整合產生式與組裝式技術建構以流程為本之工具

Integrating Generative and Composite Techniques to Construct Flow-based Tools

計劃編號: 88-2213-E-009-015
執行期限: 民國 87 年 5 月 1 日 至 88 年 7 月 31 日
主持人: 王豐堅 執行單位: 交通大學 資訊工程系
電子郵件位址: fjwang@csie.nctu.edu.tw

一、中文摘要

本計劃主要整合屬性文法[1] (Attribute Grammars – AGs) 與物件導向這兩種技術來開發以程式流程為基礎之輔助工具。這兩種技術的整合可以作為產生式與組裝式程式技術的示範。整合的目的在於兼採屬性文法容易描述語意的特性,以及物件導向技術在視窗環境中易於設計 GUI 操作的優點。本計劃中採用先前的計劃成果 MVS[3] (Model-View-Shape)程式庫與屬性文法的延伸 OOAG (Object-Oriented AG)[5][6] 整合，並保有各自的優點來發展以流程為基礎之工具。到目前為止，已經採用此整合技術發展出兩個程式流分析工具以及一個程式切割器。程式流分析工具可協助程式發展者利用視覺化的方式描述程式流程； 而且可以藉由 OOAG 能漸進分析程式內容的能力，在程式的撰寫過程中，提前偵測出可能的錯誤或異常現象。

關鍵字: program generator, attribute grammar, program analysis, flow-based tools

Abstract

The project presents an approach to construct language-based tools by combining attribute grammars and object-oriented technology. The combination of AGs and OO exemplifies the integration of generative and compositional programming techniques. The approach, called OOAG, is presented to effectively construct language-based tools that deal with fine-grained language semantics as well as a mass of graphics-drawing activities. It consists of two inter-related parts: a model-view-shape class framework and an AG++, an object-oriented extension to traditional AGs, is intended to preserve both advantages introduced by respective OO and AG models, such as rapid prototyping, reusability, extensibility, incrementality, and applicability. So far, a flow-based editor associated with two flow-analyzer prototypes, DU/UD tools and a program slicer, have been implemented using OOAG on Windows programming environment. The flow-based editor can be used to construct programs by specifying the associated flow information in a visual way, while (incremental) flow analyzers incorporated into the editor can help analyze incomplete program fragments to

locate and inform the user of possible errors or anomalies during programming.

二、緣由與目的

在軟體發展的過程當中，我們常常使用許多圖表來描述軟體的行為特性。例如在設計階段，我們會用各式各樣的流程圖，如控制流程、資料流，狀態轉換圖，以及 E-R 圖等，以便從各種面向 (facet) 來表達程式內含的資訊。採用這些圖表的優點之一是可以幫助我們了解並維護現存的軟體，而且此優點會隨著視覺化工具的日漸成熟而增加。

然而，從頭建立一套以流程為本的工具[2]是一項相當耗時而費力的事情。因此建構這類工具必須輔以一套有效的方法論，讓此方法論能以有系統的方式加快發展的過程，而且能以較合理的成本開發出此類工具。乍看之下，要達成此目標並不容易，但我們發現這類工具都有類似的特點，就是這些工具(1)都是針對文法樹這種資料結構進行程式特性的分析，同時(2)把結果呈現到視窗上。

針對特點一，我們可採用高階規格語言來描述文法樹的資料結構處理，以及文法樹上的語意分析。至於特點二，由於前兩年的研究結果即是一套以 MVS (Model-View-Shape) 架構為基礎所設計出的 flow-based 工具，此架構非常適合本年度的研究中特點二的需求。

另外，從過去 flow-based 工具的發展經驗顯示，若語意分析方面的程式採用傳統做法撰寫，而不採用規格產生程式，會造成程式碼不容易瞭解與維護的困擾。原因是一個語意分析的動作通常橫跨語法樹上多種不同型態的節點，且需分析的資料常散置於非鄰近的節點內，造成一個語意分析動作的程式碼要分別寫在各個 C++ 語言的類別方法 (class methods) 內；另外，各自獨立的語意分析動作，就程式維護的觀點來看，應該各自成一模組，但目前的做法卻會把本來應該各自獨立的程式碼放在同一個 C++ 類別內，提高了維護的複雜度。因此我們希望引進屬性文法規格語言來取代原先以手寫的 C++ 語意分析程式，以帶來自動產生程式碼的好處；另外，因為屬性文法規格可以把各自獨立的語意分析動作寫在不同的模組內，所以也帶來容易瞭解與維護的好處。簡言之，本研究計劃的目的在於：

- 在建構 flow-based 工具的過程中，讓語意分析動作的程式碼可以藉由程式自動產生。
- 在建構 flow-based 工具的過程中，有關視窗顯示部分的功能，可以透過 MVS 所提供的程式庫與模型達成。
- 透過規格語言描述 flow-based 工具的程式邏輯，以解決以往 flow-based 工具的程式邏輯難以瞭解與維護的困擾。

