ソフトウェア工学 50－1
（1986. 9. 24）

タイミング制御を取入れたデータフロー図に基づく論理モデル作成法

橋本恵二

富士通株式会社　ソフトウェア開発企画本部

オフィス業務のシステム分析と仕様化のためのモデルを提案する。このモデルは通常のデータフロー図にタイミングを取入れたものである。現実の業務システムを，事象とその発生タイミングによって分析すると，従来のデータフロー図による分析よりも容易にシステムの論理的な像を抽出することができる。また，このモデルでは記述の自由度が著しく制限されるため，分析結果の検証も容易になる。

このモデルを用いて実際の大きなシステムを分析する方法を提案する。そこでは，対象とする業務システムをローカルな視点から論理化し，それを段階的に積み上げていく方法を採る。ローカルな視点を統合していく過程は，かなり機械的なモデルの操作によって行えることも示す。

"A Logical Modeling Technique by Extended Data Flow Diagram with Timing" (in Japanese)

by Keiji HASHIMOTO

(Software Development Planning Group, Fujitsu Limited,

1-17-25 Shin-Kamata, Ohta-ku, Tokyo, 144, Japan)

A modeling technique for systems analysis and specification in developing office data processing systems is presented. The model introduces timing into the conventional data flow diagram by representing events which trigger the transformation activities in a system. It is shown that logical aspects of real office procedures can be extracted by analyzing them in terms of events and timing. Since the timing consideration puts an intense restriction on description, the model offers an easy way to verify the described system.

In order to analyze a real-scale office system, a practical method is also considered. Analyzing small parts of the system separately, one can combine them in a relatively mechanical fashion until the entire system is modeled.

1．序論

システムやソフトウェアを開発するに先立って行われる要求分析，要求定義の重要性が認識されて久しい。これまで，要求を記述するための形式的なモデルと技法が種々提案されている[1]。

特に，オフィスにおける業務システムの分析には，階層化されたデータフロー図（ＤＦＤ）を用いた方法がよく知られている[2)3)]。オフィスの業務は，帳票や電話などによる情報の伝達，人間やＯＡ機器による情報の加工，ファイリングのような情報の保管，といった機能で構成されている。このような形態をデータフロー，処理，及びデータストアという概念で抽象化するモデルがＤＦＤである。まず現行の業務をＤＦＤで抽象化し，計算機による処理部分を分析し切り出していくことによって，計算機システムに対する要求仕様を定義していく。

ＤＦＤによる仕様記述において強調される概念は，記述の論理性である。つまり，要求仕様は特定の物理的な実現手段を仮定せずにシステムの論理的側面を記述することを主眼とする。要求仕様に記述されたシステムと論理的に等価なシステムを，現実のハードウェア，ソフトウェア技術を用いて実現する過程はそれ以降の設計フェーズとして区別する。

しかしながら，実際のシステム分析において，システムの物理的要因を識別することは極めてデリケートな問題である。分析者の判断によってかなりの自由度がある。この問題を改善するために，システムの実現手段は“完全”であると仮定してＤＦＤを記述するという提案がある[4)]。

本論文では，このような研究の延長として，ＤＦＤにタイミングを取り入れたモデルとそれによる分析法を考察する。現実の業務システムを，事象とその発生タイミングを手掛りに分析することによって，システムの論理的な構造を比較的容易に導くことができることを示す。またこのモデルでは，従来のＤＦＤよりも記述の自由度が著しく制限されるため，分析の結果を検証することが容易になる。

次に，実際の大規模な業務システムの分析方法を提案する。この方法は，対象とする業務システムを複数のシステムに分割し，段階的な論理化を積み上げることによって対象システム領域全体のモデルに至る方法である。この方法では，ローカルな視点による分析と機械的なモデルの操作によって論理化が行えるため，システム全体を理解している分析者を特に必要としない。

最後に，このモデルによって計算機システムに対する要求仕様を記述する方法を議論する。要求仕様は，計算機の運用を含めた新しい作業の形態がどのようなものかという利用者からの視点と，仕様として記述されたものが論理的に一貫しているという開発者からの視点が共存しなければならない。このような２面性を持つ仕様の形式を考える。

2．システム分析と論理化

実際のオフィスで行われる情報処理活動は，多くの人間が関与する複雑なプロセスから成っている。それは電話による情報のやりとり，帳票の作成，転送，保管などの様々な形態を取る。また複雑な例外処理や２重３重のチェック機構が組み込まれているのが普通である。

