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

M2004MM032　寧　静

指導教員　　青山　幹雄

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開発の問題とアプローチ

組込みソフトウェアとは，機器に組み込まれて，制御を中心に機能を実現するソフトウェアである．

2.1. 組込みソフトウェアの特徴

組込みソフトウェアの特徴として，機器に組み込まれるためハードウェアに依存すること，制御を中心に行うこと，制御が一定時間内に終了しなければならないリアルタイム性を持つことなどが挙げられる．制御の流れを図 1 に示す．

センサが外部信号を感知し，そのイベントを OS に送る．OS が持つスケジューラによってアプリケーションでイベントに対応したタスクを起動する．タスクの起動方法[2]には周期的と非周期的の 2 つがある．タスクの実行により，イベントに対応した制御を行うための制御信号に変換され，OS とアクチュエータに送られる．

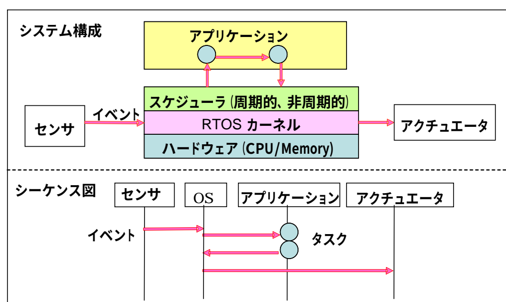

図 1 システム構成と制御の流れ

2.2. 組込みソフトウェアの開発における問題点

本研究では，組込みソフトウェア開発における問題点を以下の 2 点に分けて考える．

(1)　ソフトウェアアーキテクチャ設計[3]における問題

並行処理を行うためにタスクの起動とタスク間の相互作用の制御が重要であり，以下の問題が考えられる．

1)　タスクの起動問題：複数のタスク起動方法が同一のシステムに混在している．

2)　タスクの相互作用問題：大規模な組込みソフトウェアの制御ではタスクの数が莫大になり，組み合わせが複雑である．また，並行処理を行う際にタスク間には，相互作用を持つと設計が一層複雑になる．

(2)　デザインパターンの再利用の問題

組込みソフトウェア開発のためのデザインパターンが提案されているが，実際にデザインパターンが広く設計に使われていないのが現状である．

2.3. 解決アプローチ

本研究では，組込みソフトウェアのアーキテクチャパターンとデザインパターンの特徴と制約条件を整理し，その対応関係を分析する．適切なデザインパターンの適用方法を提案する．組込みソフトウェアの制御の設計はアーキテクチャパターンを決定した後に用いるべきデザインパターンを決定する．アーキテクチャパターンとデザインパターンはそれぞれ特徴や制約条件を持つ．1 つのデザインパターンは適用できる範囲があり，すべてのアーキテクチャパターンに適用できるわけではない．

3. 組込みソフトウェアのパターン

3.1. 組込みソフトウェアのアーキテクチャパターン

一般に組込みソフトウェアのアーキテクチャパターン[5]として，タイムトリガ，イベントトリガの 2 つが知られている．

(1)　タイムトリガの特徴

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．イベントの発生が周期的である．

(2)　イベントトリガの特徴

イベントトリガはイベントの発生によって起動する．イベントの発生は一般に非周期的である．

3.2. 組込みソフトウェアのデザインパターン

リアルタイムの状態遷移のデザインパターン[4]として次の 2 つが知られている．

(S1) 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，一定周期で起動する．

(S2) 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を減らす．

制御をタスクとして実現するためのデザインパターン[1]と

して，次の 4 つが知られている．

(T1) 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．

(T2) 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．

(T3) 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を制御する．

(T4) 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．

4. デザインパターンの適用方法

4.1. アーキテクチャパターンとデザインパターンの関係

組込みソフトウェアのデザインパターンはタスクの起動方法と構造の観点から状態遷移のデザインパターンと実行するためのデザインパターンの 2 段階に抽象化できる．デザインパターン間の関係を図 2 に示す．状態遷移のデザインパターンを用いた設計プロセスを図 3 に示す．

