ソフトウェア工学 29－2
（１９８３．５．２５）

大規模ソフトウェア（ＡＩＭ／ＲＤＢ）の開発手法

関根　裕（富士通㈱）

０．はじめに

大規模ソフトウェアの開発においては，設計，作成，保守の全工程にわたって，一貫した開発思想が必要である。当社に於ける大規模リレーショナルデータベースシステム（ＡＩＭ／ＲＤＢ）の開発にあたって，われわれはＣＣＴＳの概念を導入し，この考え方に従って，設計から保守まで一貫した開発をおこなった。

ＣＣＴＳ（Closed Control Tree Structure）とは，狭義には，制御結合，データ結合関係を構造化した，高級言語のパッケージに近い概念である。また，広義には，このＣＣＴＳを用いた，設計から専用デバッグシステムを含む，一つの構造化プログラミング手法ということができる。

本手法は，高級言語によるプログラム作成の長所を取り入れ，短所を補ったシステム記述言語によるソフトウェア開発の一手法である。ＣＣＴＳ構造に適した領域管理と，独自のディスパッチングシステムの採用により，トップダウン開発，テストを容易にした。さらに，このＣＣＴＳ手法にあわせた専用デバッグシステムを導入し，多様なスタブ処理機能，デバッグ機能を実現し，効果的なトップダウンテストをインタラクティブに行なうことを可能とした。

ＣＣＴＳ手法の特徴を要約すると以下の通りである。

(1) 高級言語のパッケージに近い概念をシステム記述言語に取り入れ，設計段階から，プログラムの構造を簡明にした。

(2) ＣＣＴＳ構造にあわせた独自の領域管理ディスパッチシステムを取り入れ，信頼性の高いコーディング，柔軟なリンケージを実現した。

(3) ＣＣＴＳ構造に連動する専用デバッグシステムを開発し，ダミーモジュールを作成することなく，トップダウンデバッグを実現し，容易で強力なデバッグを可能とした。

１．ＣＣＴＳの概念

まず，われわれがどのようにして，ＣＣＴＳの概念に到ったかの思考過程を述べ，これによってＣＣＴＳの目的と考え方を明らかにしたい。次にＣＣＴＳの意味と構成を少し厳密に述べ，これがどのように目的と考え方に合致しているかをみる。

１．１　ＣＣＴＳの目的と考え方

昨今，ソフトウェアエンジニアリングは，盛んに研究されているが，実際にメーカが数１００Ksteps 以上の大規模ソフトウェアを作成する際に，実用となる技術は，案外少ない。これは，ソフトウェアエンジニアリングがある特定の分野，または問題に対して研究されたものが多く，汎用の大規模システムを作成する際の技術は，比較的少ないためである。

そこで，われわれはリレーショナルデータベースシステムを作成するにあたって，作成手法に対する要求を洗い出すことから始めた。その結果，以下の７点の機能要求があり，これらを同時に満たすものとして，ＣＣＴＳを考えた。

(1) 構造化プログラミング手法の徹底

(2) システム記述言語による高級言語のパッケージ機能の導入

(3) 専用デバッグシステムの必要

(4) 実行時性能の向上

(5) リエントラントプログラム作成の必要

(6) １００～２００ステップ単位のモジュールでの分割コンパイルの必要

(7) 例外処理をパッケージのマスタモジュールに集約し，このような出口へ直接に異常処理が出られること

従来，これらの一部の条件を満たすものは，存在したが，全部の条件を同時に満足するシステムは難しかった。例えば，ＰＬ／ＩやＦＯＲＴＲＡＮを採用して，(6)の条件を考えると，各モジュールは，外部手続きとなり，(2)の機能の導入が困難となる。ここで静的外部変数を用いてモジュール間の通信を行なえば，(5)の条件が満たされない。また，Ａｄａを用いれば(3)，(4)，(7)の実現が難しく，それ以上に，十分信頼できるＡｄａシステムが既に存在していなければならない。

従って，実際にシステムを作成する上で，われわれは，ＣＣＴＳ概念を導入してシステム記述言語で記述することによって，これらを同時に解決した。以下に(1)～(7)をＣＣＴＳでどのように解決したかを述べる。

