

プロダクト統合のためのソフトウェア仕様化・設計法のモデル化手法の提案

佐伯元司[†] 井口和久^{†*}
郭文音^{†**} 篠原正紀[†]

本論文では、ソフトウェア仕様化・設計法のモデル化手法を提案し、その有効性について議論する。種々の分野のシステムや複雑なシステムの仕様化・設計を行うのに1種類の手法を使うのではなく、いくつかの手法を組み合わせ使用したほうが効率的である。その際、複数の方法論で記述された仕様をどのように統合し、一つの仕様にするかが問題となる。複数の方法論を組み合わせ使えるようにするために、実体関連モデルに基づいた方法論のモデル化手法を提案する。このモデル化手法は、各方法論で作られるプロダクトだけでなく、方法論で行われる作業をもモデル化している。このメタモデルは、各方法論に含まれる共通概念を用いて構成されており、これにより異種の方法論に基づいて作成されたプロダクトを統合することもできる。

Modeling Software Specification & Design Methods for Product Integration

MOTOSHI SAEKI,[†] KAZUHISA IGUCHI,^{†*} KUO WENYIN ^{†**}
and MASANORI SHINOHARA [†]

This paper discusses a meta modeling technique for various software specification & design methods and its benefit. It is more effective to use multiple methods rather than a single method in order to specify various kinds of systems and complex systems. It is an essential point how to integrate sub specifications developed by the multiple methods into a final specification. We propose the meta model based on Entity Relationship Model to use multiple methods in the supporting tool. This meta model includes not only the model of products constructed in the method but also the model of the activities in it. For integrating the sub specifications which are developed by the different methods, it is made of the common concepts that various methods have.

1. ま え が き

ソフトウェア開発過程において、仕様化・設計は初期の段階に位置する。それゆえ、高品質のソフトウェアを効率的に開発するには、仕様化・設計過程の支援が重要である。これまで、構造化手法^{1),2)}やオブジェクト指向法^{3),4)}などの種々の手法が開発され、実用化への研究や支援ツールの開発がなされている。しかし、一つの手法があらゆる分野のソフトウェアシステムの設計に対して適しているわけではない。たとえば、構造

化分析法¹⁾はリアクティブシステムのリアクティブ動作を記述することには適していないことが指摘されている⁵⁾。あらゆる分野のソフトウェアシステムの仕様化・設計に適した一つの万能の仕様化・設計法（以下、設計法と呼ぶことにする）をつくることは難しい。それよりも、ソフトウェアシステムの各部分に合わせて、それぞれに適した設計法を組み合わせ利用したほうが現実的であろう。つまり、任意の箇所、任意の段階で、対象領域にふさわしい設計法を組み合わせ選択して使い分けたほうがよいであろう。そのためには、さまざまな設計法を蓄積し、必要なときに任意の設計法を利用できるような支援環境が不可欠である。このような支援環境においては、各設計法に従った支援を行うことに加えて、問題となるのは複数の設計法で開発された仕様をどのように統合するかである。また、Statecraft⁶⁾や OMT⁶⁾に代表されるように、一つのシ

[†] 東京工業大学工学部情報工学科
Department of Computer Science, Faculty of Engineering, Tokyo Institute of Technology

* 現在, NHK 放送技術研究所
Presently with NHK Science and Technical Research Laboratories

** 現在, (株)日立製作所
Presently with Hitachi Corporation

システムを複数の設計法（例えばデータフロー図、状態遷移図や実体関連図など）で仕様化する手法も提案されており、複雑なシステムの記述に効果を発揮している。このような手法においても、異なる設計法で作成された複数の仕様を統合する機構が重要である。

種々の設計法を扱うことができる支援環境の実現を容易にするために、MetaEdit や Metaview などのメタシステム^{7),8)}やメタ CASE⁹⁾といった考え方が提案されている。このようなシステムでは、設計法の記述を入力データとし、その設計法を支援するツールを生成することができる。このときに問題となるのは、設計法をどのようにモデル化（メタモデリングという）し、記述するかである。記述したものはメタモデルと呼ばれる。本論文では、異なる設計法で開発される複数の仕様を統合するためのメタモデリング手法を提案する。これまで提案されてきたメタモデリング手法はいずれも仕様の構造のみをモデル化するのみであり、仕様の統合という観点から作られたものではない。

本論文では、まず、従来のメタモデリング手法の問題点を述べ、本研究でとったアプローチ—統合のための共通メタモデルの構築と作業手順情報のモデルの追加—について概説する。第3章で共通メタモデルの詳細を、第4章では本メタモデルの効用と問題点を調べるために、試作した統合化機能を持つ支援ツールのプロトタイプについて述べる。第5章で試作経験から得られた本手法の利点と問題点を議論する。

2. メタモデリングの方針

2.1 従来のメタモデルの問題点

ソフトウェアの設計法の主要な目的は、ソフトウェアの設計作業の進め方を指示することである。ソフトウェアの設計法は必要な仕様書とその仕様書を得るために必要な作業手順を教えてくれる。従って、すべての設計法は

