ソフトウェア工学 73－4
（1990．7．19）

オブジェクト指向ソフトウェア設計法の一手法

機能的視点に基づくオブジェクト展開とその枠組み

岸 俊行
日本電気航空宇宙システム（株）

大規模リアルタイムシステム向けのソフトウェア設計法を提案する。この手法はオブジェクト指向設計法を拡張したもので、トップダウン、かつ再帰的にシステムの詳細化ができる、弾力性に優れた設計ができる、多視点からの設計記述ができるという特徴を持っている。また、オブジェクト間のＤＦＤを導入することにより、オブジェクトの属性とスコープの決定に明確な指針を与えた。本論文では、この設計法について例題を用いて説明した後、その基盤となる枠組みについて述べ、本設計法におけるオブジェクト指向の概念の有効性、及び他のオブジェクト指向設計法との比較について考察する。更に、開発のライフサイクルと支援ツールについても論ずる。

Object Oriented Design Methodology

Object Decomposition Approach Depends on Multiple Functional Views

Toshiyuki Kishi

NEC Aerospace Systems

NEC Nakagawara Technical Center, 5-22-5, Sumiyoshi-Chou,
Fuchuu-Shi, Tokyo, Japan 183

We propose software design methodology for large scale of realtime systems, which extends object oriented approach. This methodlogy enables designers to do top-down and recursive system refinement, to design flexible systems, and to describe a system from multiple views. Also it provides them with a guide to identify attributes and the visibility of each object by introducing data flow diagram among objects. In this paper, we illustrate our methodology using a design case study and consider the base of this methodology, effect of object orientation in our approach, and comparison with other object oriented methodology. We also discuss about software life-cycle and supporting tools.

1. はじめに

近年、ハードウェア能力の向上により、コンピュータを利用したシステム開発がますます大規模化している。他方、ソフトウェアの生産性・品質の向上のためには要求分析・設計工程が重要であるということがわかってきた。

大規模システムに対する設計の方法論の要件としては、次の項目が挙げられる。

①トップダウン、かつ再帰的なシステムの詳細化が可能であること
②詳細化の手順に明確な指針があること
③設計が弾力性に優れていること
④グラフィカルな表記法があること
⑤多視点による記述ができること
⑥計算機による支援ツールがあること

多視点による記述ができるとは、設計者に対して対象システムのある一つの側面のみに着目した記述を許し、かつ、それらの記述の集まりをシステム全体の矛盾の無い記述にできることである。この要件は、次の二つの理由により大規模システムの設計に対して有効である。一つは、設計者である人間が複数の側面から同時に物事を捕らえていくことを得意としないということであり、もう一つは、大規模システムではデータや制御の流れが複雑で、どのような表記法でもすぐに記述が見にくくなったり、スペースの問題で書きにくくなったりするということである。

1970年代に提唱された構造化設計法[5]は、①②④⑥は満たしていたが、③の設計の弾力性に欠けていた。1980年代の半ばにBoochによって提案されたオブジェクト指向設計法[1]は、構造化設計法の弱点であった設計の弾力性を抽象化と情報隠蔽によって改善したが、詳細化の手順について明確な指針を与えておらず大規模システムへの適用は困難であった。その後、Boochの手法の拡張として機能に着目してオブジェクトを展開していく設計法がJaloteによって提案された[2]。この手法は、システムの詳細化の手順に明確な指針を与えたが、グラフィカルな表記法は提供されず、また多視点という考えも明示的には扱われなかった。

本論文では、Jaloteの手法を更に拡張した大規模リアルタイムシステム向けのソフトウェア設計法を提案する。この設計法は、トップダウンに、かつ再帰的にシステムを詳細化できる、設計が弾力性に優れている、並びに多視点からのグラフィカルな設計記述が可能であるという特徴を持っている。設計の手順について２章で述べ、３章では例題を用いて設計の具体例と表記法について説明する。そして、４章では設計の基盤となる枠組みについて述べ、本設計法におけるオブジェクト指向の概念の有効性、及び他のオブジェクト指向設計法との比較について考察する。更に、この設計法に基づく開発のライフサイクルと支援ツールについても論ずる。

