ソフトウェア工学 85－6
（1992. 5. 21）

# ソフトウェア仕様化/設計法のデータベース化について

郭 文音　　佐伯 元司

東京工業大学 工学部 電気電子工学科

本論文では、種々のソフトウェア仕様化/設計法を蓄積するためのデータベースについて述べる。我々はこのようなデータベースをメソッドベースと呼んでいる。このシステムは、設計者が選択した設計法に従って設計者の作業をガイドする。このため、設計過程で作られたプロダクトや作業活動を記録する機構を持っていなければならない。さらに、設計者が途中で設計法を変えたり、異なる設計法を用いて生成されたプロダクトを統合したりするため、設計法同士の意味的な関係も保持されていなければならない。我々は、まず実体関連モデルを用いて、個々の設計法を形式的に表現する手法について述べる。この表現がプロダクトや作業活動を蓄積するためのデータモデルである。さらに、プロダクトや作業活動の他に、形式的に表現された設計法をデータとして、メソッドベースに蓄積するためのデータモデル（メタモデルという）について述べる。

# Database of Software Specification & Design Methods

Kuo Wen-yin　　Motoshi Saeki

Department of Electrical and Electronic Engineering
Tokyo Institute of Technology

This paper discusses a certain type of database systems where various software specification & design methods are stored. We call it a method base system. The designer will be lead through a design process step by step, according to the method which he selected. Subsequently, a method base system can hold the information about the produced documents and about the performed activities in the design process. Furthermore, the semantical relationships among the stored design methods should be stored in a method base system so that the designers can change his design methods in the process. We have developed the technique to make formal representations of design methods by using the Entity-Relationship model(ER model). These formal representations as well as the products and the actual design activities can be considered as data to be stored in a method base system. Therefore we have constructed a logical data model (we call it a *meta model* of these formal representations of the design methods.

# 1 まえがき

ソフトウェア開発の過程において、仕様化/設計は初期の段階に位置する。従って、効率的にソフトウェアを開発するためには、誤りや不備のないソフトウェア仕様化/設計を行わなければならない。ソフトウェア仕様化/設計を支援するため、ジャクソン開発システム [1] やオブジェクト指向分析/設計法 [2] や構造化分析/設計法 [3] といった種々のソフトウェア仕様化/ 設計法（以下、単に設計法という）が提案されている。

しかし、これらの設計法はあらゆる分野のソフトウェアシステムの設計に適しているというわけではない。例えば、構造化分析法 [3] 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 [4]。すべての分野のソフトウェアシステムの設計に適した一つの万能の設計法を作ることは難しいと思われる。それよりも、ソフトウェアシステムの分野に合わせて、各々適した設計法を利用したほうが現実的であろう。つまり、任意の段階で対象領域にふさわしい設計法を選択して使い分ける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、必要なとき任意の設計法を利用できるデータベースシステムが必要である。

本論文では、このようなデータベースシステム (—メソッドベースシステムと呼ぶ [5]—) を提案する。一つの設計法に従って生成されたドキュメントを蓄積管理するデータベースシステムは開発されているが [6]、メソッドベースシステムでは、設計法に従って生成されたドキュメントや設計作業履歴を蓄積するだけでなく、様々なソフトウェア仕様化/設計法そのものを蓄積する。そのため、仕様化/設計法をデータとして扱うためのデータモデルが必要である。本論文では、仕様化/設計法の表現法及びそれを蓄積するためのデータモデルを提案し、議論する。設計作業履歴を Design Rationale(設計作業中に下した判断の理由) という観点から蓄積するための一般モデルは提案されているが [7]、設計法自身でなく作業プロセスを対象としている。

