プロトタイプ手法を支えるLUCINA支援ツール

LUCINA Support Tools for Rapid Prototyping

真田正二

要　約　本稿では，LUCINA開発技法を支援するツールとその適用方法について紹介する．

プロトタイプ支援ツールは，LUCINAにおける業務分析フェーズのプロトタイプの作成に伴うTUXEDO環境およびSTDL環境の構築を劇的に軽減するツールである．またシステム構築フェーズにおいても，インタフェース仕様記述ができた段階で実アプリケーションの各コンポーネントに相当する仮コンポーネントおよびそれらの実行環境を自動的に作り出すことができる．実コンポーネントの単体テストが終了するたびに，仮コンポーネントと置き換えていくことで，アプリケーションを完成させることができる．

単体テスト支援ツールは，TUXEDOおよびSTDL環境のない環境でコンポーネントの単体テストを可能にする．中央の開発環境には，結合テスト用のTUXEDO管理下のアプリケーションを置き，周辺の開発環境ではTUXEDOを必要としない単体テスト環境を必要なだけ作るという方法で，開発作業の並行性を高め，生産性を向上することができる．

Abstract　This paper introduces support tools for LUCINA development methodology and their application.

In the business analysis phase using LUCINA, the prototyping support tool reduces dramatically the construction effort of TUXEDO and STDL environment required for prototyping. Further in the system construction phase, when the interface description is completed, this tool can automatically generate dummy components corresponding to the components of the actual application and their runtime environment.

The unit test support tool makes it avalable to test each component in non-TUXEDO and non-STDL environment. This improves the parallel development, thus enhances the productivity of the application development by providing many unit test site without TUXEDO and STDL around the central integration test site with them.

1. はじめに

コンポーネント指向の開発技法である[1][2]LUCINAの特徴として，どのプロファイルにおいても開発プロセスの流れは共通であり，一旦習熟すれば他のプロファイルに移っても混乱なく開発を進めることができる．LUCINA for COBOL/STDL[3][4][5]は分散トランザクション処理システムを開発するためのLUCINAのプロファイルである．

このプロファイルで提唱するアーキテクチャは，すでにUNIX機十数台から数十台が連携する複数のトランザクション処理システムで採用されており，大規模分散トランザクション処理システム環境として十分な実力を発揮している．しかし，これまでの開発はすべて手作業で行われており，今後の継続的な採用に向けて開発支援ツールが望まれていた．

例えば，分散トランザクション処理の実行環境を構築するためには，各処理コンポーネントのモジュールを作成すると共に，TPモニタであるBEA　TUXEDO（以下

TUXEDOと略す）の構成ファイルにそれらを記述したり，STDLで規定されるさまざまな環境変数を設定する複雑な作業が必要であった．

今回紹介するLUCINA支援ツール群は，LUCINA for COBOL/STDLの特性に着目して新たに開発したツールであり，プロトタイプ支援ツール[6]および単体テスト支援ツール[7]からなる．これらを利用することで，プロトタイプを作成して要件定義を練り上げるLUCINA手法の適用が容易になり，かつ単体テストをトランザクション環境から切り離して実施できることから，並行開発による生産性の向上にも寄与できる．さらに，仮コンポーネントからなるアプリケーション全体のプロトタイプを作成しておき，単体テスト済みの実コンポーネントに次々と置き換えていくことで完成させるプロトタイプ手法も実践することができる．

2. 分散トランザクション処理開発技法 LUCINA for COBOL/STDL

2.1 コンポーネント指向に適したSTDL言語

LUCINA for COBOL/STDLは，分散トランザクション処理部分を記述する上で，オープングループ（旧X/Open）標準の構造化トランザクション定義言語STDL（Structured Transaction Definition Language）を用い，業務処理部分を記述する上でCOBOL言語（またはC言語）を用いる．STDLによりTPモニタに依存しないトランザクション処理が記述でき，COBOL言語（またはC言語）は各機種における標準言語であることから，アプリケーションの移植性と相互運用性を実現できる．また，言語構造上コンポーネント指向の開発に適しており，LUCINA開発技法を適用する上で都合がよい．今回の議論の前提となるSTDL言語の特徴を紹介しておく．

