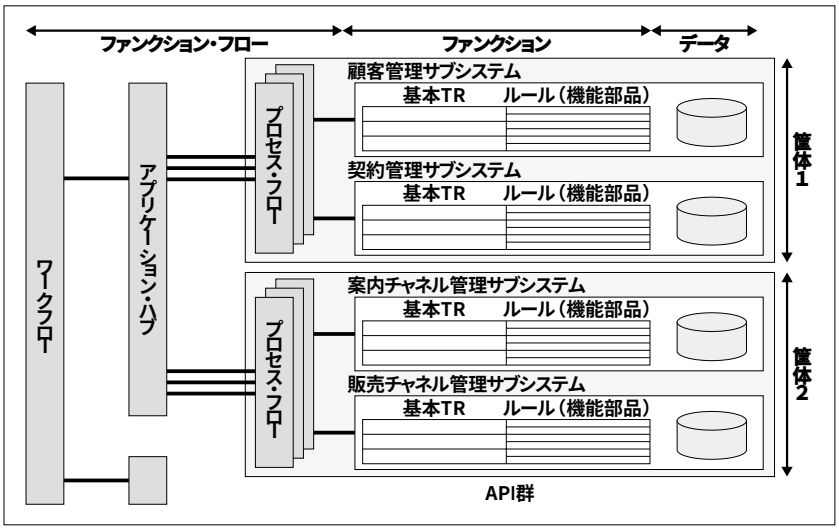

図4. サブシステム内の処理構造

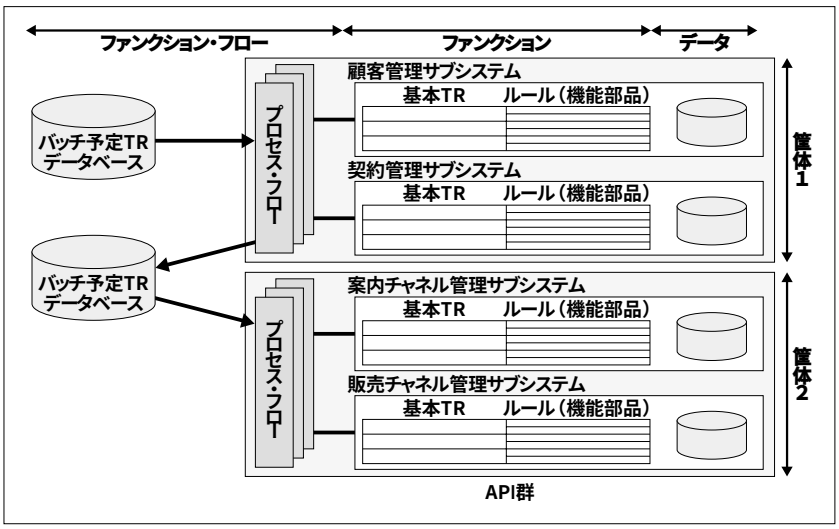

図5. バッチ処理の処理構造

バッチ処理の処理構造を表しています。

図5の中で、バッチ処理用の入力TRをためる機能として設定しているのが、バッチ予定TRデータベースです。バッチ予定TRデータベース上に保管されているバッチ処理用入力TRは、オンライン処理、または別のバッチ処理から内部派生のTRとして作成されます。

図6では、①の処理はオンライン処理、②の処理はバッチ処理です。オンライン処理である新契約登録から内部派生したTRが、バッチ処理である初回保険料請求処理の入力TRとなります。同様に、初回の保険料請求処理で内部派生したTRが、2回目の保険料請求処理の入力TRとなります。各処理で内部派生したTRは、バッチ予定TR上に保管されます。

この内部派生のTRを派生TR、対してオンライン入力のような外部入力のTRをトリガーTRといいます。派生TRは、トリガーTRと区別するために、派生TRを作成した日（＝登録日）を持たせて制御します。

バッチ処理のトランザクション化による利点としては、オンライン処理、バッチ処理とも、入力TRをベースにした共通の制御とすることができるため、オンライン処理との並行稼働の制御が容易になる点が挙げられます。従って、バッチ処理の積み残り対応や24時間対応もしやすくなります。

