

抽象化に基づく文章理解モデルの ソフトウェア設計法への適用

大野 浩史(*) 菊野 亨(**) 鳥居 宏次(**)

(*)日本ユニシス(株), (**)大阪大学基礎工学部

本報告では、自然語で書かれた仕様書が与えられるソフトウェア開発を前提として、開発作業の効率的な支援について考察する。ソフトウェア設計のための具体的方法論として、構造化分析法(SA法)、構造化設計法(SD法)、ジャクソン構造化プログラミング(JSP法)とオブジェクト指向設計法(OOD法)を考える。なお、各方法論の初期ステップの作業内容は複雑で、かつ、高度な内容を含むため、次のステップへの移行が困難になっている。

与えられた仕様書に上述の4つの方法論を適用する前に、文章理解モデルを仕様書に適用して、仕様書(自然語)の意図の正しい理解を確認することを提案する。ここで、文章理解モデルとは自然語で書かれた仕様書の意図を抽出するために、筆者らが開発したモデルである。文章理解モデル上で確認され、構成された意味表現(Kグラフ、Pグラフ、Fグラフ)を利用すれば、ソフトウェア設計のための各方法論における初期の作業を容易化することが期待できる。

An Application of Software Design Method Augmented with Semantic Model to Program Specification in Natural Language

Koji Ohno(*), Tohru Kikuno(**) and Koji Torii(**)

(*)Nihon Unisys Limited, (**)Faculty of Engineering Science, Osaka University

(*)Akasaka 2-17-51, Minatoku, Tokyo 107, Japan

(**)Machikaneyama 1-1, Toyonaka, Osaka 560, Japan

Abstract : This paper discusses efficient and useful supports to software development, in which software is designed for a given specification written in a natural language (Japanese). Typical design methods proposed up to now include structured analysis, structured design, Jackson structured programming and object-oriented-design. Generally speaking, an initial design step is very hard to execute, since it needs sound and complete understanding of a given specification.

To support the initial design step, it is proposed here that, before applying any design method mentioned above, semantic information is extracted based on a semantic model from a given specification. The semantic model, which has already been proposed by authors, consists of three graphs (K-graph, P-graph and F-graph). It is considered that by using semantic information in these graphs, the initial design step is executed effectively with relatively straightforwardly.

1. まえがき

自然語を用いて記述されたプログラム仕様には次の問題が存在することが指摘されている^{[12][14]}。

(N1) あいまいさ、及び、漠然性が含まれる。

(N2) 暗黙の前提の下に、記述の省略が行われる。

従って、同じプログラム仕様が与えられても、システム分析者や設計者毎に自然語の意図している内容の理解が異なり、複数の異なるプログラムが開発される可能性がある。その中の幾つかは間違ったプログラムである危険性もある。そのため、自然語による意図を正しく理解することがプログラム開発の前提として重要となる。

一方、プログラム開発は、通常、要求分析、システム設計、詳細設計、コード化、テスト、保守の6つの段階で行われる^{[8][9]}。この内、要求分析、及び、システム設計は、そこでの作業結果が開発されるプログラムの品質を決定するという意味で、重要な過程である。これらの過程での作業を効率的に支援する目的で多くの方法論、及び、それに基づいたツールが提供されてきている。方法論の具体的な例としては、

- (1) 構造化分析法^[11]
- (2) 構造化設計法^[7]
- (3) ジャクソン構造化プログラミング^[5]
- (4) オブジェクト指向設計法^{[11][15]}

等が知られている。

しかし、これらの方法論は、通常、自然語を用いて例示的に記述されている。従って、開発の現場で具体的に方法論を適用するには、一般に多くの困難さを伴う。

ここでは、この方法論の適用に伴う困難さを軽減、あるいは、解消する試みについて議論する。それには基本的に次の2つの解決策が考えられる。

- (A1) 形式的記述…形式的言語を用いて、方法論を厳密に、かつ、詳細に記述する。^{[3][4]}
- (A2) 自然語理解…方法論の各過程において自然語処理を利用して、自然語のもつあいまいさや漠然性を解消する。