2. 設計の手順

本設計法の手順は次のようになる。

①システムのコンテキストダイアグラムを記述する

②機能全体を表現するプロセスを一つのオブジェクトとする

③オブジェクトの各機能ごとに
　③－①インフォーマルな仕様を記述する
　③－②機能に関連するオブジェクトを明らかにする
　③－③機能とオブジェクト間のＤＦＤ（データフローダイアグラム）を記述する
　③－④上で出現した各オブジェクトの機能（外部仕様）を明らかにする
　③－⑤機能（いま対象としている方）を分割する必要があるか？
　　THEN　分割した各機能について
　　③－①～⑤を繰り返す

③－②で現われるオブジェクトは、コンテキストダイアグラムのターミネータ、機能へ

の入力データ、並びにインフォーマルな仕様から抽出したオブジェクトのどれかである。また、機能を分割するかどうかの基準は構造化分析におけるプロセスの分割の基準と同様である。

④③の過程で出現した各オブジェクトごとに
④－①オブジェクトを分割する必要があるか？
THEN ③を行う
そのオブジェクトが既存のオブジェクトライブラリーを用いて実現できるならばそれ以上の分割は行わない。

⑤②～④で作成したＤＦＤをＭＦＤ（メッセージフローダイアグラム）に変換する
ここでは、データフローを参考にしてメッセージ名とパラメータを決める。データフローとメッセージフローは必ずしも一対一に対応するわけではなく、また、データの流れとメッセージの方向も必ずしも一致しない。

⑥全てのＭＦＤを併合し、各オブジェクトのメッセージインターフェースを明らかにする
ＭＦＤを併合する時のヒューリスティックは、次のようになる。
a. 同じ名前のオブジェクトは同じものとみなす
b. 宛先オブジェクト、メッセージ名、パラメータデータが全て一致したメッセージフローは同じものとみなす

⑦全てのオブジェクトについて、保持すべき状態、並びにそのオブジェクトからの出力データを基に属性を明らかにする
状態属性は、オブジェクトの持つ機能に対するインフォーマルな仕様から抽出する。

⑧オブジェクト間にインヘリタンス階層を設定する
インヘリタンス階層の設定は次の二つの場合がある。
a. 同様の性質（機能・構造）を持つオブジェクトを明らかにし、それらに共通の性質を抽出した上位オブジェクトを設定する
b. 既存のオブジェクトライブラリーに利用できるものがあればその下位オブジェクトとする

⑨必要ならばＭＦＤに現われた機能もオブジェクトとして、属性とインヘリタンス階層を設定する
後述する枠組みでは機能はプロセスであり、オブジェクトではないが、インプリメントの際に機能をオブジェクトとして扱うことにより、次のような利点がある。
a. 機能を局所化することにより、機能に対する変更が容易になる〔3〕
b. インヘリタンス階層を設定することにより、差分プログラミングが実現できる
c. 機能（プロセス）がもつ局所データを属性として自然に表現できる

3. 設計の例

ここで取り上げる例題は、装置を監視するシステムで次のような機能を持つ。

a. コマンド診断機能
オペレータからの指示により装置の状態を表示する。不正なパラメータのコマンドが入力された場合にはエラーメッセージを表示する。
b. アラーム通知機能
装置でアラームが発生した場合には、発生時刻とともにアラームメッセージを表示する。
c. 定期診断機能
一定時間ごとに装置の状態を表示する。ただし、コマンド診断を行っている時には定期診断は行わない。