このような現行の業務システムを，計算機の導入を前提とした新しいシステムへと再構築するには，現行システムを充分に分析する必要がある。一般に，現実の業務システムには，その機能を担っている人間や機械が持つ技術的な要因，組織や制度上の制約，さらに地理的な要因などが錯綜している。このような要因はシステムを極めて見通しの悪いものにしており，それを“生”で理解するには，膨大な労力とドキュメントが必要となる。システム分析を行う目的は，現実のシステムから上記のような物理的要因を排除し，より単純で理解し易いシステム像を抽出することである。このためには，現実の業務を抽象的なモデルに変換する方法が必要となる。

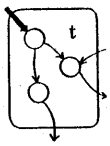
一方，システム分析の結果得られたモデルが，計算機システムに対する要求仕様の原型となるためには，それが物理的な実現手段とは独立に記述されていなければならない。通常，オフィスにおける計算機システムの導入は，従来の手作業による情報処理の機械化や部分的なオフィスオートメーションの統合化という形で行われる。つまり，オフィスにおける現行の業務も，計算機を導入した新しい業務も，論理的に等価なある仮想的なシステムを，別の技術や資源で実現したものと考えられる。システム分析とは，この仮想的システムを，現行システムの論理化という手続きで導く過程とも言える。計算機システムに対する要求仕様は，この論理モデルの上に計算機による処理をマッピングすることによって得られる。

3．オフィス情報処理の論理モデル

ここで考えるモデルは，通常のＤＦＤに事象とその発生タイミングを取入れたものである。表１にモデルの構成要素と表記を定める。

従来のＤＦＤでは，システムを論理的に記述することに対して明確な概念が用意されていない。ここでは，ＤＦＤに，以下で述べるような強い記述の制約を設けることによって，論理性の概念をより明確にすることを試みる。

表1．モデルの構成要素と表記

名　称	記　号	説　明	名　称	記　号	説　明
環　境		システム外にあってシステムと相互作用する主体。 eは識別名。	タスク		複数の処理とデータフローを包含したもの。必ず1つの事象を伴う。 tは識別名。
時　計		定期事象の発生源。 pはその発生周期で，例えばp=D(daily)，W(weekly) 等。	データフロー		レコードの流れ。dはレコード名またはファイルへのアクセスの種別。**)
ファイル		レコードの保管機能。 fは蓄積されるレコード名。	事　象		タスクが起動されるトリガ。外部事象の場合は入力データフローを兼ねる。
活　動 *)		レコードの変換機能。 aは識別名。			

*) 活動は通常のＤＦＤに用いられ，処理や変換とも呼ばれる。ここでのモデルは，タスクをこの機能の基本単位と考える。活動はタスクの意味を捉えるための補助的な道具として用いるだけである。

**) 生成，更新，参照のアクセスを区別する。それぞれ，C(create)，U(update)，R(read) と略記する。

（1）事象

システムに対する事象とは，システム内部の諸活動が起動されるトリガである。事象には外部事象と定期事象の2種類を設定する。外部事象とは，“顧客からの注文の電話”など，システムの外部（環境）からの刺激が，システムに入力情報として与えられたときに起こる。一方，定期事象とは，システム内の活動が前もって約束されたタイミングで自発的に起動されるようなとき発生する事象である。例えば“顧客に毎月ダイレクトメールを発送する”といった活動は定期事象によって起動される。この場合，顧客はシステムに対して指示をするわけではないので，情報の入力はない。

事象の捉え方で注意を要する点は，それが必ず環境から規定されていなければならない，ということである。外部事象は環境からの直接の刺激であるから問題はない。しかし，定期事象に対しては，それが発生する原因が環境から決定されていなければならない。例えば，“顧客に毎月請求書を発行する”という活動が，顧客（環境）からの要請であれば，それは定期事象によって起動される。しかしそれを随時行ってもよい場合は定期事象を設定することは許されない。

このように環境からシステムの振舞を規定するのは，それ以外の要素はシステムの実現手段に依存しているからである。例えば，帳票をシステム内の部署AからBに毎日郵送するような活動は，郵送という物理的手段に伴うコスト要因から，この業務を定期的に行っていると考えられる。活動が起動される原因を環境に求めることによって，このような物理的要因をモデルから排除することができる。

