ソフトウエア工学　15－2
（1980. 9. 5）

デザイン展開の為の機構

東京大学　理学部　情報科学科　　　山口　和紀

近年、工学の場などで展開しながら設計していく方式が発展して来つつある。その1つの例としてはUCLAのEstrinによるSARA[1]が挙げられる。SARAでは"モジュール"，"ソケット"，"インター・コネクション"の3つの要素があり、それらの組み合せで目的のシステムを記述する。その記述に使われたモジュールは再びモジュール，ソケット，インター・コネクションの3つの要素の組み合せで記述する。この様にモジュールを展開しながら、設計していこうというのがSARAの方法である。ここでSARAのモジュール，インター・コネクションをノード，アークと見て，グラフ理論的に発展、統一したものが再帰グラフ[2]である。再帰グラフではグラフの2つの要素であるところのノードとアークが再びグラフである事を許す様なグラフである。この再帰グラフで設計する場合も、まず目的のシステムをグラフで表わし、次にそのグラフの要素のノードやアークを再びグラフに展開する、という過程を繰り返す事により設計を進めていく。このSARAも再帰グラフも大規模なシステムの設計に有効な手段である。

この両者で問題となるのは、システムを記述する手段としてモジュールなどを用いるか、ノードなどを用いるかには任意性があり、又展開の機構とは独立している点である。モジュールとして記述するか、ノードとして記述するかは、むしろその展開の機構の使い方にすぎない。この様な発想の下に展開の機構を定式化したものがデザイン・エンジンである。デザイン・エンジンでは展開の機構が純粋に定式化されている為に，展開の機構を利用する様なシステムの共通の基礎となる事が出来る。

デザイン・エンジンでは展開していくデーターを表現し、操作するという性格上、必然的にデーターベースと深く関連している。つまり、展開を含めたデーターを表現するデーターベースと考えると、データー・モデルの拡張の形にとらえる事が出来る。この様なデーター・モデルを基礎に持つデーターベース・マシンができれば展開型デザインなどに有効であろう。データーベース・マシンの様な専用マシンの考え方が、このデザイン・エンジンという独特な名称の由来である。

特にデザイン・エンジンの利点として挙げられる事としては、構造型表などの事務書類に多く表われる表型式の基礎として使用できる点が挙げられる。従って電子回路の設計の為に使用されると同時に、それに使われる電子部品の管理に使用する事が出来、両者が同一の機構の上で実現出来るので，データーが共有できる利点がある。

《デザイン・エンジンの定式化》

デザイン・エンジンの展開されるデーターのデーター・モデルとしてはコッドの提案した関係型式[3]を利用した。関係型式は最も良く定式化されたモデルの1つであり、それ自体に特別な構造が無い為に扱い易い。展開を定式化したものとしては写像を利用した。関係（リレイション）には行と列があるのに対応して2種類の写像が定義される。元の関係を$T \subseteq D_1 \times D_2 \times \ldots \times D_n$とする。但し$D_i$はi番目の欄の値域とする。この時行方向の展開を表わすものとして行構造写像$f_T : T \to 2^T$,

列方向の展開を表わすものとして列構造写像 $g_T: A \to 2^A$ とする。(A は関係Tの属性(アトリビュート)の集合である。) ここで関係Tのtuple t が行方向に展開するとtupleの集合 $f_T(t)$ になり、属性 a が列方向に展開されると属性の集合 $g_T(a)$ になるという方式で展開を f_T と g_T で表わす。ここで行構造写像や列構造写像は1つの関係に対し複数個定義しても良いことにする。言葉として、f_T や g_T に対しTを基礎表, $t' \in f_T(t)$ の時 t' を t の子, t を t' の親と呼ぶ事にする。この様な呼び方は g_T に対しても同様に適用する。この定義では f_T や g_T に対し何の制限も無いが、一般には"展開"の性格から f_T や g_T は木構造か半順序構造を持つ場合が多い。但し以下の定義には、この様な制限は必要ない。

以上で関係型式の拡張をしたので、その拡張に応じて関係代数の拡張が必要となる。この拡張により展開を含めた操作が可能となる。ここでは関係代数として selection and projection, union, difference, extended cartesian product について定義する。他の演算はこれらの演算の組み合せで定義できるので(例えば関係AとBのintersection $A \cap B$ は difference により $A-(A-B)$ で実現できる。)、ここでは定義していない。

