例題拡張による実時間制御システムの開発

島川博光[†] 井戸譲治[†]
高田秀志[†] 堀池聡[†]

本論文では，時間制約が充足されるかどうかを判定できるシステムを実時間計算のための知識がなくとも開発・修正できる例題拡張法を提案し，それを制御システム開発へ適用した結果を示す．本手法では，実時間計算を専門とするエンジニアが開発した，時間制約を充たす小さな例題が，制御に詳しいエンジニアが指定した対象問題での要求に応じて拡張される．例題は形式的ルールと支援ツールにより拡張されるので，拡張時に時間制約充足の判定可能性が損なわれることはない．本手法により両エンジニアは自らの専門の作業に集中できる．また例題は小さいので，その開発と修正は容易である．本手法を鉄鋼圧延システムに適用した結果，従来方式による開発の2.7倍の効率化が達成できた．

Development of Real-time Control Systems Using Example Expansion

HIROMITSU SHIMAKAWA,[†] GEORGE IDO,[†] HIDEYUKI TAKADA[†] and SATOSHI HORIIKE[†]

Control systems would be modified to improve the stability. This paper proposes the example expansion which allows real-time control systems to be modified without knowledge on real-time computing. It expands a small example to a large scale real-time control system. This paper explains formal rules for the expansion which can maintain the timing consistency. The example expansion is evaluated through an actual system development.

1. はじめに

実時間制御システムでは，データ処理があらかじめ決められた締切りまでに終了することを意味する時間一貫性が維持されなければならない．時間一貫性が維持されるかを判定するための実用的手法として Rate Monotonic Analysis[1]（以後，RMA と略す）が提案されている．しかし，RMA を適用するには，システム内の処理間の干渉を最小限に抑え，システム実装や実行ハードウェアによって変化する各タスクの実行時間を正確に知らなければならない．つまり，制御システムの構築には，対象固有の制御技術だけでなく，RMA 特有のプログラミング技術が必要である．よって，実時間計算のためのプログラミング技術を持った実時間システムエンジニアが，対象ごとの制御技術に詳しい制御エンジニアの要求に従い，対象固有のデータ処理を実現するという形で個々の制御システムが開発されてきた．

† 三菱電機産業システム研究所
Industrial Electronics and Systems Lab., Mitsubishi Electric Corporation

現場においては，制御対象の安定性を増すために，スキーマの追加・変更やデータ獲得・提供のためのシステム要素の再構成を制御エンジニアはよく必要とする．しかし，膨大なプログラム・サイズを持つ実システムを時間制約を守ったうえで改修することは容易でない．実時間制御システムの再構築のために部品統合のアプローチ[2]が提案されているが，多数の部品から時間一貫性を維持できるような組合せを選択するには高度な実時間計算に関する知識が要求される．実時間計算技法に不慣れな制御エンジニアにはこの手法は使いにくい．また，部品は汎用性を増すために多くの冗長部分を含み，得られたシステム内の各タスクの正確な実行時間を知ることが困難である．

制御システム構築のために我々は時間一貫性が維持されるかどうかを判定可能とする計算モデルとして ActiveRing モデルを提案した[3)~5)]．本論文では，制御エンジニアが指定したスキーマやシステム構成に関する要求にあわせ，ActiveRing モデルに基づく対象システムを生成する形式的な手法を提案する．我々の手法では，まず，時間制約を充たす，対象問題で使われるあらゆるデータ型を含んだ最小スキーマに基づく

最小構成の例題システムを実時間システムエンジニアが作成する．さらに実時間システムエンジニアはこの例題システムを実装するソースコードを拡張する方法を指定する．この方法に基づき，例題システムを制御エンジニアが指定した対象問題での要求にあわせ拡張することで，時間制約の充足性を判定可能な大規模システムを短期間で開発する．本論文で提案する手法を我々は例題拡張と呼んでいる．この手法は

- 例題内の拡張される手続きを一貫性を維持する機構から切り離す，
- 拡張される手続き内の構造を維持する，
- 拡張される手続き内の文の依存関係を維持する，

という特長を持ち，実時間計算に関する知識を持たない制御エンジニアによる再構築時に時間制約充足の判定可能性が損なわれることを防ぐ．実時間システムエンジニアによる，時間制約の充足に関する開発と検証は小さな例題に対して行うので，その期間は短く，また，修正も容易である．例題拡張を用いて，我々は実際の制御システムを開発した．その結果，従来方式の2.7倍ほどの開発効率化ができた．

以下，2章において基礎となる計算モデルを示す．3章ではシステム開発の形式的手法を定義する．4章で本手法を実際のシステム開発で評価し，関連研究との比較を行ったうえで，最後にまとめを行う．

2. 実時間制御システムのためのモデル

2.1 優先度継承とRMA

優先度継承法[6]は，共有情報の排他制御による閉塞時間を既知とする手法である．いま，2つのタスク τ_l,τ_h の優先度を $p_l, p_h\,(p_l < p_h)$ とする．τ_l が共有情報を排他的に読み書きしているときに τ_h がその情報を読み書きしようとした場合，優先度継承法では τ_l の優先度を p_h に一時的に格上げし，両者の中位の優先度を持ち，この情報を必要としない第3のタスクに τ_l の実行が横取りされることを防ぐ．これにより τ_h を閉塞するタスクで優先度が p_h よりも低いものは τ_l だけとなり，τ_l が共有情報の読み書きにかかる時間が排他制御による τ_h の閉塞時間となる．

