ソフトウェア工学 84－6
（１９９２．３．３）

制御ソフトウェアのソフトウェア・アーキテクチャ・モデル

(株) 東芝 システム・ソフトウェア技術研究所

清水洋子　　猪野 仁

我々は，制御ソフトウェアを想定し，仕様の多様化に応じた部分的なソフトウェアの変更を容易にする開発環境の研究を進めている．

その第一段階として，制御ソフトウェアの特徴を分析し，これと親和性があり，かつ変更容易な構造を狙いとしたソフトウェア・アーキテクチャ・モデルを構築した．これは，大きな処理の流れを状態遷移モデルで規定し，これをベースに処理のある局面での動作仕様をif-thenの形式で与えるものであり，状態遷移図およびif-thenルールによる形式的な仕様記述を伴なう．

本論文ではこのモデルについて説明し，これをエレベータ制御ソフトウェアに適用した例について紹介する．

Software Architecture Model for Control Software

Yoko Shimizu　　Masashi ino

Systems and Software Engineering Laboratory,TOSHIBA Corp.

70 Yanagi-cho Saiwai-ku KAWASAKI,210,Japan

Because of specifications' variousness, the environment in which software is modified easily is expected. We are studying it especially about control software.

This paper introduce the software architecture model that is developed for control software, aimed at matching with the characteristic of it and at being easy to modify.

In this model, the main control flow is provided on state transition model, and the action specifications is based on it by if-then rules.

The explanation of this model and the example applied to the control software for elevator are shown.

1 はじめに

制御システムにおいては，ソフトウェアへの依存度が増加すると共に，ニーズの多様化も進んできている．したがって多様なニーズをソフトウェアで吸収する必要があり，標準的なソフトウェアを部分的に変更してこれに対応したいという要求は高い．しかし部分的な変更を行うにも，ソフトウェアの広い範囲を理解し，更に変更後の周囲への影響を追跡しなければならない等，かなり困難な作業になるというのが現状である．

この問題を解決するために，我々は以下の２点が必要であると考える．

(1) 明確なソフトウェア・アーキテクチャ・モデルと形式仕様記述

(2) プログラム自動生成，仕様検証等のコンピュータ支援

(1) は，ソフトウェアをどのような機構で作成するかを明確に規定し，変更容易性の観点からの機能分割のモデルを与え，これに則った機構で仕様を形式的に記述するということである．ソフトウェア全体の機構および機能分割が明確であることは，ソフトウェアの理解に大きな助けとなり，また変更容易性を考慮した機能分割により，変更による影響範囲の追跡作業を軽減することができる．

(2) は，(1) における形式仕様記述を基礎にしており，プログラムへの機械的変換の定式化によりプログラム自動生成が可能となり，また仕様記述段階での検証へと発展できる．

本論文ではこの第一段階として，(1) に挙げたソフトウェア・アーキテクチャ・モデルと形式仕様記述について検討し，これをエレベータ制御ソフトウェアに適用した例について紹介する．

2 制御ソフトウェアの特徴

モデルを検討するにあたり，対象としている制御ソフトウェアの特徴について述べる．

まず，想定している制御ソフトウェアの概要を図１にコンテキスト・ダイアグラムの形で示す．制御ソフトウェアは４種類の外部実体（図中では四角で表示）と関連している．ユーザからの要求，およびスイッチ，センサからのシステム状態という情報を入力とし，これらを基に動作を決定し，制御対象ハードウェアおよびHI系ハードウェアに対して，決定した動作を実行させるための動作指令を出力する．

このような制御ソフトウェアにおける特徴は，大きく２つある．

(1)「処理順序の規定」と「ある局面での断片的動作仕様」に分離できる．

後に適用例として用いるエレベータ制御ソフトウェアを例に考えてみる．

エレベータ制御では，「止まっている状態からブレーキを外して加速を始め，やがて減速してブレーキをかけ，再び停止する」という様に，処理に順序が存在する．一方でエレベータの動作仕様は，ある局面における断片的動作という形で抽出するのが考え易い．例えば「止まっている時だけドアを開けて良い」，あるいは「到着する時に到着ランプを点灯する」といった具合である．つまり，エレベータの大まかな処理の順序を規定しておき，個々の局面において「～の時は～する」といったif-then形式で動作仕様を抽出するという様に，両者を分離することができる．

