概念データモデルに基づくプログラム仕様記述言語の実時間処理への拡張

岡本克己[†☆] 橋本正明[†☆☆]

対象世界の構造的側面，すなわち対象世界に現れる概念のつながりに着目した，概念データモデルに基づくプログラム仕様記述言語を，実時間処理へ適用するための拡張方法を提案する．従来，実時間処理用の仕様記述言語には，事象や状態遷移といった対象世界の動的側面，すなわち対象世界の時間の流れに着目した言語が多かった．しかし，それらの言語は，大量の情報処理が混在する実時間処理の仕様を理解しやすく記述するのには必ずしも適していない．このため，構造的側面に着目した実時間処理用の仕様記述言語を研究することが重要である．ところで，構造的側面に着目した言語では，時間的な関係も概念のつながりと同様な構造として記述される．このため，実時間データ入力や，実時間データ出力，長時間連続稼働といった実時間処理の特徴に対しては，言語仕様や，プログラム生成法，言語適用法に特別な対策を要する．そこで本論文では，筆者らが研究中の概念データモデルに基づくプログラム仕様記述言語 PSDL (Program Specification Description Language) の上でそれらの対策を述べる．具体的には，実時間データ入出力相互の時間的前後関係を取り扱うための入出力タイミング制約を記述する言語仕様や，プログラムを生成するためのタイミング不一致の検出解決法，因果関係を考慮した言語適用法などを述べる．また，プログラム生成法の一部について行った実験についても述べる．

An Extension of Conceptual Data Model-based Program Specification Description Language to Real-Time Processing

Katsumi Okamoto[†☆] and Masaaki Hashimoto[†☆☆]

This article presents how a conceptual data model-based program specification description language, which is adapted to describing the structural aspect of universes of discourse, is extended to real-time processing. Most real-time processing program specification description languages are tuned to the dynamic aspect of universes of discourse. However, real-time processing programs have sometimes come to include more complex control structures, so comprehensible description of the structural aspect is important. When the real-time processing programs are described by the language tuned to the structural aspect, we must extend the language for the real-time data inputs and outputs, and the long time operations of the programs. Therefore, this article describes the extensions: 1) the language specification introducing timing constraint, 2) the program generation method detecting and solving timing clash between the real-time data inputs and outputs, 3) the language application method based on causality, and the experiment in a part of the programgeneration method.

1. はじめに

今日，信頼性が高いプログラムを効率よく作成する方法が大きな課題となっており，その一つの解決策としてソフトウェア再利用が注目されている．そこで著者らは，従来効果を上げてきたプログラム再利用をさらに改善するため，プログラム仕様の再利用について研究している．具体的には，再利用対象仕様の理解や拡張を容易にするため，概念データモデルの一つである ER (Entity;-Relationship) モデルと，そのモデル上の従属性制約とを適用した**プログラム仕様記述言語 PSDL** (Program Specification Description Language)[1),2)]を研究している．

ところで，通信システムなどの実時間処理の分野では，PAISLey[3)] や LOTOS[4)]などの多くの仕様記述言語は，事象や状態遷移といった対象世界の動的側面を理解しやすく記述している．しかし複雑な関係を持った大量情報の処理が混在した実時間処理に対しては，そのような言語がいつも適合するとは限らず，対象世

† ATR 通信システム研究所
ATR Communication Systems Research Laboratories
☆ 現在，住友金属工業(株)
Recently with Sumitomo Metal Industries, Ltd.
☆☆ 現在，九州工業大学
Recently with Kyusyu Institute of Technology

界の構造的側面を理解しやすく記述できることも重要となる．構造的側面を理解しやすく記述するのには，データベースの分野で発展してきたERモデルが適している．そこで筆者らは，ERモデルを適用したPSDLで実時間処理を記述して，プログラムを生成する研究を行った．

当初から研究しているバッチ処理[1),5)~7)]では，プログラム実行前に入力データが全て準備されており，出力データはプログラム実行後に得られればよい．それに対して，実時間処理は，1)プログラム実行中に各入力ポートにデータがランダムに到着する**実時間データ入力**，2)データ入力後ただちに計算結果を出力しなければならない**実時間データ出力**，3)**長時間連続稼働**，といった特徴を持っている．

これらの特徴は，動的側面，すなわち対象世界の時間の流れに着目して仕様を記述するPAISLey[3)]などの言語とは親和性がよい．しかし，構造的側面に着目したPSDLでは，対象世界の時間的な関係も概念のつながりと同様な構造として記述されるため，上記の実時間処理の特徴に対しては，a)言語仕様や，b)プログラム生成法，c)言語適用法，に特別な対策を要する．そのうち，すでにa)とb)の一部については断片的に報告した[2),8)]．

そこで本論文の目的は，実時間処理の上記の特徴1)，2)，3)が，構造的側面に着目した仕様記述言語PSDLの上記a)，b)，c)へ及ぼす問題点を系統的に解明して，その解決法を提案することである．また，プログラム生成法の一部について行った実験も述べる．以下，第2章にERモデルを適用した仕様記述言語で実時間処理を取り扱う際の問題点を解明する．第3章でPSDLの言語仕様を概説し，第4章でプログラム生成法とその実験について述べる．第5章で全体を考察する．

2. ERモデルと実時間処理

ERモデルは，元来データベースのデータ構造を設計するため，対象世界の情報構造を，概念のつながりに着目して理解しやすく記述することを目的として提案され，最近ではオブジェクト指向分析でも使用されている[9)]．データベースを設計するためのERモデルは，各時点で見た対象世界の静的な情報構造を記述すればよく，情報の動的な変更の方はプログラムが担っている．

一方，PSDLのような仕様記述言語に適用されたERモデルは，変更前と変更後の情報を含めて情報構造を記述しなければならない．さらに，情報の変更自体も表すため，従属性制約と呼ばれる計算方法の記述法をERモデルに拡張している[1)]．

