SBI 21 におけるオープン環境を利用した開発基盤

The Application Development Environment using Open Products in SBI 21

奥 野 信 幸

要 約 SBI 21 は，地域金融機関向け次世代勘定系システムパッケージであり，専用の開発環境を保有している．この開発環境は，PC 上で開発したプログラムを実行環境であるメインフレーム上でも稼働させるという考え方に基づいて設計されている．また，SBI 21 は，オブジェクト指向技術を前提としたアプリケーション構造になっているため，本開発環境は，それらを支援する機能や，PC 上でのテスト実行を支援する機能を提供する．これにより，従来開発負荷の大きかったテスト工程において，テスト検証負荷の軽減や仕様変更にともなう手戻り工数の削減といった効果を上げている．

本稿では，この PC 開発環境の特長および機能に関して説明するものである．

Abstract SBI 21 (Strategic Banking Integrated-system for 21st century) is the new banking system solution package for regional financial institutions and has an attractive development environment. Its development environment is designed on the basis of the concept that programs implemented on workstation and/or personal computers will be running on Series 2200 execution environment without modifications.

In addition, SBI 21 has the application structure based on the object-oriented technology, thus its development envitonment provides various capabilities to support the object-oriented technology and program testing processes in the purpose of quality assurance. These capabilities bring enormous effects in decreasing testing and verification works, and eliminating the iteration round phases due to specification changes in a software development cycle.

The paper describes features and capabilities of the development environment running workstations and/or personal computers.

1. はじめに

SBI 21（Strategic Banking Integrated-system for 21st century の略）は，地域金融機関向け次世代勘定系システムパッケージであり，2001 年 1 月カットオーバを目指し，現在開発中のシステムである．このパッケージは，金融ビッグバンに代表される環境変化に迅速に対応できるシステムを目指しており，保守性の向上を考慮した，従来の勘定系パッケージとは異なる新しい発想のアプリケーション構造を有している．本システムでは，このアプリケーションを開発，保守するための専用の開発環境を独自に開発した．この開発環境の特長は，大きく三つある．

1） 本システムの実行環境は，2200 系メインフレームであるが，開発環境構築に際しては，2200 系メインフレーム用の開発環境を PC 上に構築するという考え方をとらず，あくまで，PC 上で製造からテスト確認まで実施したプログラムを実行環境用にコンバートして実行環境でも稼働させるという考え方をとっている．従って，将来，本システムの実行環境が UNIX などのオープン環境に変更になったとしても，開発環境自体に大幅な変更が発生しない構造となっている．

2） 本システムは分析/設計工程にオブジェクト指向技術を採用しているが，開発言語にはオブジェクト指向技術を支援する機能を有していない COBOL を採用している．従って，単純に分析設計結果をプログラム作成に結び付けることはできないため, 本開発環境ではオブジェクト指向技術を支援する機能を提供している．

3） 本開発環境は，品質に直結し，作業負荷の大きいテスト工程を支援する機能に重点を置いている．単体テスト用の専用ローカルデータベースを保有し実行環境をエミュレートすることにより，PC 上での単体テストを可能としている．また，テスト実行時の期待値とテスト結果との比較機能などがあり，更にテストデータに関しては，前回テストデータを保存し，それを再利用してテストを自動実行する機能など充実した機能を提供しており，作業負荷の軽減を図っている．

本稿は，この PC 開発環境の特長及び機能に関して説明するものである．

2. SBI 21 のアプリケーション構造

本開発環境は，SBI 21 のアプリケーション構造に密接に関係している．従って，まず簡単にこのアプリケーション構造に関して記述しておく．

本システムのアプリケーション構造は，部分的/段階的な書き替えが可能な構造を目指している．具体的には，ある業務の括り（本システムでは,「サブシステム」と呼ぶ）の単位で独立しており，そのサブシステムで取り扱うデータと，その操作がサブシステム内にカプセル化された構造になっている（図 1）．

この構造を実現するため，本システムでは，オブジェクト指向技術の考え方を採用し，特に，オブジェクト指向で言うところのカプセル化の概念を取り入れ，データベースの参照制約を独自に設定した．これにより，各サブシステム間は入出力パラメタ

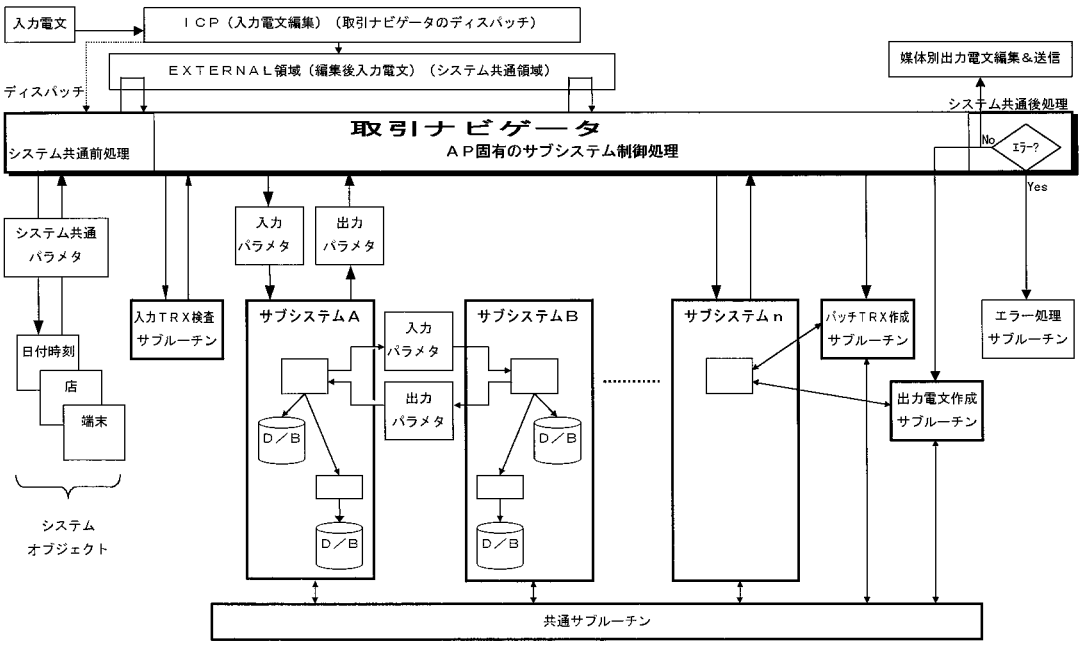

図 1　オンラインシステムのアプリケーション構造概念図

だけを経由して情報交換を行うため，入出力パラメタさえ保証していれば，各サブシステム内部の実装形態が変化しても他のサブシステムに影響を及ぼすことはない．また，サブシステム内にカプセル化されたデータベースは，サブシステムの外側から直接アクセスできない構造であるため，サブシステムの外側にあるプログラムは，サブシステム内のデータベースの構造を意識する必要はなくなる．

