PIMOSの例外処理方式

越村三幸[1]，佐藤裕幸[1]，近山隆[1]，藤瀬哲朗[2]，松尾正浩[2]
1: (財) 新世代コンピュータ技術開発機構 2: (株) 三菱総合研究所

1 はじめに

PIMOSは並列推論マシン用OSであり、ユーザ・プログラムを監視・制御しユーザの過ちからシステム全体、ひいてはユーザ自身を守ることを主な仕事としている。

PIMOSはFlat GHC(FGHC)を基本とするKL1で記述され、ユーザ・プログラム自身もKL1で記述される。FGHCでは全てのゴールが論理積関係にあり、一つのゴールの失敗は全体の失敗を引き起こす。このことは、ユーザの異常状態がシステム全体に伝播することを意味している。また全てのゴールは同一レベルにあり、あるゴールが他のゴールを安全に監視・制御するのが難しい。そこでゴール間に何らかの階層を導入する機能が必要となる。このためにKL1に導入した機能が荘園機能である。

荘園機能によって、監視・制御するプログラム(メタレベルプログラム)と監視・制御されるプログラム(オブジェクトレベルプログラム)を分離し階層化することができる。PIMOSはこの機能を利用してユーザレベルとOSレベルを分離し、ユーザプログラムを監視・制御している。

荘園機能を用いることにより、ユーザレベルの失敗をOSレベルに波及させないようにできる。また、異常状態からの復帰等も可能となる。本稿では、荘園機能を用いたPIMOSの例外処理方式について述べる。

2 荘園の機能

荘園は以下のようなプリミティブを用いて生成する。

```
execute(Goal, Control, Report, Tag)
```

ここで、各引数は以下のようなものである。

Goal: 荘園の中で実行すべきゴール列。荘園の中で荘園を作ることも可能である。

Control: 制御ストリーム。荘園内の実行を制御するためのコマンド(実行開始,実行中断,実行放棄等)を荘園の外部から送るのに用いる。

Report: 報告ストリーム。荘園内の実行の結果によるさまざまな情報(実行放棄,例外発生等)を、荘園外部に伝達するのに用いる。

Tag: 荘園がネストしている場合、それらを識別するためのタグ。

上述の方法で生成した荘園内のゴール群は、荘園外とは独立した論理積を成す。すなわち、荘園内での失敗は荘園内に閉じたものであり、荘園外のゴールを巻き添えにすることはない。

3 荘園の例外処理機能

3.1 例外の報告

荘園内の実行中に例外(算術演算エラー,実行の失敗)が生じると、以下のようなメッセージが報告ストリームに流れる。

```
exception(Info, Goal, NewGoal)
```

ここで各引数の意味は以下の通り。

Info: どのような例外が生じたかについての情報。これに基づいてメタプログラムはどのような処理を行うか判断する。変数を含んでいるとメタプログラムがデッドロックする可能性があるので基底項であることが保証されている。

Goal: 例外を起こしたゴールの情報。これに対して例外処理の操作を行う。ただし、これは任意項なので操作自体は NewGoal で指定し例外を起こした荘園内で実行しないと、メタレベル自身が例外を起こしてしまう。

NewGoal: 例外処理として、例外を起こしたゴールの代わりにオブジェクトレベルで実行すべきゴールを指定するための変数。具体化されていないことが保証されている。

例外の原因別に適当なタグを割当てる。例外の報告は、例外を起こした荘園,その親荘園,そのまた親,という先祖荘園の中で、例外原因タグにマッチするタグを持つ最も近い先祖荘園の報告ストリームにされる。こうすることによって、特定の例外だけを取り扱い、他の例外はより外側の荘園に任せるような記述が可能になる。

3.2 積極的な例外の生起

オブジェクトレベルで積極的に例外を起こすことができるように、以下のような組込み述語用意されている。

```
raise(Info, Data, Tag)
```

ここで、各引数は以下のようなものである。

Info: 例外に関する情報。この引数が基底項に具体化されるまで例外の生起は遅延される。

Data: 任意のデータ。この引数は具体化されていてもいなくてもよい。メタプログラムが中身を読まなくても良いデータを渡すのに用いる。

Tag: 例外のタグを指定する。

この述語が実行されると、Tagで示される先祖荘園の報告ストリームに例外の発生が通知される。述語raiseによる例外を特にソフトウェア定義例外と呼ぶ。このとき3.1「例外の報告」で示される例外情報のInfoは述語raiseの引数Info、Goalは引数Dataに対応している。

述語raiseはオブジェクトプログラムとメタプログラムの通信に用いることもできる。PIMOSにおいて、ユーザはソフトウェア定義例外を用いてPIMOSに様々な要求をする。(5「PIMOSの例外処理」)

3.3 例外後の実行の継続

荘園内では例外を起こしたゴールの実行は中断され、NewGoalの具体化を待って、元のゴールの代わりにNewGoalを実行する。この際、荘園にはNewGoalが具体化されたらそれを実行しようとするゴールがあると考えて良い。このゴールがあるため、NewGoalを具体化しないうちに荘園全体が先に成功裏に終了してしまうことはない。

"Exception Handling in PIMOS"
Miyuki KOSHIMURA[1], Hiroyuki SATO[1], Takashi CHIKAYAMA[1], Tetsuro FUJISE[2], Masahiro MATSUO[2]
1: Institute for New Generation Computer Technology 2: Mitsubishi Research Institute