本章では，ERモデルと従属性制約を適用した仕様記述言語で実時間処理を取り扱う際の，言語仕様，プログラム生成法，言語適用法へ及ぼす問題点を系統的に解明し，その解決法を述べる．

2.1 言語仕様

対象世界の情報の枠組を記述する上で，対象世界の時間に関する概念はERモデルとその上の従属性制約を用いて記述することができる．ただし，実時間入出力データのタイミングの取り扱いについて特別な考慮が必要である．以下に，ERモデルとその上の従属性制約を用いて時間に関する概念を取り扱う方法を述べ，タイミングの取り扱いに関する問題点とその解決策を説明する．

事物または事象の発生や，変化，消滅の時刻，またその間の時間は，事物や事象を表す実体の属性として表すことができる．なお，前述のように仕様記述言語に適用されたERモデルは変化前と変化後の情報を含めて情報構造を記述しなければならないので，実体自体が変化したり，消滅するという考え方はされない．変化や消滅に対しては，その事実を表す新たな実体が現れなければならない．

事物や事象の発生や，変化，消滅の時間的前後関係，また事物や事象の継続期間の包含関係は関連型で表すことができる．さらに，事象の発生時刻と消滅時刻とを表す属性から，事象の継続時間の属性値を得るような計算は，属性値に関する従属性制約で取り扱うことができる．また，事象の状態に基づいて，次に起きる事象が得られることや，上記のように事象の変化や消滅を表す新たな実体が得られることは，新しい実体を他の実体から計算して得るという実体存在に関する従属性制約を用いて定めることができる．さらに，たとえば二つの事象の発生時刻を表す属性から，その値の大小関係を判定して，事象発生の前後関係を表す関連を得ることなどは，関連存在に関する従属性制約として取り扱うことができる．

なお，実時間処理では事象相互の時間的前後関係が処理上重要な意味を持つことが多い．このため，時間的前後関係を表す関連型に，時間的な前と後とを表す“pre”，“post”を実体の役割として記述する必要がある．ところが，二つの実時間データ入出力ポートで発

生する入出力事象相互の時間的前後関係を検知することは，ER モデルに時間の概念がないので，前述の従属性制約では繁雑な記述になる問題点が判明した．そこで，二つのポートで発生する事象の前後関係を検知して，その関係を表す関連を生成するタイミング制約を導入した．

2.2 プログラム生成法

実時間処理のプログラム仕様から手続き型のプログラムを生成する際には，入出力データのタイミングを考慮する必要がある．

2.2.1 入出力データのタイミング不一致

実時間入力データはプログラム実行中にランダムに到着するので，ある出力データを得るための計算に必要な入力データが，すべて同期して到着するとは限らない．その場合，本論文では，その入出力データに**タイミング不一致**があるといい，タイミング不一致のデータを見つけ出すことをタイミング不一致の検出という．

ところでタイミング不一致があれば，先に到着した入力データは，他の入力データが到着するまでプログラム中で待ち合わせてから計算しなければならない．そのため，プログラム中にテーブルのようなデータ待ち合わせ機構が必要になるが，その機構を，タイミング不一致がないデータにも適用したのではプログラムの実行効率が悪化する．そこで，その機構はタイミング不一致のデータのみに適用したプログラムを実現することを，本論文ではタイミング不一致の解決という．

タイミング不一致の検出と解決は，対象世界の時間の流れに着目した言語では問題になることが少ないが ER モデルと従属性制約を用いた言語では計算が並行的に記述されるため，問題となる．その問題点の解決方法を以下に述べる．

2.2.2 タイミング不一致の検出

以下の点を考慮して，タイミング不一致を検出する．

（1） 実時間入出力ポートの相互関係

二つの入力ポートからおのおの入力される二つの実体の属性値が，同じ制約の計算で同時に用いられる場合，ランダムに到着する実時間入力のため，一方の入力ポートに一つの実体が到着する間に，他方の入力ポートに複数個の実体が到着することがある．そのため，後者の複数個の実体について待ち合わせが必要となり，タイミング不一致が発生する．

（2） 状態遷移への実時間入力

実時間処理で状態遷移を記述する場合，同一ポートから順次入力される事象を表す実体が状態遷移を引き起こす．この場合，“pre” の状態を表す実体は，“post” の状態を表す実体を計算して得るため，次の実時間入力を待ち合わせることが必要となり，タイミング不一致が発生する．

（3） タイミング制約

二つの実時間入力ポートの間に，2.1 節で述べたタイミング制約が記述されており，時間的に先に入力される側の実体M個が，時間的に後に入力される側の1個の実体と関連を持つ場合，時間的に後に入力される側の実体が入力されるまで，先に入力されたM個の実体を保持しておく必要がある．このため，タイミング不一致が発生する．

2.2.3 タイミング不一致の解決

タイミング不一致を解決したプログラムは以下のように実現できる．

（1） データドリブン方式による実行制御

実時間処理ではプログラム実行中に実時間入力データがランダムに到着し，しかも実時間データ出力のため，計算可能なデータはただちに計算して出力しなければならない．そのため，データの到着によって計算手続きを起動するデータドリブン方式で，被生成プログラムを制御することが必要である．

（2） プログラムのデータ構造

タイミング不一致が存在する箇所の実体型または関連型については，同じ型の実体または関連のデータが2個以上，待ち合わせることが起こりうる．そのような実体型や関連型には配列変数でテーブルを割り付ける必要がある．

2.2.4 ガーベジコレクション

実時間処理プログラムは長時間連続稼働するので，2.2.3 項（2）で述べた配列変数では，実体などを表すデータがオーバーフローする問題が生じる．そのため，実時間の性質を用いて，不要になったデータを消去するためのガーベジコレクション法を作成した．詳細は 4.3 節に述べる．