STDL言語では，構造化という名前が表すように，言語要素が

- データ型定義
- タスクグループ仕様
- タスク定義
- プロセッシンググループ仕様
- プレゼンテーショングループ仕様
- メッセージグループ定義

という六つの要素に構造化されている．このうち，仕様と名付けられている要素はインタフェースを規定し，定義と名付けられている要素は具体的な実装部分である．すなわち，仕様と実装が明確に分離されている．

例えば，タスクグループ仕様は，タスクグループに含まれるタスクのインタフェースを記述し，タスク定義がSTDL言語で書くその具体的なプログラミング部分である．プロセッシンググループ仕様は，プロセッシンググループに含まれるトップレベルプロセッシングプロシージャ（TPP，タスクから直接呼ばれるプロシージャ）のインタフェースを記述し，COBOL言語（またはC言語）でその具体的なプログラミングを記述する．このように，言語的には明確に分離されたインタフェース記述をもたないCOBOL言語であっても，STDL言語でそのインタフェースを記述することができる．

データ型定義は，COBOL言語，C言語，STDL言語など異種言語の混在する環境

でのデータの型を STDL 言語で記述する要素であり，COBOL 言語や C 言語のデータ型へのマッピングは，STDL コンパイラが担う．

なお，従来の集中トランザクション処理をサポートする言語要素であるプレゼンテーショングループについては，LUCINA for COBOL/STDL では使用しない．また，メッセージグループ定義については本稿では言及しない．

2.2 分散トランザクション処理プロファイルの概要

LUCINA for COBOL/STDL が想定しているアプリケーションモデルは，図 1 のとおりであり，一般にはビジネスサービス層を含む複数のコンポーネントシステムがコンポーネントバスで接続される大規模分散トランザクション処理システムを想定している．プレゼンテーション層は，クライアントサーバモデルではクライアントプログラムを搭載した PC であり，WEB 連携モデルでは，WEB ブラウザを搭載した PC およびそれと連携する WEB サーバ機である．

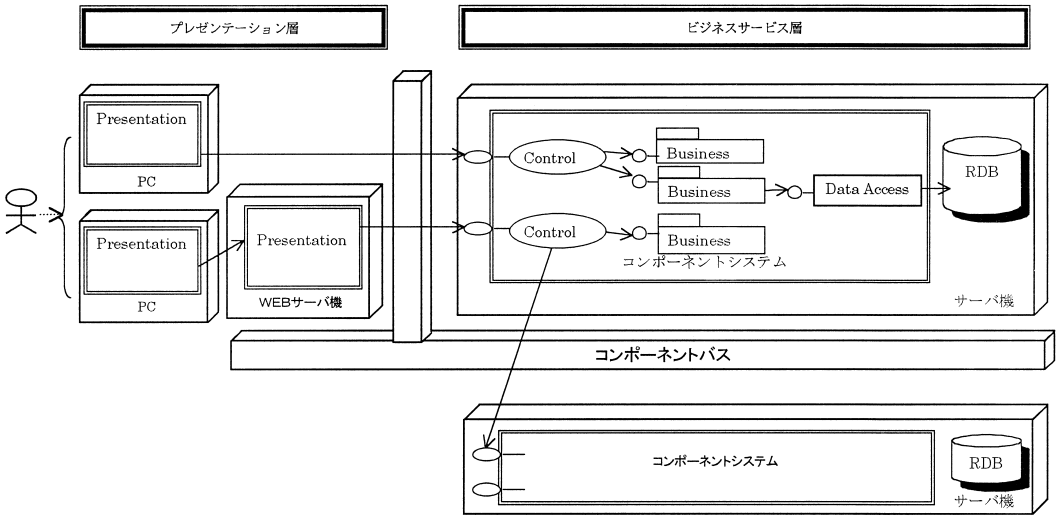

図 1　LUCINA for COBOL/STDL のアプリケーションモデル

一つのコンポーネントシステムは，ビジネスサービス層をなすコンポーネントの集まりであり，論理的にまとまった処理単位を形成している．コンポーネントシステムは外界に対してインタフェース（図 1 の長円で表される）を公開しており，コンポーネントへのアクセスはインタフェースを介してのみ行われる．利用者からの入力を受け付け，出力を返すプレゼンテーション層もコンポーネントシステムへのアクセスはインタフェースを介して行う．