RMA[1]は，任意の時点で横取り可能な周期タスク集合に対し，その各要素を周期内に実行できるかを判定する手法である．情報を共有するタスク集合のためにRMAと優先度継承法の併用が考えられている．いま，T_i，C_i をそれぞれタスク τ_i の周期，単独実行時の1周期分の処理時間とする．状況により実行パスが替わるタスクでは最大時間を要するパスの処理時間を C_i とする．周期の短いタスクから順に高い固定優先度を与えれば，以下の条件を満たす n 個の周期タスクの集合は周期内に実行されることが証明されている[6]．

$$\frac{C_1}{T_1}+\ldots+\frac{C_n}{T_n}+max(\frac{B_1}{T_1},\ldots,\frac{B_{n-1}}{T_{n-1}}) \leq n(2^{\frac{1}{n}}-1)$$

ここで，B_i は τ_i が自分より優先度の低いタスクに閉塞される最大時間である．各タスクの実行と閉塞の最大時間が分かれば，制約時間内のタスク実行を保証できるかどうかを上式で判定可能である．

RMAでは，周期を持たないタスクには最低優先度が与えられ，バックグラウンドで処理される．また，非周期実時間タスクに対しては，その締切りを周期と見なすことにより，周期タスクとともに締切り内の終了を保証できる．このような保証は，締切りを持つタスクに使われるCPU能力の割合が低いので悲観的[1]だが，残されたCPU能力は最低優先度のタスク実行に使える．この手法は簡便性，周期性，絶対的保証の点で連続制御に適しており実用的である．

2.2 要　　求

制御システムが扱うスキーマ内の変数であるデータ項目には，その値の源に関して，値がセンサから得られる**基礎データ項目**と値が既獲得のデータ項目値から計算される**導出データ項目**の別がある[7]．

監視対象や制御対象からのデータの獲得とそれらへの制御コマンド送出はあらかじめ決められた時間制約内に終了しなければならない．時間制約を持つ処理がそれを守って終了するさい**時間一貫性**が維持されているという．処理が生み出す価値を考え，時間制約を示す締切り[8]は，それまでに終了できないと価値がその時点で0となる**厳密な締切り**と，それを過ぎると時間経過とともに価値が減少しやがて0となる**緩やかな締切り**に区別される．

制御システムには，時間一貫性を維持した，

- 周期的・非周期的時制データの獲得，
- 最新時制データの計算と提供，
- 状態変化時の時制データの計算と提供

が要求される．とくに状態変化時の処理には厳密な締切りを持った緊急処理が含まれる．さらに，制御システムは，実時間計算のプログラミング技術を持たない制御エンジニアにより再構築できることが望まれる．

2.3 エリアとオブジェクト

前述の要求を満たす制御システムの構築のために，我々はエリアとオブジェクトからなるActiveRingモデル[5]を提案した．ActiveRingモデルは，入力エリア，循環エリア，出力エリアを持つ（**図1**参照）．入力エリアには対象の状態を示すデータが発生後十分速やかに書き込まれるものとし，ここから獲得されたデー

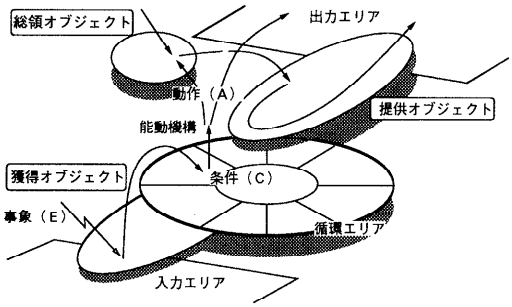


図1 ActiveRing モデル
Fig. 1 ActiveRing model.

タが獲得時刻とともに循環エリア上で時制データとして保持される．出力エリアには制御のために循環エリア上のデータから計算されたデータが書き込まれる．

ActiveRing モデルでは，1つの獲得スキーマに基づいて1つの獲得オブジェクトと1つの循環エリアが組をなす．この組は，異なる周期ごとに，そして，非周期獲得の種別ごとに用意される．獲得スキーマは基礎データ項目と導出データ項目からなる．各獲得オブジェクトはタイマからの周期的通知や非周期データの到着を事象と見なし，獲得機構と能動機構を順に起動する．獲得機構は，入力エリアからデータを基礎データ項目値として獲得し，これらから導出データ項目値を計算する．前回と今回の獲得値の差分をとるなど，既獲得のデータより今回の獲得値を導出してもよい．能動機構は，ECA モデル[9]に基づき，制御対象の状態変化に反応する．能動機構は，獲得のたびに最新データに対する条件を評価するので状態変化を素早く発見できる．条件成立時には出力エリアにデータが書き込まれる．このデータは，能動機構自身が最新データを使って計算する場合と，最新以外のデータを使い計算することを能動機構が総領オブジェクトを介して提供機構に依頼する場合がある．提供オブジェクトは，外部からの検索依頼の要求により指定された期間の時制データの並びを時系列として循環エリアから読み，出力データを計算する．異なる優先度の検索依頼の並行処理のため提供オブジェクトは複数用意できる．総領オブジェクトは1つだけであり，外部から受け取った検索依頼の要求を適切な提供オブジェクトに割り当てる．

2.4 独立性の高いメソッド

獲得オブジェクトと提供オブジェクトは循環エリアを共有するので，データの論理的一貫性を維持するため排他制御が必要である．ActiveRing モデルでは，既獲得の最新時制データの獲得時刻と次に上書きされる

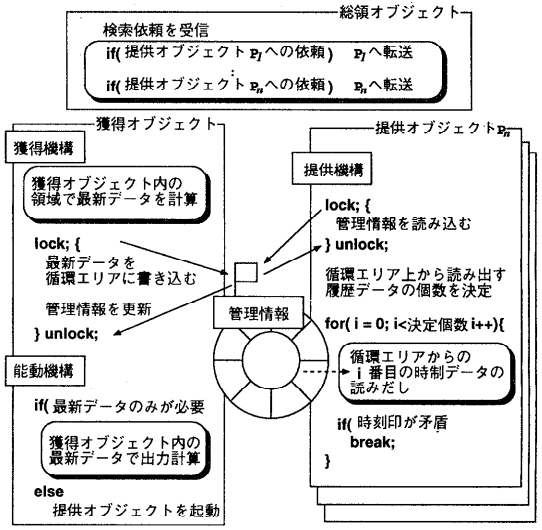


図2 メソッドの概要
Fig. 2 Outline of methods.