以下では、２章で述べた手順に従ってこの例題の設計を行う。

①コンテキストダイアグラムを記述する
ターミネータは、装置監視センサ、時計、入力装置、出力装置の四つになる。（図１）

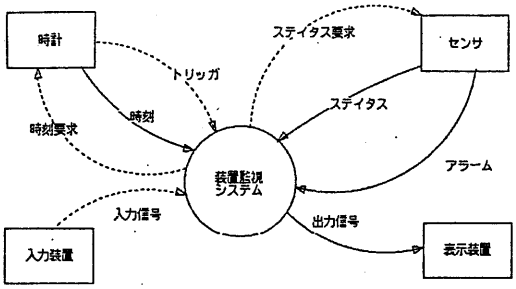

図1　コンテキストダイアグラム

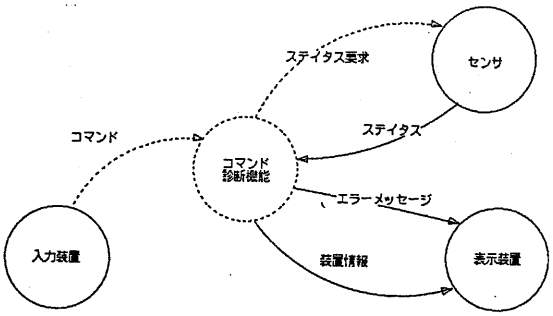

図2　コマンド診断機能の実現ＤＦＤ

②機能全体を表現するプロセスを一つのオブジェクトとする
　装置監視システムをオブジェクトとする。

③オブジェクトの各機能ごとに
③－①インフォーマルな仕様を記述する
　装置監視システムオブジェクトのコマンド診断機能に対するインフォーマルな仕様は次のようになる。

（コマンド診断機能）
　ユーザからコマンドが入力されたらコマンドのパラメータをチェックし、有効なコマンドならば装置センサにステイタスを要求し、パラメータにエラーがあればエラーメッセージを表示する。また、センサからステイタスを受信したらステイタスをチェックし、正常ステイタスならば装置情報を表示し、異常ステイタスならばエラーメッセージを表示する。

③－②機能に関連するオブジェクトを明らかにする
　関連するオブジェクトは、センサ、入力装置、出力装置である。

③－③機能とオブジェクト間のＤＦＤを記述する
　ＤＦＤを図２に示す。ここで実線の円がオブジェクトを表わし、破線の円が機能を表わす。実線の矢印、破線の矢印は通常のＤＦＤと同様にそれぞれデータフロー、コントロールデータフローを表わす。

③－④上で出現した各オブジェクトの機能（外部仕様）を明らかにする
（センサオブジェクトの機能）
　ステイタス要求を受け取ったら装置の状態を読みだし、装置監視システムに送信する。
（表示装置オブジェクトの機能）
　a.エラーメッセージを表示する
　b.装置情報を表示する
（入力装置オブジェクトの機能）
　ユーザからの入力を感知したら入力に対応するコマンドを装置監視システムに送信する。

③－⑤機能（いま対象としている方）を分割する必要があるか？
　THEN　分割した各機能について
　　③－①～⑤を繰り返す
　コマンド診断機能はステイタス要求機能とステイタス表示機能の二つに分割され、各々に対して③－①～⑤を繰り返す。
　これらの機能に対するインフォーマルな仕様は次のようになる。

（ステイタス要求機能）
　コマンドが入力されたらそのコマンドからパラメータの有効／無効を取り出し、有効ならば装置センサにコマンドを送信し、無効ならばコマンドエラーコードに対応するエラーメッセージをコマンドエラーメッセージテーブルから取り出して表示する。
（ステイタス表示機能）
　装置センサからステイタスを受信したらそのステイタスから正常／異常データを取り出し、正常ならば装置情報を取り出して表示す

る。異常の場合はステイタスから取り出したステイタスエラーコードに対応するエラーメッセージをステイタスエラーメッセージテーブルから取り出して表示する。

これらの二つの機能に対するＤＦＤを図３と図４に示す。この時点で機能は十分に分割されたので、装置監視システムのコマンド診断機能に関する詳細化は終了する。同様にアラーム通知機能、定期診断機能についても詳細化を行う。

④③の過程で出現した各オブジェクトごとに
④－①オブジェクトを分割する必要があるか？
　　THEN　③を行う
③の過程で出現したオブジェクトは、センサ、時計、入力装置、出力装置、コマンド、ステイタス、コマンド（ステイタス）エラーテーブルなどである。入力装置・出力装置などは更に分割する必要があるので③の操作を繰り返す。エラーテーブルは既存のテーブルオブジェクト（与えられたキーに対応する値を返す）を用いて実現できるので、分割は行わない。

⑤②～④で作成したＤＦＤをＭＦＤ（メッセージフローダイアグラム）に変換する
ステイタス要求機能のＤＦＤ（図３）、ステイタス表示機能（図４）に対応するＭＦＤをそれぞれ図５、図６に示す。ＭＦＤでも同様に実線の円がオブジェクト、破線の円が機能を表わす。また、実線の矢印はメッセージフローを表わし、メッセージフローにつけられた名前はメッセージ名であり、括弧内はパラメータである。

