Vol.2012-SE-176 No.1
Vol.2012-EMB-25 No.1
2012/5/21

制御ソフトウェアのモデルベース開発への移行方法

山口鉄平[†]　新田泰広[††]　稲葉雅美[††]　秋山義幸[††]
杉山達也[††]　川上真澄[†]　島袋潤[†]　小川秀人[†]

制御機器のソフトウェア開発では，制御パラメータの調整工数削減による開発工数削減を目的に，モデルベース開発の適用がすすんでいる．制御式を設計する制御設計者とソフトウェアを開発するソフト開発者の違いや，制御式が書かれた制御仕様書と機器の動作実現方法が書かれたソースコードの違いにより，従来の開発方法からモデルベース開発への移行は困難となっている．本報告では，モデルベース開発時に，制御設計者によるモデルの修正とシミュレーションが可能な新たなモデルベース開発への移行方法を提案する．提案方法は，3 パターンでのソースコードのリファクタリングと，モデルとソースコードのデータの受け渡しおよび処理の呼び出しを定義するという特徴を持つ．

Introduction Method of Model-Based Development for Control System Software

TEPPEI YAMAGUCHI[†]　YASUHIRO NITTA[††]
MASAMI INABA[††]　YOSHIYUKI AKIYAMA[††]
TATSUYA SUGIYAMA[††]　MASUMI KAWAKAMI[†]
JUN SHIMABUKURO[†] AND HIDETO OGAWA[†]

There is a need to use a model-based development for reducing man-hours to adjust control parameter in a software development. It is an issue shift from code-based development to model-based development. In this report, we propose an introduction method of model-based development using separation and reconfiguration of source code. In this method, we classify differences between structure of control specifications and structure of source code in 3 patterns. We show the methods of the separation of the source code in each pattern. We also show easy reconfiguration method of the source code.

1. はじめに

制御機器のソフトウェア（以下，制御ソフトと呼ぶ）開発は, 実機を用いた制御パラメータ調整が多く, 一般的に, 制御パラメータの調整のための再設計とソフト開発の工数が多く発生することがある．開発工数削減のために，制御ソフト開発では，モデルベース開発 (Model Based Development: 以下，MBD と呼ぶ)の適用がすすんでいる[1]. MBD により，制御式やパラメータを設計する制御設計の段階で，シミュレータを用いた調整を実現し，制御パラメータの調整工数の削減をめざしている[2].

しかし，制御ソフト開発には，開発者の観点の違いや開発物の表現の違いがあり，制御式やパラメータが書かれた制御仕様書をもとにソースコードを作成する従来の開発方法から MBD への移行は容易ではない．本報告では，制御ソフト開発の特徴を述べ，MBD への移行における課題を述べ，MBD への新たな移行方法を提案する．

2. 従来の移行方法の課題

制御ソフト開発への MBD 適用は，制御パラメータの調整工数削減による開発工数削減をめざしている．従来の制御ソフト開発では，実機を用いて制御パラメータ調整をおこなっている（図 1）．そのため，制御パラメータを変更する再設計の度に，ソフト開発の工数を必要とした．それに対して，MBD では，制御設計の段階で，シミュレーションを用いたパラメータ調整をおこなう（図 2）．そのため，従来再設計の度におこなっていたソフト開発の工数を削減することができる．すなわち，MBD 移行後には次の 2 要件を実現しなければ適用の目的が達せられない．

【要件 1】制御設計段階でのシミュレーション
【要件 2】制御設計者によるモデルの修正

現在，制御仕様書からソースコードを作成する従来のソフト開発から MBD へ移行するための方法には，

【移行方法 1】制御仕様書のモデリング
【移行方法 2】ソースコードのモデリング

という 2 つの方法がある[3].

制御仕様書に書かれている情報だけでは，制御機器を動作させる情報が足りない．そのため，【移行方法 1】で移行する際には，部分的なモデリングやシミュレーションは可能となるが, 制御機器のシミュレーションは困難になる（表 1）.

