状態遷移をベースとした部品合成システムの試作

7L-2

加地　浩一　岸　美保子　松村　一夫
株式会社　東芝　システム・ソフトウェア技術研究所

1．はじめに

ソフトウェア生産の工業化を目指すＩＭＡＰシステム[1] (Integrated software MAnagement and Production support system) の一環として、ソフトウェアの生産性と信頼性の向上を目的に、部品合成の研究を進めている。そのため我々は、生産現場のアプリケーション・ソフトウェアを分析したところ、状態遷移による設計の枠組みが有効であることが得られた[2]。この経験をもとにして、ユーザ（開発者）とシステムとの会話をベースとした協調支援型の部品合成システムを試作したので報告する。

2．部品化方式

状態遷移は、システムの内部状態のモデル化に適しており、プロセスの制御方法やマンマシン系の操作仕様を規定する場合などに有効である[2]。状態遷移図は、図1の設計観点1のように、状態（円）、状態の遷移（矢印）、イベント、アクションの記述からなる（詳細は後述）。ここでイベントとアクション部については汎用性のある機能単位として部品化が可能であると考えられる。

我々は、設計標準化・部品標準化を部品合成の前提として考えている。設計標準化の枠組みとして状態遷移による方法を、また部品化の枠組みとしてデータ抽象化による方法を採用した。状態遷移の分析により得られるイベントとアクション部については、汎用性・再利用性を考慮しデータ抽象化による部品化の検討を行う。つまり、共通データの抽出を行い、それに対し関連性の高い操作の一体化を行う（以下、データ抽象化により部品化を行い、データと操作をひとまとまりの単位としてまとめたものをパッケージと呼ぶことにする）。パッケージの作成方針としては、関連性の高いデータを集め、それらのデータを操作する処理部品群を集める。つまり、パッケージ内においては凝集性を高める。しかしパッケージ間においては、パッケージの可搬性や保守容易性を考慮し独立性を高めることを基準とする。

上記部品化方式によるパッケージ化と対象分野の範囲を制限して運用するということを前提とした、部品合成のモデルを定義する。

3．部品合成のモデル

次の２つの設計観点により部品合成を考える（図1）。

設計観点1．状態遷移による動作仕様の記述。
設計観点2．パッケージの相互作用による具体化。

設計観点1は、状態遷移により対象システム内部の動きを規定するもので、ユーザが最初に記述し入力を行う。

Development of A Program Synthesis System Based on The State-Transition
Koichi Kaji, Mihoko Kishi and Kazuo Matsumura
Systems & Software Engineering Lab., TOSHIBA Corp.

前記の通り、状態遷移はシステムの内部状態のモデル化に有効で、動作仕様の厳密な記述を与えることができる。この入力に対し設計観点2によりシステムは、部品検索戦略やパッケージの知識（ルール）を用いてイベントやアクション部への部品の検索や組み込みの支援を行う。この際、適宜合成過程情報を提示し、ユーザの判断を仰ぎながら進める。知識は、例えば部品間インタフェースの整合性を検証するものである。またパッケージの凝集性を利用した部品検索は、例えば図1の例では、a2の操作部品の候補としてOP13とOP22が競合している状況を示しているが、パッケージの参照回数が多いものを優先する（この場合はOP13が選択される）などの戦略を用いる。設計観点1と2を繰り返し、徐々に詳細化して行くと考える。

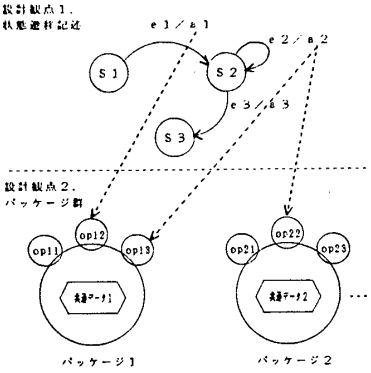

図1．部品合成のモデル

4．部品合成システムの構成

状態遷移記述言語、部品記述言語及び合成支援について、例題（キーカウント：Ｃ言語のキーワードの出現頻度を累計する）をもとにその概要を述べる。

4．1　状態遷移記述言語

状態遷移に階層化の考えを導入し、段階的に上位レベルから下位レベルの記述へと容易に詳細化ができるようにした。図2にキーカウントの場合の階層的状態遷移記述例を示す（現状のプロトタイプ・システムでは、図2の記述に等しいテキスト形式で記述する）。円は状態、矢印は状態の遷移、／の左側にイベントを右側にアクシ

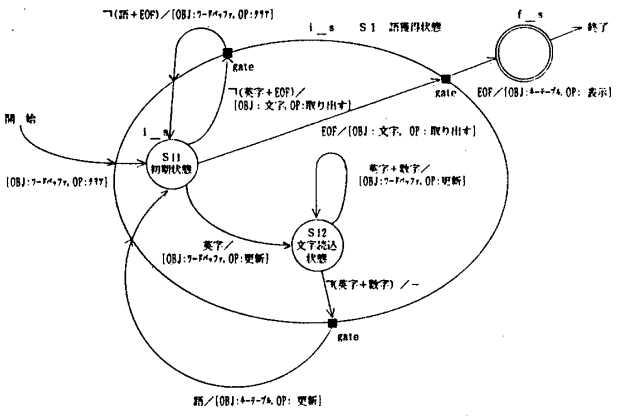

図2．状態遷移記述例

ョン要求を記述している。イベントは、否定（¬）及び論理和（＋）の記述を可能としている。アクションは、各状態およびイベントに対応した機能要求をオブジェクト名（OBJ）とオペレーション名（OP）により記述することにした。またゲート（gate）は状態のレベル（層）が異なるところに使用され、下位レベルのイベントと上位レベルのイベントとを同等に認識するための“つなぎ”として用いる。例えば、図２のS11の状態において ¬(英字＋EOF）のイベントが生起した場合、状態の上位レベルにおいて、¬(語＋EOF）のイベントが生起したと認識する。状態遷移記述の手順としては、より抽象度の高いレベルから具体的なレベルへと段階的に記述して行くが、一段階具体化する毎にゲートの記述を行いレベルの整合をとる必要がある。