# 2 メソッドベースシステムに対する要求

メソッドベースシステムにしてどんな要求があるかを実際のユーザ (設計者) の立場から考えてみよう。図 1はユーザとメソッドベースシステムのインタラクション列の例を示したものである。まず、メソッドベースシステムはユーザに対し、蓄積してある仕様化/設計法をメニュとして提示してくれる。ユーザはメニュから設計法、例えばジャクソン開発法を選ぶ。設計法が選択されると、メソッドベースシステムはその設計法に従ってユーザをガイドする。つまり、ユーザはメソッドベースが提示してくれる作業手順に従ってソフトウェアを設計していく。例えば、ジャクソン開発法が選択されている場合では、メソッドベースシステムはジャクソン開発法の最初のステップ-"イベントを列挙する (enumerate events)" という作業をするようにしてユーザに表示する。ユーザはこのメッセージに従ってイベントを列挙し、メソッドベースシステムに入力していく。

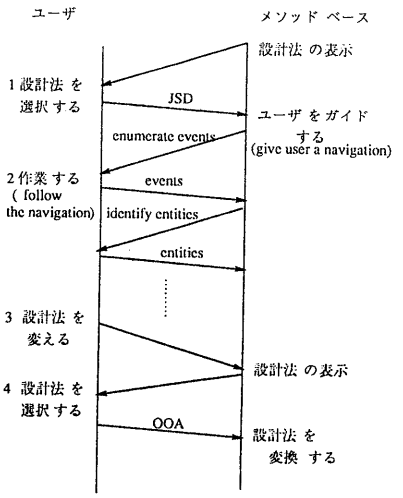


図 1: メソッドベースの使用

メソッドベースシステムは作業の結果得られた生成物 (プロダクトという) を整理、蓄積する機能も持っている。ユーザは、このようにメソッドベースシステムが出力する作業手順のメッセージに従いながら、設計結果を順次入力していく。

設計作業中に、ユーザは自分の選択した設計法が適当でなかったと気づくことがある。この場合ユーザは設計法を変えることができる。その際、今まで行なってきた作業の結果を無駄にしないためにも、メソッドベースがプロダクトを選択された設計法に従った構造に変換してくれることが望ましい。例えば、ユーザが何らかの理由でジャクソン開発法が不適切であったと気づいたとする。その時、ユーザはメソッドベースシステムから、設計法のメニューをもう一度もらい、他の設計法、例えばオブジェクト指向分析法 (OOA) を選択する。メソッドベースシステムは今までの作業結果であるイベントのリストを対応するオブジェクト指向分析法のプロダクトに変換してくれる。

以上のことをまとめると、メソッドベースシステムは以下のような機能を持っていることになる。

1. 設計法が蓄積され、いつでもユーザの指示に応じて設計法が選択できること。
2. ユーザを設計法の作業手順に応じてガイドする、つまり作業を支援してくれること。
3. ユーザが途中で設計法を変換できること。

メソッドベースシステムは種々の設計法を蓄積するため、これらのモデル化/表現法を確立する必要がある。設計法のモデル化/表現法は、上記の機能を達成するために、以下の要求を満足する必要がある。