(1), (2)については，図1のようにプログラム構造を階層化した。

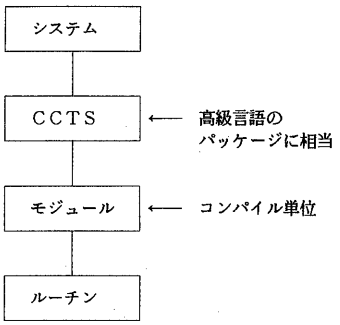


図1．プログラム構造の階層化

つまり，プログラマが一つのモジュールの作成にあたって，いくつかのルーチン群を用意することは，ごく自然であり，これらルーチン群は，モジュールの外からは，どのように動作しているか知る必要はない。一方，あるモジュール群をひとまとめにして，大きな機能単位として管理したい。これがＣＣＴＳと呼ばれるモジュール群を集めた袋であり，高級言語のパッケージ機能にあたるものとして，これを導入した。

(3) については，３．で詳述する。専用デバッグシステムの導入でダミーモジュールを作成することなく，スタブ機能を実現し，任意のモジュールを独立にデバッグ可能とした。

(4), (5)については，２．で詳述する。これは，ＣＣＴＳ手法に合せた領域管理とディスパッチ手法で解決した。例えば，ＣＣＴＳ対応，モジュール対応の寿命を持つ変数，及び任意の寿命を持つ変数と寿命による領域管理によって性能向上とリエントラント性を保証した。

(6)の分割コンパイルについては，各モジュールを高級言語の内部手続きとしたのでは，一般に達成されない。そこで，各モジュールをコンパイル単位として，(2)で述べたように，その上にＣＣＴＳの概念を導入した。

(7)の例外処理は，大きな機能単位であるＣＣＴＳの中で統一的に管理できるように工夫した。これは，各モジュール単位にはできるだけ例外処理を意識せず，ＣＣＴＳの中で一元的な管理を可能とした。

１．２　ＣＣＴＳの意味と構成

上記のＣＣＴＳの目的と考え方を具体的にどのように構成したかについて説明する。

ＣＣＴＳの構成ルールは次の通りである。

① ＣＣＴＳは一つのマスタモジュールとこれに従属するスレーブモジュール群により構成される。つまりＣＣＴＳはマスタモジュールをルート（根）とするモジュール群による木構造をなす。これがＣＣＴＳの名の由来でもある。

② ＣＣＴＳの各モジュールはマスタモジュールをのぞいて，他のＣＣＴＳより呼び出されることはない。つまり，ＣＣＴＳはマスタモジュールの唯一の入口点を経由して呼び出し得る。

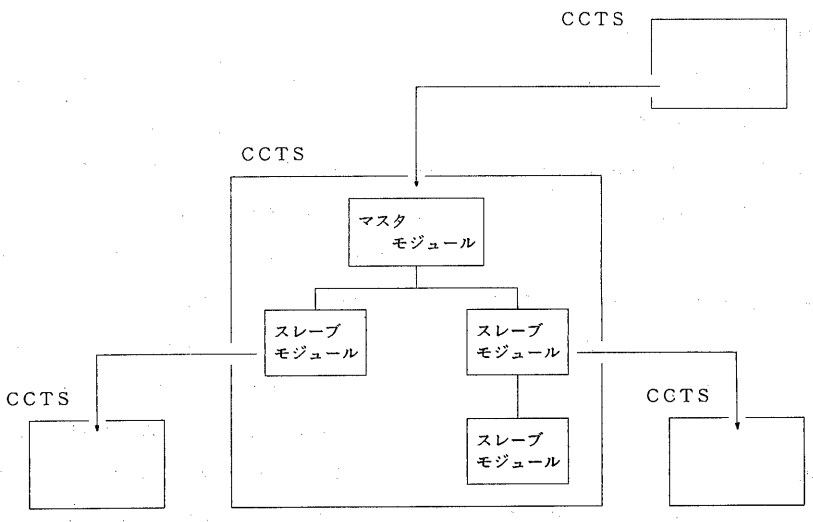


図2．ＣＣＴＳの構成

③ ＣＣＴＳのどのモジュールも他のＣＣＴＳを呼び出すとができる。

④ ＣＣＴＳのモジュールは，呼び出し元へ正常復帰する他に，マスタモジュールの例外処理出口へ直接復帰することができる。これによってＣＣＴＳ内の例外処理はマスタモジュールによって一元管理される。

