情報システム 35－1
（1991. 7. 16）

ワーク・フローに基づくビジネス・アプリケーション構築方法

浅川　康夫　　小松　秀昭　　濱　利行
日本アイ・ビー・エム株式会社東京基礎研究所

一つの目的のために行われるオフィス・ワークの流れを、処理すべきフォームの流れと、フォームを処理するためにとられる行動とフォーム上のフィールドとの依存関係から捉え、一連の処理の自動化、ユーザーの操作に応じた動的な実行順序の制御、仕事の状態に従ったナビゲーションなどを可能とするシステムの構築方法を示す。

A Method Based on Work Flow to Build Business Applications

Yasuo Asakawa, Hideaki Komatsu, and Toshiyuki Hama
Tokyo Research Laboratory, IBM Japan Ltd.

I show a method to define a work flow in office work as the flow of forms to be processed and the dependency among actions for processing forms and fields on forms, and to build the system which uses the definition and allows automated exection of a series of applications, dynamic control of exection flow based on user operations, navigation based on the status of work, etc.

【はじめに】

パーソナル・コンピュータがオフィス環境で広く利用されるようになって、いったいどれだけオフィス・ワークが効率良く行われるようになっただろうか。確かに、ワープロやDTPによって奇麗な文書が容易に作成できるようになったし、スプレッドシートによって計算処理も楽になった。また、通信ソフトによって、電子メールやホスト・アプリケーションの利用が可能になった。しかし、これらのソフトウェアは、オフィス・ワークの中のある特定の処理を支援するツールとして利用されるだけで、オフィス・ワーク全体の流れの中で、他の処理と連係して効率化をはかるという形で利用されるケースは少ない。
言い換えれば、ツールは増えたが仕事の仕方そのものを変えるものにはなっていない。

オフィス・ワークといってもいろいろなものがあるので、ここでは一般的な事務処理、例えば、出張旅費の精算とか銀行支店における融資手続きといったものを考える。そして、出張旅費精算・融資手続きといった一まとまりの仕事全体をプロセスと呼ぶことにする。一つのプロセスの中にはいろいろな行動が含まれる。例えば、

・適当なフォームを探す
・データの収集、必要ならデータベース検索をしたり、以前の文書を調べたり、人に聞いたりする
・データをフォームに書き込む
・計算する
・フォームにサインする、印鑑を押す
・（社内）メールで送る
・承認をもらう
・フォームを保管する、あるいは、破棄する
・後の処理を誰かに任す
・電話などで連絡する
・　…

などが考えらえる。また、プロセスと個人の仕事全体を見てみると、

・同じ種類の、あるいは、異なった種類の多くのプロセスを同時に抱えている
・それらのプロセスは、その人にとってよく知られているものである場合もあれば、たまにしか処理されないあまり知られていないものの場合もある
・同じ種類のプロセスでもその中に含まれるデータによっては処理の仕方が異なる場合もある
・プロセスの処理の仕方そのものが変わることもある
・短期間で終るプロセスもあれば、長期間かかるものもある

といったことが言える。人は、このような状況のもとで、出張旅費精算や融資手続きといった個々のプロセスの中の行動を適宜判断し、ツールを選択・起動・操作して、仕事をしている。

本研究では、プロセス全体をひとつのアプリケーションとして統合し、その中の仕事の流れ（ワーク・フロー）を考えることで、複雑なオフィス環境下で人の判断・行動を支援・省力化するシステム、ワーク・フロー・マネジメント・システム（ＷＦＭＳ）を考案

した。

【アプローチ】

オフィス・ワークをみてみると、オフィスにおけるかなり多くのプロセスが、その目的を達成するために必要なフォームを処理する、処理のために必要なデータを揃えるためにいろいろな行動をしたり、ツールを使ったりするといったかたちで動いている。そこで、我々は、プロセスを、