本報告では、上述の4種類の方法論に対しA2を実行する。理想的には、先ずA1を実行した後でA2を実行す

べきであるが、A1については提案が行われているにとどまっており、その結果は未だ利用できない。従って、ここでは文献[1][5][7][11]の方法論に対してA2を実行することにした。なお、A2中で用いる自然語処理は、筆者らがN1、N2の解消を目指して開発した文章理解モデル^[10]に基づいて行う。

2. ソフトウェア設計法

2.1 ソフトウェア開発

典型的なソフトウェア開発の流れを図1に示す。先ず、要求定義によって開発すべきソフトウェアの設計目標を要求仕様書の形で設定する。システム設計では、要求仕様書に基づいて、システムの構造と機能を決定する。引き続き、詳細設計においてアルゴリズムとデータを詳細に定義する。その後、コーディング、テスト、保守へと進んでいく。

ここでは、その内のシステム設計に注目する。システム設計の善し悪しは最終的に作られるシステムの性質にかなり直接的に影響を与える。しかも、この過程での作業手順は必ずしもアルゴリズムとして確立されておらず、経験に基づく直観や勘(ヒューリスティックス)を活かした作業部分が多く含まれている。

2.2 方法論

ソフトウェア設計法(ここでは図1中のシステム設計と詳細設計を指す)を効率的に、かつ、正しく実行するためのガイドラインとして多くの方法論が提案されている。^{[8][9]}ここで取り上げる方法論の概要を図2の(1)～(4)に示す。

システム設計を支援する方法論では“機能”に注目した作業を行う。そこで中心的な役割を果たすものがデータとそれを利用するための基本的な操作である。この2つの概念の定義の解釈をめぐって種々の考え方があり、それが方法論の適用を困難にしている。これらの概念の対応関係を表1に示す。

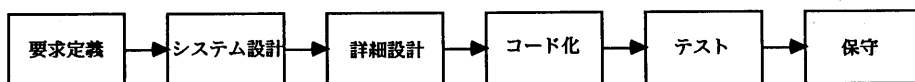


図1 ソフトウェア開発の過程

2.3 提案する枠組み

上で述べた問題点の解消を目指して、図3に示す処理方式を提案する。すなわち、要求仕様書の意図している内容を正しく理解する過程を、要求定義とシステム設計の間に挿入するという提案である。図3中のFグラフとは文献[10]で導入された自然語理解モデルである。

具体的には、図4に示す形式で各方法論との結合を行う。同図でSA法(Step2)とは、自然語理解の結果を利用すればSA法のStep2(図2参照)までは簡単に適用できることを表す。従って、次はStep3から引き続いて実行すればよい。他の方法論についても同様である。

表1 基本概念の対応

本報告、 及び、 文献[10]	SA法、 及び、 SD法	JSP法 (JSD 法 ^[5])	OOD法
対象	目的語	データ (Entity)	オブジェクト
操作	はっきりした 動作を示す1 つの動詞	演算 (Function)	動作

(1) 構造化分析法(SA法)^[11]

- Step1 データフロー図の作成
- Step2 データ辞書の作成
- Step3 プロセス仕様書の作成

(2) 構造化設計法(SD法)^[7]

- Step1 バブル図の作成
- Step2 入出力処理、変換への分割
- Step3 階層構造図の作成
- Step4 詳細化の終了判定

(3) ジャクソン構造化プログラミング(JSP法)^[5]

