E-R モデルを用いた視覚的プログラミング言語：PSDL-GR とその一実現法

佐藤正和† 橋本正明† 寺島信義†

本稿は，視覚的プログラミング言語 PSDL-GR の提案とその一実現手法，およびプログラミング環境について述べたものである．PSDL-GR は，データ表現手段として E-R モデルを採用していること，および制約指向の処理方式を実現している点に特徴がある．E-R モデルは元来，データベースのスキーマ設計用に開発されたもので，データベース検索用の言語のモデルとして採用された例はあるが，一般的なプログラミングのツールとしての利用例はない．E-R モデルをプログラミング言語におけるデータ表現の手段として利用することにより，このモデルの持つ優れたデータの視覚的表現力をプログラミングに活用することができる．本言語では，E-R モデルのアトリビュートやエンティティ，リレーションシップ間の関係を制約として定義しこれらの制約を解くことをプログラムの実行とみなす方式をとっている．E-R モデル上のデータ依存関係の視覚的な表現を可能にするとともに，プログラムの再利用性や記述性の向上をはかることが狙いである．本言語の実現方法としては，制約論理型言語を用いた．本方式によって，カージナリティが1対1の範囲において制約の双方向伝搬が可能であり，その他の範囲は高階述語の導入によって処理できることが示された．さらに，本方式に基づくプログラミング環境のプロトタイプを作成し評価を行った．

A Visual Programming Language: PSDL-GR and Its Implementation

Masakazu Sato,† Masaaki Hashimoto† and Nobuyoshi Terashima†

In this paper, we describe a visual Language PSDL-GR, its implementation method and the programming environment. PSDL-GR is the language specialized by two kinds of aspects—E-R model based and constraint oriented. Originally, E-R model was developed for database schema design. It is used for some database query languages, but not been used as data modeling means in universal programming languages. PSDL-GR makes it available to utilize remarkable data representation capability of the E-R model as a basis of programming. The computation in PSDL-GR is based on the constraint satisfaction. Dependencies between attributes, entities, or relationships are defined by three kinds of constraints. To find entities which satisfy these constraints means execution of a program. These two aspects contribute to visualization of programming. To implement PSDL-GR, we use constraint logic programming language. We have developed prototype system for PSDL-GR and evaluate availability of this language.

1. はじめに

プログラム開発に視覚的なインタフェースを利用する視覚的プログラミング (Visual programming) は，従来のテキストベースの方法に比較して開発効率の向上をもたらすものと期待されている．本稿では，E-R モデルに基づいた新たな視覚的プログラミング言語 PSDL-GR を提案し，その実現法を示す．

N. C. Shu によれば，視覚的プログラミングはその目的によって以下のように類別できる[1]．1) 視覚情報の操作自体を目的としたもの，2) 視覚的対話機能を支援するもの，3) 視覚的表象を用いてプログラミングを行うもの．このうち視覚的プログラミング言語と呼べるのは 3) の範疇であり，その実現方式によってさらに以下のように分類されている．a) 図式プログラミング言語 (Diagrammatic programming language) —フローチャート等の従来紙上で使われていたプログラムの図的記述法をユーザインタフェース上で提供するもの，b) アイコニックプログラミング言語 (Iconic programming language)—プログラムの操作対象をアイコンで表現しそれらの選択や関係づけによってプログラムを表現するもの，c) 表指向プログラミング言語 (Forms-oriented programming lan-

† ATR 通信システム研究所通信ソフトウェア研究室
Communication Software Department, ATR Communication Systems Research Laboratories

guage) ―プログラミングのユーザインタフェースとして表を用いるもの．図式プログラミング言語は，従来のプログラミングの方法論を画面インタフェース上で実現したに過ぎず，視覚情報を利用することで得られるプログラミングの質的向上への寄与に乏しい．一方，アイコニックプログラミング言語や表指向プログラミング言語は，データフローや制約解消に基づく非手続き的な処理を導入するなど，2次元情報の利点を活用するための本質的な工夫が見られる．

アイコニックプログラミング言語の例としては，ThingLab[2),3)]，Pict[4)]，LabVIEW[5)]，等がある．これらは，プログラミングやプログラムの動作を可視化することによる教育効果，理解性の向上，物理現象のシミュレーションなどを目的に開発されたものである．一方，表指向プログラミング言語としては，QBE[6)]，FORMAL[7)]，および多くのスプレッドシート型と呼ばれるプログラミング言語が存在し，データベースの問い合わせなど比較的多量の構造化されたデータの処理に利用されている．

PSDL-GR は，上記の2種類の言語の性格を合わせ持つ視覚的プログラミング言語である．この言語の特徴は，データ表現に E-R モデル (Entity and Relationship Model)[8)] を用いていること，および，プログラムの実行方式として制約解消モデル[9)] を用いている点にある．これによって，アイコニックプログラミング言語が持つ優れたプログラムの理解性と，表指向プログラミング言語が持つデータ構造の表現力，さらには，制約解消モデルの持つプログラムの定義の容易性と再利用性を統合した環境を目指している．

E-R モデルは，元来データベース設計のために用いられていたもので，データの存在やその間の関連を視覚的にモデル化するのに適した表現手法である．橋本らは，E-R モデル上にデータ間の依存関係をデータフローモデルに従って定義し，非手続き的なプログラム仕様から手続き的な言語を生成するプログラム仕様記述法 PSDL を開発した[10)~13)]．PSDL-GR はデータ間の依存関係を制約として解釈することで，PSDL を拡張し視覚的プログラミング言語へと発展させたものである．E-R モデルを用いた視覚的プログラミング言語の事例はいくつか存在するが[14),15)]，これらはいずれもデータベースの問い合わせを目的とした関係代数演算を実現する特定アプリケーション向けの言語である．これに対し，PSDL-GR ではより一般的なデータ間の計算を実現する．

PSDL-GR の制約解消機構の実現に当たっては，制約論理型言語[16)~18)]を利用した．これは，1) 制約論理型言語のスキーマが E-R モデルと適合性を持つこと，2) 制約論理型言語の制約解消機構を利用できること，の理由による．

制約論理型言語への基本的な変換手法については文献 21) で述べた．今回，変換手法を見直すとともに，さらに処理系のプロトタイプを作成した．以下，本論文では，PSDL-GR の概要，PSDL-GR の実現手段としての制約論理型言語への変換手法，PSDL-GR のプロトタイプシステムについて述べる．