公開しているインタフェースを変更しない限り，コンポーネントシステム内の各コンポーネントにアルゴリズムや局所的データに変更を加えても，外界に対する影響はない．

コンポーネントシステムの粒度としてはかなり大きな処理単位を想定しており，例えば会社内のシステムでいうと，人事システムや経理システムのような 1 システムがそれぞれコンポーネントシステムとなる．コンポーネントシステム間のアクセスは，

例えば経理システムから人事システムのデータを参照したい場合に行われる．各システム内では日常的プログラムの修正作業が行われるが，インタフェースを変更しない限り，外界への影響は及ばず安全な運用が保障される．

コンポーネントシステム内の各コンポーネントにはそれぞれの役割に応じて4つステレオタイプが割り振られている．他のLUCINAプロファイルの中で言及されるバウンダリタイプは，当プロファイルではコンポーネントバスが担うので特に意識する必要はない．

・プレゼンテーションタイプ
利用者からの入力を受け付けたり，入力データを検査したり，画面遷移を制御する役割をもつ．データの処理の依頼は，コントロールタイプに対して行い，出力を利用者に返す．

・コントロールタイプ
外界とのインタフェースの役割をもち，コンポーネントシステムの入り口となる．プレゼンテーションタイプまたは他のコントロールタイプからの要求を受け付ける．受け付けた要求は，トランザクションを開始させてビジネスタイプあるいは他のコントロールタイプを呼び出し，処理を依頼する．処理結果の例外処理も担当し，例外が起きていればトランザクションをロールバックする．例外が起きていなければトランザクションをコミットする．その後，結果を呼び出し元に返す．

・ビジネスタイプ
コントロールタイプや他のビジネスタイプから呼ばれ，ビジネスルールに則って抽象化されたレベルでのデータの処理を行う．具体的なデータアクセスは，データアクセスタイプを呼び出すことで実施する．

・データアクセスタイプ
ビジネスタイプからデータアクセス要求を受け取り，実際にデータベースの入出力を行う．

図1においては，コンポーネントシステムを一つのサーバ機に配置している．アプリケーションを構成する実行モジュールの管理上はそのほうが望ましいといえるが，複数のコンポーネントシステムを一つのサーバ機に配置することも可能である．その場合，複数コンポーネントシステムの実行モジュールを統合して管理することになる．先ほどの例でいうと，人事システムの実行モジュールを追加するような変更でも，経理システムについても考慮を払わなければならない．

WEB連携モデルのWEBサーバ機の役割は，サーバ機と同じマシン上に配置してもかまわない．

2.3 実装イメージ

図2に，LUCINA for COBOL/STDLの実装イメージを示す．コンポーネントバスの役割は，TPモニタであるTUXEDOが担う．サーバ機上のプログラム記述言語はCOBOL（またはC言語）とSTDLというコンパイラ言語であり，データベースはトランザクション処理をサポートする任意のリレーショナルデータベースである．これらの採用により，ミッションクリティカルな大容量高速トランザクション処理に十分