ソースコードには，制御仕様書に書かれている情報以外に，機器の動作を実現するための情報が書かれている．そのため，【移行方法 2】で移行する際には，制御に関する情

[†] (株)日立製作所 横浜研究所
Hitachi, Ltd. Yokohama Research Laboratory
[††] 日立アプライアンス(株) 清水空調本部
Hitachi Appliances, Inc. Shimizu Works

報とソースコードのみに存在する情報が混じったモデルとなり，モデルの修正が困難になる（表 1）．

これらのことから，【要件 1】を満たしつつ，【要件 2】を満たすことは，【移行方法 1】や【移行方法 2】では困難である．

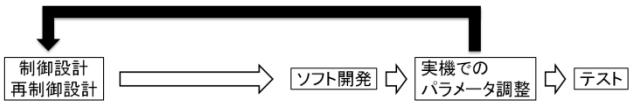

図 1　実機を用いたパラメータ調整

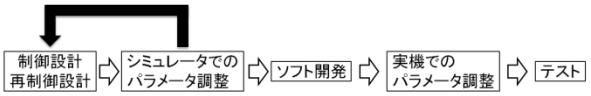

図 2　シミュレータを用いたパラメータ調整

表 1　移行方法の比較

移行方法 / 要件	【移行方法 1】	【移行方法 2】
【要件 1】	困難	可能
【要件 2】	可能	困難

3. 提案方法

3.1 制御ソフト開発の特徴

制御ソフト開発では専門性の違いから，制御式の設計をおこなう制御設計者とソフトウェアを開発するソフト開発者が異なることが多い．制御設計者は，機器の動作環境や動作に観点を置き，機器を動作環境や，動作としての制御式，制御定数，制御式や制御定数を適用する際の条件などに興味を持つ．ソフト開発者は，機器の動作の実現方法に観点を置き,機器の電源投入から終了までの動作の流れや，データ処理や通信処理，タスクの管理などに興味を持つ．すなわち，それぞれの担当者の興味には表 2 のような差がある．

表 2　担当者ごとの興味

	制御設計者	ソフト開発者
観点	機器の動作環境，動作	機器の動作の実現方法
興味	動作環境，制御式，定数，条件表（設定値切替，指令，値変化）	動作の流れ，データ演算，通信，タスク管理

制御ソフトの動作は，メモリの特定箇所に書き込まれたセンサ値の読み込みや内部状態を入力として，制御設計で設計した制御式や定数などで複数のアクチュエータの制御値を算出し，メモリの特定箇所への保存や通信によってアクチュエータへ出力することで実現される（図 3）．すなわち，制御機器の動作は，制御設計で設計される制御式や定数などに，メモリや通信を介した制御値の読み書きといったソースコードのみで定義されるものを加えることで実現する．そして，この制御仕様書の定義とソースコードのみの定義はきれいに分かれるわけではなく，混じった形でソースコードとして定義される（図 4）．

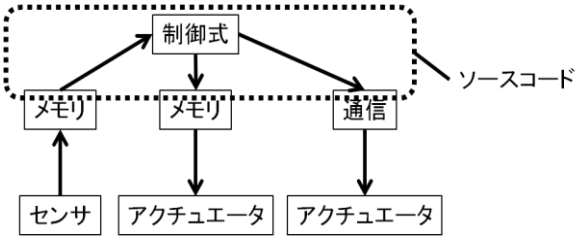

図 3　制御機器の入出力の流れ

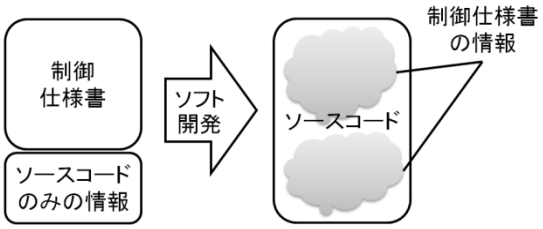

図 4　情報が混じったソースコード

3.2 提案方法

先に挙げた課題を改善した MBD への移行方法として，興味によるソースコードの分離と，モデルとソースコードのデータの受け渡しおよび処理の呼び出しの定義をおこなう新たな移行方法を提案する．提案方法は下記の 3 ステップ（図 5）で MBD へ移行する．