2. E-R モデル

準備として E-R モデルについて簡潔に紹介する．E-R モデルは対象世界をエンティティ (Entity) とリレーションシップ (Relationship) によってモデル化する．エンティティは対象世界における**物**や**事実**の存在を表し，リレーションシップはエンティティの間の**関係**を表す．

エンティティは，複数のアトリビュート値 (Attribute value) によって特徴付けられる．同種のアトリビュートの組によって特徴付けられるエンティティの集合をエンティティタイプ (Entitytype) と呼ぶ．アトリビュートのうちエンティティタイプ内の各エンティティを識別するのに用いられるものを特にキーアトリビュート (Key attribute) と呼ぶ．キーアトリビュートは一般に複数個定義可能だが，以下の議論では簡単のため1個を仮定する．エンティティを e_i，キーアトリビュートを Xk，その他のアトリビュートを X_i で表すと，エンティティタイプ $\boldsymbol{E}$ は以下のように表せる．

$$\boldsymbol{E}=\{e_1, e_2, \cdots, e_n\}$$
$$e_i=<Xk, X_1, X_2, \cdots, X_n>$$

リレーションシップは，2 ないしそれ以上のエンティティ間に関係が存在することを，これらのエンティティのキーアトリビュートの組によって表す．エンティティタイプと同様に，同種のリレーションシップの集合をリレーションシップタイプ (Relationship-type) と呼ぶ．リレーションシップを r_i，リレーションシップタイプを $\boldsymbol{R}$ で表すと，

$$\boldsymbol{R}=\{r_1, r_2, \cdots, r_n\}$$
$$r_i \in \{<id(e_1), id(e_2), \cdots, id(e_n)> \mid e_1 \in E_1, e_2 \in E_2, \cdots, e_n \in E_n\}$$

ここで，$id(e_i)$ はエンティティ e_i のキーアトリビュー

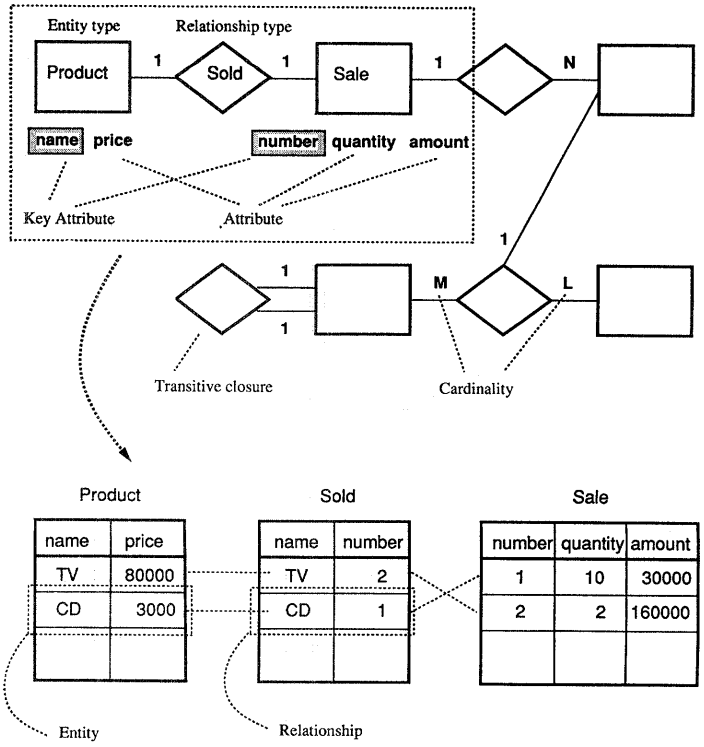


図 1 E-R モデルの例
Fig. 1 Example of the E-R model.

トである．

E-R モデルは，E-R ダイヤグラムと呼ばれる図式表現で記述可能である．図1に例を示す．矩形はエンティティタイプを，菱形はリレーションシップタイプを表す．リレーションシップタイプとエンティティタイプを結ぶ枝上の数値はカージナリティと呼ばれ，各エンティティに対しリレーションシップで関係付けられる相手エンティティの個数を表す．同図の下部に，E-R モデルを従来のリレーショナルモデルで表現した例を示す．

3. PSDL-GR

3.1 定　　義

PSDL-GR は，E-R ダイヤグラム上に定義された制約を解く言語である．プログラムはデータ構造を規定する E-R モデルとデータ間の関係を規定する制約から成る．制約には以下の3種がある．

- エンティティ存在従属性制約（C_{ED}）
  あるエンティティタイプ内の新たなエンティティが，他のエンティティの内容に依存して生成される場合の条件．C_{ED} が成立した場合には，生成されたエンティティと被参照エンティティ間にリレーションシップを生成する．被参照エンティティタイプのアトリビュートを引数とする条件式として与えられる．
- リレーションシップ存在従属性制約（C_{RD}）
  あるエンティティ間にリレーションシップが生成されるための条件．リレーションシップタイプがインスタンスとして与えられている場合は，与えられたインスタンスと C_{RD} を充足するリレーションシップタイプの集合積が有効なリレーションシップタイプとなる．上記エンティティを含むエンティティタイプのアトリビュートを引数とする条件式として与えられる．
- アトリビュート値従属性制約（C_{AD}）
  同一エンティティタイプ内で定義される場合は同一エンティティ内のアトリビュート値の依存関係を規定する．複数のエンティティタイプ間に跨って定義される場合はリレーションシップで関連付けられたエンティティ内におけるアトリビュート値の依存関係を規定する．アトリビュートを引数とする等式として与えられる．リレーションシップのカージナリティが1対 m の場合，すなわち引数として複数エンティティのアトリビュートを採る場合を特に集合関数と呼ぶ．

プログラムは初期エンティティタイプ $\boldsymbol{E}_{init}$ と初期リレーションシップタイプ $\boldsymbol{R}_{init}$ および制約で与えられる．

$\boldsymbol{E}_{init\,0}, \boldsymbol{E}_{init\,1}, \cdots, \boldsymbol{E}_{init\,n}, \boldsymbol{R}_{init\,0}, \boldsymbol{R}_{init\,1}, \cdots, \boldsymbol{R}_{init\,m},$
$C_{ED\,0}, C_{ED\,1}, \cdots, C_{ED\,l}, C_{RD\,0}, C_{RD\,1}, \cdots, C_{RD\,k},$
$C_{AD\,0}, C_{AD\,1}, \cdots, C_{AD\,j}$
$(i, k, l, m, n \geq 0)$

ここで，$\boldsymbol{E}_{init}$ は未定アトリビュート値を含んでいる．プログラムの実行は上記制約を充足するエンティティとアトリビュート値を求めることを意味する．すなわち，制約を充足するエンティティタイプを $\boldsymbol{E}_{solved}$ と

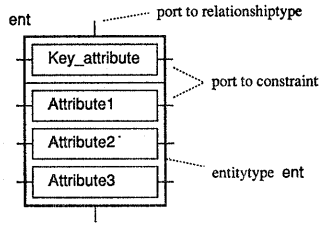


図 2 エンティティタイプの図式表記
Fig. 2 Graphic symbol of entitytype.

