ソフトウェア工学 122－4
（1999．3．18）

ユーザタスクの形式的記述に基づく インタラクティブシステム設計法の提案

池田瑞穂　高田喜朗　関 浩之

奈良先端科学技術大学院大学 情報科学研究科

現在アプリケーション開発工程において，ユーザインタフェース部分の開発の占める割合が大きくなっている．しかし，従来の設計アプローチには，設計のガイドラインが断片的で設計方法論として体系化されていないなどの問題がある．

本研究では，設計の上流工程から適用できる，体系化されたインタラクティブシステムの設計法を提案する．この設計法では，ユーザタスクの形式的記述を一連の設計プロセスへの入力とする．そして，前段階の仕様を満たすような変換に基づいて詳細化する．仕様に現れるイベントをユーザが行うものとシステムが行うものに分類し，それらが使うUI部品などを指定することで，プロトタイプを自動生成する．

本稿では「通信販売による本の購入」を例題として本手法の説明を行う．

An Interactive System Design Method Based on a Fomal Specification of a User's Task

Mizuho Ikeda　Yoshiaki Takata　Hiroyuki Seki

Graduate School of Information Science
Nara Institute of Science and Technology

In development of an application system, designing the user interface comes to occupy a larger part. Various techniques have been proposed and are used for interactive system design. However, it is often pointed out that a design process of an interactive system highly depends on designers' experience and lacks a systematic methodology.

In this paper, a systematic method of interactive system design is proposed. First, the user's task is specified as a task flow diagram. The diagram is refined by decomposing each task into several subtasks which represent system's actions and user's ones. A prototype can be generated automatically from the detailed task diagram augmented by information on user interface. In the paper the proposed method is explained by using "Book purchasing problem."

1　まえがき

数年前まで，情報システム開発の対象は定型的な業務の支援が大半であった．そのため解決すべき問題の構造が明確で，伝統的なソフトウェア設計法が有効であった．また，メモリ使用量や計算時間などを最適化してプログラムの効率性を高めることを目標に，アプリケーション開発が行われていた．しかし昨今では，計算機の目覚しい普及により，アプリケーションの開発環境，利用目的，利用環境などが大きく変化してきた．ソフトウェアが多機能化し，また，ユーザのニーズも多様化してきた．それに伴って現在，アプリケーション開発工程において，ユーザインタフェース部分の開発の占める割合が大きくなっている．そのようなインタラクティブシステムの開発には，特有のさまざまな設計技法が用いられる [1, 7].

しかし，従来の設計アプローチには以下のような問題がある．

- 断片的な設計ガイドラインが与えられているだけで，設計方法論として体系化されていない．
- 要求定義，ユーザ挙動モデル，メンタルモデルなどが場当たり的に用いられ，全体の整合性がない．
- プロトタイプ作成とそれを使った実験の繰り返しのみで設計を改善していくのは開発コストが大きい．

これらの原因から，アプリケーションの品質は設計者や開発者の経験や感性に依存しがちで，ユーザの要求を満たしたものにならないことが現実に多く起こっている．

そこで本研究では，設計の上流工程から適用できる体系化されたインタラクティブシステムの設計法を提案する．提案する設計法は以下の２つの特徴を持つ．

(1) **仕様の形式的記述と詳細化．**
初期仕様を形式的記述法を用いて記述し，できるだけ前段階の仕様を満たすように詳細化する．これにより詳細化前後の不整合を起きにくくする．また，仕様を形式的に記述することで，プロトタイプの自動生成や仕様の形式的検証が可能となり，設計作業の効率化を計ることができる．

(2) **ユーザタスクの記述を初期仕様とする．**
ユーザタスクの形式的記述を，上記の詳細化プロセスへの入力とする．「インタラクティブシステムは，ユーザが行いたいタスクをユーザと協調して実行するためのものである」と考えると，その設計の初期段階はユーザタスクの構造や流れの記述とするのが自然である．

