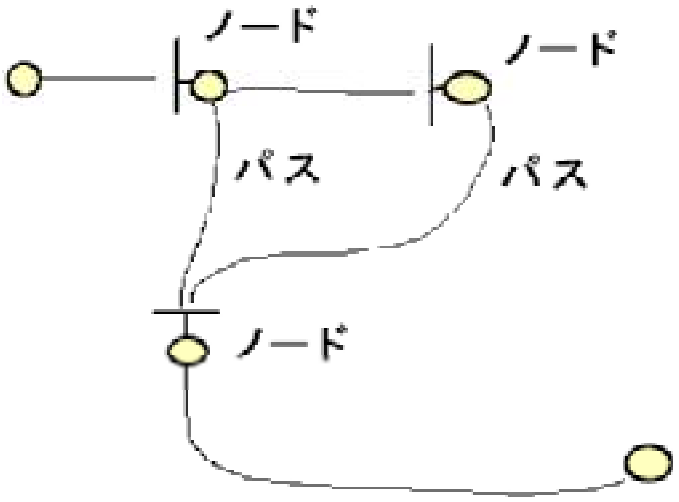
考察：「データーフロープログラミング」

１．まえがき

まとまった処理を担当するプログラムをノードとし、ノードからノードへの制御の推移をパスと呼ぶことにする。複雑な問題解決をこのノード群とその上のパスセット（フロー）で、実現しよう、というのが論旨である。並列処理機能を持った、こうしたプログラミングスタイルをデータフロープログラミングと本論では呼んでいる。



このようなプログラミングスタイルを持ったシステムとして、メインフレームのジョブ制御文がある。あるプログラムの実行が終わった時の、次に実行するプログラムをジョブ文で指定する、つまり、プログラムの実行スケジュールを定義できるものである。使用する入出力ファイルもジョブ文で記述する。ジョブ文は簡単なフロー管理を行うのである。

コンポーネントウェアというプログラミングスタイルも、ノード群とフローを分離して行う点が同じである。プログラミングをし易くしてくれる方法論である。

２．データフロープログラミングの考え方

ノードはプロパティとか、フィールド、プロセス（並行処理の基本単位）をまとめて扱う単位で、実際にはあるメモリ領域となる。Java ではクラスとして定義されるべきものである。

２．１　動作概要

Java での実現を想定して、データフロープログラミングの動作を以下に論じていきたい。

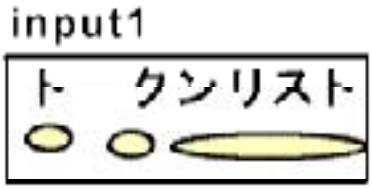
先ず、パラメータクラスとプロセスクラスを作る。パラメータクラスは処理に必要なデータ、処理結果を保存するクラスであり、プロセスクラスは並列に動く処理を表現したクラスである。

次に、パラメータとプロセスの出会いを表現するノードクラスと、データの流れを表現する、フロー管理クラスを作る。

（１）パラメータクラス

入力パラメータも出力パラメータもプロセスが処理する（できる）データ構造をしていなければならないわけで、プロセスの標準化に合わせて、データ構造も標準化していくべきものである。

パラメータ例：



（２）プロセスクラス

実際の処理を行うクラスで、実行時にオブジェクトを作り、平行処理で実現されるものである。

パラメータとの結合と並行処理の管理はノードクラスが担う。

パラメータの名前は、input と output、それと update の３種があって、入力データ、出力データ、更新データを表す。具体的には、

入力は、input1、input2、・・・input100、・・・

出力は、output1、output2、・・・output100、・・・

更新は、update1、update2、・・・update100、・・・

プロセスの外部とのデータのやりとりは、この３種のパラメータのみである。内部データの名前は自由であるし、Java のプリミティブやクラスを自由に使える。

（３）ノードクラス

ノードクラスは、プロセスクラスとプロセスクラスに渡すパラメータクラスを定義する。

並列処理を実現する、中核的な作業をするクラスである。ノードクラスは連携を取

っていて、並列処理のネストを管理している。並列処理の大本を停止させたときにその中で生成したすべての並行処理を停止させるこたができるようにしている。プロセスの起動パラメータの設定完了をもって、このノードに登録されているプロセスクラスのオブジェクトを作り、それにパラメータ群を渡して start させる。

フロー管理クラスがプロセスクラスを直接扱うことはしない。ノードクラスとの対話で処理を実現していく。

（4）フロー管理クラス

データフロープログラミングを実現しているクラスである。パラメータクラス、プロセスクラス、ノードクラスを利用して、並行処理による問題解決を実現するファッセドパターンを実現する。

