2M－2

CASEツールにおける上流から下流までの一貫性の問題と対策

森本　順子　　松本　憲幸　　岡山　敬　　金子　博
株式会社　東芝

1．はじめに

オブジェクト指向分析や構造化分析／設計などの方法論を用いて上流工程で作成された要求モデル（要求仕様）は、プログラミング言語や最終的な実装条件となるコンピュータのアーキテクチャを意識しない。これに対して、下流工程で生成される実装モデル（プログラム）は、ＯＳやハードウェアなどの実装形態に依存しており、処理速度やプログラムサイズの面で効率がよいと言われるプログラミング言語を用いるほど、コード実行効率・コンパイラ処理の効率に依存した仕様を持つ。

そのため、要求仕様とプログラミング言語での表現の間にはギャップが存在し、ＣＡＳＥツールによって要求仕様とプログラムを自動変換／逆変換しようとすると、変換の過程で、プログラミング言語に依存した何らかの問題（情報の変化、ゆがみ、欠落など）が発生する可能性がある。

ここでは、リアルタイムＣＡＳＥツールの一貫支援機能で生じる、プログラミング言語に依存する問題について考察し、その対策を述べる。

2．プログラム言語に依存する問題

(1) 処理単位のエクステントの相違

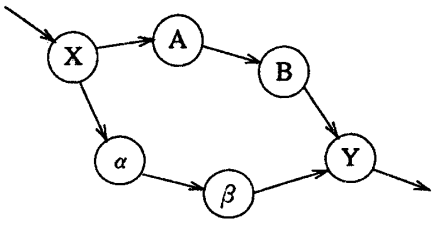
要求段階で記述された有限状態マシン[1]は、そのシステムの中で無限のエクステント（生存時間、有効時間、時間的な広がり）を持つ。つまり、外部からのイベントに呼応して内部処理を実行し、状態変化/出力を行った後も待機しなければならない。

一方、手続き型プログラミング言語で記述される処理単位（関数／サブルーチン）は、手続き的な制御の流れの中で、その処理単位に制御が渡った瞬間に起動され、上位に制御を戻すと終了するため、動的なエクステントを持つ。

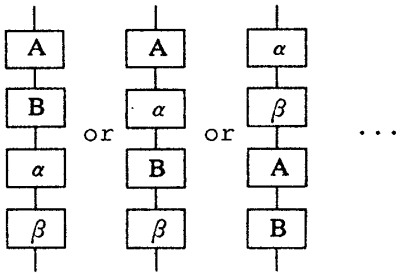
(2) アルゴリズムの実行形式の相違

DFD/CFDは、処理の並行性を保ったままで仕様の記述が可能である。この仕様を並行処理の手段が用意されていない言語（およびOS）で記述するには、すべての処理を、手続き的な一つの制御の流れの上に配置する必要がある。これを行うには、フローチャート、PADなどのアルゴリズムを定義するチャート、またはプログラミング言語自体を使用する。

フローチャートなどで並行処理を定義することは可能であるが、プログラムに変換する際にプログラミング言語への妥協（ここでは、並行性という情報の欠落）が生じる。また、チャートの段階で並行性を放棄した場合、チャートとプログラムの対応関係は可逆性を含めて向上するが、要求モデルとチャートとの間に妥協が生じ、どちらにしろ、要求モデルとプログラムの可逆変換は困難となる。



(a) 妥協前



(b) 妥協後

図１　並列処理情報の欠落

"Problems and Measures of the consistency of upper and lower CASE tools"
Junko Morimoto, Noriyoshi Matsumoto,
Takashi Okayama, Hiroshi Kaneko
TOSHIBA Corporation

これは、実装に使用するプログラミング言語によっては、１つの処理に対して要求モデルベースと実装ベースの複数のチャートが存在し得ることを示している。

(3) アルゴリズムの詳細化レベルの相違

プログラミング言語のプリミティブ処理は、言語ごとにレベルの差があるため、アルゴリズムの詳細化のレベルに相違が生じる可能性がある。これは、以下の理由による。

・制御構造の違い

プログラミング言語がサポートする制御構造(case, do-whileなど)の相違によって、実行可能なアルゴリズムとするまでにブレークダウンすべきレベルが異なる。例えば、do-whileの構文を持たないアセンブリ言語は、条件分岐のレベルまで詳細化された表現となる場合がある。

・ライブラリ関数サポートの有無

数値演算や入出力制御などは、ライブラリ関数サポートの有無により、プリミティブな処理とみなせる場合がある。サポートの無い言語では、内部に制御構造を持つような複雑なアルゴリズムで表現されることになる。

・処理対象レベルの違い

高級言語ほど、抽象化された手続きやデータ構造がサポートされるため、実装を意識した余計な制御の設計は必要ない。一方、アセンブリ言語などの低レベル言語では、レジスタやメモリなどが処理の対象となるため、C言語で実装すると１ステートメントの処理が、条件分岐を含む複数ステートメントとなる場合がある。

この他、プログラミング言語に依存して発生する問題として、データ／処理のスコープ、要求モデルと実装モデルの処理効率の相違、リポジトリに登録されるプリミティブ・エンティティの相違などが挙げられる。

3．対策

これらの問題点の解決策として、以下の対応が考えられる。

(1) 処理単位のエクステントの相違

無限エクステントは、オブジェクト指向言語のインスタンスの概念で実現することができる。手続き型プログラミング言語を用いる場合は、処理単位自体でなく、処理単位に対する有限状態マシンとしての内部情報を、無限のエクステントで記憶する機構を持たせる必要がある。

(2) アルゴリズムの実行形式の相違

コメントやリポジトリ機能によって、要求定義の情報が、変換のどの過程でどのように変化したかを記憶したり、両チャートの相違を検証する機構が必要である。

(3) アルゴリズムの詳細化レベルの相違

アセンブリ言語などの低レベル言語では、コメント文やラベルを利用することにより、ある程度の構造的情報をソースコードに持たせることができる。

4．おわりに

本稿では、リアルタイムＣＡＳＥツールでソフトウェア開発を一貫支援する場合に、プログラミング言語に依存して発生する問題とその対策について述べた。これはまた、最終的に完成したプログラムと要求モデルとの厳密な対応関係をリポジトリなどで管理することの必要性と、プログラムからのリバースエンジニアリングの限界を示唆している。

今後は、この問題を解決するための機構をＣＡＳＥツールに持たせるよう検討していく予定である。

参考文献

(1) Hatley他:リアルタイム・システムの構造化分析，日経BP社，1989

(2) 金子他:設計方法論に基づいたCASEツールの落とし穴，情報処理学会第49回全国大会講演論文集，5-91，1994