- Step1 入出力データの構造の決定
- Step2 プログラム構造の決定
- Step3 スキーマ論理の作成
- Step4 プログラムコードの作成

(4) オブジェクト指向設計法(OOD法)^{[11],[15]}

- Step1 オブジェクト、属性の決定
- Step2 オブジェクトの動作の決定
- Step3 メッセージの決定
- Step4 インヘリタンスの決定

図2 ソフトウェア設計法の手順

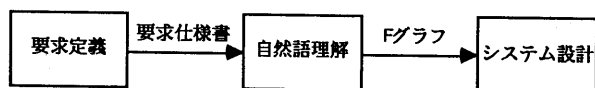


図3 提案する枠組みの説明図

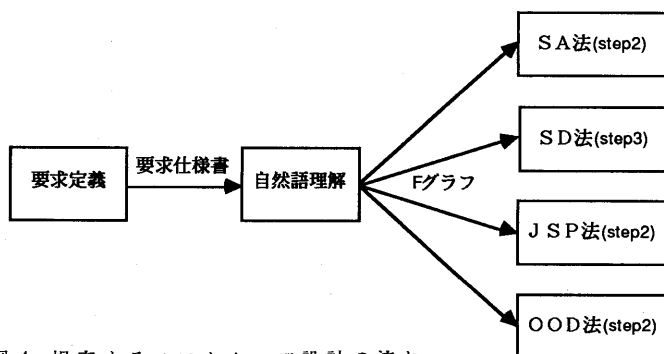


図4 提案するソフトウェア設計の流れ

3. 電報解析問題

自然語^{[2][13]}によるプログラム仕様の例として、ここでは電報解析問題を考える。次の4.と5.での説明はこの例を用いて行う。文献[13]に記述されているプログラム仕様を図5に示す。

4. 文章理解モデル

4.1 意味表現の枠組み

プログラム仕様（自然語）の文章の理解についての基本的な考えを図6に示す。仕様書を構成する記述内容は“操作”に関する記述と“対象”に関する記述から構成されている。ここでは“対象”に関する記述(具体的には、名詞、動名詞を指す)に注目し、その記述内容を表現するためのモデルを導入する。

モデルは“ある対象が他の対象を抽象化したものであるという関係(kind-ofとpart-of)”に基づいているため、抽象化モデルと呼ぶ。更に、抽象化モデル同士を統合するための機能化モデルも導入する。図1の点線部分で示すように、2つの記述の照合を行うことにより、仕様書の理解の妥当性の検討も可能となる。

4.2 抽象化モデル

対象記述の単文を表2に示す6種類の型に分類する。次に、これら6種類の単文の意味を表現する抽象化モデルとして2つのグラフKとPを導入する。Kグラフはkind-of文, attribute文, equal文, value文に基づいて構成されるラベル付き木である(図7参照)。Pグラフは、図8に示すように、part-of文, element文, equal文に基づいて構成されるラベル付き木である。

電報の流れを処理するプログラムを求めたい。この流れはある装置上の文字、数字、空白記号の記号の系列として表され、予め定められた大きさの区分でバッファ領域に転送できる。電報内の各語は空白記号の系列で区切られ、各電報は'ZZZZ'という語で区切られる。電報の流れは空電報(語が存在しない電報)が現れると終了する。各電報は課金される語数と長すぎる語の有無が調べられる。'ZZZZ'及び'STOP'の語は課金されず、13字以上の語は長すぎるとみなされる。これらの処理の結果は各電報に課金される語数と長すぎる語の発生を示すメッセージが添えられて、みやすい形のリスティングとして表示される。

図5 電報解析問題

表2 対象記述の分類

番号	型名	形式	備考
1	attribute文	Aは性質Bをもつ	属性値 種類 (親子関係) 部分全体関係 実現値
2	value文	性質Aは値Bをとる	
3	kind-of文	B, CはAの一つの種類である	
4	part-of文	AはB, Cからなる	
5	element文	AはBを要素にもつ	
6	equal文	AとBは同じである	

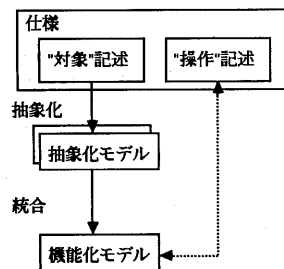


図6 仕様の文章理解

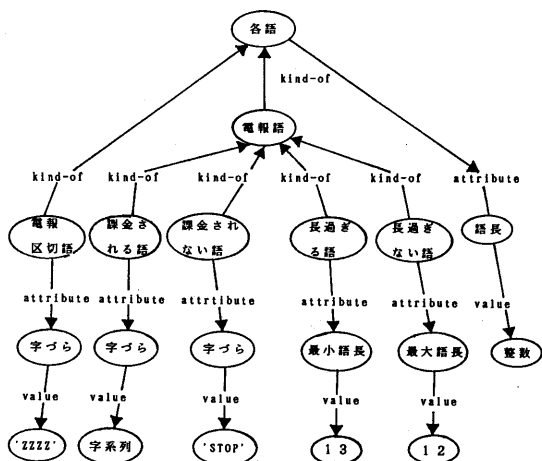


図 7 K グラフ (抽象化モデル)

