「Kellner のソフトウェアプロセス問題」の記述の試み

7J－1

岡田世志彦[1] 松永泰明[1] 飯田元[1] 井上克郎[1] 鳥居宏次[1] 永岡渡[2] 梅本肇[2] 酒井睦雄[3]

[1]大阪大学基礎工学部 [2]日立西部ソフトウェア（株） [3]（株）日立製作所

1. はじめに

ソフトウェアの開発過程（ソフトウェアプロセス）の記述方法を評価するための例題として Marc Kellner の「ソフトウェアプロセスモデリングのための例題」[1]が提案されている．本稿ではこの問題について，ソフトウェアプロセスにおける複数の作業の流れとその同期に特に着目し，ペトリネットや開発過程記述言語ＰＤＬ (Process Description Language) [2] [3]を用いて記述した．

2. Kellner の問題の概要

Kellner の例題はソフトウェアの変更作業の一部を 14 ページにわたって英文で規定している．この作業をいろいろな方法でモデル化／記述することによって，それぞれの方法の理解や把握をすることができる．ここでは，その中の "核問題" と呼ばれる部分についての記述を行なう．この問題全体が規定する作業には Develop Change and Test Unit という名前がつけられており，次の８つのサブステップに分けられている．

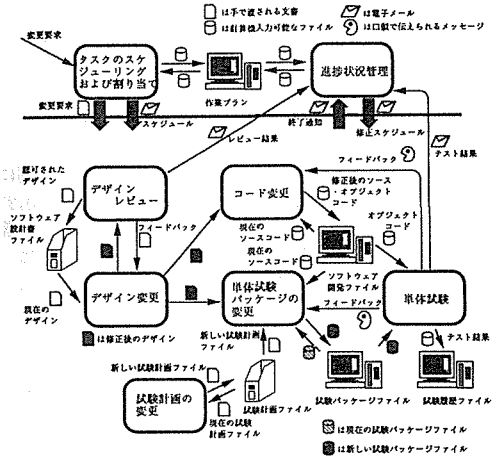

図１　各ステップ間の関係の概略

(1) **Schedule and Assign Tasks** （タスクのスケジューリングおよび割り当て）ソフトウェアの変更に関する作業のスケジュールを作成し，個々のタスクをメンバに割り当てる．

(2) **Modify Design** （デザイン変更）要求変更により影響を受ける単体コードユニットのデザインの修正を行なう．

(3) **Review Design** （デザインレビュー）修正されたデザインのレビューを行なう．結果によってはデザインの変更のステップをやり直す．

(4) **Modify Code** （コード変更）デザインの変更をコードに実現し，コンパイルを行ないオブジェクトコードを生成する．

(5) **Modify Test Plan** （試験計画の変更）要求変更に関する機能の試験を行なうための試験計画の変更を行なう．

(6) **Modify Unit Test Package** （単体試験パッケージの変更）試験計画の変更に従って，試験パッケージの変更を行なう．

(7) **Test Unit** （単体試験）修正されたコードについて試験パッケージを実行し，その結果の解析を行なう。試験が不合格である場合は，ソースコードの修正，単体試験の修正の一方または両方を行なった後，再試験を行なう．

(8) **Monitor Progress** （進捗状況管理）上記の各ステップの進捗状況を管理する．スケジュールからの逸脱が深刻な場合には，再スケジューリングを行なう．

各ステップにはそれぞれ様々な開始・終了条件などが付加されている．また，ここで行なう変更は比較的局所的で，あるモジュールを変更しても他のモジュールには影響を与えない．別のモジュールについて新たに変更作業を行なったり，関連するモジュールの整合性を調べるという作業は必要ない．プロジェクトマネージャがスケジューリングし，仕事を割り当てることでこのプロセスは開始され，新しいコードがテストに合格すると終了する．各ステップ間の関係の概略を図１に示す．

3. ステップ間の同期の記述

まず，この核問題について，特に同時に並行して行なわれる複数のステップの同期に着目することにし，これらのステップのタイムチャートを記述してみた（図２）．このようなタイムチャートでは，以下の点について明確に記述することが困難である．

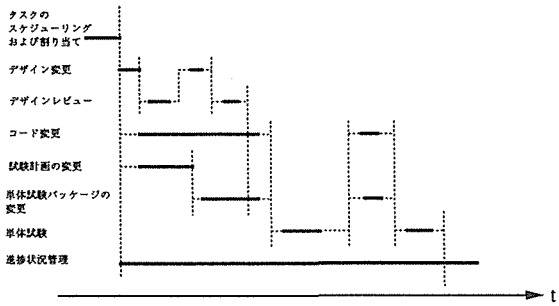

図２　各ステップのタイムチャート

・タスクのスケジューリングおよび割り当てが終了した後であれば，いつ始めてもよいようなステップ（例えばコード変更）についての記述．

・どちらが先に終了しても良いが，その両方が終了しないと，つぎのステップに移れないコード変更と単体試験パッケージの変更のようなステップの記述．

・デザインが認可されてからでなければ終了できないコード変更などのステップの記述．

・単体試験が不合格であるときのように，コード変更，単体試験パッケージの変更の両方または片方が繰り返されるような場合のプロセスの記述．

そこで、ステップの同期をより明確に記述するために，４章では，ペトリネット[4]を用いて記述を試みた（図３）．

A Description of Kellner's Software Process Problem
Yoshihiko OKADA[1] Yasuaki MATSUNAGA[1] Hajimu IIDA[1] Katsuro INOUE[1] Koji Torii[1]
Wataru NAGAOKA[2] Hajimu UMEMOTO[2] and Mutsuo SAKAI[3]
[1]Faculty of Engineering Science, Osaka University.
[2]Hitachi Seibu Software Co., Ltd. [3]Hitachi, Ltd.