1. 設計作業を支援するため、実際に行なわれた作業の履歴や、設計法に従って生成されたプロダクト (中間生

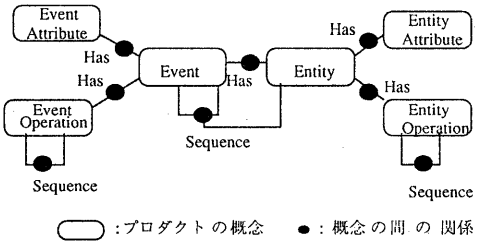


図 2: JSD のプロダクト記述

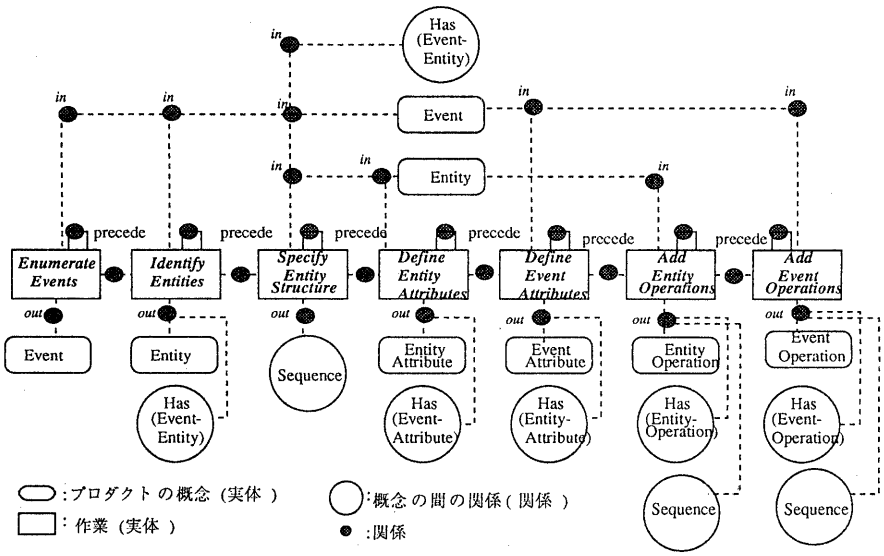


図 3: JSD の作業記述

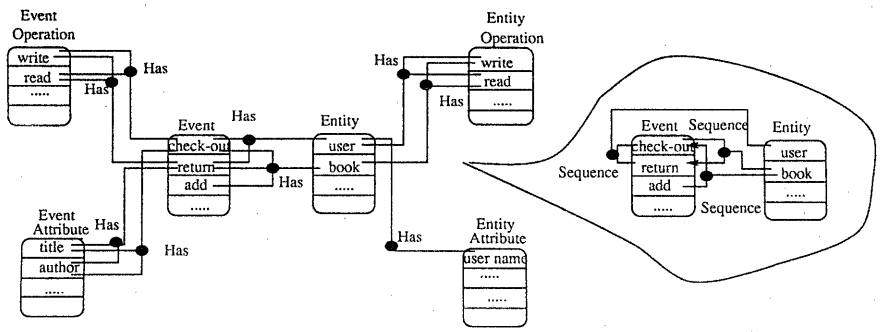


図 4: JSD 法による図書館のプロダクトの例

表 1: ジャクソン開発法の Modeling Stage

| | Activities | Produced Documents |
|---|---|---|
| 1 | Enumerate Events | List of Events |
| 2 | Identify Entities | List of Entities |
| 3 | Specify Entity Structures | Entity Structure Diagram |
| 4 | Define Entity Attributes | Entity Structure Diagram |
| 5 | Define Event attributes | Entity Structure Diagram |
| 6 | Add Operations of Entities | Entity Structure Diagram |
| 7 | Add Operations of Events | Entity Structure Diagram |

成物も含む) が記録されるようなモデル化が必要である [8]。

2. 作業中に現在使用している設計法を別の設計法に変換することを支援するために、個々の設計法は統一的なモデル化がなされていなければならない。

本研究では、データモデル化の一般的手法である実体/関連モデル (Entity/Relationship Model)[9] を用いて設計法を記述する。

# 3 設計法の記述

## 3.1 設計法記述のためのモデル

設計法は必要なプロダクトとそのプロダクトを得るため必要な作業やその手順などを教えてくれる。従ってすべての設計法は、

- プロダクトの視点 : どんなプロダクトが作られるか、つまりプロダクトの構造。
- 作業の視点 : どんな作業をどの順序で行なわなければならないか。

の二つの視点で整理することができる。ジャクソン開発法のデータモデルを例にとって簡単に説明する。

表 1 はジャクソン開発法の Modeling Stage で生成されるプロダクトと作業のリストである。例えば、"Enumerate Events" 作業は問題領域中から event (事象) を抽出し列挙する作業で、その作業の結果 event のリスト (List of Events) が出力される。