2．2　文解析での動作例

和文解析を例に、データフロープログラミングを説明していく。処理としては、

（1）和文分かち書き化処理

文を、単語（連語）の列に分割していく処理である。

（2）単語の意味取得処理

単語の意味情報を辞書やプロセスなどから取得する処理である。

（3）単語の文法処理

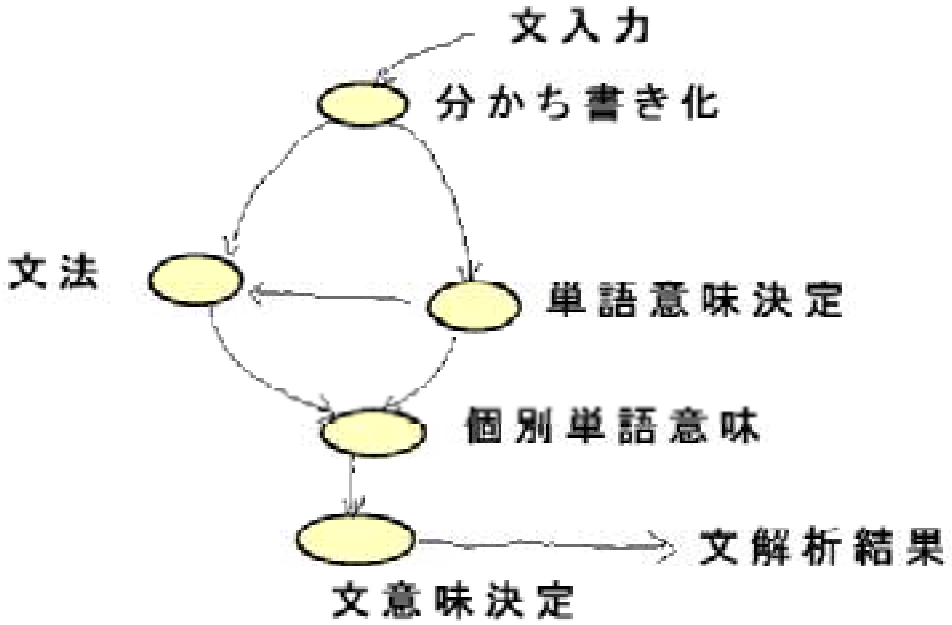
格の解析とか、係り受けの解析を行う。

（4）文の意味の同定。

解析結果を総合して、文の意味を決定する。

これらの処理は、曖昧性があり、各処理結果は複数である可能性もあるし、処理結果が相互に関連して、可能性を絞っていくのに利用されたりする。

曖昧性処理としては、各処理が１文毎に完結した複数の選択肢を生成していく方法と、和文分かち書き化処理で一単語切り出し、単語の意味処理をし、単語の文法処理をして、順次文全体の解析に至る方法がある。前者の方法は、複数の解釈がある場合、それら全部を選択肢として機械的に生成していき、最終的に、文の意味の同定で、有意味ののものになるものだけを生かすという適者生存を基本とする。後者の方法は、デフォルトの解釈をもって、順次、次段の処理に移っていき、問題があったとき、バックトラッキングを掛けるものである。



文法処理で、難しいのは、次の２つである。

（１）並置詞の解析

（例文）「私が学校へ行くこと、または、彼女が美容院にいくことは、重要な問題である」

通常、「または」とか、「と」、「、」などの並置詞は深刻な曖昧性問題を引き起こすし、文のどの部分が並置されるかということは、文構造の決定に大きく依存するため、文解析を非常に難しくする。

（２）助詞の解析。特に、「も」「は」「が」「を」「に」「と」「の」の処理である。

（例文）「私も果物はりんごが好きだ。」

「も」「は」「が」の各格がなにであるか、解析するのは難しい。単語の意味、助詞の配置、文全体の意味を得ないと格を決定できない。

このような解析プログラムを沢山用意し、プロセスクラスとしてライブラリを構成させる。プロセスを要素として、コンポーネントウェアの要領で、並行処理を実現させていく。

３．プログラム API

（１）パラメータの抽象クラス

DFparameter

DFparameter parm=DFparameter("パラメータ名")；　パラメータの定義
parm.setFocus("フォーカス名")　　；　パラメータの管理情報の設定

フォーカスはパラメータデータの出所を分かるようにするためのもので、複数のプロセスの解析結果を総合していくときに使う。

（２）プロセスの抽象クラス

DFprocess()

DFprocess process=DFprocess("プロセス名","プロセスのクラス名")

スレッドプログラミングで、停止フラグとか、パラメータの標準的なインターフェースを保持する。

（３）ノードの抽象クラス

DFnode()

DFnode node=DFnode("ノード名")　　；ノードの定義
node.addProcess(プロセス)　　；ノードにプロセスを登録する。
node.addInputParameter(パラメータ)　　；入力パラメータを登録する。
node.addOutputParameter（パラメータ）　；出力パラメータを登録する。
node.addUpdateParameter（パラメータ）　；更新パラメータを登録する。
node.start(ノード)　　；ノードに登録しているプロセス群を起動する。