４．２　部品記述言語

前記部品化方式により得られるパッケージの記述形式を定めたものである。図３にパッケージの記述例を示す。パッケージは、データ部品（data）と処理部品（proc）とからなり、各部品は仕様部（spec）と本体部（body）からなる。仕様部は、部品の検索情報や機能概要、また部品が直接参照している下位の部品名や関連性の高い部品名（前処理など）、さらに部品の典型的な使用形式を定めたイニシャル部品や部品の適用方法の正しさをチェックするルールなどから成る。本体部はターゲット言語のソースコードから成る。

４．３　合成支援

合成の基本的な方法は、状態遷移記述言語により記述された対象システムの要求に対し、状態及び遷移の情報については対象システムのスケルトン（骨格）として変換され、イベント記述とアクション記述のオブジェクト名・オペレーション名は部品検索のキーとして使用し、該当部品（パッケージ内の処理部品）の検索を行い、部品組み込みにおける整合性チェックの後スケルトンへの組み込みを行う。図４に、状態遷移記述（テキスト形式）からＣ言語のソースコードへの変換イメージを示す。以下部品検索支援と部品組み込み整合支援について述べる。

(1) 部品検索支援

部品検索は、状態遷移記述のOBJ及びOPのキーワード検索により行うため、類似部品が多数存在する場合、該当部品の候補が競合することが考えられる。この場合の競合を解消する方法として、まず、ユーザの要求記述におけるOBJ及びOPのキーワードをもとに候補部品を洗い出す。ここで、候補部品が参照しているデータ部品、及び、候補部品が所属しているパッケージに注目し、より多くの候補部品が関係（参照する又は所属する）しているデータ部品又はパッケージをより重要と判断する。そこで、これらに関係する候補部品を優先的に選択するような優先度処理を行う。

一方、該当部品の候補が存在しない場合（例えばOBJが一致しOPが一致しない）も考えられるが、この場合においてもパッケージの凝集性より、パッケージの参照頻度の高いものを提示することにより、有効な再利用支援ができると考える。

(2) 部品組み込み整合支援

部品組み込みは、上記検索戦略により選択された部品に対して行われるが、部品間の整合性は充分とは言えない。部品記述言語のところで述べたが、部品の仕様部に、関連部品情報や部品の典型的な使用形式を記述している。これらの仕様情報を用いて、使用すべき関連部品の洩れのチェックや、部品の使用方法の正当性をチェックする。

一方、部品の合成過程を提示する機能も備えている。局所的な整合性は上記のようにチェックされるが、全体の整合性や性能の効率をチェックするため、設計者が適宜合成過程を確認できる機能を準備することは有効であると考える。

５．おわりに

状態遷移記述の入力に対し、パッケージの相互作用による部品合成を支援する考えを示した。現在、この考えによるプロトタイプ・システムはEWS･AS-3000上で稼働中である。また、いくつかの単純な事例に対し適用を行い、評価を行っている。現在のところ、状態遷移記述とパッケージは比較的すっきり分離できており、部品再利用・合成の枠組みとして有効であるとの感触を得ている。今後、現場のアプリケーションに対しこの考え方を適用し、評価を行うと共に機能強化・拡張を進めて行く。

```
package( text_io ){
    pkg_spec:
            (概要: 文字の入出力関数)
            (参照データ型部品: )
            (参照処理型部品: )
            (前処理: )
            (後処理: )
    end_pkg_spec:

    data( strbuf_d ){
        spec:
            (概要: 入力文字列バッファに関するデータ)
        end_spec:
        body:
            #define   BUFSIZE  100
            char   buf[BUFSIZE];
            int    bufp = 0;
        end_body:
    }

    proc( getch ){
        spec:
            (概要: 一文字単位の直接コンソール入力を行う)
            (検索情報: [OBJ: 文字, OP: 取り出す])
            (参照データ型部品: strbuf_d )
            (参照処理型部品: )
            (前処理: )
            (後処理: )
            (イニシャル部品: )
            (チェック・ルール: )
            (引数: )
        end_spec:
        body:
            return( bufp > 0 ) ? buf[--bufp] : getchar();
        end_body:
    }
        :
        :
}
```

図３．パッケージの記述例

```
/*** 状態s11における、イベントが英字の場合の遷移の記述 ***/
(英字: [OBJ:ワードバッファ,OP:更新];s12)  ---- 但し(<イベント>;<アクション>;<次状態>)

                ⇓ 変 換

if ( ● ― ) {    ---イベント判定部品からのreturn値として英字に対しTが返された
    bufchg();    --- [OBJ:ワードバッファ,OP:更新]の部品候補としてbufchg()が得られた
    goto LBL_s12; ---次状態への分岐
}
    :
    :
LBL_s12:          --- 状態s12のスケルトンの開始
```

図４．ソースコードへの変換イメージ

［参考文献］

[1] 高橋，他「ＩＭＡＰシステム(1)～(10)」
情報処理学会第31回全国大会予稿集
pp489-508 (1985)

[2] 加地，他「設計標準化に基づく部品合成システムの考察」　情報処理学会　ソフトウェア工学研究会予稿集　SE-60-4 (1988)