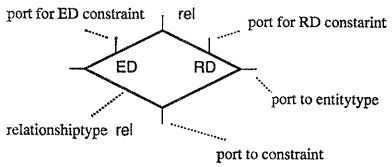


図 3 リレーションシップタイプの図式表記
Fig. 3 Graphic symbol of relationshiptype.

したとき，実行結果は，

$$E_{solved\,0}, E_{solved\,1}, \cdots, E_{solved\,n}$$

ただし，$E_{solved\,i}$ は，$E_{init\,i}$ の未定アトリビュートに値を代入，あるいは，新たなエンティティの追加によって得られるエンティティタイプである．実行の過程で，制約を充足するのに必要なリレーションシップやエンティティが適宜生成される．

3.2 表 記 法

PSDL-GR におけるエンティティタイプ，リレーションシップタイプ，制約は図式表現で記述される．**図 2～図 4** にこれらの表記法を示す．プログラムはこれらの構成要素を結線することで定義される．

3.3 プログラム例

図 5 に示した事務処理計算のプログラム例を用いて言語の概要を説明する．プログラミングの手順としては，E-R モデルを用いて対象世界をモデル化することから開始する．この例では，対象世界は製品，売上，顧客，請求およびその間に定義された関係，売り，購入，請求先より成り立っている．これらはそれぞれ，エンティティタイプ product, sale, customer, claim およびリレーションシップタイプ sold, buy, claimed に相当する．

product と sale 間に定義された制約は，sold によって関係づけられたエンティティ上のアトリビュート値の間の以下の依存関係で示す C_{AD} である．

Amount＝Quantity＊Price

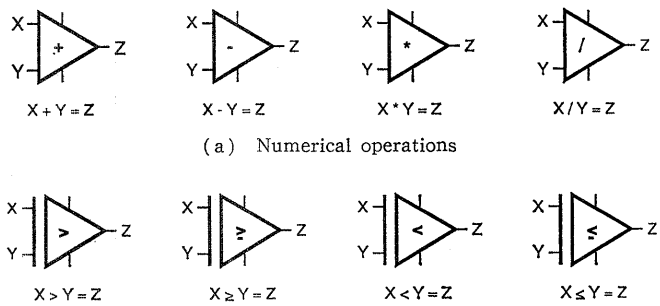
sale と customer 間に定義されている制約も C_{AD} であるが，customer の一つのエンティティに対して sale の複数のエンティティが対応している集合関数である（集合関数は，C_{AD} の種類によって識別される）．これは以下の関係を表している．

$$\text{Total}=\sum_i Amount_i$$

ただし，$Amount_i$ は Total を含む各々のエンティティに対し，そのエンティティとリレーションシップで関係づけられた複数のエンティティに含まれる Amount の一つを表す．customer と claimed 間に定義された制約は C_{ED} である．以下の条件が満たされた場合に，被参照エンティティに対応して claim 内に新たなエンティティが生成される．

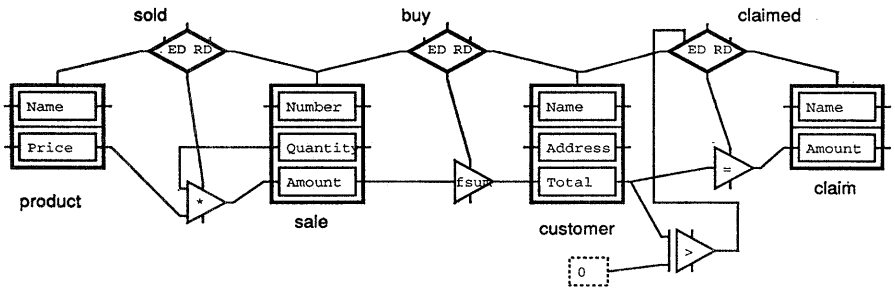
Total＞0

図 5(b) はエンティティおよびリレーションシップの初期インスタンスで，'_' は未定値を表す．図 5(c) は制約を解消した結果求まったエンティティである．



(a) Numerical operations
(b) Numerical relations
(c) Identity relation
(d) Boolean operation
(e) Aggregation functions

図 4 制約の図式表記
Fig. 4 Graphic symbols of constraint.



(a) E-R Model with constraint

sold

sale	product
101	camera
102	stereo
103	videotape
104	bag
105	watch

buy

customer	sale
hanako	101
hanako	102
yuki	103
yuki	104
natsuko	105

claimed

claim	customer

product

Name	Price
camera	80000
stereo	123400
videotape	660
bag	2200
watch	45670

sale

Number	Quantity	Amount
103	3	_
102	2	_
105	5	_
101	1	_
104	4	_

customer

Name	Address	Total
hanako	tokyo	_
yuki	yokohama	_
natsuko	nara	_
alison	wonderland	_

claim

Name	Amount

(b) Initial instances

product

Name	Price
camera	80000
stereo	123400
videotape	660
bag	2200
watch	45670

sale

Number	Quantity	Amount
103	3	1980
102	2	246800
105	5	228350
101	1	80000
104	4	8800

customer

Name	Address	Total
hanako	tokyo	326800
yuki	yokohama	10780
natsuko	nara	228350
alison	wonderland	0

claim

Name	Amount
hanako	326800
yuki	10780
natsuko	228350

(c) Solved instances

図 5 PSDL-GR の例
Fig. 5 Example of PSDL-GR.