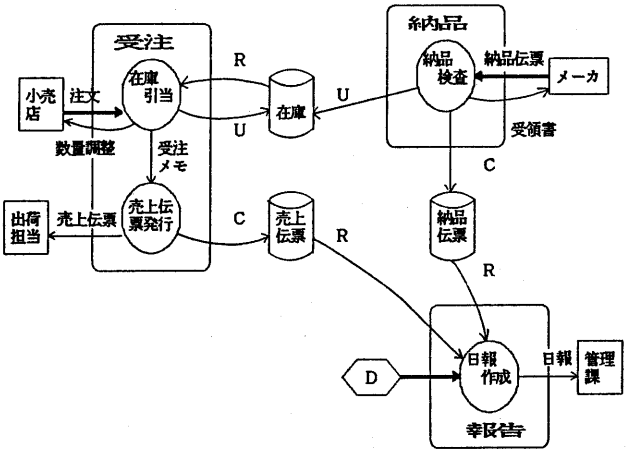
（2）タスク

タスクとは，事象によって起動される一連の活動をまとめたものである。1つの事象が発生したとき，それをシステム内のある活動が関知し，それがデータフローを媒介とした連鎖反応を引起こすと考える。この一連の連鎖反応を，機能の最小単位と考え，タスクと呼ぶ。タスクは事象による応答（ファイルや環境への情報出力）を終えると再び事象を待つ状態に戻る。

ここで，タスクに対して次の仮定を設定する。つまりタスクの処理時間を無限小とし，処理の誤りは全くないものとする。前者の仮定は，オフィスの計算機システムにとっての環境は，人間の動作であり，それに比べて計算機の処理速度は充分に速いという事実に基づく近似で

図1．システムの記述例

“受注”，“納品”というタスクはそれぞれ小売店，メーカという環境からの外部事象によって起動される。一方，タスク“報告”は管理課からの要請によって，毎日の伝票類を集計して送付する。それぞれのタスクは固有の事象を持ち，必然的にタスクはファイルを通じて他のタスクと相互作用する。



ある。後者の仮定を行うのは，処理の誤りは活動を担う主体に付随する現象だからである。一般に，現実の業務は処理誤りに対するチェック機構が複雑に組み込まれていてシステムの見通しを悪くしている。このような要素を排除するために，システム内部の処理を完全とみなす。

（3）ファイル

タスクはそれぞれ固有のタイミングで起動されるため，タスク間の情報のやりとりは，情報を有限時間保管するファイルを通じて行われる。

ファイル内のレコード編成やそれに伴うアクセス法は想定しない。ここではファイルは“理想的”なＤＢＭＳによって管理され，任意のタスクから，任意の項目をキーとしてアクセスできるものとする。またアクセスに要する時間は，タスクに対して仮定したように，無限小とする。

事象やタスクは，環境を定義すれば一意に決定される。しかし，ファイルの分割の基準は環境からの要請では定まらない。そのため，ファイルの設定単位を，次のように定める。１つのファイルレコードを作成するタスクは必ず１つとし，タスクは高々１つのファイルレコードしか作成できないものとする。タスクは固有の事象を持つため，実世界の出来事と１対１に対応する。このようにファイルの単位を設定すると，システムが蓄える情報と実世界の出来事の対応が良くなる。

以上のような構成要素で業務システムを記述した例が図１である。

（4）モデルの構造と検証

以上のような制約を，モデルの形式的なルールとして表現すると次のようになる：

①環境，タスク，ファイルはそれぞれの間で相互作用しない。

②タスクを起動する事象は１つ有り，１つに限る。

③ファイルレコードの生成アクセスは必ず１つ，参照アクセスは少なくとも１つなければならない。

③タスクは必ず環境またはファイルへの情報出力を行う。特に，定期事象によって起動されるタスクは，環境への出力を持たねばならない。

ここで③の制約は，定期事象というものが環境からの要請で定められていることから来る。この制約は，後に述べるように，システムの物理的な性質を除去する上で重要な役割を果たす。