⑤ ＣＣＴＳに固有の領域は，マスタモジュールが持つ。これはＣＣＴＳが木構造をしていることによって，ＣＣＴＳとマスタモジュールの寿命が一致していることを利用している。

⑥ 各モジュール対応の寿命域は，各モジュールの作業域に持つ。これを図示すれば図２のようになる。

２．ＣＣＴＳの実現手法

ここで，ＣＣＴＳの実現手法として，ＣＣＴＳのリンケージ，領域管理，ディスパッチ手法について少し具体的に述べることにする。

２．１　ＣＣＴＳのリンケージ

ＣＣＴＳでのモジュール間のリンケージは，すべてマクロによって展開する。プログラマは，プロローグコードやエピローグコードにわずらわされることなく，高級言語のイメージでプログラムを作成していくことができる。

まず，手続き宣言部(PROC)で，マスタモジュールかスレーブモジュールか，さらにスレーブモジュールの場合には，どのマスタモジュールに属するスレーブモジュールかを記述する。また，必要ならばここで，ＣＣＴＳ全体の例外処理出口も宣言しておく。

実際のエピローグコードは，手続きの終了宣言（END)のなかですべて展開し，呼び出し元への正常復帰処理（RTN)や例外処理出口への飛び出し(EXIT)は，ここへの飛び込みだけが展開される。

また，手続きの呼び出し(CALL)については，後述する。

PROC …… 手続きの宣言（マスタ／スレーブ），プロローグ処理，スタックプッシュダウン，リンケージ確立，共通制御表の取込み，例外処理出口の登録（マスタのみ）

END …… 手続きの終了宣言，エピローグ処理，スタックポップアップ，メモリ管理によって取得された領域の返却

RTN …… 正常復帰処理

EXIT …… 例外処理出口への飛び出し

CALL …… 手続きの呼出し

各マクロの関係を図３に示す。

２．２　ＣＣＴＳと領域管理

作業用の領域は，プログラム論理によって，その必要となる契機，不要となる時期が多種多様である。一方，作業域は必要になった時点で，そのつど取得でき，取得した領域の返却については，できる限り意識せずに済ませたいものである。

ところで，ＣＣＴＳ構造というプログラム構造から，その動きに合わせた適切な領域管理が構成できるはずである。ＣＣＴＳの構造を考

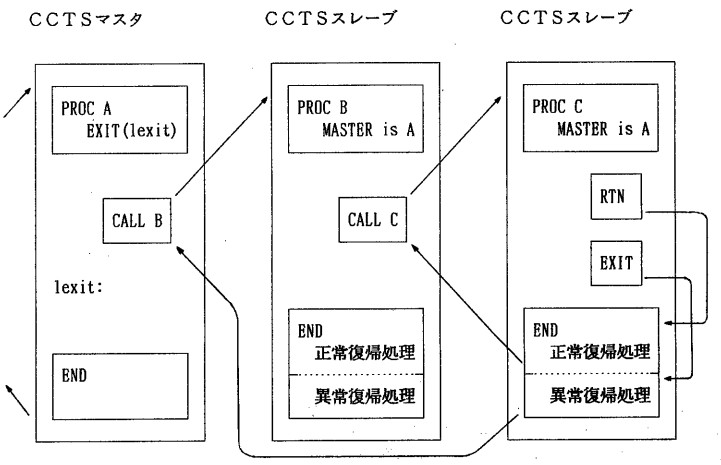


図3．ＣＣＴＳのリンケージマクロ

えると，作業域はその寿命から，次のように分類して考えることができる。

・ ルーチン固有のローカルな作業域（ルーチンからの復帰時には不要となる）

・ ＣＣＴＳ固有のローカルな作業域（ＣＣＴＳからの復帰時には不要となる）

・ ＣＣＴＳ間で受け渡される作業域

領域取得時にその寿命を規定することによって，その領域の性質を明示し，さらにＣＣＴＳのリンケージプロセスで自動返却することで，プログラムは，より簡潔に，わかりやすくなる。また，作業域をＣＣＴＳ構造に則して階層化することにより，情報隠蔽の効果もある。

