柔軟な仕様生成のための協調モジュール演算機能*

6V－2

蓬莱 尚幸†
新ソフトウェア構造化モデル研究本部
情報処理振興事業協会 (IPA)

1. はじめに

ソフトウェア開発における仕様プロセス、特に、その前プロセスである要求プロセスの出力(要求)をもとにどのように形式的仕様を記述してゆくかという「仕様生成」について、要求プロセスの特徴を考慮に入れ、要求の追加／変更などに柔軟に対応しながら形式的仕様記述を得るための方法論の提案に向けて研究を進めてきた。これまでに、既に記述した仕様に対する追加 / 変更 / 削除などを必要とするような仕様生成の様々な場面で、どのような「柔軟な」対応が必要であるか、について考察してきた [3]。本稿では、これらの考察を踏まえつつ、柔軟な仕様生成を行なうために必要な仕様記述言語の新しい機能を提案する。

2. 仕様生成における課題

要求プロセスが矛盾を含むバラバラの要求から矛盾のない統一的な要求を作り出すプロセスであるのに対して、仕様プロセスは要求プロセスで作り出された要求を仕様書としてきっちりと定義してゆくプロセスである。要求は断片的に生成される。そこで、仕様プロセスでは、要求が生成されるごとに、それを段階的に仕様化してゆく機能が必要となる。この機能を「仕様生成」と呼ぶ。

要求は、断片的に生成されるのみならず、後になって変更されることもある。つまり、要求はある時点でみると整合性がとれているが、時系列的にみると守備一貫しない。そこで、常に最新の状態に仕様を保つためには、仕様生成のステップ間の整合性の保証を意図したいわゆる段階的詳細化 [1] では要求の時間的推移に追従できないことがあると考えられる。

我々は、仕様生成の各ステップを、既存の仕様 (一つ前のバージョン) に対する追加 / 変更 / 削除も行ないながら、既存の仕様と新たな要求から得た断片的な仕様を入力とし新たなバージョンを出力とする既存の仕様の変更も行なうような演算と捉える。この演算を指定するための枠組、および、入力となるモジュール群がその指定をもとに矛盾の解消などを協調して行なう機構を仕様記述言語に与えることにより、柔軟な仕様生成を行なうための基礎とする。なお、断片的な仕様を要求からどのように得るかは重要な問題であるが、本稿の議論の範囲外である。

*Module Cooperation Mechanism for Flexible Specification Generation, Hisayuki Horai, Laboratory for New Software Archtectures, Information-technology Promotion Agency, Japan (IPA)

†(株) 富士通研究所 より出向

3. 本稿で用いる仕様記述言語

柔軟な仕様生成のための協調モジュール演算機能は、多くの仕様記述言語に付加可能な一般的な機能である。協調モジュール演算機能のために必要な言語拡張は、1. 矛盾することもありうる複数の仕様 (モジュールの集合) を扱う枠組、2. 新しい仕様の生成方法 (演算) を指定するための枠組、である。仮に、本稿では OBJ 言語 [2] を簡単化した代数的仕様記述言語を例にとり、協調モジュール演算機能について説明する。

3.1. モジュールと仕様

モジュールとしては、関数モジュール (fmod) とソートモジュール (smod) の二つを用意する。関数モジュールの定義では、定義する関数名とシグニチャと定義 (axiom) を指定する。また、ソートモジュールの定義では、定義するソート名と定義 (スーパーソート名またはリストや列挙型などのソートコンストラクタ) を指定する。この言語では、これらのモジュールの集合としての仕様 (spec) を扱う。仕様は名前付けられ、仕様生成の各ステップで複数の仕様を扱えるようにする。図 1は仕様およびモジュールの抽象構文である。定義の途中のモジュールを扱えるように、axiom とスーパーソートは省略可能となっている。

```
<spec> ::= <名前> { <fmod> | <smod> }*
<fmod> ::= <名前> <入力ソート>* <出力ソート> <axiom>*
<smod> ::= <名前> [ <ソートの定義> ]
```

図 1: モジュールと仕様

3.2. 仕様生成機能 recipe

仕様生成 (recipe) 機能は、複数の材料をもとに一つのモジュール群 (仕様) を作る機能である。図 2はその抽象構文である。recipe 機能では、まず始めに空の仕様を用意し、材料に指定されている仕様やモジュールや axiom を 1 つずつ加えてゆく。その際、ソートの間の関係の矛盾や axiom の衝突が発見されると、新たに加える材料が正しいとして、生成中の仕様内に矛盾がなくなるように、既存の部分を自動的に変更する。

各材料とともに加工方法 (method) を指定することで、その材料を加える時に、それ自身を変形したり矛盾解消の方法を変更することができる。以下に、これまでに考

```
<recipe> ::= <仕様名> { <材料> [ <method> ] }*
<材料>* ::= { <仕様> | <モジュール> | <axiom> }
```