Step1. 興味によるソースコードの分離
Step2. モデルとソースコードの連係定義
Step3. 制御仕様書とソースコードからのモデリング

興味によるソースコードの分離後のモデリングにより，制御に関する情報とソースコードのみに存在する情報が混じったモデルとなることを防げる．そこで，MBD への移行の Step1 として，制御設計者の興味と一致する部分とソフト開発者の興味と一致する部分に整理，分離する．

しかし，ソースコードの分離後，制御仕様書と一致するソースコードだけから作成したモデルだけでは,【移行方法 1】と同じであり，シミュレーションが困難となる．シミュレーションの際には，ソースコードのみの定義も動作させる．そのため，モデルとソースコードのみの定義の連係定義が必要となる．MBD への移行の Step2 として，モデルとソースコードのみの定義に関係するデータをリストアップし，それらデータの代入処理の作成という連係定義をおこなう．

Step1, Step2 をおこなった後に，Step3 として，制御式や定数などモデルの基本的なことは制御設計書に基づきモデリングする．加えて，モデルとソースコードの連係に関してはソースコードに基づきモデリングする．

Step1 および Step2 が提案手法の特徴である．そこで本稿では，Step1 および Step2 の詳細について，次節以降で述べる．

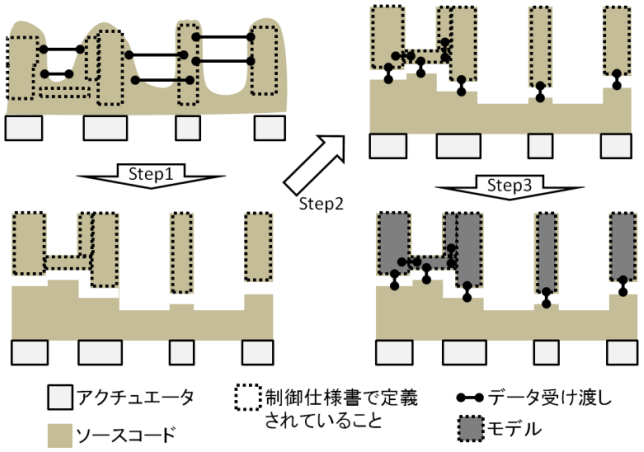

図 5　提案方法による MBD への移行方法

4. 興味によるソースコードの分離

本章では，MBD への移行の Step1 としておこなう，興味によるソースコードの整理, 分離について詳細に説明する．制御仕様書とソースコードの違いについて述べ，ソースコードを制御設計者の興味の部分とソフト開発者のみの興味の部分に整理，分離する 3 パターンについて述べる．

4.1 制御仕様書とソースコードの違いの分類

1.1 節で述べたように，機器の動作は，制御仕様書の情報と，ソースコードのみに含まれる情報を足し合わせたソースコードによって実現される．そのため，機器を動作させるソースコードには，制御仕様書で定義されていない情報も含む．

また，制御設計者とソフト開発者はそれぞれの興味に基づき，制御仕様書とソースコードを作成する．制御仕様書やソースコードを作成する際には，それぞれの観点や興味に基づき，極力小さな労力で作成する．そのため，制御設計者の観点としては異なることであり，別々に定義したことでも，ソフト開発者の観点では同一のことであり， 1 つに定義することがある．このようなことがあり，制御仕様書とソースコードの構造は異なる．

シミュレーションのために，ソースコード全体を利用しつつも，モデルの修正しやすさを実現するために制御仕様書のモデリングが望まれる．そこで，ソースコードの構造を制御仕様書の構造へリファクタリングし，ソースコード全体の情報を失うことなく制御仕様書で定義することを明確にする（図 5）．

複数製品の制御仕様書とソースコードを比較した結果，制御仕様書とソースコードが異なる状態として，次の 3 状態があった．

1.類似データ演算がまとめられている
2.アクチュエータと条件の優先順位が異なる
3.制御値設定処理が追加されている

次節以降で，各状態の詳細とどのようにリファクタリングするかを，3 パターンで述べる．

4.2 類似データ演算の分離パターン

本節では，類似データ演算がまとめられている状態の詳細と，どのように分離するかを述べる．

この状態は，制御仕様書では複数の箇所で定義されていることが，似た処理やデータを利用するため，ソースコードではまとめて定義されている状態である．先に述べたとおり，制御設計者の興味から見ると，制御やデータの定義をおこなっていき，結果的に複数箇所で似た定義となった場合である．また，ソフト開発者の興味から見ると，似たデータ演算が複数箇所にあり，極力小さな労力でソースコードを作成するために，類似データ演算をまとめて 1 カ所で定義した場合である．