図 1 に示したようにオンラインシステムのアプリケーションは，勘定系システムをそれぞれの業務の役割に対応して数 10 個のサブシステム（例：普通預金，当座預金，定期預金，定期積金，証書貸付，割引手形等）に分割した構造となっている．

一つの取引（例：普通預金の入出金）の処理は，これらのサブシステムを組合せ，制御しながらおこなうことになるが，本システムでは，「取引ナビゲータ」と名づけた制御プログラムがその役割を担う．

この構造の詳細な説明，並びにこの構造が考え出された背景等については，本書掲載の「オブジェクト指向技術による勘定系システムのリエンジニアリング」を参照されたい．

3. SBI 21 の開発環境の概要

3.1 開発環境の考え方

従来の開発環境は，特に実行環境がメインフレームの場合，その実行環境向けに開発/提供されたものが多く，実行環境が変更になれば，それに対応した開発環境を新たに用意するということが多かった．これに対し，本開発環境は，PC（Windows NT）上でプログラム作成/単体テスト工程まで実行し，実行確認がとれたプログラムを実行環境用に変換して稼働させるという方針で開発されている．この場合，Windows NT と実際の実行環境との相違を吸収する必要があるが，それは，ソースコード変換機能や実行環境のデータベースをエミュレートする機能等で対応している．

従って，本システムは，現時点では，実行環境は当社のメインフレームを想定しているが，今後の技術動向によって実行環境が UNIX 等のオープン系プラットホームになった場合でも，開発環境自体は最小限の変更で対応でき，かつ，作成されたプログラムも極力そのまま使用できるような工夫を施している（図 2）．

3.2 開 発 方 法

本開発環境は，全ての機能を一から開発しているわけではなく，いくつかは，市販ツールを利用することで，開発工数の削減，開発時期の短縮を図っている．例えば，単体テスト支援機能のうち，コンパイル/デバッグ部分については，MF　COBOL の GUI 機能をそのまま利用し，MF　COBOL では提供されていないテストデータの作成機能やテスト実行時の期待値と実際のテスト結果とのコンペア機能といった補完機能を独自に付加している．

独自に開発した機能の大部分は，Delphi を使用して開発している．これは，GUI の開発言語として操作性にも優れ，実績もあり，一般的な評価も高いことによる．また, オブジェクト指向技術を前提とした言語であることも採用した要因の一つである．また，後述する単体テスト支援機能のうち，テストデータを入力する部分については EXCEL を使用している．これは，新たにデータ入力画面を作成しなくても，EXCEL

◆ Windows NT環境で作成したプログラムをメインフレームへ

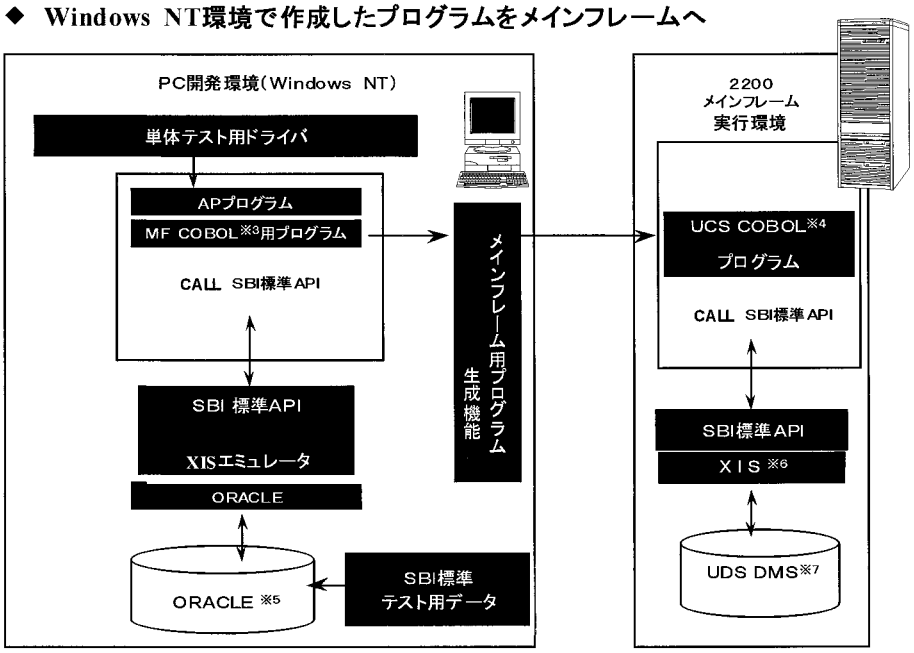

図 2　SBI 21 の開発環境の考え方

の基本機能/操作性で充分データ入力画面の機能を満たしていることと，EXCEL で作成されたデータそのものが，2 次加工しやすいという利点によるものである．また，Delphi との相性もよく，非常に扱いやすいソフトウェアであったことも理由の一つである．

3.3 全体概要図

本開発環境は，各開発工程に対応したツールを提供している．表 1 がその機能一覧である．

表 1　工程と開発環境の機能一覧

工程	開発環境の機能
分析／設計工程	オブジェクトモデル図作成機能 リポジトリエクスプローラ プログラムスケルトン／構造体生成機能
プログラミング工程	ＰＡＤ描画機能 規約原則チェック機能
単体テスト工程	テストデータ／テスト環境作成機能 テスト実行支援機能 テスト結果コンペア機能 テストデバッグ機能 Automated Testing 機能
結合・総合テスト工程	実行環境支援機能
保守工程	プログラム管理機能

図 3 は，各ツール間の関係を表す全体概要図である．図からもわかるように，各工程でツールから作成される成果物は，次工程の入力成果物になっている．例えば，上

流工程で分析設計したオブジェクトの定義情報は，専用のリポジトリで管理され，その情報から次工程（下流工程）の入力成果物となるプログラムのスケルトンや構造体が生成される．また，実行環境支援機能により，Windows NT 上で単体テストまで実行したプログラムから，実行環境上で稼働可能なプログラムが生成され，自動転送される．

