1A3-2

並行プロセスモデルに基づく産業用ロボットシステムの階層分散制御

Hierarchical and Distributed Control of Industrial Robotic Systems Based on Concurrent Process Modeling

安田　元一（長崎総合科学大）

Gen'ichi YASUDA, Nagasaki Institute of Applied Science, yasuda_genichi@pilot.nias.ac.jp

This paper describes a systematic methodology which implements discrete event hierarchical and distributed control of industrial robotic systems based on concurrent process modeling. A hierarchical Petri net simulator with additional capabilities of command sending and system status monitoring to and from its external environment was developed. Combined with the simulator, real-time multitasking control of robots and machines is implemented such that the discrete event dynamic behavior of a real robotic system is in accordance with that of the net model of the system. The methodology used is applied to produce semi-automatically control software using multithreaded programming for an example robotic cell.

Key Words: Industrial robotic systems, Concurrent processes, Modeling, Hierarchical and distributed control

1. 緒言

ロボットを用いた生産システムにおいては、複数の知能化機器を要素とする並行プロセスのモデル化と統括制御が必要となる。本研究では機器に対するコマンド実行と状態監視の機能を付加した離散事象ネットモデルのシミュレータ開発を行い、複数の機器の動作を並行して実行するマルチスレッドプログラミングを利用した統合制御システムの構築と，シミュレーションと実験によるシステムの動作確認を行った。

2. 並行プロセスモデルとシミュレータ

ペトリネットは相互作用する並行プロセスを含む離散事象システムを表現するのに適しており、その要素として事象を示すトランジション、状況を示すプレース、事象と状況の順序関係を示すアーク、状況が保持されていることを示すトークンがある。ネットにおいて事象が発生し、状況が次の状況に移り変わるときにトランジションが発火するという。システムのシミュレーションはトランジションの発火則にしたがい、プレース内のトークンが次のプレースに移動し状態遷移していくことで実行される。本研究では機器の動作状態の監視と制御を考慮して状況・事象ネットを対象とし、発火則は全ての入力プレースにトークンがあり、かつ全ての出力プレースにトークンがないとき発火可能とする。さらにネットモデルの内部および外部からトランジションの挙動を制御するためのゲートを組み込み、信号入力を可能とした。

ソフトウェアは WindowsXP のもとで Visual C# 2005 を使用して作成した。ネットモデルによるシステムのモデリングでは、Visual C#のイベント駆動のプログラムにより、マウスのクリックとドラッグ＆ドロップを用いて描画したい要素の選択と位置設定、およびアークの描画を行う。描画時に座標データが取得され、アーク接続時にトランジションおよびプレースの範囲内にアークの始点、終点の座標が入っていなかった場合や、トランジションまたはプレース同士をアークにより接続しようとした場合、接続エラーを表示する。

描画完了の後、ボタン操作によりネットモデルの内部データを自動的に作成する。内部データにはネットのトランジションとプレースの接続関係、トランジションとゲートの配置関係、複数のトランジションが接続されている競合プレースに対して接続されているトランジション番号、および各プレースにおけるトークンの有無があり、描画時に得る座標データをもとに作成し、ウィンドウに表示することによりシステムの状態監視が可能である。また座標データを記憶させることでグラフィックデータの色替え、修正、保存、展開が可能である。

ネットモデルの初期マーキングを配置した後、『シミュレーション開始』ボタンを押すと、描画したモデルに対するシミュレーションを開始する。１つのトランジションに対して接続関係にあるプレースを調べ、プレースが入力側接続のときはマーキング表によりそのプレースにトークンがなければ 0、プレースが出力側接続のときはマーキング表によりそのプレースにトークンがあれば 0、それ以外の場合は 1 の値をもつデータを作成し、入力側と出力側それぞれについて各プレースのデータの和をとり、その値がプレースの総数より少なければトランジション発火不可であることを示す。全てのトランジションに対して上記を調べることで、現時点でトランジションの番号ごとに発火可能（1），発火不可（0）を表したトランジション情報表を作成する。

トークンの状態により、これ以上発火ができない状況をデッドロックという。発火可能なトランジションがない、すなわちトランジション情報表の値がすべてのトランジションにおいて 0 ならばデッドロックと判定し、シミュレーションおよび機器制御を停止する。

デッドロックと判定されなかった場合、トランジション情報表において 1 の値をもつ発火可能トランジションについて外部ゲート条件が成立しているか否かをゲート条件表と対応する外部信号の値から調べ発火不可と判断されたトランジションに対してトランジション情報表の値を 0 に書き換える。トランジション情報表を用いることでシステムの停止状態が外部ゲートの成立を待っているためなのか、それともデッドロックなのかを判定することができる。

この時点で発火可能なトランジションがなく、さらに外部ゲートが存在する場合に外部ゲート状態待ちとなる。これは外部ゲート条件テストが終わった時点でのトランジション情報表による判定であり、もし外部ゲートが存在する場合は外部信号によって発火不可と判断されるものがあると考えられるからである。シミュレータによる機器の制御を行う場合は、機器の動作完了を次回のトランジション発火許可条件とするために外部ゲートを用い、動作完了信号を受信したら外部ゲート設定を自動的に発火許可状態に変更するようにした。

実験システムはロボット(Panasonic製 PanaRobo KS-V20)、コンベア、加工機からなり、コンベア1により搬入されたワークをロボットが加工機へとローディングし、加工機で作業を終了したワークをロボットがコンベア2へアンローディングし、コンベアが搬出するという作業を実施する。このときロボット、コンベア、加工機がそれぞれ他の機器と同期をとりながら協調動作を実行している。実際に機器制御を行う場合はシミュレーション開始前にシミュレータ内においてトランジションごとにコマンドの登録を行う。