結局，モデルは図２に示すような，『環境－タスク－ファイル』という３層の構造に帰着する。この構造はシステム全体に渡って一様であり，言わばシステムの“フ

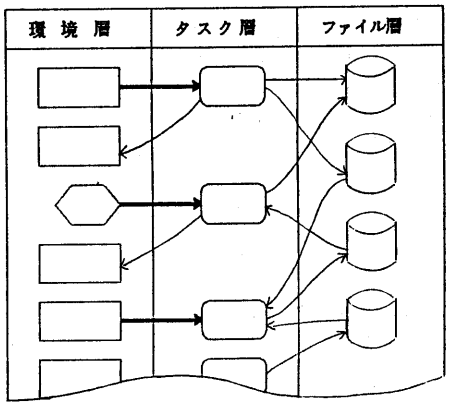


図２．システムの『環境－タスク－ファイル』構造

それぞれの層内の要素は互いに相互作用を行わず，独立に振舞う。この３層の構造はシステム内を通して一様であり，“フラット”なシステム像を強調する。

ラット”な像を強調する。従来のＤＦＤではシステムの理解を助けるために，階層化が重要な役割を果たしている。しかし，階層化のしかたは分析者の判断に委ねられているため，その自由度が大きい。特に，ＤＦＤをどこまで詳細化するかについては全く基準がない。ここでは，実世界の出来事の発生タイミングから規定されるタスクを変換機能の最小単位と考えることによって，歯止めをかけている。タスクは互いに独立性が高いため，モデルに曖昧な抽象化や階層化の概念を登場させる必要はない。

このようなモデルの形式的な構造以外に，モデルの意味的な正しさを検証することが必要になる。次節で述べる分析過程では，モデルの操作が中心になるため，このことは特に重要である。

モデルの意味的な正しさの検証は，次のような断片的な確認を繰り返すことによって行うことができる。まず個々のタスクの正しさとは，

・入力レコードから出力レコードのすべての項目値が得られる。
・ファイルの参照，更新を行う場合，ファイル内のレコードを特定する情報が与えられている。
・タスクの処理が条件によって異なる場合，その条件を判定する情報が与えられている。

一方，ファイルに対する検証を行うには，個々のファイルとそれと相互作用するタスクに注目して次のことを確認する：

・ファイル内のレコードに対する生成，更新，参照アクセス間の時間的な順序関係に矛盾が生じないような事象の発生タイミングが，環境側で保証されている。

この検証は，タスクの処理時間を無限小としているために，事象の発生タイミングの順序だけが問題となる。

4．システムの分析法

対象とする業務システムを，前節で述べたような論理モデルに置き換える方法を考える。

現実の大きなシステムの分析を行うとき，システム全体を最初から理解しなければならないような方法では，実際に分析を行うことが不可能になる。特に複数の業務にまたがるシステムを考えるとき，その全体を把握できる人間はいないのが普通である。このような場合にも耐えられる分析法を考える必要がある。

ここでは，対象とする業務領域を既存の組織単位や場所によって小さなローカルシステムに分割し，それぞれ独立に論理化することによって，システム全体のモデルに至る方法を提案する。

（1）ローカルシステムの論理化

前節で述べたように，ここでのモデルは，その環境によって規定される。１つのローカルシステムに注目したとき，それが相互作用する主体は，対象システムにとっての環境以外に，システム内の他のローカルシステムもある。前者を外部環境，後者を内部環境と呼んで区別しよう。

分析を行うには，外部環境も内部環境も等しく環境とみて，個々のローカルシステムの業務を前節のモデルに置換える。そこでの手順は基本的に次のようになる：

①システム外部から入力されるデータを外部事象とする。
②定期事象は必ず環境への情報出力を伴うため，それを出力するタイミングが環境からの要請から来ているかどうかを考える。もしそうなら，出力を行う活動に定期事象を付ける。
③外部事象に対して，それによって論理的に実行可能な活動を情報の流れに沿って追っていく。定期事象に対して，情報出力を行う活動，それの直前に行うことが可能な活動を情報の流れを遡って追う。このようにしてまとめられた活動群を１つのタスクとする。
④以上の手順でまだ所属の定まらない活動が存在すれば，それは情報の紛失や処理誤りに備えた活動と考えられるので，それをモデルから除去する。
⑤タスクの間を流れるデータをファイルに設定する。

もっとも，実際の問題の分析には，業務の内容の詳しい調査と業務の精通者の協力が必要となる。そのためにも分析の単位となるローカルシステムは充分小さく設定する必要がある。