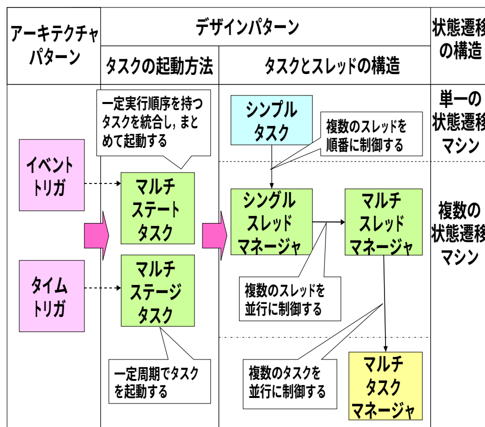

図 2 デザインパターン間の関係

(1) ユースケース分析と相互作用分析を行い，イベントの時間的流れを整理する．

(2) タスクの起動方法が周期的または非周期的のいずれかを決定する．周期的にイベントを発生する場合タイムトリガを適用する．非周期的にイベントが発生する場合イベントトリガを適用する．従って，相互作用分析を行った後にタスクの起動方法によってアーキテクチャパターンを決定できる．

(3) アーキテクチャパターンが決まればデザインパターンが限定できる．アーキテクチャパターンに適用できるのはそのアーキテクチャパターンが持つ特徴や制約条件と一致するデザインパターンである．

(3a) タイムトリガの場合は，タスクを起動する周期が制約条件となる．マルチステージタスクは一定周期内でタスクを起動するために用いられる．従って，タイムトリガとマルチステージタスクは周期という制約条件が一致する．設計プロセスでは，タイムトリガを決定した後に周期分析とマルチステージタスク条件分析を行い，マルチステージタスクを適用するか否かを判断する．

(3b) イベントトリガの場合は，イベントによってタスクを起動する．一定の実行順序を持つタスクを統合しまとめて起動するためにマルチステートタスクを用いる．イベントトリガとマルチステートタスクにはイベントによってタスク間の相互作用を分析する必要がある点で共通している．設計プロセスでは，イベントトリガを決定した後にタスクごとの状態分析とマルチステートタスク条件分析を行い，マルチステートタスクを適用するか否かを判断する．

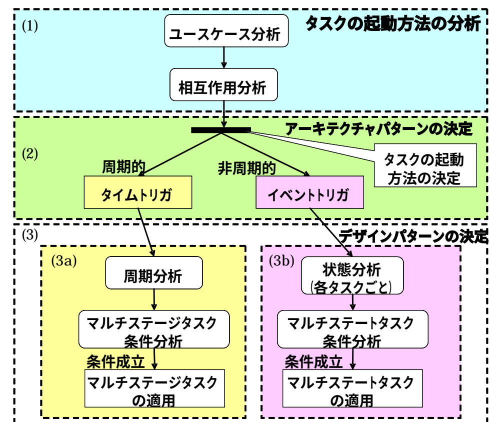

図 3 デザインパターンを用いた設計プロセス

4.2. デザインパターン間の関係

組込みソフトウェアのデザインパターンが状態遷移の構造の観点から分類できる．さらにデザインパターン間の関係をスレッド単位とタスク単位に分けて考える．

4.3. マルチステージタスクの適用方法

4.3.1. マルチステージタスクの適用条件

以下のマルチステージタスクを用いるための 2 つの制約条件を示す．

(1) タスクがタイムトリガアーキテクチャを用いて設計されている．タスク処理が一定周期ごとに繰り返して行う．周期内のタスクの処理能力が限られている．

(2) タスクの実行時間が周期より長く，1 つの周期内に実行し切れず，タスクを分割しなければならない．

周期の決定条件にハードウェアの物理的な制約とソフトウェアの処理能力がある．表 1 にマルチステージタスクの適用事例と周期決定条件の分類を示す．

表 1 ： マルチステージタスクの適用範囲

周期の決定条件	適用事例
物理的	CD-R / DVD データ転送，車輪制御
ソフトウェア的	ディスプレイ更新，データ転送

4.3.2. マルチステージタスクを用いた設計プロセス

タイムトリガの設計にマルチステージタスクを用いた設計プロセスを図 4 に示す.

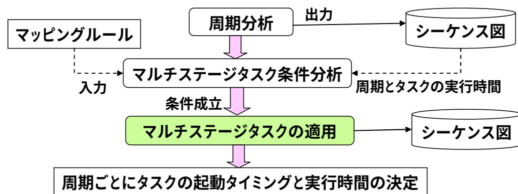

図 4 マルチステージタスクを用いた設計プロセス

(1) 周期分析: 制御条件としての物理的また, ソフトウェア処理能力による周期を分析し, タスクの実行時間を明確にする. シーケンス図を用いて制御の流れを表す.
(2) マルチステージタスク条件分析: マルチステージタスクを用いるためのマッピングルールと周期, タスクの実行時間を比較し, 条件が成立した場合, マルチステージタスクを適用する.
(3) マルチステージタスクの適用: タスクの実行が周期内に終了するように周期ごとにタスクの起動タイミングと実行時間を決定する. シーケンス図を用いてマルチステージタスクを適用した後の制御の流れを表す.