〈Selection and projection〉

Selection と projection は非常に似た演算なので1つの演算にまとめた。この演算ではもとの関係から必要な行と列をぬき出して関係を作る演算である。ここで、演算の対象となる関係をTとする。このTからどの欄をとり出すか(projection)を指定するする為に単射 $s: \{1, \ldots, k\} \to \{1, \ldots, n\}$ を使う。ここでprojectionした結果の i 番目の欄はもとの関係の $s(i)$ 番目の欄を取って来ることにする。次にselectionを規定する為に述語 $P(t)$ を用いる。この時selectionしてprojectionした結果の関係 T' は

$$T' = \{(t_{s(1)}, t_{s(2)}, \ldots, t_{s(k)}) \mid (t_1, t_2, \ldots, t_n) \in T \wedge P((t_1, t_2, \ldots, t_n))\}$$

と定義できる。これは関係代数のselectionとprojectionを組み合せた式である。次に行構造写像 f_T がこの演算の結果 $f_{T'}$ となるが、この $f_{T'}$ を f_T からどの様に作るかを指定する為に述語 $Q(f_T, T, T', t, t')$ を用いる。この述語を用いて T' のtuple t に対し

$$f_{T'}(t) = \{t' \mid Q(f_T, T, T', t, t') \wedge t' \in T'\}$$

と定義する。この $Q(f_T, T, T', t, t')$ のごく普通の使い方としては $Q(f_T, T, T', t, t') \equiv t' \in f_T(t)$ と定義するものがある。この様に定義すると結果の $f_{T'}$ は f_T の T' への制限になる。亦、$Q(f_T, T, T', t, t') \equiv (t' \in f_T(t)) \vee (t = t_0 \wedge t' = t_0')$ と指定すれば $f_{T'}$ は f_T の T' への制限であるが、それに加え t_0 が t_0' の親になるという変更が加わる。こうして、述語Qの定義の仕方によって、f_T に色々な変更を加えて $f_{T'}$ を作る事が出来る。列構造写像 g_T も述語 $R(g_T, A, A', a, a')$ を用いて次の様に定義される。 $a \in A'$ に対して

$$g_{T'}(a) = \{a' \mid R(g_T, A, A', a, a') \wedge a' \in A'\}$$ 但し A, A' は各々 T, T' の属性の集合である。

この場合も R の定義の仕方によって g_T に色々な変更を加えて $g_{T'}$ を作る事が出来る。

次に union, difference, extended cartesian product を定義する。これらの演算は2つの関係 T, T' から結果の関係 T'' を作り出す為のものである。

〈union〉

関係に関しては通常のunionと同じである。

$$T'' = T \cup T'$$

又 $f_{T''}$ については原則的に f_T と $f_{T'}$ のunionになるがそれぞれの写像の定義域が異なるので場合分けする。$t \in T''$ に対し

$$f_{T''}(t)=\begin{cases} f_T(t) & t\in T-T' \text{ の時} \\ f_{T'}(t) & t\in T'-T \text{ の時} \\ f_T(t)\cup f_{T'}(t) & t\in T\cap T' \text{ の時} \end{cases}$$

と定義出来る。次に$g_{T''}$に対しては、g_Tと$g_{T'}$が一致しているべきであるという条件をつける。もし一致していない場合、unionを作る為には、まずselection and projectionで説明した様にg_Tと$g_{T'}$を変更して一致させる。

$g_{T''}=g_T=g_{T'}$

と$g_{T''}$は定義する。(右側の"="は代入ではなく等号が成り立っていなければいけないという条件を示している。)

<difference>

unionと同様に定義される。関係に関しては

$T''=T-T'$

と定義する。$f_{T''}$に関しては$t\in T''$に対して

$f_{T''}(t)=\{t' \mid t'\in f_T(t)\wedge t'\in T''\}$

と定義する。これはf_TのT''への制限と一致している。g_Tと$g_{T'}$に関してはunionと同じ条件をつける。すなわち

$g_{T''}=g_T=g_{T'}$

と定義する。

<extended cartesian product>

関係に関しては

$T''=T\times T'$

と定義する。厳密には$T\subseteq D_1\times D_2\times\ldots\times D_n$, $T'\subseteq D_1'\times D_2'\times\ldots\times D_{n'}'$とした時$T''\subseteq D_1\times D_2\times\ldots\times D_n\times D_1'\times D_2'\times\ldots\times D_{n'}'$で$T''=\{(t_1,t_2,\ldots,t_n,t_1',t_2',\ldots,t_{n'}') \mid (t_1,t_2,\ldots,t_n)\in T\wedge(t_1',t_2',\ldots,t_{n'}')\in T'\}$と定義されるが省略して上記の様に書く。ここでtuple $t=(t_1,t_2,\ldots,t_n)$と$t'=(t_1',t_2',\ldots,t_{n'}')$に対し$t''=(t_1,t_2,\ldots,t_n,t_1',t_2',\ldots,t_{n'}')$を$t$と$t'$のconcatenationと呼び$t\cdot t'$で表わす。この時, $t\in T$, $t'\in T'$に対し