プロセスに必要なすべてのフォームの処理の流れ、およびフォームの処理に必要な行動の集まり

として捉えることにした。すなわち、

1．どのようなフォームが必要か
2．各フォームにはどのようなフィールドがあるか
3．フォーム上のフィールドはどのように配置されて、表示されたり印刷されたりするのか
4．あるフォームはフィールドの値がどのようなときに処理が終ったと見なされるか
5．あるフォームの次に処理するフォームはどのようなものがあって、それらはどういう条件のとき選ばれるのか
6．あるフィールドをうめるために、どのような行動をとれるか、それらはどのようなフィールドが埋められたときに実行できるか

と言ったものでプロセスを定義する。ここで、行動はアプリケーションの呼出に相当し、単なる計算の場合もあれば、データベースの検索や他の人との通信、あるいは操作する人への行動（コンピュータの操作に限らない）の指示だったりする。以後、プロセスの構成要素としての行動をアクションと呼ぶ。1・2・3はプロセスと人とのインターフェイスを、4・5はフォームの流れを、6はアクションとフィールドの関係を規定するもので、それぞれ、View Definition、 Flow Definition、 Work Definitionと呼ぶ。

このような捉え方は二つの点で重要である。ひとつは、このように定義されたプロセスは、現実に行われている処理の仕方に近いものであり、運用する立場から受け入れ易いこと。もうひとつは、個々のアクションに相当するアプリケーションをどう書くかという問題はあるものの、プロセス自体の定義は、プログラミングのスキルはないけれどプロセスは良く知っているビジネスの専門家が容易に定義できることである。

そして、本研究では、このようなプロセスの定義から、

・常に、人の処理の必要なフォームを表示する
・フィールドへの新たな入力に対して実行可能なアクションを自動的に起動する
・アクションの結果をフィールドにフィード・バックする

という処理を基本し、更に、

・一度入力されたフィールドのデータ修正にともなうアクションの再実行（複数のアクションにまたがることもある）

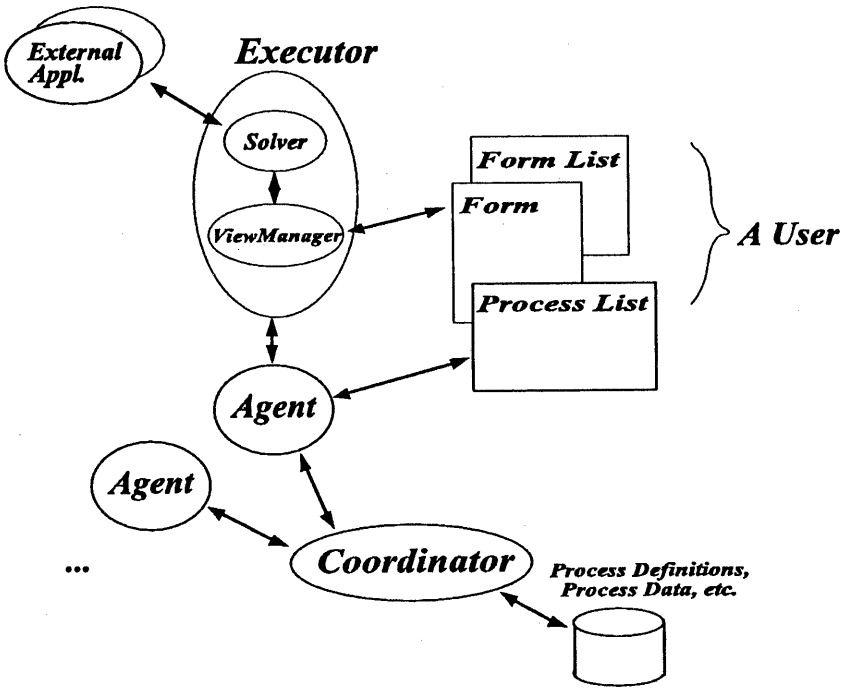

図１．ＷＦＭＳの構成

・システムのガイドによらずに、いつでも人が割り込んで主導権を取り、好きなところを見たり入力できること
・複数アクションの同時平行実行
・プロセスの中断・再開の自由
・全てのプロセスの状況の管理