(2)「動作の核となるハードウェア」と，「HI系ハードウェア」とで性質が異なる．

再びエレベータの例で述べると，動作の核となるのは「かご[1]」と「ドア」であると考えられる．またHI系としては，到着ランプやアナウンス等がある．同じハードウェアである両者の性質の違いとは，前者はこの制御システムに必ず存在するものであり，その動作決定までの過程が複雑であるのに対し，後者はニーズに応じて追加／削除が頻繁にあり，その動作の決定は単純である，という点である．

[1] 人が乗るもの．いわゆるエレベータ．

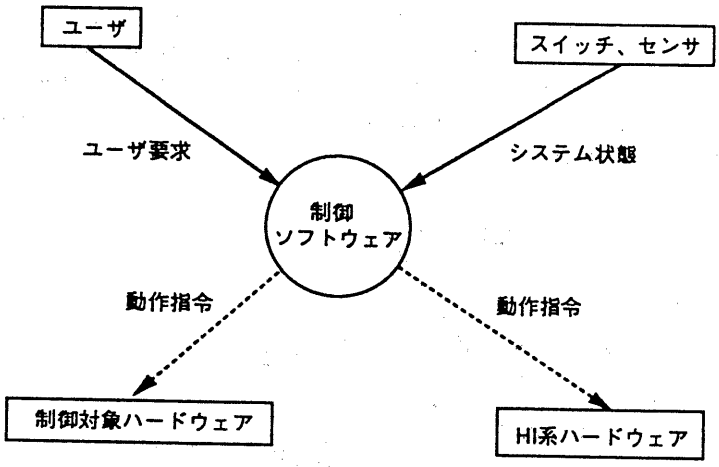

図1　制御ソフトウェアの概要

3　ソフトウェア・アーキテクチャ・モデル

前述の特徴を考慮した制御ソフトウェアのソフトウェア・アーキテクチャ・モデルの全体像を図2に，DFD[2]の記述で示している．

以下では，本モデルの概要を説明し，分割した個々の機能の外部インタフェースおよび形式仕様記述について述べていく．

3.1　概要

図2において丸で描かれているのが機能であり，本モデルは応用動作，基本動作，順序制御の3つの機能に分割されている．個々の機能の概要は以下の通りである．

応用動作

ニーズの多様性によって生じる応用的な動作仕様を扱う．

HI系ハードウェアに対する動作指令はここから直接出力するが，動作の核となるハードウェアに対しては，すぐに動作指令を出さず，動作の要求という形で基本動作に委ねる．

基本動作

ニーズによる変化のない基本的な動作仕様だけを扱う．

応用動作からの入力である要求と基本動作仕様と合わせ，同じハードウェアに対して対立する動作要求があるときには，それらの優先度によって競合の解消を行ない，最終的に動作を決定し，これを命令として順序制御に送る．

順序制御

処理の順序を規定する．

[2]データ・フロー・ダイヤグラム

順序制御は状態遷移モデルをベースに行う．これはある状態でイベントを受けとると，適切なアクションを起こして次の状態に遷移していく，という情報の組合せでシステムの動きを記述するモデルである．

並行に状態が遷移するものがある場合，これらは別々に順序制御をしなければならないので，それぞれに一枚STD[3]を記述する必要がある．ソフトウェア作成の立場からは並行には動かないと思われても，delayのために並行な動きをする場合がある．delayとは，ハードウェアに動作指令を出した後，一定時間の遅延(delay)を持たせてから実際の動作を実行するといったもので，ハードウェアの制約等から生じる．あるハードウェアに動作指令を出力し，実際には一定時間の遅延の最中に，他のハードウェアに動作指令を出力した場合，両者は並行して状態が遷移する結果となるため，それぞれ一枚ずつのSTD記述が必要となる．