こうした作業域の階層構造は，スタック，ヒープに共通して適用される。

（1）スタック

レジスタ退避域を含むＣＣＴＳリンケージに必要な領域や，大きさが静的に決定できるプログラム作業域，ＣＣＴＳ内共通作業域は，スタック形式の領域管理が適している。

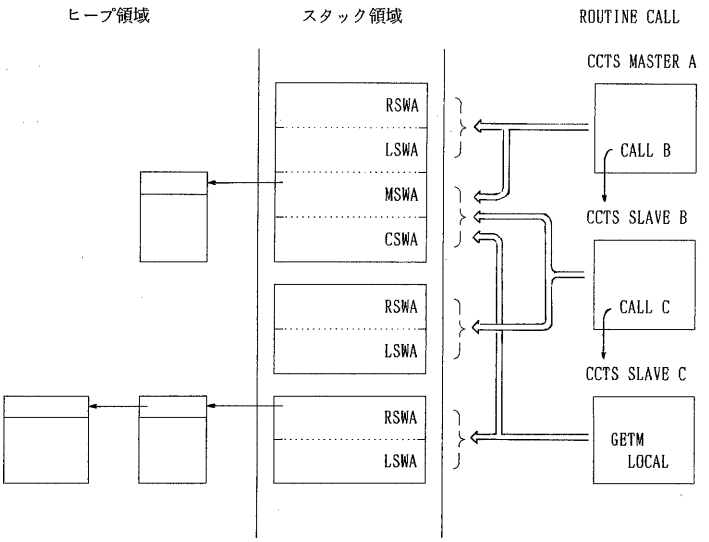
これらの取得，解放は，ＣＣＴＳのリンケージと同期して自動的に行う。

（2）ヒープ

扱うデータによって大きさの動的に変る作業域，ＣＣＴＳ間にわたる作業域は，スタックとは別に管理した方がよい。領域取得時に，その寿命（ルーチン内，ＣＣＴＳ内，ＣＣＴＳ間）を明示することによって，復帰処理内での自動返却を可能とした。

図４で，もう少し具体的に説明する。ＣＣＴＳは，はじめに必ずマスタが呼び出される。ここでは，Ａとした。ＰＲＯＣマクロの展開の中で，ＲＳＷＡ，ＬＳＷＡ，ＭＳＷＡ，ＣＳＷＡという領域がスタック上にとられる。このうち，ＲＳＷＡ，ＭＳＷＡはＣＣＴＳリンケージのための制御領域であり，プログラマが直接，参照・更新することがない。ＬＳＷＡは，ルーチン固有の，ＣＳＷＡはＣＣＴＳ内で共通の作業域である。

マスタモジュールＡが，スレーブモジュールＢを呼び出すと，ＣＣＴＳ共通の領域は，すでにＡのＰＲＯＣ展開中でとられているので，ＲＳＷＡ，ＬＳＷＡだけがスタック上に積み上げられる。さらに，スレーブモジュールＣが呼び出されれば，ＲＳＷＡ，ＬＳＷＡが積み上げられる。



RSWA (Register System Work Area)：レジスタリンケージ用制御領域
LSWA (Local System Work Area)：　ルーチン固有作業領域
MSWA (Master System Work Area)：　ＣＣＴＳ制御領域
CSWA (CCTS System Work Area)：　ＣＣＴＳ固有作業領域

図４．ＣＣＴＳと作業域

ヒープは，必要になった時点でマクロによって取得する。このとき，取得する領域の寿命を指定する。モジュール内でローカルな領域という指定があれば，ヒープの取得マクロは，要求された領域を切り出して渡すと同時に，ＲＳＷＡにつないでおく。ＣＣＴＳの中で寿命があるという指定があれば，ＭＳＷＡにつないでおく。こうすることによって，ＥＮＤマクロによる自動返却が可能となった。

２．３　ディスパッチャ

仮想空間へのプログラム・ロードの単位であるロードモジュールは，いくつかのルーチンより構成されるが，その組合せは，仮想空間の有効利用やロードモジュールのロード回数などの点から，最適な組合せを考えるべきである。また，こうした最適な組合せが，システムの初期作成時から決定できるわけではないから，あるルーチン群が同一ロードモジュール内に存在することを前提としたようなプログラム作りはすべきでない。

ＡＩＭ／ＲＤＢのディスパッチャは，ロードモジュール構成によらず，同一の呼び出し手続き(CALL)でのモジュール呼び出しを可能にする。すなわち，呼び出すモジュールがどのロードモジュールに属しているかに無関係に呼び出すことができる。