- プロダクトの視点：どんなものが作られるか。作られるものの構造はどうなっているか*。
- 手順の視点：どんな作業をどのような順序で行わなければならないか。

の二つの視点を持っている。

これまで、実体関連モデルを用いて設計法をプロダクトの視点からモデル化した研究はいくつかなされており^{10)~12)}、設計法のメタモデルをプロダクトを蓄積するリポジトリのスキーマとすることにより支援環境の実現^{8),7),13)}もなされている。例えば、これらの手法を用

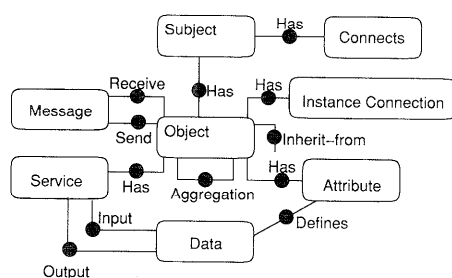


図1 オブジェクト指向分析法で作られるプロダクト
Fig.1 Products constructed in OOA.

いてCoad & Yourdonのオブジェクト指向分析法のオブジェクト図(Object Diagram: ODと略す)をプロダクトという視点から実体関連モデルを用いてモデル化すると、図1のようになるであろう。図中の楕円は実体、黒丸は関連を表す。例えば、オブジェクト図中のObjectは固有の属性(Attribute)やサービス(Service)を持っている(関連Has)。図中で、実体で定義されるものを、以下ではその設計法が持っている固有の概念と呼ぶことにする。直観的には、設計法が持っている概念とは、作成されるプロダクトのあるままとった部分を指している。

しかし、従来のメタモデリング手法はいずれも以下のような問題点がある。

1. 設計法ごとにプロダクトの視点からのメタモデルができ、複数の設計法を扱う支援環境下では新たにメタモデル間に関係を導入しなければ、異なる設計法で開発されたプロダクトの統合は扱えない。
2. メタモデルの構築に手間がかかる。新しい設計法を支援環境に追加しようとする場合、その設計法を表現するメタモデルを作成したり、すでに組み込まれているすべての設計法のメタモデルとの関係を定義しなければならない。
3. 手順の視点をほとんど考慮していない。次に何を行えばよいかの情報を作業者に提示することは重要な支援の一つである。これを行うためには、手順に関する情報をモデル化する必要がある。

本論文では上記のような問題に対し、以下の節で述べるようなアプローチ—共通概念を用いたメタモデル(共通メタモデルと呼ぶ)と作業手順情報のモデル化—を行う。

2.2 共通概念の導入

設計法が持っている概念の名前は異なっている、本質的には同じものを指す概念もある。例えば、Booch⁴⁾によれば、オブジェクト指向法の“Object”と

* ここでのプロダクトは最終的に得られる仕様だけでなく、作業の途中で作成される中間生成物も含む。

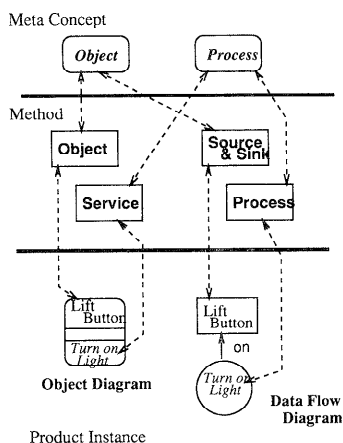


図2 プロダクトの統合機構

Fig. 2 Mechanism for integrating products.

いう概念は、データフロー図では“Data Store”（データベース）や“source&sink”（外界とのインタフェース）に対応する。つまり、Objectのカテゴリに属するプロダクト部分をデータフロー図ではData Storeもしくはsource&sinkと見なすことができることを意味する。図2に具体例を示す。図に示すようにオブジェクト指向分析法で作成されたオブジェクト図中に“Lift Button”というラベルのふられたobjectがあり、同一システムを記述したデータフロー図（Data Flow Diagram）中に同じ“Lift Button”というラベルのふられたsource&sinkがあれば、それらはシステムの同じ構成要素を表していると考えてもよい。このとき、二つの仕様は仕様中に共通に含まれているLift Buttonという構成要素を介して連結される（図2）。また、オブジェクト図が持っているserviceという概念は、そのobjectが持っている機能に該当するため、データフロー図中ではprocess概念に対応すると考えられる。従って、object “Lift Button”が持っているservice “Turn on Light”（ボタンのランプを点灯する）は、データフロー図中のprocess “Turn on Light”と連結することができる。ここでの連結は、そのプロダクトの部分につけられたラベル（例えば、“Lift Button”や“Turn on Light”など）が同じで、しかもプロダクト部分の概念が対応しているときに行われる。このように共通概念を用いて、一つのシステムについて異なる設計法で作成された仕様、つまり異なる視点で作成された仕様を統合することが可能となる。

このような設計法に共通する概念をあらかじめ用意しておくことにより、設計法のメタモデルの構築も能率良く行える。新しく設計法を支援環境に追加しようと

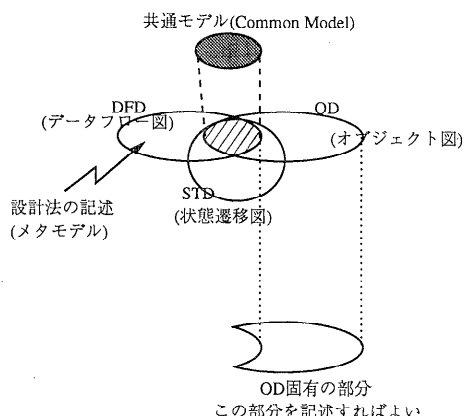


図3 共通概念を用いたメタモデリング法

Fig. 3 Meta modeling based on the common model.