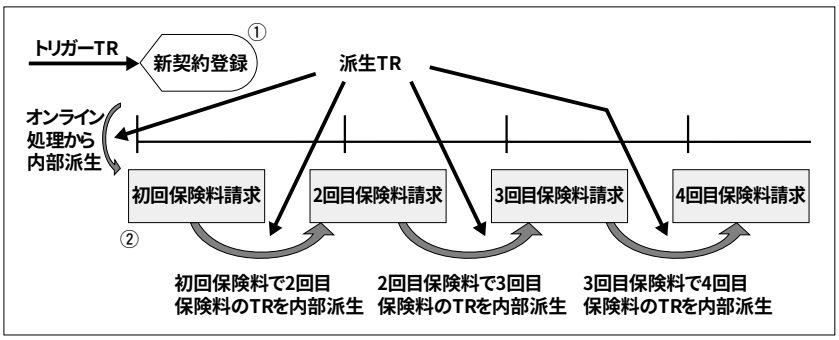

図6. トリガーTRと派生TR

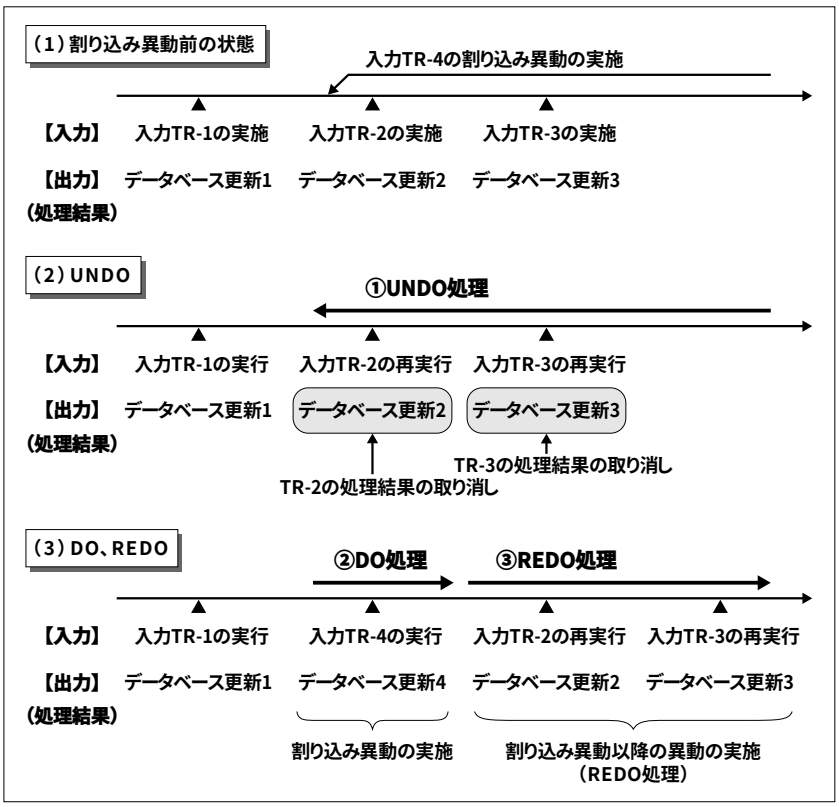

図7. UNDO/REDO機能の振る舞い

4. UNDO/REDO機能の基本機能

4.1. UNDO/REDO機能の基本的な動き

UNDO/REDO機能の基本機能として何が必要かを明確にするために、割り込み異動の場合、UNDO/REDO機能がどのような振る舞いをするのか整理します。

図7のようにある契約で、「入力TR-1～入力TR-3」の異動が行われた状態に対して、「入力TR-1」と「入力TR-2」の間に「入力TR-4」を割り込み異動として行う場合を例にして説明します。

入力TR-4を割り込み異動するには、以下の順序で処理を行います。

①UNDO処理→「入力TR-4」が処理する時点までさかのぼっての処理結果の取り消し

図7の例だと、「TR-2」と「TR-3」の処理結果（データベース更新2とデータベース更新3）を取り消すことにより、「入力TR-1」の処理終了後の時点に処理結果を戻すことです。

②DO処理→割り込み異動の実行

「入力TR-1」の処理終了後の時点に環境を戻した後、「割り込み異動TR-4」を実行し、データベース更新を行います。これにより、処理結果の順番は、「TR-1」「TR-4」となります。