4. 制約解消の実現法

4.1 E-R モデル上での制約伝搬

図5の例では，初期インスタンスとして product の Name, Price と sale の Number, Quantity が与えられており，Price, Quantity から Amount への計算が実行されている．しかし制約解消という意味では，未定変数は，

Amount＝Quantity＊Price

の等式のいずれのアトリビュートであっても構わない．例えば，Amount, Quantity が先に決定すれば，Price を求める計算が起動される．このようにエンティティを1対1に対応づけるリレーションシップ上の C_{AD} では制約の伝搬は双方向に進行する．また制約は三つ以上のエンティティに跨って伝搬する場合もある．

C_{AD} の伝搬と C_{RD} の伝搬は相互に関連している．**図6**にこの様子を示す．破線で表されたアトリビュートおよびリレーションシップは初期状態で未定である．同図の制約を評価する場合，評価の順序は任意で良いわけではない．C_{RD} 3は，アトリビュート値 A1 および A2 が決定した後に評価可能となり，C_{AD} 4

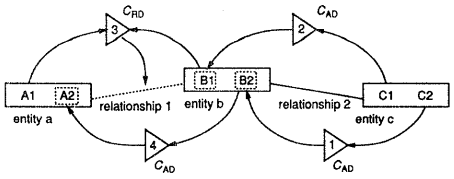


図 6 制約伝搬
Fig. 6 Constraint propagation.

は，relationship 1 が存在し，アトリビュート値 A 2 ないしは B 2 が決定した後に評価可能となる．このように各制約の評価に際しては，必要な変数が決定した後初めて実際の評価が行われる遅延評価の機構が必要である．同図の例では，1，2，3，4 ないしは，2，1，3，4 の順で評価が行われなければならない．

C_{AD} が集合関数の場合は，一つのエンティティに対し不特定多数のエンティティが対応するため逆方向の制約伝搬には意味がない．したがって，伝搬方向は単方向に制限する．C_{ED} も同様に考える．

4.2 制約論理プログラミング

前節で述べたように，制約を解くためには各種の制約に基づく計算順序を適切に決定することが必要である．一つの方法として，すべてのアトリビュートについてその値の代入状況を監視し，各状態において次に実行すべき適切な制約を順序決定していく方法が考えられるが，容易に推察されるようにオーバヘッドが大きく現実的ではない．

ここでは，PSDL-GR を制約論理プログラムに変換し実行する方法を用いた．制約論理プログラムに変換することにより PSDL-GR の記述法に若干の制限が加わるが，一方で，制約論理プログラムの強力な制約解消機構を利用できる，PSDL-GR の意味を制約論理によって明確化できる，という利点がある．

制約論理プログラミングは，通常のホーン節に制約記述を可能にした拡張ホーン節（以降，これを単にホーン節と呼ぶ）をプログラムの基本単位としている．

$$A_0 \ :- \ A_1, A_2, \cdots, A_n, c_1, c_2, \cdots, c_m.$$

ただし，A_i は原子論理式，c_i は素制約を表す．

制約論理プログラミングの実行は，論理プログラミングと同様にサブゴールを再帰的に後向きに探索する．探索の過程で変数のユニフィケーションが行われるのも同様であるが，この際，変数と制約を連立方程式とみなし，可解であればゴールが成功すると定義する点が特徴である．また，この連立方程式の解は，ユニフィケーションにより必要な変数が決定された時点で評価される．すなわち，前節で述べた遅延評価の機構が実現される．

5. 制約論理プログラムへの変換

5.1 2 項 関 係

まず基本的な関係として，**図 7** に示すような二つのエンティティタイプが一つのリレーションシップタイプにより結合されている場合（2項関係）を考える．

エンティティやリレーションシップはそれぞれホーン節のファクトに写像する．例えば，

```
e("TV", 80000).
e("CD", 3000).
    ⋮
r("TV", 2).
r("CD", 1).
    ⋮
f(1, 10, _).
f(2, 2, _).
    ⋮
```

等となる．第1項はキーアトリビュートに，無名変数 '_' は未定アトリビュートに対応している．

e，f 間のリレーションシップは特に指定されなければこれらの直積で与えられ，インスタンスや C_{RD} があればそれらの積集合である．よって，リレーションシップで対応づけられるエンティティの組は，

$$\text{e\_r\_pair}(Xk, X_1, \cdots, X_n, Yk, Y_1, \cdots, Y_m) :-$$
$$\quad e(Xk, X_1, \cdots, X_n), f(Yk, Y_1, \cdots, Y_m),$$
$$\quad r'(Xk, Yk), C_{RD}.$$

ただし，$Xk, X_1, \cdots, X_n$ は E のアトリビュート，$Yk, Y_1, \cdots, Y_m$ は F のアトリビュート，r' はリレーションシップのインスタンスが与えられている場合は r を，与えられていない場合は true とする．

C_{AD} は e_r_pair 上のアトリビュートの依存関係として定義される．さらに C_{ED} を e から f へ伝搬する

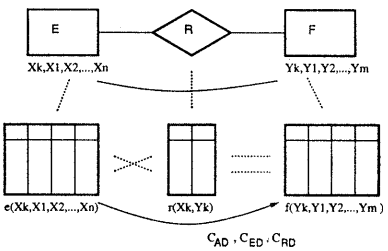


図 7 2項リレーションシップ
Fig. 7 Binomial relationship.

ものと仮定すれば2項関係は以下の節で表せる．

e_r($Xk, X_1, \cdots, X_n, Yk, Y_1, \cdots, Y_m$):—
e($Xk, X_1, \cdots, X_n$),
f_r($Xk, X_1, \cdots, X_n, Yk, Y_1, \cdots, Y_m$),
C_{AD}.

f_r($Xk, X_1, \cdots, X_n, Yk, Y_1, \cdots, Y_m$):—
C_{ED}.

f_r($Xk, X_1, \cdots, X_n, Yk, Y_1, \cdots, Y_m$):—
f($Yk, Y_1, \cdots, Y_m$),
r'(Xk, Yk), C_{RD}.

以下は，制約の例である．

C_{ED}: $X_1 > 0$

C_{RD}: $Xk = Yk$

C_{AD}: $Y_2 = X_2 * Y_1$

また，プログラムの実行は以下のゴールを与えることにより起動される．

?- e_r($Xk, X_1, \cdots X_n, Yk, Y_1, \cdots, Y_m$).

リレーションシップが三つ以上のエンティティタイプ間に定義されている場合(多項リレーションシップ)も，同様の方法で変換可能である．

5.2 集合関数

集合関数の概念は述語論理での高階の概念に相当する．これを実現するため以下のような高階述語を導入する．

aggregate(*Goal*, *Var*, *Agg_func*, *Ret*).