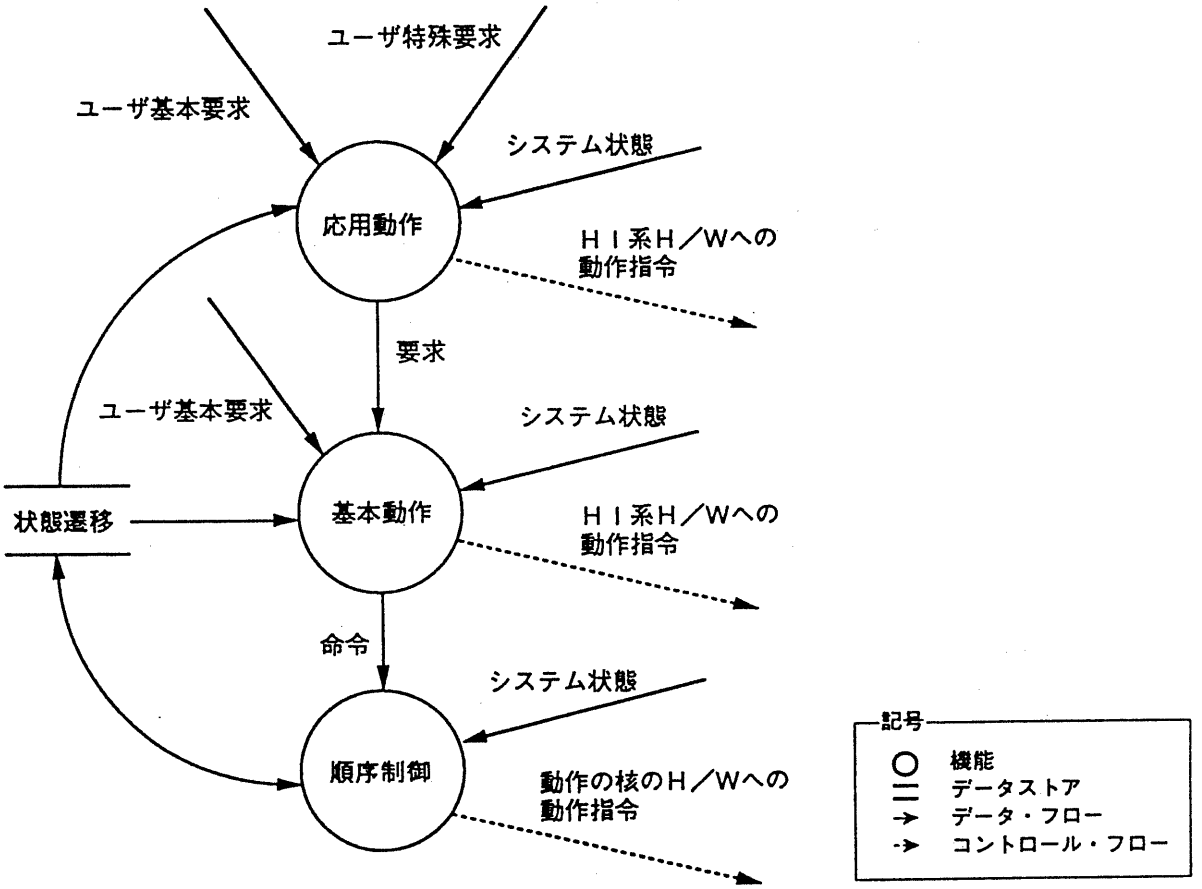

図2　ソフトウェア・アーキテクチャ・モデル

[3]State Transition Diagram(状態遷移図)

3.2 外部インタフェース

各機能の外部インタフェース(入出力)は，図2において，丸で示される各機能への入力データフローが参照可能データであり，出力データフローが書き込み可能データという形で規定されている．これを少し詳細に説明する．

応用動作

応用的な動作仕様を扱っているため，この機能で参照するユーザ要求は，制御システムが基本的に備えている**ユーザ基本要求**，および応用のために特に準備された**ユーザ特殊要求**の両方である．更にスイッチ／センサから入力される**システム状態**，**順序制御**が公開している**状態遷移**を参照することにより，応用的な動作を決定する．