以上のようなジャクソン開発法を実体/関連モデルを用いて形式的に表現してみよう。図 2 はプロダクトという視点から、図 3 は設計作業を視点から記述したものである。ここで、図は複雑にならないように、プロダクト部分と作業部分を分けて書いてある。ジャクソン開発法には event や entity(対象)、attribute(属性) などのプロダクトの要素を表す基本的な概念があり、その概念間に関係 (relationships) が存在する。例えば、entity は event を "持っている (has)" という関係がある。

作業を表す実体は三種類の関係 "in"、"out" と "precede" をもっている。"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表す。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。"precede" は作業の順番、つまりジャクソン開発法の手順を表している。図 3 では、"enumerate events" という作業からはじめて、次に "Identify Entity" を行う。この作業は、抽出された event を用いて entity を抽出する作業である。

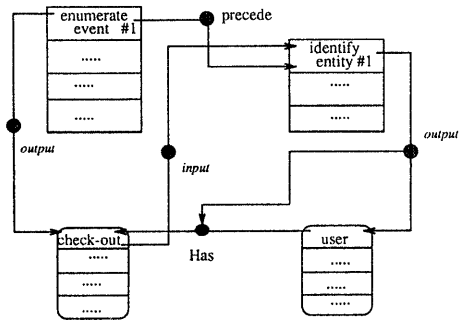


図 5: JSD 法による図書館の作業の例

## 3.2 設計過程の蓄積

図 4 と図 5 は 図 2、図 3 のジャクソン開発法の記述 (データモデル) に従って蓄積された、図書館システム [10] の設計過程の一部である。プロダクトの部分と作業の部分にわけて説明する。

作業の結果得られたプロダクトは、図 4 のように蓄積される。"check-out" や "return"、"add" などは "Event"、"user" や "book" などは "Entity" のインスタンスである。また、"check-out" は その Attribute として "title" や "author" などを、その Operation として "write" や "read" などを持っている。Entity Structure Diagram によって表される Event 列 (Event の生じる順序) は、関係 Sequence (ある event は、ある event よりも先に生じる) で表される。例えば、"book" には三つの Event があり、"add", "check-out", "return" の順で生じる。

図 5は、作業の履歴を図 3の構造に従って蓄積したものである。Enumerate Events のインスタンス "enumerate event# 1" が行われ、Event "check-out" が得られている。次に行われた作業は "identify entity #1" で、これは "check-out" を入力とし、Entity "user" と関係 "Has" のインスタンス ("check-out"—"user") を出力している。

# 4 メソッドベースの論理的構造

前節では、個々の設計法の表現について述べてきたが、この節では、記述された設計法がどのようにして蓄積され、ユーザに使用されるかについて述べる。まず、メソッドベースシステムは、図 6のように二つのレベルから構成される。生成されたプロダクトや作業履歴をデータとして保存しているレベルをオブジェクトレベルと呼ぶ。オブジェクトレベルは、ユーザがソフトウェア仕様化/設計作業を行なうためのレベルである。設計法をデータとして蓄積しているレベルをメタレベルと呼ぶ。メタレベルでのデータモデルは、前節で述べたような設計法記述をデータとして扱うためのモデルであり、これをメタモデルと呼ぶ。メタレベル