類似データ演算がまとめられている例として図 6 の中央から左部分を示す．制御仕様書では，アクチュエータ A のモード 1 の制御をひとまとまりと考え，定義する．モード 1 では制御 a, 制御 b, 制御 c を定義している．また，アクチュエータ A のモード 2 の制御として，制御 a’, 制御 b’ を定義している．この制御 a と制御 a’は似た制御であるものの，モードごとの制御を設計した結果，似た制御となったものである．そのため，制御設計者としては制御 a と制御 a’を 1 つにすることは意識しない．それに対して，ソフト開発者の興味はデータ演算であり，極力小さな労力でソースコードを作成することを目指すため，重複を減らすために制御 a と制御 a’を 1 つにすることを意識する．そのため，ソースコードでは，制御 a と制御 a’のデータ演算を 1 つにまとめ，モードによる条件分けを部分的におこない，演算するよう定義する．このように，制御仕様書とソースコードの構造は異なる．

この状態のリファクタリング方法は，共通部分を複製したのちに，制御ごとにデータ演算の分離をおこなう（図 6 の中央から右部分）．複製する部分は制御仕様書を参照し，ソースコードが制御仕様書のひとまとまりと一致するように選択・複製する．複製後の分離により，制御仕様書で定義する情報とソースコードのみで定義する情報が分離され，ソースコードからモデリングしたモデルであったとしても，制御設計者によって不要な情報が含まれていないモデルとなる．そのため，そのモデルの修正は制御設計者がおこなえる．

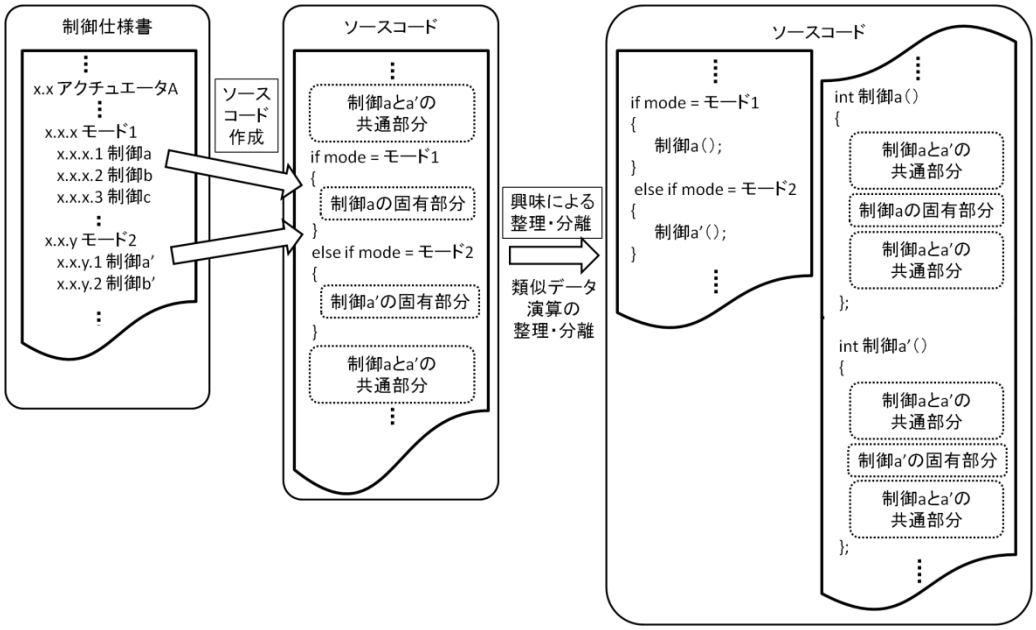

図 6　類似データ演算がまとめられている状態とリファクタリング例

4.3 アクチュエータと条件の優先順位の入れ換えパターン

本節では，アクチュエータと条件の優先順位が異なる状態の詳細と，どのように優先順位を入れ換えるか述べる．

この状態は，制御設計者とソフト開発者の観点が異なり，興味のまとまりが異なるため，制御仕様書とソースコードの構造が異なる状態である．例えば，制御仕様書の最も大きなまとまりはアクチュエータごとだが，ソースコードでは動作状態ごとである．このように，制御設計者とソフト開発者がそれぞれの考えで制御仕様書とソースコードを定義した結果，アクチュエータと条件の優先順位が異なる場合が発生する（図 7 の中央から左部分）．