③REDO処理→TR再実行による、取り消した処理結果の復活

「入力TR-2」と「入力TR-3」の処理結果は①の時点で取り消されているため、現在にまで処理結果を戻すには各入力TRを再実行し、再度データベース更新を行う必要があります。ただし、「TR-4」の処理結果が新たに存在するため、「入力TR-2」「入力TR-3」の処理結果は、以前と変わる場合があります。

再実行により、割り込み異動終了後の処理結果の順番は、「TR-1」→「TR-4」→「TR-2」→「TR-3」の順序に変わります。つまり、割り込み異動の場合は、「UNDO処理」→「DO処理（割り込み異動の実施）」→「REDO処理」により、新たな異動処理の順序の整合性が取られることになります。

4.2. UNDO/REDO機能の基本機能

4.1節で記述したUNDO処理とREDO処理の動きに基づいて、UNDO/REDO機能として必要な基本構造を考え、そこから基本機能を洗い出します。

まず、処理の基本構造として以下のようなモデルを想定します（図8）。

UNDO処理は、ある時点までの処理結果を、すべて取り消すことです。図8の例では、データベースと計上データが処理結果に当たります。

処理結果を取り消すといっても、どの時点までが取り消し対象になるかは、割り込み対象TRが来るまで分かりません。そのためには、処理結果については、各入力TRごとの更新内容を履歴として保管（ヒストリー化）する必要があります。データベースのようなストック系のデータについては、更新内容をヒストリー化して保管するのは当然ですが、計上データのようなフロー系のデータについても、データベースと同様にヒストリー化する必要があります。それは、業務処理に対してその計上データが取り消されたことを知らせるために、計上データのヒストリーを基に取り消し用計上データを作成する必要があるからです。

REDO処理は、UNDO処理で取り消した処理結果を復活させるために、再度入力TRを実行し直すことです。そのためには、過去

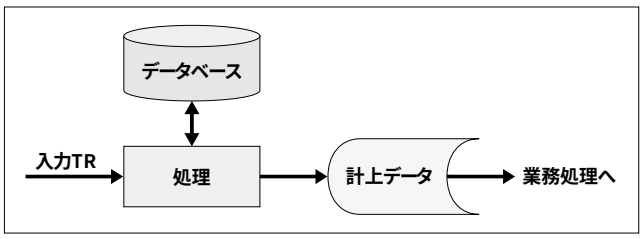

図8. 処理の基本構造

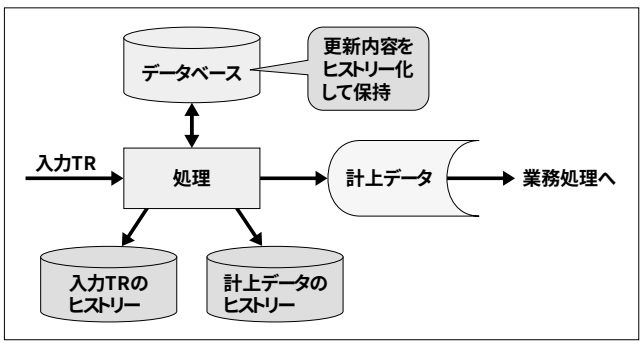

図9. UNDO/REDO機能を行うための基本構造

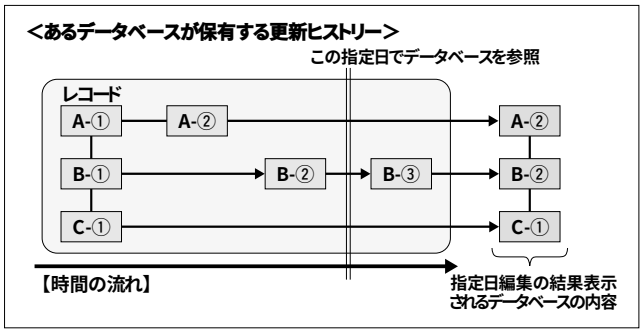

図10. 指定日編集機能の概要

の入力TRを時系列化して保管（ヒストリー化）する必要があります。

以上のことから、UNDO/REDO機能を行うための基本構造は図9のようになります。

前述したUNDO/REDO機能の基本構造を制御するために必要な基本機能を洗い出すと、次のようになります。

（1）データベースの指定日編集機能

UNDO処理を行うには、データベースをヒストリー化する必要があります。データベースをヒストリー化した場合には、指定した日（指定日）でデータベースの内容を参照するために、特殊な参照機能を準備する必要があります。

