代数的仕様記述と図式仕様記述の相補的役割について†

——複眼的システムモデル——

古川忠始††* 本位田真一††
大須賀昭彦†† 津田淳一郎††

ソフトウェア・システムの仕様記述手法として種々のものが提案されている．一方，システムの仕様記述手法に対しては，形式性，読解性，実行可能性，厳密性，適用可能性，検証可能性等のいくつかの要件がある．既存の単独の手法では，それらの要件をすべて満たすことは困難である．本論文では新たな手法を構築するための前段階として，既存の手法を組み合わせ，その方法の限界について考察する．直観的な理解性に優れたSA手法と，厳密な記述と検証が可能な代数的仕様記述をとりあげ，両者の利点を失わず融合する方法を提案する．SA手法におけるデータフローダイアグラムに対して，ダイアグラムに現れるデータの性質を抽象データ型の表現を用いて代数的仕様記述に基づくオブジェクトとして記述する．この融合した手法による設計の手順を示し，LIFT制御システムを記述した例を用いて，説明する．本手法により直観的な読みやすい表現モデルを用いて設計の任意の段階でシステムに対して明らかになっている範囲の仕様を必要な抽象度で記述できることを示す．さらに，記述した仕様の検証と抽象実行の可能性について述べ，最後に組合せ手法の限界について考察する．

1. はじめに

ソフトウェア・システムの仕様記述手法として従来より，多くの提案がなされてきた．仕様記述手法に対する要件としては，形式性，読解性，実行可能性，厳密性，適用可能性，検証可能性等があるが，ある一つの既存の提案された手法によって，すべての要件を満たすことは困難であることが認識されている．そのために例えば以下の3点のような特徴を備えた複数のモデルを用いる手法（本論文ではこれを複眼的システムモデルと呼ぶ）の確立が要求されている．

- 一つのシステムについて，複数の視点からモデル化する．
- 複数のモデル化は展開の各レベルで行われる．
- 複数のモデルの違いを利用した，より良いシステムの把握，表現．

一般に，可能な限り多くの要件を満たすためには(a)新たな手法を開発すること，あるいは，(b)既存の手法を組み合わせることの二つが考えられるが，著者らは，(a)の新たな手法を開発する前の段階として，(b)のアプローチの限界を明確に認識する必要があると考える．すなわち，可能な限りの要件を満たすように要件のなかの一部を確実に満たす既存手法をいくつか選択し，各々の手法を組み合わせそれを評価し，組合せ手法の限界を見極めるのが先決である．そこで本論文では，この(b)の段階として，既存手法の組合せについて論じる．異なる特徴を持った手法を組み合わせる場合には，

(イ) 自然な形で融合させること．
(ロ) 各々の手法の利点を保つこと．
(ハ) 互いに他の手法の欠点を補完すること．
(ニ) 融合した手法における設計手順を明確にすること．

を満たしていなければならない．特に，(ハ)においては，ある手法の欠点を補うものが他の手法の利点として存在する場合には問題ないが，互いに欠点として持っている項目に対して，いかにして対処するかが大きな課題である．

本論文では，可能な限りの要件を満たすことを目的として，相反する特徴を有している図式仕様記述手法と形式的仕様記述手法を融合することを試みるが，この際に，上の要件，(イ)～(ニ)にどのように対処するかについても論じる[12]．

図式記述法としては，書きやすく，記述内容を直観的にとらえやすいものとしてSA手法[1]で用いられるデータフローダイアグラムをとりあげる．SA手法は1978年にTom DeMarcoによって提案された手法であるが，これはデータフローダイアグラム，ミニ仕様書およびDD（データディクショナリ）から構成されるものである．システムの記述はコンテクストダイア

† A Complemental Role of Algebraic Specification and Graphical Specification by TADASHI FURUKAWA, SHINICHI HONIDEN, AKIHIKO OHSUGA and JUNICHIRO TSUDA (Systems and Software Engineering Laboratory, Toshiba Corporation).

†† (株)東芝システム・ソフトウェア技術研究所
* 現在 (株)古川電機製作所

グラムと称する一枚の特別なデータフローダイアグラムを記述することから始まる．次にコンテクストダイアグラム上の一つのプロセスをその下のレベルのサブプロセスに展開したデータフローダイアグラムを構成する．このデータフローダイアグラムはプロセス（バブル），データフローおよびデータストレージから構成される．SA 手法では主として，データフローダイアグラムにおいて機能展開を繰り返しながら設計を進めることになるが，その読解性，理解性は評価され，現在数多くの CASE (Computer-Aided Software Engineering) ツールで採用されている．また，形式的仕様記述法として代数的仕様記述法[2),3)]をとりあげ，抽象データ型の表現を用いてシステム内を流れるデータの形式的な記述を行う．代数的仕様記述法は，抽象データ型を仕様記述する方法として 1975 年頃に提案されたものであり，その理論的基盤を多ソート代数においている．多ソート代数は同一種類のデータ集合（ソート）の族と，ソート間の演算の集合から構成され，各演算の意味は等式論理によって与えられる．この演算の意味は始代数という概念を通じてモデル論的に定義することができる．この手法におけるソートとその間の演算という代数モデルは SA 手法のデータフローモデルと近い関係にあり，両モデルを融合することにより，データの設計と機能の設計を統合化でき，SA 手法に欠けているデータ記述能力を補うために有効である．ここでは SA 手法の読みやすいモデルを用いながら設計の任意の段階システムに対して明らかになっている範囲の仕様を必要な抽象度で記述できる記述法を提案し，機能の展開を行う方法についても考察する．

2. SA 手法と代数的仕様記述の得失

SA 手法と代数的仕様記述の両者の長所，短所を**表 1** に示す．

SA 手法におけるデータフローダイアグラムは機能とその間に流れるデータという分かりやすいモデルに基づいており，直観的に理解しやすい簡潔な図式記述である．しかし，データフローダイアグラムを中心にデータディクショナリやミニ仕様書などの記述法を組み合わせて用いる明確なデータ中心の設計手法であるにもかかわらずデータの記述の側面はあまり強調されていない．また，SA 手法では機能の中身については，最下層のミニ仕様書を書くまではなにも記述しないという特徴があり，仕様を任意の段階で書き留めるためには記述力が不十分である．さらに，設計作業の主要な部分である機能の展開についての明確な指針がなく，設計者の経験とセンスに負うところが大きい．

表 1 SA と代数的仕様記述の長短
Table 1 Merits and demerits of SA and algebraic specification.