ノードから起動したプロセス内で、さらにノードを起動することがある。そのネストを管理するために、start の引数にノードを指定する。

node.stop()　　；ノードから起動している全てのプロセスを停止する。
node.getActiveProcess()　　；実行中のプロセスを知る。

４．発展

データフロープログラミングの発展形を考えてみます。

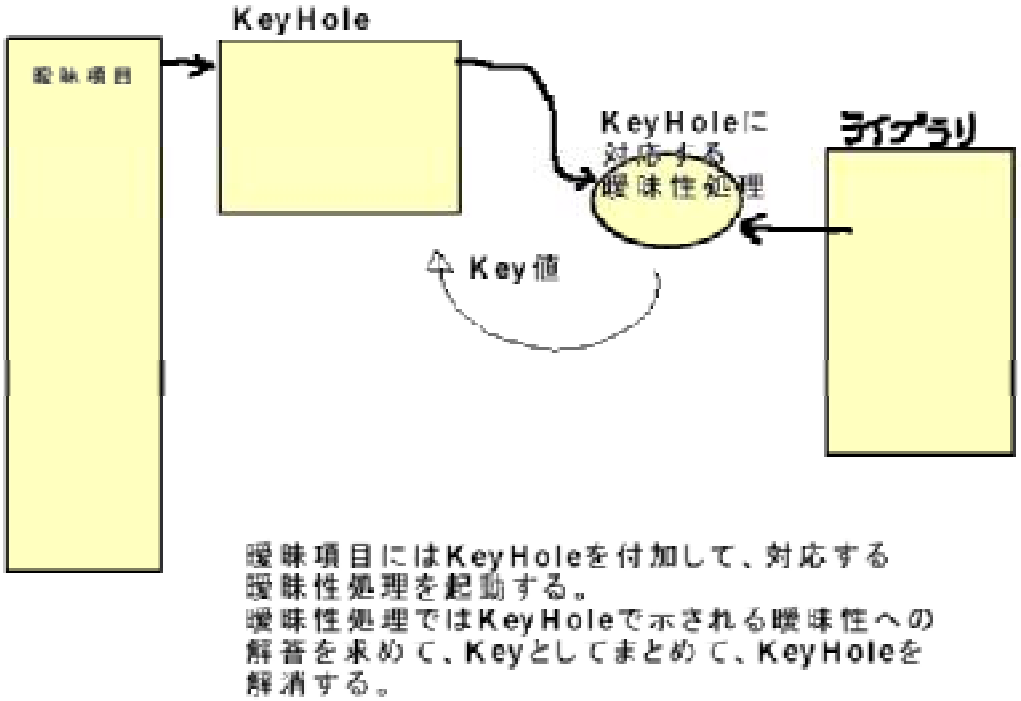
４．１　黒板システム

特定のパラメータを更新モードで共有することは、プログラミングをエレガントにする、一つの方策である。次々に文を解析して、意味を持ったフレーム構造を組み立てる処理を、フレーム構造のデータを黒板システムにしておくと、並行処理で、漸次曖昧性を低減していくということが簡単に行える。

黒板システムを利用するコンポーネントプロセスを統合して管理する制御システムを黒板にすることもできる。コンポーネントは黒板システムを利用して、全体の処理に影響を与え、全体からはパラメータの黒板システムによって制御される。こんな複雑系プロセッシングが実現できる。

４．２　KeyHole-Key システム

曖昧性をどう管理していくか。曖昧性にシステムで定めた識別子をつけて、専用の解析プロセスを実行させて曖昧性を低減していくのが良いようである。曖昧性識別子を管理するオブジェクトを KeyHoke とし。解決したデータを Key としよう。



４．３　学習システム

プリミティブなプロセスライブラリをデータフローで目的の大きな作業をするプロセスに組み立てていく。それは、フローを XML などの文書ファイルで構成し、コマンドとして実行していくようにすれば、Java でも自動プログラミングが可能になることを意味する。

５．まとめ

データフローに着目した、並列処理プログラミングを考察してきた。このような試みは、いかにプログラム構造をエレガントにするかという日々の努力の一里塚みたいなものである。最終的には学習機能をもったプログラムに至ると思う。その時、此処に述べてきたようなデータフローのようなプログラミングになるかどうかは、まだ分からない。でも、データフロープログラミングは今注目されているコンポーネントウェアによく似た手法なので、そんなに的はずれなアイデアではないと感じる。

学習システムでは、曖昧性解析の他に、カテゴリ分析、カテゴリ構築という、さらに大きな解析処理が必要になってくる。何らかの工夫が今後もなされていくことだろう。