この述語は，*Goal* を証明するすべての *Var* の集合を引数として *Agg_func* を計算し結果を *Ret* とする，これを用いて，集合関数を含む2項関係を以下のように変換する．

e_r($Xk, X_1, \cdots, X_n, Yk, Y_1, \cdots, Y_m$):—
f($Yk, Y_1, \cdots, Y_m$),
aggregate(
c($Xk, X_1, \cdots, X_n, Yk, Y_1, \cdots, Y_m$),
[$X_i, \cdots, X_j$],
func,Y_i).

c($Xk, X_1, \cdots, X_n, Yk, Y_1, \cdots, Y_m$):—
e($Xk, X_1, \cdots, X_n$),

ただし，C_{ED} や他の C_{AD} は含まれないものとする．

5.3 素 E-R の組み合わせ

前節までに述べた E-R モデルは，一つのリレーションシップタイプに1ないし複数のエンティティタイプが結合した形態である．以降これを**素 E-R (Primitive E-R)** と呼び，*ER* と記述する．プログラムは素 E-R の組み合わせによって構成される．例えば，

図8の例では，プログラムは三つの素 E-R：ER_1, ER_2, ER_3 から成っている．カージナリティが1対1の場合を考えると，素 E-R の結合に従って C_{RD} や C_{AD} は伝搬するため複数の素 E-R に跨る制約を解く必要がある．これは，以下のように，ER_1, ER_2 のアトリビュートを ER_3 に継承することで実現する．

e_r3($Wk, W_1, \cdots, W_n, Xk, X_1, \cdots, X_n$,
$Yk, Y_1, \cdots, Y_n, Zk, Z_1, \cdots, Z_l$):—
⋮
new_e3($Wk, W_1, \cdots, W_n, Xk, X_1, \cdots, X_n$,
$Yk, Y_1, \cdots, Y_m$),
f_r($Yk, Y_1, \cdots, Y_m, Zk, Z_1, \cdots, Z_l$),
⋮
new_e3($Wk, W_1, \cdots, W_n, Xk, X_1, \cdots, X_n$,
$Yk, Y_1, \cdots, Y_m$):—
e_r1($Wk, W_1, \cdots, W_n, Yk, Y_1, \cdots, Y_m$).
new_e3($Wk, W_1, \cdots, W_n, Xk, X_1, \cdots, X_n$,
$Yk, Y_1, \cdots, Y_m$):—
e_r3($Xk, X_1, \cdots, X_n, Yk, Y_1, \cdots, Y_m$).

ただし，$Wk, W_1, \cdots, W_n$ は $E1$ のアトリビュート，$Xk, X_1, \cdots, X_n$ は $E2$ のアトリビュート，$Yk, Y_1, \cdots, Y_m$ は $E3$ のアトリビュート，$Zk, Z_1, \cdots, Z_l$ は $E4$ のアトリビュートである．ゴール ?-e_r3(…). を与えることにより，解 e_1, e_3, e_4 ないし e_2, e_3, e_4 が導出される．

さらに，より一般的なケースを考えると，E-R ダイアグラムは**図9**のようなネットワーク構造をなす．これをどのような素 E-R の結合とみなすかを一意的に定めることは困難である．そこで，素 E-R 間やエンティティタイプ・リレーションシップタイプ間に親子関係を導入する．リレーションシップタイプ上の→が上位(親)の方向を示すと定義すると，PSDL 図は→の方向に沿った辺を上昇辺，反する方向の辺を下降辺，エンティティタイプ・リレーションシップタイプを頂

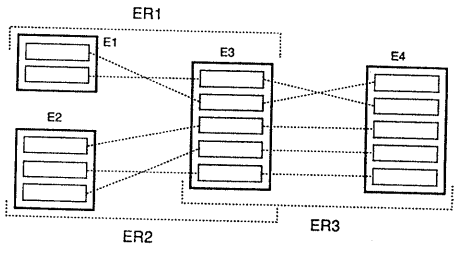


図 8 素 E-R の複合
Fig. 8 Composite of primitive E-R.

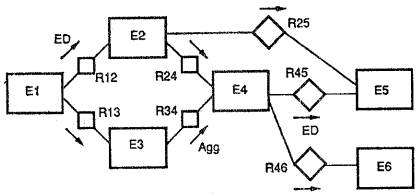


図 9 PSDL-GR のネットワーク構造
Fig. 9 PSDL-GR network structure.

点とする有効グラフとみなすことができる.

この親子関係は,以下に述べるように制約論理プログラムに変換する際のホーン節の入れ子関係に対応している.したがって,リレーションシップは複数の親子関係に沿って存在しうるが,兄弟間に跨ることはない.図9の例では,E1, E2, E5 内のエンティティがリレーションシップで結ばれることはありうるが,E2, E5, E4 間にはそのような関連は存在できない.ユーザは,上記の性質と以下の制限を考慮した上で,リレーションシップタイプ上の→を適当に定義する.

1. →の方向は集合関数および C_{ED} の方向と一致する.

2. 有効グラフ上に閉ループが存在しない.

制限 1. は,集合関数や C_{ED} の制約伝搬が単方向であることに対応している.制限 2. は,親子関係を一意に定めるための措置である.

システムは,親を持たないすべてのエンティティタイプの共通の親として,仮想的な頂点 R_0 を付加する.このとき,図9は**図 10** に示すような親子関係の木として表現できる.これを根から探索して各素 E-R を変換する.頂点間の親子関係に従って,エンティティタイプ間,リレーションシップ間,および素 E-R 間にも親子関係を定義する(例えば,R_{24} と R_{34} は R_{45} の子である)と,親の素 E-R と子の素 E-R の接続は,集合関数や C_{ED} を含まない場合では以下のように実現される.

$e\_r_i(\chi_{ER}(ER_i)) :-$
　$new\_e_j(\chi_E(E_j))$,
　$new\_e_k(\chi_E(E_k))$,
　　⋮
　$f\_r(\chi_{local}(ER_i))$,
　　⋮

ここで,各 new_e に対し,

$new\_e_j(\chi_E(E_j)) :-$
　$e\_r_j(\chi_{ER}(ER_j))$.
$new\_e_j(\chi_E(E_k)) :-$
　$e\_r_k(\chi_{ER}(ER_k))$.
　　⋮

ただし,$E_j, E_k, \cdots$ は ER_i のリレーションシップ R_i と下降辺でつながったエンティティタイプ,ER_j, $ER_k, \cdots$ は ER_i の子の素 E-R である.また,χ_{ER} は変数のリストを表し,以下のように帰納的に定義される.