などの機能を含むシステムＷＦＭＳを構築する方法を考案した。

【ＷＦＭＳの構成と処理方式】

ＷＦＭＳは大きく分けて、WFM/Executor、WFM/Agent、WFM/Coordinatorの三つから構成される。WFM/Executorは実行中の各プロセスに一つ存在し、そのプロセスの実行に必要なデータ、フロー、アクションの管理を行う。WFM/Agentは個人毎に一つ存在し、その人が実行できるすべてのプロセスを管理する。WFM/Coordinatorはオフィスに一つ存在し、ユーザーやグループの管理、ユーザーやグループ毎のプロセスのデータの管理（データ・サーバー）、プロセスの移譲、異なるプロセス間でのデータのやり取りといったことを行う。図１はこれらの関係を示したもので、現在のプロトタイプとしてはWFM/ExecutorとWFM/Agentが実現されており、一個人内での管理が可能である。以下では、実際の実行の中心と成るWFM/Executorについて説明する。

WFM/Executorは二つのコンポーネント、SolverとViewManagerで構成される。Solverはプロセス内のデータおよびアクションを管理する。一方、ViewManagerはフォーム、フォーム内のフィールドとデータの対応、フォームの流れおよびユーザーとのインタラクションを

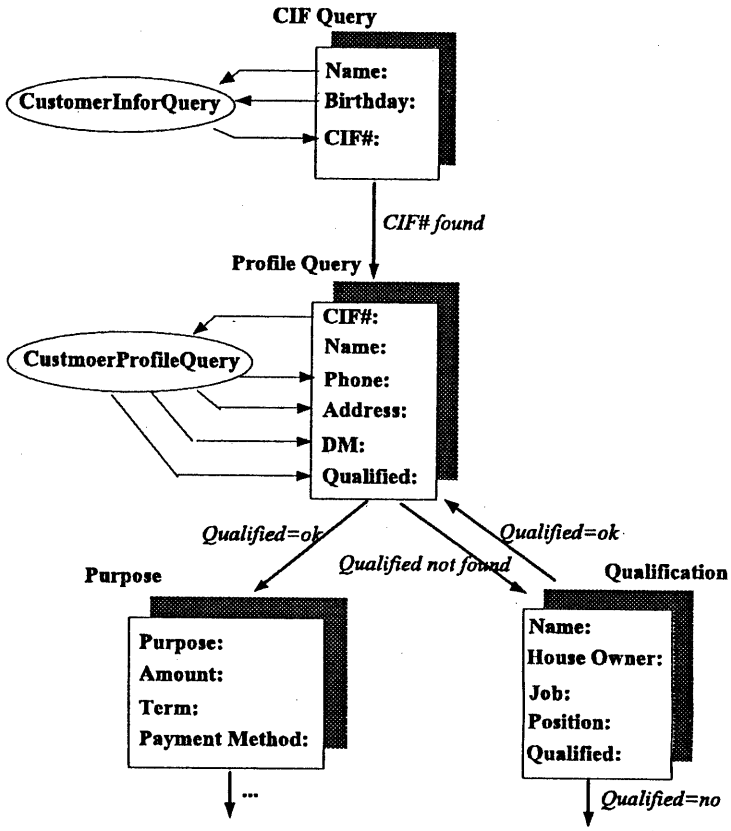

図2.プロセスの視覚的定義の例

管理する。二つは平行して動きながら、メッセージによる通信でお互いのデータを交換している。ユーザーによる入力はViewManagerを通してSolverに送られ、なんらかのアクションによって求まったデータはSolverからViewManagerに送られ、フォームの表示に反映される。従って、ユーザーとのインタラクションとアプリケーションの実行は独立に行われる。あるフィールドを入力したあとで、データベースの検索が始まったとしても、ユーザーによるオペレーションが妨げられることはない。

ViewManagerはView DefinitionとFlow Definitionをもとに、プロセスに定義されている全てのフォームのリストを、各フォームがいま処理可能かどうか、処理が済んでいるかといったステータスといっしょに表示するとともに、一番処理すべきと思われるフォームを表示する。そのフォームは、フォームのフローを最初からたどって、行き着いたフォームの中から順番に選ばれる。そして、フォームのリストおよびフォームの選択・表示は、ユーザーからの入力やSolverからのデータ更新のメッセージがある毎に更新される。