2.3 言語適用法

ER モデルとその上の従属性制約を適用した仕様記述言語において実時間処理を取り扱う際の，言語適用上の注意事項と制限事項を述べる．

（1） 時間の次元と実時間

たとえば，ある人が今，昨日の出来事を話している

とすると，そこには“今”と“昨日”との二つの時間次元が現れている．この場合，時間次元は，昨日の出来事を今話しているという事象の時間的な自由度，すなわち昨日の時間の流れと今の時間の流れを指している．その時間次元を情報層で記述する場合，各々の次元の記述方法に本質的な差異はない．しかし，データ層とアクセス層とで実時間へ対応付けられるのは，その中で“今”を表す時間次元だけである．

（2） 状態遷移

長時間連続稼働する実時間処理プログラムでは，対象世界に存在するものの性質変化を状態遷移で表すことが多い．状態遷移をERモデルとその上の従属性制約で記述するには，おのおのの時点のものを実体で表し，それらの実体は同じ実体型に含める．その実体型に対して推移閉包の関連型を定義して，その関連型の上の従属性制約に状態遷移関数を記述する．

（3） 因果関係

実体間の因果関係[10]は，実体間の時間的前後関係を示す関連型を用いて表すことができる．しかし，実体は関連型の上の従属性制約を通して相互に参照可能である．このため，因果関係を表す関連型の上の従属性制約については，時間の流れの方向と，参照による情報の流れの方向とが一致するようにプログラム仕様を記述しなければならない．もし方向が一致しなければ，過去の事象へ未来の事象が影響を及ぼすという不自然な記述になる．

3. 仕様記述言語 PSDL

PSDLプログラム仕様は，プログラム入出力データで表された情報の枠組みを定める**情報層**，その入出力データの構造を記述する**データ層**，入出力ファイルや入出力ポートへのアクセス方法を定める**アクセス層**，といった3階層から構成される．本章では，3.1節の例題を記述した**図1**のプログラム仕様と，それを図示した**図2**を用いてPSDL言語仕様を説明する．

3.1 例　題

Jacksonのシステム開発法の説明で用いられたエレベータ問題[11]を簡略化し，例題として用いる．エレベータは6階建ての全階をサービスし，最上階を除いた各階には上行きの押しボタンがある．また，最下階を除いた全ての階に下行きの押しボタンがある．エレベータ内には，階番号が記された六つの押しボタンがある．エレベータは，モータによって上げ下げされ，エレベータ・シャフトの全階にあるセンサーが，エレベータの位置を感知して現在の階をエレベータ管理システムへ知らせる．

モータ・コマンドには，a)モータを止めるためのSTOP，b)エレベータを上向きに動かすためのSTART_UP，c)エレベータを下向きに動かすためのSTART_DOWN，の三つがある．

エレベータは，いずれかの階で待機しており，ユーザからの要求があると動き出す．エレベータ管理システムは次にエレベータが進むべき方向や各階に止まるかどうかを，現在のエレベータの進行方向と，エレベータの現在位置と，ユーザからの要求とによって決定する．また，ユーザ要求がなければ，エレベータはすべてのユーザ要求を満たし終わった階で待機するものとする．なお，ドアの開閉は考慮しない．

3.2 情 報 層

図1における，00行目のINFORMATION文と90行目のDATA文の間の仕様が情報層であり，図2では最上位に図示している．

（1） 実体型，属性，関連型

実体型は，図2では太い四角形で示しており，図1では01行目のE（Entity type）文で記述する．

実体型の各々の属性は05行目のA（Attribute）文で記述する．ここで，05行目のNUM（NUMber）はその属性の定義域が数値であることを示し，07行目のSTR（STRing）はその属性の定義域が文字列であることを示す．また，対象世界で同時に存在する実体の個数を，実体型ごとに02行目のTEN（Temporary Entity Number）文で記述する．02行目の“−3”は個数が3個以下であることを示し，15行目の“1”は常に1個であることを示す．

関連は，実体を結んだ破線で示し，これらの関連を含む関連型は，太いひし形で示す．おのおのの関連型は，図1の61行目のR（Relationship type）文で記述する．それに続けて，関連型で対応付けられた実体を62，64行目のC（Collection）文で記述し，実体型内の一つの実体が持つ関連の個数を63，65行目のRN（Relationship Number）文で記述する．65行目の“M”は個数が0個以上であることを示し，63行目の“1”は1個であることを示す．なお，実体型とその役割は，C文に“役割名．実体型名”という形で記述する．対応付けている実体型が相互に異なっている場合は，62行目のように役割を省略できる．