耐え得るアーキテクチャを実現している．

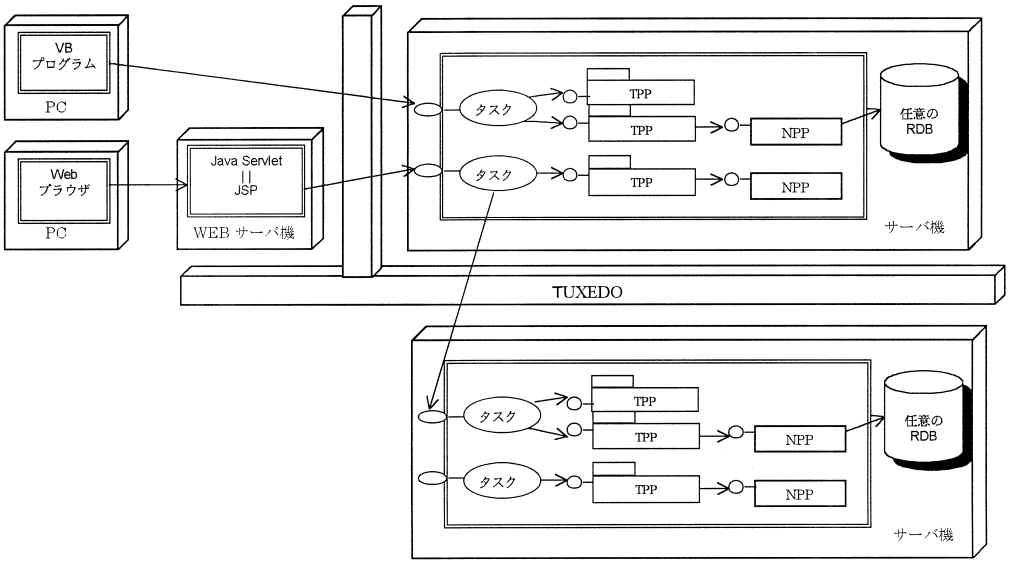

図 2　LUCINA for COBOL/STDL の実装イメージ図

各ステレオタイプの記述言語は，次のとおりである．

・プレゼンテーションタイプ

クライアントサーバモデルの場合，Visual　Basic 言語．WEB 連携モデルの場合 Java 言語．

・コントロールタイプ

STDL 言語で記述したタスク．トランザクション処理に関する記述はここにだけ現れる．タスクは他のタスクを呼び出したり同一マシン上のプロシージャを呼び出すことができる．その記述は特定の TP モニタに依存せず，分散環境で相互接続および相互運用が可能である．

・ビジネスタイプ

COBOL 言語で記述したプロシージャ．STDL タスクから直接呼ばれるトップレベルプロセッシングプロシージャ（TPP）になる．なお，C 言語で記述することも認めている．

・データアクセスタイプ

COBOL 言語および埋め込み SQL 言語で記述したプロシージャ．TPP から呼ばれるネステッドプロセッシングプロシージャ（NPP）になる．ビジネスタイプ同様，C 言語での記述も認められる．NPP の代わりに，SQL のストアドプロシージャとして実装することも可能である．

タスクは，それ自身が実行モジュールである．TPP と NPP はリンクされ，実行モジュールとなる．したがって，タスクからタスクの呼び出しおよびタスクから TPP の呼び出しは，コンポーネントバスである TUXEDO を経由した実行モジュール間の遠隔プロシージャ呼び出し（RPC：Remote Procedure Call）の形態となる．

3. LUCINA支援ツール

3.1 LUCINA支援ツールのねらい

STDL言語の採用によってコントロールタイプはTPモニタに依存せず，またTPPおよびNPPではトランザクション処理に関係する記述が一切不要であるので，アプリケーションの記述全体はTPモニタに依存しない．したがって，プログラム構築フェーズでは特にTUXEDOに関する知識を必要としない．

しかし，アプリケーションを構成するサーバ機上の実行モジュールはすべてTUXEDOの管理下になければならず，そのためTUXEDO構成ファイルに各種パラメタと共にそれらを指定しなければならない．また，STDL言語で規定された数多くの環境情報を設定しなければならず，それらはかなり複雑な操作となる．

すなわち，STDL言語はプログラムの記述に関する抽象度を高めているものの，実行環境の抽象化にまで踏み込んでいない．LUCINA支援ツールは，実行環境の抽象度を高め，LUCINAを実践するためのプロトタイプ作成や作成したコンポーネントの単体テストにおける，TUXEDOの高度な専門知識の習得を不要にする．