した場合、そのメタモデル全体を記述する必要はなく、設計法のどの概念がどの共通概念に該当するかを記述し、共通概念以外の設計法固有の部分定義すれば、設計法のプロダクト部分のメタモデルが得られる。つまり、共通部分を設計法のプロダクト部分のスーパークラス、追加する設計法をそのサブクラスと考え、その差分を記述すればよい。さらに、新たに追加する設計法とあらかじめ支援環境に組み込まれているすべての設計法との関係を定義するのではなく、プロダクト統合のためには共通概念との対応を定義さえすればよい。このようにモデル化に際しての労力も軽減できる。以上の概要を図3に示す。図中で例えば、ODのメタモデルを定義するにはそのすべてを記述する必要はなく、共通概念との対応と右端の楕円の白抜き部分（ODのメタモデルの共通メタモデル以外の固有の部分）を記述さえすればよい。具体例は第3章の図8で述べる。また、どのような共通概念を抽出したかについても、第3章で述べる。

2.3 作業手順情報のモデル化

作業者に次に何をすればよいかの情報を提示するには、設計法の作業手順をモデル化し、蓄積しておく必要がある。設計法が規定する作業手順は逐次的な順序関係がほとんどであるため、簡単な順序関係が記述できるような手法であればよい。作業手順の提示だけでなく、実際に作業者がどのような作業を、どのような順序で行っていったかという作業履歴を作業中に保持できるようなモデルにしておくことは、作業のやり直しを支援したり、設計理由（Design Rationale）を抽出したりする上で重要である。

以上の点を考慮し、プロダクト部分のメタモデルと同様に実体関連モデルを用いて、図1のオブジェクト

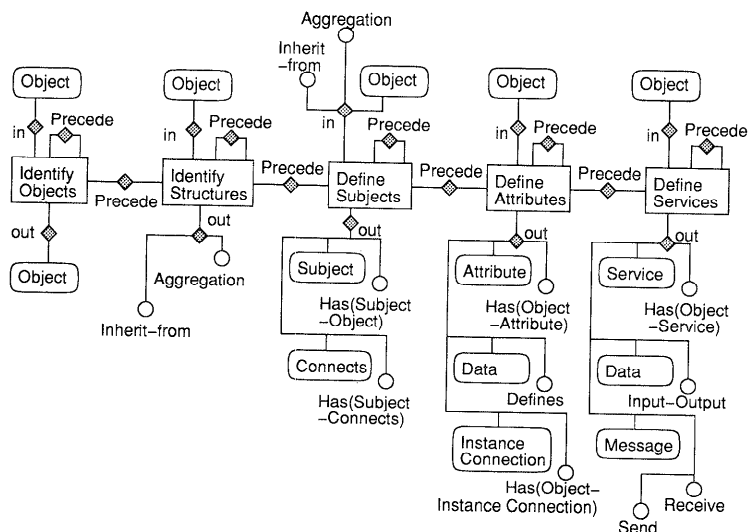
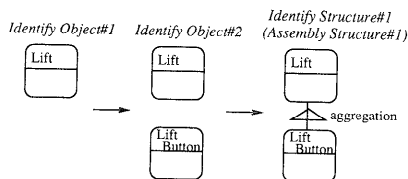


図4 オブジェクト指向分析法の作業手順

Fig. 4 Procedure in OOA.



(a) 作業例

(a) An example of activities

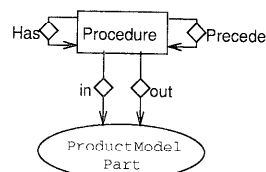
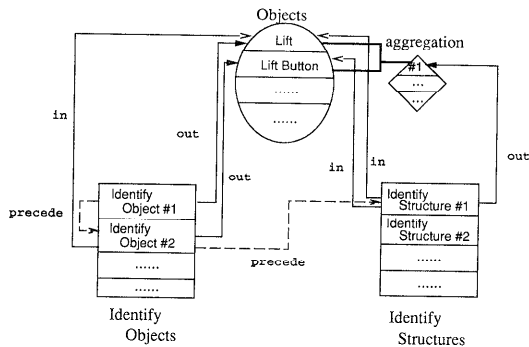


図6 手順を表すメタモデルの一般的な構造

Fig. 6 Structure of a meta model for method procedures.



(b) 作業履歴の蓄積

(b) Recording activities

図5 作業例と作業履歴

Fig. 5 An example of activities and its record.