（2） 制　約

属性値従属性制約（AD：Attribute value Depend-

```
00: INFORMATION
01:   E request
02:     TBN -3
03:     EC post.request_renewal.pre.request
04:       ON post.request_renewal.pre.request.outstanding == "NO"
05:     A floor              NUM
06:       := post.request_renewal.pre.request.floor
07:     A direction          STR
08:       = post.request_renewal.pre.request.direction
09:     A outstanding        STR
10:       = IF (.request_satisfaction..elevator.floor == floor
11:         && .request_satisfaction..elevator.next_action == "STOP")
12:           "YES"
13:         ELSE    "NO"
14:   E elevator
15:     TBN 1
16:     EC post.state_transition.pre.elevator
17:       ON post.state_transition.pre.elevator.next_action == "STOP"
18:     A floor              NUM
19:       = post.state_transition.pre.elevator.floor
20:         ON post.state_transition.pre.elevator.next_action == "STOP"
21:     A direction          STR
22:       = IF (post.state_transition.pre.elevator.floor < floor)
23:           "UP"
24:         ELSE
25:           IF (post.state_transition.pre.elevator.floor > floor)
26:                 "DOWN"
27:           ELSE    post.state_transition.pre.elevator.direction
28:     A next_action        STR
29:       = IF (INCLUDE(.request_satisfaction..request.floor == floor
30:           && (.request_satisfaction..request.direction == direction
31:             || .request_satisfaction..request.direction == "EL"
32:             || (direction == "UP" &&
33:               MAX(.request_satisfaction..request.floor) == floor)
34:             || (direction == "DOWN" &&
35:               MIN(.request_satisfaction..request.floor) == floor)))
36:         "STOP"
37:       ELSE
38:         IF (post.state_transition.pre.elevator.next_action
39:               == "START_UP"
40:           || post.state_transition.pre.elevator.next_action
41:               == "START_DOWN"
42:           || post.state_transition.pre.elevator.next_action
43:               == "PASS")
44:           "PASS"
45:         ELSE
46:           IF (INCLUDE
47:           (.request_satisfaction..request.floor > floor))
48:             "START_UP"
49:           ELSE    "START_DOWN"
50:   E motor
51:     TBN 1
52:     EC .motor_command..elevator
53:         ON .motor_command..elevator.next_action != "PASS"
54:     A command            STR
55:       = .motor_command..elevator.next_action
56:   R request_renewal
57:     C pre.request
58:       RN M
59:     C post.request
60:       RN M
61:   R request_satisfaction
62:     C .request
63:       RN 1
64:     C .elevator
65:       RN M
66:       RC .elevator.post.state_transition.
67:             pre.elevator..request_satisfaction
68:           ..request.pre.request_renewal.post.request.
69:   R state_transition
70:     C pre.elevator
71:       RN 1
72:     C post.elevator
73:       RN 1
74:       RC post.elevator.post.elevator_order.
75:             pre.state_transition.post.elevator.pre
76:           && (pre.elevator.next_action == "START_UP"
77:             || pre.elevator.next_action == "START_DOWN")
78:       RC post.elevator.post.elevator_order.pre.elevator.pre
79:           && pre.elevator.next_action == "PASS"
80:   R elevator_order
81:     C pre.elevator
82:       RN 1
83:     C post.elevator
84:       RN 1
85:   R motor_command
86:     C .elevator
87:       RN M
88:     C .motor
89:       RN 1
90: DATA
91:   I request_data
92:     O request_record
93:       = request
94:       = IF (POST) request_satisfaction..request
95:         TRN 1
96:         %1d request_floor
97:           = request.floor
98:         %4s request_direction
99:           = request.direction
100:  I elevator_data
101:    O elevator_record
102:      = elevator
103:      = IF (PRE) request_satisfaction..elevator
104:        TRN M
105:      = IF (PRE) elevator_order.pre.elevator
106:        TRN 1
107:      = IF (POST) elevator_order.post.elevator
108:        TRN 1
109:        %1d elevator_floor
110:          = elevator.floor
111:  I motor_data
112:    O motor_record
113:      = motor
114:        %10s motor_command
115:          = motor.command
116: ACCESS
117:   RTD request_d INPUT    request_record
118:   RTD elevator_d INPUT   elevator_record
119:   RTD motor_d   OUTPUT motor_record
```

図 1 PSDL プログラム仕様
Fig. 1 PSDL program specification.

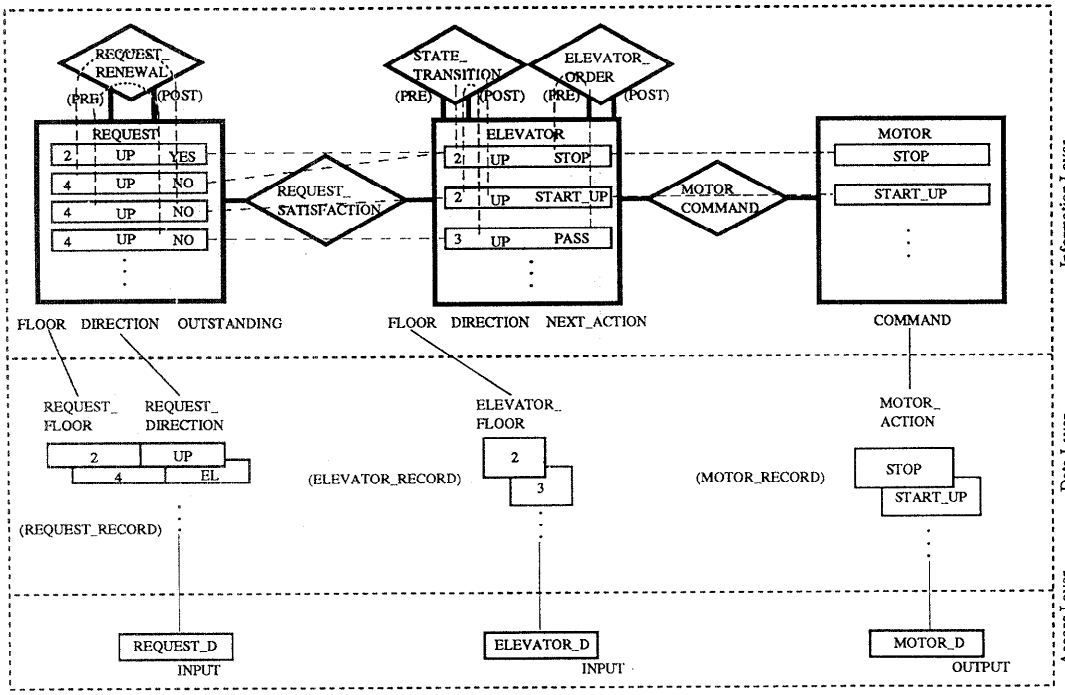


図 2 PSDL プログラム仕様図
Fig. 2 Program specification illustration.