⑥全てのＭＦＤを併合し、各オブジェクトのメッセージインターフェースを明らかにする
装置監視システムのメッセージインターフェースは次のようになる（一部分）。メッセージ名の後ろの括弧内はパラメータである。

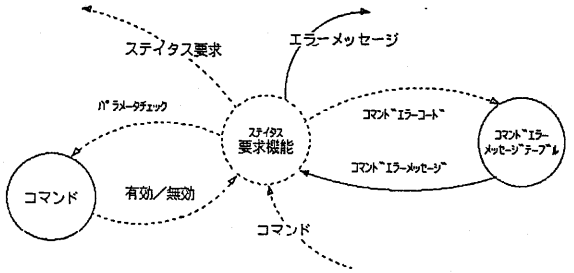

図3　ステイタス要求機能の実現ＤＦＤ

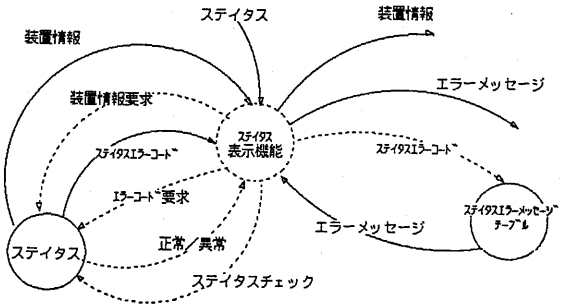

図4　ステイタス表示機能の実現ＤＦＤ

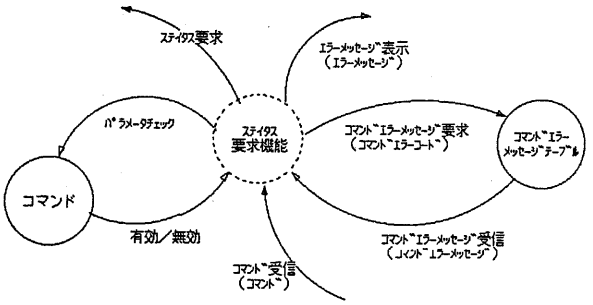

図5　ステイタス要求機能の実現ＭＦＤ

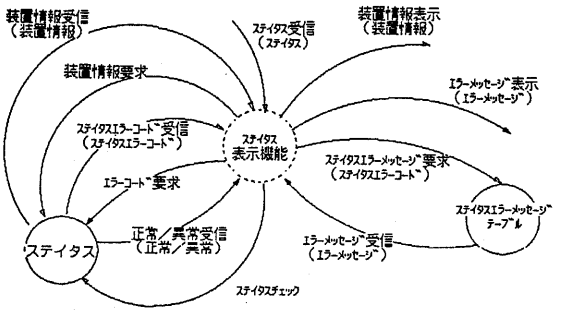

図6　ステイタス表示機能の実現ＭＦＤ

＜装置監視システム＞
仕様部インターフェイス：
　なし
実現部インターフェイス：
　（入力）
　コマンド受信（コマンド）
　ステイタス表示（ステイタス）
　装置情報受信（装置情報）
　　・
　　・
　（出力）
　ステイタス要求（）
　装置情報表示（装置情報）
　　・
　　・

⑦全てのオブジェクトについて、保持すべき状態、並びにそのオブジェクトからの出力データを基に属性を明らかにする

仕様の制約条件により、コマンド診断機能を実行中は定期診断機能は実行されない。そこでこの二つのプロセスを制御するための制御情報を両方のプロセスを管理している装置監視システムオブジェクトの属性として定義する。

また、コマンドから出力されるパラメータの有効／無効情報、ステイタスから出力される装置情報、ステイタスエラーコードなどはオブジェクトが保持すべき情報であるのでそれぞれコマンド、ステイタスの属性とする。一方、センサから出力されるステイタスは要求時に装置から検出するもので属性とはしない。（ただし、この場合ステイタスも属性としてキャッシュしておくこともできるわけで、属性にするかどうかの判断は、実際には、オブジェクトのインプリメントの方法によっている）

