Vol.2010-SE-169 No.8
2010/7/23

大規模情報制御システムの段階マイグレーションにおけるプログラム動作等価性検証に関する考察

望月智之[†]　志村明俊[†]
加藤博光[†]　武澤隆之[††]

大規模情報制御システムの段階マイグレーションでは，社会インフラを担うシステムであるため，現行機能の担保が重要となる．そのため，段階マイグレーション時に混在する新旧モジュール間の影響範囲の極小化と，現行機能担保の確認が必要となる．本稿では，情報制御システムのプログラム構成の特徴に着目し，影響範囲を極小化するモジュールの抽出方式と，機能担保を確認するためのタスクレベルでの新旧プログラム実行による等価性検証方式を提案する．

Discussion about Program Behavior Equivalence Check on Gradual Migration of Large-Scale Information Control Systems

Tomoyuki Mochizuki[†] , Akitoshi Shimura[†],
Hiromitsu Kato[†], and Takayuki Takezawa[††]

On gradual migration of large-scale information control systems, keeping current functions is important. It needs minimization of influences between old and new modules on gradual migration and confirmation of keeping current functions. In this paper, focusing on characteristics of program composition of information control systems, we propose an extraction method for modules to minimize influences. And we propose a behavior equivalence check method with execution of old and new programs by a task unit to confirm keeping current functions.

1. はじめに

鉄道システムや電力システムのような大規模情報制御システムでは，長年の運用に伴う改造・改修により，プログラムが複雑化・肥大化してきている．そこで，生産性・保守性の向上や新しいニーズへの追従に向け，ソフトウェアのマイグレーションが叫ばれている．このマイグレーションは，現行の複雑化・肥大化したプログラムの機能を踏襲しながら，生産性・保守性の高いプログラムを作成する．その際，システムが大規模で，人員確保の面で一度にすべてを置き換えることが困難なため，時間をかけて段階的に行うことになる．

情報制御システムは社会インフラを担うシステムであるため，システムの挙動が変化してしまうことは大きな問題となる．段階マイグレーションの過程において，マイグレーションを行っていない旧モジュールと，マイグレーションにより新規に開発した新モジュールが共存する．そのような場合であってもシステムとしての挙動を変化させないためには，新旧モジュール間の影響範囲を極小化するようにモジュールを抽出する必要がある．また，マイグレーションの前後でプログラムの機能が担保されてことを確認する手段が必要となる．

従来のマイグレーションでは，情報隠蔽[1]によるモジュール分割[2]やリファクタリング[3][4]を行いながら，現状のプログラムからモジュールを抽出し，再利用可能なソフトウェア資産[5][6]を作成していく．そして，同値分割や限界値分析，設計モデル[7]などを利用して，テストを網羅的に行うことで現行機能担保確認する．

しかし，情報制御システムは，スループットをあげるために共有メモリを使用した構成となっているため，簡単にモジュールを抽出することはできない．また，抽出して再構成した新モジュールが現行機能担保をしているかを確認するためには，単純にはその新旧モジュール間をテストで比較すれば良い．しかし，現行プログラムから抽出する関数群は元々他の関数郡と密接に関係しているため，現行プログラムを変更することなく比較対象となる旧モジュールだけを抽出することは難しい．

本稿では，情報制御システムのプログラム構成の特徴に着目し，影響範囲の極小化する機能モジュールの抽出方式と，現行機能担保を確認する動作等価性検証方式を提案する．抽出方式では，ロジックとデータアクセスの分離と，制御対象によるロジックの分離に着目して，データアクセスの共通関数を分離し，ロジックの共通関数を独立化してモジュール化する部分を抽出する．検証方式では，段階マイグレーションを

[†] 株式会社 日立製作所 システム開発研究所
Systems Development Laboratory, Hitachi, Ltd.
[††] 株式会社 日立製作所 情報制御システム社
Hitachi, Ltd., Information & Control Systems Div.