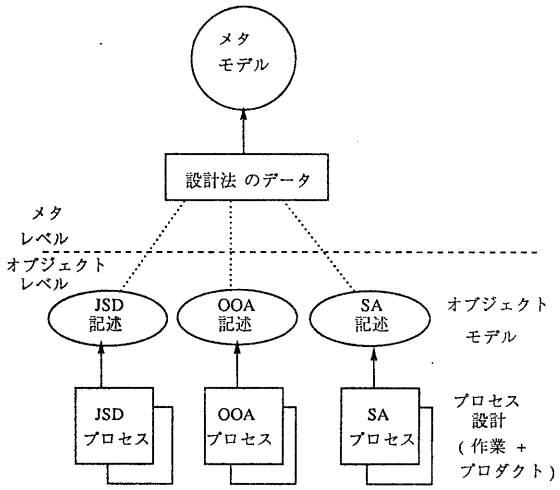


図 6: メソッドベースの論理的なデータ構造

のメタモデルに対して、オブジェクトレベルにあるデータモデルをオブジェクトモデルと名づける。オブジェクトモデルは設計法を前節のように表現したものであり、かつプロダクトや作業履歴蓄積のためのデータモデルである。

ユーザが設計法を選択すると、メタレベルにあるその設計法の記述が検索され、オブジェクトレベルで検索された記述がオブジェクトモデルとして構築される（例えば、図6のオブジェクトレベルの JSD 記述や OOA 記述 、SA 記述など)。生成されたプロダクトと作業履歴は、構築されたオブジェクトモデルに従って記録される。

JSD を例にとり、メタレベル、メタモデル、オブジェクトレベル、オブジェクトモデルの関係を説明する。図7はこれらの関係を表したもので、横軸の上部がメタレベル、下部がオブジェクトレベルである。縦軸の左部が蓄積されているデータ、右部がそのデータモデルを表している。設計法は図7の第2象限 (左上部分) のような形で蓄積されている。例えば、JSD 法がもっているプロダクト概念 (例えば、Event, Entity など)、作業 (例えば、Enumerate Events など) が構造化されて蓄積されている。第一象限 (右上部分) はその構造を記述したデータモデル、すなわちメタモデルで、その詳細については次の節で述べる。ユーザが JSD 法を選択すると、第2象限で示されているようなデータ中を検索し、第4象限 (右下部分) のような JSD のオブジェクトモデル (図6 の JSD 記述) を構築する。ユーザはこのモデルに従って JSD の設計作業を続け、第3象限 (左下部分) のようにその作業履歴とプロダクトがオブジェクトレベルに蓄積される。

## 5 メタレベル

メタレベルでは、設計法の記述をデータとして扱う。また、設計法を変換するためには、設計法間の意味的関係が蓄積されていなければならない。この関係は、ある設計法に従って生成されたプロダクトを別の設計法で使用できるように変換する際、新しい設計法ではそれらがどのプロダクト概念に対応するかを示す。

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。例えば、JSD の “entity” という概念は OOA の “object” と対応している。すなわち、使用している設計法を JSD から OOA に変えると、JSD で得られた “entity” は、そのまま OOA の “object” に変換することができる。

我々はこのような設計法を説明する共通概念を用意し、それを組み合わせて設計法の持つ概念と対応づける。これにより、メソッドベースに新しい設計法を追加する際には、その設計法と蓄積されているすべての設計法との対応関係を考える必要はなく、その新しい設計法を共通概念に分解すればよいだけである。図8に共通概念を利用して構成したメタモデルを示す。メタモデルはオブジェクトモデルの二つの視点—プロダクトからの視点、作業からの視点と同じように二つ部分に分けられている。

### 5.1 プロダクトの部分

種々の設計法を分析し、プロダクト部分に関して六つの共通概念— “ State” と “ Event” (システムの動作を表す概念)、“Process” と “Data” (機能を表す概念)、“Entity” と “Association” (構造を表す概念) と、これらの共通概念の間の１３の関係を抽出した [11]。メタモデルのプロダクト部分は、これらの共通概念とその間の関係から成り、実体/関連モデルを用いてこれを表現すると、図8のようになる。共通概念および概念間の関係は、“実体” として表わされている。メタレベルでは、オブジェクトモデルのプロダクト部分の概念 (Event, Event Attribute など) はこの六つの共通概念のどれかに分類され、プロダクト部分の関係 (Sequence や Has など) はこの図8の “共通概念の間の関係” に分類される。例えば、図8に示しているように、共通概念の関係 “State-state-event” は、Event が起こったら、現在の State から次の State へ変わるという遷移関係のカテゴリを表現している。実体 “Method” のインスタンスは、ソフトウェアの設計法の名前、例えば JSD, OOA などである。