SolverはWork Definitionから、データとアクションの依存関係をあらわすグラフと生成する。これをデータ・フロー・グラフと見立てて、入力の揃ったアクションから順次起動していく。各アクションは、お互いに、そして、Solverとも独立で非同期で動き、同時に複数のアクションも動き得る。アクションからの結果は、ViewManagerがメッセージをもってSolverにユーザー入力を知らせたと同様にメッセージで帰す。Solverは起動可能なアク

ションをすべて起こしたあとは、ViewManagerやアクションからのメッセージを待つ。そして、メッセージが届くと、それをデータ・フロー・グラフに反映させて、あらたな起動可能なアクションはないか調べて、あれば起動する、ということを繰り返す。

図２は、あるプロセスの定義の一部分を視覚的に表現した例である。この中には、四つのフォームと二つのアクションが定義されている。この例で上の処理を説明すると、まず、ViewManagerはフォームCIFQueryを表示する。ユーザーが入力可能なフィールドNameとBirthdayを埋めると、ViewManagerはそれをSolverに送る。SolverはNameとBirthdayがもとまったことで、アクションCustomerInfoQuery（例えばホストのデータベース検索）を起こす。CustomerInfoQueryはCIF#の値が求まるとそれをSolverに送る、SolverはCIF#がもとまったことであらたなアクションCustomerProfileQueryを起動するとともにCIF#をViewManagerに送る。ViewManagerはCIF#がもとまったことで、フォームCIFQueryの処理は終ったとみなし、フィールドCIF#とNameが埋まったフォームProfileQueryを表示する、と言うように進む。

以上は基本的な処理の流れであるが、Solverには、通常のデータ・フロー処理にはないいくつかの特徴的な処理が必要となる。

１．一つのフィールドを埋めるアクションが複数あることを許す。これは図３のような場合で、アクションＸもアクションＹもフィールドＦを埋めるためのデータベース検索であるけども、どちらのアクションで埋めてもいいといった場合である。アクションは平行して動けるので、このような出力フィールドの共有がある場合、なんらかの排他制御が必要である。

２．一度入力したフィールドの訂正を許す。訂正は取消と再入力の二つに分けられる。再入力はただの入力と同じなので、ここでは取消についてだけ考える。入力の取消、すなわはち非未定値を未定値に変更することは、それを入力として起動された一連のアクションが無効となることを意味するので、それらのアクションによって埋められたフィールドを取消する、さらにその取り消されたフィールドを入力としたアクションによって埋められたフィールドをも取り消すという作業を繰り返し行なう必要がある。更に、起動中でまだ値を返していないアクションの入力を取り消す場合、取消作業が終ってから値を返してくることが考えられるので、そのような値を無効にする処理も必要である。また、図３のように出力フィールドを共有している時の取消では、無条件に出力を取り消すことは出来ない。例えば、フィールドＦがアクションＹによって埋められている場合、フィールドＡが取り消されたからといってフィールドＦを取り消せない。

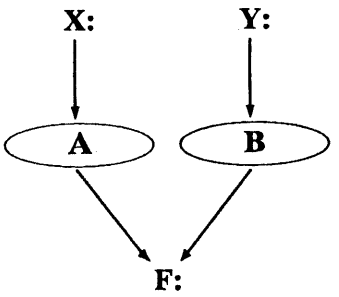

図３．アクションの出力共有

３．コミット型のアクションを許す。これは、一度実行してしまうと取り消すことが出来ないアクションで、このタイプのアクションの実行をするときは、その全ての入力フィールドおよびそれらの値を決めた全てのアクションの全ての入力フィールドも取消不

可にする必要がある。

このような処理を実現するために、Solverのなかでは各フィールド毎に、

・フィールドの値
・そのフィールドを埋めたアクションの識別子（プロセス内のアクションにはユニークな番号がついている）
・そのフィールドを埋めるアクションが起動された時のタイムスタンプ