循環エリアの位置を示した管理情報が優先度継承法に基づいて排他制御される[3),4)]．RMA による解析が容易となるように，ActiveRing モデルではオブジェクトのメソッドは**図2**に示す概要に則って構築される．

総領オブジェクトは検索依頼要求を処理する提供オブジェクトを決定する．総領オブジェクトは他のオブジェクトに対し排他制御すべき情報を持たない．獲得オブジェクトの獲得機構は，最新時制データを循環エリアに書き込む間，管理情報を施錠する．能動機構の条件評価と最新データを用いた出力データ計算では，獲得オブジェクトが最新データを持っているので排他制御は必要ない．過去の時制データが必要な場合のみ，能動機構は提供オブジェクトに時制データ読み出しを依頼する．一方，提供オブジェクトは，管理情報を読み出す間のみ施錠し，上書きされないであろう部分から時制データを排他制御せずに読み出す．提供オブジェクトは読み出した時制データの獲得時刻から上書きされていないことを確かめ，もし上書きされていれば読み出しを打ち切る．獲得オブジェクトと提供オブジェクトの各メソッドはこの管理情報の排他制御を除いて互いに影響を与えない．

新たなデータの獲得の前に最新値の提供を繰り返しても意味がないので，獲得より頻繁な提供はありえない．RMA では提供オブジェクトより高い優先度が獲得オブジェクトに与えられる．管理情報の排他制御のため提供オブジェクトが獲得オブジェクトを閉塞する時間は短い．さらに優先度継承法により，その最悪値が分かる．また，各メソッドの最悪実行時間は，それ

を単独で実行することにより測定可能である．よって，ActiveRing モデルに基づくシステムに RMA を適用することができる．RMA に基づき優先度を与えるので，厳密な締切りを持つデータ処理は，スケジュール可能と判定されれば必ず締切り内に終了する．緩やかな締切りを持つタスクは，最低の優先度で処理されるが，バックグランド処理に大きな CPU 能力が残されている悲観的な RMA のもとでは処理時間はデータ量に比例することを統計的に保証できる．

ActiveRing モデルでは締切りはデータ鮮度に応じて設定される．最新データを扱う獲得には厳密な締切りが与えられる．最新データから周期的に出力を計算する提供タスクには厳密な締切りが，最新以外のデータを扱う提供タスクには緩やかな締切りが与えられる．状態変化への反応には以下の別がある．

最新データを使うタスク 獲得オブジェクトが評価，出力の計算を厳密な締切りを持つ一連のタスクとして処理する．

最新以外のデータを使うタスク 評価は獲得オブジェクトが行うが，出力の計算は提供オブジェクトが緩やかな締切りのもとで処理する．

前者は緊急性の高い反応を実現し，後者は緊急性のない処理が負荷を高めることを防ぐ．

3. 例題拡張法

3.1 生成系の生成によるシステム開発

制御システムの再構築は簡単でなければならず，再構築後も，時間一貫性が維持されていなければならない．このため，RMA が適用可能な制御システムを容易に開発できる手法を考える．

RMA の適用に必要な処理時間の正確な見積りには，まず上限が得られ，かつ，この上限と実際の処理時間との差ができるだけ小さいことが求められる．実時間制御システムでは各問題特化のデバイスを使ってデータが獲得され，スキーマも問題ごとに変わるので，ソースコードを生成することが有効である．それはソースコードをコンパイルし実機上で実行することで正確な実行時間を測定することができるからである．

我々の手法では，まず，対象問題で使われるあらゆるデータ型を含んだ最小スキーマを扱う最小構成の ActiveRing モデルに基づくシステムを作成し，これを例題とする．この例題のオブジェクト・メソッドのうち対象問題に依存する手続きを要求にあわせて拡張することによって大規模システムを短期間で開発する．本論文ではこの手法を例題拡張とよぶ．

我々は図 **3** に示されるような生成系の生成系を提供する．これは，実時間システムエンジニアが作成した例題システムのソースコードと拡張方法から，対象システムの生成系をつくり出す．つくり出された生成系は，制御エンジニアが示した対象問題での要求ごとに対象システムをつくり出す．これにより，実時間システムエンジニアを煩わさずに制御エンジニアが実時間制御システムを再構築できる．なお本論文では，ソースコードは，多くの実時間システムの開発言語である C 言語で書かれていると仮定する．

3.2 例題システム

本論文では，スキーマと図 1 のオブジェクトの構成が拡張されるものとする．ActiveRing モデルでは，

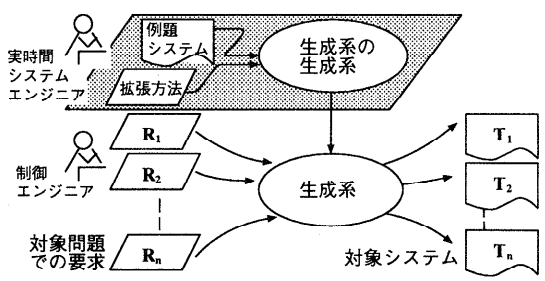


図 3 生成系の生成系
Fig. 3 Generator generator.

```
l1:  void write_RB(sp)
l2:      struct _scene_RBschema *sp;
l3:  {
l4:      char area[INPUT_AREA_SIZE];
l5:      int offset, length;                                   b_hd
                  /* 入力エリアのデータをareaへ */
l6:      read_RB_inputArea(&area, INPUT_AREA_SIZE);            b_init
                                                               b_prl
                  /* 基礎データ項目の計算 */
l7:      offset = 8; /* offset in input area */
l8:      length = 2; /* length of input data */
l9:      conversion(&(sp->alpha), area + offset, length);      b_bsc
l10:     offset = 16;
l11:     length = 1;
l12:     conversion(&(sp->beta), area + offset, length);
                  /* 導出データ項目の計算 */
l13:     maximum(&(sp->max),
l14:             &(sp->alpha),
l15:             &(sp->beta) );                                b_drv
l16:     minimum(&(sp->min), &(sp->alpha), &(sp->beta));       b_avs
                                                               b_bdy
l17: }                                                         b_tl
                                                               b_whl
```

図 4 コードブロックの例
Fig. 4 Example of codeblocks.