指向分析法を手順の視点からモデル化した例を図4に示す。図中では、長方形は一つの作業を表している。作業を表す実体は三種類の関係“in”、“out”と“precede”を持っている。“in”と“out”はどのプロダクトがどの作業に入力されるか、どのプロダクトが出力されるかをあらわす。すなわち、“in”と“out”はプロダクト

と作業手順とを連結する役割を持っている。このように作業の入出力関係を導入したのは、その作業を行うのに必要なものは何だったか、その作業によって何が得られたかの情報を残すためである。“precede”は作業の順番、つまりオブジェクト指向分析法中の手順を表している。設計法の作業順序はほとんどこのような単純な順序関係であるので、作業を実体関連モデルの実体で表し、順序関係を関連で表現することができる。図4では、“Identify Objects”(オブジェクトを識別する)という作業からはじめて、対象世界のオブジェクトを抽出する。次の作業は“Identify Structures”で、抽出したオブジェクトを用いてオブジェクト間の関係を抽出する。

図5に作業の簡単な例と、その履歴が図4のモデルに従ってどのように蓄積されていくかを示す。この例は、まず“Lift”オブジェクト、“Lift Button”オブジェクトをこの順で抽出し、その後これらのオブジェクトの間の関係“aggregation”を抽出するという作業である。

図6に作業手順を表現するメタモデルの一般的な構

造を実体関連図で表現したものを示す。図中の Procedure が “Identify Objects” といった一つの作業を表す。一つの作業がより細かい複数の作業に分割されていることがある。そのような階層構造を記述するために、Procedure 間の関連 Has が用意されている。

本研究の目的はプロダクト統合にある。作業順序まで考慮すると、手順は設計法によって異なり、統合に利用できるような共通する部分を抽出できるかどうかは難しい問題である。従って、本稿では手順に関するメタモデルの共通部分を抽出し、それをプロダクト統合に用いるという事は行わない。

3. 統合のためのメタモデル

前節でも述べたように、設計法ごとに設計法が持っている概念名は異なっているが、本質的には同じもの、つまり共通概念が存在する。まず、種々の設計法を調査し、設計法が持っている固有の概念を抜き出す。具体的には、設計法は自然言語で記述されているため、その教科書やマニュアルの記述より、キーワードを抜き出し、その中でプロダクトやプロダクトの部分を表す名詞をその設計法固有の概念とした。実際の適用事例¹⁴⁾に基づき、異なる設計法で概念間の対応があるかどうかを検討し、有効と思われる共通概念としてまとめる。図7に共通概念とその間の関係を示す。

各種の設計法のプロダクトが含んでいる共通概念は、“State”, “Event”, “Process”, “Data”, “Object”, “Association”の六つとこれらの間の関係からなる。この六つの概念は以下の三つのカテゴリに分類される。

1. 動作的側面

システムの時間的な振舞いをモデル化するための概念。State と Event。

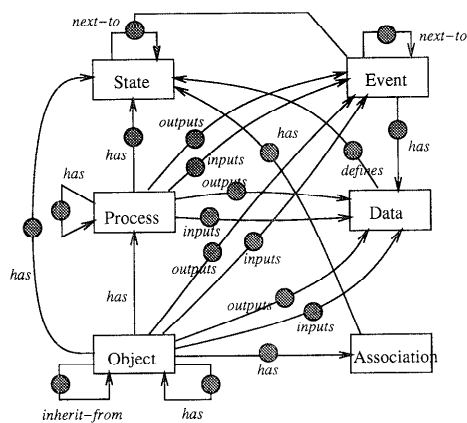


図7 共通概念を用いて構成したメタモデル

Fig. 7 Meta model constructed of common concepts.

2. 機能的側面

システムの機能をモデル化するための概念。システムの機能単位を表す Process とそれに対する入出力となる Data。

3. 構造的側面

システムの物理的な構造をモデル化するための概念。それ自身で存在可能な物理的要素である Object とそれらの間の関係を表す Association。

図8にオブジェクト指向分析法のプロダクト固有の部分を示す。共通概念およびその間の関係で、オブジェクト図で使用する際の名前が異なるものも、固有の部分として名前の付け換えを行っている。例えば、Object 間の関係 Has は、オブジェクト図では集約関係 Aggregation という名前を使用するため、改めて名前の付け換えを行っている。共通メタモデルにはないオブジェクト図固有の概念は、Subject 概念と、Subject-Object 間の関係 Contains である。オブジェクト図を支援環境でできるように登録するには、図8のような共通メタモデルにはない固有の部分

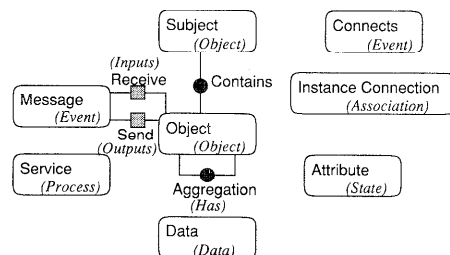


図8 オブジェクト指向分析法の固有部分

Fig. 8 Method-specific parts of OOA.

表1 設計法の概念と共通概念との対応

Table 1 Relationship between method-specific concepts and common concepts.