本稿では「通信販売による本の購入」を例題として本設計法の説明を行う．

1.1 関連研究

形式的記述をユーザインタフェース設計に応用する研究は多くなされているが，現実のシステム設計において仕様を記述する目的に使えるものは少ない．

GUI（グラフィカルユーザインタフェース）の仕様を形式的に与えることでその形式的検証とプロトタイプ自動生成を行う方法が研究されている [8]．これは，上述の特徴 (1) に関して本研究と同じ目的を持つものである．しかし [8] では，仕様を作成する段階について特に方法論が与えられていないという問題がある．導入されている仕様記述言語は，通常のプログラミング言語風で直観的に理解しにくく，初期段階の仕様の作成には向かない．

また，[8] を含め従来のほとんどの設計手法には，「ユーザの行いたいタスクの構造を記述し，それに基づいて設計する」という考えが含まれていない．これらの設計法の多くでは，まずシステムに要求されている各機能を実現し，それらの機能の実行に必要な情報をユーザから与えてもらうことを目的として，インタフェースの設計が行われる．しかし，上

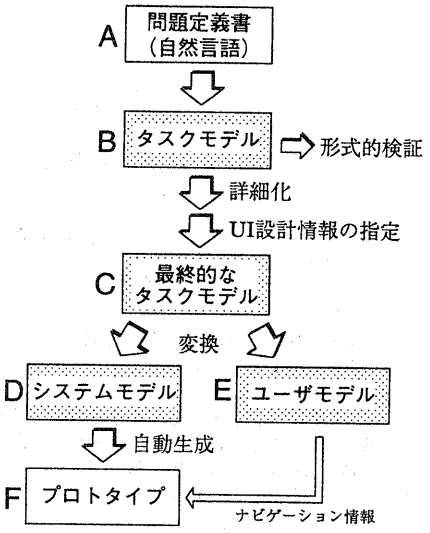

図 1: 提案する設計法

の特徴 (2) で説明したとおり，ユーザインタフェースの設計は本来，ユーザの行いたいタスクの構造や流れに着目して行われるべきである．

ユーザの振る舞いの形式的モデルを設計に利用する試みとして [6] がある．これは，ユーザおよびシステムのモデルをそれぞれペトリネットで記述し，ユーザがシステムを使用する際に起こる不具合をモデル上で検出する，という枠組について述べたものである．しかし，[6] は設計法として体系化されておらず，またペトリネットはタスク構造を自然に書き表す目的には適していない．本研究では，ユーザタスクの構造の形式的記述に基づく，体系化された設計法を提案する．

2 提案する設計法の概略

図 1 に本設計法の概略を示す．ここで，矩形は作成されるドキュメント，矢印はドキュメント間の変換処理を表す．

本設計法における設計手順は以下のようになる．

(1) 問題定義を記述する．
(2) **タスクモデル**を作成する．…タスクとサブタスクの関係や実行手順などのタスクの構造を記述する．これには 4 節で定義する**タスク図**と呼ぶ形式的記述法を用いる．
(3) タスクモデルを詳細化する．…各タスクをそれ以上分解できないレベルまで繰り返し詳細化す

る．それ以上分解できないタスクをイベントと呼ぶ．

(4) ユーザインタフェース（UI）設計情報を指定する．…タスクモデルと現実の入出力操作や操作画面との対応を決めるため，必要な情報を指定する．

(5) システムモデルを導出する．…タスクモデルにおいて，ユーザが行うイベントを受信，システムが行うイベントを送信と考えることで，システムモデルを得る．

(6) プロトタイプを自動生成する．

また，ステップ (2) や (3) の後に以下を行う．

(7) タスクモデルに対し，形式的検証を行う．…満たすべき性質を時相論理式で記述し，作成されたタスクモデルがその性質を満たすかどうか形式的に検証する．

以下では「通信販売による本の購入」を例題として，上記の各ステップを説明する．

3 設計手順

3.1 問題定義書の作成

問題の定義を自然言語で記述する．すなわち，設計するシステムの目的や必要とされる機能，対象とするユーザなどを記述する．

この例では以下のようになる．

> ユーザが目的の書籍を見つけ，注文し，入手できるような計算機システムを作成する．対象とするユーザは，WWW を利用でき，通信販売を利用したことがあるとする．

3.2 初期タスクモデルの作成

問題定義を基に，タスクの構造を表現したタスクモデルを作成する．これにはタスク図という記法を使用する（4 節）．

図 2 はこの例におけるタスクモデルの一部である．ここでは，「通信販売で本を買う」タスクは，「本と冊数を指定する」「発送先を指定する」「支払方法を指定する」の 3 タスクの並行実行で実現できることが記述されている．また，「本と冊数を指定する」タスクは，検索によって「本を探し」た後「冊数を指定する」ことで実現されること，「冊数を指定」した後，任意の回数「冊数を変更」できること，なども記述されている．

