7J-4

組込みソフトウェア向けデザインパターンとその適用方法

寧 静† 青山 幹雄‡

南山大学 大学院 数理情報研究科† 南山大学 数理情報学部 情報通信学科‡

1. はじめに

組込みソフトウェアの開発では，タイミングなどの多様な設計条件に応じたソフトウェアアーキテクチャと制御ルールの設計が必要である．本研究では，組込みソフトウェアの設計条件に応じた幾つかのデザインパターンを整理し，適切なデザインパターンの適用方法の提案と評価を行う．

2. 組込みソフトウェアのデザインパターン

組込みソフトウェアの設計においてアーキテクチャパターンとデザインパターンが知られている．

2.1. 組込みソフトウェアアーキテクチャパターン

組込みソフトウェアにはプログラムの起動方法の観点から異なるアーキテクチャが混在しており，設計を複雑にする．本稿では代表的な 2 つのアーキテクチャを取り上げる．

(1) タイムトリガ
タイムトリガはタイマを使って，一定周期でプログラムを起動する．

(2) イベントトリガ
イベントトリガはイベントの発生によって起動する．イベントの発生はランダムな場合がある．

2.2. 組込みソフトウェアのデザインパターン

一般にリアルタイム制御を行う組込みソフトウェアのデザインパターンとして，次の 4 つが知られている．

(1) シンプルタスク：1 つの状態遷移マシンで構成され，各状態遷移が単一スレッドで実行される．

(2) シングルスレッドマネージャ：複数の状態遷移マシンで構成され，シングルスレッドで実行される．従ってイベントの待ち行列は単一である．

(3) マルチスレッドマネージャ：複数の状態遷移マシンで構成され，マルチスレッドで実行するように制御する．

(4) マルチタスクマネージャ： 複数の状態遷移マシンで構成され，メッセージを介して独立して並行に実行するように制御する．

さらに,タスクの実行制御の観点から次の 2 つのデザインパターンが知られている．

(5) マルチステートタスク：複数のタスクが一定の制御順序に従って実行される場合，それらのタスクを複数の状態遷移からなる 1 つのタスクに統合する．タスク間の相互作用の組み合わせが減り，設計の難しさを軽減できる．

(6) マルチステージタスク：類似の処理を繰り返し実行するようなタスクを短時間の複数の周期に分割し，一定周期で確実に起動する．

2.3. デザインパターン間の関係

デザインパターンを状態遷移の構造，タスクとスレッドの構造とタスクの起動方法の 3 つの観点から分類する．それによりデザインパターン間の関係を示す．

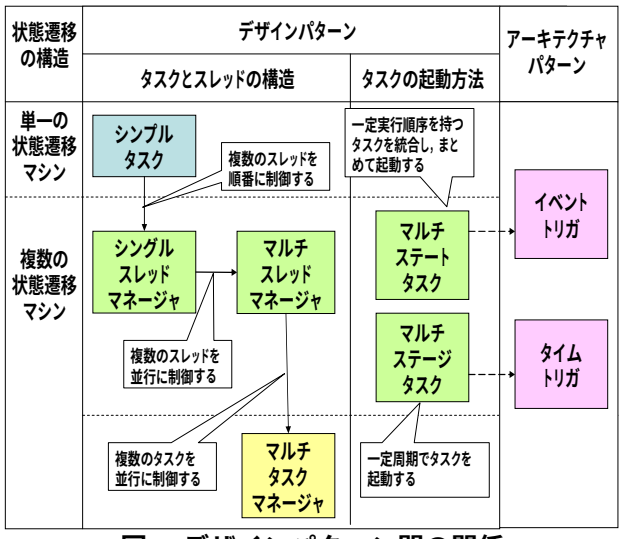

図 1 デザインパターン間の関係

アーキテクチャパターンのイベントトリガとタイムトリガにデザインパターンが適用できる．

イベントトリガの場合，一定の制御順を持つ複数のタスクを統合するためにマルチステートタスクを用いられる．

タイムトリガの場合，周期的に制御を行われる複数のスレッドを統合し，周期時間内に完了させるためにマルチステージタスクが用いられる．

Design Patters for Embedded Software and its Application Method

† Shizuka Nin, ‡ Mikio Aoyama

† Graduate School of Mathematical Sciences and Information Engineering, Nanzan University

‡ Faculty of Mathematical Sciences and Information Engineering, Nanzan University

3. デザインパターンの適用方法

本稿では，イベントトリガにマルチステートタスクを，タイムトリガにマルチステージタスクを適用する方法を考察する．

3.1. マルチステージタスクの適用方法

各タスクを周期ごとに順次実行する．各タスクが必ず 1 周期内に実行を完了しなければならない．例えば，複数のセンサを遠隔監視するシステムがある．パルス数を数え，その値をグローバル値に代入し，ディスプレイ情報を更新する処理を行う．