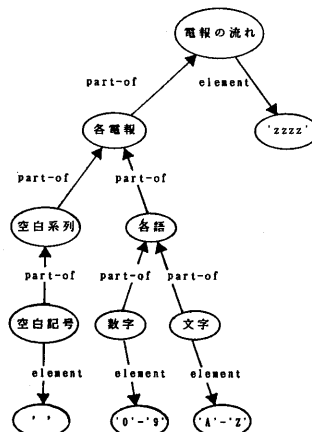


図 8 P グラフ (抽象化モデル)

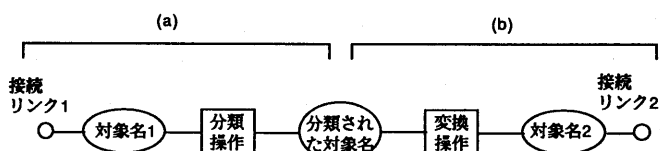


図 9 結合図

リンク 2	⑧	①	②	③		⑥	⑦	④	⑤			
対象名 2	各語	各語	電報語	各電報	処理の 結果	—	バッファ上 の記号系列	記号	各電報	各電報		
変換 操作	(a)	(f)	(b)	(c)	(d)	(e)	(j)	(k)	(l)	(m)		
クラス 名	文字・数字	空白 記号	電報語	電報 区切語	空電報		空系列	有記号 系列	長過ぎ ない語	長過ぎ る語	課金さ れめ語	課金さ れる語
分類 操作		(g)		(h)		(i)		(n)	(o)		(p)	
対象名 1	記号	各語	各語	各電報	記号 系列	バッファ上 の記号系列		電報語		電報語		
リンク 1	⑦	⑧	①	⑤	④	③	⑥	②		②		

図 10 F グラフ (機能化モデル)

4.3 機能化モデル

仕様に含まれる機能に関する意味を表現するため、図9に示す結合図を導入する。図9中の(a)はkind-of文に対応しており、入力された対象を(属性値に基づいて)幾つかのクラスに分類することを表す。一方、(b)はpart-of文に対応しており、入力された幾つかの対象に変換を行って1つの出力対象を求めることを表す。

接続リンク、及び、同じ対象名を利用して結合図同士を接続することにより意味表現を行う。その接続を表現する機能化モデルをFグラフと呼ぶ。図5の電報解析問題の仕様書に対するFグラフを図10に示す。

5. ソフトウェア設計法への適用

ここでは文章理解モデルを具体的な4つのソフトウェア設計法に適用する(図4参照)。

5.1 構造化分析法(SA法)

SA法の各ステップに対する変換について述べる。

Step1(データフロー図の作成)…Fグラフのリンク1からリンク2へ至る道上の変換操作をバブル(節点)とする。次に、その道上の対象名1をバブルの入力枝のラベル、対象名2をバブルの出力枝のラベルとする。同じラベルをもつ有向枝同士を重ね合わせる。

Step2(データ辞書の作成)…データフロー図の有向枝のラベルとして現れている対象名がnで、かつ、Kグラフ中に名前がnの節点が存在し、しかも、attributeの出力枝が接続しているとき、nに対するデータ辞書を作成する。辞書の中味はKグラフ中のnからの出力枝(attribute)に接続する節点名とする。

作成されたデータフロー図、データ辞書を図11、図12にそれぞれ示す。

5.2 構造化設計法(SD法)

SD法に対してはStep1、2を飛ばして、Step3を直接に実行できる。

Step3(階層構造図の作成)…Fグラフに基づいて補助グラフを構成し、それを階層構造図に変換する。補助グラフの節点はFグラフ中の操作とする。次に、Fグラフ上で分類操作から変換操作に向かう(長さが2の)道が存在すれば、対応する節点間に有向枝をひく。但し、途中で経由するクラス名が(変換操作に接続する)対象名2の特殊な要素になっているなら、その有向枝にチェック印をつけ、t枝と呼ぶ。t枝でない有向枝の入る節点(変換操作)からリンク2、リンク1を経由して到達可能な節