今回紹介するLUCINA支援ツールには，プロトタイプアプリケーションの実行環境を作り出してくれるプロトタイプ支援ツールと，単体テストをTUXEDO環境から切り離して実施できるようにする単体テスト支援ツールがある．これらの概要とLUCINA for COBOL/STDLにおける適用方法について紹介する．

3.2 プロトタイプ支援ツール

LUCINA開発技法においては，業務分析フェーズの中で要件定義および論理設計においてそれぞれアプリケーションのプロトタイプを作成し，ユーザ要件の確認およびシステム・アーキテクチャの詳細化を練り上げ，実アプリケーションの実現可能性を確認する．このため，開発の初期段階にもかかわらず，プロトタイプを稼動させるためにTUXEDOおよびSTDL言語規定の環境情報の専門知識が必要となる．プロトタイプを簡単に構築できるような支援ツールがあれば，この作業は大幅に軽減できる．ここで紹介するstdlprotは，まさにこの目的にそって開発されたサーバ機側のプロトタイプ作成支援ツールである．プレゼンテーション部分に関する支援機能は備えていない．また，各種業務ロジックのテンプレートを備えているわけではない．

STDL言語の特徴として，タスクおよびTPPのインタフェース記述はタスクやTPPのプログラミング記述と分離しており，タスクグループ仕様およびプロセッシンググループ仕様というソース単位に書かれる．図3は，タスクグループ仕様，プロセッシンググループ仕様およびそれらで使われるデータ型定義の例である．例えば，インタフェース記述については，タスクtask_aはrec 01型の入出力引数を1個受け渡すというように読む．また，TPP　proc_bはCOBOL言語で記述され，rec 01型の入力引数を1個受け取ると読む．

なお，プロトタイプを作成するためには，コンポーネント間の呼び出し関係の情報が必要となる．図4は，stdlprotに入力する呼び出し関係情報ファイルの例である．呼び出し関係情報については，クライアントclient.cはタスクtask_aとtask_bを呼ぶ．タスクtask_aはTPP　proc_aとproc_bを呼ぶ．タスクtask_bはTPP　proc_bを呼ぶというように読む．

```
! インタフェース記述
TYPE rec01
  RECORD
    test_id TEXT SIZE 12;
    result  TEXT SIZE 4;
  END RECORD;
TASK GROUP SPECIFICATION taskg_1
  UUID “09205806-7446-11D0-9355-08002B3B1E19”;
  VERSION 1.0;
  TASK task_a
    USING rec01 PASSED AS INOUT;
  TASK task_b
    USING rec01 PASSED AS INPUT;
END TASK GROUP SPECIFICATION;
PROCESSING GROUP SPECIFICATION procg_1
  LANGUAGE IS “COBOL”;
  PROCEDURE proc_a
    USING rec01 PASSED AS INOUT;
  PROCEDURE proc_b
    USING rec01 PASSED AS INPUT;
END PROCESSING GROUP SPECIFICATION;
```

図 3　インタフェース記述

```
! 呼び出し関係情報ファイル
*CLIENT   client.c
   task=task_a(taskg_1)
   task=task_b(taskg_1)
*TASK   task_a(taskg_1)
   proc=proc_a(procg_1)
   proc=proc_b(procg_1)
*TASK   task_b(taskg_1)
   proc=proc_b(procg_1)
```

図 4　コンポーネント間の呼び出し関係の記述

このようなコンポーネント間のインタフェース記述とこれらの呼び出し関係の記述さえ用意できれば，stdlprot はそれらを入力して，図 5 のようなプロトタイプアプリケーションを作り出してくれる．

作り出された各仮コンポーネントは，自分が受け取る引数の記述，制御が渡ってきたことをログファイルに書き出す命令，および呼び出し関係情報に記述されたコンポーネントへの CALL 文を含んでいる．加えて，これらのソース・プログラムをコンパイルする make ファイル，でき上がった実行モジュールを TUXEDO に登録するための TUXEDO 構成ファイル，STDL 言語で規定された環境情報を定義する環境情報定義ファイルおよび実行用シェルプログラムが生成される．これらを利用してとりあえずプロトタイプアプリケーションを実行させることができる．