ＣＡＬＬマクロ，ロードモジュール管理テーブル，ディスパッチャの連係によって，対象モジュールの呼び出しを行なうが，図５に示すように，ロードモジュールがロードされていれば，直接，対象モジュールを呼び出し，ロードされていなければ，ディスパッチャが呼び出される。動的な判断をできるだけ行なわないことで，オーバヘッドは，きわめて少ないものとなっている。

ロードモジュールの構成は，いつでも変更可能であり，柔軟な対応ができる。

３．ＣＣＴＳとトップダウンテスト

大規模ソフトウェアの開発に於て，テストの占める割合は非常に大きい。ＡＩＭ／ＲＤＢの開発に当たってこのテストを効果的に行うために，トップダウンテストを採用した。ここでもＣＣＴＳの開発手法が有効に利用された。

トップダウンテストを支援するために，ＲＤＢＴＥＳＴと呼ぶテストシステムを導入した。このＲＤＢＴＥＳＴは，ＣＣＴＳの実現手法（リンケージの一元化と領域管理）を利用することによって，多様な

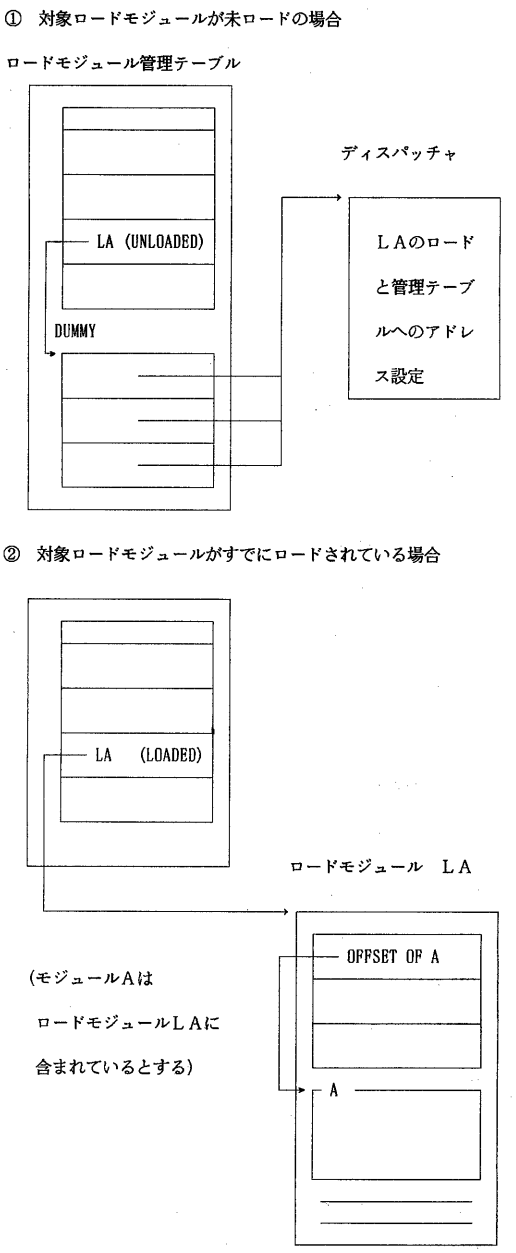


図５．ディスパッチシステム

スタブ支援機能やデバッグ支援機能を提供する。このシステムを使ってトップダウンテストをインタラクティブに効率よく行なうことができた。

ＣＣＴＳのモジュール構成の考え方をトップダウンテストの進め方に応用した。このことによって，トップダウン手法の効用を十分に引き出すことができた。ここでも，ＲＤＢＴＥＳＴはＣＣＴＳの汎用的なテストドライバーとして重要な役割を演じた。

なお，このＲＤＢＴＥＳＴはFACOM OSIV／F4 TSSのTESTコマンド（以下，RDBTESTと区別するためにTSS TESTと呼ぶ）のもとで動作する。

３．１　ＲＤＢＴＥＳＴの機能

ＣＣＴＳでは，モジュール間のリンケージが一元管理されている。ＲＤＢＴＥＳＴはこのリンケージと領域管理をシミュレートすることによって，モジュール間の制御の受渡しに介入することができる。