	長所	短所
S A 手法	馴染みやすいモデル	データの記述力低い
	簡潔な図式表現	機能の働きが記述できない
	データ中心設計	機能展開基準不明確
代数的記述法	明確な意味論	とらえにくいモデル
	抽象的な性質定義	関係が分かりにくいシンボリックな表現
	仕様レベルの検証・実行	展開基準不明確

代数的仕様記述はシステムを代数構造に基づく抽象データ型で記述しようとするもので，明確な意味論に基づいた記述言語を持ち，任意のレベルで抽象的なデータの性質定義が行えるという特徴を持っている．そのため，仕様のレベルでの検証・実行が可能であり，多くの研究がなされている．しかしながら，ソートとその間の演算という代数モデルは直観的にはとらえにくいモデルであり，厳密な記述のためのシンボリックな表現は構成要素間の関係を把握しにくいものにしている．また，SA 手法と同様に，設計を進める際のデータの展開の基準が明確でなく，設計者の経験に大きく依存している．

3. SA 手法と代数的仕様記述による設計手順

3.1 SA 手法と代数的仕様記述の接点

2 章で SA 手法と代数的仕様記述の特徴について考察したが，この章ではこの二つの手法を結び付ける際の接点となる部分について述べる．一つはシステムのモデルとそれを表す図式表現について，もう一つはデータの設計の記述法についてである．

SA 手法においては，プロセスとその間を流れるデータというモデルでシステムを記述する．その際に中心となるのがデータフローダイアグラムである．一方，代数的仕様記述では抽象データ型の記述が中心で，これはソートとソート間の演算というモデルでシステムを記述するものである．ここではソートと演算の関係を表す図式としてシグニチャグラフが用いられる．

図 1，図 2 にデータフローダイアグラムとシグニチ

ャグラフの例を示す．図1のダイアグラムは，X, Y, Z の三つのプロセスの間を a〜e の5種類のデータが流れるシステムを表している．一方，図2のダイアグラムは，a〜e の5種のソートを持ち，$X1, X2, Y, Z$ の4種のオペレーションを持つデータタイプを表している．一見して分かるとおり，この二つのダイアグラムはアローとバブルを入れ換えると相互に変換可能である．ただし X については，これを二つに分割したものを $X1, X2$ とする必要がある．このことは，代数的仕様記述におけるオペレーションが値として一つのソートのデータを返すように定義されていることに由来する．この出力データによる分割の観点はSA手法における機能展開の指針の一つとなりうるものである．

このような良く似た二つのダイアグラムが表す観点は，2章で述べたようにSA手法と代数的仕様記述の特徴の違いとなって現れてくる．端的にいえば，システムの全体を把握しながら設計を進めるためにはデータフローダイアグラムの観点が有効で，それとは逆にデータの性質を形式的な意味論（この場合は多ソート代数に基づく意味論）のもとで記述するにはシグニチャグラフの観点が有効である．ただし，データフローダイアグラムとシグニチャグラフは等価なモデルではない．シグニチャグラフを変換して得られるデータフローダイアグラムはデータストレージを含まない単純なもののみであり，データフローダイアグラムを変換して得られるシグニチャグラフの間に階層性を見いだそうとするのは分かりやすい方法とはいえない．

一方，SA手法において，データの記述はデータディクショナリという形で記述される．このデータディクショナリは主にデータの構造をBNFなどの表現で記述するもので，単純な繰返し構造とand, orでデータの構造を階層的に記述するという方法をとっている．ここでは，データの持つ意味や，値の特別な意味付けについては，自然言語で記述することになっており，自明の定義語（self-defining term）と呼ばれる，意味定義を必要としないと仮定されている単語によって，意味が最終的に決められる．自明の定義語に最終的な意味が依存するという体系から，データの設計はボトムアップ的な色彩が強く，トップダウンで途中まで設計されたデータの意味は，自然言語によるコメントと，設計者の頭の中のイメージによってしか意味付けできない．このようなデータの記述に，代数的仕様記述の抽象データ型の概念を導入することで，任意の抽象度のレベルでデータの構造と，データに対する操作の定義を代数意味論に基づいて記述できる体系が得られると考えられる．データに付随する操作は，プロセスが，入力データを加工して出力データに変換する際に利用されるサブプロセスとしてとらえることができ，その操作の抽象的な定義をより具体的な定義に置き換えていくのがデータの側面からの設計過程である．

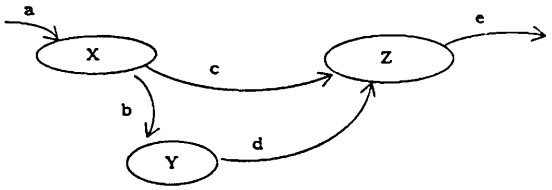


図1 データフローダイアグラムの例
Fig. 1 An example of data-flow-diagram.

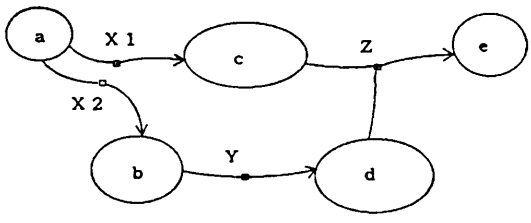


図2 シグニチャグラフの例
Fig. 2 An example of signature graph.

3.2 パラダイムの融合

SA手法と代数的仕様記述の接点について述べてきたが，これを手がかりにして両者のパラダイムの融合を考える．これは，SA手法の分かりやすい表現モデルを通してシステムを見ながら，代数的仕様記述のフォーマルな記述を用いて各設計段階で必要な抽象的定義を十分に記述しようというものである．そのためには，SA手法におけるプロセスとデータというモデルに対応するように代数的な記述の体系を構成する必要がある．

3.3 オブジェクト

データフローダイアグラムの考え方に従って記述を進めていくために，プロセスとその入出力データという単位に対して，記述を構成するように，オブジェクトという単位を**図3**のように決める．また正確な定義は付録1に示す．例えば，図1のデータフローダイアグラムのなかの Z というプロセスは**図4**，**図5**のように表現できる．また，各データに対して，その性質を記述するために，データオブジェクトという単位を**図6**のように導入する．これは，データフローとしてダイアグラム上に現れるデータおよびデータストレー

object: <オブジェクト名>
inSort: <入力データ>
outSort: <出力データ>
locSort: <内部データ>
opns: <演算の型定義>
eqns: <等式による演算の性質定義>

図 3 オブジェクト
Fig. 3 Object.