	設計法の概念	対応する共通概念
オブジェクト図 (OOA ²⁵⁾)	Object	Object
	Subject	Object
	Service	Process
	Message	Event
	Instance Connection	Association
データフロー図	Process	Process
	Mini-Spec	Process
	Data Flow	Event
	Data Store	Object
	Source & sink	Object
	Data Dictionary	Data
状態遷移図	State	State
	Input Event	Event
	Output Event	Event

義し、図4の手順の視点からのメタモデルを定義すればよい。共通メタモデルを用いない場合は、1)図1,2)図4のメタモデルの定義と、3)すでに登録されている設計法すべての概念と図1中に出現している実体との対応関係の定義の三つが必要であるが、共通メタモデルを用いることにより定義しなければならない部分を減らすことができる。

表1にオブジェクト図、データフロー図、状態遷移図が持っている概念と共通概念の対応関係を示す。

4. 支援ツールのプロトタイプ

前節までに述べてきた共通メタモデルの効用を確認するために、オブジェクト図、データフロー図、状態遷移図の三つのメタモデルを、共通メタモデルを用いて作成し、支援環境のプロトタイプをUNIXワークステーション上に作成した。作業者がプロダクトの入力・編集作業を行いやすいように、専用の図エディタを作成し、それを用いて、メタモデルに従ってプロダクトを作成できるようにした。従って、この部分が通常のCASEツールに該当する部分と考えることができる。このプロトタイプの実現方法や機能は、本論文の主旨ではないので、最小限の説明にとどめる。

専用のエディタを起動すると、図9のようなウィンドウがワークステーションの画面上に現れる。これらのエディタは、作業者が選択した設計法に従って仕様を作成するためのツールで、仕様を入力編集するためのエディットウィンドウと作業手順をガイドするガイドウィンドウの二つから構成されている。図の例は、リフト制御システム¹⁵⁾の開発例で、Coad&Yourdonの手法に従ってオブジェクト図を作成している。図の左半分が入力編集用のエディットウィンドウ、右半分がガイドウィンドウである。プロダクトの入力・編集はエディットウィンドウ上で、アイコンメニューを選択することによって行うことができる。例えば、図9のウィンドウの上部最左端のオブジェクトを表すアイコンをクリックすると、オブジェクトの入力が行える。

ガイドウィンドウは、設計法の作業手順を2.3節で述べたような記述に従って、作業者に見やすい形式で表示する。作業者が現在行っている作業は、ガイドウィンドウ中で表示が反転され、楕円で囲まれて表示される。次に行う作業の表示も反転され、これにより作業者は次に何を行えばよいかを知ることができる。表示が反転された作業の右側には、その作業で作成されるプロダクト(図4中の“out” relationship でつながれ

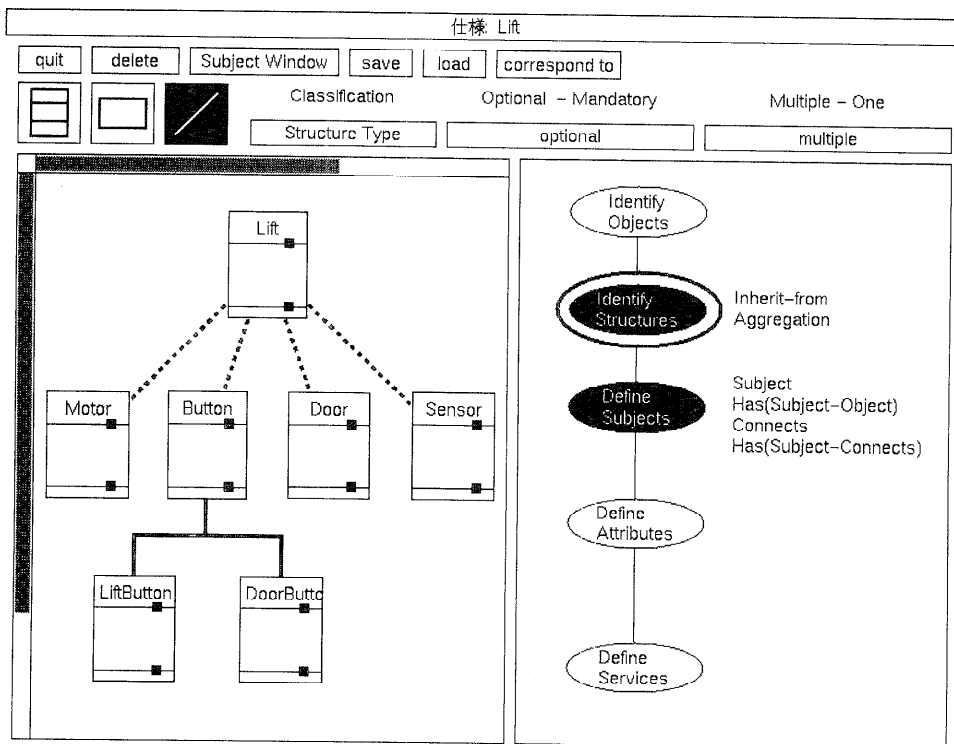


図9 オブジェクト図のエディタ
Fig. 9 Object diagram editor.