そして，このシミュレートによって多様なスタブ支援機能やデバッグ支援機能を提供する。ＲＤＢＴＥＳＴのコマンド一覧を表１に示す。

主なスタブ処理機能には，次のようなものがある。

- モジュール呼出しを単に迂回する（SKIPコマンド）。
- モジュールの呼出し，復帰時点でTSS TESTのモードへ移行し必要な処理を行う（TESTコマンド）。
- モジュールの呼出し，復帰時点で予め適当な処理が設定されている中断点へ分岐しその処理を実行する（TESTコマンド）。
- モジュールの実行に必要なテーブル等をロードする（LOADコマンド）。
- 呼出しモジュールを変更する（RROGコマンド）。

さらに，デバッグ支援機能として次のようなものがある。

- モジュール間の呼出し，復帰に関して，各モジュールの開始・終了の時点でモジュール名を出力する（LABEL コマンド）。
- モジュール間の呼出し，復帰時にレジスタの内容を出力する（TRACE コマンド）。
- 任意のモジュールの呼出し・復帰時点で，それまでに呼び出されたモジュールの呼出し・復帰時のレジスタトレースを出力する（LISTコマンド）。

その他，実行を制御するものとして，モジュールの呼出点に中断点を設定するコマンド（INTERRUPT コマンド）がある。また，スタブ処理を効果的に行うために，ＲＤＢＴＥＳＴの用意した中断点に予め適当な処理を登録しておく機能などがある。

表１．RDBTEST のコマンド一覧

コマンド	機　能
CALL	テスト開始モジュールの指定。
END	テストの終了。
INTERRUPT	モジュール呼出し時の中断点の指定。
LABEL	ラベルトレースの設定・解除。
LIST	トレース情報，レジスタの内容の出力。
LOAD	テーブル，モジュール等のロード。
REND	ABEND 出口の終了（TEST終了）。
RGO	中断点の終了，実行の開始。
PROG	呼出しモジュールの変更。
SKIP	呼出しモジュールの実行の迂回，中断点への分岐。
TEST	TSS TESTモードへの移行。
TRACE	モジュール呼出し時のレジスタトレースの設定。

３．２　ＲＤＢＴＥＳＴの実際

ＲＤＢＴＥＳＴは，TSS TESTのもとで使用される（図６参照）。TSS TESTは，プログラムのインタラクティブデバッグを支援するためのシステムであり，中断点の設定，メモリの内容表示，メモリへの値設定などの様々な機能を持っている。

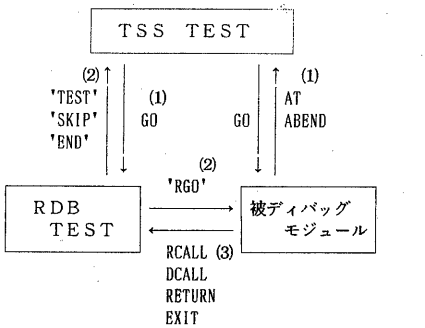


図６．RDBTEST の動作環境

被デバッグモジュールは，TSS TESTの下で動作するＲＤＢＴＥＳＴから起動される。このモジュールのデバッグは基本的にはTSS TESTの豊富な機能を用いて行われる。そして，他のモジュールを呼び出そうとする場合にＲＤＢＴＥＳＴが介入する。このとき，ＲＤＢＴＥＳＴの提供する機能を用いて適切なスタブ処理を行うことができる。

図７にＲＤＢＴＥＳＴの使用例を示す。この例では，モジュールJXZTPZ00がテストされているが，このテストの時点ではこのモジュールが呼び出すJXZTPZ03はまだ作成されていない。そこで，JXZTPZ03の呼出し時にはスタブの処理を行わねばならないが，この例ではスタブルーチンを利用するのではなく，呼出し時にTSS TESTのモードへ移行し，TSS TESTの機能を使って代行している。

```
 RDB
CALL JXZTPZ00              ←①テスト開始点の指定。
 ENTRY JXZTPZ00
 RSET
INTERRUPT JXZTPZ03         ←②呼出し時の中断点の設定。
 RSET
LABEL ON                   ←③ラベルトレースの指定。
 RSET
TRACE JXZTPZ02             ←④呼出し時のレジスタトレースの指定。
 RSET
RGO                        ←⑤実行の開始。
 ENTRY JXZTPZ01            ←⑥ラベル及びレジスタのトレース。

 ENTRY JXZTPZ02
  0R  20285059   1R  000CDD08   2R  600DCF06   3R  000DBBE0
  4R  00000000   5R  00000000   6R  000DCF6C   7R  00000000
  8R  00000004   9R  000DC000  10R  400CD9C4  11R  000CDCA4
 12R  000CE008  13R  000DCF6C  14R  000DCF54  15R  000DCE28
 RETURN JXZTPZ02
  0R  20285059   1R  000CDD08   2R  600DCF06   3R  000DBBE0
  4R  00000000   5R  00000000   6R  000DCF6C   7R  00000000
  8R  00000004   9R  000DC000  10R  400CD9C4  11R  000CDCA4
 12R  000CE008  13R  000DCF6C  14R  600DCF62  15R  000DCE28
 ENTRY JXZTPZ03
 RSET
SKIP                       ←⑦呼出しモジュールの迂回。
 RETURN
 RSET
TEST                       ←⑧TSS TESTモードへの移行。
 ..... TEST MODE START .....
 TEST
       (省略)
GO
 RSET
RGO                        ←⑨中断点からの再実行。
 RETURN JXZTPZ01
 RETURN JXZTPZ00
 RDB
```