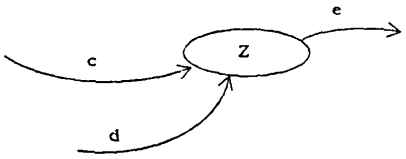


図 4 オブジェクト Z
Fig. 4 Object Z.

object: Z
inSort: c, d
outSort: e
locSort:
opns:
eqns:

図 5 オブジェクト Z
Fig. 5 Object Z.

dataobject: <データオブジェクト名>
sort: <データ名>
opns: <演算の型定義>
eqns: <等式による演算の性質定義>

図 6 データオブジェクト
Fig. 6 Dataobject.

ジについてそれらの型と構成，そして適用可能な演算について記述するものである．このデータオブジェクトは一般的な抽象データ型の表現と同じである．これについても正確な定義は付録1に示す．例えば，プロセス Z へ入力されるデータ c が，図7，図8のようなデータオブジェクトとして定義されたならば，プロセス Z は，入力されたデータ c について，ほかに s というデータと，op 1, op 2 という演算を利用できることが分かる．このデータ c のシグニチャグラフは図7のように表せるが，データフローダイアグラムに現れてくるデータは c というソートの部分のみである．このようなデータ記述を用いることで，データフローダイアグラムだけでは，陽に記述しにくかったデータの構成や適用可能な演算，関連するデータソートなどの背景情報が表現しやすくなる．

プロセス Z の定義について，object: Z は，locSort, opns と eqns の項目は何も記入されていない（図4参照）．これは，図4だけからはプロセス Z に対してこれだけの情報しか得られないことを示している．もし，Z がこれ以上展開の必要のない一つの機能からなっている場合には，Z の opns の項は $Z: c\ d \rightarrow e$ の一つのみとなり，eqns の項は，$Z(c,d)=***$ という等式で Z の機能が記述される．この右辺（***）の部分に実際には c と d と e のデータに対する操作（各データオブジェクトの opns に現れる演算）を組み合わせて構成する，Z の働きが記述される．このとき Z の等式の右辺がすでに定義された演算のみでは記述できず，新しい演算を導入する場合には，その演算を Z の opns の項目に定義しておく必要がある．このようにして記述された opns の項の演算は Z 自身を除いて Z を展開したときのサブオブジェクトとなる．また，図1の X のように出力を複数持つオブジェクトについては，出力一つに対して一つの機能になるように分割を行う．この機能の分割については次節で述べる．

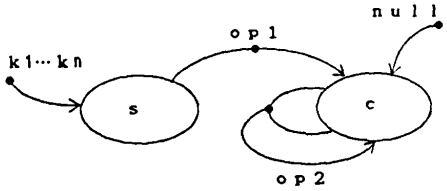


図 7 データ c のシグニチャグラフ
Fig. 7 The signature graph of data c.

dataobject: c
sort: c, s
opns: null: →c
k1…kn: →s
op 1: s→c
op 2: c, c→c
eqns: c 0, c 1, c 2, c 3∈c.
op 2 (null, c 0) =c 0
op 2 (c 0, null) =c 0
op 2 (op 2 (c 1, c 2), c 3)
=op 2 (c 1, op 2 (c 2, c 3))

図 8 データオブジェクト c
Fig. 8 Dataobject c.

3.4 機能の展開

オブジェクトの機能の展開については，三つの指針がある．

（展開ルール1） 出力データ一つに対して機能が一つ対応するように分割する．

これにより入力を共有しているような並列的な機能の分離を行う．一般に代数的仕様記述では，機能に対応する演算は副作用のない関数の形で定義されるため，値として返すデータは1種類に制限される．本論文においても，仕様記述を代数的に行うために，出力データが一つになるような分割を行う．

（展開ルール2） 等式の右辺の記述に基づき展開する．

すでに出力が一つになった機能について，入出力のデータオブジェクトに定義されている演算を組み合わせてオブジェクト表現の eqns 項の等式表現を考える．等式の右辺に記述される式は機能の働きを表現していると同時に機能の展開を表している．これは入力データをいくつかの機能が順番に，前の処理の結果を入力とするように処理を進める直列的な関係を含んだ機能の分解に相当する．例えば，$A(x,y)=D(B(x),c(y))$ という等式は，A が B,C,D という三つの機能の組合せに展開されることを示し，入力データ x を B が，y を C がそれぞれ処理した結果を D への入力とすることを表す．

（展開ルール3） データストレージはデータオブジェクトとしてまとめる．

システムの内部状態に相当するデータとその読み出し，書換え等の演算をデータオブジェクトの形にまとめ，これらの演算の展開は他のものとは区別して進める．この種のデータはダイアグラム上ではデータストレージとして現れ，一時的なデータの蓄積もしくは不特定多数のプロセスからの参照のためのデータベースとしての性格を持っている．このような実行時に値が決まるデータに直接関連したオブジェクトを局所化して展開を進めることで，仕様の静的な検証の範囲と動的な実行による検証の際に注目すべき部分を明確に示すことができる．

3.5 複眼的システムモデルによる設計手順

このモデルを用いたシステム設計の手順を以下に示す．また，手順を図にしたものを**図9**に示す．

＊step 1＊ 対象システムと外界とのデータの入出力を特定し，コンテクストダイアグラムを書く．

＊step 2＊ 対象システムのオブジェクト表現を作る．すなわち，コンテクストダイアグラムに現れる対象システムの名前をオブジェクト名としてとり，外界との入出力データをそれぞれ inSort, outSort 項に記述する．

＊step 3＊ 入出力に現れる各々のデータに対して，データオブジェクトの記述をする．データオブジェクト名には，ダイアグラムに現れたデータ名をとり，そのデータを構成する元になるデータや，加工して得られるデータ等，密接な関連を持つと思われるデータの型名を sort 項目に列挙する．データを構成するための演算や加工するための演算などそれらのデータ群の間に定義される演算を opns の項目に型定義し，特別な意味を持つ定数記号についても定数関数として定義する．また，演算の持つ性質について eqns の項目に等式の形で表現できる程度に明確化されたも