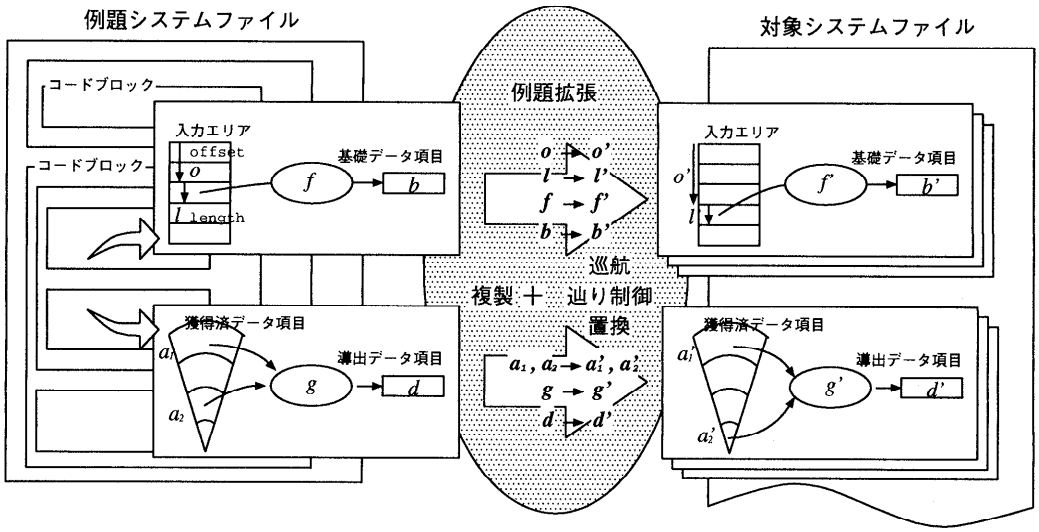


図5 例題拡張
Fig. 5 Example expansion.

図2が示すように，いったん依頼を転送すると総領オブジェクトは提供・獲得処理には影響しない．獲得と提供のメソッドでは，排他制御される管理情報はスキーマにもオブジェクト構成にも依存しない．ActiveRingモデルに基づけば，論理的一貫性を維持する機構は対象問題に依存する手続きから切り離される．問題依存の手続きは図2の影付きの枠で囲まれた部分となる．この問題依存の手続きは，適用問題でのオブジェクト構成とスキーマにあわせて拡張できるように，以下のように記述される．

総領オブジェクトが検索依頼をどの提供オブジェクトに転送すべきかは問題に依存する．オブジェクト構成は総領オブジェクトのメソッドに影響する．このメソッドは図2のように条件文の並びで実装される．並びの拡張により対象問題に対応する．

スキーマの拡張は獲得オブジェクトと提供オブジェクトのメソッドに影響する．スキーマへのデータ設定はその内部の個々のデータ項目に値を設定していくことにより実現される．**図4**に簡単な獲得手続きの例を示す．図の左端は行番号である．この手続きでは，入力領域上のデータが一括して局所変数 **area** に読み出され，そこから基礎データ項目 **alpha**，**beta** の値が設定される．その後，この2変数値のうち大きい方が導出データ項目 **max** に，他方が **min** に設定される．l_7 から l_9 までの行は基礎データ項目を計算する部分で，オフセット **offset** とデータ長 **length** で指定される入力エリア上のデータを使い，関数 **conversion** が **alpha** の値を設定する．一方，l_{13} から l_{17} までの行は基礎データ項目値より導出データ項目 **max** を計算する部分である．例題の基礎データ項目のための獲得メソッドは入力エリア上のデータからの変換法を示し，導出データ項目のための獲得メソッドは既獲得のデータ項目値からの導出法を示すものである．また，獲得スキーマに基づくデータを出力エリアに書くべき提供スキーマに基づくデータに変換する能動機構と提供機構でも，提供スキーマ内の個々のデータ項目への変換法が示される．これらの変換法は，他の変換法との置き換えが可能となるように，局所変数のみを使い入力値から出力値を計算する関数で実装される．これら関数は対象問題ごとに必要なものが用意される．

3.3 拡張の方針

総領オブジェクトのメソッドの拡張は図2のコード

検索依頼を受信

if（提供オブジェクト **P** への依頼）**P** に転送

の2行目の条件文を複製し文字列 **P** を置換することを反復することにより実現できる．

図4の手続きを例としスキーマ拡張の方針を示す．**図5**が示すように，多数のデータ項目を計算するコードを生成するには，基礎データ項目と導出データ項目に応じて例題の該当部分を選択し，その中の文字列を置換することを反復すればよい．基礎データ項目に対しては，オフセットとデータ長，変換関数，基礎データ項目を表す文字列が置換され，導出データ項目に対しては，既獲得のデータ項目，変換関数，導出データ項目を表す文字列が置換される．

拡張の基本方針は，対象問題での要求にあわせて例

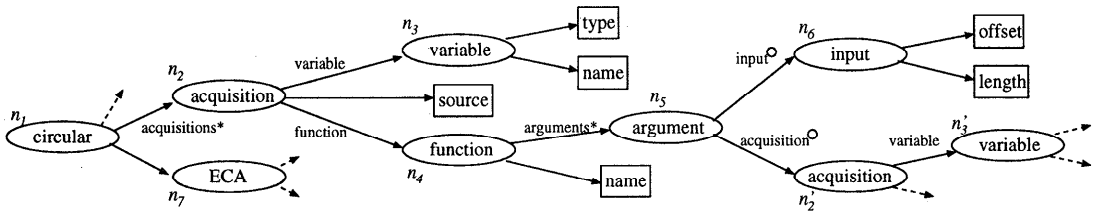


図6 対象システム定義の内部表現
Fig. 6 Internal structure of target system definition.