図７．RDBTEST によるテストの実行例

このように，ＲＤＢＴＥＳＴとTSS TESTの機能を上手に組み合せることによって効果的なデバッグを進めることができる。そして，ＲＤＢＴＥＳＴの多様な支援機能により，スタブの処理に柔軟に対応でき，ここではスタブルーチンはほとんど不要となった。

３．３　トップダウンテスト

トップダウンとは言っても，現実にはシステム全体を上から下へ画一的にテストしていくことはできない。

実際のテストの進め方にはＣＣＴＳのモジュール構成の考え方を応用した。ＣＣＴＳはあるひとまとまりの機能を提供する。そこで，一つ又はいくつかのＣＣＴＳを単位としてトップダウンテストを適用した。

一方，前にも述べたようにＲＤＢＴＥＳＴはＣＣＴＳの一元化されたリンケージをシミュレートしており，すべてのＣＣＴＳに対して，テストドライバーとなっている。そしてＲＤＢＴＥＳＴを使うことによってすべてのＣＣＴＳは並行にテストできる。

このようなテストの進め方では，ＣＣＴＳ単位のテストがトップダウンに行なわれるために各ＣＣＴＳ間のインタフェースが早期に検証できる。ここでは，ＣＣＴＳの入口が一つということが重要なポイントとなっている。

また，比較的小さなまとまり毎に十分なテストが実施できると共に，デバッグも容易になる。ここでは，ＲＤＢＴＥＳＴの機能が活かされる。

さらに，汎用的なテストドライバーによって，テストのスケジューリングや人の配分など，マネージメントの問題がくみし易くなった。

４．まとめ

プログラムとは，一定の目的のデータ処理を行なうための体系であり，この体系には階層がある。この体系の自然な階層化という観点を逆に規約化してしまうと，コンポーネント向きの独立性が高く，メンテナンスの容易なプログラミング方法が浮びあがってくる。これが，ＣＣＴＳであったということができよう。

このＣＣＴＳ手法によって，特殊な手法を用いず，一般的な手法の組合せで大規模システムの開発を可能とした。このようにＣＣＴＳ手法の最大の強みは，その「自然さ」にあるということができる。

また，専用デバッグシステムＲＤＢＴＥＳＴは，ＣＣＴＳの一元化されたリンケージを使用することによって，各種のツールの導入をも容易とし，インタフェースのフォーマットダンプや，シンボリックデバッグ用の仕掛けの作成も可能とした。ＲＤＢＴＥＳＴはすべてのモジュールの汎用的テストドライバーとして，たいへん拡張性に富んだものとして構成できた。

【参考文献】

[1] Barry W Boehm "Software engineering - as it is", Proceedings ICSE pp.11-pp.21 (1979).

[2] Anthony I. Wasserman and Steven Gutz "The Future of Programming", CACM pp.196-pp.206 (March,1982).

[3] Flaviu Cristain "A recovery mechanism for moduler software", Proceedings ICSE pp.42-pp.50 (1979).

[4] Edsger W. Dijkstra "The structure of "THE"-Multiprogramming system", CACM pp.341-pp.346 (May,1968)

[5] ADA reference manual, SIGPLAN Notices 6 (1979).

[6] "FACOM OSIV/F4 TSSコマンド文法書"