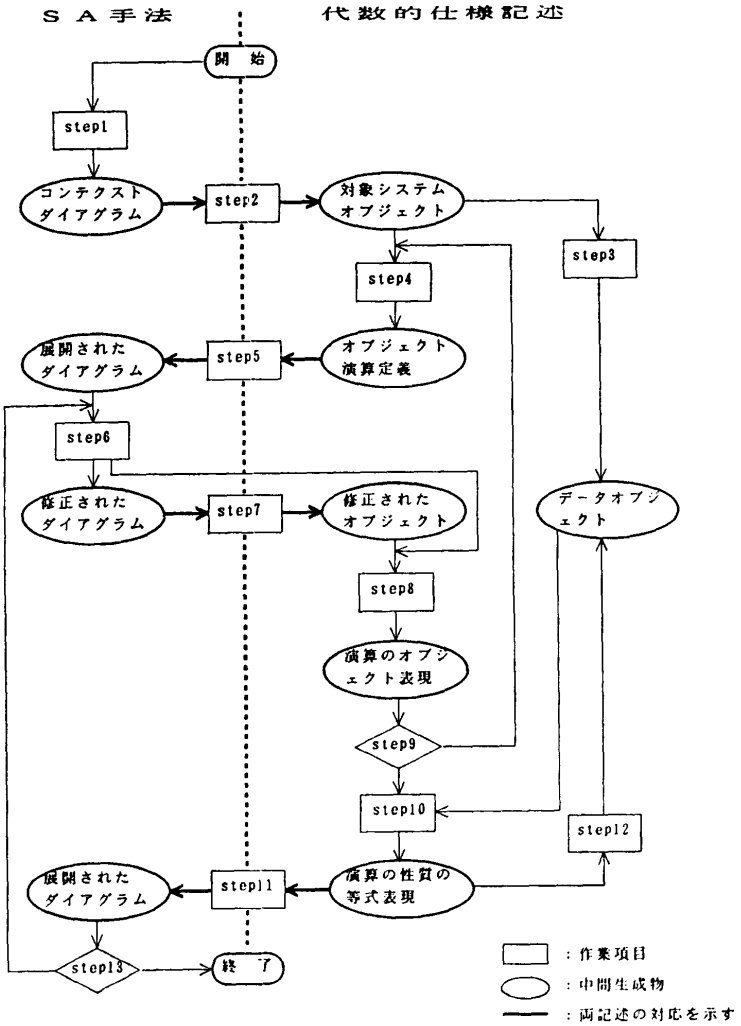


図9 複眼モデルによる設計手順
Fig. 9 Design procedure of compound model.

のについて記述する．

＊step 4＊ 展開ルール1に従い，オブジェクトの出力データを一つ選び，そのデータの内容に影響力を持つと思われる入力データをあるだけ選び出す．それらの入力データをドメインに，出力データをレンジに持つ演算を考え，名前を付けてオブジェクトの opns 項目に記述する．この際，中間的なデータの導入が必要であれば適切な名前を付け locSort 項に記述し，その時点で明確になっている性質についてデータオブジェクトの記述を作成する．

＊step 5＊ 各演算をバブル（サブプロセス）とし，演算の入出力をデータフローとして展開されたデータフローダイアグラムを作成する．このとき，ダイアグラム上にデータストレージがあれば展開ルール3に従い，特に意識してデータオブジェクトとして記述し，これ以後そのデータストレージを直接操作する演算はすべてこのデータオブジェクトに定義する．

＊step 6＊ データフローダイアグラムを検討し，抜けているところがあればデータフローダイアグラム上で修正する．修正する必要がなければ step 8 へいく．

＊step 7＊ step 6 でのデータフローダイアグラムの修正をオブジェクト記述に反映させる．

＊step 8＊ 各バブルの演算名をオブジェクト名に持つオブジェクトの記述を作成する．その際 opns の項目にはオブジェクト自身の名前も演算として登録しておく．

＊step 9＊ step 7 での修正により出力データが複数になったオブジェクトがあれば，step 4 に戻って再び機能展開する．すべてのバブル（演算）の出力が一つ (opns 項において) で，これ以上の修正がないなら step 10 へいく．

＊step 10＊ 各オブジェクトについてそれぞれの演算の入力データと出力データの間にどのような関係があるかを考える．その際，step 3 で定義した入力データと出力データ各々のデータオブジェクトの記述を参照しながら，そこに定義されている定数や演算を用いて，eqns 項目に等式の形で各演算の働きを定義する．既存の演算のみでは表現できない場合には新しい演算を適切な名前を付けて opns 項に付加しドメインとレンジについての定義を記述する．また，中間的なデータが必要になった場合にはオブジェクト記述の locSort 項に記述を付加する．

＊step 11＊ オブジェクト自身の名前を持つ演算に関する eqns 項の等式が，再帰表現になっている場合にはそのオブジェクトはそれ以上展開しない．再帰的でない表現すなわち他の演算の組合せで表現されている場合は展開ルール2により右辺に現れる演算をサブプロセスとしてオブジェクトを展開したデータフローダイアグラムを作成する．

＊step 12＊ 新たに導入されたデータについてデータオブジェクトの記述を構成する．また，ここまでの過程で明確になったデータの性質について，すでに作成されているデータオブジェクトのそれぞれについても記述を付加する．

＊step 13＊ プリミティブ以外の演算が eqns 項にすべて定義され，展開すべきオブジェクトがなくなったら終了，そうでなければ step 6 へ．

設計の途中では，データフローダイアグラムで全体の構成を確認しながら，各オブジェクトの記述を進めていく．その際に必要な演算を付け加え，必要に応じ修正し，明らかになったところから記述を進めていく．

4. 例　　題

前章で述べた手順で LIFT コントロールシステムの例題[4]の記述を試みた．

この例題の内容を付録2に示し，記述の過程を以下に示す．

step 1：問題の文面から LIFT controlsystem と外界とのデータの入出力を特定し，コンテクストダイアグラムを書く（**図 10**）．図で四角形で表されているのが対象システムの外界を表し，中央の円で表されるシステムとのデータの入出力が矢印で表されている．

step 2：LIFT controlsystem のオブジェクトの記述を行う（**図 11**）．ここでは，6種の入力と4種の出力データがそれぞれ記述される．

step 3：図 11 に現れるデータに対応したデータオ

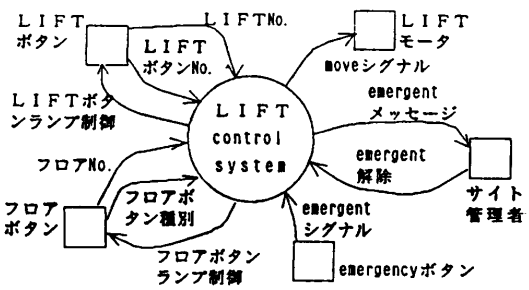


図 10 コンテクストダイアグラム
Fig. 10 Context diagram.