各機能の詳細な説明については，次章以降で説明する．

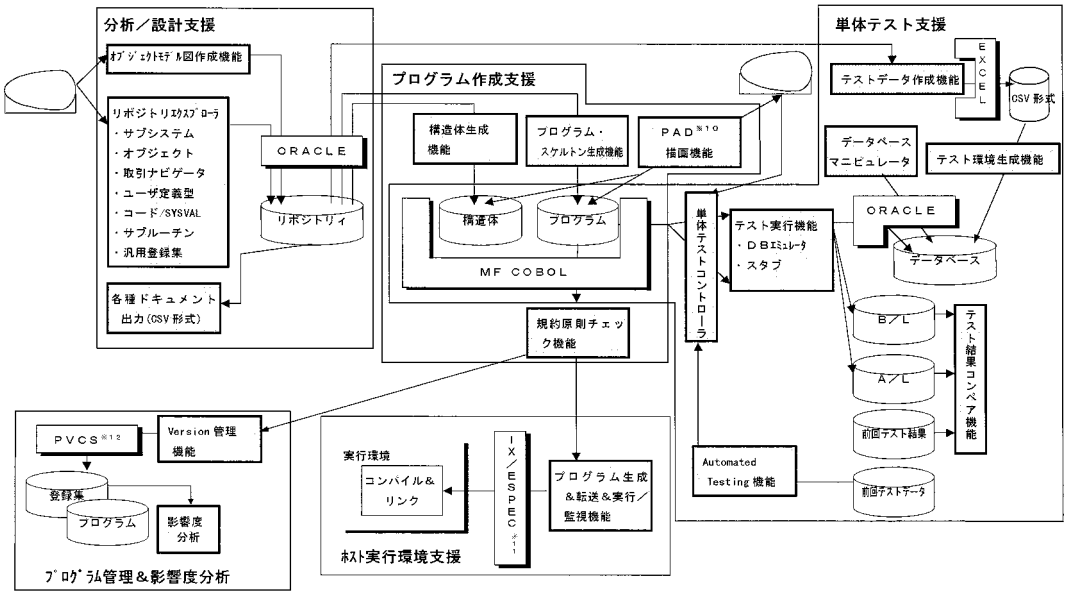

図 3　SBI 21 開発環境全体概要図

4.　分析設計工程支援

分析設計工程を支援する機能はモデリングを支援する部分とオブジェクトを定義する部分に大きく分かれる．

4.1　モデリング支援

オブジェクト指向技術では，その役割と責任をもとにオブジェクトを導出するところから始まるが，その際，導出されたオブジェクト間の静的関連や動的関連をモデリングしていくことが重要な作業となる．

本開発環境では，静的モデルを作成するツールを提供し，モデル作成時の作業を支援している．

1)　オブジェクトモデル図作成機能

オブジェクトの静的モデルの表現方法には，さまざまな表記法が存在しているが，SBI 21 は，OMT（Dr. James Rumbaugh が主唱しているオブジェクト指向技法）ベースに独自のオブジェクト表記法を定めている．OMT の場合，オブジェクト間の主な関連は，図 4 のように表記される．しかしながら，この表記法のままでは，オブジェクト間のメッセージパスの方向性が不明確であるため，本シ

ステムでは，オブジェクト間のメッセージの方向性を示す矢印（→）を新たに追加している．

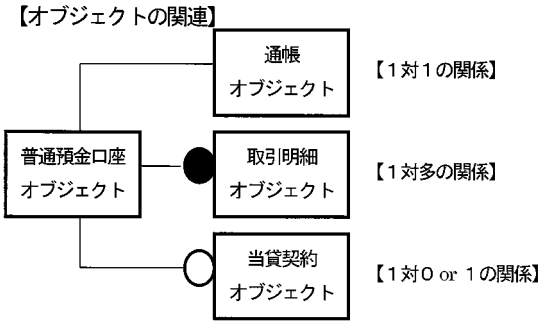

図 4　オブジェクトモデルの表記法

オブジェクトモデル図作成機能は，この表記法をもとにモデル作成を支援するツールで，オブジェクトの役割や関係を表わす罫線処理を単に市販のワードプロセッサで作成するよりも，はるかに簡単に作図できる．特に，モデルの追加・削除が発生した場合，市販のソフトウェアでは一つ一つの図や罫線を手動で整形し直す必要があるが，本ツールの場合，1オペレーションで，図を適正な箇所に自動整形する機能を有しており，変更負荷の軽減を図っている．

図5がオブジェクトモデル作成機能の定義画面とそれにより作成したオブジェクトモデルの例である．

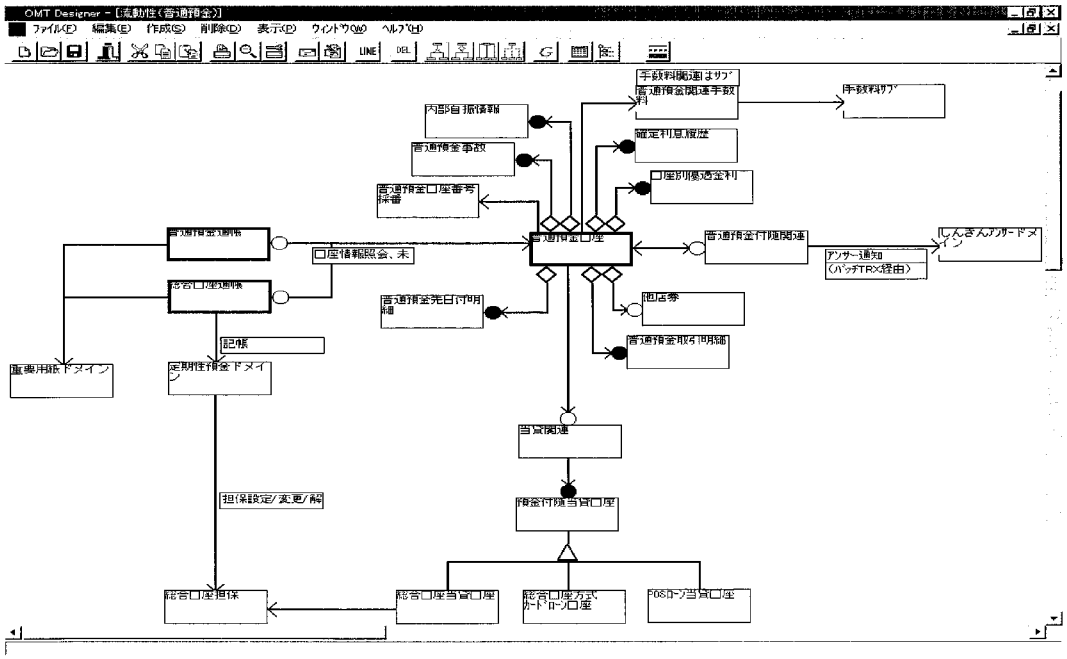

図 5　オブジェクトモデル図作成機能の定義画面

2) 動的モデルの作成機能

本システムでは，動的モデル図としては，Jacobson のインタラクション図を採用しているが，この部分に関しては，特にツールは用意せず，EXCEL を使用して記述することにしている．これは，インタラクション図の場合，EXCEL の機能で必要とする作成機能がほぼ充足できるからである．