適用した新モジュールと適用していない旧モジュールが混在したものを新プログラムと捉える．そして，現行の旧プログラムと新プログラムを，機能単位であるタスク（プロセス）単位で比較することで，新モジュールで現行機能担保ができているかを確認する．

2. 大規模情報制御システムの段階マイグレーションにおける課題

2.1 大規模情報制御システムの構成と特徴

大規模情報制御システムは，図 1 のように，計画系の情報システムと，設備系の制御システムの中間に位置するサブシステム（装置）の集合である．計画系の情報を受けて，設備をどのように動かすかを決定し，設備系システムに制御指示を発行する．

サブシステムは，複数のタスクと共有メモリによって構成される．タスクは，過去の CPU やメモリが貧弱な環境において，一度に処理できる単位で機能を纏めたものである．タスク間は共有メモリを使用してデータを共有する．サブシステムは，これらのタスクを順番に実行することで，設備のコントローラへの制御命令の発行や，ユーザへの情報提供を行う．

大規模情報制御システムは長期間利用される社会インフラであるため，信頼性が重要となる．そのため，システムを一度構築した後の改造・改修では，既存のプログラムには手を加えず，継ぎ足し開発を行うことになる．このような部分開発が長年続くと，若手への技術伝承が行われなくなり，システム全体を俯瞰できる人材が減少する．これに伴い，プログラムの構造が，修正時の影響範囲がわかりにくい，保守性の低いものとなる．いわばプログラムのブラックボックス化というべき状況である．

2.2 段階マイグレーションにおける課題

ブラックボックスから脱却し，生産性・保守性の向上や，新しいニーズへの追従に向け，ソフトウェアマイグレーションが叫ばれている．情報制御システムは規模が大きいため，一度にすべてを置き換えることは難しい．そのため，図 2 のように，複雑化・肥大化したプログラムの中から再利用可能なモジュールを抽出してプログラムを再構成するマイグレーションを段階的に繰り返すことにより，最終的に全体を置き換える手法を取らざるを得ない．

この段階マイグレーションを行うためには，下記のような課題がある．

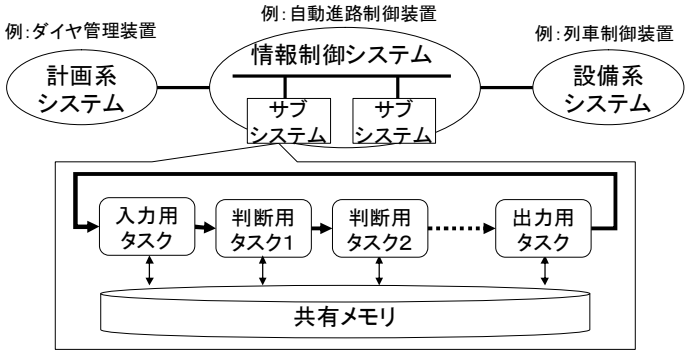

図 1 情報制御システムの構成

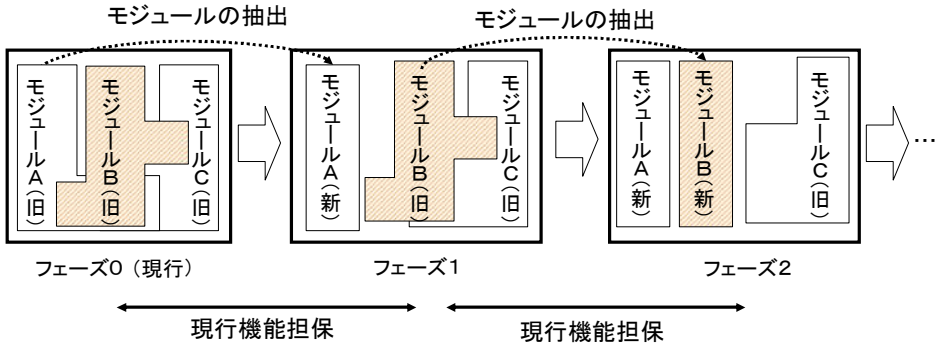

図 2 段階マイグレーション

(1) 影響範囲の極小化

段階マイグレーションの過程では，マイグレーションを行っていない旧モジュールと，マイグレーションにより新規に開発した新モジュールが共存することになる．そのような場合であっても，システムとしての挙動を変化させないためには，新旧モジュール間の影響を極小化するように，モジュールを抽出する必要がある．

保守性の高いソフトウェア資産[5][6]となるモジュールを抽出するためには，情報隠蔽[1]によるモジュール分割[2]やリファクタリング[3][4]により，重複部分を除去し，モジュール間の関係が疎結合にすることが知られている．しかし，情報制御システムは，共有メモリを介してデータを共有するように設計されており，共有メモリを切り離すことは難しい．システムが大規模で変更時の影響範囲の見積りが容易ではないため，マイグレーション後も共有メモリを前提とした構成においてのモジュール抽出が必要である．

(2) 現行機能担保の確認

情報制御システムは，長期運用される社会インフラを担うシステムであり，現行動作している機能をなくすことはできない．段階マイグレーションの各過程において，マイグレーション前後のプログラムの間で機能が担保されていることを確認する必要がある．