（2）ローカルシステムの結合とタスクの縮約

これまでローカルシステム毎に設定したタスクは，内部環境からの制約で小さく設定されている。この制約が取除かれると，事象による活動の連鎖反応がより広範囲に拡大され，別に設定されていたタスクが，より少数のタスクに縮約される。また，ローカルな視点では必要であった活動が，よりグローバルな視点からは物理的な活動であることが発見され，消滅するタスクもある。

この操作を行うには，次の手順を踏む：

①結合すべきローカルシステム間で相互作用するタスク対を繋いで，仮のタスクを作る。
②仮タスクがアクセスするファイルで共通のものがあればそれを１つにする。この時点で仮タスクが消滅することがある。
③仮タスクが外部事象によるものであれば，それが求めるタスクとなる。

④仮タスクが定期事象によるものであり，かつその発生タイミングがその時点での環境に規定されているならば，仮タスクは求めるタスクである。

⑤それ以外の場合，仮タスクを起動する事象が何かを考える。それは必ずシステム内の他のタスクに求められるはずである。

このようなタスクの縮約は，筆者の経験では，モデルの形態からかなり機械的に行うことができる。縮約の典型的なパターンを図3に示す。

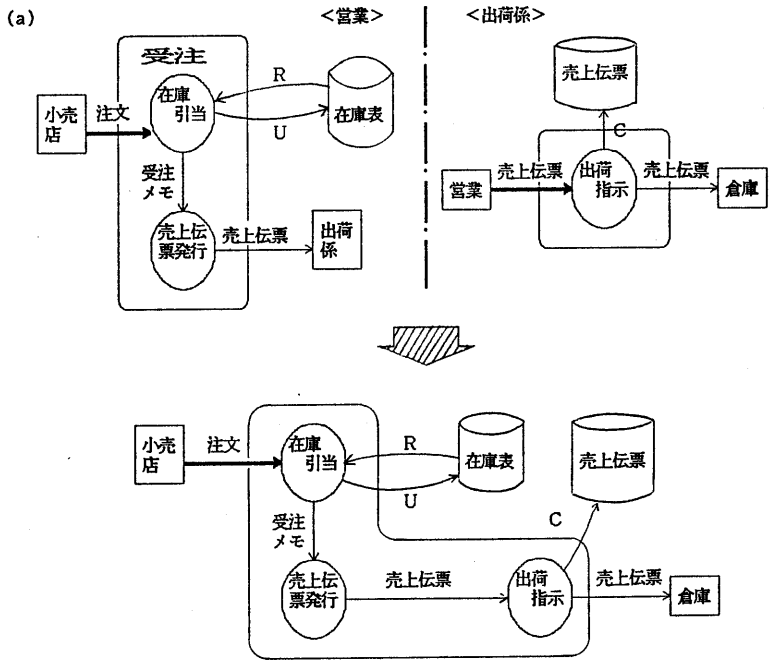
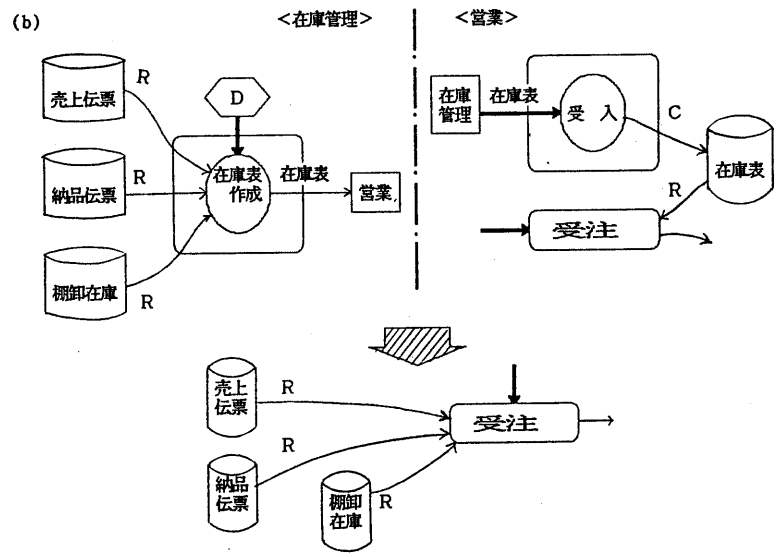