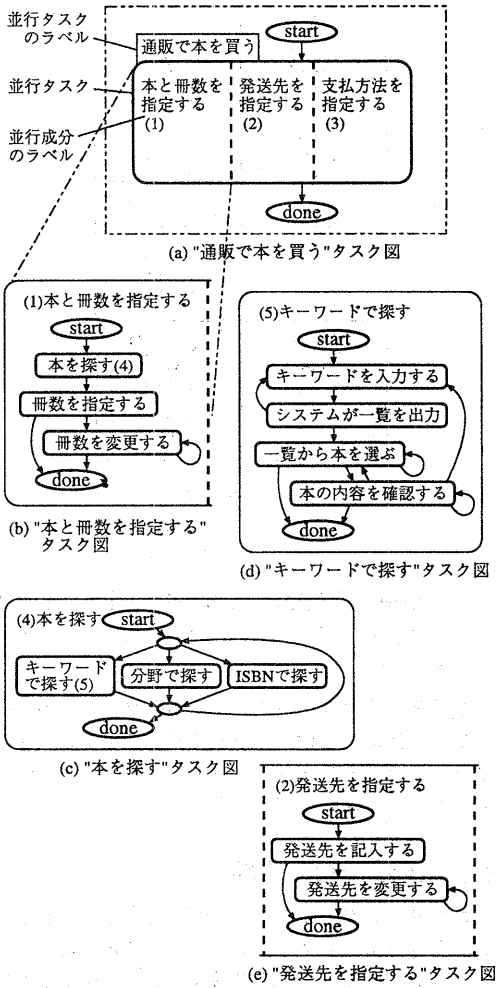

図 2: タスクモデル

このように，タスク図はタスクを階層的に表現できるので，概略を記述した後に各部分を記述していくような段階的な詳細化が行いやすい．

3.3 タスクモデルの詳細化

記述されたタスクモデルを詳細化していく．これは，例えば以下の (1) または (2) の操作を繰り返すことによって行う．

(1) 原始タスク（それ以上分割されていないタスク）をタスク図または並行タスクに書き換える．

(2) 図 3 のように頂点を追加する．この図は 3 通りの追加の方法を示しており，左側が追加前，

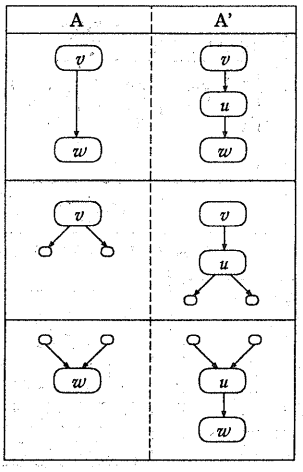

図 3: 頂点の追加

右が追加後を表す．すなわち，頂点 u を追加して，左のような部分グラフを右のように書き換える．

最終的に，タスクモデル中の各原始タスクがユーザやシステムの 1 動作となるまで詳細化する．このような各原始タスク（ユーザやシステムの 1 動作）をイベントと呼ぶ．

3.4 UI 設計情報の指定

ユーザインタフェース（UI）部分を実装するために必要な情報をタスクモデル上に指定する．

システムとユーザの間で送受信が行われるイベントを外部イベントと呼ぶ．外部イベントにおける送受信は，入力欄やボタンなどの UI 部品を通じて行われる．そこで，各外部イベントについて，それが使用する UI 部品を設計者が与える．また，UI 部品を表示するための画面の一領域をフレームと呼ぶことにし，いくつかのサブタスクにフレームを静的に割り当てる．フレーム上に表示される UI 部品を一定の規則に従って決め，またフレームの表示開始時点なども規則的に決めることで，システムが実行の各時点で表示すべきユーザインタフェースが定まる．