基本動作

ユーザ要求としては**ユーザ基本要求**だけを参照する．その他**応用動作**と同様に**システム状態**および**状態遷移**に加え，**応用動作**からの入力である**要求**も参照する．最終的な動作決定後は，**順序制御**に対して命令を出力する．

順序制御

入力は状態遷移モデルにおけるイベントと対応するが，イベントとしては，**基本動作**から入力される**命令**およびスイッチ／センサから入力される**システム状態**が考えられる．更にもう1つ，複数のSTD間の制約，すなわち他のSTDが特定の状態にあるときだけ次の状態に遷移するといった表現を可能とするために，他のSTDの状態をもイベントとして記述できるように拡張する．

出力は状態遷移モデルにおけるアクションに対応し，これはそのSTDが記述しているハードウェアに対する**動作指令**である．

状態として抽出するものはシステム内部で意味を持つものであるが，具体的には**システム状態**データの入力により遷移する状態，およびハードウェアに**動作指令**を出すことで遷移する状態を考慮すれば良い．

この他に，**基本／応用動作**との重要なインタフェースとして，**状態遷移**がある．**順序制御**は，状態遷移モデルをベースにして現状のシステムの状態を保持しているわけだが，これを**基本／応用動作**へ公開する訳である．図2上ではデータストアとして示されているが，唯一**順序制御**において書き込みが許されており，**基本／応用動作**両方において参照されているデータである．したがって，これが順序制約とif-thenルールをつなぐインタフェースとなる．

実際の内容は2つあり，1つはシステムが今どの状態にあるかという情報，もう1つは，システムがちょうど今その状態に遷移したところであるというタイミングを伝える情報である．これは，**基本／応用動作**内でのif-then形式の仕様において，「ある状態のときに～する」あるいは「ある状態に変わるときに～する」といった，2つの参照形態があるからである．

3.3 形式仕様記述

最後にこれら各機能に対する形式仕様記述について簡単に触れる．

応用動作

この機能についての形式仕様記述は，特徴のところでも述べたようにif-then形式で記述できる．**ユーザ基本要求**，**ユーザ特殊要求**，**システム状態**，**順序制御**の4つの入力データがif部に現れ，**基本動作**に対する**要求**の出力がthen部に記述される．

基本動作

この機能も**応用動作**と同様にif-then形式で記述できる．if部に現れるのが，**ユーザ基本要求**，**システム状態**，**状態遷移**，**要求**の4種類であり，then部には**順序制御**に対する**命令**出力を記述する．

順序制御

この機能は，状態遷移モデルに対する記述法の１つであるSTDによって形式的に記述することができる．入力．出力がそれぞれイベントとアクションに対応する．

4　適用例

以上述べてきたモデルをエレベータ制御ソフトウェアの例題に適用してみる．

順序制御

エレベータの場合，動作の核となるハードウェアは，かごおよびドアである．そしてこれらは並行して状態が遷移する．これは，例えばドアが開いている最中でも，人の乗り降りによりかごの位置が床からずれたときには，かごの位置補正を行うことなどからも分かる．したがって，かご，ドアそれぞれに対して，STDを一枚ずつ記述することになる．

ここではドアについて，順序制御の仕様をSTDで記述した例を図３に示す．

出力(アクション)に描かれているのは戸開閉指令だけあり，これはON(図では「戸開閉指令」と記述)がドアを開ける動作指令，OFF(図では「$\overline{\text{戸開閉指令}}$」と記述)がドアを閉める動作指令に対応している．

基本動作からの入力としては，ドアに対する動作命令である戸開閉命令を準備する．これをONするとドアを開ける命令，OFFするとドアを閉める命令となる．

その他の入力データとしては，システム状態として入ってくる完全戸閉，完全戸開，戸閉直前といったデータ，およびかごのSTD上の状態である．図３では，完全戸閉しているドアはかごが停止している状態でなければ開かないような仕様記述がなされている．