プログラムの動作が担保されているかの確認は，テストを網羅的に行うことでの確認が主流である．同値分割や限界値分析による古典的なテストのほか，設計モデルを作成し，そこから網羅的なテストケースを生成するモデルベーステストがある．

そこで，段階マイグレーションにより開発した新モジュールが，現行プログラムの該当部分から現行担保されているかを確認するためは，単純にはその新旧モジュール間をテストで比較すれば良い．しかし，現行プログラムから抽出する関数群は元々他の関数群と密接に関係しているため，現行プログラムを変更することなく比較対象となる旧モジュールを抽出することは困難である．よって，段階マイグレーションの対象であるモジュール単位でのテストは難しい．

また，モジュールベーステストを適用する場合には，新規に開発した新モジュールだけではなく，旧モジュールについてもモデルを作成する必要がある．これは作業工数的に困難であるため，他の方法を取ることになる．

よって，段階マイグレーションを適用していない旧モジュールがある場合であっても，段階マイグレーションを行ったモジュールの現行機能が担保されていることを確認する手段が必要である．

3. 課題解決に向けたアプローチ

3.1 モジュール抽出方式

段階マイグレーションの過程において，新旧モジュール間の影響を極小化するようにモジュールを抽出するために，情報制御システムのプログラムの構成に着目する．

情報制御システムでは，計画系データ，設備系データ，前の処理での実行結果を，共有メモリ上のテーブル（変数）に保存する．これらのテーブル間には関係があり，テーブル間のデータを組み合わせて加工し，判断処理の材料となるデータを作成する．そのため，情報制御システムのプログラムでは，この加工処理を様々なところで実行する．そこで，ロジックとデータの分離の観点から，共有メモリからデータを取得・加工するデータアクセス処理を共通処理として分離する．

また，情報制御システムでは，制御対象を判定してしてから，それらに対する判断処理を実行する．そのため，制御対象ごとに処理を纏めることができる．ただし，過去のリソースの少ない時代に，判断処理の中で共通的に使われているものがある．これらのロジックの共通関数は，長年の継ぎ足し開発の過程で複雑化してきている．特に，開発当初は共通関数となっていたが，途中で制御対象判断による分岐処理を行う拡張が行われたものもある．そこで，制御対象ごとのモジュールに分離し，ロジックの共通関数は使われるモジュールの数だけ用意する．共通関数の中には実行されない処理があるので，段階マイグレーションの過程で不要部分を除去する．重複するロジック共通関数の分だけメモリが余分にかかるが，モジュール間で呼び出し関係がなくなり，これによりモジュール間の関係が疎結合になる．

纏めると，図 3 のように，まずデータアクセスの共通関数である F7，F8 を分離する．さらに，制御対象ごとにロジックをモジュール化する．その際，ロジックの共通関数である F5 は制御対象別のモジュールで別々の関数として扱う．これにより，モジュール間が疎結合になるため，段階マイグレーションにより新モジュールに置き換えた場合でも，他の部分への影響範囲は極小になる．

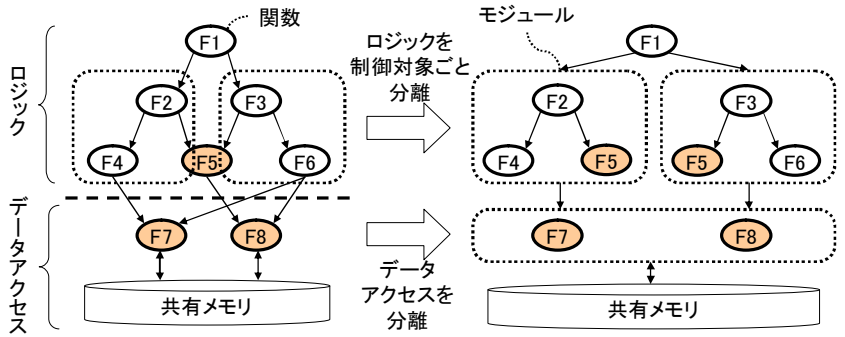

図 3 モジュールの抽出方式

3.2 動作等価性検証方式

段階マイグレーションにより作成した新モジュールが，現行プログラムの該当する部分との間で現行機能担保していることを確かめるために，段階マイグレーションを適用したモジュール単位ではなく，タスク単位で動作等価性検証を行う．タスク単位であるのは，タスクが情報制御システムにおける機能の単位であり，他のタスクとの接続が共有メモリだけとシンプルであるため，テストの単位として適切であるためである．