“Method” から出ている関係 “Has” は、設計法がどの概念をもっているかを示している。例えば、関係 “Has” が “Method” と “State” の間にあるということは、“Method” にあるインスタンスが “State” に属する概念を持っているということを表す。図9は、図8のモデルに従って、設計法を蓄積したものの一部である。図9のように、“Method” のインスタンスの一つである JSD は、共通概念 “Event” を表すプロダクト概念名として “Event” をもって、さらに “State” を表す “Entity Attribute” を持っている。OOA では、そのプロダクト概念間の関係である “Inherit-from” は共通概念間の関係 “Entity-entity ” に属し、図中の二本の点線が “Inherit-from” が “Objcet” と “Objcet” の間の関係であることを表している。

### 5.2 作業の部分

作業部分を表すメタモデルは、図8に示すように、“Activity” と “Activity-relationship” とからなる。“Activity” 設計法の作業名、“Activity-relationship” は作業が持って

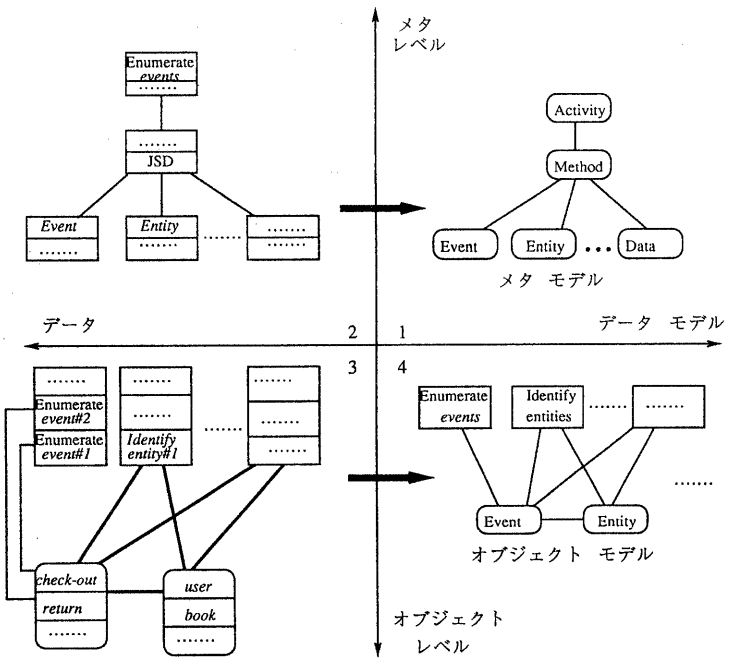


図 7: データモデルとデータ

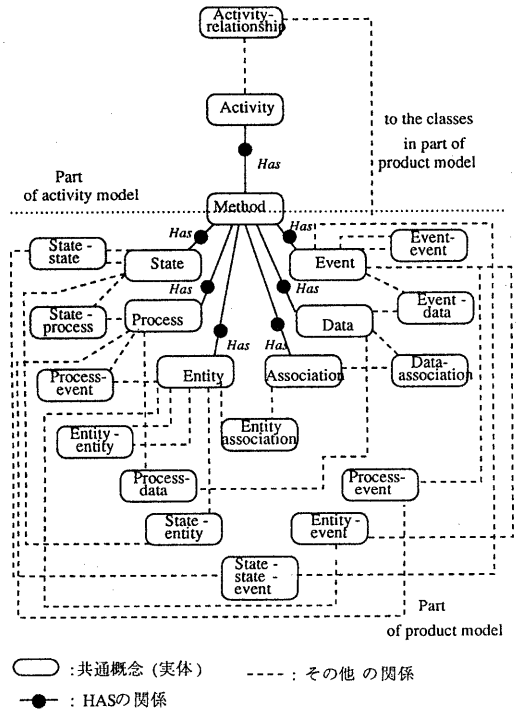


図 8: メタモデル

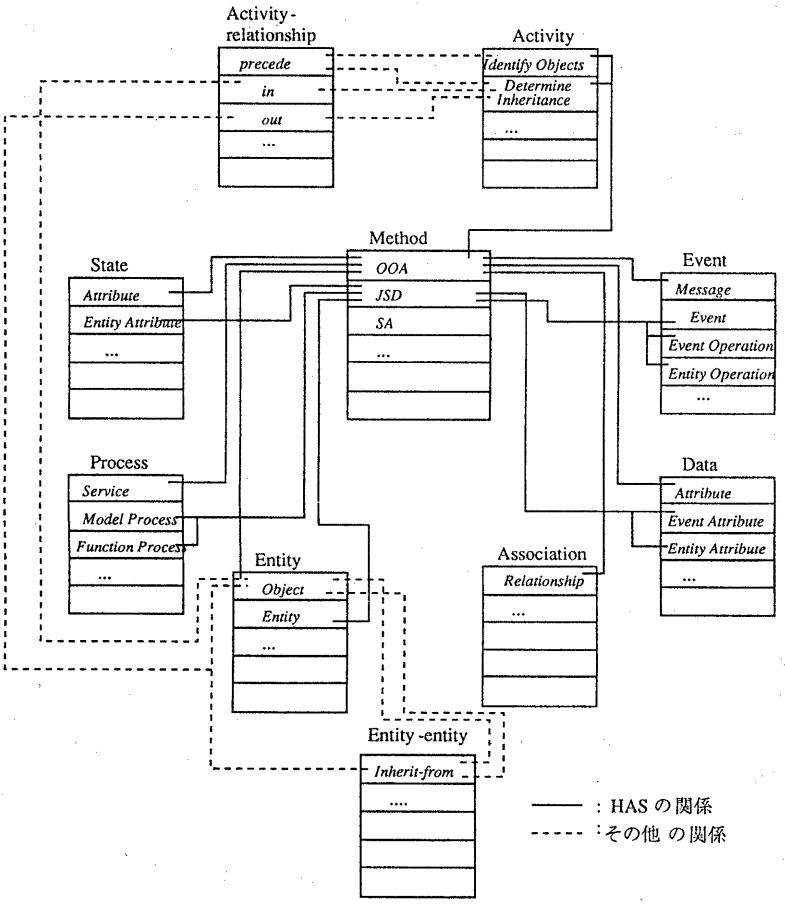


図 9: メタレベルにある設計法のデータ