4 例外処理プログラム例

簡単な例外処理プログラムは、オブジェクトレベルで例外が発生した場合、オブジェクトレベルを放棄するものである。これは、次のように呼びだされる。

```
..., execute(UserGoal, Cont, Res, 全部),
        exception_handler(Res, Cont), ...
```

exception_handler の定義は次の通り。

```
exception_handler([Msg|Res], Cont) :-
    Msg = exception(Info, Goal, NewGoal) |
        Cont = [実行放棄].
```

例外を起こしたゴールを取り敢えず成功させて、実行を継続したければ次のようにする。

```
exception_handler([Msg|Res], Cont) :-
    Msg = exception(Info, Goal, NewGoal) |
        NewGoal = true,
        exception_handler(Res, Cont).
```

算術演算時にアンダフローが発生したときに、そこをゼロで置き換える場合は次のようにする。

```
exception_handler([Msg|Res], Cont) :-
    Msg = exception(Info, Goal, NewGoal),
    Info = under_flow |
        NewGoal = (Goal = (X1 := Y1), X1 = 0),
        exception_handler(Res, Cont).
```

ここで、アンダフローを発生させたユーザ・ゴールは $X := 10^{-10} \times 10^{-10}$ 、メッセージ中の Info は under_flow 、 Goal はこのゴール $X := 10^{-10} \times 10^{-10}$ であるとする。 Goal は変数を含む項なので、それの分解(上の例では Goal = (X1 := Y1)) や unification (X1 = 0) の操作は NewGoal に指定してオブジェクトレベルで行う必要がある。メタレベルで行うとメタレベルで失敗,デッドロックが発生する場合がある。

5 PIMOS の例外処理

PIMOS が扱う例外には、オーバフローやゼロ除算などの言語処理系が検出する言語定義例外と、ソフトウェア(主に PIMOS)で検出するソフトウェア定義例外がある。後者は主にユーザとシステムとの通信に用いられ、さらにタスク定義例外とシェル定義例外に分けられる。これらの例外は、それぞれ処理する内容が異なっているため、例外を受け取る荘園のタグを変えることにより、別々の場所(荘園)で受け取られ処理される。

5.1 タスク定義例外

ユーザが PIMOS に依頼した仕事は、「タスク」と呼ばれる単位で管理されている。タスクは荘園機能を用いて実現され、 PIMOS の資源管理の単位である。 PIMOS が管理する資源にはリダクション数等の KL1 で定義されている言語定義資源と入出力装置等の OS で定義されている OS 資源がある。

ユーザの OS 資源の利用は、PIMOS に対して通信ストリームを介して要求メッセージを送ることで行われる。従って、ユーザは、PIMOS と通信を行うためのストリームを獲得しなければならない。PIMOS との通信を行うためのこのストリームを「ジェネラル・リクエスト・ストリーム」と呼んでいる。ジェネラル・リクエスト・ストリームは、ストリーム獲得例外を発生することにより、獲得することができる。この例外を PIMOS 定義例外という。

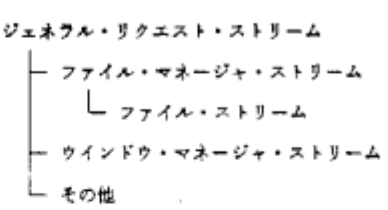

図 1: 階層ストリームの構成

ユーザは、ジェネラル・リクエスト・ストリームを獲得するために、以下のような述語を呼び出す。

```
raise(ストリーム獲得, Str, task)
```

各タスクを管理している task_handler は、次のようになる。

```
task_handler([Msg|Res], Cont, GRS) :-
    Msg = exception(Info, Str, NewGoal),
    Info = task(ストリーム獲得) |
        NewGoal = pimos_protector(Str, GRS1),
        merge(GRS1, GRS2, GRS),
        task_handler(Res, Cont, GRS2).
```

ここで、 GRS は各管理プログラムが保持しているジェネラル・リクエスト・ストリームの口である。pimos_protector はユーザの悪意を持ったメッセージから PIMOS を保護するためのものである。これは次のように呼ばれる。

```
..., execute(UserGoal, Cont, Res, task),
        task_handler(Res, Cont, GRS), ...
```

実際にウインドウを生成したり、ウインドウに対して入出力操作(getc, putc 等)を行うためのストリームは、例外ではなく、ジェネラル・リクエスト・ストリームに獲得メッセージを送ることで獲得する。

例えば、あるファイルをオープンしたい時は次のようにする。先ず、ジェネラル・リクエスト・ストリームを獲得し、そこへファイル・システムへのストリーム獲得要求メッセージを送る。そして得られたストリームに対してファイル・オープン要求を送る。こうして、ユーザは実際に入出力操作を行えるストリームを獲得することができる。

このように、ジェネラル・リクエスト・ストリームを核として、OS 資源を確保することができる (図1)。資源を階層化することにより、ユーザからの要求を分散して処理することができる。

5.2 シェル定義例外

5.1「タスク定義例外」で述べた機構により、メタレベルはオブジェクトレベルの要求に応じて様々なサービスを提供できる。PIMOS が提供するシェルの標準入出力獲得やシェル変数の参照・設定といった機能もこの機構を利用して提供される。このシェルとのストリームを獲得する例外をシェル定義例外と呼んでいる。

6 おわりに

平板な FGHC に荘園機能を導入することでメタレベルとオブジェクトレベルを分離できるようになる。そしてこの機能を利用することで PIMOS はユーザ・プログラムを安全に監視・制御することができ、ユーザの異常状態の伝播を抑制できる。またソフトウェア定義例外を利用することにより、ユーザは PIMOS に対して様々な要求を行える。

今後、これらの開発をマルチ PSI 第2版上で行なっていく。