題の手続きの構成要素を反復的・選択的に使用しその中の文字列を置換するというものである．また，拡張の前後でプログラムの正しさが維持されなければならない．対象システムのソースコードを得るには

- 手続きの構成要素
- 対象問題での要求
- 反復回数，選択条件・置換を示す拡張ルール
- プログラムの正しさを維持するうえでの指標

が必要であり，以下この順に各々を説明する．

3.4 コードブロック

例題システムの手続きは，その構成要素ごとに拡張される．手続きの構成要素として我々は以下のいずれかにあたるコードブロックを考える．

- 分割されずに使用される連続した行の最大の連接
- 0回以上使用される，コードブロックの連接
- その要素のたかだか1つが選択されるコードブロックの連接

メソッドのソースコード中に連続して現れる行 $l_i, ...l_j$ からなるコードブロック b_p を記法 $b_p(l_1, ..., l_m)$ で表現する．記法 $b_p(b_1, ..., b_k)$ は b_p が $b_1, ..., b_k$ の連接であることを表している．

コードブロックには，変更なく使用されるものと文字列置換して使用されるものがある．前者は固定であるといい，後者は可変であるという．他のコードブロックを含まない場合，そのコードブロックは不可分であるという．我々はコード・ブロックを以下に示す機能階層をなす6種類に分類する．

part は rigid cell と adaptable cell の連接である．
　rigid cell は固定・不可分である．
　adaptable cell は繰返し使用可能な cytoplasm と nucleus の連接である．
　　cytoplasm は可変・不可分である．
　　nucleus は gene と part からなり，これらを選択的に使用できる．
　　gene は可変・不可分で，選択的に使用される．
　　part

手続き全体は1つの part と見なされる．part は nucleus に含まれるので，拡張される手続きはコードブロックの入れ子構造で表現される．

あるコードブロックに対し，その構成要素であるコードブロックを順序を維持して複製し，文字列を置換する．ただし，複製にさいしては adaptable cell はその構成要素の連接を反復して使用でき，nucleus はその構成要素のたかだか1つを選択して使用する．この作業を各コードブロックで再帰的に繰り返すことにより，例題を拡張する．よって，例題の手続きにおけるコードブロックの入れ子構造と連接は対象システムでも維持される．

3.5 対象システムの定義

例題拡張では，対象システムにおける，スキーマや図1で示したオブジェクト構成などに関する要求にあわせて，例題手続きが拡張される．制御エンジニアが指定したスキーマ，オブジェクト構成などに関する対象システムでの要求を**対象システム定義**とよぶ．対象システム定義が規定する情報の一部を各オブジェクトごとに以下に示す．

総領 要求とそれを処理する提供オブジェクトの組
獲得 獲得スキーマ内のデータ項目とその値の設定法，ECA ルール
提供 提供スキーマ内のデータ項目とその値の設定法

我々は ActiveRing モデルに基づく制御システムの指定に特化された言語 Servish を開発した．制御エンジニアが指定した対象システム定義はこの言語のパーザにより変換され計算機内で 図6 に示されるような木構造で表現される．図中，長方形で示された節はリテラル値を持つ終端節で，楕円で示された節は1つ以上のメンバからなるデータ構造体で実装される非終端節である．非終端節から別の節に伸びている弧はその弧上に付けられたラベルで示されるメンバ変数で出元の節が行き先の節を参照することを意味する．ラベルに添えられた＊は，出元の節が複数の行き先の節を反復して参照していることを意味し，ラベルに添えられた○は，出元の節が複数ある行き先の節から1つを選

択することを意味する．

3.6 拡張ルール

本節では，必要な記法を準備したうえで，拡張ルールを定義する．

記法 $l_i \Leftarrow n_j$ はソースコード行 l_i の文字列が節 n_j に従って置き換えられることを示しており，1つのコードブロックで実施される**置換** R は $R = \{l_i \Leftarrow n_k, ..., l_j \Leftarrow n_l\}$. により表現される．文字列が置き換えられないとき，置換は ϕ である．

例題手続きのコードブロックの入れ子構造と順序を対象システムに移転するために，例題手続きは辿られながら拡張される．**辿り制御**は深さ優先の辿りで実現できる連接以外の方法でソースコードを辿ることを表現する規則である．辿り制御 T は，記法 $itr(n.m)$, $sel(n.m)$, $case(l)$ のいずれかである．ここで，m は対象システム定義における非終端節 n が参照するメンバである．記法 $itr(n.m)$ は adaptable cell 内のコードブロックが反復的に利用される回数を示す．コードブロックは n が m を参照する回数だけ反復利用される．一方，選択は nucleus とその構成要素である gene と part の組合せによって表現される．nucleus とともに指定された記法 $sel(n.m)$ とその構成要素とともに指定された記法 $case(l)$ の組合せは，節 n のメンバ m の値がリテラル値 l に等しいとき，この構成要素が選ばれることを表す．

反復回数，選択条件，置換すべき文字列は対象システム定義に示されているので，コードブロックと節との対応をとりながら，弧を巡航する必要がある．いま，コードブロック b が節 n_e に対応するとする．さらに節 $n_{r_1}, ..., n_{r_k}$ が b 内での文字列置換もしくは b の構成要素を辿るうえで必要であるとする．$\{n_{r_1}, ..., n_{r_k}\}$ 内の任意の要素 n_{r_m} に対して節 n_e から節 n_{r_m} へ至るパスを巡航とし，巡航を構成する弧の集合 $L(n_{r_m})$ を考える．**巡航集合**とは，これら集合の和

$$\bigcup_{1 \le m \le k} L(n_{r_m})$$

である．記法 $n_d \leftarrow n_s$ は n_s から n_d に至る弧が存在することを示す．巡航集合 E は $E = \{n_r \leftarrow n_i, ..., n_k \leftarrow n_e\}$ として表現される．