いる関係を記述する。例えば、図9中のOOAでは、作業"Identify objects"は"Activity"のインスタンスで、関係"precede","in","out"が"Activity-relationship"インスタンスである。作業"Determine inheritance"に注目すると、関係"Has"でMethod"OOA"に関係づけられているのがわかる。また、この作業は"Activity-relationship"のインスタンス"precede"や"in"、"out"を持っていることがわかる。作業"Identify objects"を実行してからこの作業を行なうので、順序関係"precede"が作業"Identify objects"と作業"Determine inheritance"と関係づけられている。入力/出力関係を表す"in"と"out"は、"Determine inheritance"の入出力となっているプロダクトと関係づけられている。

### 5.3 メタモデルを利用した設計法の変換

図9を用いて、設計法の変換メカニズムを説明する。ここで、採用された設計法—JSD、新しい設計法—OOAへ変換すると考えてみよう。オブジェクトモデルはメタレベルで、共通概念とこの概念の関係に分類されて蓄積されている。従って、JSDのプロダクト概念がメタモデル上でどの共通概念や共通概念間関係に属しているかを調べることによって、OOAのプロダクトに変換することが可能となる。例えば、OOAの"Message"は共通概念"Event"に属しており、JSDの"Event"も同じ共通概念"Event"に含まれている。つまり、両者は対応づいていることがわかる。従って、メソッドベースシステムが"Method"と共通概念"Event"の関係をたどって、JSDのEventをOOAのMessageに変更することが可能である。

