統合化機構を用いた上流CASE環境実現の一方式

5H－10

池田 健次郎　岸 知二

NECマイコンソフトウェア開発研究所

1 はじめに

近年のオープン化の波にともない、単機能なツールと統合メカニズムによりサービスを提供するCASEツールが増えてきている[1]。

その様な構成にすると、ユーザの好みにあったエディタの利用といったツール単位での差し替えが可能であり、同じ記法をサポートしている別の方法論のCASEツールにおいて、同じダイアグラム・エディタを組み合わせて使用するといったツールの組合せの変更も行なうことができる。

本稿では、上流CASEを実現する際に直面した問題の解決法としてツールの作業コンテキストに着目した統合化の方式を提案し、その実現方法について報告する。

2 問題点および解決法

多くのCASEツールで用いられている統合化機構では、ツールの差し替えや組合せの変更を行なおうとすると、統合の為の環境設定のみならずツール自体も修正しなくてはならない。

1. ツールを統合する場合、他のツールを呼び出す方式としてはsocket等のプロセス間コミュニケーションの機構を利用して、直接呼び出す方式が考えられる。

   この方式では、呼び出されるツールの情報(呼び出し関係等)は、ツール間の関係という形で呼び出し側のツール内に固定的に持たれる。よって、同じ機能を有する別のツールへ差し替えたり、ツールの組合せを変更する為には呼び出しツール側の修正を必要とする。

2. 呼び出し関係をツール間の関係として持つ事による問題を解決する為に、似通った機能を持つツール群をまとめ、それらを抽象化したクラスと言う概念を用いる呼び出し方式が考えられる。

   この方式では、呼び出されるツールはクラスにより指定し、要求するサービスはクラスに定義される共通的な機能により指定されるので、同じ機能を有するツールを同じクラスとして定義することにより、ツールの差し替えに柔軟に対応することが出来る。

   しかしながら、呼び出し関係はクラス関係として呼び出し側のツール内に固定的に持たれるため、ツールの組合せの変更に対しては、呼び出しツール側の修正を必要とする。

ユーザが自由にツール構成のカズタマイズを行なえるようにするには、ツール側の変更をせずにツールの差し替え、組合せの変更が可能になる統合化機構を用意することが望ましい。

3 提案する方式

3.1 外部制御による解決

ツールの組合せの変更に対しても呼び出し側のツールを修正せずに対応出来るようにする為には、呼び出し関係をツール内に固定的に持つのではなくツールの外で管理すればよい。

そこで、ユーザに直接サービスを提供するツールの他にツール呼び出しを制御する機構を設け、その制御機構を通してクラス指定によるツールの呼び出しを行なうようにする。

また、呼び出されるツールは、現在作業中のツールでの処理対象物は何か、処理はどの様な状態になっているかによって変わってくるので、呼び出し関係は作業の状態(コンテキスト)により導かれるようにする。

3.2 方式の概要

本稿で提案する方式の概要を図1に示す。

他ツールを呼び出す場合、呼び出し側のツールは制御機構に対して現在の作業コンテキストを送る。ツール制御機構は、呼び出し側のクラスと、呼び出し時の作業コンテキストから、呼び出されるクラス、呼び出される機能、呼び出し時のパラメータを判定/抽出する。

呼び出されるクラスは、呼び出し側のクラスと呼び出した時点でのツールの作業コンテキストにより決定される。呼び出されるクラスが一意に決定できない場合は、ユーザに問い合わせを行ない決定する。

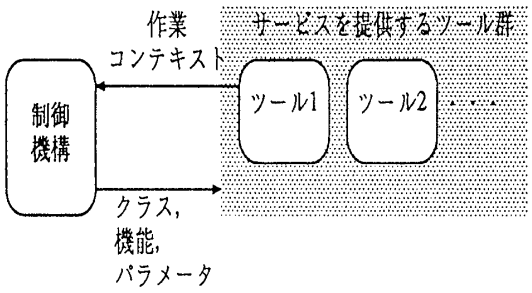

図 1: 外部制御機構によるツールの呼び出し

呼び出される機能は、呼び出し側のクラスと呼び出した時点でのツールの作業コンテキスト、および、呼び出されるクラスにより決定される。呼び出される機能が一意に決定できない場合は、ユーザに問い合わせを行ない決定する。

呼び出し時のパラメータは、呼び出し側ツールの作業コンテキストから抽出される。パラメータの数や種類、抽出方法等は、呼び出されるクラス、呼び出される機能によって決定される。

4 インプリメント例

4.1 ツール構成

提案した統合化の方式を用いて上流 CASE ツールを開発した。開発したツールは、数種類のダイアグラムを編集するためのエディタと、ダイアグラム中からの分析 / 設計情報の検索、ダイアグラムの記法 / 論理チェック等の機能を提供するいくつかのユーティリティからなる。

4.2 クラスの定義

これらのツールの間で呼び出しを行ないサービスを提供する為に、表 1に示す様なクラスおよび機能の定義を行なった。

クラス名	機能
EDITOR	READ, WRITE, QUIT
SEARCHER	SEARCH, QUIT
CHECKER	CHECK, QUIT

表 1: 定義したクラスおよび機能の例

EDITOR クラスはダイアグラムを編集するエディタの為のクラスで、データファイルの読み込み (READ)、書き出し (WRITE)、終了 (QUIT) といった機能を有している。

SEARCHER クラスや CHECKER クラスについても、各々固有の機能と終了 (QUIT) の機能を有している。

4.3 作業コンテキストの定義

他ツール / 機能の呼び出しについて、呼び出し関係という観点から見た場合、ツールの作業コンテキストは、呼び出し側ツールの操作対象物とツールの状態によって定義できる。

ダイアグラム・エディタの場合、ダイアグラム中の描画要素のうち、どの種類の描画要素を選択しているかと言うことで、エディタの状態を表わす方法が考えられる。

上記の様にエディタの状態を定義しておくことにより、ダイアグラム中でデータを表す記号を選択している場合にはデータフローダイアグラムのエディタを呼び出し、ステートを表す記号を選択している場合にはステートダイアグラムのエディタを呼び出すと言った定義を行なえるようになる。

そこで EDITOR クラスでは、エディタが開いているデータファイルのパス名と、選択状態にある描画要素の種類および識別子をもって作業コンテキストを定義した。

4.4 効果

以上に述べたインプリメントをする事により、ツール開発中に呼び出しツールの変更 (組合せの変更) が生じた時にも定義データを修正するだけで呼び出し側のツールを修正する必要はなかった。

5 おわりに

ツールの作業コンテキストに着目し、ツール間の呼び出し関係をツール自身から分離する事により、ツールの差し替えや組合せの変更をするための一方式を提案した。

現在のインプリメントではクラスに依存した形で作業コンテキストを定義しているが、新たなクラスが追加された時に、既存クラスのコンテキストの定義を変えずに呼び出し関係の定義や必要な情報の抽出が出来る保証はない。

作業コンテキストを、ツール (クラス) に依存しないかたちで如何に定義するかが今後の課題である。

参考文献

[1] 鯵坂 恒夫: 開放型 CASE プラットフォーム、コンピュータソフトウェア Vol.10,No.2、日本ソフトウェア科学会 (Mar 1993)