object: LIFT-control-system
inSort: LIFTNo.,
LIFT ボタン No.,
フロア No.,
フロアボタン種別,
emergent シグナル,
emergent 解除
outSoft: LIFT ボタンランプ制御,
フロアボタンランプ制御,
emergent メッセージ,
move シグナル
locSort:
opns:
eqns:

図 11 コンテクストダイアグラムに対応したオブジェクト記述
Fig. 11 The object representation of context diagram.

dataobject: フロア No.
sort: フロア No.
opns: F1, …Fm: →フロア No.
eqns:
dataobject: フロアボタン種別
sort: フロアボタン種別
opns: to-up: →フロアボタン種別
to-down: →フロアボタン種別
eqns:
dataobject: フロアボタンランプ制御
sort: フロア No.,
フロアボタン種別,
フロアボタンランプ制御
opns: off: フロア No., フロアボタン種別→フロアボタンランプ制御
on: フロア No., フロアボタン種別→フロアボタンランプ制御
eqns:

図 12 データオブジェクトの例
Fig. 12 Dataobjects example.

ブジェクトの記述を構成する（**図 12** にその一部を示す）.

step 4 : LIFT controlsystem オブジェクトのそれぞれの出力データについて，対応する入力データを列挙し，それぞれの入出力の組に名前を付け opns の項を記述する（**図 13**）. 出力データに直接対応のつかない入力データについては，中間的なデータを想定する. locSort 項の LIFT 状態はこのときに導入される. さらにこの LIFT 状態を出力データとする opns の項も追加する.

step 5 : 図 13 の 7 個の演算をそれぞれデータフローダイアグラムのバブルに対応させ，入出力をデータフローとしてダイアグラムを作る（**図 14**）. さらに LIFT 状態に関するオブジェクトを作成する.

step 6 : 図 14 のダイアグラムを検討し抜けているところがあれば修正する. この例では**図 15** に示す＊印のついたデータフローの付加を行った.

step 7 : ダイアグラムに対する修正をオブジェクトの記述に反映させたものが**図 16** である（図中では修正された部分を太字で示す）.

step 8 : 各バブルについて演算名をオブジェクト名に持つオブジェクトの記述を作る.

step 9 : step 7 の修正により出力データが複数になった LIFT 動作指示とフロア動作指示という二つのオブジェクトについて，再び step 4 の手順で機能展開を行う. **図 17** では該当するオブジェクトのうち LIFT 動作指示について，setLIFTstop と makeMove の二

object: LIFT-control-system.,
inSort: LIFTNo.,
LIFT ボタン No.,
フロア No.,
フロアボタン種別,
emergent シグナル,
emergent 解除
outSort: LIFT ボタンランプ制御,
フロアボタンランプ制御,
emergent メッセージ,
move シグナル
locSort: LIFT 状態
opns: LIFT ボタン制御: LIFTNo., LIFT ボタン No., LIFT 状態
→LIFT ボタンランプ制御
フロアボタン制御: フロア No., フロアボタン種別, LIFT 状態
→フロアボタンランプ制御
emerg メッセージ送信: emergent シグナル
→emergent メッセージ
LIFT 動作指示: LIFTNo., LIFT ボタン No.
→move シグナル
フロア動作指示: フロア No., フロアボタン種別
→move シグナル
emerg 状態制御: emergent シグナル→LIFT 状態
emerg 解除処理: emergent 解除→LIFT 状態
eqns:

図 13 演算を定義したオブジェクト
Fig. 13 Object with definitions of operators.

つの演算に展開した. **図 18** に展開結果のデータフローダイアグラムを示す. 展開が終わり，出力が一つになったバブルについてそれぞれのオブジェクトの表現を作成する（**図 19**）.

step 10 : 新たに定義されたオブジェクトのうち setLIFTstop について入力データと出力データの間の関係を記述したものを**図 20** に示す. ここでは新たなデータとして停止階リストが導入され，setLIFTstop を構成するための演算がいくつか新しく定義されてい

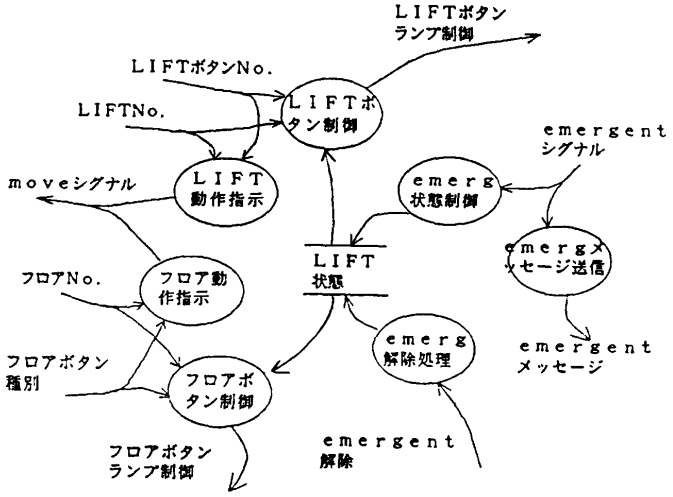


図 14 展開したデータフローダイアグラム
Fig. 14 Decomposed data-flow-diagram.

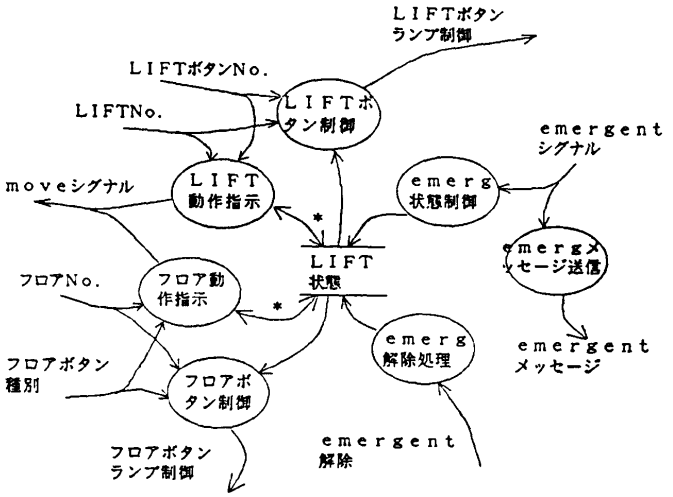


図 15 修正されたデータフローダイアグラム
Fig. 15 Modified data-flow-diagram.