ency constraint) は，属性の値を得るための制約である．図2では，たとえば関連 request_satisfaction で対応付けられた実体 request と elevator において，実体 request が満たされたか否かを表す属性 outstanding の値は，実体 request の属性 floor や direction と，実体 elevator の属性 floor や direction とから計算して得る．図1では，10 行目の＝(equal) 文で記述する．AD が参照する属性は，“属性名” あるいは “役割$_1$. 関連型名$_1$. 役割$_2$. 実体型名$_1$.….属性名” の形で記述する．前者は，値が得られる属性と同一の実体の属性を指し，後者は，値が得られる属性の実体と関連で対応付けられた実体の属性を指す．また，＝文中の IF 句の条件式の形式はC言語と同じである．なお，一つの実体に2個以上の実体が対応付けられる場合は，AD に集合関数を記述しなければならない．たとえば，図1中の INCLUDE は，引数の条件式を満たす実体が存在すれば “1”，存在しなければ “0” を返す集合関数である．MIN，MAX は各々，最小値と最大値を返す集合関数である．

実体存在従属性制約 (ED : Entity existence Dependency constraint) は，新しい実体を他の実体から計算して得るための制約である．図2では，たとえば実体 elevator の属性 next_action が “PASS” でなければ，モータへのコマンドを出すため，その elevator に関連 motor_command で対応付けられた実体 motor を得る．図1では 52 行目の EC (Entity existence Condition) 文で記述する．なお，新しい実体が得られるか否かの条件式は ON 句で記述する．また，ED が参照する実体は，“役割$_1$. 関連型名$_1$. 役割$_2$. 実体型名$_1$.….役割$_m$. 実体型名$_n$” の形で記述する．

関連存在従属性制約 (RD : Relationship existence Dependency constraint) は，新しい関連を，その関連で対応付けられる実体から計算して得るための制約であり，R 文と C 文に続いて図1の66行目の RC (Relationship existence Condition) 文で記述する．RC 文は，“述語名 (役割. 実体型名. 属性名, …)” あるいは “役割$_1$. 実体型名$_1$. 役割$_2$. 関連型名$_1$.….役割$_{m-1}$. 実体型名$_n$. 役割$_m$” という形で記述する．前者は述語が真になれば新しい関連が得られる場合に用いる．後者は，既存のいくつかの関連のつながりによって結ばれた二つの実体の間に，新しい関連が得られる場合に用いる．たとえば，66 行目の RC 文は，関連 request_satisfaction が，実体 request と elevator の間の既存の関連 state_transition, request_sa-

tisfaction, request_renewal のつながりによって得られることを示す．

3.3 データ層とアクセス層

図1の90行目のDATA文と116行目のACCESS文との間の仕様がデータ層である．データ層では，図2の中段に示すように，入出力データ中のデータ項目の並び方をデータ型で階層的に記述する．まず基本データ型は，C言語と同様に定義されるデータ形式と共に96行目の％文で記述し，連接集団データ型は92行目のO(sequence Order)文で，繰り返し集団データ型は91行目のI(Iteration)文で，選択集団データ型はS(Selection)文で記述する．S文の例は図1にない．データ層と情報層との関係は以下のように記述する．まず，情報層にはERモデルを適用したので，入出力データは実体やその属性，関連を表すものと見なす．そこで，97行目の％文に続く＝文で基本データ型を実体型の属性に対応付ける．また，93行目のO文に続く＝文で属性に対応付けた基本データ型のデータの並びを対応付ける．さらに関連型も，その並びを定義した連接集団データ型に対応付ける．しかし，この例は図1にない．

二つの入出力ポートで各々発生する入出力の時間的な前後関係を明確に記述するために，その時間的な前後関係を，**入出力タイミング制約**(TC：input-output Timing Constraint)で情報層の関連型と対応付ける．具体的には，94，103行目の＝文で“＝IF(PRE)関連型名．役割．実体型名”と“＝IF(POST)関連型名．役割．実体型名”という形で記述する．ここで前者をpre TC，後者をpost TCと呼ぶ．たとえば，94，103行目のTCは，ある時点に到着した実体elevatorと，その直後に到着した実体requestとを，関連request-satisfactionが対応付けることを定めている．さらに，TCから得られる同一実体につながる関連の個数は，95，104行目のTRN(Timing Relationship Number)文で記述する．その中の“M”と“1”は，前述のRN文における定義と同じである．

アクセス層の仕様は図1の116行目のACCESS文の後に記述し，図2では最下位に示している．各装置とつながる**実時間データ入出力**ポートをデータセットと呼び，117行目のRTD(Real-Time Dataset)文で，入出力ポート名，入出力区別，対応するデータ構造を順に記述する．

4. 実時間処理プログラム生成法

実時間処理のPSDLプログラム仕様から手続き型のプログラムを生成する方法を述べる．また，その一部をPSDLコンパイラで実験したので，その結果も述べる．

4.1 タイミング不一致の検出

タイミング不一致検出のため，まずPSDLプログラム仕様から有向グラフを作成する．グラフには，構造節点に分類される実体型節点や属性節点，関連型節点，データセット節点とがあり，また制約節点に分類されるAD節点やRD節点，ED節点，TC節点とがある．なお，データセット節点は，そのデータセットに対応付けられたデータ型もまとめて表している．グラフの有向枝は構造節点と制約節点の間に張る．枝の方向は，情報層では実体や，属性値，関連が制約から参照される方向と，制約から得られる方向に合わせ，データ層とアクセス層では入出力の方向に合わせる．

このグラフを2.2.2項のタイミング不一致検出の考慮点に当てはめると，2.2.2項(1)～(3)が図3(1)～(3)となる．なお，**図3**(1)に示すように，一つの入力ポートと，計算が行われる節点との間に出力ポートへの分岐がある場合，実時間出力を妨げてはいけないので，データ待ち合わせ節点を分岐点と計算が起こる節点との間に限定する．