1. もし ER_i が子を持たないなら,

$$\chi_{ER}(ER_i)=\chi_{local}(ER_i)$$

ただし,

$$\chi_{local}(ER_i)=\{E \text{ のアトリビュート} \mid E \text{ は } ER_i \text{ に含まれる}\}$$

2. もし ER が子を持つなら,

$$\chi_{ER}(ER_i)=\chi_{local}(ER_i) \cup \bigcup_{er \in \varepsilon R} \chi_{ER}(er)$$

$$\varepsilon R=\{ER_j \mid ER_j \text{ は } ER_i \text{ の子である}\},$$

さらに $\chi_E(E_i)$ は,

$$\chi_E(E_i)=\bigcup_{er \in \varepsilon R} \chi_{ER}(er)$$

$$\varepsilon R=\{ER_j \mid ER_j \text{ は } E_i \text{ の子である}\}$$

素 E-R が集合関数や C_{ED} を含む場合は,その部分で制約伝搬が途切れるため,特別な考慮が必要である.この素 E-R を ER_a とすると,この部分は以下のように変換を行う.

1. ER_a のリレーションシップがないものとし分割された部分木に対しては,上述の手順で変換を行う.

2. ER_a については,最初に下位の部分木を実行す

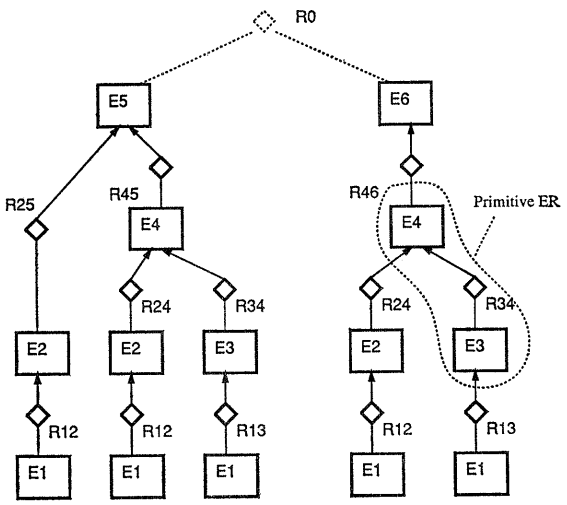


図 10 PSDL-GR の木構造
Fig. 10 PSDL-GR search tree.

るための節を生成し，次に ER_a 自身に対応する節を生成する．

3. ER_a の上位の ER の変換の際に，ER_a を実行するための節を挿入する．

6. プログラミング環境

ワークステーション上に，PSDL-GR の視覚的プログラミング環境のプロトタイプを作成した．インタプリタに PrologIII を，ユーザインタフェースにXウィンドウの Motif ライブラリを使用し SUN 4/SS 2 上に実現している．

図 11 にプログラミングの様子を示す．プログラミングは，1) エンティティタイプおよびリレーションシップタイプを定義する，2) これらの間の制約を定義する，3) エンティティおよびリレーションシップの初期インスタンスを与える，の手順で行われる．エンティティタイプ，リレーションシップタイプ，制約は，それぞれマウスによってメニューの中から選択され，画面上に配置され，結線される．アトリビュートやカージナリティは，各エンティティタイプやリレーションシップタイプに対応したサブウィンドウ内で定義される．エンティティやリレーションシップの内容を表示するためのサブウィンドウ（インスタンスウィンドウ）も用意されている．これはインスタンスの入力用を兼ねており，ユーザは，これを利用して初期インスタンスを入力する．

図 12 に実行時の様子を示す．RUN ボタンを選択すると，画面上で定義されたプログラムがインタプリタへ送られ，上述の変換操作の後，実行される．結果はインスタンスウィンドウ上に表示される．

7. 考　　察

PSDL-GR のプロトタイプを使用した主観的な評価を述べる．システムの設計段階で予想されたように，プログラムの理解性の向上と修正の容易さでは従来の言語にない利点が得られた．一般に，宣言的なプログ

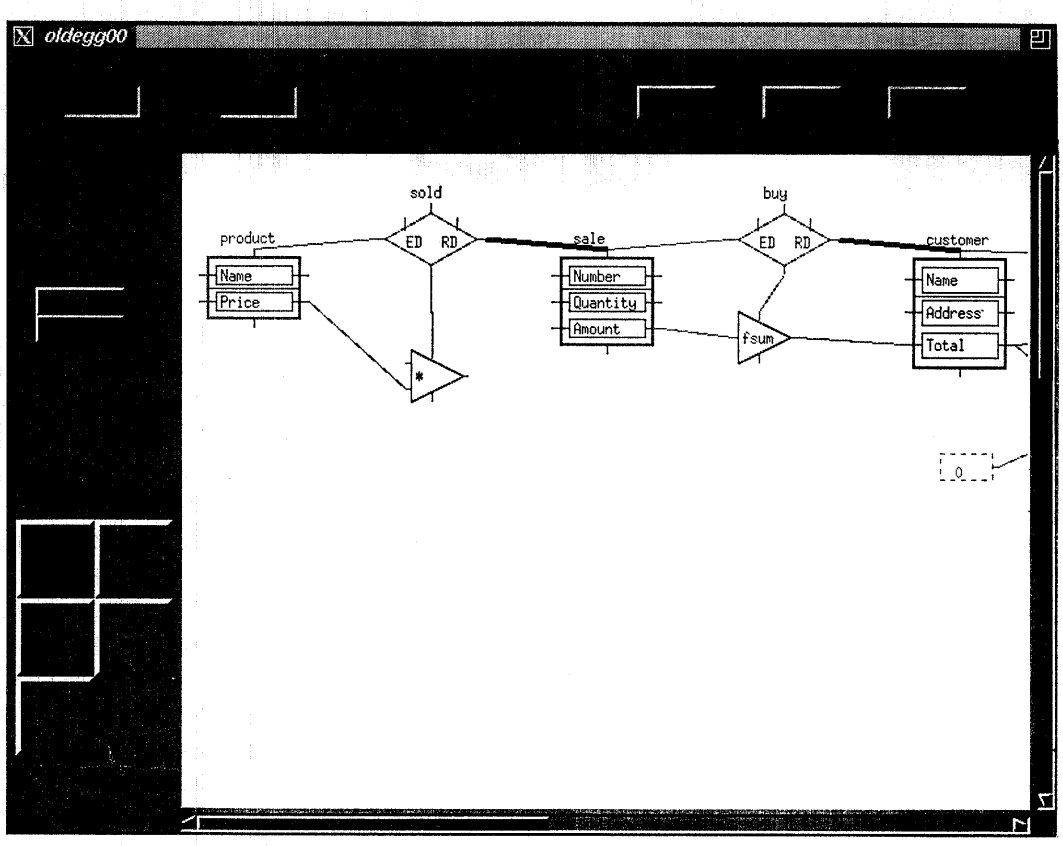


図 11 プログラミング環境（編集時）
Fig. 11 Programming environment (editing).