図3．タスクの縮約パターン

(a)ローカルシステム営業の受注というタスクが出力する売上伝票によって，他のローカルシステム出荷係の出荷指示という活動が起動される。2つのローカルシステムを結合させると，2つのタスクは論理的に一連の活動となり，1つのタスクに縮約される。



(b)ローカルシステム在庫管理は毎日在庫表を作成して営業に送付する。ローカルシステム営業はそれをプールし，受注というタスクで用いる。両者を結合させると，在庫表を毎日送付する理由がもはや環境から規定されなくなるため，これらの活動は物理的であることが分かる。このパターンは，在庫表が冗長なデータであることに起因するタスクの縮約であると考えられる。

(c)営業が毎日売上伝票を総務に送付し，総務はそれに基づいて請求書を発行する。これらを結合させると，中間の図のような形になる。しかし，もし請求書を毎日送付することが環境である出納係からの要請でないならば，このタスクの事象を他に求めなければならない。今の例では売上伝票を発行する受注というタスクがその事象を持っていると考えられるので，請求書を発行するタスクは受注に吸収される。2つの部署の間の伝票送付が，例えば社内メールのような物理的な手段で行われていることによって，送付というタスクが存在していたと考えられる。

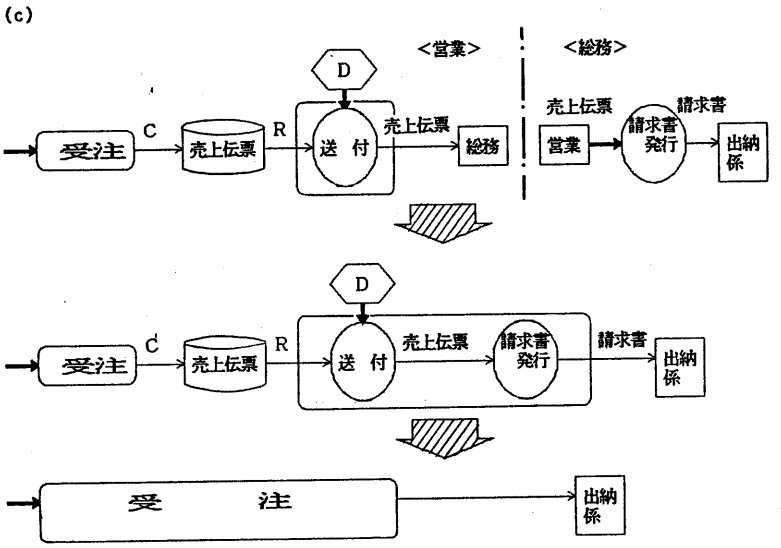
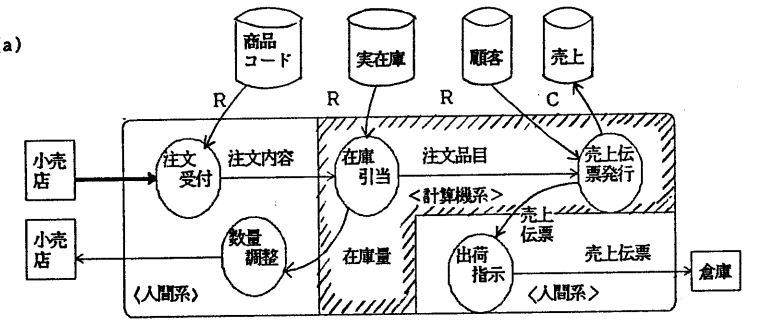
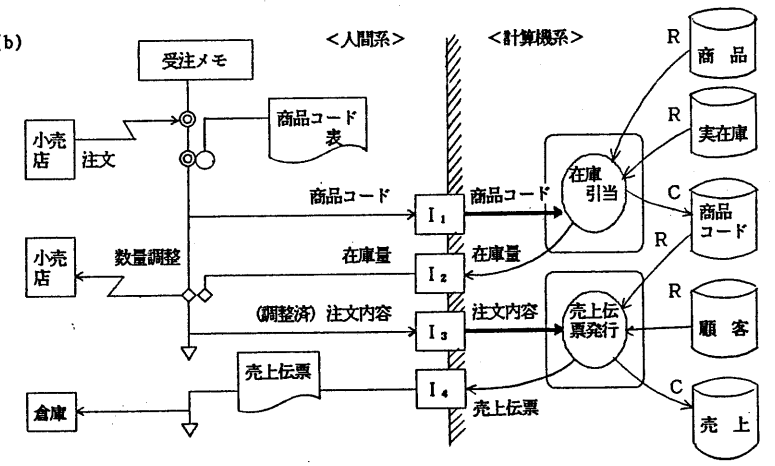