4.2 設計情報の登録

1) オブジェクトの登録

オブジェクトの導出を終えると，次に各オブジェクトの属性と外部インタフェースを定義しなければならない．SBI 21 の場合，属性は，データベースに格納されるレコード定義であり，外部インタフェースとは各オブジェクトがもつサービスとそのサービスに必要な入出力パラメタのことである．

本開発環境では，これらのオブジェクトの設計情報を専用のリポジトリに，「リポジトリエクスプローラ」と呼んでいるツールで定義し，情報を一元管理している．

オブジェクトを登録する場合は，オブジェクトの集合体であるサブシステムを事前に登録しなければならないが，この定義もこのツールで同様に定義可能である．

図 6 がリポジトリエクスプローラの定義画面である．図からもわかるように，左ペイン上にサブシステム－オブジェクト－属性，入出力パラメタといった個々の構成要素を木構造図で表示し，右ペインにそれぞれの定義画面を表示する構成になっている．また，定義画面は，複数画面開くことができ，画面間でのコピー＆ペーストも可能な仕組みとなっている．この画面構成は，リポジトリエクスプローラ以外の他のツールについてもほぼ同じ構成で統一している．

また，このリポジトリエクスプローラは，データ入力時に，本システムの各種規約に違反していないかを内部的にチェックしている．

2) リポジトリからの Import/Export 機能

リポジトリに定義した情報は，本システムで定義している EXCEL ブック形式の外部仕様書へ Import/Export が可能であり，リポジトリとドキュメントの 2 重メンテナンスが発生しない仕組みになっている．また，更に定義情報を CSV 形式のテキストとしても Export 可能であり，ユーザ側で簡単な EXCEL マクロを作成すれば，独自のプログラム一覧表や管理帳票なども作成可能である．

3) ユーザ定義型

各オブジェクトの属性や入出力パラメタを定義する場合，そのデータ項目の型・桁を定義しなければならない．一方，本システムのアプリケーション構造では，属性はカプセル化されており，属性に定義されているデータ項目をオブジェクトの外側から参照するためには，そのデータ項目を入出力パラメタに定義し，それを経由して受渡しする必要がある．すなわち，一つのデータ項目は，属性や入出力パラメタの両方に定義される場合がある．従って，そのデータ項目の型・桁に変更が発生した場合，関連する全ての定義を同期をとって修正しなければならない．このため，本開発環境では，予め，型・桁を「ユーザ定義型」と呼んで

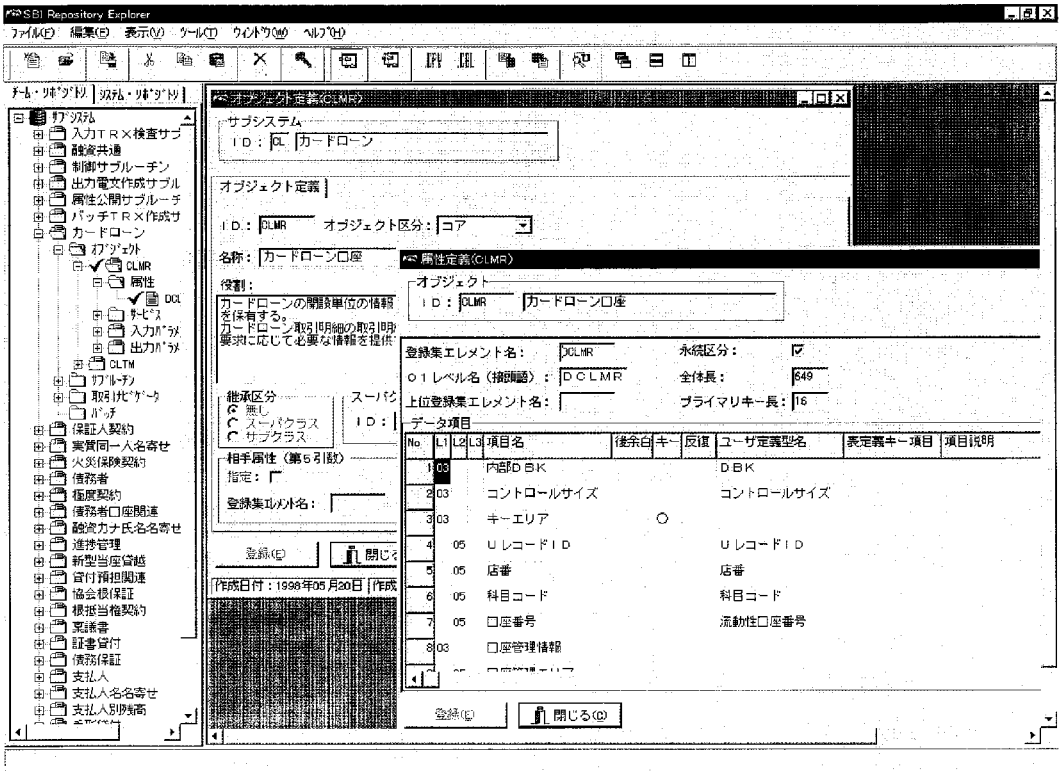

図 6　オブジェクトエクスプローラの入力画面

いる専用の定義情報として登録しておき，個々の属性や入出力パラメタのデータ項目を定義する際には，型・桁を直接定義せず，このユーザ定義型を指定するという方法を採っている（図 6）．

これによって，データ項目の型・桁の変更が発生しても，個別に属性や入出力パラメタの定義を変更する必要はなくなり，ユーザ定義型のみの変更で対応可能となっている．また，後述するが，属性や入出力パラメタは，COBOL の構造体として作成されるが，この場合でも，ユーザ定義型を修正し，構造体の修正指示を行うだけで，そのユーザ定義型を使用しているデータ項目を含む全ての構造体の一括変更が可能となる．

この機能は，本システムをユーザに適用する際のカスタマイズに対しても有効であり，本システムのデータ項目の型・桁を，導入したユーザの型・桁に合せざるを得ない場合でも，ユーザ定義型の変更と各構造体の再作成だけで対応可能となっている．

4.3　プログラムスケルトン・構造体の生成

各オブジェクトは，1 本のプログラムとして実装される．一般的にオブジェクト指向技術を支援する言語であれば，メソッド（本システムでは，「サービス」に相当）単位にエントリを定義できる．しかしながら，COBOL は，そのような言語機能を有していない．従って，一つのオブジェクトを 1 本の COBOL プログラムとして実装す