そこで, 図 4 のように, 段階マイグレーションにより作成した新モジュール（F2, F4, F5 を再構成したもの）と，まだ段階マイグレーションを適用していない旧モジュール（F3, F5, F6, F7, F8）が混在したものを新プログラム（新タスク）と捉える．そして，現行の旧プログラム（旧タスク）と新プログラム（新タスク）を同一の環境下で実行し，実行結果が等価であることをテストにより確認する．

段階マイグレーションによって新モジュールに変わった部分以外は，旧プログラムと新プログラムとの間の構造は同じである．比較対象となる関数群を現行プログラムから抽出することは難しいが，それらを含む旧プログラムと新プログラムを実行することは可能である．機能単位であるタスク単位で新旧プログラムを比較することにより，新モジュールで現行機能担保ができているかを確認する．

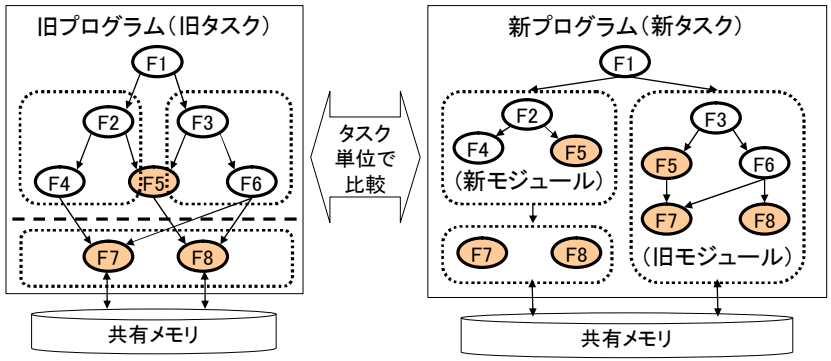

図 4 動作等価性検証方式

4. 鉄道運行管理システムへの適用試作

4.1 鉄道運行管理システム

鉄道運行管理システムの段階マイグレーションを想定して，上記で説明した新旧プログラムの共存実行による動作等価性検証機能の試作開発を行った．システム構成を図 5 に示す．本試作での鉄道運行管理システムは，自動運行制御装置(PRC: Programmed Route Control)と，指令員が利用する制御卓と，列車や信号機等の設備の動作を模擬する列車走行シミュレータからなる．制御卓と PRC が図 1 の情報制御システムであり，列車走行シミュレータが制御系システムである．計画系システムは含まないが，計画系のデータであるダイヤデータを稼動時の初期値として与える．

本試作でのマイグレーションの対象は，PRC で動作する列車が出発しても良いかを判断する出発判断タスクである．システムが適切に動作しているかは，制御卓の画面を見ることにより確認する．

4.2 新旧モジュール共存アーキテクチャ

3.1 節で示したモジュール抽出方式に従い，新旧モジュールを共存実行するための新旧モジュール共存アーキテクチャを図 6 に示す．新旧モジュールが共存する新プログラムは，新モジュール，旧モジュール，辞書，コントローラからなる．

新モジュールは，段階マイグレーションにより現行プログラムから抽出して再構成したロジック処理の集合である．旧モジュールは，段階マイグレーションを適用する前の処理の集合である．

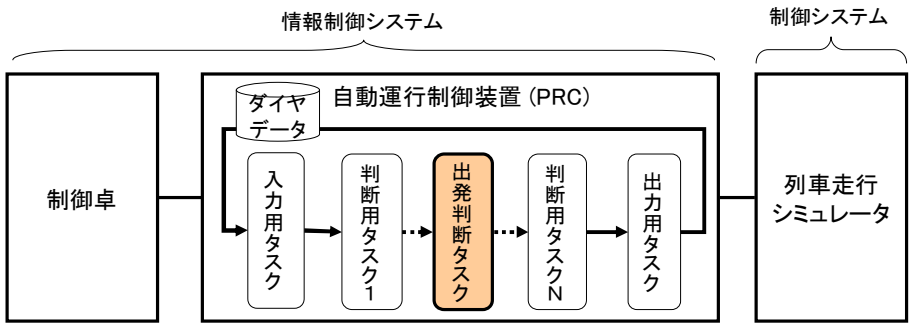

図 5 鉄道運行管理システム

辞書は，データアクセスの共通関数を纏めたものである．ロジックから共有メモリに自由にアクセスできると，共有メモリを介して他の機能モジュールに影響を与える可能性がある．辞書は，新モジュールが共有メモリにアクセスするためのインターフェースを提供し，ロジック処理からの共有メモリへのデータアクセスの統制をかける．