3.2.1　業務分析フェーズでの適用

業務分析フェーズにおけるプロトタイプ作成の目的は，要件定義から導き出される要求リスクを検証して実アプリケーションのアーキテクチャを確定することにある．

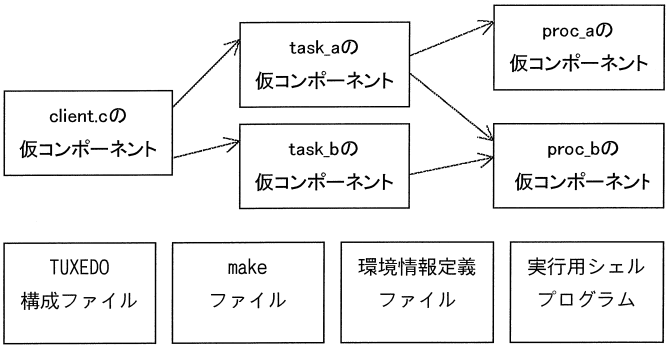

図5　作り出されたプロトタイプアプリケーション

要求リスクには，メモリ使用量，ディスク使用量，応答時間，1秒間に処理する最大トランザクション量などが含まれる．この時点でのプロトタイプアプリケーションは，必ずしも実アプリケーションと同等のコンポーネント構成である必要はない．何らかのサブシステムを想定した簡単な構成が選ばれるであろう．

これら測定対象にそって，プロトタイプアプリケーションの各コンポーネントをさまざまに修正してみることで，各種の測定値が得られる．例えば，引数の長さやデータベースのレコードサイズをいろいろ変化させて応答時間への影響を検証するとか，実行モジュールに追加されるTUXEDOのバッファサイズを測定することができる．

また，TUXEDO構成ファイル中のさまざまなパラメータやSTDL言語規定の環境情報を修正して性能がどう変化するか検証してみることができる．例えば，TUXEDO上では一つの実行モジュールを複数枚起動することが可能であるが，枚数を変化させたときの応答時間への影響やメモリ使用量の変化を測定することができるし，STDL言語規定の障害管理機能を利用するかしないかで処理効率がどう変わるか検証しておくことができる．

3.2.2　システム構築フェーズでの適用

業務分析フェーズでアプリケーションアーキテクチャを決定し，システム構築フェーズに入るとコンポーネント設計と物理設計が行われ，システム全体のコンポーネント構成とそれらのインタフェースが決定される．これらの情報をLUCINAの開発支援ツールの一つであるSEWB+（日立社製）に入力すると，SEWB+がSTDL言語のデータ型定義，タスクグループ仕様，プロセッシンググループ仕様のソースファイルを生成してくれる．また，各コンポーネント間の呼び出し関係も決まる．

実装に入る段階でこれらの情報をstdlprotに入力すると，図5よりも規模の大きい，実アプリケーションと同じコンポーネント構成をもったプロトタイプアプリケーションが生成され，実行も可能になる．そこで，各仮コンポーネントのソースファイルを物理設計書と共に構築担当のプログラマに配布する運びとなる．プログラマは，物理設計書を基に0から構築するのではなく，配布された仮コンポーネントを修正するという形で実装を行う．単体テストを終えると，プロトタイプアプリケーションの仮コンポーネントを実コンポーネントに置き換えて結合テストに移る．インタフェースを変えていないのであるから，この置き換えを行ってもプロトタイプアプリケーション

は依然，実行可能である．

こうして順次仮コンポーネントを実コンポーネントに置き換えていくことでプロトタイプアプリケーションは実アプリケーションに近づいていく．全部の仮コンポーネントが実コンポーネントに置き換わると統合の段階が終わり，システムテストが可能になる．