$f_{T''}(t\cdot t')=\{u\cdot u' \mid Q(f_T,f_{T'},t,t',u,u',T,T',T'')\wedge u\in T\wedge u'\in T'\}$

と定義する。ここで、f_Tと$f_{T'}$から$f_{T''}$をどの様に作るかを指定する為にQなる述語を利用している。例えばf_Tをそのまま残す場合には$Q(f_T,f_{T'},t,t',u,u',T,T',T'')\equiv u\in f_T(t)$と定義すれば良い。次に$g_{T''}$は$a\in A''$に対して

$$g_{T''}(a)=\begin{cases} g_T(a) & a\in A \text{ の時} \\ g_{T'}(a) & a\in A' \text{ の時} \end{cases}$$

と定義する。ここでA, A', A''は各々T, T', T''の属性の集合である。

以上で関係代数の拡張については終了した。この定義では基礎表に関する限り通常の関係代数と同じになる様にしてある。尚、通常の関係代数での演算間の関係、例えば関係TとT'のintersectionはdifferenceを使って$T-(T-T')$となる点などを保存する様に定式化してある。

次に写像の具体的な実現法などについての考え方を明確化する為に、関係型式の上にデザイン・エンジンを実現する方法について考える。但し、上の定義からも分かる様に、属性が値として使われたり、欄を指定するものとして利用できなければならない。

《実現法》

基礎表T, 行構造写像f_T, 列構造写像g_Tを各々、関係T, F, Gで表わす方法

を考える。ここではf_Tをtuple idの指定で定義できる様に基礎表にtuple idをつける。つまりTが$T \subseteq D_1 \times D_2 \times \dots \times D_n$なる関係の時、tuple idとなるdomain Nを1つ決めて$M \subseteq N \times D_1 \times D_2 \times \dots \times D_n$とする。この時MがTのtupleにtuple idをつけたものである事を保障する為に次の2つの条件をMに課す。

$$\exists m\,(m, t_1, t_2, \dots, t_n) \in M \Leftrightarrow (t_1, t_2, \dots, t_n) \in T \quad \cdots\cdots (1)$$

$$\forall (m, t_1, t_2, \dots, t_n), (m', t_1', t_2', \dots, t_n') \in M\ (m = m' \Leftrightarrow (t_1, t_2, \dots, t_n) = (t_1', t_2', \dots, t_n')) \quad \cdots\cdots (2)$$

(1)ではTのtupleはMに表わされ、MにはTのtupleを表わしたものしかない事を保障し、(2)ではtuple idが一意的にtupleを決定する事を保障している。数学的には、Tが与えられた時、条件(1),(2)を満すMが作れる事を証明しておくべきだが、データ・ベースではTが有限なのでMの作り方は明らかであろう。次にGとしてはTの属性の集合をAとした時$G \subseteq A \times A$なる関係で

$$(a, a') \in G \Leftrightarrow (a' \in g_T(a)) \quad \cdots\cdots (3)$$

を満すものとする。Fを定義するにはtuple idとtupleの間の関係が問題になるので2つの函数ρ_M, πを定義しておく。ρ_Mはtuple idからtupleを求める写像である。つまりρ_M: $N \to T$であり、

$$\rho_M(m) = t \Leftrightarrow ((m, t_1, t_2, \dots, t_n) \in M \wedge t = (t_1, t_2, \dots, t_n))$$

と定義する。ここでρ_MはMに依存するのでMを明記した。このρ_MはMの条件(2)により1対1写像なので逆写像が存在する。次にtuple idを取る写像πを定義する。πはπ: $M \to T$であり

$$\pi(t) = t' \Leftrightarrow (t = (m, t_1, t_2, \dots, t_n) \in M \wedge t' = (t_1, t_2, \dots, t_n) \in T)$$