図10のように、データベースは別々の実行日で各セグメント内容を更新し、その内容をヒストリー化して保有します。都合良く指定日の当日にすべてのセグメントを更新したヒストリーが存在するとは限らないので、指定日を基準にして各セグメントの最新の更新ヒストリーを選択し、その更新ヒストリーの内容を編集して指定日のデータベース内容として表示する必要があります。

（2）入力TRヒストリー管理

入力TRをヒストリー化して保管するために必要な機能です。入力TRは、処理の終了後にこのデータベースに保管します。REDO処理時として再実行するTRは、当データベース上から選択して使用することになります。

（3）計上データ・ヒストリー管理

計上データをヒストリー化して保管するために必要な機能です。計上データはフロー・データなので、計上データを作成するのと同時に、計上データと同一の内容を当データベースに保管します。UNDO処理時に出力する取り消し用計上データは、当データベースに保管した計上データを基に作成します。

4.3. 従来の割り込み異動との違い

ここでは、UNDO/REDO機能の基本機能についてまとめてみましたが、従来の割り込み異動とはどのような点で異なるのでしょうか。

従来の割り込み異動の特徴を挙げてみると、次のようにな

ります。

(1)生命保険の場合：データベース、計上データ、入力TRともにヒストリーは持たない。

－割り込み異動の結果を外部（例えば手処理など）で作成しておいて、その結果を最新の内容としてデータベースを更新する。

－計上データの取り消し、再計上は、ロジックによって復元する。

(2)損害保険の場合：データベースのみヒストリーを保有。計上データ、入力TRについてはヒストリーを持たない。

－指定日編集で参照したデータベースの内容に対して、割り込み異動を行う。ただし、入力TRヒストリーはないため、割り込み異動以降のデータベースの内容は変更されない。

－計上データの取り消し、再計上は、データベース上のヒストリーを用いて、ロジックによって復元する。

最大の違いは、UNDO/REDO機能が、過去に処理を行ったTRを保存しておいて再実行するという点です。

単に過去の処理結果にすぎない更新ヒストリーを基にして、その時点の内容を作成するだけの従来の割り込み異動では、対応できる範囲に限界があります。責任準備金の計算を例に挙げても、従来の固定利率の商品ならまだ対応できるかもしれませんが、金融商品特有の変動利率による責任準備金の再計算については、とても複雑すぎて対応することは困難です。

UNDO/REDO機能ならTRの再実行を行うため、責任準備金の計算を含むすべての処理を行い、新たな処理結果によるデータベースの更新と計上データの再作成を自然に行うことができるのです。

5. UNDO/REDO機能の拡張機能

実際の保険システムにUNDO/REDO機能を実装する場合は、基本機能だけでは対応できない部分が存在します。本章では、その部分をUNDO/REDO機能の拡張機能として述べていきます。

5.1. 複数サブシステム対応の課題

システムは、複数のサブシステムから構成されています。従って、入力TRは複数のサブシステムにまたがって処理を行わなければならない場合があります。このような場合、UNDO/REDO機能についても複数のサブシステムにまたがることを考える必要があります。また、サブシステムに含まれる処理には契約処理と取引処理があるため、複数サブシステム対応を検討する場合には、このことも念頭におく必要があります。

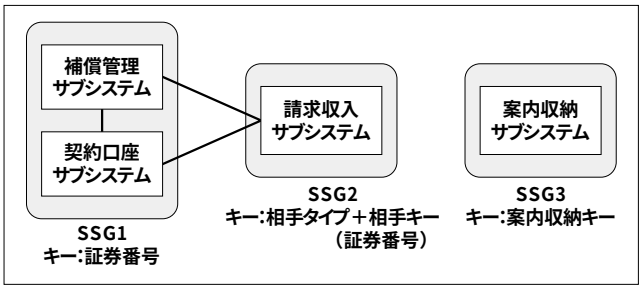

図11. サブシステム・グループの例

UNDO/REDO機能を複数のサブシステムにまたがって行うには、「サブシステムごとにキー体系が異なる」という問題があります。

UNDO処理、REDO処理を行う場合、入力TRが保有するキー体系を基に、UNDO処理時のデータベースの削除、REDO処理時の再実行用TRの収集などを行います。入力TRと同一のキー体系を持つサブシステムのみが、UNDO処理とREDO処理の対象であるなら、入力TRが保有するキー体系を使用してUNDO処理とREDO処理を行えばよいため、特に問題はありません。