システムテストでは，実行モジュールの立ち上げ枚数を増やしたり，データベースのチューニングをしたりし，負荷テストを実施してトランザクション処理システムとしてのシステム性能を引き出す作業が実施されよう．

このように，プロトタイプから徐々に実アプリケーションに近づけていくプロトタイプ手法が使えることが，LUCINA for COBOL/STDL における分散トランザクション処理システム構築の大きな特徴となっている．

3.3 単体テスト支援ツール

トランザクション処理アプリケーションは TUXEDO の配下で実行しなければならないため，単体テストであっても複数のプログラマが一つの TUXEDO 構成ファイルの下で同時に個別のテストを行うことができない．誰かが終了するまで待たされる順次テストになる．大規模開発環境において多くのプログラマが一つの TUXEDO 環境の中で順次に単体テストを行うのは，生産性を悪化させる大きな要因となる．また，TUXEDO 配下であると，標準入出力がディスクファイルに切り替えられてしまうために COBOL 言語（または C 言語）用に用意されている一般のデバッガやアニメータなどのデバッグツールは，使用できないか不便な使い方になるという問題もある．

COBOL 言語（または C 言語）で書く TPP および NPP に注目すると，トランザクション処理の管理下ではあるがトランザクション処理に関する記述をまったく含んでいない．そのため，同一サーバ機であっても TUXEDO，したがって STDL もない環境で単体テストができれば，結合テスト環境とは別に各プログラマが同時に独立してテストを実施できるので都合がよい．また，標準のデバッグツールも有効に活用できる．単体テスト支援ツール stdlpptest は，まさにこれらの目的で開発された．

トランザクションの開始とコミット/ロールバックに関する記述は本来 STDL タスク側で記述するため，STDL から呼ばれる TPP や NPP では必要となるデータベースへの結合，トランザクションのコミットやロールバックに関する記述を含まない．したがって, TUXEDO および STDL がない環境で単体テストを行おうとする場合には，テスト対象プログラムのソースを修正しなくてもよいように，呼び出し側でそのような処理を記述しておく必要がある．

stdlpptest は，プロセッシンググループ仕様とそこで使われるデータ型の定義を入力して，単体テスト用メインプログラム，make ファイル，テストコントローラ，環境情報設定ファイルを生成する．図 6 で示すように，このテストコントローラの中でデータベースのアクセス環境，1 フェーズコミットに必要な処理などを記述しているので，テスト対象のソース修正なしで単体テストを実施することができる．

生成される単体テスト用メインプログラムにはテスト対象プログラムを呼び出す CALL 文が含まれている．現時点では引き渡す入力引数には省略時の値が設定されているだけなので，テストにあたってはテストしたい値を設定するようメインプログラ

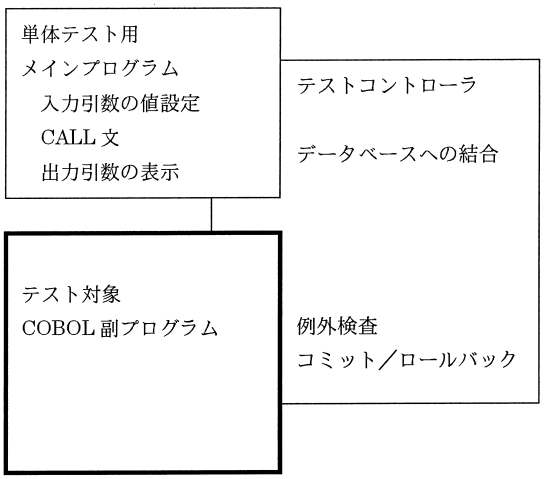

図 6　単体テストの仕組み

ムのソースを修正しなければならない．また，戻ってきた出力引数は標準出力に表示されるようになっているが，結果の値が正しいかどうかを自動判定させたいのであれば，IF文などを追加する必要がある．将来的には，テスト用の値および判定用の値複数セットをファイルに設定しておき，同一テスト対象プログラムの複数回の呼び出し結果を自動判定できるようにstdlpptestを改良する予定である．