ているプロダクト)が表示される。図9の例では、枝を表すアイコン(左から3番目の反転表示された)をクリックしているため、現在 Identify Structures (集約関係や一般化関係などのオブジェクト間の関係を入力する)作業を行っていると判定され、この作業と次に行う作業 Define Subjects が反転表示されている。また、これらの右側には出力となるプロダクト、例えば Identify Structures の場合は一般化関係を表す Inherit-from と集約関係を表す Aggregation が出力として表示されている。

ガイドウィンドウは手順を表示するだけで、作業者の実際の作業手順を強制する機構は持たせてはいない。実際の開発作業では、作業の後戻りなどのプロダクトの修正が頻繁に起こるため、作業者は自由に入力編集作業が行えるようになっている。

3章で述べた複数の方法論で作成されたプロダクトの連結が行われた例を図10にあげる。例では、作業者は Lift の仕様化をオブジェクト図とデータフロー図を使って行っている。作業者がオブジェクト図(左上

のエディットウィンドウ)中のオブジェクト“Lift Button”をクリックすると、データフロー図(右下のウィンドウ)の対応する Source&Sink の“Lift Button”も反転し、これが二つの仕様を結び付けている要素の一つであることを作業者に知らせる。複数の仕様を結び付けている要素を作業中に削除しても、削除を行ったエディットウィンドウ内のプロダクト以外には、その影響は及ばない。例えば図10で、オブジェクト図のエディットウィンドウ内の“Lift Button”を削除しても、データフロー図中の“Lift Button”も自動的に削除されてしまうことはない。

オブジェクト図中の Lift Button をクリックした際、まだデータフロー図のエディットウィンドウが表示されていない場合は、画面上にウィンドウが表示され、対応する要素を含むデータフロー図が表示される。データフロー図がまだ作成されていない場合は、データフロー図用のエディットウィンドウが表示され、ウィンドウ内に Source&Sink の“Lift Button”と Data Store の“Lift Button”が表示されるのみである。これ

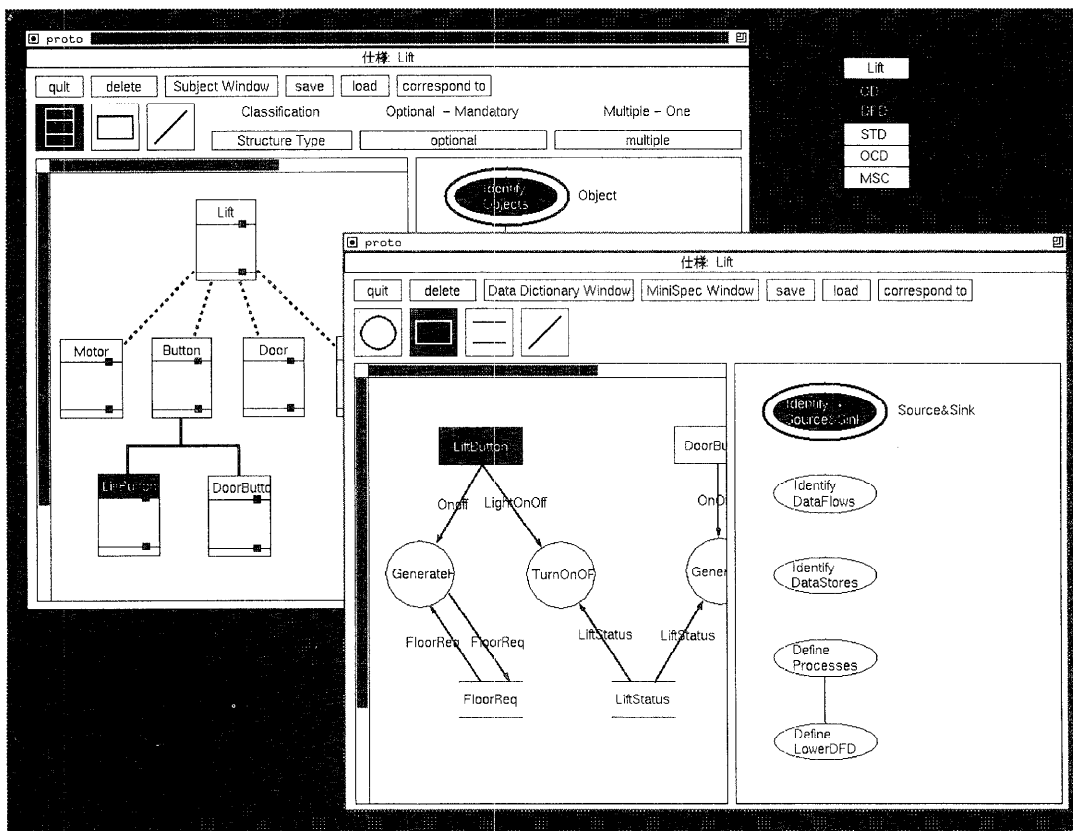


図10 オブジェクト図エディタとデータフロー図エディタ
Fig. 10 Object diagram editor and data flow diagram editor.