対応している概念がない場合は、メソッドベースシステムがユーザに、新しい設計法のもとでその概念に対する作業を要求する。例えば、OOAは共通概念"Association"に属するプロダクト概念"Relationship"を持っているが、JSDには対応する概念がない。従って、ユーザはOOA"Relationship"を作る作業、"Identify Relationships"を実行する必要がある。

作業の対応関係については、"Method"と"Activity"の関係によって、対応する作業を見つけ出すことができる。

## 6 あとがき

本論文では、ソフトウェア仕様化/設計作業を効率的行なうために、ソフトウェア仕様化/設計法を蓄積するメソッドベースシステムを提案した。オブジェクトモデルに従って実際の設計のプロセスを記録し、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できると考えられる。

設計者にソフトウェアの設計をガイドすることができるようなメソッドベースシステムは、検索機能を持つハイパーテキストシステム[12]に基づいて実現することができると思われる。しかし、設計対象となっているソフトウェアの分野に適切な設計法を選ぶためには、どの設計法がどの問題分野に適するかを明らかにする必要がある。このソフトウェア設計法と問題分野との関係を明らかにするには、設計法の運用経験を分析する、つまり種々のタイプの問題を設計・記述し、そのプロセスを比較検討していくという実験的手手法が有効であろう[13]。また、設計作業は複数の人間によって行なわれることも非常に多いため、作業の分担方法等[14]を含めて、共同設計作業の支援も考えていくことも今後の課題である[15]。

## 参考文献


[1] M.A. Jackson. *System Development.* Prentice-Hall, 1983.

[2] P. Coad and E. Yourdon. *Object-Oriented Analysis.* Prentice Hall, 1990.

[3] T. DeMarco. *Structured Analysis and System Specification.* Yourdon Press, 1978.

[4] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, and A. Shtul-Trauring. STATEMATE : A Working Environment for the Development of Complex Reactive Systems. In *Proc. of 10th ICSE*, pp. 396–406, 1988.

[5] K. Wen-yin and M. Saeki. Method Base : Database of Software Design Methods & Specifications. In *Joint Conference on Software Engineering '92*, pp. 311–318, 1992.

[6] 外山, 磯田. リポジトリ仕様の比較. 情報処理学会ソフトウェア工学研究会, Vol. 80, pp. 35–42, 1991.

[7] C. Potts. A Generic Model for Representing Design Methods. In *Proc. of 11 th ICSE*, pp. 217–226, 1989.

[8] 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.

[9] P. Chen. Entity-relationship Model : Towards a Unified View of Data. *ACM. Trans. on Database Systems*, Vol. 1, No. 1, pp. 9–36, 1976.

[10] *Problem Set for the 4th International Workshop on Software Specification and Design: Proc. of 4th International Workshop on Software Specification and Design*, 1987.

[11] 佐伯. ソフトウェア仕様化/設計の方法論の形式化について. 情報処理学会ソフトウェア工学研究会, Vol. 72-4, , 1990.

[12] J. Conklin. Hypertext : An Introduction and Survey. *IEEE Computer*, Vol. 20, No. 9, 1987.

[13] 古宮他. 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. 69, , 1989.

[14] 西村, 本位田. 分散環境における仕様化作業とそのプロセス分割基準の検討. 情報処理学会ソフトウェア工学研究会, Vol. 85-7, , 1992.

[15] 垂水. グループウェアのソフトウェア開発への応用. 情報処理学会誌, Vol. 33, No. 1, pp. 22–31, 1992.