単体テスト環境は，現状ではSolaris環境でのみ稼動するが，将来的には他のUNIX機でもWindows 2000機でも稼動できる予定であるので，結合テスト環境はTUXEDOを導入した高価なUNIX機，単体テスト環境はTUXEDOなしの安価なPCという選択も可能である．PC上で単体テストを終えたコンポーネントは，必要であれば漢字コードの変換を施して結合テスト環境に移される．

4. おわりに

LUCINA for COBOL/STDLを支援するツールとそれらを利用したプロトタイプ手法を紹介した．

LUCINA支援ツールの効果をまとめると，次のようになる．

- プロトタイプ作成ツールの利用により，業務分析フェーズにおけるシステムアーキテクチャの決定をより厳密なものにすることができる．
- システム構築フェーズにおけるプロトタイプ手法の開発では，物理設計の終了時点で，仮コンポーネントではあるがすべてのコンポーネントが揃った結合テスト環境が用意できるため，全体作業の見通しがよくなる．
- 実装段階での単体テストツールの利用により，TUXEDO環境外での単体テストが可能になるため，多くのプログラマを擁する大規模プロジェクトにおいても，コンポーネントの並行開発による開発期間短縮が見込める．
- 単体テストを終えた実コンポーネントは随時仮コンポーネントと置き換えていくことができ，通常はその都度用意しなければならないテスト用スタブやテスト用メインプログラムを，結合テスト環境にある仮コンポーネントで代用する

ことができる．

・複雑な TUXEDO 構成ファイルや STDL 言語規定の環境情報の設定も，自動生成されたものに対する変更を施すための知識があれば十分であり，専門知識の学習時間を短縮できる．

これらにの効果より，LUCINA が狙いとするプロトタイプによる業務分析フェーズの練り上げ，およびコンポーネントの並行開発による開発期間の短縮が容易に実現できる．ただし，これらの支援ツールは業務ロジックの作成を容易にすることを狙いとしているわけではないので注意されたい．その役目は，業務ロジックのテンプレートを登録して生産性を向上させる狙いをもつ SEWB＋に委ねられる．

なお，紹介した LUCINA 支援ツールの開発が可能になった背景には，STDL 言語がコンポーネントのインタフェースと実装を分離した言語構造をもつこと，ポインタ型を扱わず，トランザクション処理の電文として送受信するデータ型しか取り扱わないため，テストデータの値の設定や確認が簡単なことが挙げられる．特に単体テスト支援ツールについては，ポインタ型あるいは参照型を扱う C 言語，C＋＋言語，Java 言語について同様の試みを行う場合には，テストデータの作成に関し，より複雑な検討が必要となる．

参考文献

［1］ 松倉司,「異種ベンダツール連携による統合開発環境基盤」, 日本ユニシス技報, Vol. 20 No 2, 2000.

［2］ 羽田昭裕,「コンポーネント指向開発，現実的な開発アプローチ」，ユニシス・ニュース, 1999. 8.

［3］「LUCINA［基礎編］第 1.0 版」, 日本ユニシス, 2000.

［4］「LUCINA for COBOL/STDL［開発プロセス編］第 1.0 版」, 日本ユニシス, 2000.

［5］「LUCINA for COBOL/STDL［実践編］第 1.0 版」, 日本ユニシス, 2000.

［6］「LUCINA for COBOL/STDL［STDL プロトタイプ作成支援ツール編］第 1.0 版」, 日本ユニシス, 2000.

［7］「LUCINA for COBOL/STDL［STDL 単体テスト支援ツール編］第 1.0 版」, 日本ユニシス, 2000.

執筆者紹介

真 田 正 二（Shoji Sanada）

1969 年早稲田大学理工学部数学科卒業．同年日本ユニシス(株)入社．シリーズ 2200，A シリーズの各種言語プロセッサの開発および保守に従事．現在，E ビジネス技術部技術 2 室に所属し，UNIX 機および Windows 2000 機用 STDL 言語のコンパイラ開発に従事．