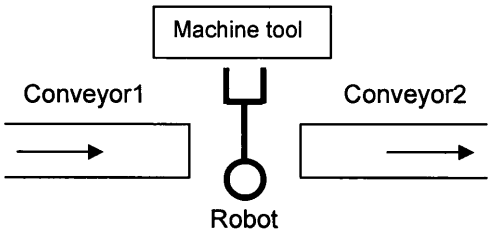

Fig. 1 Configuration of robotic cell

3. マルチスレッドを利用した制御システムの設計

複数の機器で構成されるシステムのマクロなネットモデルを上位のコントローラに配置し、上位からの作業命令を下位のコントローラが受けて詳細な作業を実行するための制御システムを構築した。マクロなネットモデルで機器の動作を表すプレースにトークンが入ることで下位のコントローラにコマンドが送信され、所定のシーケンス動作を実行する。下位のコントローラは一連の動作を終了すると上位コントローラに終了信号を送信し、上位コントローラはこの受信によってトークンの移動が可能になる。複数機器による並列動作や作業の選択などにおいて、上位コントローラはネットモデルを用いた統合を行い、発火トランジションの決定を行う。今回は下位コントローラとして、ロボットには市販のロボットコントローラ、コンベアやマシニングセンタなどにはプログラマブルロジックコントローラ(PLC)を用い、送受信とシーケンス動作のプログラムをあらかじめ組み込んで使用した。

シミュレータを用いた統括制御のためのソフトウェアとして、ネットモデルの作成、描画用プログラムのほか、上位コントローラ用として、外部ゲート入力が可能なシミュレーションプログラムと、下位コントローラ用としてシリアル通信によるタスク実行のためのプログラムが必要となる。これら複数の内容のプログラムを協調させて効率よくシステム全体の制御を実施するために1台のPC上でVisual C# 2005を使用しマルチスレッドプログラミングを用いて作成した。各スレッドの処理内容を図2、3に示す。

シミュレーション開始によりシミュレーションスレッドが起動し無限ループの状態となる。シミュレーションスレッドの動作によりトランジションが発火するとタスク実行スレッドが起動し、そのトランジションに設定されたコマンドを機器に送信する。機器からの返信内容を繰り返し状態検知処理し、正常終了が確認できたら、共有変数である外部ゲート変数を書込み処理により変更してスレッド実行を終了する。コマンドの送信後、シミュレーションスレッドは外部ゲート条件テストにおいて、発火不可（外部ゲート状態待ち）と判定される状況が続くが、外部ゲート変数の読出し処理でオンと判定されることでゲート条件表の信号値を更新し次のトランジションの発火条件につなげる。

シミュレーションスレッドの外部ゲート条件テストプログラムとタスク実行スレッドの外部ゲート設定変更プログラムからの共有変数へのアクセスには共通のメソッド関数を用い、C# の lock ステートメントで全体をくくることで排他制御を行った[1]。実験ではモデリングスレッド以外の各スレッドは並行して動作する必要があるため実行の優先順位は同一とした。実行結果よりシミュレーションスレッドとタスク実行スレッドが、メソッド関数を実行し処理を取得できたタイミングで交互に切り替わりながら処理を実施していることが確認できた。各機器ごとにタスク実行スレッドと共有変数である外部ゲート変数を設け、各々のスレッドが非同期かつ並行して動作することで分散的な制御が達成できる。

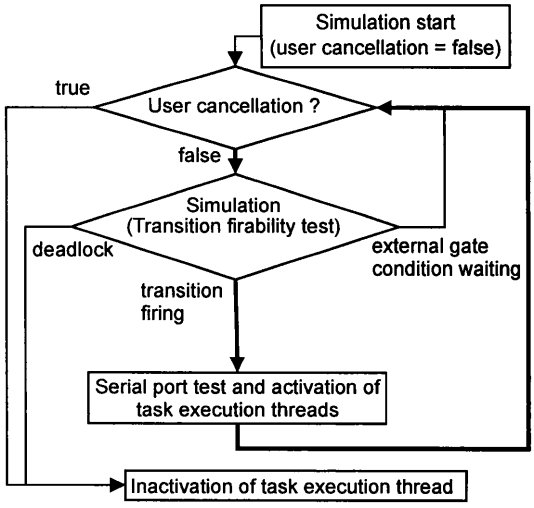

Fig. 2 Flowchart of simulation thread

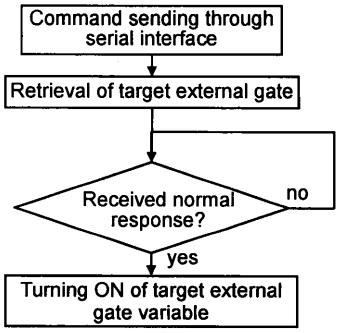

Fig. 3 Flowchart of task execution thread

4. 結言

生産システムにおける各機器の動作と複数機器による協調動作を表現するネットモデルに基づくシミュレータとシステム制御用ソフトウェアをマルチスレッドプログラミングを用いて作成し、実験とシミュレーションによる動作確認を行った。今回は1台のPCを用いてプログラム作成を行ったが、上位コントローラおよび各機器ごとの下位コントローラにそれぞれ高速のマイコンを割り当て、ネットワーク上でリアルタイムOSを用いて並列処理を行うことによりさらにスムーズな制御が可能になると考えられる。

文 献

[1] http://msdn.microsoft.com/ja-jp/library/cc434762(v=VS.71).aspx, "プログラミングC#、第15章 マルチスレッドプログラミング".