Flageアーキテクチャのための仕様合成メカニズム*

4N－8

蓬莱 尚幸†　本位田 真一‡
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

変更に柔軟に対応するソフトウェア開発技術の確立がFlageアーキテクチャ[2]の目的である。本稿では、まず要求の追加/変更/削除に対応しながら仕様記述を柔軟に合成してゆくための言語機能[1]を概説した後に、エージェント指向のソフトウェアアーキテクチャFlageのための仕様合成メカニズムについて考察する。

我々は、ソフトウェア開発における仕様プロセスにおいて、要求プロセスの出力(要求)をもとに形式的仕様を記述してゆく機能を「仕様生成」と呼ぶ。一般的に、要求からの仕様の記述をするためには、1.要求からの仕様化、2.既存の仕様への変更の伝播、の2つの活動が必要である。仕様生成は後者の活動であり、それを支援するための言語機能を提供することが本研究の目的である。

2章では、既に提案した仕様合成のための柔軟なモジュール演算について述べる。3章と4章では、Flageのための仕様合成メカニズムの異なる形態を提案する。

2. 仕様合成と柔軟なモジュール演算

仕様合成のために必要な言語機能は、高度なモジュール化と合成方法の指示である。

2.1. 高度なモジュール化

仕様合成では要求変更の前後の仕様を同時に扱う必要がある。そこで、互いに矛盾することがありうる複数の仕様を扱うことができるモジュール化が必要である。我々は、代数仕様を例にとり、仕様を関数モジュールとソートモジュールを複数含む集合と捉え、抽象構文が図1で示される言語$\mathcal{F}$を設計した。

```
<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: 言語$\mathcal{F}$における仕様とモジュール

2.2. 合成方法の指示

仕様合成の指示には、その仕様合成で参照したり変更したりする材料(複数)と合成方法(各材料毎)と新しく生成される仕様名の指定が必要となる。我々は仕様合成の指示として$\mathcal{F}$にrecipe機能を追加した。recipe機能の抽象構文を図2に示す。

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

図2: recipe機能

材料として仕様単位だけでなくより小さい単位で指定できることで、より柔軟に要求変更に対応できるようにした。すなわち、言語$\mathcal{F}$における仕様(<spec>)の他にモジュール(<fmod>と<smod>)や公理(<axiom>)も材料として指定できる。

合成方法の指示(<method>)としては、単純な上書き、名前の付け替え、場合の追加、特例の追加、データ構造による分解、自由変数の導入、変更伝播の制限などの柔軟なモジュール演算を用意した。これらのモジュール演算では副作用を積極的に利用することで既存の仕様も変更できるようになっている。これらを指定することで、詳細化や一般化、例外処理、引数の数の変更などを行なうことができる。一般的に仕様合成では後から現れた新しい要求を反映した材料を優先することになるので、合成方法の省略時のデフォルト値を単純な上書きとした。

2.3. recipe機能の評価システム

現在、我々はrecipe機能の評価システムchefを開発中である。recipe記述を1個与えられたchefは、まず空の仕様をシステム内部に用意する。次に、指定されている材料を1つずつ環境から取り込みその材料と対応した合成方法を用いてシステム内部の仕様と合成する。最終的にすべての材料を合成し終ると、システム内部の仕様にrecipeで指定された仕様名を付けて環境に書き込む。その際、もし同名の仕様が既に環境内に存在すれば、その仕様を新しいものに置き換えることになる。

chefではシステム内部の仕様と各材料を合成する際に、変更が複数箇所に伝播するメカニズムを持っている。たとえば、ある関数モジュールの引数の数が変更されると、その関数モジュールだけでなくそれを呼び出しているモジュールにも新しい引数が導入される。このような変更の伝播を行ないながら材料を1つずつ合成してゆくために、chefは図3のような二重ループ構造をとっている。

3. Flageによる仕様生成支援環境

