属性文法を基にした制御系向け階層型プログラミング(３)[1]

2D－2

田中裕之　白倉隆雄　中川裕之[2]

キヤノンソフトウェア株式会社[3]システム研究所

1 はじめに

ソフトウェアに対する要求の正当性や実現可能性を高める仕様記述手法のひとつとして操作的手法がある。操作的手法に基づく仕様記述については、これまでも数多くの提案がなされてきたが、これらは我々が対象とするFA(Factory Automation)システム、即ち、制御対象がどのような状態にあるかを判別しその状態の組み合わせにより次の動作を決定するようなシステムの仕様を記述するには、十分満足できるものとは言い難かった。そこで、我々は状態をモジュール分割条件とした属性文法計算モデルに基づいた仕様記述言語を提案し[1]、その有効性を確認してきた。

通常処理の記述に関してはその効果が十分確認できたものの、制御システムの特徴である例外処理、中でも、ハードウェア異常の検知に用いられるタイマ割り込みによるタイムアウト処理については、うまく記述できないという問題があった。つまり、タイムアウト処理の記述については、タイマ割り込みが発生した場所、即ち、通常処理を記述する場所とは異なったところに記述しなければならずシステム全体の可読性を低下させていた。

実現しようとする機能を主体に考えた場合、システム仕様としては、例外処理と通常処理は区別されることなく対比的に並べて記述できたほうが、システム全体の見通しはよい。そこで、[1]の枠組み、即ち、状態の組み合わせに対して次の動作を決定する手法の中で、タイムアウトに関する記法を付加し、制御系向け仕様記述言語の拡張を行った。

2 記述方法

2.1 通常処理の記述

[1]で提案した記述方法について簡単に説明を行う。我々はシステムを状態遷移中心にとらえ、生成規則の左辺を非終端記号(いわゆるモジュール)、右辺を非終端記号または状態遷移を引き起こすモジュールで構成する。制御対象の状態に応じた動作記述のために、状態のパターンマッチ機能を用意し、[]内にマッチさせる状態名を記述する。各モジュールは属性文法と同様に合成属性と継承属性の２種類の属性を持つことが出来る。その評価規則は{}内に記述し、構成要素に付随する属性と下請関数を用いて記述する。

```
Restration ::=  [SELECT] down Release
    {
        Release.inh = down.sym;
    }
          | [FREE]
          ;
Release ::= release up
    {
        Release.sym = up.sym;
        release.inh = Release.inh;
    }
    ;
```

例えば上の記述において、Restrationという動作は、状態のパターンマッチによる分岐記述を行っておりSELECTという状態のときにdownとReleaseに分割され、Releaseはreleaseとupに分割される。Releaseを分割する際の属性評価規則は、upで生起した(合成)属性をReleaseで生起する属性とし、releaseに引き渡す(継承)属性にReleaseが受け取った(継承)属性をコピーすることを表している。

---

1 Hierarchical Programming Based on Attribute Grammer for RealTime System

2 Hiroyuki Tanaka,Takao Shirakura,Hiroyuki Nakagawa

3 Canon Software inc.,3-9-7 Mita,Minato-ku,Tokyo,Japan

2.2 例外処理の記述方法

我々はタイムアウト処理が必要となる状態を「モジュールが終了しない」状態と捉える。この概念は動的論理[2]の中に見ることができ、プログラムpが終了しないという直観的な意味を持つ論理式は[p]falseと表現できる。

動的論理において[p]ϕという論理式は「プログラムpが終了するならば論理式ϕを満たす状態で終了する」という意味を持っているが，我々は「モジュールpを実行し、その実行が終了したならばϕをみたす状態である」と解釈する。また、動的論理において[p]ϕの否定は$\neg$([p]ϕ)，即ち<p>$\neg\phi$である。<p>ϕという論理式は「プログラムpの少なくともひとつの実行列が論理式ϕを満たす状態で終了する」という意味を持っているが，我々はこれを「モジュールpを実行し、論理式ϕを満たす状態で終了した状態」と解釈する。動的論理においてはpというプログラムの持っている性質に注目しているが、我々は、その式が評価されるそのときのpの実行により生ずる状態に注目している。

そこで我々はモジュールpの実行中にタイマ割り込みによりタイムアウトを起こすような状態の表現として[p]falseという記述を採用し、<p>trueをタイムアウトを起こさない状態として記述する。この2つの状態によって次の動作を決定するという記述は、まさに、タイムアウト処理の記述と一致することになり、前述の状態のパターンマッチによる分岐記述の枠組みに一致するので、従来の概念を変えることなく記述することができる。

構文としては、パターンマッチに使われる状態として、[モジュール名]false、<モジュール名>trueという２つの状態を[1]に追加する。タイムアウト処理が必要な状態か、そうでない状態かの分岐により、タイムアウト後の処理とタイムアウトしない後の処理とを対比的に並べて書くことができる。

```
X ::= [ <a>true  ] b c
    | [ [a]false ] q;
```

例えば、上記の記述はモジュールaを制限時間(後述)付きで実行し、制限時間内に終了したら続けてb、cを実行し、制限時間内で終了しない場合には(aを強制的に終了し)続けてqを実行することを示す。

またタイムアウト処理には制限時間という概念があるので、タイムアウト処理を必要とするモジュールに対してのみ、timeoutという特別な(継承)属性を付加し、制限時間を与えるものとする。

3 記述例

ロボットハンド入れ替えシステムのハンド交換記述の一部（ハンド返還部）を例にとる。

ハンド返還(Restration)の手順はセレクト状態(SELECT)ならば、アームをスピード5で下ろし(down)後述のReleaseを行う。非セレクト状態(FREE)のときは何もしない。Releaseの手順は制限時間付きでハンドを解放(release)する。制限時間内に終了したらアームをスピード10で上げる(up)。制限時間内に終了しない場合にはハンド解放を中止し警告(alert)を出す。この記述例は以下の様になる.

```
Restration ::= [SELECT] down Release
    {
        down.speed = 5;
    }
            | [FREE]
            ;
Release ::= [ <release>true  ] up
    {
        release.timeout = 10;
        up.speed = 10;
    }
          | [ [release]false ] alert
          ;
```

4 まとめ

例外処理と通常処理を区別することなく対比的に並べて記述することが可能となり、システム仕様の可読性を向上させることができた。今後の課題として、タイムアウト処理以外の例外処理、データ、動的プロセス、プロセス間同期(通信)を記述可能としたい。

参考文献

[1]白倉・冨樫・中川:属性文法を基にした制御系向け階層型プログラミング(1). 情報処理学会第51回全国大会講演論文集、1995

[2]Fischer,M.J. and R.E.Ladner,Propositional Dynamic Logic of regular programs,*J.Comput. System Sci.* 18(2) ,(1979),194-211.