ところが、入力TRの処理がキー体系の異なるサブシステムにまたがる場合、入力TRのキー体系とは異なるキー体系を保有するサブシステムに対して、どこまで整合性を取りながらUNDO処理とREDO処理を制御できるかについて考える必要があります。

ここでは、サブシステム・グループという概念を取り入れて、複数のサブシステムにまたがるUNDO/REDO機能がどこまで可能かを検討します。

5.2. サブシステム・グループ（SSG）

UNDO/REDO機能はキー体系を基に制御を行うため、同一のキー体系を保有するサブシステムは、一つのグループと見なして制御できます。従って、複数のサブシステム間のUNDO処理とREDO処理を考える場合、個々のサブシステム同士の関係で検討するのではなく、同一のキー体系を持つサブシステムを一つのグループとしてまとめた方が、より汎用的に考えられます。この同一のキー体系を保有するサブシステムのグループを、サブシステム・グループ（SSG）と名付けます。

図11は、サブシステム・グループ（SSG）の参考モデルです。この例では、キー体系が異なるため、SSGを「SSG1～SSG3」の三つに分けています。また、補償管理サブシステムと契約口座サブシステムは同一のキー体系（証券番号）を保有するため、同一のSSGにカテゴライズしています。

5.3. 同一SSG同士の場合の検討

この場合は、UNDO処理およびREDO処理上、特に問題は

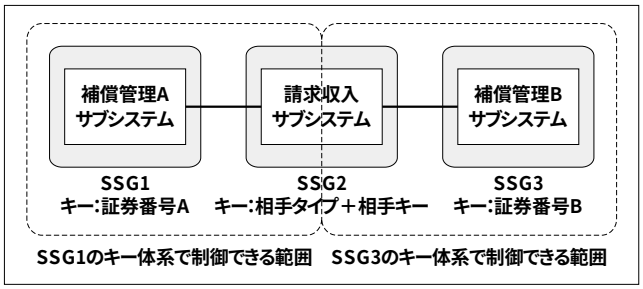

図12. SSGが関連する場合の例

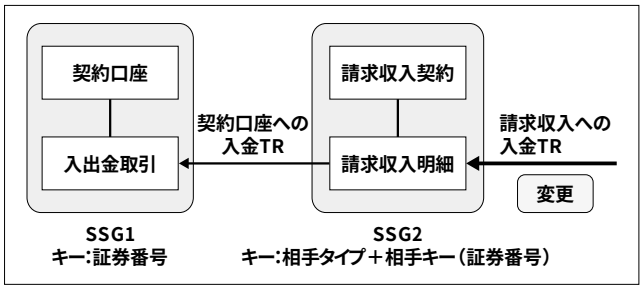

図13. サブシステムまたがりの取引処理

ありません。

(1)SSGが同一(＝データベース上のキー体系が同一)であるため、双方のサブシステムに対して同時にUNDO処理を行うことは容易です。

(2)REDO処理において、TRヒストリー・データベースからREDO用のTRを収集することも、同一のキー体系で収集できる(つまり、サブシステムは異なっていても同一キー体系のため、同一のサブシステムと見なすことができる)ため、容易です。

