文書処理とヒューマンインタフェース 23 － 4
（１９８９. 3. 8）

文書プログラミングについての一考察

京嶋仁樹　　上林憲行　　N. V. ゴパル
富士ゼロックス(株)システム技術研究所　　富士ゼロックス情報システム(株)

オフィス業務のプログラミングについての研究はかなり行われているが、文書処理業務に関しては、自動化されている業務は依然として少ない。このことは、データとしての構造を持たない文書を対象としていたこと、および、文書の伝達を記述の対象としなかったことに起因する。これに対し、構造化された文書と分散環境の視点から文書処理業務をとらえ、それを記述するものが文書プログラミングである。文書プログラミングでは、すべての文書はプログラミングの対象となる構造化されたデータであり、文書処理はその構造間の写像として記述される。本論文では、文書プログラミングを定義し、その可能性と課題について検討する。

Document Programming

Masaki Kyojima　　Noriyuki Kamibayashi　　N. Venu Gopal
System Technology Research Laboratory　　Fuji Xerox Information Systems Co., Ltd.
Fuji Xerox Co., Ltd.
57-6, Yoyogi 3-chome, Shibuya-ku, Tokyo

A great part of office activity involves the creation and manipulation of documents. Such activities have until recently appeared impossible to automate. However, the advent of structured documents has changed the outlook of many problems inherent to this area. In this paper we propose document programming, a term we use to denote the programming of such document-related office activities, as a solution for these problems, describe the document programming model of the office, and discuss its potential in automating document-related office work.

1. はじめに

パーソナル･コンピュータやワークステーションなどのオフィス･システムがオフィスに導入されたことにより、オフィス業務は電子化された環境で行われるようになった。これらのシステムは、データベースの操作、あるいは文書のページ割り付け処理など、オフィスに普遍に存在する業務を自動化したものであった。これらのシステムが、業務の効率化に貢献したことにより、オフィス業務の電子化の効用が明確となった。しかし、オフィス業務が、小規模な業務の集合体であることや、その処理が組織別、部門別、個人別に少しづつ異なるという多様性を有していることにより、これらのオフィス･システムで自動化される業務は、オフィス業務全体からみれば、限られたものであった。

このようなオフィス業務の多様性の克服のために、オフィス･システムを個々の業務用に最適化することが考えられた。このために、最適化システムとして、プログラミング･システム(以下オフィス･プログラミング･システムと呼ぶ)が付加され、オフィス業務の自動化はより進展した。このことにより、プログラミングによるオフィス･システムの個別最適化の重要性が認識された[1]。

本論文においては、このようなオフィス･システムの個別最適化について、文書処理の自動化の視点から考察する。

現在のオフィス･プログラミング･システムは、その記述の中心をフォーム･プロセッシングにおいている。フォーム･プロセッシングは、フォームに対するデータの入出力口が決まっており、一般の文書と比較して、処理の記述が容易であった。このような、限定された文書処理のプログラミングは、オフィス業務の自動化を一層推し進めるとともに、より高度な一般の文書処理のプログラミングの必要性を顕在化させた。しかし、プログラミングのための一般的な文書処理のモデルは、これまでなかなか議論されなかった。

最近の研究の成果により、このような状況は徐々に克服されつつある。

その要因となったものの一点は、電子的な文書そのもののデータとしての構造が提案されるようになってきたことである[2][3][4]。文書がデータとしての構造を持つことにより、文書処理を、構造を持つデータの処理として記述することが可能になる。

もう一点は、文書処理が複数のコンピュータがネットワークでつながった分散環境下で行われるようになったことである。文書は、情報の媒体であり、それゆえに、文書処理は文書の伝達を伴う。文書処理が分散環境下で行われるようになったことにより、このような文書の伝達を電子的にシミュレートすることが可能になった。

このような状況の変化により、電子的な文書処理のモデルについて論じることが可能になった。

上記のような背景をふまえて、本論文では文書処理を対象としたプログラミングについて考察する。特に、文書処理を分散環境下における構造をもつデータの処理と考え、そのプログラミングを文書プログラミングと呼ぶこととした。文書プログラミングの概念と定義を明確化し、可能性と課題を検討するのが本論文の目的である。