この有向グラフを2.2.2項の考え方で解析することによって，タイミング不一致を検出する．

4.2 タイミング不一致の解決

タイミング不一致を解決したプログラムをPSDLでは，以下のように実現する．

(1) プログラムのデータ構造

タイミング不一致が存在する箇所の実体型または関連型には配列変数でテーブルを割り付ける．一方，待ち合わせの必要な実体型や関連型にはスカラ変数を割り付けて，被生成プログラムの実行効率を図る．

そこで，データドリブン方式は入力データと配列変数のデータのみ適用すればよい．そのため，配列変数の各要素にフラグ変数を付加し，データが入っている参照可能な要素のフラグのみを“ON”にする．

(2) プログラムの手続き構造

プログラムの手続きは1個以上の手続きブロックから構成され，そのブロックは1個以上の従属性制約の処理ルーチンから構成される．各ブロックは入力デー

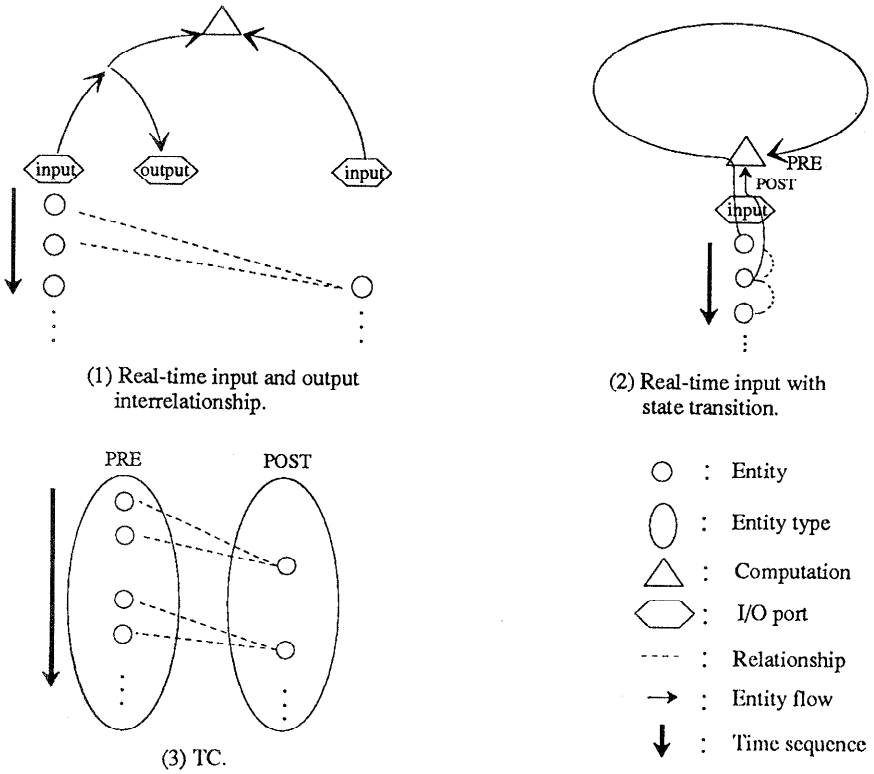


図 3 タイミング不一致検出
Fig. 3 Timing clash detection.

タ，または参照可能フラグが “ON” の配列変数のデータを参照して，同じブロック内の従属性制約間はスカラ変数でデータを受渡しながら計算を行い，その結果を配列変数に代入するか，または出力する．さらに，配列変数を参照している AD や ED，RD のインスタンスにもフラグを設け，実行済みのものは “ON” にする．このフラグは後述のガーベジコレクションで用いる．

（3） プログラムのコード生成

上記のように自動的に設計されたプログラムのデータ構造，手続き構造に沿ってスケルトンを割り当て展開してコード生成を行う．スケルトンは，C言語のマクロとして記述されたもので，数行から 10 数行のCコードからなる部品の集まりである．

4.3 ガーベジコレクション

PSDL では対象世界の事物や事象が発生，変化，消滅するたびに，その事実を表す実体が生成されるが，実体の消去を指示する文はない．したがって2.2.4項で述べた，不要になったデータを消去するためのガーベジコレクションが不可欠である．

本論文のガーベジコレクション法は，実行が終了した AD や ED，RD のインスタンスと，参照が全て終了した実体と，その実体につながる関連とを表すデータを消去する．ところが，実体の型が同じであっても，実体につながる関連のつながり方が常に同じとは限らない．たとえば，図2の中の実体型 elevator の2番目に示した実体 “2 UP START_UP” には関連 elevator_order はつながっていないが，その他の実体には elavator_order がつながっている．このため，関連のつながりに基づいて実行される AD については，実体につながる関連がそれ以上現れないことを保証しなければ，その実体に対する参照の終了を確認できない．

これは，制限された範囲で，実時間の性質を用いて以下のように保証できる．まず，同一の実体型の間に “pre”，“post” の役割付きで定義された関連型については，その関連が実時間に沿って順次現れ，実時間を遡って現れることはないものとする．たとえば，図2

において，実時間に沿って現れる関連 state_transition でつながれた三つの実体 elevator について，その1番目と3番目をすでに関連 elevator_order がつないでいるので，2番目の実体には関連 elevator_order が現れないものとする．したがって，最新の関連より古い関連と，それらの関連にのみつながっている実体を消去してよい．

4.4 実　　験

上記のプログラム生成法のうち，データドリブン方式とガーベジコレクション法とが正しく動作することを実験で確認した．タイミング不一致検出解決については，そのグラフ処理が，別途研究中の入出力データ構造不一致検出解決[6),7)]のためのグラフ処理法と類似しているので，後者の完成後，前者を組み込む予定である．

（1） 実験用 PSDL コンパイラの作成

コンパイラは，その大部分へ自己記述を適用して SUN 3/60 C 上に作成した．その規模は 13,000 行であった．その他にC言語で作成された500行のスケルトンがあった．また，PSDL プログラム仕様中の関数や述語はC言語で定義し，その規模は 1,000 行であった．

（2） 被生成プログラムの効率

表1に二つのテストプログラムの評価データを示す．なお，表中の elevator は図1のエレベータ問題であり，AB は簡単な通信の例題である AB (Alternative-Bit) プロトコル[12)]問題である．今回の実験ではタイミング不一致の検出と解決を適用しなかったので，表1からわかるように，被生成Cプログラムの実行効率の悪さは，手書きCプログラムの 12 倍から 17 倍であった．

表 1 評価データ
Table 1 Evaluation data.