5.4. 関連するSSG同士の場合の検討

サブシステム化する目的の一つは、システムを分解してカプセル化することにより、複数のサブシステムから利用されることを可能にすることです。図12の例では、請求収入管理サブシステムのキー体系は、複数のサブシステムから利用できるように「相手タイプ＋相手キー」となっています。また、この例では、請求収入管理サブシステムを利用しているサブシステムは「補償管理A」と「補償管理B」です。キー体系は双方で異なっているため、それを受けて請求収入管理サブシステムの相手キーは相手により「証券番号A」または「証券番号B」となります。この例のように、SSG同士のキー体系が関連している場合は、相手のSSGを自分の一部と見なして、相手のSSGに属するサブシステムを関連付けて(支配下において)使用することが可能となります。図12の例でいえば、SSG1のサブシステムは、SSG2の配下のサブシステムが保有するデータベースを関連付けて(支配下において)使用することが可能です。ただし、使用範囲は「SSG1のキー体系で制御できる範囲」に限られます。

(1)UNDO処理に関する検討

複数のSSGがある場合、自分が属するSSGとは別のSSGに対して同時にUNDO処理を行うように制御することは、次の理由により可能です。

あるSSGに属するサブシステム(例では、補償管理サブシステム)は、自分が関連(支配)しているサブシステム(例では、請求収入管理サブシステム)を知っており、さらに別のSSGである請求収入管理サブシステムのキー体系の一部に、補償管理サブシステムのキー体系が組み込まれていることを知っています(そうでなければ、自分の一部と見なして利用することなどできないはずです)。

従って、自分が属するSSGに対するUNDO要求と同時に、関連するSSGに対してもキー体系が判明しているため、UNDO要求を出すことができます。相手側のSSGとしてもUNDO要求が来た時点で、相手のSSGおよびSSGのキー体系が判明するため、そのキー体系を基にして相手側のSSGに関するUNDO処理を行うことができるのです。

(2)REDO処理に関する検討

キー体系が異なるため、TRヒストリー・データベースはSSG単位で物理的に異なったものとして存在します。従って、REDO処理において、TRヒストリー・データベースからREDO用のTRを収集する場合、SSG1のキー体系による収集だけでなく、SSG2のキー体系による収集も同時に必要となります。

この場合でも、SSG1からSSG2に対するUNDO要求により、SSG2側も親のSSG、および親のキー体系が判明しているため、そのキー体系を基にしてSSG2側として必要なREDO用のTRを収集することが可能となります。

(3)取引処理に関する検討

取引処理がサブシステムにまたがって行われる場合を検討します。

図13では、請求収入管理サブシステムへの入金処理が、契約口座サブシステムへの入金TRを派生し、契約口座の入金処理はその派生TRを入力TRとして処理をしています。

この例で、請求収入への入金TRに対して過去にさかのぼっての変更が行われたとします。

請求収入の処理結果(入金取引)と契約口座の処理結果(入金取引)のUNDO処理は、左記の(1)の理由により同時に行うことが可能であり、かつ行わなければなりません。その理由は、契約口座の入金取引は、請求収入の入金取引の結果を受けて行われるため、請求収入側に変更が入った場合は、その変更を反映して契約口座の入金取引を行わなければならないからです。

REDO処理では、取引処理の入力TRの収集時に再実行対象TRと非対象TRの選択が必要になります。

図14を参照してください。①の時点で割り込み異動が行われた場合、TRヒストリー・データベースから再実行用のTRを収集

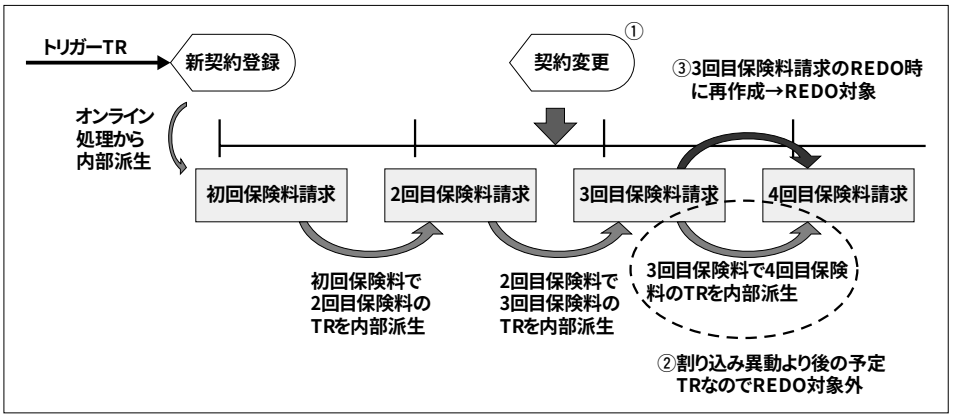

図14. 派生TRの差し替え

する際に、②の「4回目保険料請求」のTRはREDO対象には入りません。その理由としては、そもそもの処理の流れとして「4回目保険料請求」のTRは、「3回目保険料請求」の処理の結果として自然に作成されるからです。「3回目保険料請求」処理が再実行されることにより、再度「4回目保険料請求」のTRは再作成され(③)、実行されることになります。