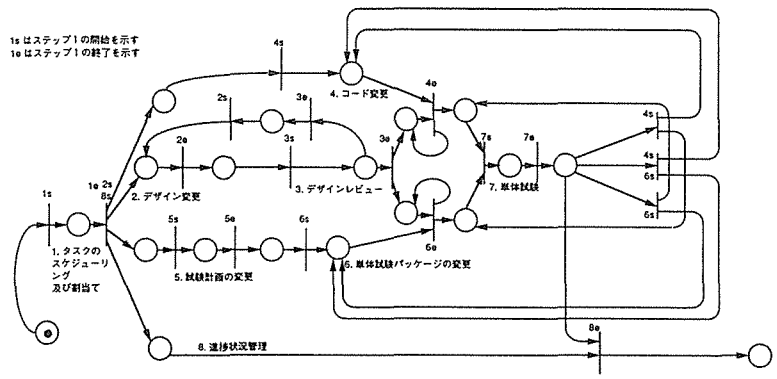

図3　ペトリネットによる記述

4.　ペトリネットによる記述

図3において，トランジション（縦棒）はステップの開始・終了を意味する．また，プレース（円）はステップの開始・終了のための前提条件・完了条件を示す．

ペトリネットを用いることにより，次のようなことが明確に記述できた．

・タスクのスケジューリングおよび割り当てが終了と同時に，デザイン変更，進捗状況管理が開始される．また，この時点でコード変更と試験計画の変更が開始可能になるが，直ちに始める必要はない．

・デザイン変更，デザインレビューの終了には，デザイン認可後の終了と，さらに変更の繰り返しの場合の終了がある．

・単体試験パッケージの変更は，試験計画の変更の終了後に開始される．

・コード変更，試験計画の変更，単体試験パッケージの変更はデザインが認可されていなくても開始できるが，コード変更，単体試験パッケージの変更については，デザインが認可されていなければ終了できない．

・単体試験は，コード変更，単体試験パッケージの変更の終了後開始され，試験の結果によってコード変更，単体試験パッケージの変更がともに繰り返される場合と，どちらか一方が繰り返される場合の３通りが考えられる．

・単体試験が成功して終了した場合には，進捗状況管理が終了可能となり全ステップが終了する．

5.　ＰＤＬによる記述

ペトリネットによる記述から，開発過程記述言語ＰＤＬへの変換を行ない，動作の確認を行なった．

ＰＤＬは関数型言語であり，ツールの起動，ウィンドウのオープン・クローズなどの操作を行なうための基本関数を備えており，これらの関数を組み合わせてソフトウェアプロセスを記述する．また，ＰＤＬインタプリタによってその記述が実行可能である．

ここでは，次のような変換を行なった（図4）．

図中 a. のプレースの変換において，トークンは１個ずつ処理することを仮定する．したがってプレースに入る矢印の本数は関数の記述には関係しない．プレースから出ていく矢印の本数についての分岐だけを考えればよい．ここではメニュー関数を用いて人間が選択することにしたが，適当な状態遷移確率を与えて，自動的にシミュレートするような関数に変更することもできる．

図中 b. のトランジションの変換において，入力が複数ある場合には，全ての前提条件をみたすかどうかを判定している．また，出力が複数ある場合には全ての処理が並列に実行される．すなわち，トランジションに m 個のプレースからの入力があり n 個のプレースへの出力がある場合には，m 回の条件判定が行なわれ1回だけ n 個の関数が並列に実行される．ＰＤＬでは@で結ぶことにより並列に実行することを示す．

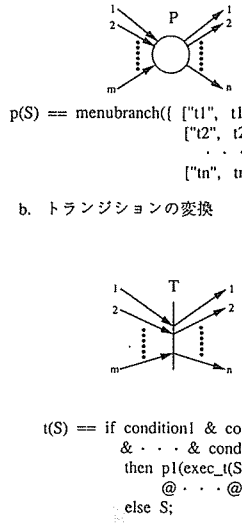

図4　ペトリネットからＰＤＬへの変換

これらの変換は最も基本的なものであり，前提・完了条件が常にみたされることがわかっている場合や，分岐しないような場合には，もっと簡略化して関数を記述している．また，あるトランジションについて前提条件がみたされてもすぐに発火する必要がない場合には，待ち状態となる関数を加えて記述している．

このようにして作成されたＰＤＬの記述をＰＤＬインタプリタで直接実行し，適切な動作が得られた．具体的には，画面上でプロセスの実行が開始されるとウィンドウが開き，終了した場合には閉じる．また，選択が必要な場合にはメニューが表示される．また，この記述をもとに各ステップをより詳細に記述すれば，実際に開発支援を行なうことも可能である．

6.　おわりに

Kellner のソフトウェアプロセスモデリングのための例題の核問題についてプロセスの同期に着目した記述を行なった．今後は，視点を変えて，ソフトウェアプロセスの中で現れるプロダクトに着目した記述や，拡張問題の記述についても取り組むことを考えている．

文　　献

(1) Marc Kellner: “Software Process Modeling Example Problem”, Private Note, August, (1990).

(2) 荻原剛志, 井上克郎, 鳥居宏次: “ソフトウェア開発を支援するツール起動自動制御システム”, 信学論 (D-I), J72-D-I, 10, pp.742-749 (平 1-10).

(3) 稲田良造, 荻原剛志, 井上克郎, 鳥居宏次: “ソフトウェア開発過程の形式化とその詳細化による支援システムの作成－ＪＳＤを例として－”, 信学論 (D-I), J72-D-I, 12, pp.874-882 (平 1-12).

(4) James L. Peterson: “Petri Net Theory and the Modeling of Systems”, Prentice-Hall (1981).