以下、第2章では、これまでのオフィス業務のプログラミングについて概観し、そこから、文書プログラミングに対する要求を抽出する。第3章では、分散環境下での文書処理および文書そのものについてのモデルを説明し、その後に、文書プログラミングを定義する。第4章では、文書プログラミングの可能性と課題について考察する。最後の第5章では、この論文における考察をまとめ、これからの研究の方向性について述べる。

2. オフィス業務のプログラミングと文書プログラミング

オフィス業務のプログラミングの発展は、その記述の対象となるデータの構造の汎用性の変化によってその流れをとらえることができる。

オフィス業務のプログラミングは、個々の業務別にアプリケーション･プログラムを作成することから始まった。これらのアプリケーション･プログラムは、特定の業務を解析し、COBOLあるいはBASIC等のプログラミング言語によって記述された。これらのプログラムにおけるデータの構造はプログラムごとに違い、ほとんど汎用性がなかった。

次に、2次元の会計用紙の表計算を対象としたスプレッドシートが作られた。スプレッドシートがプログラミングの対象とするデータ構造は、2次元の表であり、この構造に落とすことのできるデータを用いたオフィス業務のほとんどを記述することができた。

データ構造の汎用性がより向上したのが、現在のオフィス･プログラミング･システムである。オフィス･プログラミング･システムは、データ構造としてフィールドを持っており、このフィールドを文書に挿入することによって任意のフォームをプログラミングの対象とするこ

とができた。

このように、現時点までに、文書の部分集合であるフォームの処理までをプログラミングの対象とすることのできるデータ構造が実現されている。しかし、これによって記述することのできる業務は、オフィス業務全体からみれば、限られたものである。以下では、現在の代表的なオフィス･プログラミング･システムであるCUSP(CUStomer Programming)[5] とOBE(Office-By-Example)[6][7]について概観し、これらのシステムでは未解決な課題を抽出する。

2.1 現在のオフィス･プログラミング･システム

ここでは、前述のCUSPとOBEについて比較する。比較の視点は以下の3点である。

① 記述対象

② 記述力

③ ユーザインタフェース

▸ CUSP

CUSPは、XeroxのStarワークステーション[8][9]上に実装されているユーザ向けの簡易プログラム言語である。CUSPは、データの収集や、文書処理など、種々のオフィス業務に適用され、多くのユーザを獲得している。

① CUSPは、フォーム･プロセッシングと、デスクトップ上のアイコンの操作を記述の対象としている。

処理対象である文書はStarワークステーション上で処理される文書であるが、CUSPでは文書中に挿入されたフィールドに対してのみ、その処理を記述することができる。したがって、CUSPにおいて文書はフィールドの列であり、各フィールドは識別子によって区別される。また、文書内のテーブルにおける各セルもフィールドと同様に扱うことができる。

② フィールド内の値の操作、数値演算を記述することができる。

③ CUSPプログラムは、手続型の簡易言語であり、テキスト･エディタを使って作成される。CUSP言語は英語に似た形態をしており、ユーザにとってより親和性の高い言語となっている。また、Daniel C. Halbertは、Starワークステーションの環境をSmalltalk上でシミュレートしたSmallStar[10]にProgramming-By-Exampleのプログラミング･システムを導入し、プログラミングに必要な負荷を減らすことを考えた。

▸ OBE

OBEでは、オフィス業務の中心をデータベース･マネジメントにおいており、データベース･マネジメント･システムであるQBE[7][11]と、文書処理、メイル、ビジネス･グラフなどとの統合をはかっている。

① OBEの記述対象は、リレーショナル･データ･ベース(RDB)である。文書や、ビジネス･グラフは、RDBから検索したデータを流し込むためのフィールドが挿入され、検索されたデータのビューとして使われている。

② 文書処理に関しては、RDBから検索された値、または、演算された値を流し込むこと以外の処理をプログラミングの対象とはしていない。

③ QBEのコマンドは、対象となるRDBと同じ構造を持ったテーブルを使って記述される。RDBの特定の属性についての記述は、テーブルの同じ属性の下に記述される。記述される位置によって、どの属性に対する記述かが特定される。

2.2 オフィス･プログラミング･システムの課題と文書プログラミング

▸ オフィス･プログラミング･システムの残した課題

前述したように、CUSPとOBEはフォーム･プロセッシングの記述を可能にしているが、ユーザの要求には、まだ十分答えきれていない。これについて以下で、①記述対象、②記述力、③ユーザインタフェース、に分けて検