テスト・プログラム仕様	elevator	AB
PSDL 記述（行数）†	159	100
被生成Cプログラム（行数）	1,983	1,132
手書きCプログラム（行数）	169	68
データ定義部の比率（被生成／手書き）††	6.04	10.32
手続き部の比率（被生成／手書き）††	12.96	17.38
全行の比率（被生成／手書き）††	12.51	16.51

† Cによる関数，述語定義を含む．
†† CプログラムからCコンパイラ Gcc Ver. 1.36 によって生成したアセンブラ・プログラムのステップ数の比率．

（3） プログラムの作成工数

二つのテストプログラムは，PSDL による作成工数は共に約2日間であり，C言語による作成工数はエレベータ問題が5日間，AB プロトコル問題が1日であった．

5. 考　　察

本論文の内容を考察し，今後の課題を述べる．

（1） 実時間処理と PSDL 言語仕様

PSDL の情報層は，プログラムで処理される対象世界の情報の枠組みを記述しているので，バッチ処理や実時間処理といった処理形態の影響を受けることは少ない．実際，3.2 節で述べた情報層の言語仕様はバッチ処理[6)]と比較して，TEN 文しか差がない．このため，情報層の仕様を実時間処理とバッチ処理との間で共用する可能性が高まり，ソフトウェアの再利用性が向上する．

一方，データ層とアクセス層は実時間処理形態の影響を受けており，その典型は TC である．二つの入出力ポートで各々発生する入出力の時間的な前後関係を取り扱うことは，PAISLey[3)]のように1次元の時間の流れに従って仕様を記述する言語では容易である．しかし，時間の関係も概念のつながりと同様に構造として捉える PSDL では特別な工夫が必要なので，TC を導入した．

（2） プログラム生成とタイミング不一致

本論文では実時間処理に特有なタイミング不一致の検出解決法を述べ，文献 6) ではバッチ処理に特有な構造不一致の検出解決法を述べた．ところが，たとえばオンライン在庫管理で1画面の帳票に何件もの入出庫が記入でき，その帳票が実時間で処理される場合，構造不一致がタイミング不一致に混在して現れる．しかし，両者の関係はまだ検討していない．今後は両者の検出解決法を関係付けることが必要である．

本論文の PSDL コンパイラにはタイミング不一致検出解決法を組み込んでいないため，被生成プログラムの実行効率の悪さは手書きのものと比較して 12 倍から 17 倍であった．一方，構造不一致検出解決法を組み込んだバッチ処理用の PSDL コンパイラでは，1.3 倍から 2.5 倍におさまっている[7)]．実時間処理においてもタイミング不一致検出解決法を組み込めば，同程度の実行効率が得られるものと予想される．そのため，実験を行って実行効率を見積ることが必要である．

（3） 実時間処理のガーベジコレクション

従来，ストップスタート方式のガーベジコレクションでは，ガーベジコレクションによるプログラム停止時間が長くなるので，実時間ガーベジコレクションではプログラム停止時間を一定に抑える研究がなされてきた[13]．一方，4.3 節で述べた本論文のガーベジコレクションの特徴は，実時間で順次現れる関連型の関連が，時間を遡って現れることはないという実時間の性質を用いて，不要となったデータを自動的に検出することにある．ところで pre 側の RN の値が “M” の場合，実体につながる関連がそれ以上現れないことは保証できないので，本論文の方法のみでは不要データの検出ができない．これは今後の研究課題である．

（4） PSDL の時間概念

対象世界に存在する実体間の時間的前後関係は，その実体間の因果関係によって表すことができる[10]．したがって，実体間の時間関係を表すために，時計概念を導入した絶対時間を適用する必然性はなく，実体間の因果関係を表すことさえできれば，その実体間のみの時間的前後関係を相対時間として表すことができる．PSDL はこの考え方に基づいており，実時間の因果関係を表す制約を，実体間の時間的前後関係を表す制約として用いた相対時間を適用している．

（5） プログラムの作成工数と記述量

二つのテストプログラムの間の作成工数および記述量の差は，4.4 節（3）と表 1 に示すように PSDL では小さく，C言語では大きい．PSDL では，AB プロトコル問題のように状態遷移が 1 箇所だけで起きている単純な問題でも，ある程度の作成工数と記述量が要求されている．一方，エレベータ問題のように elevator と request の 2 箇所で状態遷移が起きている複雑な問題になっても，C言語ほどには作成工数と記述量が増えていない．これは，対象世界の構造的側面に着目した PSDL の利点が現れているものと考えられる．

（6） 対象世界の構造的な見方と動的な見方との変換

対象世界を構造的に捉える言語も動的に捉える言語も必要なものは全て表せるように設計されるので，その記述能力に差はない．両者の違いは，同一の対象世界を記述した際，仕様の記述性や理解性の差となって現れる．記述性や理解性は，仕様を作成したり再利用する際に重要なので，両言語は共に存在意義を持っている．