このステップは以下の 2 つの作業からなる．

(1) イベントを，ユーザイベントとシステムイベント，および外部イベントと内部イベントに分類する．

(2) 外部イベントが使う UI 部品の指定や，サブタスクへのフレームの割り当てを行う．

以下，これらについて説明する．

3.4.1 イベントの分類

イベントにはユーザの意思決定によるものと自動化できるものの双方が含まれるので，イベントをユーザが行うものとシステムが行うものに分類する．これらをそれぞれユーザイベント，システムイベントと呼ぶ．さらにそれらを，ユーザとシステムの間で送受信が行われる外部イベントと，それ以外の内部イベントに分類する．すなわち，タスクモデル中の各イベントを以下の 4 つに分類する．

- ユーザ外部イベント
  … e.g.,「検索キーワードを入力する」
- ユーザ内部イベント
- システム外部イベント
  … e.g.,「検索結果の一覧を表示する」
- システム内部イベント
  … e.g.,「データベース内を検索する」

それぞれの集合を E^{ext}_U, E^{int}_U, E^{ext}_S, E^{int}_S とする．

3.4.2 UI 部品の指定とフレームの割り当て

タスクモデル中の外部イベントの集合を E^{ext} とする．各外部イベント $t \in E^{\mathrm{ext}}$ に対して，t が使う UI 部品を設計者が与える．t が使う UI 部品の集合を $w(t)$ とする．

ただし，利用できる UI 部品の仕様にタスクモデルが適合していなければ，タスクモデルを変更する必要がある．これは主にイベントの挿入や置き換えなどで行われ，タスクの構造を大きく変えることはないと考えられる．図 4 は，図 2 のタスクモデルに対する UI 設計情報の指定の例である．この例では 3.4.3 節で述べる方針に従って指定作業が行われている．

設計者は，ラベルを持つ任意のタスクにフレームを持たせることができる．t が持つフレームを $f(t)$ とし，t がフレームを持たないとき $f(t) = \perp$ とする．

ここで，各 UI 部品がどのフレームに属するかという関係を以下のように定義する．

定義 1 (フレームに属する UI 部品) *UI* 部品 w とタスク t との関係 belongsto を以下のように定義する．

w belongsto t
$\Leftrightarrow t \in E^{\mathrm{ext}} \wedge w \in w(t)$
$\vee\ \exists t' \in V[t] \wedge f(t') \neq \perp \wedge w =$ ボタン to$f(t')$
$\vee\ \exists t' \in V[t] \wedge f(t') = \perp \wedge w$ belongsto t'.

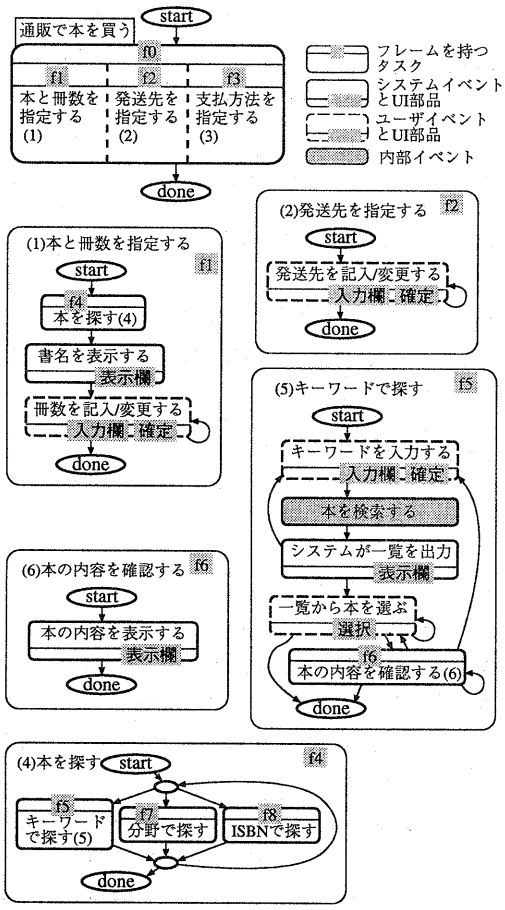

図 4: UI 設計情報を加えたタスクモデル

ただし，ボタン to$f(t')$ はフレーム $f(t')$ の表示開始を指示するための *UI* 部品である．$V[t]$ は t のサブタスクの集合である（定義 6）．