討する。

① 記述対象について

これらのシステムは、本来データとしての構造をもたない文書を対象としている。それを補うために、文書内にデータの入出力のためのフィールド等を挿入している。したがって、プログラムのデータとなることを前提に予めフィールドを挿入した文書のみがプログラミングの対象となりえる。

しかし、通常の文書は、それがデータとなるプログラミングを意識して作られることはない。また、新しいプログラムを作成するときに、新しいフィールド付き文書を作る必要がある。したがって、文書が文書としてのデータ構造をもち、フィールドなどの特別なものを挿入しなくても、プログラムのデータとすることができることが必要である。

また、どのシステムも、単一のコンピュータ上における文書処理を記述の対象としている。しかし、文書の伝達は文書処理業務の中で重要な位置を占めるものであり、これを記述することが必要となる。

② 記述力について

フィールドは、電子的な紙の上に置かれる。紙の上に表現されるデータの入出力については、これによって記述することが可能である。しかし、文書のもつ構成やレイアウトについての情報をフィールドに持たせることはできない。このため、文書の再構成やレイアウトの変更、文書構造のマッチングなど、紙の上に表現されたデータ以外の情報を必要とする文書処理業務を記述することができない。したがって、紙の上に表現されたデータだけでなく、文書のもつ構成やレイアウトについての情報をデータとして文書にもたせる必要がある。

③ ユーザインタフェースについて

どのシステムもプログラマとして、プログラミングの経験を持たないオフィス・ワーカを想定している。したがって、学習時間が短く、プログラミングの経験の必要がなく、かつ、記述力の高いユーザ・インタフェースが必要となる。

▸ 文書プログラミングに対するニーズ

現在のオフィス・プログラミング・システムに対する上記のよう課題を満たすものとして、本論文では、文書プログラミングを提案する。したがって、文書プログラミングは、以下のような要件を満たすことが必要であると考えられる。

a) 文書が、紙の上の情報だけでなく、その構成やレイアウトについての情報をもち、それらが構造をもったデータであること

b) オフィスにおける文書の流れを記述することができること

c) プログラミングの経験を持たないオフィス・ワーカに親和性の高いユーザ・インタフェースを持つこと

このような要求を満たす文書プログラミングを提供することにより、電子化・分散化されたオフィス環境の潜在的なメリットをより多く引き出すことができると考えている。

▸ 文書プログラミングの持つ技術的な背景

上記のような文書プログラミングに対する要求を満たすことは、これまで技術的に困難であった。しかし、以下のような技術的な進歩により、上記の要求を満足するものを考えることが可能になった。

a) 文書のモデルとしてストラクチャド・ドキュメント[2]やhypertext[3]などが提案され、データとしての文書の構造を考えることが可能になったこと。

b) オフィス環境として、分散環境が徐々に定着しはじめ、オフィス・ワーカ間の情報伝達を電子的にシミュレートすることが可能になったこと。

これにより、文書そのものとその流れをモデル化することが可能になり、そこでの文書処理をプログラミングの対象として考えることができるようになった。

3. 文書プログラミング

ここでは、まず、文書プログラミングで記述しようとする分散環境における文書処理と、その文書処理の対象となる文書のモデルについて検討する。その後、文書プログラミングを定義する。

3.1 文書フロー･モデル

前述したように、オフィス業務は、文書がネットワークを介して複数の人間の間を流れることによって実行される。このような文書の流れは図1のようにとらえることができる。

図1では、文書処理を行う実態(オフィス･ワーカ、あるいは、コンピュータ)をノードと考える。文書は、これらのノードの間を流れていく。文書がノードに到達するとノードでは文書の加工/修正、新しい文書の生成が行われ、次のノードに送られる。このような文書の流れを文書フローと呼ぶ。