る.

step 11: 新しく定義された演算，停止階読出，停止階追加，停止階書込によって構成されている setLIFTstop の等式記述に基づき，データフローダイアグラムに展開したものを**図 21** に示す.

step 12: step 10 で導入した停止階リストというデータについてはもう少し性質が明確になるまでデータオブジェクトの記述は後回しにする一方，LIFT 状態に関する演算として setLIFTstop を LIFT 状態というデータオブジェクトの構成要素として登録する(**図 22**)．これにより他のオブジェクトを展開する際に setLIFTstop を再利用できる.

紙面の都合で以下は割愛するが，step 13 の終了条件が満たされたときに設計は終了する.

5. 評　価

データフローダイアグラムを書きながら代数的なオブジェクトの記述を平行して記述することで，全体が把握しやすく，抜けを発見するのも容易になる．さらに，データフローダイアグラムには明示的に現れないデータを構成する他のデータやデータに適用可能な演算群をデータオブジェクトに記述することで，設計を進める際の部品群としての効果が期待できる.

また，代数的な厳密な記述をすることで始代数意味論に基づいた等式論理を用いて，仕様の静的な検証が行える．第一に，step 4 で記述するオブジェクトの演算の型定義から，システム全体のデータの型の整合性検証が可能である．第二に，step 10 で記述する演算の性質の等式表現に基づいて，等式論理上の推論を用いて機能展開の階層間の整合性，すなわち，上位のオブジェクトの性質を，下位のオブジェクトの集合が満たしているかどうかの検証が可能である.

ただし，上のような検証は，各オブジェクトの演算が内部状態を持たない，つまり参照の透明性が保持される範囲についてのみ有効である．内部状態が現れる場合には，3.4 節で述べた機能展開の指針により，内部状態を扱うオブジェクトを局所化することで，静的な検証の枠から外れる部分を明確にしておき，その部分については抽象実行による動的な検証を行う必要がある.

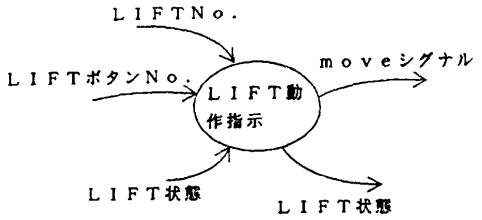
本手法が基にしている SA 手法と代数的仕様記述は，拡張されていない基本的なモデルであるため，タイミングの記述やデータ型の包含関係の扱い等，両者の手法の持つ記述の限界が本手法にも反映している．SA 手法のリアルタイム系への拡張[5),6)]，代数的仕様記述における型構成子の導入，他のオブジェクトからの仕様の取り込み[14),15)]等の拡張されたモデルの融合については今後の重要な課題である.

6. 他の手法との比較

本章では以下の四つの観点から他の手法との比較を行う.

object: LIFT-control-system
inSort: LIFTNo.,
　　LIFT ボタン No.,
　　フロア No.,
　　フロアボタン種別,
　　emergent シグナル,
　　emergent 解除
outSort: LIFT ボタンランプ制御,
　　フロアボタンランプ制御,
　　emergent メッセージ,
　　move シグナル
locSort: LIFT 状態
opns: LIFT ボタン制御: LIFTNo., LIFT ボタン No.,
　　LIFT 状態→LIFT ボタンランプ制御
　　フロアボタン制御: フロア No., フロアボタン種別,
　　LIFT 状態→フロアボタンランプ制御
　　emerg メッセージ送信: emergent シグナル
　　→emergent メッセージ
　　LIFT 動作指示: LIFTNo., LIFT ボタン No.,
　　LIFT 状態→move シグナル, **LIFT 状態**
　　フロア動作指示: フロア No., フロアボタン種別,
　　LIFT 状態→move シグナル, **LIFT 状態**
　　emerg 状態制御: emergent シグナル→LIFT 状態
　　emerg 解除処理: emergent 解除→LIFT 状態
eqns:

図 16 修正されたオブジェクトの記述
Fig. 16 Modified object representation.



object: LIFT 動作指示
inSort: LIFTNo., LIFT ボタン No., LIFT 状態
outSort: LIFT 状態, move シグナル
locSort:
opns: setLIFTstop: LIFTNo.,
　　LIFT ボタン No., LIFT 状態→LIFT 状態
　　makeMove: LIFT 状態→move シグナル
eqns:

図 17 出力が二つになったオブジェクト
Fig. 17 An object with two output data.

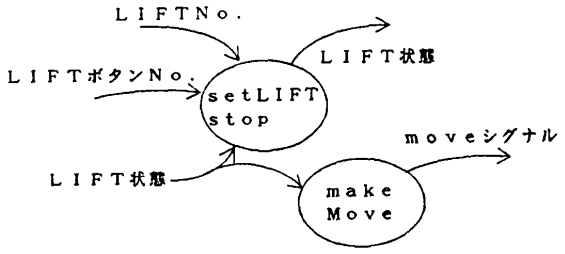


図 18 LIFT 動作指示を展開したもの
Fig. 18 Decomposed diagram of Fig. 17.

object: setLIFTstop
inSort: LIFTNo., LIFT ボタン No., LIFT 状態
outSort: LIFT 状態
locSort:
opns: setLIFTstop: LIFTNo.,
　　LIFT ボタン No., LIFT 状態→
　　LIFT 状態
eqns:

object: makeMove
inSort: LIFT 状態
outSort: move シグナル
locSort: LIFT 位置, 停止階リスト
opns: makeMove: LIFT 状態→
　　move シグナル
eqns:

図 19 サブオブジェクトの記述
Fig. 19 Representation of subobjects.

object: setLIFTstop
inSort: LIFTNo., LIFT ボタン No., LIFT 状態
outSort: LIFT 状態
locSort: 停止階リスト
opns: setLIFTstop: LIFTNo.
　　LIFT ボタン No., LIFT 状態→
　　LIFT 状態
　　停止階読出: LIFT 状態→停止階リスト
　　停止階追加: 停止階リスト, LIFTNo.,
　　LIFT ボタン No. →
　　停止階リスト
　　停止階書込: LIFT 状態, 停止階リスト→
　　LIFT 状態
eqns: for all ln∈LIFTNo.;
　　lb∈LIFT ボタン No.: ls∈LIFT 状態
　　: setLIFTstop (ln, lb, ls) =
　　停止階書込 (ls, 停止階追加 (停止階読出 (ls), ln, lb))

図 20 eqns 項を記述したオブジェクトの例
Fig. 20 An object with equational description.