このように取引処理の間をつなぐ派生TRにおいて、UNDO処理時点から後に作成されたTRは、REDO処理には不要です。従って、派生TR上に保有する「登録日」を使用し、UNDO処理時点より後に作成された派生TRなら、REDO対象外とする制御を行う必要があります。

6. UNDO/REDO機能の実装

6.1. UNDO/REDO機能の処理構造

前述したUNDO/REDO機能の基本機能と拡張機能を前提にした場合、システムへの実装時にはどのような構造にすればよいかを検討します。

UNDO処理での実装すべき機能としては、次のものが挙げられます。

(1)UNDO対象のSSGを判定する機能
(2)データベースの取り消し機能
(3)取り消し用計上データの作成機能
(4)REDO対象TRの収集機能

次に、REDO処理で実装すべき機能としては、次のものが挙げられます。

(1)REDO対象TRのため込み機能
(2)REDO対象TRの実行順序決定機能
(3)REDO対象TRの実行機能

さらに、UNDO/REDO機能とは直接には関係ないのですが、実行対象のTR収集という点から処理構造を考える上で、積み残りのバッチ予定TRの収集機能が必要です。

この機能は、入力TRより以前の実行日を持つバッチ予定TRがバッチ予定TRデータベース上に残っていた場合、そのTRを今回の実行対象のTRとして収集する機能です。なぜなら、この状態は、先に処理されるべきバッチ予定TRがまだ残っている状態と考え、処理順序の整合性を取るためには、先にこのバッチ予定TRを処理する必要があるからです。

以上の各機能を処理構造としてまとめると、次のようになります(図15)。

①UNDO CTL機能

入力TRが属するサブシステムから、UNDO処理対象のSSGを判定し、対象SSGごとにUNDO要求を出します。UNDO要求として渡す主な情報としては、入力TRのキー体系、入力TRの実行日があります。

②データベースの取り消し機能

UNDO要求を受けて、SSG単位に配下のサブシステムが保

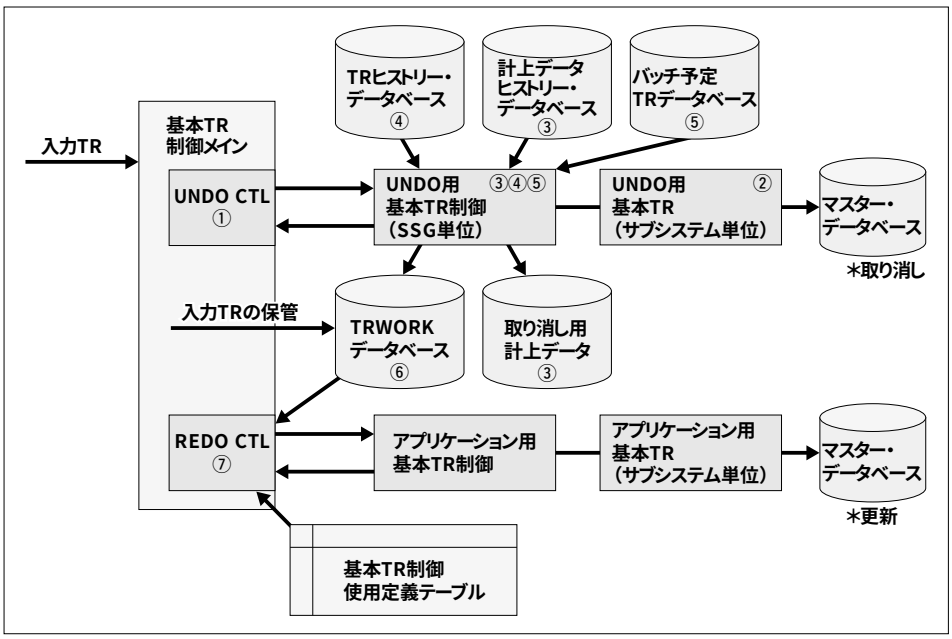

図15. UNDO/REDO機能の処理構造

有するデータベースの処理結果の取り消しを行います。個々のデータベース取り消し機能は、サブシステム単位のファンクション（基本TR）です。そのデータベース取り消し基本TRをSSG単位にまとめて制御する機能は、SSG単位のプロセス・フロー（基本TR制御）として構築します。