いま，構成要素 $b_{c_1}, ..., b_{c_m}$ からなるコードブロック b_p が対象システム定義内の節 n_e に対応しているものとする．さらに $b_{c_1}, ..., b_{c_m}$ はそれぞれ節 $n_1, ..., n_m$ に対応しているものとする．コードブロック b から対象システム定義の節 n への対応に関係付けられた**拡張ルール**とは，以下の条件を満たす巡航集合 E，辿り制御 T，置換 R の3つ組である．

- 巡航集合 E は節 n_e から $\{n_1, ..., n_m\}$ の任意の要素への巡航を構成する弧を含み，
- 巡航集合 E は節 n_e から辿り制御で使われるすべての節への巡航を構成する弧を含み，
- 巡航集合 E は節 n_e から R の任意の要素で使われている節への巡航を構成する弧を含む．

記法

$$b_p(l_i, ...l_j) \xmapsto{\alpha_p} n_e$$

は，ソースコード行 $l_i, ...l_j$ からなるコードブロック b_p が節 n_e に，拡張ルール α_p により対応していることを示している．

3.7 正しさの維持

プログラムの正しさが例題システムから対象システムに移転されるように，我々は制御依存関係とデータ依存関係[10),11)]に着目する．

制御依存関係 文 s が条件文もしくは繰返し文の条件式であり，文 t を実行するかどうかが文 s の評価結果によって決定するとき，文 s から文 t への制御依存関係があるという．

データ依存関係 変数 v が文 n で定義され文 s で参照されるとする．文 n から文 s への実行パスが存在し，かつ，この実行パスの中では変数 v は再定義されないプログラムでは文 n から文 s への変数 v に関するデータ依存関係があるという．

実時間システムエンジニアは，条件文と繰返し文の構成要素を1つのコードブロックの中に含めることにより，これらの文に起因する制御依存関係が正しく移転されるようにする．コードブロック b 内の文 s で定義された変数 v が，コードブロック c 内の文 t で参照される場合を考えよう．文 t は s に依存する．いま，v を u に置換するとする．データ依存関係を維持するために，実時間システムエンジニアは，b と c の順序を守り，s と t の両方で v を u へ置換するようにする．さらに，置換される変数名に誤りや他の置換との重複があってはならない．たとえば，制御エンジニアが対象システム定義を作成するさいに，あるデータ項目名を，獲得の場合と別のデータ項目値の導出に使用される場合とで異なって規定すれば，正しい対象システムは生成できない．このような検査は対象システムの意味に関する知識を必要とするので，本手法では Servish パーザの意味的検査として実施される．

プログラムの論理的動作としての正しさが検証された例題のメソッドを，制御・データ依存関係を維持する方法により拡張することで，正しく動作する対象システムが得られる．この対象システムは例題と同様 ActiveRing モデルに基づいているので，各メソッド

を切り出して単独で実行し最悪実行時間を測定できる．また，拡張された部分はオブジェクト間の排他制御のためのコードから切り離されており例題から変わっていない．したがって，対象システムでも優先度が高いタスクが優先度の低いタスクに閉塞される最悪時間が得られる．よってRMAを用いて対象システムで時間一貫性が維持できるかを判定可能である．

3.8 拡張例

対象システム定義の例としてあげた図6の部分木は，各データ項目値の獲得法を基礎データ項目と導出データ項目の両方の場合で表している．データ項目は複数あるので，循環エリア示す節 n_1 から個々のデータ項目値の獲得を示す節 n_2 への弧は `acquisitios*` のラベルが付けられる．個々の獲得では，関数 n_4 で計算された値が n_3 で示されるデータ項目に割り当てられる．節 n_2 の `source` の値が `BASIC` なら獲得されるデータ項目は基礎データ項目であり，節 n_6 が示す入力エリア上の値が関数 `function` の引数となる．`source` の値が `DERIVED` なら獲得されるデータ項目は導出データ項目であり，節 n'_2，n'_3 が示す既獲得のデータ項目値が引数となる☆．

図4の手続きを図6の対象システム定義にあわせ拡張することを考えよう．指定されたあらゆるデータ項目を計算するコードを生成するために，コードブロック b_{bdy} ではその構成要素 b_{avs} が節 n_1 の `acquisitions*`により表される値だけ繰り返し使用される．これは

$$b_{bdy}(b_{avs}) \overset{\alpha_{bdy}}{\longmapsto} n_1,$$
$$\alpha_{bdy}(E_{bdy}, itr(n_1.acquisitions), \phi),$$
$$E_{bdy} = \{n_2 \leftarrow n_1\}$$

として α_{bdy} の辿り制御で表現される．例題拡張の処理がコードブロック b_{bdy} からその内部の b_{avs} に移るには，b_{bdy} が対応する節 n_1 から b_{avs} が対応する節 n_2 への巡航集合が必要である．このため，α_{bdy} は巡航集合 E_{bdy} を含む．次に，コードブロック b_{avs} では，獲得されるデータ項目が基礎データ項目か導出データ項目かでコードブロック b_{bsc} と b_{drv} のいずれかが選択される．b_{avs} の拡張ルール

$$\alpha_{avs}(\phi, sel(n_2.source), \phi)$$

は，選択のために節 n_2 のメンバ `source` が参照されることを示している．基礎データ項目を計算する b_{bsc} は下記の拡張ルールにより節 n_2 に対応している．

$$\alpha_{bsc}(E_{bsc}, case(BASIC), R_{bsc}),$$
$$E_{bsc} = \{n_3 \leftarrow n_2, n_4 \leftarrow n_2, n_5 \leftarrow n_4, n_6 \leftarrow n_5\},$$
$$R_{bsc} = \{l_7 \Leftarrow n_6, l_8 \Leftarrow n_6, l_9 \Leftarrow n_4, n_3, n_6\}$$