それぞれの文書処理業務においては、図1のような文書の流れが決定されている。また、それぞれのノードで行われる文書処理も決まっている。したがって、それぞれの文書処理業務について、図2のような文書フロー･グラフを書くことができる。　図2では、{a, b, c, d, e}が文書を、{α, β, γ}が文書に施される文書処理を表している。例えば、文書cは、文書aに文書処理αが施されることによって生成される。このことは、

$$c = \alpha(a) \qquad (1)$$

と、記述することができる。また、文書eが文書aと文書bから生成されることは、

$$e = \gamma(\alpha(a), \beta(b)) = \delta(a, b) \qquad (2)$$

と、記述することができる。

同様に、すべての文書処理は、一つ以上の文書の集合Xに、文書処理λを施し、一つ以上の文書の集合Yを作成することとして、

$$Y = \lambda(X) \qquad (3)$$

ただし

$X = \{x_1, x_2, x_3, \ldots, x_m\}$

$Y = \{y_1, y_2, y_3, \ldots, y_n\}$

x_i ($1 \leq i \leq m$)は文書クラスCx_i ($1 \leq i \leq m$)のインスタンスである文書

y_j ($1 \leq j \leq n$)は文書クラスCy_j ($1 \leq j \leq n$)のインスタンスである文書

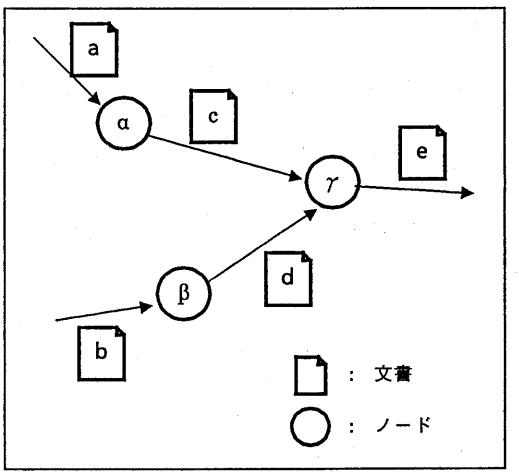

図2　文書フロー･グラフ

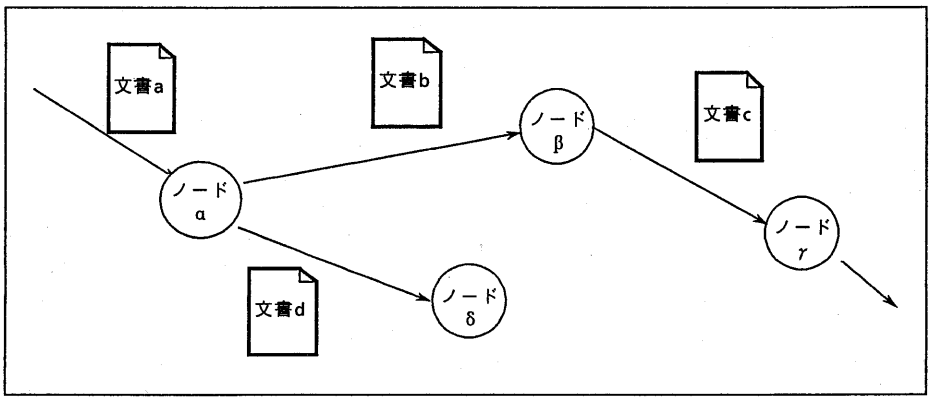

図1　文書フロー･モデル

と、定式化することができる。

3.2 文書モデル

文書プログラミングにおいては、文書の実体を構造化されたデータであると考える。すなわち、文書xは、複数の構造をもったデータとして(4)式のように定式化される。

$x = \{ xs_1 , xs_2 , xs_3 , \cdots , xs_m \}$

ただし、

文書xは文書クラスCxのインスタンスである文書

xs_i ($1 \leq i \leq m$) は文書xのもつ構造

(4)

3.3 文書プログラミングの定義

上記のような文書フロー･モデル、および文書モデルを採用したことにより、文書処理は文書のもつ構造の集合から、別の文書の持つ構造の集合への変換を行うものとして(5)のように定式化することができる。

$Y = \lambda (X)$

ただし

$X = \{ x_1 , x_2 , x_3 , \cdots , x_m \}$

$Y = \{ y_1 , y_2 , y_3 , \cdots , y_n \}$

x_i ($1 \leq i \leq m$)は文書クラスCx_i ($1 \leq i \leq m$) のインスタンスである文書

y_j ($1 \leq j \leq n$)は文書クラスCy_j ($1 \leq j \leq n$) のインスタンスである文書

$x_i = \{ x_is_1 , x_is_2 , x_is_3 , \cdots , x_is_p \}$

$y_j = \{ y_js_1 , y_js_2 , y_js_3 , \cdots , y_js_q \}$

x_is_k ($1 \leq k \leq p$) は、文書x_iの持つ構造

y_js_l ($1 \leq l \leq q$) は、文書y_jの持つ構造

(5)