UI 部品 w がフレーム f に属することを $w$ison f と書き，以下のように定義する．

$$w \text{ ison } f \Leftrightarrow \exists t \wedge w \text{ belongsto } t \wedge f = f(t).$$

□

ただし，以下の仮定が成り立つよう UI 設計情報が与えられるとする．

仮定 2 タスクモデル T 中の任意の外部イベント $t \in E^{\text{ext}}$ について，t が使う *UI* 部品 $w \in w(t)$ は，T 中のいずれかのタスクが持つフレームに属す．すなわち，

$$\forall w \in w(t) \Rightarrow \exists t' \in V^*[T] \wedge w \text{ ison } f(t').$$

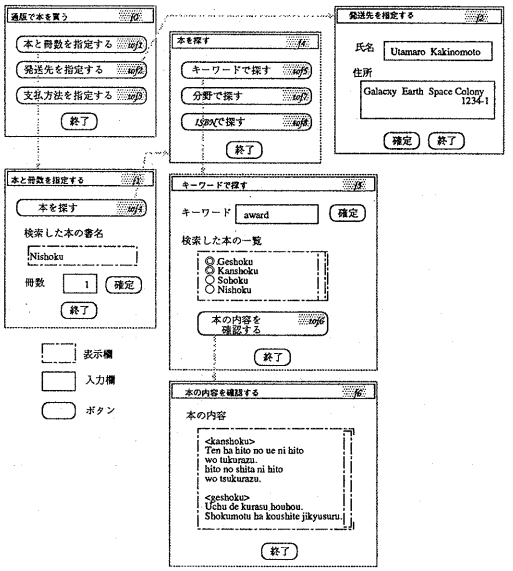

図 5: プロトタイプのユーザインタフェース

ただし，$V^*[T]$ は T の子孫タスクの集合である（定義 7）．□

指定された UI 設計情報の意味は，「ユーザおよびシステムがタスク t を実行している間，フレーム $f(t)$ および $f(t)$ に属する UI 部品が表示される」と定義される．これに合わせて，以降のステップにおいて，プロトタイプ中にフレーム表示開始等の処理がプログラムされる（図 5）．

3.4.3 UI 設計情報を指定する手順

以下の規則に従うことで，UI 設計情報の指定をほぼ自動的に行える．

(1) 各タスク図および並行タスクにそれぞれフレームを割り当てる．

(2) 「キーワードを入力する」など，ユーザからの値の入力に対応するタスクは，

- データの書き込み
- 確定操作

の 2 操作から構成されるので，このタスクに入力欄と確定ボタンを割り当てる．また，入力欄の大きさや書き込み内容の初期値は設計者が指定する．

(3) 「一覧から本を選択する」など，選択肢の中からユーザが選択を行うタスクには，以下のどちらかの UI 部品を割り当てる．

- チェックボックス集合（各要素それぞれ独立にセットまたはリセットできる）
- ラジオボックス集合（要素のうち1個のみセットできる）

(4) 表示を行うタスクに表示欄を割り当てる．表示欄の大きさは設計者が指定する．

(5) done にフレームを終了するボタンを割り当てる．

図4は，以上の規則に従ってUI設計情報が指定されたタスクモデルである．ここでは頂点の内部を3区画に分け，上部にフレーム名，下部にそのタスクが使うUI部品の種類を記述している．done に割り当てられる終了ボタンは省略されている．実線で枠を描かれた頂点はシステムイベント，点線のものはユーザイベントを表す．また，中を塗りつぶされた頂点は内部イベントを表す．

3.5 システムモデルおよびユーザモデルの導出

記述されたタスクモデルは，ユーザとシステムの両者を成分に持つ協調システムの仕様を与えていると考えることができる．つまり，両者が通信し合って作業を行うことで，記述されたタスクが実現される．