点(分類操作)に有向枝をひく(図13参照)。

こうして得られた補助グラフ上でその枝の右と左で節点数がほぼ等しくなる枝を削除し、仮想的な節点を導入する。引続き、仮想的な節点を根とする木へと変換して行く(図14参照)。但し、補助グラフ上でt枝が出ていた節点(図14上で・印のついた節点)と新しく導入した仮想的な節点においてだけ、枝の分岐を許している。

5.3 ジャクソン構造化プログラミング(JSP法)

作成されるデータ構造の全体を図15に、プログラム構造を図16に示す。

Step1(入出力データ構造の決定)…Fグラフ上の対象名1に位置し、リンク1との接続のない節点から開始して、Fグラフを下から上へ、リンク2からリンク1へと進む時に会える対象名、及び、クラス名をデータ構造図の節点とする。但し、part-of関係にある節点は縦方向に、kind-of関係にある節点は横方向に一直線に並べ、その間を枝で結ぶ。更に、Fグラフ上で節点に接続する操作名もデータ構造中に書き入れる。

次に、求めたデータ構造に基づいて、プログラム構造を作成する。その時、例えばレポート作成、初期設定、リセット等の予約語、並びに、クリーネ閉包(*)の使用が許されている。

5.4 オブジェクト指向設計法(OOD法)

Fグラフを求めた後に作成する比較照合表(文献[10]の表5)を利用すれば、Step1、2で決定すべきオブジェクトに関する種々の情報が簡単に求まる。

最終的に求まるオブジェクトの一覧表を表3に示す。

6. むすび

本報告では、文献[10]で提案した文章理解モデルをソフトウェア設計法に適用する試みについて述べた。これにより、ソフトウェア設計法を開発現場で適用する際の困難さの一部が解消されるものと期待される。

参考文献

- [1] T. DeMarco: "Structured Analysis and System Specification", Prentice-Hall(1987)(高梨, 黒田 監訳: "構造化分析とシステム仕様", 日経マグローウヒル社(1986)).
- [2] P. Henderson and R. A. Snowdon: "An experiment in structured programming", BIT, Vol. 12, No. 1, pp. 38-53(1972).
- [3] 馮, 大野, 井上, 菊野, 鳥居: "属性文法による構

造化分析法の形式的記述”，電子情報通信学会コンピュータセッション研究会資料，COMP88-3，pp. 21-30(1988).

- [4] 井上，野村，稲田，菊野，鳥居：“階層的プロセスモデルの提案とそのJSDへの適用”，ソフトウェア・シンポジウム'88論文集，pp. 315-324(1988).
- [5] M. A. Jackson：“Principle of Program design”，Academic Press(1975)(鳥居 宏次 翻訳：“構造化プログラム設計の原理”，日本コンピュータ協会(1980)).
- [6] 嵩，谷口，杉山：“代数的言語の設計と処理系”，榎本編：“ソフトウェア工学ハンドブック”，オーム社(1986).
- [7] 久保未沙：“複合設計”，情報処理，Vol. 25，No. 9，pp. 935-945(1984).
- [8] 宮本 勲：“ソフトウェア・エンジニアリング：現

状と展望”，TBS出版会(1982).

- [9] 大場 充：“ソフトウェアの開発技術”，オーム社(1988).
- [10] 大野，菊野，鳥居：“自然語によるプログラム仕様に対する理解モデルの提案”，情報処理学会知識工学と人工知能研究会(発表予定).
- [11] R. S. Pressman：“Software Engineering：A Practitioner's Approach”，McGraw-Hill(1987).
- [12] 田中，辻井：“自然言語理解”，オーム社(1988).
- [13] 鳥居，杉藤，真野，二木：“プログラム作成技術の現状に関する調査報告[1]”，電子技術総合研究所調査報告(1975).
- [14] 辻井，上原：“ソフトウェア工学と自然言語処理”，情報処理，Vol. 28，No. 7，pp. 913-921(1987).
- [15] 米澤 明憲：“オブジェクト指向計算の現状と展望”，情報処理，Vol. 29，No. 4，pp. 290-294(1988).