この状態のリファクタリング方法は，アクチュエータごとにタスクを作成し，アクチュエータごとでどの状態であるか条件判定をおこなう（図 7 の中央から右部分）．アクチュエータごとの条件判定は複製後，自アクチュエータ以外に関する条件判定を削除する．制御設計者の興味にタスク管理は存在しない．また，タスクごとの条件判定はアクチュエータが持つモードや条件表であり，動作は制御として定義される．そのため，タスク管理とタスクごとの条件判定と動作を分離することにより，制御仕様書とモデルの一致性を向上させることができる．

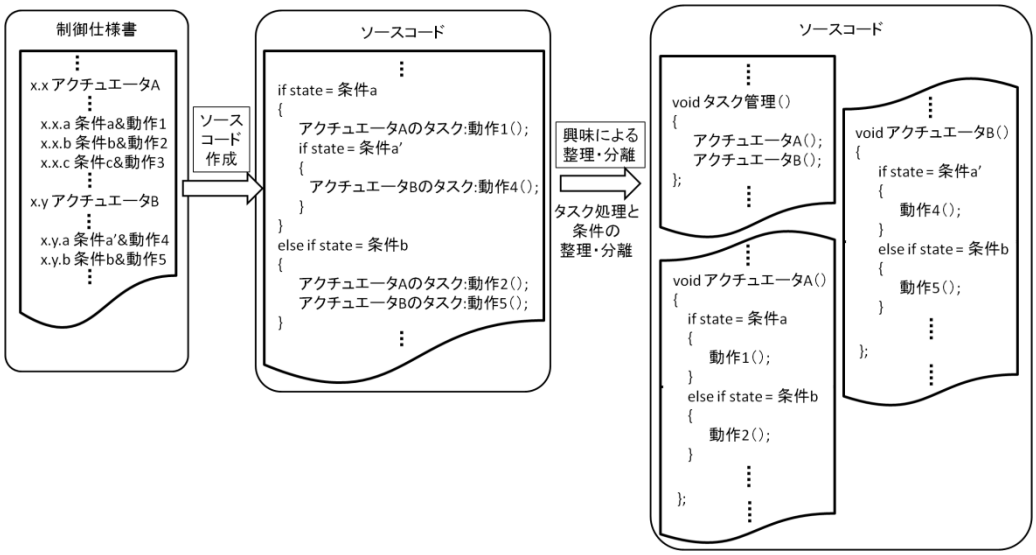

図 7　アクチュエータと条件の優先順位が異なる状態とリファクタリング例

4.4 制御値設定処理の分離パターン

本節では，制御値設定処理が追加されている状態の詳細と，追加されている処理をどのように分離するか述べる．

制御機器を動作させる際には，制御値を算出するだけではなく通信などによって制御値を設定しなければ動作させられないことがある．この状態は，制御仕様書では制御式の定義であるが，ソースコードでは制御値の算出だけではなく通信などの値設定処理をおこなう状態である（図 8 の中央から左部分）．制御設計者の興味はあくまで制御のための式や定数であり，制御値をどのように反映させるかに興味はない．しかし，ソフト開発者の興味は制御値の算出だけではなく，算出した制御値をアクチュエータに設定し，アクチュエータを動かすところにある．そのため，制御仕様書とソースコードの構造が異なる．

このパターンのリファクタリング方法は，制御値設定処理を別タスクもしくは別の処理とする（図 8 の中央から右部分）．制御値の算出と制御値設定処理を分離することで，制御式の算出部分は制御仕様書と一致する．また，制御値設定処理はモデリングしないもののシミュレーション時には必要な部分として，ソースコードとして残すことができる．