このとき，以下の規則によって，システムの挙動を表すシステムモデルと，ユーザの挙動を表すユーザモデルが得られる．

- タスクモデルにおいて，システム外部イベントを送信イベント，ユーザ外部イベントを受信イベントとしたものをシステムモデルとする．
- ユーザ外部イベントを送信イベント，システム外部イベントを受信イベントとしたものをユーザモデルとする．

ここではユーザモデルを実現したものがユーザであると考えて，このステップ以降ではシステムモデルを実現するための作業が行われる．

システムモデルおよびユーザモデルの導出を厳密に書くと以下のようになる．

タスクモデルからシステムモデルを導出する:

(1) ユーザ内部イベント $e_U \in E_U^{\mathrm{int}}$ を削除する．

(2) システム外部イベントを送信イベントに指定する．すなわち，4節で述べるタスク図のCCS記述において，イベント $e_S \in E_S^{\mathrm{ext}}$ を $\overline{e_S}$ に書き換える．

タスクモデルからユーザモデルを導出する:

(1) システム内部イベント $e_S \in E_S^{\mathrm{int}}$ を削除する．

(2) ユーザ外部イベントを送信イベントに指定する．

このとき，CCSで表現されたタスクモデル，システムモデル，ユーザモデルをそれぞれ T, S, U とすると，プロセス T が満たす性質はプロセス $(S \mid U)$ も満たすという関係が成り立つ．

3.6 プロトタイプの自動生成

システムモデルおよびUI設計情報に基づいてプロトタイプを自動生成する．すなわち，フレームおよびUI部品の表示や，ユーザとの間の入出力を制御するプログラムを生成する．

図5は，図4に基づくプロトタイプが表示するフレームおよびUI部品を示している．

3.7 形式的検証

得られたタスクモデルに対し，ユーザビリティに関する性質を満たしているかどうか形式的に検証する．ここでは以下のような手順で検証を行う．

(1) 検証したい以下のような性質 ϕ を時相論理式で記述する．

- ユーザはいつでもタスクを中断できる．
- ユーザは何回でも本を検索できる．
- ユーザはいつでも，いつかは発送先を変更することができる．

(2) タスクモデル T の下で ϕ が成り立つ，すなわち $T \models \phi$ であることを，モデルチェッキング[4]などの手法を用いて検証する．

4 タスク図

タスクモデルの記述およびその後の詳細化のために，タスク図という図的言語を導入する．

定義3 (タスク図) タスク図は，有向グラフ $G = (V, A)$ である．頂点 $v \in V$ は次の5つのうちいずれかである．

- 原始タスク
- 並行タスク
- タスク図
- start
- done

ただし，start, done はそれぞれちょうど *1* 個 V に含まれる．start の入次数および done の出次数は *0* である．また，各頂点 $v \in V$ について，start から v へのパスおよび v から done へのパスが存在する．
□

定義 4 (タスクモデル) タスクモデルはタスク図で与えられる．□

タスク図において，各頂点は全体のタスクを構成するサブタスクを表し，辺はその実行順序を表している．start は実行開始点，done は終了点を表す．すなわちタスク図は，start から done までの任意のパス（同じ頂点を複数回含んでもよい）について，そのパスに沿ってサブタスクを行うことで全体のタスクが実現できることを表している．

タスク図の記述例を図 2 に示す．角の丸い長方形および楕円が頂点を表し，矢印が有向辺を表す．頂点に図を埋め込めることによる階層構造の表現，および並行成分からなる頂点（並行タスク）の表現は，ステートチャート [2] にならっている．

並行タスク　並行タスクは，複数の並行成分からなるタスクを表す．

定義 5 (並行タスク) 並行タスクは，原始タスク，並行タスク，タスク図を要素とする空でない集合である．□

図中では並行成分を破線で区切って表す．図 2 において，「通信販売で本を買う」タスクは，「本と冊数を指定する」「発送先を指定する」「支払方法を指定する」の 3 タスクを成分とする並行タスクである．