マルチステージタスクをタイムトリガに用いない場合は, 設計されたタスクの実行時間が周期より長くなり, 設計が後戻ししなければならない可能性がある. マルチステージタスクの適用を周期分析を行った後にすることで, タスクを起動するためのタイミング設計が周期内で確実にできる.

4.4. マルチマルチステートタスクの適用方法

4.4.1. マルチステートタスクの適用条件

以下の 2 つの制約条件を当てはまる場合, マルチステートタスクが適用できる.

(1) 複数のタスクが一定の実行順序を持つ.
(2) タスク間のコントロールが単一のタスクによって制御できる.

4.4.2. マルチステートタスクを用いた設計プロセス

イベントトリガの設計にマルチステートタスクを用いた設計プロセスを図 5 に示す.

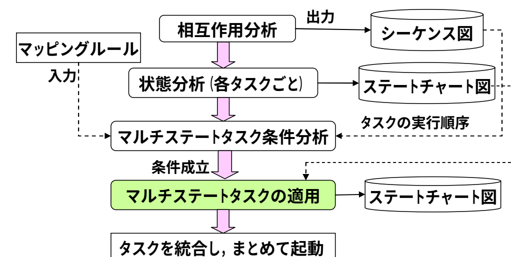

図 5 マルチステートタスクを用いた設計プロセス

(1) 相互作用分析: イベントの流れを把握するために行う. シーケンス図を用いてセンサからアクチュエータにいたるイベントの流れを表す. アクチュエータの制御状況とイベントの流れに注目し, 状態の変化を発見し, 状態を抽出する.
(2) 状態分析: 相互作用分析で抽出されたタスクごとの状態を分析し, ステートチャート図を用いて個々の状態遷移を表す.
(3) マルチステートタスク分析: マルチステートタスクを用いるための条件とタスクの実行順序を比較し, タスクの実行が一定の順序に従って行う場合は条件が成立し, マルチステートタスクを適用する.
(4) マルチステートタスクの適用: マッピングルールにマッチしたタスクを 1 つのタスクによって起動するように統合する. ステートチャート図を用いて個々に表したタスクの状態遷移を 1 つにまとめる.

大規模なシステムではイベントの数とタスクの数が莫大になる. タスク間の相互作用の数が増え, 設計が複雑になる. 相互作用分析を行い, タスク間の実行順序を明確にした後にマルチステートタスクを適用することで一定の実行順序を持つタスク群を統合しまとめて起動でき, タスク間の相互作用の管理をしやすくなり, 設計の複雑さを軽減できる.

5. デザインパターンの適用例

5.1. マルチステージタスクの適用例

軸の回転スピードをディスプレイに表示するシステムの制御に適用した例を説明する. 軸回転によって表れるパルスの数を数えて, 複数のセンサによって遠隔監視するシステムの制御を行う.

制御の流れは, パルス数えるためのタスクを実行し, 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. タスクの実行時間が 20ms である. ディスプレイの更新が 5ms 秒ずつ行うため, タスクを 5ms の周期内に実行を完了しなければならない. 相互作用分析によりタスク「パルス数え」を図 6 の左側に示す.

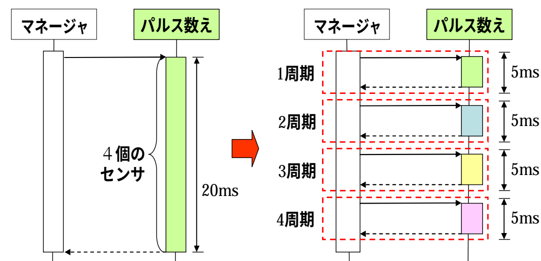

図 6 遠隔監視システムのシーケンス

(1) 周期分析： 軸回転によってパルスを数えているため, 周期は回転によって決まる. 周期が 5ms であり, タスク「パルス数え」の実行時間が 20ms である. 周期がタスクの実行時間より短い.
(2) マルチステージタスク条件分析：周期は物理的な制約条件で決まり, 周期がタスクの実行時間より短いということが周期分析から分かる. パルスの数が

軸の回転スピードによって変化するが，一定の時間内に数えられるパルスの数が限られている．タスクの実行を 5ms 以内に終了させるために処理を分割しなければならない．従って，マルチステージタスクが適用できる．