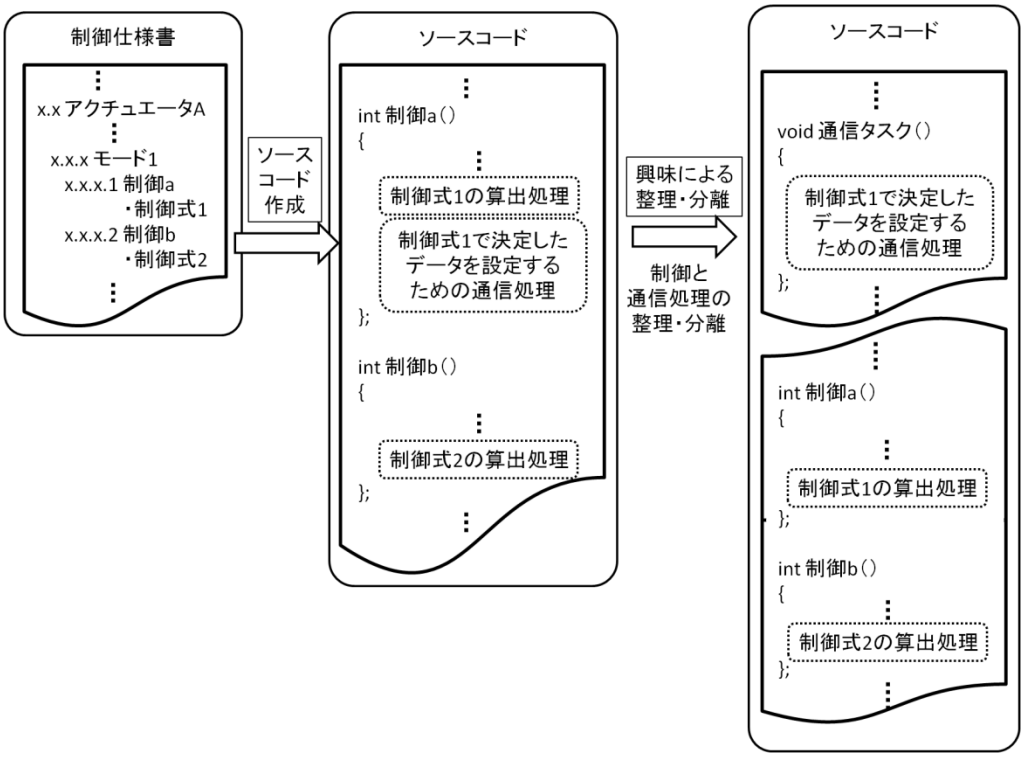

図 8　制御値設定処理が追加されている状態とリファクタリング例

5. モデルとソースコードの連係定義

本章では，MBD への移行の Step2 としておこなう，モデルとソースコードの連係定義について詳細に説明する．モデルとソースコードの連係定義としておこなうことを述べ，その際におこなうモデルの入出力定義の容易化方法について述べる．

5.1 モデルとソースコードの連係定義

ソースコードの整理や分離をおこない，制御仕様書と一致するソースコードだけから作成したモデルだけでは，制御仕様書からモデルを作成する場合と同じであり，シミュレーションが困難となる．シミュレーションの際には，モデルだけでなく，モデルで表現しないソースコード，すなわちソースコードのみの定義も動作させる．そのため，モデルとソースコードのみの定義を連係させる必要がある．

モデルとソースコードのみの定義を連係させる方法として，制御仕様書の情報を表現したソースコードの定義とソースコードのみの定義間でおこなわれることを明確にし，それをモデルとソースコード間で実現する．制御仕様書の情報を表現したソースコードの定義とソースコードのみの定義間でおこなわれていることは，処理の呼び出しや入出力定義を利用したデータの受け渡し処理がある．モデルとソースコードのみの定義間でおこなうことは，制御仕様書の情報を表現したソースコードの定義とソースコードのみの定義間でおこなっていることと同様である．すなわち，モデルとソースコードの連係定義として，制御仕様書の情報を表現したソースコードの定義とソースコードのみの定義間の関数呼び出しや変数の読み書き情報を明確にする．