文書プログラミングは文書処理λを記述するものである。λは文書のもつ構造の集合から、別の文書の持つ構造の集合への写像であり、(6)式のように定式化される。

$\lambda : X \rightarrow Y$

ただし

$X = \{ x_1 , x_2 , x_3 , \cdots , x_m \}$

$Y = \{ y_1 , y_2 , y_3 , \cdots , y_n \}$

x_i ($1 \leq i \leq m$)は文書クラスCx_i ($1 \leq i \leq m$) のインスタンスである文書

y_j ($1 \leq j \leq n$)は文書クラスCy_j ($1 \leq j \leq n$) のインスタンスである文書

$x_i = \{ x_is_1 , x_is_2 , x_is_3 , \cdots , x_is_p \}$

$y_j = \{ y_js_1 , y_js_2 , y_js_3 , \cdots , y_js_q \}$

x_is_k ($1 \leq k \leq p$) は、文書x_iの持つ構造

y_js_l ($1 \leq l \leq q$) は、文書y_jの持つ構造

(6)

3.4 例

ここでは、文書プログラミングによって記述される文書処理の例をあげる。この例では、文書の構造としてPart-ofのツリー構造を考えている。

例1 フォーム･プロセッシングの記述

オフィスでの文書処理は決められたフォームを使って行われることが多い。フォーム･プロセッシングは、フィールドをフォームに埋めこむことによりプログラミングすることが可能であり、CUSP、およびOBEでも記述の対象となっている。ここでは、文書プログラミングで同様の処理の記述が可能であることを示す。

文書を構造をもったデータと考えた場合、フォームも構造をもった文書として表現される(図3参照)。しかし、フォーム･プロセッシングでは文書の構造を操作することはない。したがって、フォームにおける文書処理は、複数の構造間のデータのマッピングとして記述することができる。

図3は、データのマッピングの図である。xsが入力フォームxの構造、ysが出力フォームyの構造である。xsのデータにはマッピング関数によって演算が施され、ysにマッピングされる。

例2 データの追加による文書の更新の記述

オフィスにおける文書処理の中で、新しいデータが入ってくる度に、特定の文書にデータを追加する業務がある。例えば、特定の分野の文献リストの文書があり、その文書は新しい文献が入ってくる度にその文献のタイトルや著者、キーワード、要約が付加され更新されるような業務である。