と定義する。この写像は主にtuple idを取る写像であるが、第一欄がtuple idか否かによらず、第一欄を取る働きをする。さてFの定義にもどるが、ρ_Mを使って$F \subseteq N \times N$を

$$(l, l') \in F \Leftrightarrow \rho_M(l') \in f_T\, \rho_M(l)$$

と定義する。

この様にしてT, f_T, g_TをM, F, Gなる関係で表わす事が出来た。

次にT, f_T, g_Tに対する関係代数をM, F, Gに対する関係代数で表わす事を考えよう。以下ではselection and projection, union, difference, extended cartesian productがM, F, Gに対する関係代数の組み合せでどう実現されるか示す。この時述語$P(t)$は$P(t')$となる。ここでt'はtにtuple idを付け加えたtupleである。又$Q(f_T, T, T', t, t')$は$Q(F, M, M', l, l')$に変更する。例えば$Q(f_T, T, T', t, t') \equiv t' \in f(t)$の時$Q(F, M, M', l, l') \equiv (l, l') \in F$と変更する。他の述語についても同様の変更をしておく。

<Selection and projection>

写像sの定義はもとのままにしておく。Pは上記の様な変更をした述語とする。演算対象の基礎表，行構造写像，列構造写像を実現したものは各々，基礎関係M，行構造関係F，列構造関係Gとし、結果の各々をM', F', G'とする。すると

$$M' = \{(m, t_{s(1)}, t_{s(2)}, \dots, t_{s(k)}) \mid (m, t_1, t_2, \dots, t_n) \in M \wedge P((m, t_1, t_2, \dots, t_n))\}$$

$$F' = \{(l, l') \mid Q(F, M, M', l, l') \wedge l \in \rho_{M'}^{-1}\pi(M') \wedge l' \in \rho_{M'}^{-1}\pi(M')\}$$

$$G' = \{(a, a') \mid R(G, A, A', a, a') \wedge a \in A' \wedge a' \in A'\}$$

但し、ここで$\rho_{M'}^{-1}\pi$はtupleの第一欄を取り出す写像であり，$\rho_{M'}^{-1}\pi(M')$はM'の中で使用されているtuple idの集合に他ならない。

次にunionの場合であるが、tuple idは関係ごとに異なる様にはしないのでunionをとった時tuple idのつけ換えをしないと条件(2)が破られてしまう。その様なtuple id

のつけ換えをする写像を一つ選びλとする。この時単射$\lambda: N\times N \rightarrow N$を使って
unionを定義する。
<union>
　Nの相異なる要素a, bを選んでおく。
$$M''=\{(\lambda(m,a),t_1,t_2,\ldots,t_n)\mid(m,t_1,t_2,\ldots,t_n)\in M\}\cup$$
$$\{(\lambda(m,b),t_1,t_2,\ldots,t_n)\mid(m,t_1,t_2,\ldots,t_n)\in M'\wedge(t_1,t_2,\ldots,t_n)\notin\pi(M)\}$$
　ここで$\pi(M)$のtupleに対しては$\lambda(\ ,a): N\rightarrow N$でtuple idを変更し，$\pi(M')-\pi(M)$のtupleについては$\lambda(\ ,b): N\rightarrow N$でtuple idを変更している。
$$F''=\{(\lambda(m,a),\lambda(m,a))\mid(m,m')\in F\}\cup$$
$$\{(\lambda(m,a),\lambda(m,a))\mid m\cdot\rho_M(m)\in M\wedge m'\cdot\rho_M(m')\in M\wedge(m,m')\in F'\}\cup$$
$$\{(\lambda(m,a),\lambda(m,b))\mid m\cdot\rho_M(m)\in M\wedge m'\cdot\rho_{M'}(m')\in M'-N\times\pi(M)\wedge(m,m')\in F'\}\cup$$
$$\{(\lambda(m,b),\lambda(m,a))\mid m\cdot\rho_{M'}(m)\in M'-N\times\pi(M)\wedge m'\cdot\rho_M(m')\in M\wedge(m,m')\in F'\}\cup$$
$$\{(\lambda(m,b),\lambda(m,b))\mid m\cdot\rho_{M'}(m)\in M'-N\times\pi(M)\wedge m'\cdot\rho_{M'}(m')\in M'-N\times\pi(M)\wedge(m,m')\in F'\}$$
　ここでFのtuple idに関しては全てλ(,a)で変更されているが，F'のtuple idに関してはMのtupleの場合はλ(,a)で変更し，M'のtupleの場合はλ(,b)で変更する。ここではその為に4通りの場合分けを行なっている。
$$G''=G=G'$$
<difference>
$$M''=\{t\mid t\in M\wedge\pi(t)\notin\pi(M')\}$$
$$F''=\{(l,l')\mid(l,l')\in F\wedge l\in\rho_{M''}^{-1}\pi(M'')\wedge l'\in\rho_{M''}^{-1}\pi(M'')\}$$
$$G''=G=G'$$
<extended cartesian product>
$$M''=\{(\lambda(m,m'),t_1,t_2,\ldots,t_n,t'_1,t'_2,\ldots,t'_{n'})\mid(m,t_1,t_2,\ldots,t_n)\in M\wedge(m',t'_1,t'_2,\ldots,t'_{n'})\in M'\}$$
$$F''=\{(\lambda(m,m'),\lambda(l,l'))\mid\lambda(m,m')\in\rho_{M''}^{-1}\pi(M'')\wedge\lambda(l,l')\in\rho_{M''}^{-1}\pi(M'')\wedge Q(F,F',m,m',l,l',M,M',M'')\}$$
$$G''=G\cup G'$$
　以上で関係代数で実現する作業は終った。ここで、λなるtuple idのつけ換えの写像を使用した。この様に写像を使う事により、tuple idのつけ換えが一意的に行なわれる為に、変換の表も必要なく定義が簡素化されている。
《ディレクトリ管理》
　デザイン・エンジンでは関係と展開を表わす関係の間に関連がある。この様な管理をまとめて表にまとめると次の様なネスティッド・テイブルの形に書ける。