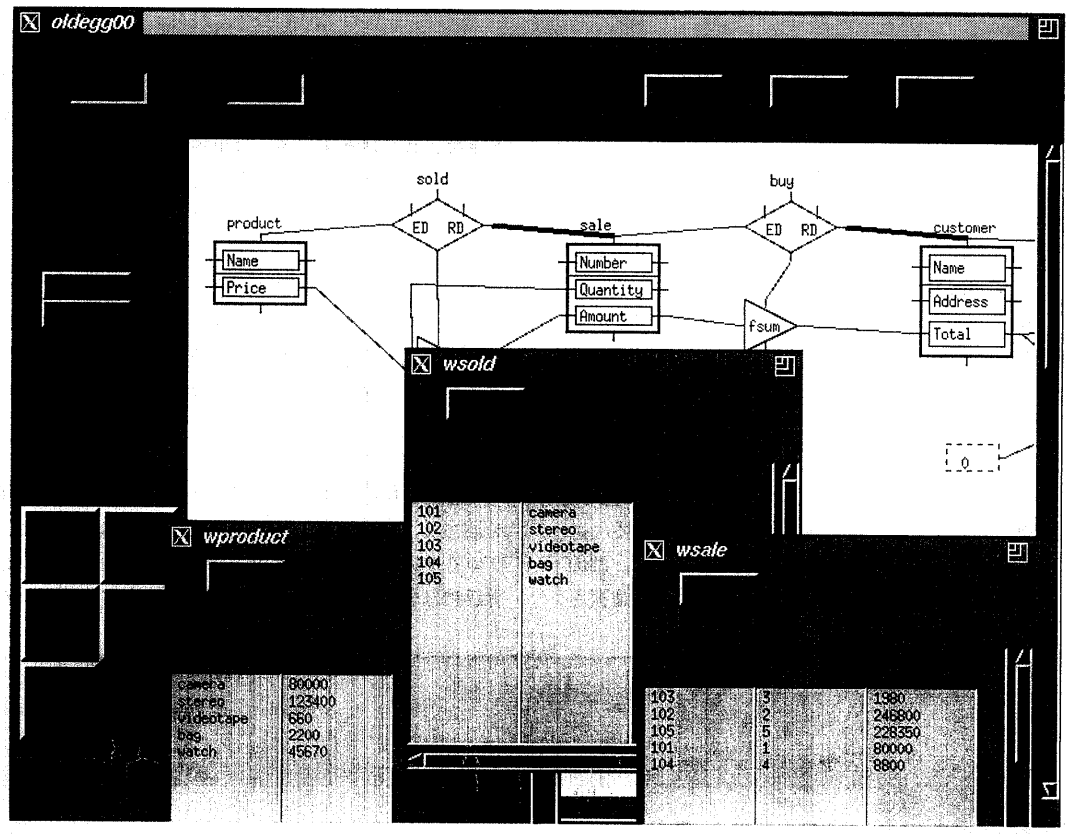


図 12 プログラミング環境（実行時）
Fig. 12 Programming environment (execution).

ラムでは，プログラムが断片的な記述の集合によって定義されるため個々の意味は明確であるが，全体的な関係を把握しづらい問題点があった．PSDL-GR における 2 次元的な表現は制約やデータ間のつながりを明確化するのに役だった．また，従来の手続き型プログラミングでは，局所的なプログラムの修正がその他の部分に及ぼす影響を把握することが必ずしも容易ではなかった．PSDL-GR では，処理が断片的な制約によって記述されていること，および制約間の関係が直観的に理解しやすいため，修正が及ぼす影響範囲が掴みやすい．

一方で，プログラムの記述性については，ユーザインタフェースに依存する部分が多く有効性の判断がつきかねた．特に，構成要素を結線する際に，折れ線の経路を指定する作業や，部品間のスペースを適当に空ける作業は設計者に負担が重いように感じられた．設計者が論理的な関係にのみ集中できるよう，部品の自動配置や整形など，ユーザインタフェースの高機能化が必要であろう．

本方式では，制約論理型言語に変換するために設けたいくつかの制限事項によりプログラミングの自由度が制限されている．第一に，集合関数と通常の C_{AD} が同一リレーションシップタイプに混在する場合，集合関数を含むリレーションシップタイプとみなされ，子にあたる部分木の制約を先に解消する制御を行う．したがって，通常の C_{AD} も部分木の中で解消可能な制約として与えられなければならない．第二に，集合関数や C_{ED} が同一リレーションシップに双方向に存在する記述は許されない．このような処理を行わせようとした場合には，中間的なエンティティタイプやアトリビュートを設ける必要がある．処理対象によっては，このような処理系の特性を考慮した E-R モデルや制約の定義が繁雑になる場合があった．

制約論理型言語による PSDL-GR の実現は，従来の論理プログラムによるリレーショナルデータベースの実現法（演繹データベース）の拡張になっている．

例えば，素 E-R のリレーションシップタイプを生成する部分は，関係代数演算の直積と選択の組み合わせとみなすことができる．拡張された部分は，E-R モデルへの対応と制約の処理である．また，演繹データベースが主としてデータの検索を目的としているのに対し，PSDL-GR ではデータの計算を目的としているのも相違点である．

8. おわりに

本稿では，E-R モデル上に定義された制約を解く言語 PSDL-GR の提案と，その実現法，および PSDL-GR のプログラミング環境の紹介を行った．

PSDL-GR は，E-R モデルをプログラミングに適用するための一手法であり，制約指向の処理方式を組み合わせることによりプログラムの図式表現を可能としている．これによって，従来の，アイコニックプログラミング言語と表指向プログラミング言語の特徴を合わせ持つ言語を実現した．