は、オブジェクト図のオブジェクトがデータフロー図中では Source&Sink と Data Store の両方に対応しているため、どちらにするかはユーザが不適切なほう（この例では Data Store の Lift Button）を選択して、削除する。

5. 議論

本章では、プロトタイプの実現（メタモデルの作成も含む）、およびプロトタイプの運用例から得られた、本手法の有効性と問題点について議論する。

5.1 設計法の統合

本手法を用いることにより、複数の設計法を組み合わせることで新しい設計法を作ること、いわゆる設計法の統合 (Method Integration)¹⁶⁾ も可能である。個々の設計法が持っている概念をどのように共通概念に対応づけるかが、それらの設計法をプロダクトレベルでどのように組み合わせるかを表している。例えば、表1の対応に従って、試作された4章のプロトタイプは、オブジェクト図、データフロー図、状態遷移図の三つを用いる新しい設計法とその支援ツールと見なすこともできる。設計法の統合の別の例として、オブジェクト図、オブジェクト通信図 (Object Communication Diagram)、状態遷移図、データフロー図といった四つの設計法を統合することにより、Shlaer-Mellor のオブジェクト指向分析法⁹⁾なども作り出すことができる。

しかし、設計法の組み合わせ方としては、本メタモデルではプロダクトのどの部分が共通に出現するかということしか記述していないため、組み合わせたときに生じる、異なる設計法で作られたプロダクト間の一貫性を保つための制約などは記述できない。

5.2 ツールの統合

本メタモデルを用いると、プロダクトを介した異なる設計法の支援ツールの統合が行える。例えば、オブジェクト図エディタ中のオブジェクトをクリックすると、オブジェクトに対応する概念を持つ設計法のエディタ、この場合はデータフロー図エディタが起動し、データフロー図の入力・編集作業が行えるようになる。

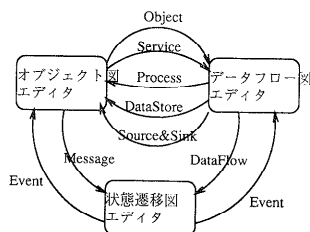


図 11 ユーティリティの遷移図

Fig. 11 Editor transfer chart.

図 11 に、オブジェクト図エディタ、データフロー図エディタ、状態遷移図エディタの三つの、プロダクトを介した遷移図を示す。例えば、データフロー図上で、Process, DataStore, Source&Sink のいずれかをクリックすれば、オブジェクト図エディタに戻ることができることを表している。この図は、表 1 で示した各方法が持っている概念と共通メタモデルとの対応関係から導き出される。このようなプロダクトを介したエディタの遷移は作業者にとって自然であり、操作しやすいであろう¹⁷⁾。

5.3 メタモデルの記述能力

作成したプロトタイプのエディタ部分は、各々の設計法に合わせてその都度作成した。これは本手法のメタモデルがプロダクトの記法（例えば、オブジェクト図ではオブジェクトは Rounded Box で書くなど）に関する情報を持っていないためである。ほとんどの設計法で用いる図的記法は基本的にはグラフ構造（節と辺）をしているため、一般化を行い、メタモデルに組み込むことが可能と思われる。記法に関する情報があれば、図エディタの自動生成系¹⁸⁾を用いて、設計法固有のエディタの生成が行えるであろう。

一つの設計法で作成されたプロダクト内での無矛盾性を保つための制約条件（例えばデータフロー図で、Process の入出力データフローと、その Process を下位レベルのデータフロー図に分解した際の下位データフロー図の入出力とが一致しているといった制約など）のチェックは、その設計法のエディタ内で可能であるが、異種の設計法で作られたプロダクト間の制約は統合の仕方に依存しているため、あらかじめエディタ内に組み込んでおくことができない。述語論理などを用いた制約の記述^{10),19),20)}をメタモデルに追加する必要がある。

6. あとがき

本論文では、各種の設計法で作成されたプロダクトを統合するためのメタモデルを提案した。各設計法を記述したメタモデルは、プロダクト部分だけでなく、作業部分の記述を持っており、この記述に従って作業の履歴が蓄積される。この記録を分析することにより、設計法を改良したり、専門家のノウハウ技術を抽出したりすることができると考えられる。作業手順部分のメタモデルは、作業者へのガイドと、作業の履歴を蓄積するためにしか使用しておらず、作業中に後戻りがあった際の支援など、積極的な活用が必要である²¹⁾。また、作業手順には記載されていないような作業のノウハウ等の提示など、作業者へのガイド法の洗練も重要

な課題である。

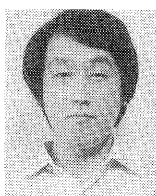
種々の設計法を蓄積し、必要に応じて組み合わせて使用できるような「設計法のデータベース」をメソッドベース (Method Base)^{22),23)} と呼ぶ。本研究で提案したメタモデルは、このメソッドベースに格納する基本的なデータでもある。PCTE²⁴⁾ のような標準的なツールを用いて実用的なメソッドベースを構築していくことも今後の課題である。