コントローラは，現行プログラムの基底関数に当たるものである．コントローラは新旧モジュールを管理し，新旧モジュールを順次呼び出すことでタスクとしての機能を実現する．

4.3 新旧プログラムの動作等価性検証

3.2 節で示した動作等価性検証方式に従い，タスクレベルで新旧プログラムを同一環境下で動作させて，実行結果を比較する．新旧プログラムを同一環境下で動かすためには，「プログラムの入出力」と「プログラムの内部状態」を同じになるようにする必要がある．

今回試作適用する鉄道運行管理システムの判断用タスクは，一定時間内の応答性能の確保，排他制御によるデータ整合性の確保を実現するために，下記のように動作する．

- タスクは周期的に順番で呼び出される．
- タスクの実行途中で割り込みが発生しない．
- タスク間のデータのやり取りは共有メモリ経由で行う．
- タスクの内部状態は共有メモリに保存する．
- 判断用タスクでは，ファイル入出力やネットワーク通信などの外部入出力はない．

そのため，同一環境下で動作させるためには，入出力と状態の保存に使用する共有メモリを同一にすればよい．そこで，新旧プログラムの実行でデータが上書きされないよう，図 7 のように，共有メモリを 2 つ用意する．共有メモリ（正）は元々タスク間でデータをやり取りするために使用していたものである．共有メモリ（副）は，新プログラムを動作させるために用意したものである．

具体的に，図 8 を用いて，コントローラが新旧プログラムをどのように実行するかを説明する．まず，前タスクからの入力となる共有メモリ（正）のデータを共有メモリ（副）にコピーする．次に，共有メモリ（正）を利用して，旧プログラムを実行する．そして，共有メモリ（副）を利用して，新プログラムを実行する．最後に，共有メモリ（正）と共有メモリ（副）のデータを比較し，比較結果をレポートして出力する．共有メモリ（正）には旧プログラムの実行結果が，共有メモリ（副）には新プログラムの実行結果が格納されるため，両者の共有メモリの内容を比較することで，新旧プログラムの実行結果が等しいかを判断することができる．これを一定期間連続運転し，比較結果レポートの内容が常に比較結果が等しいとなっていれば，新旧プログラムの動作が等価であることが確認できる．