操作	基礎関係	構造関係	
		行構造関係	列構造関係
作成	通常通り	基礎関係のtuple idの値域Nを値域に持つ二欄の関係として定義し、この関係がどの基礎関係の行構造を表現しているかを記録	基礎関係の属性の集合Aを値域に持つ二欄の関係として定義し、この関係がどの基礎関係の列構造を表現しているかを記録
削除	付属する構造関係を全て削除して、基礎関係を削除	基礎表とのつながりを消し、行構造関係を消す。	基礎表とのつながりを消し、列構造関係を消す。

《拡大と縮小》

行構造写像や列構造写像の利用例として行と列に関する拡大と縮小の演算を定義する。ここの記号として基礎関係M，行構造関係F，列構造関係Gに関して関係Sを拡大したり縮小したりすることにする。

（行拡大）

$row\text{-}zoom\text{-}in(S, F) = \{t' \mid t \in \rho_S^{-1}\pi(S),\ (t, t'') \in F,\ t' = t'' \cdot \rho_M(t'')\}$

行拡大ではSのtupleのtuple idを行構造関係Fで拡大してtupleを作っている。

次に行縮小を定義する為に一つの記法を定義しよう。関係Fに対し逆関係F^{-1}を

$(t, t') \in F^{-1} \Leftrightarrow (t', t) \in F$

と定義する。これにより行縮小は次の様に定義できる。

（行縮小）

$row\text{-}zoom\text{-}out(S, F) = row\text{-}zoom\text{-}in(S, F^{-1})$

ここで定義した行拡大/縮小では結果の関係はMと同じだけ属性を持つので，一般にはSの持つ属性にprojectionする必要がある。

次に列拡大/縮小を定義する。Sの属性を$\{b_1, b_2, \cdots, b_m\}$，Mの属性を$\{a_1, a_2, \cdots, a_n\}$とすると、列拡大した結果の関係の属性の集合は$\{a' \mid a \in \{a_1, a_2, \ldots, a_n\} \cap \{b_1, b_2, \cdots, b_m\},\ (a, a') \in G\}$となる。この集合は$\{a_1, a_2, \ldots, a_n\}$の部分集合になるので、単射$u : \{1, 2, \ldots, k\} \to \{1, 2, \ldots, n\}$により$\{a_{u(1)}, a_{u(2)}, \ldots, a_{u(k)}\}$と書ける。

（列拡大）

$column\text{-}zoom\text{-}in(S, G) = \{(t, t_{u(1)}, t_{u(2)}, \ldots, t_{u(k)}) \mid (t_1, t_2, \ldots, t_n) \in \rho_M(t) \wedge t \in \rho_S^{-1}\pi(S)\}$