参 考 文 献

- 1) DeMarco, T.: *Structured Analysis and System Specification*, Yourdon Press (1978).
- 2) Yourdon, E. and Constantine, L. L.: *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*, Prentice-Hall (1979).
- 3) Shlaer, S. and Mellor, S. J.: An Object-Oriented Approach to Domain Analysis, *ACM SIGSOFT Software Engineering Notes*, Vol. 14, No. 5, pp. 66-77 (1989).
- 4) Booch, G.: Object-Oriented Development, *IEEE Trans. on Soft. Eng.*, Vol. 12, No. 2, pp. 211-221 (1986).
- 5) Harel, D., Lachover, H., Naamad, A., Pnueli, A., Politi, M., Sherman, R. and Shtul-Trauring, A.: STATEMATE: A Working Environment for the Development of Complex Reactive Systems, *Proc. of 10th ICSE*, pp. 396-406 (1988).
- 6) Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lonrensen, W.: *Object-Oriented Modeling and Design*, Prentice-Hall (1991).
- 7) Smolander, K., Lyytinen, K., Tahvanainen, V. P. and Marttiin, P.: MetaEdit—A Flexible Graphical Environment for Methodology Modelling, *Proc. of 3rd Int. Conference CAiSE91, LNCS498*, pp. 168-193 (1991).
- 8) Sorenson, P., Tremblay, J. and McAllister, A.: The Metaview System for Many Specification Environments, *IEEE Software*, Vol. 2, No. 5, pp. 30-38 (1988).
- 9) Alderson, A.: Meta-CASE Technology, *Lecture Notes in Computer Science 509*, pp. 81-91 (1992).
- 10) Brinkkemper, S.: *Formalisation of Information Systems Modelling*, Thesis Publisher (1990).
- 11) 鯉坂：ソフトウェア意味要素の再利用, 情報処理学会ソフトウェア再利用技術シンポジウム, pp. 73-82 (1992).
- 12) 大槻：ソフトウェア仕様記述モデルの形態学, 情報処理学会ソフトウェア工学研究会, Vol. 71 (1990).
- 13) Harmsen, F. and Brinkkemper, S.: Computer Aided Method Engineering, *Proc. the 4th Workshop on the Next Generation of CASE Tools*, pp. 125-140 (1993).
- 14) 古宮ほか：仕様記述過程モデル化のための実験と分析, 情報処理学会ソフトウェア工学研究会, Vol. 89, No. 101, pp. 1-8 (1989).
- 15) Problem Set for the 4th International Workshop on Software Specification and Design, *Proc. 4th International Workshop on Software Specification and Design* (1987).
- 16) Kronl f, K. ed.: *Method Integration—Concepts and Case Studies*, Wiley (1993).
- 17) Brinkkemper, S.: Integrating Diagrams in CASE Tools through Modelling Transparency, *Information and Software Technology*, Vol. 35, No. 2, pp. 101-105 (1993).
- 18) Mitsuda, N., Ajisaka, T. and Matsumoto, Y.: A Semantic-Directed Graph Editor on PCTE, *Proc. Joint Conference on Software Engineering'93*, pp. 69-76 (1993).
- 19) Nuseibeh, B., Kramer, J. and Finkelstein, F.: Expressing the Relationships between Multiple Views in Requirements Specification, *Proc. the 15th ICSE*, pp. 187-196 (1993).
- 20) Saeki, M. and Kuo W.: Specifying Software Specification & Design Methods, *Lecture Notes in Computer Science 811 (CAiSE'94)*, pp. 353-366, Springer-Verlag (1994).
- 21) 島：ソフトウェア設計における設計履歴情報の蓄積, 活用法, 情報処理学会ソフトウェア工学研究会, Vol. 81, pp. 25-32 (1992).
- 22) Saeki, M., Iguchi, K., Kuo W. and Shinohara, M.: A Meta-Model for Representing Software Specification & Design Methods, *Information System Development Process*, pp. 149-166, North-Holland (1993).
- 23) Slooten, K. and Brinkkemper, S.: A Method Engineering Approach to Information Systems Development, *Inf. Syst. Development Process*, pp. 167-186, North-Holland (1993).
- 24) Wakeman, L. and Jowett, J. eds.: *PCTE: The Standard for Open Repositories*, Prentice Hall (1993).
- 25) Coad, P. and Yourdon, E.: *Object-Oriented Analysis*, Prentice Hall (1990).

(平成 6 年 8 月 1 日受付)

(平成 7 年 3 月 13 日採録)



佐伯 元司 (正会員)

1956年生. 1978年東京工業大学工学部電気電子工学科卒業. 1983年同大学院情報工学専攻博士課程修了. 工学博士. 同年同大情報工学科助手. 1988年同大電気電子工学科助教授. 同情報工学科助教授を経て, 現在同大学院情報理工学研究科計算工学専攻助教授. ソフトウェアの仕様記述法, ソフトウェアプロセスなどの研究に従事. 電子情報通信学会, IEEE, ACM 各会員.



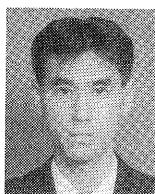
郭 文音 (正会員)

1968年生. 1992年東京工業大学工学部情報工学科卒業. 1994年同大学院理工学研究科電気電子工学専攻修士課程修了. 在学中, ソフトウェアの仕様化・設計法の研究に従事. 同年より日立製作所・ビジネスシステム開発センタ勤務.



井口 和久 (正会員)

1969年生. 1991年東京工業大学工学部情報工学科卒業. 1993年同大学院理工学研究科電気電子工学専攻修士課程修了. 在学中, ソフトウェアの仕様記述法の研究に従事. 同年NHKに入局. 現在, 同放送技術研究所次世代テレビ方式研究部に勤務. 動画処理の研究に従事. テレビジョン学会会員.



篠原 正紀

1970年生. 1993年東京工業大学工学部情報工学科卒業. 1995年同大学院理工学研究科情報工学専攻修士課程修了. オブジェクト指向分析・設計法の研究に従事. 同年NTTに入社.