図4．要求仕様の形式

(a)1つのタスクの中で計算機化すべき領域を定義する。2つの領域の境界を横切るデータフローは，マンマシンインターフェースに対応する。



(b)インターフェースを再び内部環境とみなし，論理化の範囲を縮小する。このとき，計算機系のモデルは再編成される。一方，人間系には利用者にとって関心のある新しい業務形態を記述する必要がある。ここでは通常の事務フロー図で人間系を記述している。



以上のような方法によって，原理的にはどのような大きなシステムも分析可能である。また，もし分析すべき対象システムの範囲が最初から明確でない場合も，必要なローカルシステムを適宜追加することによって，柔軟な分析が可能になる。

5．要求仕様の形式

上のモデルで記述された業務システムは，いわば開発されるべき計算機システムに対する要求仕様の原型である。システムの論理化は，利用者の諸々の要求を反映できるように充分広く行う必要がある。しかし，そのモデルはすべて計算機によって実現されるわけでなく，実際には，その中で計算機システムによる処理部分と，人間による運用系とが，コストその他の要因によって決定される。

計算機システムに注目したとき，人間による運用系は環境と見なすことができる。つまり，計算機システムに対する要求仕様を作成するには，今までのモデルの上にマンマシンインターフェースを設定し，それを改めて内部環境と考えて，論理化の範囲を縮小する，という操作が必要になる。

一方，人間系に対して，我々は再び物理的な記述が必要になる。計算機の利用者にとっての関心事項は，計算機の運用を含めた新しい作業の形態だからである。そこでは，実際の作業の手順や連続的な時間の流れという概念が復活する。

このように，２面的な仕様の記述の例を図４に示す。利用者の視点は，自らの作業の中から，計算機とのインターフェースを見るのが自然である。そこでは，タスクなどの概念は登場せず，その組織で用いられている事務フロー図などを用いるのが有効であろう。一方，開発者にとっては，計算機側からインターフェースを見るのが自然である。設定されたインターフェースをすべて満足することが，少なくとも論理的に可能かどうかが重要になる。このことは，今までに述べたモデルと検証法によって，比較的容易に保証することが可能になろう。

6．結論

オフィスシステムをモデル化するのに，システム内部の実現手段の完全性とタイミングを手掛りに分析を行うと，従来のＤＦＤに比べて論理性の概念が明確になる。また記述の自由度が強く制限されるために，書かれたものの理解し易さが向上する。また，モデルの検証を従来よりもシステマティックに行うことが可能になると考えられる。

大きなシステムの論理化を，ローカルな分析の積み上げによってかなり機械的に行える点もこのモデルの特徴になっている。これは環境から一意に規定されるタスクという明確な概念を用いていることによる。

このモデルによる要求仕様は，モデルの範囲を計算機システム領域に設定しなおすことによって得られる。そこで設定された多くのインターフェースに対して，それらの間の論理的な一貫性は今のモデルによって比較的容易に保証することができるであろう。

ここで提示した方法は，かなり極端な視点からシステムをモデル化しているため，抽象度が高い。しかし，最終的なモデルは，実時間オンラインシステムの実現モデルとしても我々のイメージに，少なくとも従来の方法より近づいていると思われる。

現在，この方法の支援環境をグラフィックツールの上で構築している。

謝辞：この論文を完成するのに指導いただいた，岡本室長，村上課長，宮成の諸氏に感謝致します。
また，研究当初から貴重な助言をいただいた森課長，平の両氏に心から謝意を表します。

参考文献

1）例えば，野木兼六：要求定義技術の動向，情報処理，Vol. 20, No. 6, 487-494 (1979)；
要求定義技術の最近の動向，情報処理，Vol. 27, No. 1, 21-30 (1986).
2）DeMarco, T. : Structured Analysis and System Specification, Prentice-Hall, 1979.
3）Gane, C. and Sarson, T. : Structured Systems Analysis : Tools and Techniques, Pretice-Hall, 1979.
4）McMenamine, S. M. and Palmer, J. F. : Essential Systems Analysis, Yourdon Press, 1984.