るためには，プログラム構造上に工夫が必要となる．

プログラム・スケルトン生成機能は，リポジトリに定義した情報をもとに，この構造を自動生成する機能である．これにより，プログラマは，生成されたスケルトンに対し，各サービスの業務処理の詳細部分のみを追記すればよく，上述した様な構造を意識する必要はない．

また，オブジェクトの属性，入出力パラメタは，COBOL の構造体として実装されるが，これに関しても，リポジトリ定義情報から本ツールが自動生成する．

5. プログラミング工程支援

5.1 エディタ

SBI 21 でのプログラミング工程とは，プログラムスケルトンに対して，処理詳細内容を追加していく作業のことである．本開発環境では，この場合の専用エディタとして，PAD 描画機能と「MF COBOL アニメータ」を提供している．

PAD 描画機能は，連接，反復，選択の基本構造を独自の記述方式で表示し，プログラムのアルゴリズムを視覚的に把握することを目的としたツールである．

特に，プログラムバグ原因の上位を占める分岐選択文（IF，EVALUATE 文）のネスト構造の不備等の検出に対して，当ツールを使用することにより，ネストの範囲等が明確になり，構造の把握が容易になるため効果的と言える．また，プログラム内のシーケンス（CALL，PERFORM 文の呼び出し関係）構造を木構造で表現できる機能も有しており，一連のプログラム内の制御の流れを視覚的に把握できる．図 7 は，PAD 描画機能でソースコードを読み込んだ図である．この場合，左ペインに，そのシーケンス構造を表示し，右ペイン側に各節のアルゴリズムが PAD として表示される．また，単に表示するだけではなく，PAD に対して，任意の修正やロジックの追加が可能であり，その修正内容は，ソースコードに自動的に反映される．

MF COBOL アニメータは，MF COBOL が標準的に提供するエディタで，他の一般的な市販ツールのエディタと同等の機能をもつエディタである．この MF COBOL のアニメータについては，6.4 節にて詳しく説明する．

5.2 規約原則チェック機能

前節にて作成したプログラムは，MF COBOL にてコンパイル（構文チェック）し，デバッグするわけであるが，本開発環境では，コンパイル前に SBI 21 の規約に違反していないかを機械的にチェックする規約原則チェック機能を提供している．

規約原則チェック機能で実施しているチェックは，主に次の 4 点である．

① カプセル化のチェック
② SBI 21 の各種規約のチェック
③ 言語の非互換チェック
④ スケルトンの破壊防止

①については，カプセル化されたデータに対する参照制約のチェックである．通常のオブジェクト指向言語であれば，言語レベルでデータの参照制約が可能であるが，本システムの場合，オブジェクト指向技術を支援する機能を有しない COBOL を採用しているため，独自の規約をもとにカプセル化のチェックをおこなっている．具体的

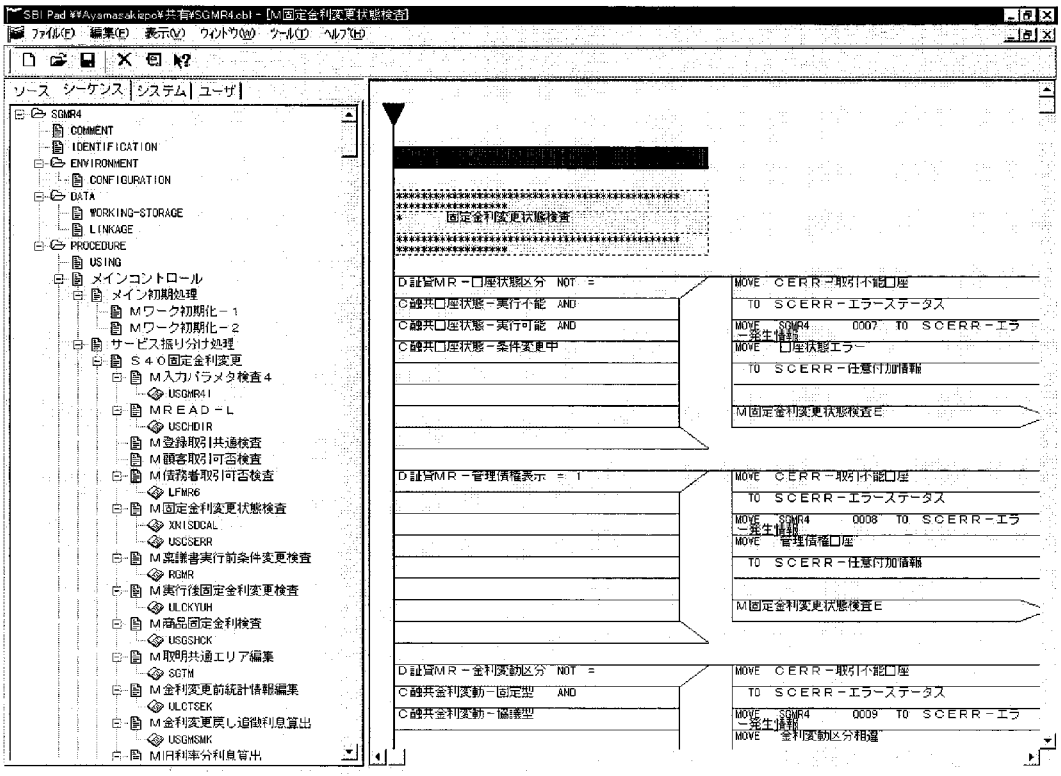

図 7　PAD 描画機能の入力画面

には，あるオブジェクトが他のサブシステムの属性を直接，ソースコード上にデータ定義していないかをチェックしている．このため，オブジェクトや属性，入出力パラメタなどの名称の一部に，サブシステム ID を必ず含めるようにネーミング規約を定めており，オブジェクトの属するサブシステム ID とは異なる属性の構造体をデータ定義していないかを判別可能にしている．

②は，本システムが独自に定めた各種規約のチェック機能である．例えば，本システムでは，構造の歪み防止のためにプログラムの参照制約を設けている．このように開発者が厳守すべき規約をシステム的にチェックすることによって，開発時に定めた規約が将来の保守段階においても，維持可能となるように配慮している．

COBOL には国際規格が存在しており，各ベンダーは，その規格に基づいた COBOL を提供している．しかし，実際にはベンダー毎に独自の機能を付加しているため，異なるベンダー間においては，COBOL と言えども完全に互換性が有るというわけではない．この非互換部分のチェックをおこなっているのが，③の機能である．特に，本システムでは冒頭でも記述したように，異なるプラットフォーム上でも可能な限りアプリケーションに修正を加えることなく稼働できることを目標にしているため，ベンダー固有の言語機能は使用しないという方針で開発している．③の機能は，この開発方針をシステム的にサポートする機能である．