このような業務は、既存の文書の構造が修正される

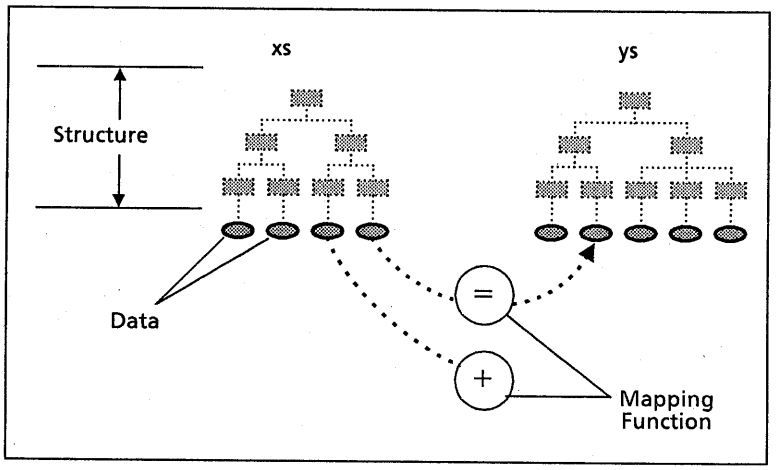

図3　フォーム·プロセッシング

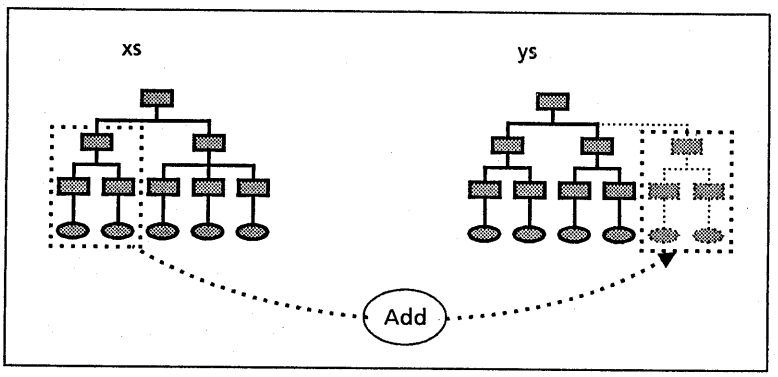

図4　データの追加

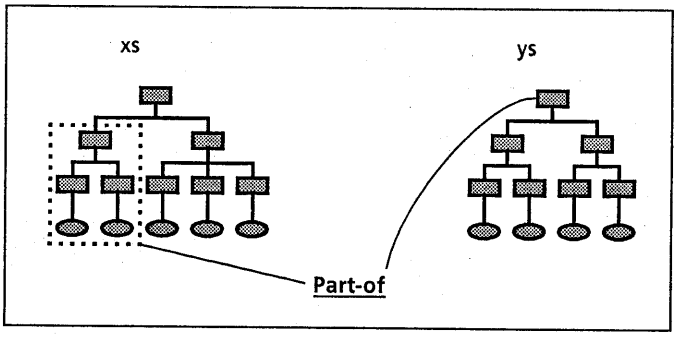

図5　制約による記述

ものとして記述することができる。

図4は、既存の構造の修正(この場合は、部分的な構造の追加)の図である。xsが入力文書xの構造、ysはxが入ってきたことにより更新される文書yの構造である。xsの一部がysに付加されることにより、yが更新される。

4. 考察

4.1 プログラムの記述方式について

前章で述べたように、文書プログラミングは、文書のもつ構造間の写像を記述するものである。このような写像は構造間の制約[12]と見ることができる。

たとえば、構造内の特定のパーツpxが、マッピング関数λによって別のパーツpyにマッピングされる、すなわち

$$py \leftarrow \lambda (px) \qquad (7)$$

は、

$$px \text{ is constrained to } py \text{ by } \lambda \qquad (8)$$

と考えることができる。ここではマッピング関数λは、制約のデバイスと考えている。

このような制約による記述を考えた場合、図4における写像は、図5のようにPart-ofのデバイスによる制約として記述することができる。

上記のように、文書プログラミングは、制約の概念との親和性が非常に高い。したがって、文書プログラミングを文書の構造間の制約として記述することが考えられる。しかし、文書プログラミングのすべてを制約として記述できるのか、また、その記述は自然なものであるかについては、今後の課題である。

4.2 構造間の写像としての文書プログラミングの持つ課題

本論文では、文書プログラミングは(6)式で示されるような、構造の集合から別の構造の集合への写像であると定義した。ここでは、その技術的な課題について述べる。

ある文書xからある文書yへの写像をλと考える。

$$y = \lambda (x) \qquad (9)$$

λが複数の写像$\{ \lambda_1 , \lambda_2 , \lambda_3 \}$に分解することが可能な場合、$\{ \lambda_1 , \lambda_2 , \lambda_3 \}$の相互間に何の依存関係もなければ、(9)式は、

$$y = \lambda_3 \lambda_2 \lambda_1 (x)$$

ただし、λ_1、λ_2、λ_3の順序はどういれ変わってもよい (10)

と記述することができる。

(10)の式は図6のような文書フロー･グラフに対応する。

上記のことから、(6)式で規定される構造の集合から別の構造の集合への写像を記述することにより、構造間の関係だけでなく、文書フロー･グラフも記述されることがわかる。このことからも、(6)式のλが、分散環境下での文書処理の記述として十分であることがわかる。

しかし、実際の分散環境において上記のこと考えた場合、いくつかの問題が残されている。

そのひとつは、構造間の関係の単位となる最小の関係についての問題である。

(9)式と(10)式で、λが$\{ \lambda_1 , \lambda_2 , \lambda_3 \}$に分解されたように、写像はどんどん分解されていく。そして、いずれこれ以上分解することの不可能な最小の関係にまで分解される。ここで、これらの最小の関係の数が有限個かと