階層構造　タスク図や並行タスクを頂点とすることでタスクの階層構造を表現できる．図 2 の「本と冊数を指定する」タスクは，「本を探す」「冊数を指定する」などを頂点とするタスク図である．記述の際は，頂点や並行成分を表す領域内にタスク図を埋め込んで描いてもよいし，親図と分けて別の領域に描いてもよい．

定義 6 (タスク，サブタスク) タスクとは，原始タスク，タスク図，並行タスクのいずれかとする．また，タスク図 t の頂点および並行タスク t' の成分をそれぞれ t, t' のサブタスクと呼ぶ．

タスク t に対し，サブタスクの集合 $V[t]$ を以下のように定義する．

$$V[t] = \begin{cases} V_t - \{\text{start}, \text{done}\} & \textit{if } t \text{ がタスク図 } (V_t, A_t) \\ t & \textit{if } t \text{ が並行タスク} \\ \emptyset & \textit{otherwise.} \end{cases}$$

また，タスク図 t の辺集合を $A[t]$ とする．□

定義 7 (子孫タスク) タスク t' が t のサブタスクであるとき，すなわち $t' \in V[t]$ のとき，$t' \sqsubset t$ と書く．ある整数 $k \geq 0$ についてタスク $t'_0, \ldots, t'_k$ が存在し，

$$t' = t'_0 \sqsubset t'_1 \sqsubset \cdots \sqsubset t'_{k-1} \sqsubset t'_k = t$$

であるとき，$t' \overset{*}{\sqsubset} t$ と書く．

タスク t に対し，子孫タスクの集合 $V^*[t]$ を以下のように定義する．

$$V^*[t] = \{t' \mid t' \overset{*}{\sqsubset} t\}.$$

□

ここで，タスクとサブタスクの関係は木構造を構成するものとする．つまり，自分自身を内部に含むタスクや，互いに要素を共有するようなタスクは存在しないとする．形式的に書くと以下のようになる．

仮定 8 $t' \sqsubset t$ ならば，$t \not\overset{*}{\sqsubset} t'$ である．また，任意の異なるタスク t, t' について，$V[t] \cap V[t'] = \emptyset$ である．□

ラベルのない頂点　各頂点はタスク名を表すラベルを持つ．ただし，ラベルを持たない原始タスクも頂点として使うことができる．ラベルのない頂点は，空のタスクすなわち何も行わないタスクを表す．これは，辺の数を減らして図を見やすくするために用いられる．図 2 において，「本を探す」タスクは，ラベルのない原始タスクを 2 個含んでいる．

意味と CCS への変換　タスク図の意味は，プロセス代数 CCS[5] を使って定義する．すなわち，タスク図は以下の変換アルゴリズムによって CCS に変換できるので，タスク図 G から変換された CCS 記述 $r(G)$ の意味を G の意味とする．

CCS への変換アルゴリズムを図 6 に示し，図 2 から変換された CCS 記述の一部を図 7 に示す．ただし図 7 では，冗長なプロセス変数を消去してアルゴリズムの出力を簡略化している．

5　あとがき

本研究では，設計の上流工程から適用できる体系化されたインタラクティブシステムの設計法を提案した．また，小規模ながら現実のシステムの設計例を挙げ，本設計法の実用性を示した．本設計法のような方法によって，プロトタイプ作成以前に体系化された設計を行うことにより，設計のコストが大幅に削減されると考えられる．特に，タスク数が多く