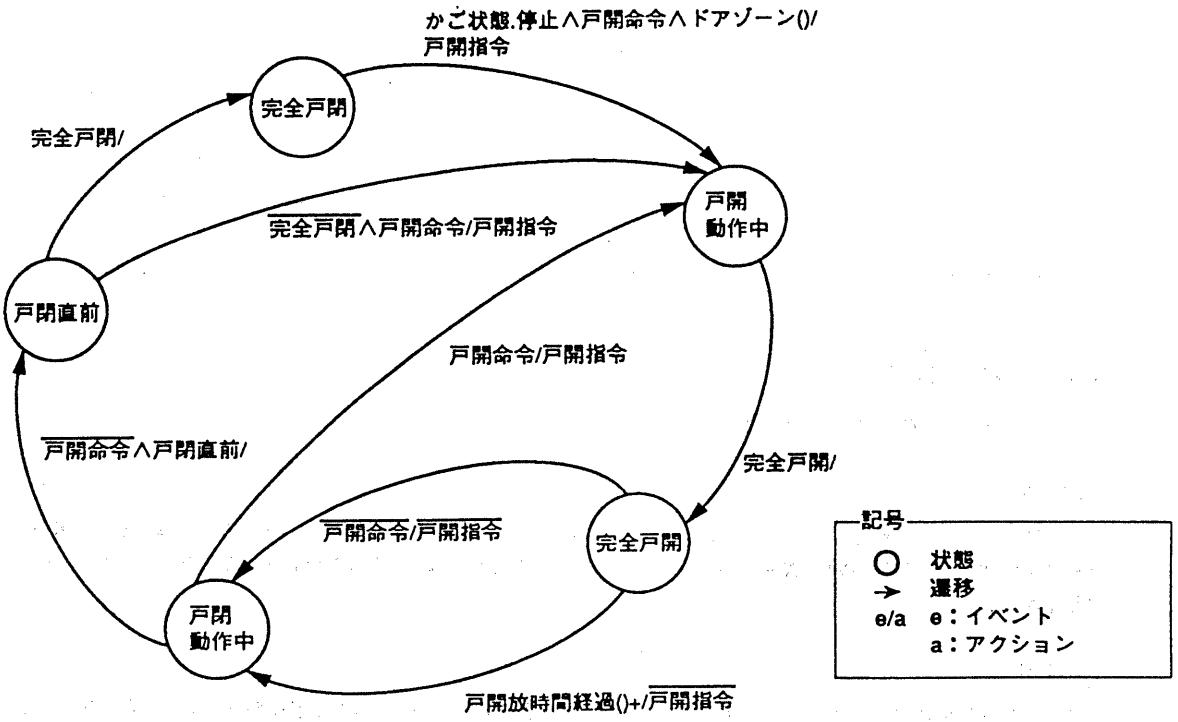

図３　順序制御仕様記述例（エレベータ制御ソフトウェア　ドア）

基本動作

基本動作としては,「戸開ボタンと戸閉ボタンの両方が押されているときは，ドアは開ける」という仕様について記述してみる．すると図4のような簡単のif-then形式で記述することができる．if部にある2つのデータはユーザ基本要求であり，then部にあるデータは命令である．そしてここでは，ユーザ基本要求間の競合を，戸開閉の優先度によって解消し，最終的に戸開の動作を決定している．

戸開ボタン　∧　戸閉ボタン　=>　戸開命令

図4　基本動作仕様記述例（エレベータ制御ソフトウェア　戸開閉）

応用動作

応用動作として，専用運転という動作仕様を取り上げてみる．これは荷物運搬時等に必要となるもので,「かごの中についている専用運転のスイッチが入ると，ホールでの呼び要求には応えずに，かご内の各階のボタンにだけ応じて運転し，またかご内の戸閉ボタンが押されるまではドアは閉まらないようにする」というものである．この仕様を形式的に記述したのが図5である．