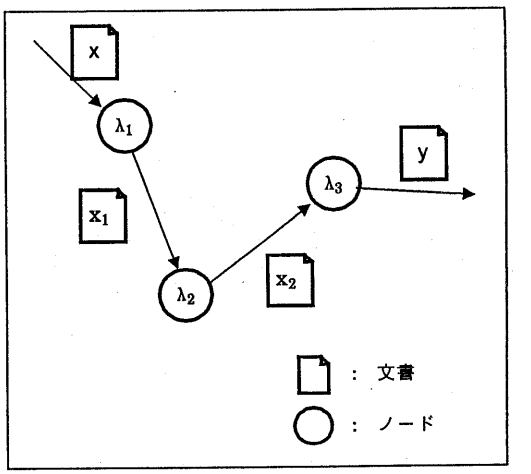

図6 文書フロー･グラフ

いうことが問題になる。もし、無限個存在するか、あるいは、無限と見なしえる程度の膨大な数が存在するのであれば、それを回避する手段が必要となる。

もうひとつの問題は、写像が複数の写像に分解されたのちに、コンピュータなどの実際の分散環境上のノードにマッピングされる際の問題である。

分解された写像の集合は、個々の写像ごとにその写像を実行することの可能なノードに割り付けられる。したがって、個々の写像ごとにその写像を実行することの可能なノードを捜す機能が必要となる。しかし、現時点ではこのような要望を満たすことのできる技術は見られない。

このような問題は、現在のところ未解決であり、今後の課題として残されている。

5. まとめ

本論文では、分散環境下におけるオフィス業務を支援する際に、文書処理をプログラミングすることが大きな効用を持つとの視点から、文書プログラミングを提案し、その概念と定義について述べた。

文書プログラミングでは、文書を複数の構造を持つデータとしてモデル化し、文書そのものをプログラムのデータとして扱おうとしている。これにより、これまでフォーム・プロセッシングが中心であった文書処理のプログラミングを、より一般的な文書処理のプログラミングにまで拡張できると思われる。

また、文書フロー・モデルにより分散環境下の文書の流れをモデル化し、複数のオフィス・ワーカを介した文書処理業務をプログラミングの対象としている。

このような、文書および文書処理への新しいモデルの導入により、分散環境下の文書処理業務についての大きな記述力を提供することができると考える。

今後は、具体的な文書の構造を仮定し、構造間の写像の解析、および記述方式の検討をおこなっていく予定である。

謝辞

まず、本論文の発表の機会を与えてくださった富士ゼロックス(株)システム技術研究所 加藤所長に感謝いたします。

また、本論文の構想をまとめる上で、同研究所内で貴重な議論がなされました。議論に加わっていただいた所員の皆さん、特に、有益な助言をいただいた楠本所員、竹岡所員に感謝します。

参考文献

[1] Tsichritzis, D.: Office Automation, D. Tsichritzis (ed.), Springer-Verlag, 1985.

[2] Furuta, R.: Specifying Structured Document Transformations, Document Manipulation and Typography, Cambridge University University Press.

[3] Smith, J. B.: An Overview of Hypertext, CACM Vol.31 No. 7, 1988.

[4] ISO/IS 8613: Information Processing - Text and Office Systems - Office Document Architecture (ODA) and Interchange Format, 3/1988.

[5] VP CUSPボタン操作説明書, 富士ゼロックス株式会社, 1988.

[6] Zloof, M. M.: Office-by-Example: A business language that unifies data and word. processing and electronic mail, IBM Systems Journal, 1982.

[7] Zloof, M. M.: QBE/OBE: A language for Office and Business Automation, IEEE Computer Vol. 14 No. 5, 1981.

[8] Smith, D. C. et al: The Star Users Interface: An Overview, AFIPS Proceedings of National Computer Conference, vol. 51, pp. 515-528, 1982.

[9] Smith, D. C. et al: Designing the Star User Interface, Byte, 4/1982.

[10] Halbert, D. C.:Programming by Example Ph.D. dissertation, Univ. of Cal., Berkeley, 1984.

[11] Zloof, M. M.: Query-by-Example: a database language, IBM Systems Journal 1977.

[12] Steele, G. L. :The definition and Implementation of a Computer Programming Language Based on Constraints, AI-TR-595, MIT.