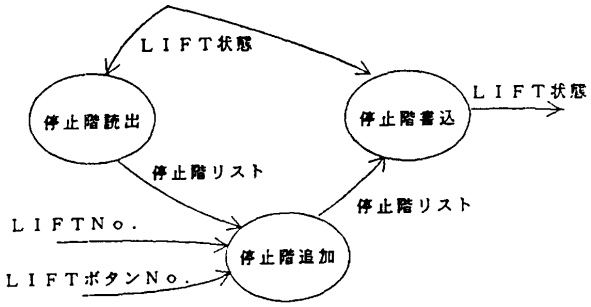


図 21 setLIFTstop を展開したもの
Fig. 21 Decomposed diagram of setLIFTstop.

(1) SA 手法とその拡張

SA 手法とその拡張については多くの研究があり[8]，コントロールフローの導入によるリアルタイム系の記述への拡張[5),6)]や，ペトリネットの解析手法をデータフローダイアグラムにトークンを導入したデータフ

```
dataobject: LIFT 状態
sort: LIFT 状態, LIFTNo., 運行状況, 運転方向,
    停止階リスト, LIFT 位置
opns: inService: →運行状況
      outofService: →運行状況
      setDirection: LIFTNo., 運転方向,
                    LIFT 状態→LIFT 状態
      getDirection: LIFTNo., LIFT 状態
                    →運転方向
      setLIFTstop: LIFTNo.
                   LIFT ボタン No., LIFT 状態→
                   LIFT 状態
      set フロア stop: フロア No., フロアボタン種別,
                       LIFT 状態→LIFT 状態
eqns:
```

図 22 データオブジェクト LIFT 状態
Fig. 22 Dataobject "LIFT-status."

ローグラフに適用しようというもの[7]，などがある．この観点から見ると本手法は，SA 手法のデータディクショナリに形式的な記述の手法を導入したものといえる．

（2） マルチビュー

STATEMATE[9]は，複数の観点からシステムの仕様を表現しようとする SA 手法とよく似た手法であり，データフローダイアグラムに相当する module-chart，コントロールフローを含んだデータフローを記述する activity-chart，状態遷移を表現する state-chart の三つの視点を用いて記述力を高めている．しかし，これらの手法はシステムを流れるデータの構造や性質の記述にはあまり力を入れていないために，機能の仕様を任意の抽象度で記述するためには語彙が不足している．また，ダイアグラムの展開の指針という面での設計のよりどころが明確でなく，SA 手法の範囲を出ていない．本手法は，データフローの観点と，代数モデルの視点を相補的に融合し，両者の観点を同時に用いることで抽象的な記述と設計の指針を得ている．

（3） 機能展開の基準

機能の展開については，ダイアグラムを代数系に変換し，その上の演算規則で展開を行う方法[10]やリアルタイム制約を伴ったデータフローダイアグラムを，有向グラフに基づくプロトタイプ言語として定式化しデータ型を導入したもの[11]などが提案されている．文献 10) ではデータフローダイアグラムのグラフ的な側面から，書換え規則を用いてシンタクティックに展開をしているがデータや機能の意味を切り離しているために，設計というよりも記述の最適化的な手法である．文献 11) の手法は，リアルタイムシステムに対するラピッドプロトタイピングを目指したものでデータフローとコントロールフローの両方に基づく展開を組み合わせるための計算モデルを提案している．本手法では SA 手法と代数的仕様記述のモデルの特徴の差に基づいて 3.4 節で述べたような展開の基準を得ている．

（4） 代数的仕様記述の拡張

代数的仕様記述に図式記述を取り入れたものとして文献 13) がある．これは，データ型の包含関係を記述するのにベン図を用いるものである．本手法では，データ型の包含関係については扱わず，データの相互間連についてデータフローダイアグラムを用いて表現する．

7. おわりに

SA 手法の読みやすく理解しやすいモデルと，代数的仕様記述による形式的なデータの記述を自然な形で融合させてシステムの仕様を記述する方法について提案し，二つの手法の利点を保ちながら欠点を補完することを示した．また，両者の欠点であった機能の展開基準について，二つのモデルを同時に関連させて用いることで明確な指針が得られた．

今後検討すべき点としては，仕様の静的検証，動的検証のための検証・実行系の実現，記述の支援を行うための自動化環境の構築，等がある．特に，検証・実行系については記述能力の検討とともに今後の大きな課題である．

謝辞 最後に，発表の機会を与えていただいた，(株)東芝システム・ソフトウェア技術研究所，西島誠一所長，高橋生宗部長，河野毅部長に感謝いたします．

また，本研究は情報処理振興事業協会の「ソフトウェア・プロトタイピング技術の調査研究」ワーキング委員会での討論から多くの情報と刺激を受けている．情報処理振興事業協会の古宮誠一氏，東京工業大学の佐伯元司助教授を始めとする委員の方々に感謝いたします．

参考文献

1) De Marco, T.: *Structured Analysis and System Specification*, Yourdon, New York (1980).
2) 稲垣，坂部：抽象データタイプの代数的仕様記述の基礎 (1〜4)，情報処理，Vol. 25, Nos. 1, 5, 7, 9, pp. 47-53, 491-501, 708-716, 971-986 (1984).
3) Ehrig, H. and Mahr, B.: *Fundamentals of Algebraic Specification 1*, Springer-Verlag EATCS Monographs on Theor. Comput. Sci.