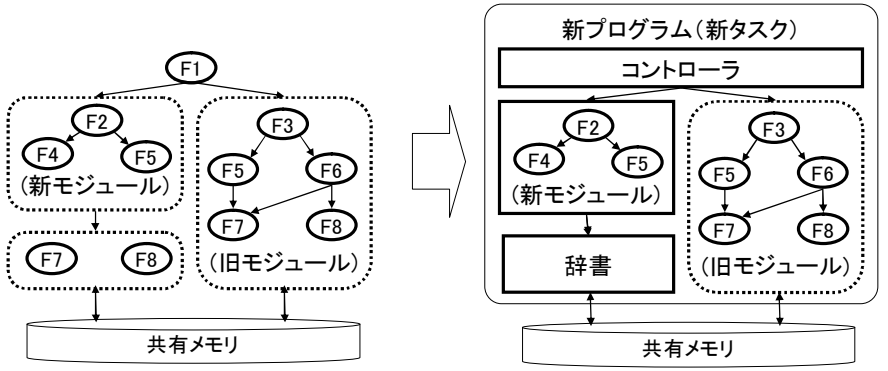

図 6 新旧モジュール共存アーキテクチャ

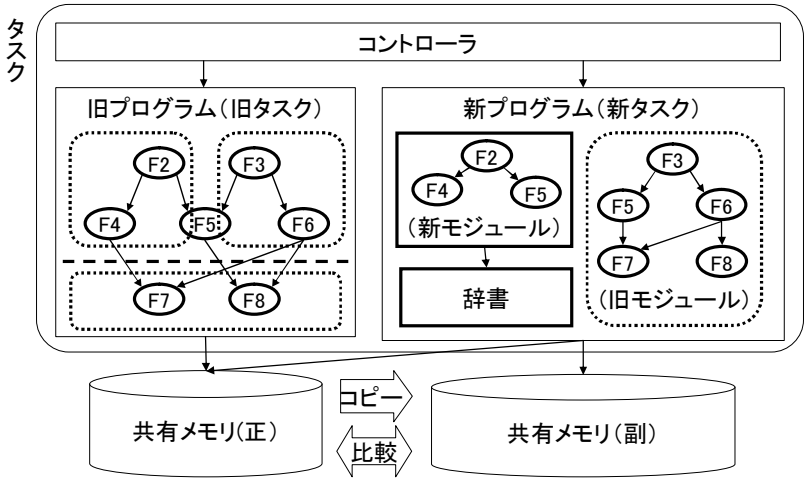

図 7 新旧プログラムの動作等価性検証

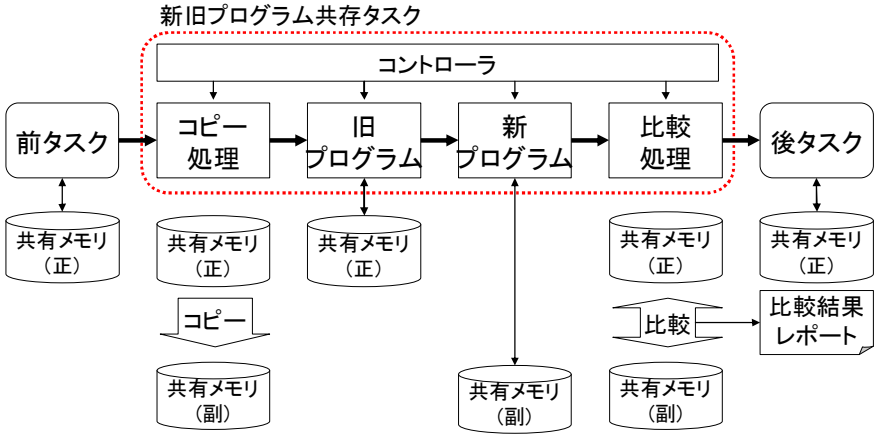

図 8 新旧プログラムの共存実行（共存実行モード）

4.4 実装上の課題と解決策

(1) 実行モードの切り替え

段階マイグレーションを進める過程で，すべてのタスクを共存実行すると，サブシステム全体の性能に影響を与えてしまう．旧プログラムから新プログラムに変更する部分に関してのみ動作の等価性を検証すれば良い．

そこで，実行モードの概念を導入し，実行モードを変更できるようにする．動作等価性を検証するタスクは，図 8 のように共存実行モードで実行する．検証が完了したタスクは，図 9 のように新プログラムだけ（もしくは旧プログラムだけ）を実行する単独実行モードに変更する．コントローラは実行モードに従い，呼び出すプログラムを切り替える．

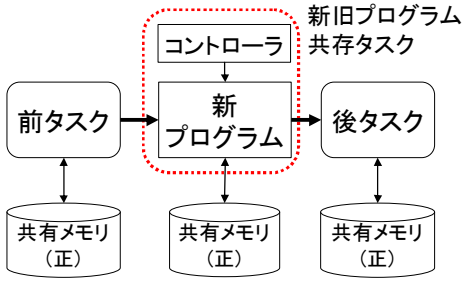

図 9 単独実行モード

(2) コピー・比較範囲の局所化

タスクが使用する共有メモリ上のデータは数百 MB ある．全データをコピーすることになると，共有メモリとして用意するサイズが従来の 2 倍となり，共有メモリの確保が難しい場合がある．また，全データのコピーや比較が，少なからず全体の性能に影響を与える可能性がある．そこで，必要とするメモリ量の削減と，コピーに要する時間の短縮のために，必要なデータのみをコピー・比較する．

タスクが使用する共有メモリ上のデータには，表 1 のような 3 種類のデータがある．このうちタスクで変更しないデータはコピーする必要がないため，(c)の変更する可能性のある変数データのみをコピー対象とする．そこで，共有メモリ（副）には(c)のデータのみを格納する．また，プログラムの実行により(c)のデータのみ変更になるため，実行結果の比較では(c)のデータのみを比較する．

(c)のデータの抽出は，データフロー解析により実現する．ソースコードから共有メモリのアドレスを格納する変数を検出し，その変数を通して値が代入されているかを調査する．代入している変数が(c)のデータである．

(3) データアクセス先の切り替え

実行モードの切り替えにより，新プログラムが利用する共有メモリ上のデータのアクセス先は変更になる．また，コピー・比較範囲の局所化により，データの種類に応じて，アクセス先の共有メモリが異なる．纏めると，表 2 のようになる．

表 1 データ種別

#	分類	データ種別
(a)	定数	システム起動時から変更することのない定数データ
(b)	変数	他のタスクからの入力であり，本タスクでは変更しない変数データ
(c)		他のタスクへの出力であり，本タスクで変更する変数データ

表 2 共有メモリのデータアクセス先

	旧プログラム	新プログラム	
	―	共存実行モード	単独実行モード
(a)定数データ	正	正	正
(b)変更なし変数データ	正	正	正
(c)変更あり変数データ	正	副	正