ところで，対象世界には構造的な側面と動的な側面とが混在する場合が多い．たとえば，筆者らが仕様再利用の実験[14]に用いた社会保険システムでは，掛金や年金の計算は構造的に捉えやすい．一方，異動や裁定，年金受取による加入者状態の変化は動的に捉えやすい．そのため，構造的な側面と動的な側面とにおのおのの言語を適用して記述性と理解性とを確保し，言語変換で一つの仕様にまとめて，仕様の分析や実行，プログラム生成を行うことが今後の課題である．

6. おわりに

実時間処理プログラムの特徴である実時間データ入出力と長時間連続稼働とが，対象世界の構造的側面に着目した仕様記述言語 PSDL へ及ぼす問題点を解明して，その解法法を提案した．PSDL において，言語仕様については，実時間データ入出力相互の時間的前後関係を取り扱うための入出力タイミング制約を導入した．プログラム生成法については，タイミング不一致を検出解決することによって，実行効率のよいプログラムが生成され得ることを示した．また，ガーベジコレクションは長時間連続稼働に必須なので，実時間を表す関連型を活用したガーベジコレクション法を提案した．なお，実時間データ入出力を処理するため，被生成プログラムはデータドリブン方式で制御するものとした．このうち，ガーベジコレクション法とデータドリブン方式とを実験して，正しく動作することを確認した．また因果関係を考慮した言語適用法などを述べた．

今後の課題としては，タイミング不一致と構造不一致との検出解決法を統合しなければならない．タイミング不一致の検出解決法を実験して，被生成プログラムの実行効率を見積ることも必要である．さらに，構造的側面に着目した言語と動的側面に着目した言語との間で変換法を研究することが重要である．

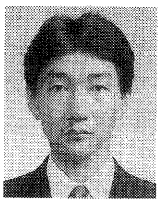
謝辞 ご指導いただいた ATR 通信システム研究所の葉原会長，寺島社長，太田室長，竹中前室長，ご意見をいただいた研究室の諸氏に深く感謝します．また，PSDL コンパイラの作成にご協力いただいた日本電子計算株式会社の諸氏に感謝致します．

参考文献

1) 橋本正明：EAR モデルに基づく情報構造記述を用いたプログラム仕様記述法 PSDM，情報処理学会論文誌，Vol. 27, No. 7, pp. 697-706 (1986).
2) Okamoto, K. and Hashimoto, M.: On Real-Time Software Specification Description with a

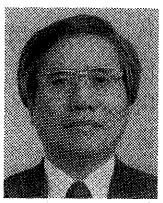
Conceptual Data Model-Based Language, *Proc. 2nd ICCI 1990*, pp. 186-190 (1990).
3) Zave, P.: An Operational Approach to Requirements Specification for Embedded Systems, *IEEE Trans. Softw. Eng.*, Vol. SE-8, No. 3 (1982).
4) Brinksma, E.: *A Tutorial on LOTOS, Protocol Specification, Testing, and Verification*, *IFIP 85*, North Holland Publishing Company, pp. 171-194 (1986).
5) 橋本正明：非手続き型言語と入出力データの構造不一致，情報処理学会論文誌，Vol. 29, No. 12, pp. 1141-1150 (1988).
6) Hashimoto, M. and Okamoto, K.: A Set and Mapping-based Detection and Solution Method for Structure Clash between Program Input and Output Data, *Proc. 14th COMPSAC 1990*, pp. 629-638 (1990).
7) 岡本克己，橋本正明：入出力データの構造不一致検出解決法に関する実験，情報処理学会論文誌，Vol. 33, No. 5, pp. 707-716 (1992).
8) Okamoto K. and Hashimoto, M.: Program Generation from Real-Time Software Specification Described with a Conceptual Data Model-based Language, *Proc. 3rd SEKE 1991*, pp. 261-270 (1991).
9) Rumbaugh, J.: *Object-Oriented Modeling and Design*, Prentice-Hall, Englewood Cliffs, N. J. (1991).
10) Durchholz, R.: Causality, Time, and Deadlines, *Data & Knowledge Engineering*, Vol. 6, pp. 469-477 (1991).
11) Jackson, M. A.: *System Development*, Prentice-Hall, Englewood Cliffs, N. J. (1983).
12) Kritzinger, P. S.: Analyzing the Time Efficiency of a Communication Protocol, *Proc. IFIP WG 6.1 4th Int. Workshop on Protocol Specification, Testing, and Verification* (1984).
13) Nilsen, K.: Garbage Collection of Strings and Linked Data Structures in Real Time, *Softw. Pract. Exper.*, Vol. 18, No. 7, pp. 613-640 (1988).
14) Okamoto, K. and Hashimoto, M.: Visual Programming with Reusable Specifications Described by a Conceptual Data Model- and Constraint-Based Language, *Proc. 4th HCI 1991*, pp. 587-591 (1991).

(平成5年3月30日受付)
(平成5年7月 8 日採録)



岡本 克己（正会員）

1963年生．1987年大阪大学基礎工学部情報工学科卒業．同年住友金属工業(株)入社．1988～1991年ATR 通信システム研究所出向．現在住友金属工業(株)情報・通信研究開発部に勤務．これまで主にソフトウェア自動作成の研究，ソフトウェア開発・保守支援システムの研究開発などに従事．



橋本 正明（正会員）

昭和21年生．昭和43年九州大学工学部電子工学科卒業．昭和45年同大学院修士課程修了．同年日本電信電話公社電気通信研究所勤務．昭和63年 ATR 通信システム研究所出向．平成5年九州工業大学情報工学部勤務．これまで主にオペレーティングシステム，中間言語マシン，データベース設計支援ツール，ソフトウェア自動作成の研究実用化に従事．著書「データ中心のプログラム仕様記述法」（井上書院）．工学博士．電子情報通信学会，ソフトウェア科学会，人工知能学会，日本建築学会，IEEE 各会員．