Vol. 6 (1985).
4) *Proc. of Fourth International Workshop on Software Specification and Design*, CS Press, Los Alamitos, Calif., April (1987).
5) Ward, P. T. et al.: *Structured Development for Real-Time Systems*, Prentice-Hall/Yourdon Press, New York (1985).
6) Hatley, D. J.: The Use of Structured Method in the Development of Large Software-based Avionics Systems, *Proc. AIAA/IEEE 6 th Digital Avionics Systems Conference* (Dec. 1984).
7) Kavi, K. M. and Buckles, B. P.: Isomorphisms between Petri Nets and Dataflow Graphs, *IEEE Trans. Softw. Eng.*, Vol. SE-13, No. 10, pp. 1127-1134 (1987).
8) Mingers, J.: Comparing Conceptual Models and Data Flow Diagrams, *Comput. J.*, Vol. 31, No. 4, pp. 376-379 (1988).
9) Harel, D., Lachover, H., Naamad, A. and Pnueli, A.: STATEMATE: A Working Environment for the Development of Complex Reactive Systems, *Proc. 10th ICSE*, pp. 396-406 (Apr. 1988).
10) Adler, M.: An Algebra for Data Flow Diagram Process Decomposition, *IEEE Trans. Softw. Eng.*, Vol. 14, No. 2, pp. 169-183 (1988).
11) Luqi and Berzins, V.: Rapidly Prototyping Real-Time Systems, *IEEE Softw.*, Vol. 5, No. 5, pp. 25-36 (1988).
12) 古川，津田，本位田：代数的仕様記述と図式仕様記述の相補的役割について，1989 情報学シンポジウム講演論文集，pp. 33-42 (1989).
13) Wing, J. M.: A Larch Specification of the Library Problem, *Proc. Fourth Int. Workshop Software Specification and Design*, CS Press, Los Alamitos, Calif., pp. 34-41 (1987).
14) Wing, J. M.: Writing Larch Interface Language Specifications, *ACM Trans. Prog. Lang. Syst.*, Vol. 9, No. 1, pp. 1-24 (1987).
15) Futatsugi, K. et al.: Principles of OBJ 2, *12th ACM Symp. on POPL*, pp. 52-66 (1985).

付録 1 オブジェクト記述の文法

＜object＞ : : = object : ＜オブジェクト名＞
inSort : ＜ソート名リスト＞
outSort : ＜ソート名リスト＞
locSort : ＜ソート名リスト＞
opns : ＜演算リスト＞
eqns : ＜等式リスト＞
＜オブジェクト名＞ : : = ＜名前＞
＜データオブジェクト名＞ : : = ＜名前＞
＜ソート名リスト＞ : : = ＜名前リスト＞
＜演算リスト＞ : : = ＜演算記述＞$^+$
＜演算記述＞ : : = ＜名前リスト＞ : [＜名前＞$^+$]→ ＜名前＞
＜演算記述＞ : : = ＜名前リスト＞ : [＜名前＞$^+$]→ ＜名前＞$^+$
＜等式リスト＞ : : = ([for all＜変数宣言記述＞ :] (＜term＞＜term＞)$^+$)$^+$
＜変数宣言記述＞ : :=[＜名前リスト＞$^+$∈＜名前＞ ;]$^+$ ＜名前リスト＞$^+$∈＜名前＞
＜term＞ : : = ＜名前＞[([＜term＞,]$^+$＜term＞)]
＜名前リスト＞ : : = [＜名前＞,]$^+$＜名前＞$^+$
＜名前＞ : : = ＜文字＞$^+$
＜dataobject＞ : : = dataobject : ＜データオブジェクト名＞
sort : ＜ソート名リスト＞
opns : ＜演算リスト＞
eqns : ＜等式リスト＞

オブジェクト，データオブジェクト文法

付録 2 LIFT コントロールシステムの例題[4]

- n 台の LIFT が m 階建てのビルに設置されている．
- LIFT と中身のメカニズムはもう決まっているものとし，LIFT をフロアからフロアへ動かすロジックを，以下の制約条件を満たすように設計する．

1—各 LIFT は各階に対応するボタンを持ち，ボタンを押すと，その階に向けて LIFT は動きだし，ボタンには明りがともる．LIFT が目的の階にたどり着くとボタンの明りは消える．

2—各階には二つのボタンがある．(ただし，最下，最上階は例外)
一つは上りの LIFT 用で，もう一つは下りの LIFT を呼ぶものである．
ボタンは押されると点灯し，次の場合に消灯する．
(1) LIFT がその階に到着し，要求されている方向に向かっている．
(2) LIFT がその階に到着し，LIFT 自体は，他の行き先を指定されていない．
(2)の場合には，フロアの上り，下りの両方のボタンが押されている場合，どちらか一方しか消灯しない．
どちら向きの LIFT とするかは，両方の待ち時間が最小になるように決定する．

3—LIFT は何も要求がないときは，最後に止まった

階でドアを閉めた状態で待つ．

4—LIFT への各階からの要求は，いつかは必ず満たされるものとし，各階は同じ優先順位を持つ．

5—LIFT の行き先の要求はいつかは必ず満たされる．また，各階への停止は，運転方向に従って，順次処理される．

6—各 LIFT は非常ボタンを持っている．

非常ボタンが押されると，警報が管理者のもとへ伝えられる．その時点で当該 LIFT は運行中止とみなされる．各 LIFT は運行中止状態を解除する機構を持つ．

(平成元年 3 月 1 日受付)
(平成元年 11 月 14 日採録)

古川 忠始（正会員）

昭和 36 年生．昭和 61 年東京工業大学総合理工学研究科システム科学専攻修士課程卒業．(株)東芝システム・ソフトウェア技術研究所，勤務の後平成元年 2 月退職．現在，(株)古川電機製作所勤務．主たる研究テーマはシステムの仕様記述法．

本位田真一（正会員）

昭和 28 年生．昭和 51 年早稲田大学理工学部電気工学科卒業．昭和 53 年同大学院理工学研究科電気工学専攻修士課程修了．工学博士．同年(株)東芝入社．現在，同社システム・ソフトウェア技術研究所研究主務．早稲田大学非常勤講師．主として，ソフトウェア工学，人工知能の研究に従事．ソフトウェアの基礎理論に興味をもつ．昭和 61 年情報処理学会論文賞受賞．著書：「ソフトウェア開発のためのプロトタイピング・ツール」(共著)，「KE 養成講座② エキスパートシステム基礎技術」(共著)，「オブジェクト指向システム分析」(共訳)．人工知能学会，日本ソフトウェア科学会，電気学会，AAAI 各会員．

大須賀昭彦（正会員）

1958 年生．1981 年上智大学理工学部数学科卒業．同年(株)東芝入社．1985 年 4 月～1988 年 12 月(財)新世代コンピュータ技術開発機構に出向．同研究所にて知的プログラミングシステムの研究開発に従事．現在，(株)東芝システム・ソフトウェア技術研究所において形式的仕様記述法の研究に従事．定理の自動証明，プログラムの検証などに興味を持つ．1986 年情報処理学会論文賞受賞．日本ソフトウェア科学会，EATCS 各会員．

津田淳一郎（正会員）

昭和 22 年生．昭和 45 年東京工業大学理工学部制御工学科卒業．同年(株)東芝入社．現在，同社システム・ソフトウェア技術研究所主任研究員．ソフトウェア管理技術の研究に従事．計測自動制御学会会員．