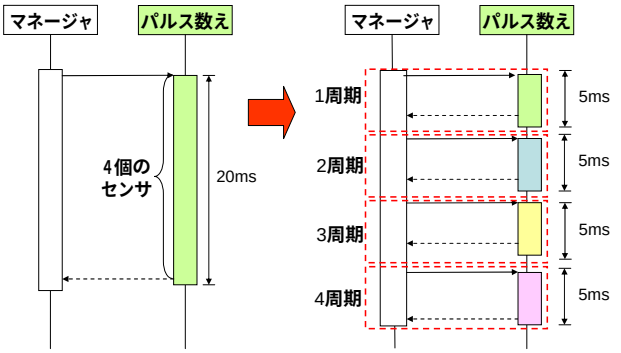

図 2 マルチステージタスクの処理の流れ

この処理を 1 つのタスクにまとめると図 2 の左側に示すように，「パルス数え」タスクの実行時間は 20ms となる．5ms 周期内でこのタスクを実行するためにマルチステージタスクを適用する．その処理の流れを図 2 の右側に示す．

パルス数えを 1 回にまとめて行うのではなく，複数のセンサに分けて 5ms ごとに処理を行う．従って，タスク全体の処理を 4 つのステージに分け，4 つの周期に渡って実行される．パルス数えの実行を分けて行うことによって，一定周期の監視を実現できる．

3.2. マルチステートタスクの適用方法

一定の制御順序に従って実行される複数のタスクを統合して，1 つの状態遷移マシンとする．

洗濯機の制御を例とし，そのユースケースを図 3 に示す．

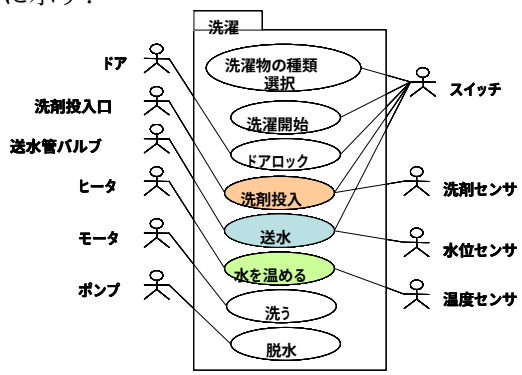

図 3 洗濯機のユースケース

図 3 の中の 3 つのタスク「洗剤投入」，「送水」と「水を温める」と，センサ，アクチュエータ間の相互作用を図 4 に示す．

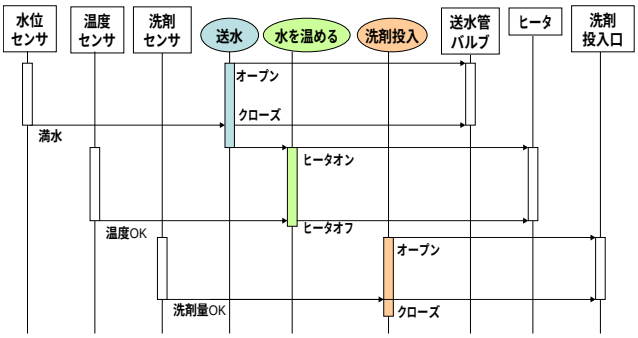

図 4 シーケンス分析

「洗濯投入」と「送水」，「水を温める」は独立であるので並行に実行できる．図 5 の左側に並行処理のステートチャート図を示す．ここで，「水を温める」は水位センサから満水の信号が受け取らなければ実行できない．「送水」と「水を温める」の処理を行う 2 つのタスクを 「水を準備する」タスクに統合する． 統合したステートチャートを図 5 の右側に示す．

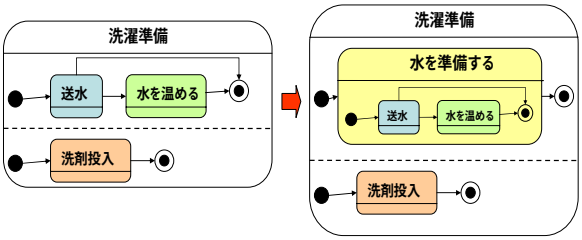

図 5 マルチステートタスクパターンの状態遷移

マルチステートタスクを適用することによって，一定の実行順序を持ったタスク群を見つけ，1 つの状態遷移に統合する．従って，複数のタスク間の相互作用の組み合わせを減らし，設計の複雑さを軽減する．

4. まとめと今後の課題

本稿では，組込みソフトウェアの設計に用いる代表的なアーキテクチャパターンとデザインパターンを分析し，デザインパターンの適用方法を考察した．今後，例題を用いて，デザインパターンの適用性をさらに検証する予定である．

参考文献

[1] M. J. Pont, Patterns for Time-Triggered Embedded Systems, Addison Wesley, 2001.
[2] H. Gommaa, Designing Concurrent, Distributed, and Real-Time Applications with UML, Addison Wesley, 2000.
[3] Event Helix.com, Real-Time Task Design Patterns, http://www.eventhelix.com.