そして，明確にした情報を用いて，モデルの関係性の定義やモデルの入出力の定義をおこなう．

5.2 モデルの入出力定義の容易化方法

モデルとソースコードの連係定義として，情報を利用してモデルの入出力定義をおこなう．これらの作業をすべて手動でおこなうと手間がかかる．ソースコードの複雑さにもよるが，我々の試行では 0.5[人時]程度要した．我々の試行は，モデリングする 3 関数とソースコードのみの定義 2 関数のデータ利用状態を明確にし，モデルの入出力定義を作成した. その際のデータ利用状態は 10 変数程度が関数間でまたがる形で利用されていた．調査範囲を広げるとその分指数的に調査時間を要し，制御機器のソースコード全体となると膨大な手間になる．加えて，多くの変数が関数間でまたがる形で利用されている場合には，モデルの入出力の定義回数も膨大になる．

そこで，モデルの入出力定義の容易化をおこなう．容易化方法は下記の 3 ステップである．なお，CRUD 表とは，各関数において各変数がどのように利用されているかを表現する表（図 9）であり[4]，ソースコード解析ツールによって自動生成が可能になっている．

Step1. ソースコード解析ツールにより CRUD 表を自動生成する．

Step2. CRUD 表でモデリングする部分としない部分を選択する．

Step3. CRUD 表の情報と選択した情報からモデルの入出力定義を自動生成する．

CRUD 表において，図 9 の変数 3 のように，ソースコード内のモデルで表現する部分とモデルで表現しない部分の両方で利用されている変数は，モデルとソースコード間を連結させる変数である．特にモデリングする部分で参照もしくは削除として利用されている変数はモデルの入力変数となり，モデリングする部分で作成もしくは更新をおこなっている変数はモデルの出力変数となる．図 9 の変数 3 はモデルの入力と出力両方となる．このように，CRUD 表におけるモデルで表現する部分とモデルで表現しない部分を選択することで，CRUD 表の参照や更新の情報からモデルの入出力定義を自動的に生成することができる．

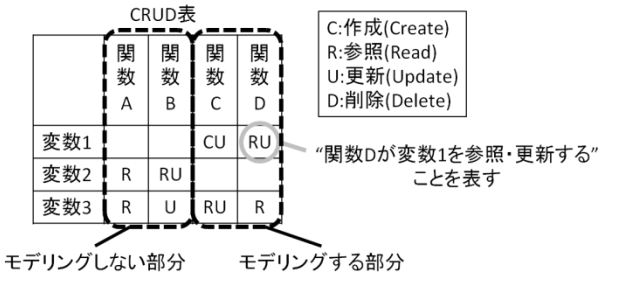

図 9　CRUD 表の例

容易化方法を利用することにより，先ほどの 0.5[人時]かかったモデルの入出力定義の手間が，0.1[人時]程度となった．おこなった作業は，ツールを利用し CRUD 表を作成し，表でモデリングする部分を選択することであった．

制御仕様書とソースコードの構造に違いがあり，一般にソースコードにおけるモデリングする部分は不明確なことが多い．しかし，制御仕様書とソースコードの構造の違いを整理・分離することにより，モデリングする部分は明確になる．モデルの入出力定義をおこなう前に，3 章で示したソースコードの整理や分離をおこなうことによって，表におけるモデリングする部分の選択は容易となる．

6. 結論と今後の予定

本報告では，制御ソフト開発の MBD への新たな移行方法として，興味によるソースコードの分離と，モデルとソースコードの連係定義による MBD への移行方法を提案した. 提案方法は製品開発への試行をおこなっているものの，未だ評価にはいたっていない．今後，評価をおこなう予定である．また，興味によるソースコードの整理，分離パターンは本報告であげた 3 パターン以外にも存在が予想される．そこで，制御仕様書とソースコードが異なる状態やその際のリファクタリングについて今後も調査をおこなう．

参考文献

1) 渡辺政彦: 組み込みソフトウェア向け開発支援環境, 情報処理学会誌, Vol45, No.1, pp.10-15., 2004.
2) 日経 BP 社: モデルに基づく開発方法論のすべて，日経 BP 社, 2006.
3) SESSAME WG2: 組み込みソフトウェア開発のためのリバースモデリング，翔泳社, 2007.
4) 加藤正恭,小川秀人: CRUD マトリクスを用いたソフトウェア設計影響範囲分析手法，情報処理学会第 73 回全国大会講演論文集, No.1, pp.249-251., 2011.