(3) マルチステージタスクの適用：タスクの実行時間が 20ms であり，周期が 5ms であるため，周期内にタスクを実行させるには，処理を 4 つのセンサで行い，1 つのセンサにつきタスクを 5ms 実行させる．処理の流れを図 4 の右側に示す．

この例ではマルチステージタスクを適用し，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，タスク全体の処理が 4 つのステージに分けられ，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．

5.2. マルチステートタスクの適用方法

マルチステージタスクを洗濯機の制御に適用した例を説明する．制御の流れは，ボタンによって洗濯物の種類を選択し，洗濯を開始する．ドアがロックされ洗剤と水を投入する．水の温度はあらかじめ決められた洗濯の種類によって必要に応じて温めることができる．水と洗剤の準備ができれば洗濯を行い，洗濯し終われば脱水処理を行う．

(1) ユースケース分析：洗濯機のユースケースを抽出し図 7 に示す．

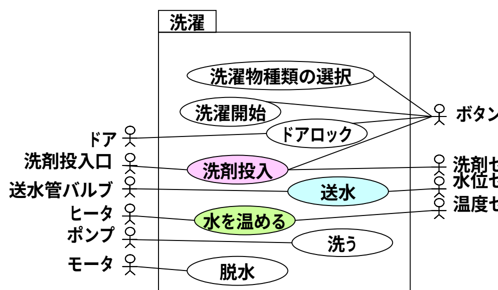

図 7 洗濯機のユースケース

(2) 相互作用分析と状態分析：3 つのタスク「洗剤投入」，「送水」と「水を温める」間の相互作用を図 8 に示す．「洗剤投入」と「送水」，「水を温める」の間に制御順序が存在しない．「送水」から「水を温める」へ遷移するために満水のイベントが必要であるため，タスク「送水」と「水を温める」間に順序関係が存在する．

(3) マルチステートタスク条件分析：相互作用分析で実行順序を持つタスク「送水」と「水を温める」を抽出できたため，2 つのタスクをまとめて起動できる．従って，マルチステートタスクが適用できる．

(4) マルチステートタスクの適用：2 つのタスク「送水」と「水を温める」を統合し，その起動を「水を準備する」タスクにまとめて行う．統合したステートチャートを図 9 の右側に示す．

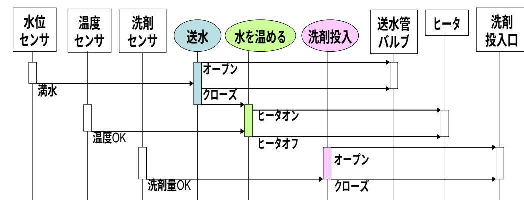

図 8 洗濯機の相互作用

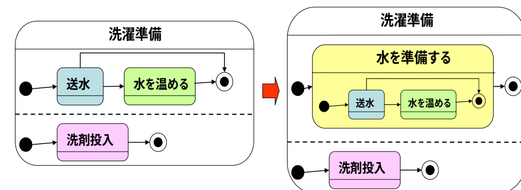

図 9 マルチステージタスクを用いた状態遷移

この例ではマルチステートタスクを適用し 2 つのタスク「送水」と「水を温める」をまとめて起動できた．大規模システムでは，マルチステートタスクの適用条件を満たす複数のタスクをまとめて一連の状態遷移にまとめることで，タスク間の相互作用を管理しやすくし，設計の複雑さを軽減できる．

6. 今後の課題

マルチステージタスクとマルチステートタスクの適用性を例題を用いてさらに検証する．実行するためのデザインパターンの適用方法の提案をする．

7. まとめ

本研究では，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た．イベントトリガとタイムトリガに注目し，状態遷移のデザインパターンマルチステージタスクとマルチステートタスクの適用方法を提案した．例を用いてその適用性を示した．

参考文献

[1] Event Helix.com, Real-Time Task Design Patterns, http://www.eventhelix.com.

[2] 藤倉 俊幸, リアルタイム/マルチタスクシステムの徹底研究, Vol.15, CQ 出版社, 2003.

[3] H. Gomaa, Designing Concurrent, Distributed, and Real-Time Applications with UML, Addison Wesley, 2000.

[4] M. J. Pont, Patterns for Time-Triggered Embedded Systems, Addison Wesley, 2001.

[5] TimeSys, The Concise Handbook of Read-Time Systems, Version1.3 2002.