専用運転スイッチ　=>　ホール呼び無視　∧　戸開放

図5　応用動作仕様記述例（エレベータ制御ソフトウェア　専用運転）

専用運転のスイッチというのは，この仕様のニーズに応じて付いている場合と付いていない場合がある．つまりこれはユーザ特殊要求にあたる．if部でこれを参照し，then部ではホールの呼びに応えない要求，戸閉ボタンが押されるまでドアを閉めない要求の2つを基本動作に出力している．

基本動作ではこれらの要求を参照した上で，何階に行くか，ドアは閉めるか，といった最終的動作を決定することになる．

5　考察

以上，制御ソフトウェアのソフトウェア・アーキテクチャ・モデルとその適用例について述べてきた．ここではこのモデルについて，先に述べた制御ソフトウェアの特徴との親和性，および我々の目標とする変更容易性という2つの観点から考察する．

5.1　特徴との親和性

特徴の1つめは,「処理順序の規定」と「ある局面での断片的動作仕様」の分離であった．これはモデル上の機能分割において，順序制御と，基本／応用動作とを分割したことに反映されている．基本となる順序の制御は状態遷移モデルに載せ，ここでの状態および状態の遷移タイミングを基本／応用動作に公開することにより，この中における断片的な動作仕様を，順序制御に関連づけて記述することが可能となった．また形式仕様記述も，流れの記述にSTD，断片の記述にif-then形式を採用し，素直な記述ができる．

特徴の2つめは,「動作の核となるハードウェア」と「HI系ハードウェア」の性質が異なる点であり，これもモデルの中に採り入れている．HI系ハードウェアは，基本／応用動作の部分から直接動作指令を出すことができ，ハードウェア自体の追加や削除の変更に対応しやすい．一方動作の核となるハードウェアに対しては，例えば応用動作に新しい動作仕様を追加する場合は基本動作に要求を出すところまでを描き，その後で要求同志が競合した場合の解決や，ハードウェアの制約について考慮する必要がない．

5.2 変更容易性

変更容易性については1節で述べた様に2つの要素がある．

1つは明確なソフトウェア・アーキテクチャ・モデルと形式仕様記述を与えることである．本モデルは3つの機能により構成されており，それぞれに形式仕様記述を定義している．各機能の役割と外部インタフェースが規定されているため，変更を施す部分を特定したり，変更による影響範囲を追跡する作業を行ないやすいと考えられる．

2つめは，形式仕様記述を用いてコンピュータ支援をすることである．本モデルで用いる仕様記述は，プログラムへの機械的な変換が可能なものである．したがって，変更作業を仕様記述のレベルで行い，そこからプログラムを自動生成するという開発環境を構築できる．プログラミング言語より上位レベルの言語で作業可能になるという点で，変更の容易性に寄与できる．

6 おわりに

現在我々は，本モデルを実例に適用する作業を進めており，変更作業の容易性を目標に，モデルの実際レベルでの適用評価を行なっている．更にモデルでの形式仕様記述に基づいて，プログラムの自動生成，仕様検証の研究も進めていく予定である．

参考文献

[1] A.Tsalgatidou and V.K.P.Loucopoulos, "Rule-Based Requirements Specification and Validation", Lecture Notes in Computer Science 436 Advanced Information Systems Engineering,1990

[2] D.J.Hatley and I.A.Pirbhai 著, 立田種宏 監訳, "リアルタイム・システムの構造化分析" 日経 BP 社, 1989

[3] I.J.Hayes,"Applying Formal Specification to Software Development in Industry", IEEE Transactions on Software Engineering,1985

[4] J.Rumbaugh and M.Blaha et.al., "Object-Oriented Modeling and Design", Prentice-Hall,1991

[5] J.S.Collofello and M.Orn,"A Practical Software Maintenance Environment", Conference on Software Maintenance-1988,1988

[6] P.Coad and E.Yourdon, "Object-Oriented Analysis", Prentice-Hall,1991

[7] P.Coad and E.Yourdon, "Object-Oriented Design", Prentice-Hall,1991

[8] P.T.Ward,"How Integrate Object Orientation with Structured Analysis and Design", IEEE Software,1989

[9] 電気学会通信教育会, "シーケンス制御工学", 電気学会, 1988