⑧オブジェクト間にインヘリタンス階層を設定する
b.既存のオブジェクトライブラリーに利用できるものがあればその下位オブジェクトとする

エラーテーブルは既存のテーブルオブジェクトと同様の性質を持つので、その下位オブジェクトとする。

⑨必要ならばＭＦＤに現われた機能もオブジェクトとして、属性とインヘリタンス階層を設定する

コマンド診断機能と定期診断機能はトリッガが異なるだけであとの処理は同じなので、上位の診断機能オブジェクトを定義し共通の性質はそこに記述する。

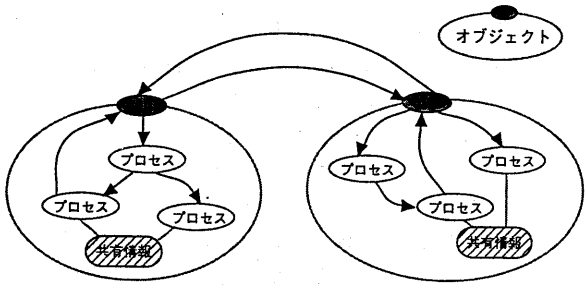

図7　設計の枠組み

4.　考察

4.1　オブジェクト指向の有効性

本設計の基盤となる枠組みを図７に示す。この枠組みでは、オブジェクトは内部に並列に動作する複数のプロセスとそれらのプロセスで共有するデータをもつ。処理は、各々のオブジェクトの管理するプロセスがオブジェクトの管理のもとに交互に通信する事によって進み、個々のオブジェクトもまた並列にメッセージを処理することができる。オブジェクトに送られたメッセージは宛先のプロセスＩＤがあればそのプロセスに送られ、そうでなければオブジェクトが判断して適当なプロセスに送る。前者は呼び出されたプロセスが呼び出したプロセスに値を返したりする場合に相当し、後者はメッセージ名に対応するプロセスをオブジェクトが呼び出す通常のメッセージセレクタ機能に対応している。そしてこの枠組みにおけるオブジェクトが設計の過

程で現われるオブジェクトに、プロセスが機能に、プロセス間の通信がメッセージフローに対応している。なお、設計のインプリメントは使用する言語や設計条件に依存しており、プロセスを関数に割り当てメッセージフローを関数呼び出しとして実現することもできるし、プロセスをタスクに割り当てメッセージフローをタスク間通信として実現することもできる。

次にオブジェクト指向の特徴が本設計法においてどのような効果をもたらしているかをそれぞれの特徴ごとに考察する。

①現実世界のモデル化が容易である。

従来の機能指向的なアプローチでは、設計とは機能を分割してそれぞれをプロセスに割り当てていくことであった。この場合の機能分割はデータの存在を意識せずに行われるので、結果としてプロセスとデータの関係は全く無秩序になってしまう。一方、本設計法では、結果としてシステムを機能分割してプロセスに割り当てていくことになるが、オブジェクトとその機能を明らかにしながら詳細化を進めていくという指針に従うことにより、データとの対応のうまくとれた機能分割になっている。このように分割されたシステムは実世界の変更（仕様変更）に対して非常に柔軟な弾力性のあるシステムになっている。

②カプセル化

ある特定のプロセス間だけでデータを共有するモデルを与えてくれる。

③インヘリタンス

この特徴は主にインプリメントにおいて有効になる。すなわち、差分プログラミングにより記述量を少なくさせることができ、また、既存のライブラリー部品利用のための枠組みを与えてくれる。

4.2 従来のオブジェクト設計法との比較

4.2.1 Booch、Jaloteの手法との比較

Boochの手法における設計の手順は次のようになっている。

①オブジェクトとその属性を明らかにする
②オブジェクトが受ける操作と要求する操作を明らかにする
③オブジェクト間のスコープを決める
④オブジェクトのインターフェイスを決める
⑤オブジェクトの操作を実現する

Boochの手法では、オブジェクトの属性やスコープを決めるための指針は与えられていないが、本設計法では、オブジェクト間のＤＦＤを導入することにより、そのための明確な指針を与えている。また、本設計法では、オブジェクトの属性の決定を以下の理由によりオブジェクトの実現の時まで遅らせている。

a. 抽象化とは、必要な属性だけを抽出する作業であり、オブジェクトに必要な属性はそのオブジェクトの機能が決まった後でないと決まらない。
b. ３章でも述べたようにオブジェクトが出力するデータを属性としてもつか、あるいは、計算によって求めるかはオブジェクトのインプリメントの方法に依存している