前述したように，プログラムのスケルトン部分に関しては，リポジトリの定義情報

から自動生成される．従って，自動生成したスケルトン部分を修正した場合には，リポジトリとプログラムとの間が不整合になる．この不整合の発生を防止する機能が④の機能である．

6.　単体テスト工程支援

6.1　単体テスト支援機能の概要

冒頭でも述べたように，本開発環境では，プログラムの品質に直結し，テスト負荷の大きい単体テスト工程を支援する機能に重点を置いて開発している．

特に，実行環境側のデータベースを Windows NT 上でもエミュレートできるように，ローカルデータベース（ORACLE）を配し，実際のデータベースに対する COMMIT & Rollback を含めたテストを可能にしている．

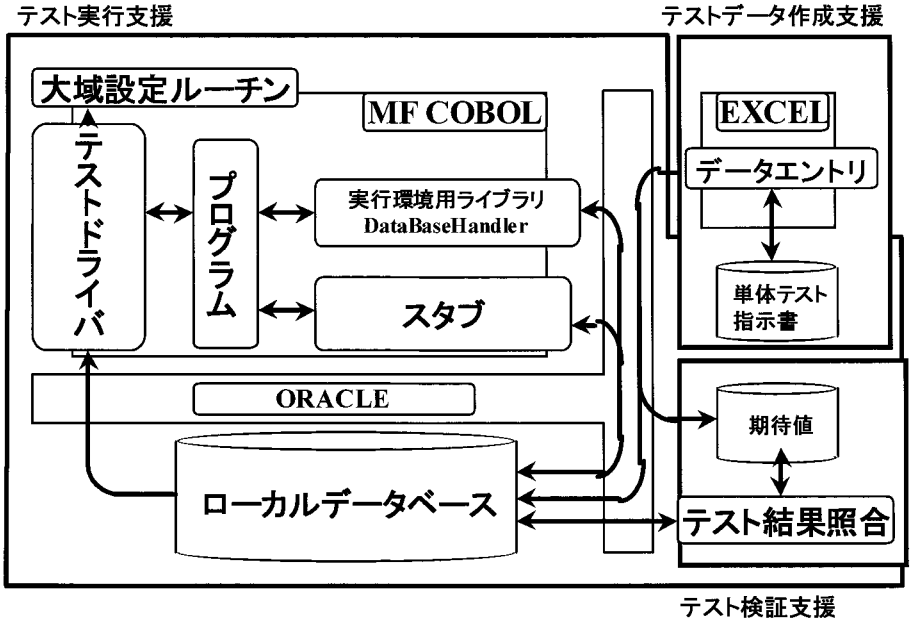

図 8　単体テスト支援機能の概念図

図 8 は単体テスト支援機能の概念図である．図からもわかるように，大きくは，テストデータ及びテスト環境の作成を支援する部分，テスト実行を支援する部分，テスト検証を支援する部分に分かれている．

6.2　テストデータ・テスト環境作成支援

単体テストをおこなうためには，まず，単体テスト用のデータを作成しなければならない．本開発環境では，この単体テスト用データを定義するために「単体テスト指示書」と呼ぶ EXCEL のブックを用意している．

単体テスト指示書に定義するデータには，以下の 6 種類がある．

① オブジェクトが使用するデータベースのデータ値

② ①に対するテスト実行後の期待値

③ テストドライバが被テストプログラムに渡す引数の値

④　被テストプログラムがテスト終了後に返す実行結果の期待値

⑤　スタブを呼び出す際にセットする引数の期待値

⑥　スタブから返される引数の戻り値

ここで記述している「テストドライバ」とは，被テストプログラムを呼び出す親プログラムが未作成の場合に代理応答するダミープログラムであり，「スタブ」とは，被テストプログラムが呼び出す子プログラムが未作成の場合に代理応答するダミープログラムのことである．いずれも，本開発環境がテスト実行時に自動生成する．

上記からもわかるように，本開発環境では，テストで使用するデータベース等のデータ値の他に，テスト実行後の期待値（上記の②，④，⑤）を定義できる仕組みを提供している．これにより，6.3 節に記述するテスト検証支援機能を利用して，テスト結果の正当性確認を機械的におこなうことが可能となり，テスト結果の検証負荷軽減を実現している．

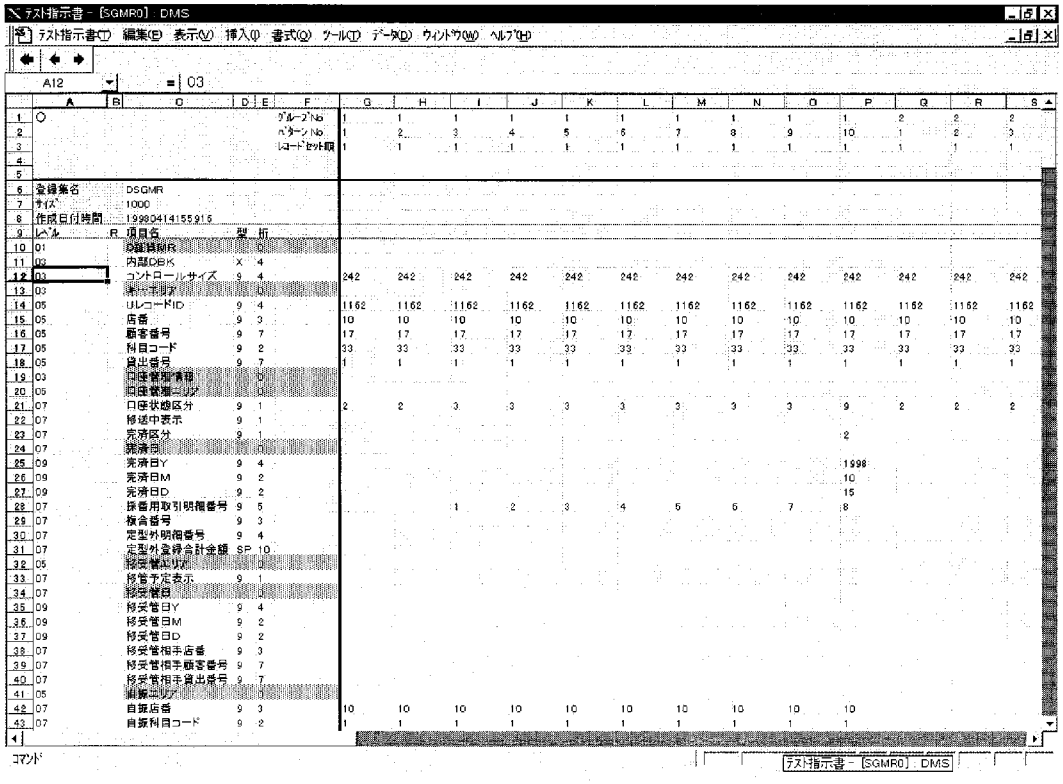

図 9　単体テスト指示書データ入力シート

図 9 が単体テスト指示書のデータ入力シートであり，このシートに実際のテストデータを入力する．図からもわかるように，左側の列にはテストで使用するデータベースや引数のデータ項目名が構造体のレイアウトの順番に表示される．従って，テストデータ作成者は，テストケース毎に右側の列にテストで必要となるデータ値，期待値を入力していくだけでよい．なお，保守段階において，単体テスト指示書に定義され