chefシステムでは仕様をシステムの外にある環境に蓄えている(具体的には仕様毎にファイルに分かれている)。chefはPrologを用いて開発しているが、将来的に

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

†(株)富士通研究所より出向

‡現(株)東芝

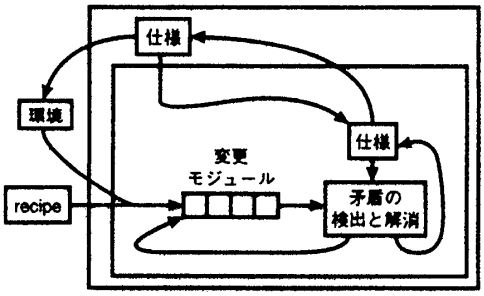

図 3: 評価システム chef

Flage を用いることで仕様を蓄える環境も統一的に扱うことができる。

図4のように、場がモジュール群を蓄える場所であり、エージェントが合成方法の指示を受けとり場を渡り歩いて材料を集めて新しいモジュール群を合成して新規の場を作成するようにモデル化することで Flage を用いて簡単に仕様生成支援環境が構築できる。その他に、要求を形式化した断片的な仕様に基づいて場を作るエージェントやモジュールやその履歴をブラウジングするエージェントも作ることができる。さらに、対象の言語が直接実行可能ならば、その実行エンジンをエージェント化することも可能である。

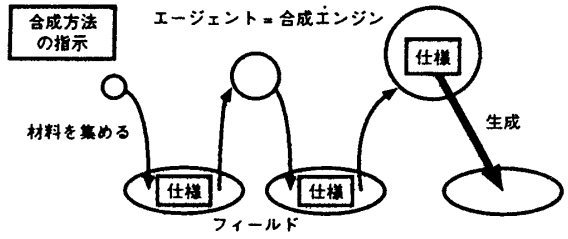

図 4: 仕様生成支援環境

場に格納される仕様記述と Flage 言語は独立である。そこで、任意の仕様記述言語に対して仕様合成機能 (高度なモジュール化と合成方法の指示) を導入すれば、その言語のための仕様生成支援環境を構築することができる。例えば、chef をエージェントの機能に組み込むことで、$\mathcal{F}$ に関する仕様生成支援環境が構築できる。このように Flage アーキテクチャは仕様生成支援環境として利用することができる。

4. Flage の仕様合成メカニズム

次に、対象の仕様記述言語を Flage 言語にした場合について考える。そこでは仕様が場とエージェントに分散されて蓄えられることになる。仕様生成を行なうエージェントは、仕様が追加 / 変更 / 削除された時に、能動的に既存の場やエージェントに働きかけてそれらの追加 / 変更 / 削除するように振舞うことになる。その際、合成指示は最初からエージェントに与えられるのではなくて、必要に応じてユーザから対話的に入手するようにできる。

また、図5のように各合成方法の定義を場に蓄えておくことにより、合成方法の指定やその評価方法を変更可能にできる。すなわち、場には対象システムの仕様を蓄える仕様場と仕様生成の評価のためのメソッドを蓄えるメソッド場が存在し、仕様生成エージェントは仕様場とメソッド場を重ね合わせながら仕様生成を行なう。

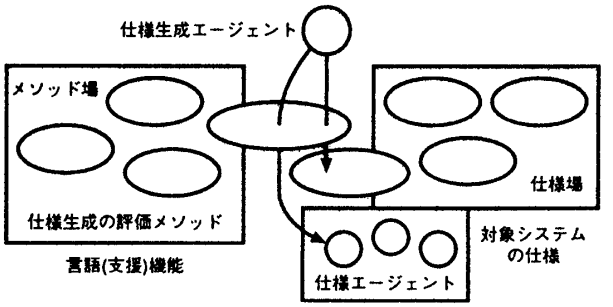

図 5: Flage 仕様合成メカニズム

同様に、検証機能や直接実行機能を持つエージェントも変更可能にできる。このように、Flage アーキテクチャ内のツールは、「言語 (支援) 機能」を蓄えたメソッド場と操作対象の仕様場を重ねて利用するエージェント群として統合できると考えられる。対象言語を Flage 言語とすることにより、前節で述べた仕様生成支援環境を越える支援が可能となる。

5. おわりに

本稿では、要求の変化に対応する仕様合成を支援するための言語機能である柔軟なモジュール演算について述べ、エージェントと場の概念に基づくソフトウェアアーキテクチャ Flage のための 2 種類の仕様合成機能について考察した。一方は様々な仕様記述言語に対して適用可能な支援環境としての Flage である。そこではエージェントと場の概念を用いることで、支援系の設計が容易となると考えられる。他方は Flage 自身に仕様合成メカニズムを導入するものである。そこでは仕様記述言語と仕様合成などの言語支援機能と支援ツール群を同一のフレームワークで保守できる。これを実現するためには、Flage 言語に対する仕様合成機能の研究が今後の課題である。

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] 蓬莱, 「柔軟な仕様生成のための協調モジュール演算機能」, 第 49 回情報処理学会全国大会, 1994.

[2] 粂野ら, 「Flage アーキテクチャの構想」, 第 51 回情報処理学会全国大会, 1995.