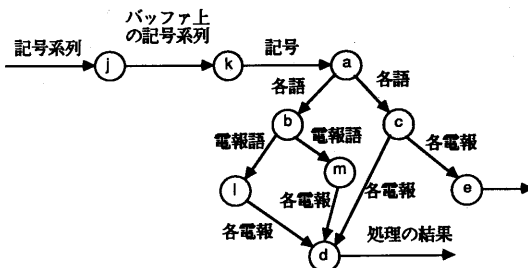


図11 データフロー図

各電報 = テキスト + 長過ぎる語の有無 + 課金する語の数
 処理の結果 = 長過ぎる語の有無 + 課金する語の数 + 電報

図12 データ辞書

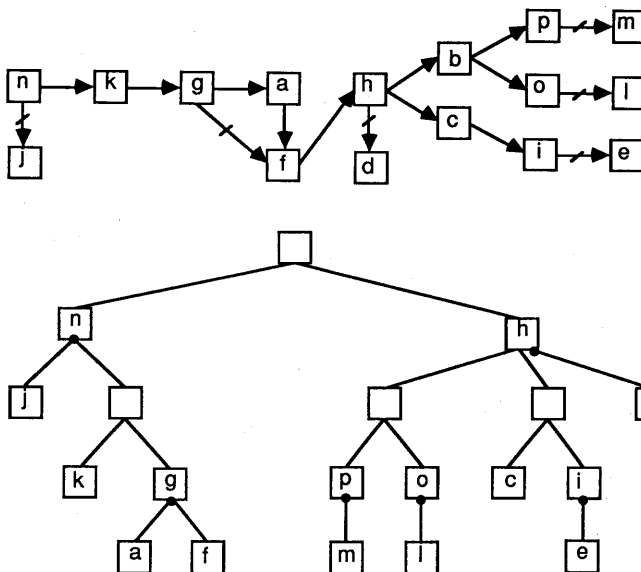


図13 S D 法適用の説明図

図14 階層構造図

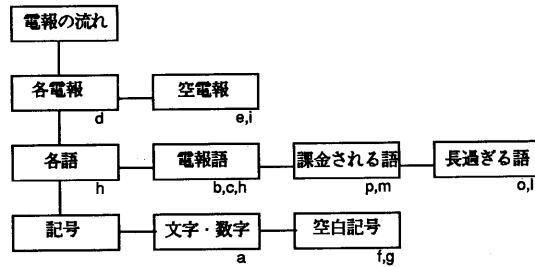


図15 データ構造

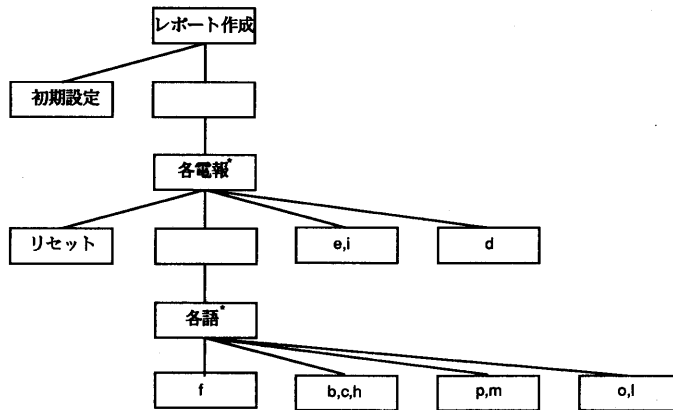


図16 プログラム構造

表3 オブジェクトの一覧表

オブジェクト	属性	動作	備考
語	—	組み立てる	a
	—	取り出す	f
	—	見分ける(電報語)	h
電報語	—	記録する	b
	—	見分ける(課金される語)	p
	長さ	見分ける(長過ぎる語)	o
電報	—	組み立てる	c
	長過ぎる語の有無	記録する	l
	課金語数	記録する	m
	文の長さ	見分ける(空電報)	i
処理の結果	課金語数	表示する	d
	長過ぎる語の有無	表示する	d
	電報	表示する	d
電報の流れ	—	終了する	e
	—	バッファ領域に転送する	j
記号	—	取り出す	k
	記号の種類	見分ける(空白, 文字・数字)	g
記号系列	長さ	見分ける	n