そこで，新プログラムでは，図 10 のように，データ名とデータアクセス先の共有メモリのアドレスの対応表を利用する．コントローラは，起動時または実行モード切り替え時に対応表を作成する．新プログラムは，共有メモリ上のデータにアクセスする際には，対応表を用いて適切なアドレスを取得し，データを読み書きする．

旧プログラムは，常に共有メモリ（正）を参照するため，旧プログラムのソースコードを特に変更する必要はない．しかし，新プログラムに含まれている新旧モジュールのアクセス先は変更になる．新モジュールは辞書を通して共有メモリにアクセスするため，辞書が共有メモリにアクセスするときに対応表を参照するようにすれば良い．一方，旧モジュールは辞書を使用しないため，ソースコード変換をかける．旧モジュールのソースコードに対してデータフロー解析を行い，共有メモリにアクセスする場所を抽出し，対応表を介して共有メモリにアクセスするように変換する．

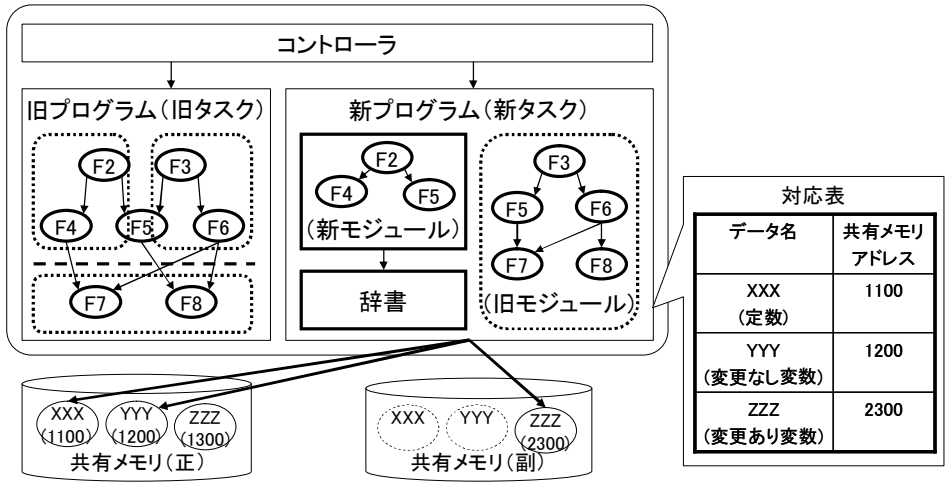

図 10 データアクセス先の切り替え

5. 考察

(1) 試作適用の結果

鉄道運行管理システムへの適用試作を実行したところ，新旧プログラムの双方を動かした場合でも，システム全体を問題なく動かすことができた．また，新プログラムにバグを入れた場合に，システムがバグに関係するデータの値が異なるという比較レポートを出力した．そして，タスクのソースコードの中から，そのデータを書き込む部分を検索し，問題箇所を特定することができた．そのため，本方式は現行機能担保を確認する上で有効な手段であると考える．

(2) 他のタスクへの影響

他のタスクに影響を与える要素には，データと動作タイミングの 2 つがある．

データは，他のタスクとは共有メモリ（正）を介して交換する．本方式で共存実行する場合，共有メモリ（正）には，長年運用された旧プログラムの実行結果が格納される．検証対象である新プログラムの実行結果は格納されない．そのため，新プログラムに問題があってもデータの面で他のタスクに影響を与えることはない．

一方，動作タイミングは，プログラムの処理時間が変わることにより，他のタスクが実行されなくなることや，判断結果が変更になるといった影響を与える可能性がある．しかし，今回の新旧プログラムの共存実行では，1 タスク分の処理時間が追加されてもタスク群の 1 周期の時間の範囲に収まっていた．また，OS やミドルウェアによりタスクの実行周期，実行順序，排他制御が保証されていた．そのため，他のタスクが実行されなくなるという影響はなかった．よって，1 周期の時間の範囲内に収まるように共存実行を設定すれば影響がないと考える．