手続き conv_to_CCS

入力: タスク t. 以下ではタスク t のタスク名を $\underline{t}$ と書く.
出力: G の CCS 記述 r. ただし, タスク t に対応するプロセス変数名は $\underline{t}$ と等しい.

```
procedure conv_to_CCS (t: タスク,
                       var r: プロセス定義式の集合);
  var
    v_i: タスク;
    N: タスクの集合;
 1: begin
 2:   if t がタスク図 then begin
        (* t = (V_t, A_t) かつ V_t = {v_0, ..., v_n} とし,
           v_0 = start, v_n = done とする.
           各頂点 v_0, ..., v_n に, r に出現しない相異な
           るプロセス変数を対応させる. v_i に対応する
           プロセス変数を P_{v_i} とする *)
 3:      for each v_i ∈ V_t − {v_n} do begin
 4:        N := {v' | (v_i, v') ∈ A_t};
           (* N = {v'_1, ..., v'_k} とする *)
 5:        if v_i ≠ start ∧ v_i がラベルを持つ then
 6:          「P_{v_i} ≝ v_i.(P_{v'_1} + ··· + P_{v'_k})」 を r に追加;
 7:        else
 8:          「P_{v_i} ≝ P_{v'_1} + ··· + P_{v'_k}」 を r に追加;
 9:      end;
10:      「t ≝ P_{v_0}」 を r に追加;
11:      「P_{v_n} ≝ 0」 を r に追加;
12:   end;
13:   if t が並行タスク then
        (* t = {t'_1, ..., t'_k} とする *)
14:      「t ≝ t'_1 | ··· | t'_k」 を r に追加;
      (* t が原始タスクのときは r に何も追加しない *)
15:   for each v_i ∈ V[t] do
        (* サブタスクに対して再帰的に実行 *)
16:     conv_to_CCS(v_i, r);
17: end;
```

図 6: タスク図を CCS 記述に変換するアルゴリズム

複雑になりがちな大規模なシステムで効力が発揮されると思われる.

本稿で述べた基本方針に基づき, 現在下記の項目について詳細な検討を行っている.

(1) タスクモデルの記述におけるガイドライン
本設計法ではタスクモデルの基本設計を順次詳細化することによりシステムの設計を完成させて行く. 適切なタスクモデルを記述するのはどのような記述法を用いようとも熟練を要する作業であり, タスクモデルの雛型や, タスクモデルのサブタスクへの分解法に関する雛型を詳細に検討する必要があると思われる.

通販で本を買う $\stackrel{\text{def}}{=}$ 本と冊数を指定する | 発送先を指定する | 支払方法を指定する

本と冊数を指定する $\stackrel{\text{def}}{=}$ 本を探す.購入冊数を指定する.P_1

$P_1 \stackrel{\text{def}}{=}$ 購入冊数を変更する.$P_1 + 0$

図 7: CCS によるタスクの記述

(2) タスクモデルの形式的検証
タスク図に対する形式的検証法について検討する. タスク図の意味が CCS を用いて形式的に定義されているので, モデルチェッキング [4] などの検証法が応用できると考えられる.

(3) プロトタイプへの変換
システムモデルからプロトタイプへの変換規則が必要である.

(4) 設計支援環境の実現
上記 (1)–(3) をふまえ, 本設計法に基づく支援環境を実現する. この支援環境が提供すべき機能として, タスクモデルの記述, 詳細化, および UI 設計情報の指定を支援するための編集機能, プロトタイプ自動生成機能, タスクモデルに対する形式的検証機能などがある.

参考文献

[1] Dix, A. J., Finlay, J. E., Abowd, G. D. and Beale, R.: *Human-Computer Interaction*, Prentice Hall Europe, 2nd edition, 1998.

[2] Harel, D.: Statecharts: A Visual Formalism for Complex Systems, *Science of Computer Programming*, Vol. 8, pp.231–274, 1987.

[3] 池田, 高田, 関: インタラクティブシステム設計におけるタスクの形式的記述とその実現, 情報処理学会第 58 回全国大会, 2C-04, 1999, 発表予定.

[4] Kesten, Y. and Pnueli, A.: Modularization and Abstraction: The Keys to Practical Formal Verification, in *MFCS '98*, LNCS1450, pp.54–71, 1998.

[5] Milner, R.: *Communication and Concurrency*, Prentice Hall, 1989.

[6] Moher, T., Dirda, V., Bastide, R. and Palanque, P.: Monolingual, Articulated Modeling of Users, Devices, and Interfaces, in *Design, Specification and Verification of Interactive Systems'96*, pp.312–329, Springer-Verlag, 1996.

[7] Newman, W. M. and Lamming, M. G.: *Interactive System Design*, Addison-Wesley, 1995.

[8] 辻野, 井上: 形式的仕様に基づく GUI プロトタイプの自動生成について, 情報処理学会研究報告, HI78, pp.13–18, 1998.