2K－9

オブジェクト指向開発アプローチ Crossover(5) - 実行確認系 II -

池田健次郎, 岸知二
NEC マイコンソフト開発環境研究所

1　はじめに

我々はオブジェクト指向開発アプローチ Crossover の検討を進めている。[1] では、特にリアルタイムシステムを例に取り、実行確認系に必要な要件や実現の為の基本方針について述べた。本稿では、実行確認系の基本実行モデル、設計モデルの対応、及びその実現方式についてべる。

2　実行確認系の要件

実行確認系は、対象システムの設計モデルの抽象度が高い場合や実環境が使えない場合に、疑似的な実行環境においてシステム全体としての動作を確認することを目的としている。

実行確認系に必要な要件は以下のとおりである。

- 実環境での (並列) 動作を疑似できること
- 設計モデルとソースコードを混在して実行できること
- 抽象度が変わっても設計モデルに大きな変更が生じない実行モデルを持っていること

3　基本実行モデル

3.1　モデルの定義

実行確認系の対象となるシステムは、メソッドの連続した処理の流れ (スレッド) が同じ時間軸上で協調しながら並列動作しているモデルとして表現できる。

システム中のオブジェクトのメソッドには、同期的に呼出されるメソッドと非同期に呼出されるメソッドがある。同期的に呼出されるメソッドが呼ばれると、呼出した側の制御が停止して呼出されたメソッドに制御が移り、呼出されたメソッドが終了すると共に、呼出した側に制御が戻る。つまり、呼出されたメソッドの実行は、呼出したスレッドの一部となる。非同期に呼出されるメソッドが呼ばれると、呼びだした側の制御は停止せず実行が継続され、それと並行して呼出されたメソッドの実行が行われる。つまり、呼出されたメソッドを起点として、新たなスレッドが生成される。

スレッドの削除は、起点となるメソッドが全ての処理を終了した時点か、スレッド自体を中断する為のメソッドが呼ばれた時点に行われる。

システムを構成するオブジェクトの生成 / 削除は、専用のメソッドを呼出す事により行われる。

3.2　設計モデルとの対応

3.2.1　抽象度の高いモデル

抽象度の高い設計モデルにおいては、ターゲット環境での実現方法がモデル上に記述される事はない。その代わり、システム全体の振る舞いを表現する為に、各メソッドが同期的に呼ばれるかメソッドなのか、非同期に呼ばれるメソッドなのか指定される。実行モデル上では、それぞれ、呼出し元のスレッドの一部として、もしくは、新たにスレッドとして表現される。

3.2.2　抽象度の低いモデル

一方、抽象度の低いモデルにおいては、タスクやプロセスとか言ったターゲット環境での概念が設

"Object-Oriented Development Approach: Crossover (5) - Simulator II -", Kenjiroh IKEDA and Tomoji KISHI, NEC Corporation

計モデルに記述されているが、その実行意味を実行モデル上で表現することが出来る。

例えば、リアルタイム OS 上のシステムの設計モデルでは Task や MailBox 等が記述されている。Task は実行モデル上で、一つのスレッドとして表現される。非同期通信を実現する MailBox は、複数のスレッドから属性値にアクセスされるオブジェクトとして表現される。

また、UNIX 上のシステムの設計モデルでは、処理の単位としてプロセスが記述される。プロセスを構成するオブジェクトと対応するスレッドの間には強い存在依存性があるが、これは、実行モデル上でのプロセスに対応したスレッドの削除と、プロセスを構成するオブジェクト群の削除として表現される。

4 実現方式

3節で定義した実行モデルをスタックフレーム切り替え方式とインクリメンタルリンカの関数フック機能を利用することとした。

これらを選択した理由は以下のとおりである。

4.1 並列動作の疑似

スレッド並列動作を疑似させる為の現実的な方法として、以下の 2 つが考えられる。

1. スレッドを実行確認系が動作する処理系が提供する並列動作単位 (UNIX 上のプロセス等) に対応させる
2. スレッドの動作状況をスタックフレームで管理し、スタックフレームを切り替える

前者の場合、スレッドを並列動作単位に対応させるだけで並列性が実現できるが、多くの場合システムにかかる負荷が大きくかかるので、スレッドの数が多い場合は事実上シミュレート不可能になる。

後者の場合は、システムにかける負荷も比較的小さく、スレッドの動作を自由に制御する事ができる。

4.2 ソースコードの混在

設計モデルでの仕様記述と既存ソースコードと混在させて動作させる方式には、

1. 仕様記述からスタックフレームを切り替えるコードを埋め込んだソースコードを生成し、同様のコードを埋め込んだ既存ソースコードと共にコンパイル、リンクしたものを動作させる。
2. 仕様記述から生成した実行可能なソースコードと既存ソースコードと共にコンパイルして、インクリメンタルリンカで関数単位で呼びだし実行する。スタックフレームを切り替えは、インクリメンタルリンカを用いて関数を呼び出す部分に組み込む。

が考えられる。

1 の方式では、スタックフレームを切り替え並列性を疑似するためのコードを何処に挿入するかと言う問題点がある。

2 の方式では、対象システムのコードが関数単位で呼び出されるので、スタックフレームの切り替えは関数呼び出し単位に固定されてしまうが、システムの振る舞いに関係するスレッド間のインタラクションはメソッド単位で行われるので問題無いと考えられる。

5 おわりに

実行確認系で動作させる為の基本実行モデルを明らかにし、要件を満たす実現方式について考察した。ここでの考察に基づき実行確認系の実装を行い、その有効性を検証していく予定である。

参考文献

[1] 前川佳春, 他: オブジェクト指向開発アプローチ Crossover (3) - 実行確認系 -. 第 49 回情処全国大会 (1994).

[2] TL ハンセン, 他:C++ アンサーブック. トッパン,(1992).