また，プログラムの処理時間が変わると，現在時刻情報にも影響を与える．ただし，情報制御システムでは，一連のタスクの開始時の時刻を現在時刻として使用するため，一連のタスクの実行時間が 1 周期の時間内に収まっていれば問題は発生しない．情報制御システムでは，現在時刻を元に計画系システムの現在の計画内容を取得し，制御系の現在の状態情報と照合することで，次の制御内容を決定する．その際，タスクによって取得した現在時刻の情報が異なると，サブシステムとしての出力結果の整合性が保てなくなる．そこで，一連のタスクの開始時の時刻を現在時刻として扱う．また，共存実行する旧プログラムと新プログラムも，同じ時刻を現在時刻情報の入力として使用するため，新旧プログラムの実行環境も同じものとなる．

なお，本方式では，1 つのタスク内で新旧プログラムを実行するため，タスクの実行時間は新旧プログラムの実行とコピー・比較分と 2 倍以上に伸びる．しかし，情報制御システムは，元々十数年前のハードウェアで動作していたものであり，近年のハードウェアの進化により速度の向上が進んでいるため，大きな影響はないと考える．また，近年の CPU のマルチコア化に伴い，新旧プログラムを別々のコアで実行することも考えることができ，その場合にはオーバーヘッドを最小限に抑えることができる．

(3) 現行担保の範囲

マイグレーションでは，プログラムの処理の実装方法の変更，プログラムが利用するデータ構造の変更の 2 方面の変更がありうる．

処理の実装方法の変更では，現在のプログラムの構造をどこまで残すかによって分類できる．本方式は，タスクレベルで新旧プログラムを比較しているため，タスク構造が従来通りであるならば,新旧プログラムの動作等価性の検証を行うことができる．しかし，サブシステム内のタスク構成の変更や，システム内のサブシステムの構成の変更がある場合には，そのまま適用することは難しい．

一方，データ構造の変更の場合は，単純に本方式を適用することはできない．利用するデータが増えただけの場合ならば,共有メモリのコピー範囲･比較範囲を修正し，拡張データを他のタスクにも渡すようにすれば良い．しかし，データの利用内容が変更になる場合には，なんらかの工夫が必要である．

(4) 障害発生時への応用

本方式は，新プログラムの単独実行時に障害が発生したときに，システムの動作を継続させながら,新プログラムの問題箇所を特定することにも利用することも考える．障害発生時に単独実行モードから共存実行モードに切り替えることにより，安定した旧プログラムの実行結果を採用しながら，サブシステムを動かすことができる．同時に新プログラムも実行して実行結果の比較を行うため，問題箇所の特定に繋げることができる．

6. まとめ

本研究では，大規模情報制御システムの段階マイグレーションにおいて，新旧モジュールが共存するときでも影響範囲が極小化するように，モジュールを抽出する方式を考案した．ロジックとデータアクセスの分離と，制御対象によるロジックの分離に着目して，データアクセスの共通関数を分離し，ロジックの共通関数を独立化してモジュールを抽出する．また，マイグレーション前後でのプログラムの機能担保を確認するために，新旧モジュールが共存するプログラムを新プログラムと捉え，機能単位であるタスク単位で，現行プログラムと共存実行することで，動作等価性を検証する方式を考案した．さらに，鉄道運行管理システムへの適用試作を行い，本方式の有効性を確認した．

参考文献

1) D.L. Parnas: On the criteria to be used in decomposing sys-tems into modules, Communications of the ACM, vol.15, no.12, pp. 1053-1058, (1972).
2) Robert W. Schwanke: An intelligent tool for re-engineering software modularity. In Proc. 13th Intl. Conf. Software Engineering, (1991).
3) 肥後芳樹，植田泰士，神谷年洋，楠本真二，井上克郎: コードクローン解析に基づくリファクタリングの試み，情報処理学会論文誌，情報処理学会論文誌 45(5), 1357-1366, (2004).
4) 肥後芳樹，神谷年洋，楠本真二，井上克郎: コードクローンを対象としたリファクタリング支援環境，電子情報通信学会論文誌. D-I，情報・システム, I-情報処理 J88-D-I(2), 186-195, (2005).
5) Judith D. Ahrens, Noah S. Prywes: Transition to a Legacy- and Reuse- Based Software Life Cycle, IEEE Computer, Vol.28, No.10, pp. 27-36 (1995).
6) Kyo Chul Kang, Moonzoo Kim, Jaejoon Lee, and Byungkil Kim: Feature-oriented Re-engineering of Legacy Systems into Product Line Assets – a Case Study, Lecture Notes in Computer Science, Volume 3714 (2005).
7) 張暁晶，星野隆: 設計モデルを用いたテスト項目抽出とテストデータ生成手法，電子情報通信学会技術研究報告. KBSE，知能ソフトウェア工学 109(41), 37-42, (2009).