一方，導出データ項目の獲得のための b_{drv} は下記の拡張ルールにより節 n_2 に対応している．

$$\alpha_{drv}(E_{drv}, case(DERIVED), R_{drv}),$$
$$E_{drv} = \{n_3 \leftarrow n_2, n_4 \leftarrow n_2, n_5 \leftarrow n_4,$$
$$n'_2 \leftarrow n_5, n'_3 \leftarrow n'_2\},$$
$$R_{drv} = \{l_{13} \Leftarrow n_4, n_3, l_{14} \Leftarrow n'_3, l_{15} \Leftarrow n'_3\}$$

置換 R_{bsc}，R_{drv} に必要な巡航集合が E_{bsc}，E_{drv} にそれぞれ示される．拡張ルール α_{bsc} と α_{drv} の辿り制御 $case(l)$ は，`source` の値がリテラル値 l に等しいとき，そのコードブロックが選ばれることを示す．

4. 実開発への適用

4.1 開発環境

例題は図7に示す流れに従って拡張される．制御エンジニアが指定した対象システムのスキーマと構成は，意味的検査ののち対象システム定義に変換される．これとは別に対象環境で正しく動作する，ActiveRingモデルに基づいた例題システムを実時間システム・エンジニアは作成しておく．この例題システムのソースコードに拡張ルールがC言語のコメントとして埋め込まれる．このファイルをタグファイルと呼ぶ．

タグファイルから対象システムの手続きを生成するためには，拡張ルールを解釈することが必要である．本手法では，拡張ルールの解釈系は，対象システムの生成系から分離されており，タグファイルから生成系のソースコードを生成する．以後，解釈系をg_gと呼ぶことにする．g_gは生成系の生成系（generator generator）を意味する．g_gは生成系のソースコードと，タグファイルから拡張ルールを取り去ったテンプレートファイルを出力する．生成系は，テンプレートファイルからコードブロックを読み出し，対象システ

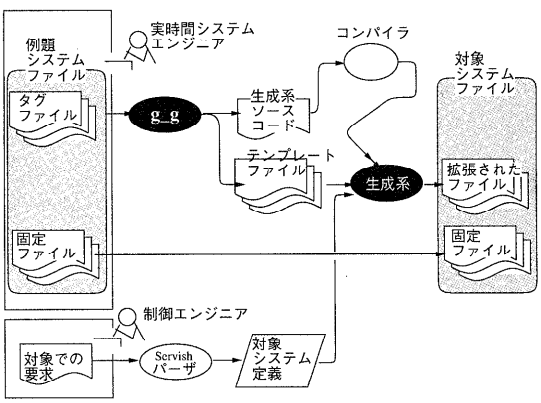


図7 処理の流れ
Fig. 7 Work flow.

☆ n'_2 と n'_3 は，n_2，n_3 と同じデータ型を持つ異なる節である．

表1 対象システムと例題システムの違い
Table 1 Difference of target system from example system.

system	要求							サイズ (line)	
	スキーマ		データ項目		オブジェクト		ECA	変更部	固定部
	獲得	提供	獲得	提供	獲得	提供	ルール		
例題	3	3	12	12	3	7	12	10,483	41,338
対象	7	367	1619	4039	7	2	358	257,828	

ム定義に従って文字列置換を行う.

4.2 実開発における評価

本節では,鉄鋼圧延ラインの実時間制御システムの開発記録をとおして例題拡張の有効性を評価する.

表1 は対象システムと例題システムの違いを示している.例題システムの設計には81人日,試験を含む実装には75人日の工数が必要であった.表より分かるように,例題システムは規模のはるかに大きいシステムに拡張されている.制御エンジニアによる対象システムの要求の指定とServishパーザが指摘した誤りの修正に5人日かかった.対象システムのサイズから見積もった,人手でのスクラッチからの開発工数は433人日である.よって,例題設計・実装からの開発では2.7倍,制御エンジニアの要求指定からの開発では87倍の速さで開発ができた.

新しい機能の追加や性能の向上のための修正においても,小さな例題は効率化に貢献する.1つの例として,例題システムで48行が改修された場合を考える.例題システムの修正と試験,タグファイルの修正に3人日が必要であった.この修正のあと生成された対象システムはもとのシステムと5872行で違っている.行数と工数だけの比較ではあるが,小さな例題により10倍程度の修正の効率化ができた.

5. 関連した研究

この論文で提案する手法の目的は,制御エンジニアが実時間制御システムの構成とスキーマを簡単に修正することができる手法を提供することである.スキーマ定義が可能なデータ獲得システムが多く提案されている[12),13)].しかし,これらのうち時間一貫性を保証したものはない.一般的で強力な実時間システム記述言語[14),15)]は,実時間計算技術に精通していない制御エンジニアを支援するのに適しているとはいえない.

システム全体のタスクをスケジュール可能にするため,文献16)にあげられた手法では各タスクの周期が調整される.また文献11)は,周期処理を中心とするシステムにおいて,時間一貫性の保証が得られなかったプログラムを修正して保証を得る手法を提案している.これらの研究は,システム構成に変更がなく個々の要素の動作周期だけを変更する場合を扱っている.実際の制御では構成やスキーマも変更される.

ROMPP[17)]は実時間データベースにおけるスキーマ進化の問題を性能多様性を用いて解決している.ここでは,時間一貫性を維持しながらスキーマを進化させるためのルールが示されているが,それを支援する形式的手法やツールは提供されていない.