ているデータベースや引数のデータ項目に追加・削除が発生した場合でも，最新のCOBOL の構造体を元にテスト指示書の再作成をおこなうことで追加・削除されたデータ項目は自動的にメンテナンスされる．従って，単体テスト指示書を手動で個別修正したり，一から作り直す必要はない．

更に，この単体テスト指示書には，テスト用ローカルデータベースの環境定義情報等も保有している．これにより，単体テストで必要となる ORACLE の表定義等は自動的におこなわれることになり，テスト作業者がテスト環境生成を意識する必要はない．

6.3 テスト検証支援

本開発環境では，テスト結果の検証作業負荷軽減を目的として，テスト実行結果の各種データコンペア機能を提供している．コンペアする対象は，以下のものである．

① データベースの B/L と A/L
② データベースの更新結果とその期待値
③ テスト実行後の被テストプログラムの引数の戻り値とその期待値
④ 被テストプログラムがスタブに渡す引数値とその期待値

特に上記②〜④の機能により，テスト結果の機械的な検証を可能にしている．コンペア結果は，各構造体の項目単位に表示し，不一致項目は，色を変えて識別できるようになっている．また，不一致項目以外は，非表示にする機能も提供し，検証範囲の特定も可能な仕組みになっている．

6.4 デバッグ支援

実行結果と期待値とが不一致になった場合，原因追求のためのデバッグ作業をおこなうことになる．しかし，それに対しては，MF COBOL アニメータの機能を活用して，再度同一のテストを実行できる機能を提供することで，その作業負荷の軽減を図っている．アニメータは，Symbolic Debug や Inspector，Conditional Breakpoint などの機能を備えており，ステップ実行や任意の箇所でのデータの内容等を把握でき，かつ修正しながらテストの実行が可能なため，不具合の原因追求を容易におこなうことができる．また，従来のメインフレームのコマンドベースのデバッグ環境に比して，GUI 化されているため，操作性や生産性の向上を実現している．

6.5 そ の 他

6.5.1 サービス間結合テスト

本開発環境では，オブジェクトのサービス間の整合性を確認するテストが可能である．例えば，図 10 に示すように，普通預金口座オブジェクトの場合，その口座（図 10 の A 口座）のライフサイクル（口座開設→入金→出金→解約）を一つのテストケースで実行・確認することが可能であるということである．このように，口座開設という取引によって作成された口座のデータベースを使用して，実際に入金や出金，解約という取引が可能か否かというような各取引間の整合性の検証を目的とするテストは，従来結合テスト工程での作業であった．しかしながら，本開発環境では，単体テスト工程という，より上流の工程で，その検証を可能にしている．このようなテストが実現可能となった要因は以下の 2 点によるものである．

・本システムのアプリケーション構造がオブジェクト指向技術を採用したことに

より，あるデータベース（本例の場合，普通預金口座）に対する操作（本例の場合，口座開設，入金等）が，基本的には，一つのプログラムに全て定義されている．

・単体テスト用のローカルデータベースを保有し，実際に Commit 処理をおこなっているため，次のテストでは，その更新結果をそのまま入力データとして使用可能である．

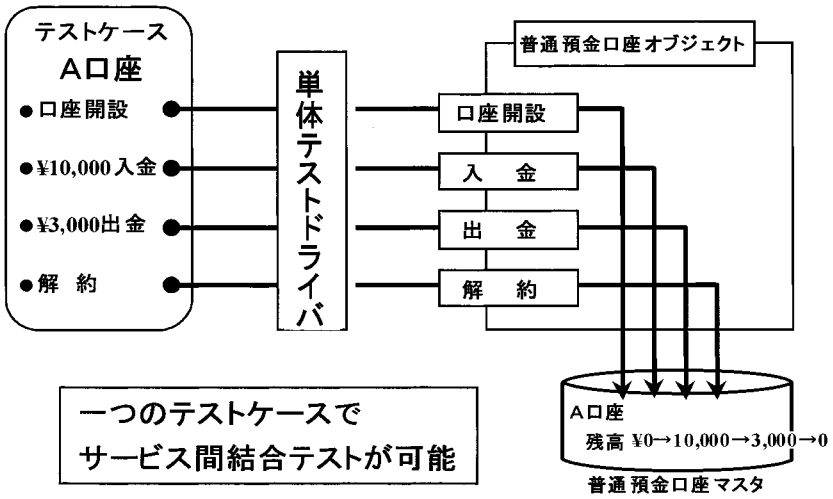

図 10　複数サービスを跨るテスト

6.5.2 Automated Testing 機能

本開発環境で作成される単体テスト指示書は，電子的に保存可能である．かつ，本開発環境は，保存された単体テスト指示書に定義されている全テストケースを一括実行し，その実行結果と前回実行結果とを一括して比較する機能を提供している．これを Automated Testing 機能（図 11）と呼んでいる．

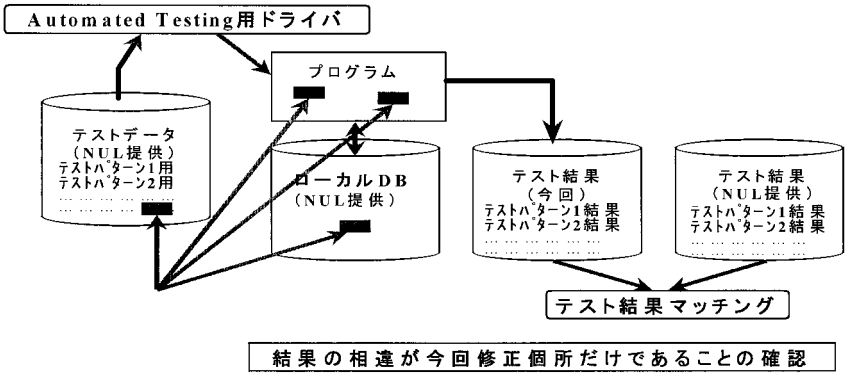

図 11　Automated Testing 機能

この機能は，主に保守工程におけるプログラム修正時のテスト作業負荷軽減を意図

して作成した機能である．この機能の提供により，プログラム修正が発生した場合，修正担当者はプログラム修正後，Automated　Testing 機能を用いてテストの自動実行をおこない，前回と今回とのテスト結果の相違部分が今回のプログラム修正に起因して発生したものか否かを検証するだけでよい．また，これにより，修正した箇所が他の処理に影響を及ぼしていないことの確認も同時におこなうことになる．