を管理する。タイムスタンプの値としてはアクションの起動毎に１づつ増えるカウンターの値を使う。

初期状態では、すべてのフィールドの値、アクション識別子、タイムスタンプは未定値という特別な値である。あるアクションが起動出来るのは、その全ての入力フィールドが非未定値であり、その入力に対して未実行であり、全ての出力フィールドのタイムスタンプが未定値の時である。起動に際しては、全ての出力フィールドのタイムスタンプに同じタイムスタンプ値を、アクション識別子としてそのアクションの識別子書き込む。これによって、一つのフィールドに対して同時に複数のアクションが起こることはない。すなわち、タイムスタンプ値の書き込みによってそのフィールドへのライト・ロック制御を行なう。

あるフィールドは、そのタイムスタンプとおなじタイムスタンプをもったアクションからしか埋めることは出来ない。それ以外のアクションからの値は無視する。フィールドの取消は、その値、タイムスタンプ、アクション識別子を未定値にする。同じアクションの異なる呼び出しは異なるタイムスタンプを持つので、アクションの入力フィールドの取消後に、それ以前に起動したアクションによって出力フィールドが埋められることはない。

あるアクションの入力フィールドの取消にともなって、その出力フィールドを取り消す場合、その出力フィールドの中でアクション識別子がそのアクションと一致するものだけを取り消す。従って、他のアクションによって埋められたフィールドを取り消すことは避けられる。

コミット型のアクションの起動に際しては、そのすべての入力フィールド及びそれらのアクション識別子でたどれるアクションのすべての入力フィールドのアクション識別子を"commit"に変更する。"commit"をアクション識別子の値として持つフィールドの取消を許さないとすることでコミットの処理が出来る。

【おわりに】

このようにして構築されたＷＦＭＳを使うとことの利点を整理すると、オフィス・ワーカーにとっては、

・個々のアプリケーションの呼出を知らなくて良い
・個々のアプリケーションへのデータの渡し方、あるいはアプリケーションからのデータのもらい方を知らなくて良い
・訂正に対する必要な再計算・再処理を気にしなくて良い
・各プロセスの状態がわかる（どれをするべきかの判断がし易い）
・処理するべきところがいつもわかる
・処理の中断・再開が簡単で、いつもやるべきところからかじまる

・場合によってはアプリケーションの操作そのものをしなくてすむ
・先に判っていることを入力してしまうなど、操作が自由
・データの管理が一貫していて、省力化される
・他人の仕事との調整が行われる

といった点がある。一方、プロセス・アプリケーションを開発する立場からみると、

・ハード・コードされる部分は、アプリケーションとのインターフェイスの部分だけで、それ以外は簡単な定義で済んでしまうので、開発や修正が容易である
・プログラマではなく、ビジネスの専門化がプロセスの定義をできるので、変化に対応しやすい

などが上げられる。また、管理者からみると、

・すべてのプロセスのデータがオンラインにあるので管理用のデータの収集が容易になる
・オフィス内のプロセスの状況の把握が容易になる

といったことがある。

しかし、ＷＦＭＳが本当にオフィス環境に受入れられ、上述のような利点が享受出来るためには、

・現在のプロトタイプでは実現されていない、WFM/Coordinatorの実現
・Object Lens[1]などでみられるようなオブジェクト・システムとの融合
・サインや印鑑、手書きされた文書を組織や社会の中での意味を失わずにどこまでオンライン化出来るか、出来ない部分をどう補うか
・出来たとして、それが組織や社会に受け入れられるか

といったことが解決される必要があると思われる。

最後に、本研究おいて、有益な議論をしていただいた戸沢、山下、小宮の三氏に感謝します。

【参考文献】

[1] Kum-Yew Lai, Thomas W. Malone, and Ken-Chiang Yu, 'Object Lens: A "Spreadsheet" for Cooperative Work', ACM Trans. of Office Information Systems, Vol.6, No.4, October 1988