6. おわりに

本論文では,時間一貫性を維持するための制御システムのための計算モデルであるActiveRingモデルに基づいて,小さな例題を対象問題での要求にあわせ拡張する例題拡張法を提案した.例題は実時間計算を専門とする実時間システムエンジニアが作成し,要求は制御を専門とする制御エンジニアが指定する.

時間一貫性を維持するかどうかを判定できる性質を損なわない形式的手法と支援ツールにより例題は実システムに拡張される.実時間システムエンジニアは小さな例題を設計・実装すればよく,開発が効率化される.たとえ機能の追加などのために例題を修正するとしても,改修・検証にかかる負荷は小さい.制御エンジニアは実時間システムエンジニアの手を煩わせることなくシステムを再構築できる.

本手法を実システム開発に適用した結果,従来方式の2.7倍の効率化が達成できた.ActiveRingモデルと例題拡張法は,トンネル換気制御,下水処理,鉄鋼圧延や食品製造のプラントに適用され,開発負荷の削減に貢献している.

今後は例題設計の支援法を研究する予定である.

謝辞 本論文の作成にあたりご指導いただいた京都大学情報学研究科上林教授,垂水助教授に深謝します.

参考文献

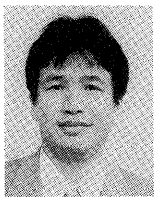
1) Liu, C.L. and Layland, J.W.: Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment, *J. ACM*, Vol.20, No.1, pp.46–61 (1973).
2) Stewart, D.B., Volpe, R.A. and Khosla, P.K.: Design of Dynamically Reconfigurable Real-Time Software using Port-Based Ob-

jects, *IEEE Trans. Softw. Eng.*, Vol.23, No.12, pp.759–776 (1997).
3) Shimakawa, H., Ohnishi, H., Mizunuma, I. and Takegaki, M.: Acquisition and Service of Temporal Data for Real-Time Plant Monitoring, *Proc. 14th Real-Time System Symposium* (1993).
4) 島川博光，水沼一郎，竹垣盛一：プラント履歴データの実時間獲得・提供システム，信学論，Vol.J78-D-I, No.8, pp.798–806 (1995).
5) Shimakawa, H., Ido, G., Takada, H. and Takegaki, M.: Active Transactions Integrated with Real-Time Transactions According to Data Freshness, *Proc. 3rd Real-Time Tecnology and Application Symposium* (1997).
6) Rajkumar, R.: *Synchronization in Real-Time Systems, A priority Inheritance Approach*, Kluwer Academic Publishers (1991).
7) Özsoyoğlu, G. and Snodgrass, R.T.: Temporal and Real-Time Database: A Survey, *IEEE Trans. Knowledge and Data Engineering*, Vol.7, No.4, pp.513–532 (1995).
8) Ramamritham, K.: Real-Time Database, *International Journal of Distributed and Parallel Database*, Vol.1, No.2, pp.199–226 (1993).
9) McCarthy, D.R. and Dayal, U.: The Architecture of An Active Database management Systems, *Proc.of ACM SIGMOD* (1989).
10) Gallagher, K.B. and Lyle, R.: Using program Slicing in Software Maintenance, *IEEE Trans. Softw. Eng.*, Vol.17, No.8, pp.751–761 (1991).
11) Gerber, R. and Hong, S.: Slicing Real-Time Programs for Enhanced Schedulability, *ACM Trans. Programming Languages and Systems*, Vol.19, No.3, pp.67–73 (1997).
12) Boyer, S.: *SCADA: Supervisory control and data acquisition*, Instrument Society of America (1993).
13) Ozkul, T.: *Data Acquisition and Process Control using Personal Computers*, Marcel Dekker, Inc. (1996).
14) Ishikawa, Y., Tokuda, H. and Mercer, C.W.: An Object-Oriented Real-Time Programming Language, *Computer*, Vol.24, No.10, pp.67–73 (1991).
15) Niehaus, D. Stankovic, J.A. and Ramamritham, K.: A Real-Time System Description Language, *Proc. 1st Real-Time Application and Technolog Symposium* (1995).
16) Kuo, T.W. and Mok, A.K.: Incremental Reconfiguration and Load Adjustment in Adaptive Real-Time Systems, *IEEE Trans. Comput.*, Vol.46, No.12, pp.1313–1325 (1996).
17) Zhou, L., Rundensteiner, E.A. and Shin, K.G.: Schema Evolution of an Object-Oriented Real-Time Database System for Manufacturing Automation, *IEEE Trans. Knowledge and Data Engineering*, Vol.9, No.6, pp.956–977 (1997).

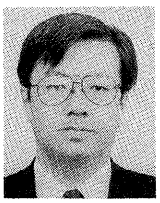
(平成10年7月13日受付)
(平成11年2月 8 日採録)

島川 博光（正会員）
1961年生．1984年京都大学工学部情報工学科卒業後，同大学院に進学．1986年三菱電機入社．1993年から1994年まで米国マサチューセッツ大学客員研究員．実時間システム，データベースの研究に従事．IEEE，ACM各会員．

井戸 譲治（正会員）
1969年生．1992年大阪大学工学部通信工学科卒業後，同大学院に進学．1994年三菱電機入社．現在，産業システム研究所にて実時間システム，データベースの研究開発に従事．



高田 秀志（正会員）
1968年生．1991年京都大学工学部情報工学科卒業後，同大学院に進学．1993年三菱電機入社．現在，産業システム研究所にてデータベースの研究開発に従事．1998年システム制御情報学会論文章授賞．システム制御情報学会会員．



堀池 聡（正会員）
1957年生．1980年京都大学工学部数理工学科卒業後，同大学院に進学．1982年三菱電機入社．1986年から1987年まで米国マサチューセッツ工科大学客員研究員．工学博士．分散実時間システムの研究に従事．電子情報通信学会，IEEE各会員．