Jaloteの手法では機能を明示的に扱うことによりオブジェクトを展開していき、分割された機能がどこかのオブジェクトに属するまで展開を繰り返す。これに対して本設計法ではシステム全体を表わすオブジェクトという概念を導入したことにより、どのオブジェクトにも属さないような機能も前述の枠組みを壊すことなく扱うことができる。この仕組みはオブジェクト間の制約条件を扱う場合にも有効である。

また、Boochの手法でもJaloteの手法でもAdaをターゲット言語としているため、インヘリタンスの利用のための手順は含まれていない。

4.2.2 Wardの手法[3]との比較

Wardの手法の基本的な手順は、ＤＦＤに現われたプロセスとデータストアを抽象化によってまとめていくことによりボトムアップにオブジェクトを識別していくやり方である。この手法では最初にシステムを機能的に分解してからオブジェクトとして積み上げていくわけで、結果的に本設計法と同じ機能分割になったとしても、実世界を自然にモデル化できるというオブジェクト指向の特徴を最大限に生かしているとはいえない。ただし、ここではオブジェクトの識別の方が機能の識別よりも容易であるという前提に立っている。

4.3 開発のライフサイクル

プロトタイピングはユーザの要求を早期に引き出して仕様変更を少なくするという目的の他に、早いうちに設計の正しさを確認し後戻りを防ぐという目的でも行われる。オブジェクト指向開発では、モジュール（オブジェクト）にどのように機能を割り当てるかという作業の他に、どのようなインヘリタンス構造をつくり、どのレベルのオブジェクトにどの機能を割り当てるかという機能的設計法には無かった作業が追加される。そのうえ悪いことに、悪いインヘリタンス階層は、機能的アプローチ以上に設計の弾力性を低下させる。そのため、オブジェクト指向開発においては、プロトタイピングが非常に重要な役割を果たす。本論文で提案した設計法においても設計をSmalltalkなどでインプリメントし、オブジェクトの切り出し方やオブジェクトへの機能の割り当て、及びインヘリタンス階層の良さを早期に確認しフィードバックをかけることが必要である。

また、この設計法は個々の設計記述を矛盾なく合成するために計算機による支援が必須である。支援としては次のようなものが考えられる。

①ＭＦＤの自動併合
②ＤＦＤ・ＭＦＤの検証
③設計記述の入力支援

5. むすび

大規模リアルタイムシステム向けの設計法として機能的視点に基づいたオブジェクト展開によるオブジェクト指向設計法を提案した。オブジェクト指向はソフトウェア開発のライフサイクルを一貫してサポートできるモデルであり、将来のソフトウェア開発において非常に有望なアプローチである。今後は、この設計法を実際に適用しながら更に改良していくとともに、オブジェクト指向を基盤としてソフトウェアのライフサイクル全体を計算機により統合的に支援する環境の構築を検討していく予定である。

<謝辞>

研究の機会を与えて下さった田中社長、山口常務、並びに有益な助言を与えて頂いた香川副主任をはじめソフトウェア生産性推進部の皆様に感謝します。

<参考文献>

[1] G.Booch, "Object-Oriented development", IEEE Trans. Software Eng., vol.SE-12,no.2, pp.211-222, Feb. 1986
[2] P.Jalote, "Functional Refinement and Nested Objects for Object-Oriented Design", IEEE Trans. Software Eng., vol.SE-15, no.3, pp.264-270, Mar. 1989
[3] I.Jacobson, "Object Oriented Development in an Industrial Environment", OOPSLA '87 Proc, pp.183-191, 1987
[4] P.T.Ward, "How to Integrate Object Orientation with Structured Analysys and Design", IEEE Software, 1989
[5] Edward Yourdon, 「構造化手法によるソフトウェア開発」,日経マグロウヒル,1987
[6] 宮下,田口,岸,小林, 「要求分析における知識ベースの適用に関する調査研究」, 情報処理振興事業協会, 1988