更に，修正内容によっては，テストケースを追加する場合もあると思われる．この場合，新たにテストケースが追加になった単体テスト指示書を保存し，次回のテストで使用することにより，テストケースのバリエーションが自然に増えていき，テスト密度を高めていくということが可能である．

この機能は SBI 21 導入時のカスタマイズ作業負荷軽減にも有効である．本システムでは，今回のアプリケーション開発で使用した単体テスト指示書を商品として導入ユーザに提供する．従って，導入ユーザはカスタマイズによるプログラム修正後，この単体テスト指示書で上記機能を使用することにより，カスタマイズした部分のテスト確認を容易におこなうことができ，導入工数の負荷の軽減が図られる．

7.　今後の予定している機能

7.1　実行環境支援

Windows NT 上で作成し，単体テストまで終了したプログラムは，実行環境であるメインフレーム上でコンパイル/リンクする必要がある．本開発環境では，Windows NT からメインフレーム上にソースコードをファイル転送後，コンパイル/リンクをおこない，その実行結果を Windows NT 上に返す機能を提供する．現時点では，IX/ESPEC で，メインフレームとの制御をおこなう GUI ツールを開発する予定である．また，パラメタなどのテキストファイルの転送は，WinFP（ユニシスの HMP　IX シリーズのソフトウェアで，2200 上のファイルを PC 上の GUI で操作可能なソフトウェア）にておこなう．

更に，従来実行環境側で管理していた実行環境の環境定義やパラメタ情報などを Wiindows NT 上のリポジトリに定義し，項目間の正当性検査等をおこない，実行環境用パラメタを生成し，実行環境へ転送する機能も提供する．

7.2　プログラム管理

プログラム管理については，PVCS などの市販ツールを前提に提供する予定である．このなかでは，ソースコードのバージョン管理やプログラムリリースの管理も含まれる．

また，開発案件単位での開発管理や一つのプログラムを複数案件で修正する場合の多重開発の管理等の仕組みも提供する．

この他，影響範囲の特定作業に使用されるインパクトレポートなど，保守工程を支援する機能も提供する．

8.　お　わ　り　に

実際の SBI 21 のアプリケーション開発にて，本開発環境を使用した結果，以下のことが明らかになった．

① テストデータ作成工数の増加
② 検証工数の大幅な削減と検証漏れの防止
③ 仕様変更での手戻り工数の削減

①に関しては，従来の単体テスト工程では，テストデータは，テスト実行に必要なデータのみであったが，本システムでは，その他にテスト実行時の期待値もテストデータとして作成していることに起因している．また，当然，テストデータの作成者は，単なるプログラマではなく，そのプログラムの設計者もしくは，少なくとも業務知識の有資格者に限定されるので，テストデータ作成工数は，従来に比べて増加する．

②に関しては，①で作成したデータに基づく機械的な検証作業が実現できたことによる効果である．本システムでは，プログラミング＆単体テスト部分を協力会社へ外部発注しているが，従来の開発であれば，納品受け入れ検証は，設計者や業務知識の有資格者といった限られた人が，協力会社から納品されたソースコードとテスト結果を目視検証する作業であった．この場合，検証に費やす時間・コストに比して，人的検証による検証漏れや抜けといったリスクは高くなるため，結合テスト工程以降で品質を上げる努力をしなければならなかった．しかしながら，本システムの場合，協力会社から納品されたソースコードと単体テスト指示書を，無条件に規約原則チェック機能と Automated　Testing 機能で再実行し，その実行結果を機械的に検証できる．作業そのものも，ツールの実行と結果の確認といったルーチンワーク的な作業が主体となっているため，検証作業の大部分を設計者以外の要員で代替でき，最終確認のみを設計者がおこなう運用を採っており，検証工数の大幅な削減と検証時における設計者の負荷軽減が可能となった．また，協力会社側から納品されたテスト結果を検証するわけではなく，受け入れ側の環境で納品されたプログラムとテストデータをもとに再実行した結果の内容を検証するため，納品されたプログラムとテスト結果に不整合が生じるということもなく，より確実で精度の高い検証作業が実施可能である．

③に関しては，Automated　Testing 機能の効果によるものである．従来，仕様変更が発生した場合，修正作業に比して，テストの作業負荷は非常に大きかった．しかしながら，この機能を使用することで，機械的に全テストケースの実行・確認が可能になるため，作業負荷が軽減され，かつ，高品質の成果物を開発することが可能である．

最後に，本開発環境の他のシステムへの流用性について記述する．冒頭でも記述したが，本開発環境は，SBI 21 専用の開発環境であり，本システムのアプリケーション構造を前提としたツールである．しかしながら，正確に言うと，上流工程（分析・設計支援）を支援するツール群は，本システムのアプリケーション構造/規約に沿ったツールになっているが，プログラム作成工程以降のツールに関しては，極力，通常の COBOL のプログラム開発でも適用可能な構造になるように設計している．従って，プログラム作成工程以降のツールに関しては，そのままの適用は無理にしても他システムへの流用度は高い．但し，「5.2 規約原則チェック機能」でも記述したように，ベンダー固有の言語機能などベンダー間での相違による制約はともなうので留意する必要がある．

参考文献　［1］ J. マーチン/J J. オデル,「オブジェクト指向方法序説実践編」,（株）トッパン.
［2］ J. ランボ/M. ブラハ/W. プレメラニ/F. メディニ/W. ニールセン,「オブジェクト指向方法論 OMT モデル化と設計」,（株）トッパン.
［3］ 二村良彦, 情報工学基礎講座 4 プログラム技法―PAD による構造化プログラミング―, オーム社, 1984.

執筆者紹介　奥 野 信 幸（Nobuyuki Okuno）
1986 年大阪市立大学経済学部卒業．同年日本ユニシス（株）に入社．都銀の SE サービス，TRITON の開発環境適用を経て，SBI 21 の企画業務と PC 開発環境の開発業務に従事．現在，ビジネスソリューション一部開発二室所属．
E-mail : Nobuyuki.Okuno@unisys.co.jp