図 2: 仕様生成機能

案した代表的な加工方法を挙げる。

名前の変更: モジュールを加える際に名前を変更するための method(>>) は、詳細化や一般化を行なうときに便利である (後述)。例えば、fmod m1 >> m2, m3 ; は、モジュール m1 の名前を m3 に変え、モジュール m1 とシグニチャが等しく axiom がないモジュール m2 を新しく作る。

データ構造分解: 例えば、decomposition(list) は、関数モジュールの引数について「リスト型 vs その要素型」という矛盾がある場合に、リストを分解し要素に関数を適応した結果をリスト化するように関数を変更することで矛盾解消を行なわさせるための method である。もし指定しないときに同様の矛盾があれば、どちらのソートが変更に由来するかに従って型を一致させるように仕様を変更する。

場合分けの追加: 新たに加える axiom が既存のものと矛盾しないことを示す method (another_case) により、新たな場合分けを追加する。これが指定されていないときは、axiom の変更とみなし、既存のものは捨て去られる。

例外の追加: special_case(form) は、新たに加える axiom A との矛盾を解消するために、form を含む既存の axiom の条件に not(A) を加えることを指示する method である。form を省略すると、既存の全ての axiom の条件に not(A) が加わる。

4. 記述例

4.1. 引数の変更の伝播

図3の左上は、関数 F が関数 G で詳細化されているが、G はまだ詳細化が行なわれていないモジュールである。今、G の詳細化を行なう作業中に G には引数として x 以外に a も必要であることがわかったとする。このような仕様 X の変更を行なうためには、図3の右上の recipe の 2) のように G を「上書き」すればよい。すると、F と G のシグニチャは矛盾を起こすので、自動的に F に新たな引数が加えられ、X は図3の下のようになる。さらに、引数の変更はモジュールの参照関係を通して関連する関数モジュールに伝播してゆく。このように、協調モジュール演算を用いると、引数の数を変えるような高度な変更も用意に行なうことができる。

```
spec X is                  | recipe X is
  fmod F : x -> y is       |   ingredients
    axiom F(X) = G(X);     |     1) spec X;
  end;                     |     2) fmod G : x, a -> y;
  fmod G : x -> y;         | end;
end;                       |
-----------------------------------------------------
spec X is
  from F : x, a -> y is
    axiom F(X,A) = G(X,A);
  end;
  fmod G : x, a -> y;
end;
```

図 3: 引数の変更の伝播

```
recipe X is                | recipe X is
  ingredients              |   ingredients
    1) spec X;             |     1) spec X;
    2) spec ■;             |     2) axiom F(X)=F1(X);
    3) axiom G(X)=G1(X);   |   method
end;                       |     1) fmod F >> F, F1;
                           | end
```

図 4: 詳細化と一般化

4.2. 詳細化と一般化

図3左上の関数 G を関数 G1 で詳細化するためには、図4の左のように、新しい概念である G1 を仕様化した仕様 N と G と G1 の関係を示す axiom を次々と加えるだけでよい。また、図3左上の関数 F を一般化するためには、図4の右のように、既存の F を F1 という名前 (新しい状況では別名で呼ばれるはずである) に変えた後、一般化された (新しい)F と F1 の関係を与えればよい。さらに decompose や another_case などの method を指定することで、より複雑な詳細化 / 一般化が行なえる。

5. おわりに

本稿では、代数的仕様記述を例にとり、柔軟な仕様生成のために仕様記述言語に付加すべき協調モジュール演算機能について述べた。例に用いたシグニチャの矛盾を利用した引数の変更の伝播は仕様変更に対するモジュール間協調の一つである。

現在、JR の運賃計算 (歴史的に何度も変更され、多くの例外を含む) を例にとり、仕様生成実験を行なっている。また、代数的仕様記述言語への協調モジュール演算機能の実現を検討している。今後の研究課題としては、言語の実現、加工方法指定の (半) 自動化やガイドシステム、加工方法の定義のユーザ解放 (現在はシステム定義) などが挙げられる。

謝辞

本研究は、産業科学技術研究開発制度「新ソフトウェア構造化モデルの研究開発」の一環として情報処理振興事業協会 (IPA) が新エネルギー・産業技術総合開発機構から委託をうけて実施したものである。

参考文献

[1] E. W. Dijkstra. "Selected Writings on Computing: A personal Perspective", Springer-Verlag, 1982.

[2] J. A. Goguen, T. Winkler. "Introducing OBJ3", Tech. Rep. SRI-CSL-88-9, SRI International, 1988.

[3] 蓬莱尚幸, 「形式的仕様記述プロセスにおけるモジュール協調」, SE-96, 情報処理学会ソフトウェア工学研究会, 1994.