③取消用計上データの作成機能

UNDO CTLからのUNDO要求を受けて、現時点から入力TRの実行日までの取消用計上データを計上データ・ヒストリー・データベースから作成します。

④REDO対象TRの収集機能

UNDO CTLからのUNDO要求を受けて、現時点から入力TRの実行日までのREDO対象TRをTRヒストリー・データベースから収集します。その場合、該当するトリガーTRについてはすべて収集します。また、派生TRについては登録日と入力TRの実行日との比較により、REDO対象分のみを選択して収集します。

⑤バッチ予定TRデータベースからの積み残りTRの収集機能

入力TRの実行日よりも以前の実行日を持つバッチ予定TRを、バッチ予定TRデータベースから収集します。

③④⑤の機能については、対象データベースがSSG単位のデータベースであるため、SSG単位で制御することになります。従って、プロセス・フロー（UNDO用基本TR制御）の一部として構築します。

⑥REDO対象TRのため込み機能

UNDO処理により収集されたREDO対象TRは、TRため込み用のデータベースに展開し、制御する必要があります。TRため込み用のデータベースを、「TRWORKデータベース」と名付けます。TRWORKデータベースには、UNDO処理時の収集したREDO対象TRだけでなく、REDO処理中に再度派生したTRについてもREDO対象TRとして保管します。

⑦REDO CTL機能

REDO CTLには、2種類の機能があります。

一つは、REDO対象TRの実行順序の決定です。TRWORKデータベースの中には、TRヒストリー・データベースから収集してきたTRや再実行中に派生したTRが、REDO対象TRとして存在します。そのTRをREDO処理する場合、各TRの保有する実行日をキーにして時系列化し、最も古いものから順番にREDO処理を行うという制御をします。

もう一つは、REDO対象TRの実行です。

REDO対象TRの実行とは、具体的にいえば、REDO対象TRが保有する情報（例：トランザクション・コード）を基に対応するアプリケーションのプロセス・フロー（基本TR制御）をCALLし、REDO対象TRが保有するデータをINPUTデータとして渡し、再実行処理を行う制御のことです。この方法により、DO処理時と同一のアプリケーションをREDO処理時にも使用することが可能になります。この場合、「トランザクション処理コードとプロセス・フロー（基本TR制御）の関係」は、テーブル化してプロセス・フロー（基本TR制御）の独立性を損なわないようにすることが必要です。

さらに、入力TR自身についても、TRWORKデータベースに保管することにより、REDO対象TRと同一の仕組みで実行できます。特に、割り込み異動の入力TRの場合、実行日は最古の実行日となるため、「割り込み異動の実行→REDO対象異動の実行」という制御が自然とできることになります。

この処理構造図の中では、UNDO CTL機能とREDO CTL機能については、プロセス・フロー（基本TR制御）を制御する機能であると考え、プロセス・フローを2層化し、上位のプロセス・フロー（基本TR制御メイン）として位置付けています。

7. おわりに

本論文では、金融商品化された保険に対する契約管理アプリケーションという観点で、特にUNDO/REDO機能の検討、システムへの実装方法を中心に論じてきました。まだ課題はありますが、今後はそのような課題を順次検討し、保険金融商品のみではなく複合金融商品に対応した契約管理アプリケーション・システムについての概念から実装方法までを確立させていきたいと考えています。

（ページ数および表記上の観点から、著者の了解を得て編集部にて手を入れてあります）

[参考文献]
[1] マイケル・ポーター『競争優位の戦略』ダイヤモンド社、1985年
[2] 刈屋 武昭『金融工学とは何か―「リスク」から考える―』岩波書店、2000年
[3] 加藤 修『現代保険概論』中央経済社、2000年