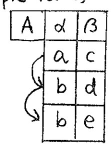
と定義する。

（列縮小）

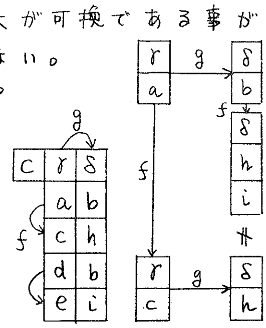
$column\text{-}zoom\text{-}out(S, G) = column\text{-}zoom\text{-}in(S, G^{-1})$

ここで基礎関係自体を演算の対象にしなかったのは，もし基礎表の一部を拡大したり縮小したりしていて、joinなどの演算を行なうと、その時見えていない関係の部分まで変更せざるを得なくなってしまうからである。

明らかとは思うが，拡大して縮小すると元にもどる。亦縮小して拡大すると兄弟が加わる（構造関係が木構造と仮定する。）。行拡大/縮小の場合のこの性質は行拡大/縮小がtuple idにより定義されているという事実に基づいている。Tuple idのつかない関係Aについて考えてみる。関係Aは属性α, βを持ち，現在tuple (a, c), (b, d), (b, e)を持つとする。行構造としてはtuple (a, c)を展開するとtuple (b, d)になり(b, d)は(b, e)になるとする。この時属性αのみを持つ関係Bがtuple (a)を持つとする。この関係BをAを用いて拡大し縮小すると関係(a), (b)となり元と一致しない。



もう1つ、この拡大/縮小の有効な性質として行拡大と列拡大が可換である事が挙げられる。この場合もtuple idを使わない関係Cでは成立しない。関係Cの属性をγ, δとしCはtuple (a, b), (c, h), (d, b), (e, i)を持つとする。$g(\gamma) = \{\delta\}$，$f((a, b)) = \{(c, h)\}$，$f((d, b)) = \{(e, i)\}$とする。この時属性γのみの関係Dがtuple (a)を含むとすると、列拡大を先に行なうと結果の関係はtuple (h), (i)を持つが、行拡大を先に行なうと結果の関係はtuple (h)のみ持つ。



《応用》

以前に試みた例としては、ROMモジュールの設計と管理がある。

ROMモジュールの設計の場では導入部で述べた様に、電子回路をブロック図から回路図に展開しながら設計する機構としてデザイン・エンジンを使用する。次にROMモジュールの管理の場ではモジュール（ボードやチップなど）の管理の為にデザイン・エンジンを使う。この場合、デザイン・エンジンの構造写像はネスティッド・テイブル[4]の構造を表現するのに使われ、そのネスティッド・テイブルを使用して在庫管理が出来る。ネスティッド・テイブルを表わす例として、ディレクトリ管理の所にあるネスティッド・テイブルは次の様に表わされる。基礎表は属性として'操作'，'基礎関係'，'構造関係'，'行構造関係'，'列構造関係' を持つ様な関係（但し'構造関係'には空値域が対応するもの）とする。次に列構造写像 g として g('構造関係')＝{'行構造関係'，'列構造関係'} で他の属性については空集合を対応させるものと定義する。この基礎表と構造写像によりネスティッド・テイブルが表現されている。

この様にデザイン・エンジンで展開型デザインが出来るだけでなく、ネスティッド・テイブルの様に在庫管理の用途にも利用できる。このことは一つの定式化が多様な用途に利用できるというだけでなく、両者がデーターを共有できるという利点がある。例えば、設計を若干変更した時、現在のモジュールの在庫でできるか、という様な質問にも対応する事が出来る。

《参考文献》

1. G. Estrin, "A Methodology for Design of Digital Systems – Supported by SARA at the Age of One," Proc. AFIPS National Computer Conference, June 1978, Anaheim, California, pp. 313–324 (1978, AFIPS Press).

2. T. L. Kunii and M. Harada, "SID: A System for Interactive Design," Proc. AFIPS National Computer Conference, Anaheim California, May 1980, pp. 33–40 (May 1980, AFIPS Press).

3. E. F. Codd, "A Relational Model of Data for Large Shared Data Banks," Comm. ACM 13, pp. 377–387 (1970).

4. H. Kitagawa, T.L. Kunii, M. Harada, S. Kaihara and N. Ohbo, "A Language for Office Form Processing (OFP) – With Application to Medical Forms –," Proc. Third World Conference on Medical Informatics, October 1980, Tokyo Japan (1980, North-Holland, in press).