本論文で述べた方式によって，PSDL-GR は制約論理型言語で解釈可能であり，カージナリティが1対1の範囲であれば，制約の双方向伝搬が実現できる．このことは，同種の処理に対してデータの入出力方向に依存せず同じプログラムが適用できることを意味する．さらに，プログラムの変更が制約の追加・削除によって行われるため，プログラムの修正が局所的な範囲に留まる．これらから，プログラムの再利用性の向上が期待できる．

PSDL-GR で取り扱える問題の範囲は，現在のところ，図4で示した制約の組み合わせで定義可能な範囲に限られている．再帰処理を含まない，ほとんどの事務処理系の計算に適応可能である．

今後の課題としては，より制限の少ない制約記述を可能とする処理方式，再帰的な処理の実現法，高機能なユーザインタフェース，定量的なシステムの評価，等があげられる．

謝辞 本研究の機会を与えていただいた葉原耕平会長に感謝いたします．貴重なご意見・ご討論に参加いただいた竹中豊文前室長(現 NTT)，太田理室長，および ATR 通信システム研究所の諸氏に感謝いたします．

参考文献

1) Shu, N. C.: Visual Programming: Perspective and Approaches, *IBM System Journal*, Vol. 28, No. 4, pp. 525-547 (1989).
2) Borning, A.: The Programming Language Aspects of ThingLab, a Constraint-Oriented Simulation Laboratory, *ACM Trans. on Programming Languages and Systems*, Vol. 3, No. 4, pp. 353-387 (1981).
3) Borning, A.: Defining Constraint Graphically, *Proc. of CHI '86*, pp. 137-143 (1986).
4) Glinert, E. P. and Tanimoto, S. L.: Pict: An Interactive Graphical Programming Environment, *Computer*, Vol. 17, No. 11, pp. 7-25 (1984).
5) Vose, G. M. and Williams, G.: LabVIEW: Laboratory Virtual Instrument Engineering Workbench, *BYTE*, Vol. 11, No. 9, pp. 84-92 (1986).
6) Zloof, M. M.: QBE/OBE: A Language for Office and Buisness Automation, *Computer*, Vol. 14, No. 5, pp. 13-22 (1981).
7) Shu, N. C.: FORMAL: A Froms Oriented Visual Directed Application Development System, *Computer*, Vol. 18, No. 8, pp. 38-49 (1985).
8) Chen, P. P.: The Entity-Relationship Model—Toward a Unified View of Data, *ACM Trans. Database Syst.*, Vol. 1, No. 1, pp. 9-36 (1976).
9) Steel, G. L. Jr.: The Definition and Implementation of a Computer Programming Language Based on Constraints, *AI-TR-595* (1981).
10) 橋本正明：プログラム仕様記述のための計算指向 EAR モデル，情報処理学会論文誌，Vol. 27, No. 3, pp. 330-338 (1986).
11) Okamoto, K. and Hashimoto, M.: On Real-Time Software Specification Description with a Conceptual Data Model-Based Language, *Proc. of ICCI '90*, pp. 186-190 (1990).
12) Hashimoto, M. and Okamoto, K.: A Set and Mapping-based Detection and Solution Method for Structure Clash between Program Input and Output Data, *Proc. of COMPSAC '90* (1990).
13) Okamoto, K. and Hashimoto, M.: Program Generation from Real-Time Software Specification Described with a Conceptual Data Model-based Language, *Proc. of SEKE '91*, pp. 261-270 (1991).
14) 秋口忠三：ER モデルに基づく図式問い合わせ言語とその実現法，第28回情報処理学会プログラミングシンポジウム予稿集，pp. 189-199 (1987).
15) Angelaccio, M., Catarci, T. and Santucci, G.: QBD: A Graphical Query Language with Recursion, *IEEE Trans. Software Eng.*, Vol. 16, No. 10, pp. 1150-1163 (1990).
16) Colmerauer, A.: Equations and Inequalities on Finite and Infinite Trees, *Proc. of the International Conference on Fifth Generation*

Computer Systems 1984 (1984).
17) Jaffar, J. and Lassez, J-L.: Constraint Logic Programming, Technical Report, IBM Thomas J. Watson Research Center (1986).
18) Jaffar, J. and Michaylov, S.: Methodology and Implementation of Constraint Logic Programming System, Technical Report, Computer Science Department, Monash University (1986).
19) Colmerauer, A.: Introduction to Prolog III, *Proc. of the 4th Annual ESPRIT Conference*, Brussels, North Holland (1987).
20) 溝口文雄，古川康一，Lassez, J.-L.（編）：制約論理プログラミング，共立出版 (1989).
21) Sato, M. and Hashimoto, M.: A Constraint Satisfaction System on the Entity and Relationship Model, *Proc. of SEKE '92* (1992).
22) Battista, G. D., Kangassalo, H. and Tamassia, R.: Definition Libraries for Conceptual Modeling, *Data and Knowledge Engineering 4*, pp. 245-260 (1989).
23) 岡本克己，橋本正明：制約指向の概念モデルを用いた高次部品化によるプログラム合成，電子情報通信学会ソフトウェアサイエンス研究会報告，SS 89-18 (1989).

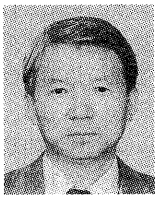
(平成4年6月15日受付)
(平成5年10月14日採録)

佐藤 正和

昭和34年生．昭和57年早稲田大学理工学部電子通信学科卒業．昭和57年日本電気株式会社入社．昭和59年早稲田大学理工学研究科博士課程入学．昭和61年同前期（修士）修了．平成元年同後期（博士）修了．昭和61年9月より平成2年3月まで早稲田大学情報科学研究教育センター助手．この間，画像符号化，文書画像処理，学内LAN等の研究および実用化に従事．平成2年ATR通信システム研究所研究員．以来，ソフトウェアの自動生成，ソフトウェア仕様の検証等の研究に従事．工学博士．電子情報通信学会，ソフトウェア科学会，IEEE各会員．

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．



寺島 信義（正員）

昭和39年東北大学工学部通信卒業．同年電電公社（現NTT）入社．以来，OS，プログラミング言語，ネットワーク・アーキテクチャ，知識ベース，エキスパート・システム，自然言語の研究に従事．平成3年6月ATR通信システム研究所に勤務．臨場感通信会議システム，ソフトウェアの自動生成などの研究に従事．現在同研究所代表取締役社長．工学博士．早稲田大学客員教授．日本工学アカデミー広報委員会委員．電子情報通信学会評議員．