三、結果與討論

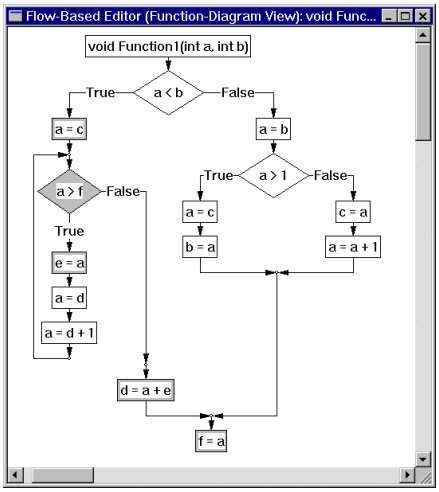
在本計劃的研究過程中，我們已經掌握了如何同時採用整合式與組裝式技術來建構以程式流為基礎的程式發展工具。物件導向技術在程式再利用以及圖形化介面展現出其優秀的特性。屬性文法在程式語言的語意描述以及程式碼自動產生方面，有其正規性與漸進式(incremental)分析的好處。

在研究過程中，我們利用物件導向技術來描述屬性文法語法樹當中的節點，並在屬性文法的語法樹中允許設計者使用非近端節點存取(remote access)的功能來描述語意；並允許具有同類特質的屬性集中(aggregate)起來，同時進行語意分析的計算。這些針對屬性文法的擴充，簡化了語意描述者的描述動作，讓語意敘述變得簡

潔；同時也提昇了語意分析優化(optimization)的可能性。

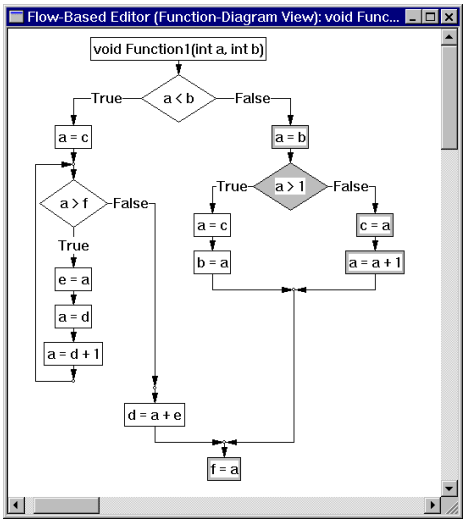
至於 MVS 視覺化程式庫，由於它已經把程式流工具所需的 UI 操作包裝成物件導向的形式；在與物件導向屬性文法(OOAG)整合的時候，顯示其高度的架構相容性與再利用性。

圖一所示的工具是一個變數 define-use 的分析工具，針對 a=c 這個敘述，透過 OOAG 的語意描述，可以找出 use *a* 的敘述，亦即 e>f、e=a、d=a+e、f=a 等敘述，並且透過 MVS 的程式庫，把這些敘述所相對應的節點標示(highlight)出來。



圖一、產生的 DU 工具之範例畫面

圖二所示的工具是一個簡易的程式切割器，針對 a = b 敘述進行正向(forward)的程式切割，由 OOAG 的語意規格描述(參見附錄)，把 a > 1、c = a、a = a+1 等使用到變數 a 的敘述找出來，並且由 MVS 程式庫把這些敘述所對應的節點標示(highlight)出來。



圖二、產生的程式切割器範例畫面

四、計劃成果與自評

本年度的計劃成果已經完成了整合式與產生式這兩種程式技術整合的初步嘗試，並且將其用於產生以流程為基礎之程式工具。

此研究除了規劃出此兩種技術整合時所需的軟體架構與設計原則之外，並且以簡單的例子示範如何使用此整合技術。大體上，研究結果與原先設定的假設相符合，從結果與討論中所示的範例，我們可以得知產生式與整合式兩種程式技術可以整合起來，產生以程式流為基礎的程式輔助工具。

本年度計劃之部分研究成果[4][7][8]已發表於相關領域之會議與期刊。

五、參考文獻

1. Alblas H., "Introduction to Attribute Grammars," in *Lecture Notes in Computer Science - Attribute Grammars, Applications and Systems*, Alblas, H. and Melichar, B. (eds.), Springer-Verlag, 1991, pp. 1-15
2. Arefi, F. et al., "Automatically generating visual syntax-directed editors," *Communications of the ACM*, Vol. 33, No. 3, March 1990, pp. 349-360.
3. Hu, C. H. and Wang, F. J.,

"Constructing flow-based editors with a model-view-shape architecture," *Proceedings of the International Computer Symposium (ICS'96),* Taiwan, 1996, pp. 391-397.

4. Hu, C.-H. And Wang, F.-J., "Constructing an Integrated Visual Programming Environment," *Software – Practice and Experience*, 1998.
5. Wu, P. C., Wang, F. J., and Yang J. T., "A Semantic Specification Language for Compiler Construction," *The Proceedings of the National Science Council*, Vol. 20, No. 1, pp. 23—41, 1996
6. Yang, J. -T. And Wang. F. -J., "A Specification Language for Compiler Construction Based on Attribute Grammar," *Proceedings of the International Computer Symposium (ICS '96)*, Taiwan, pp.398-405.
7. Yang, J.-T., Hu, C.-H., Wang, F.-J., and Chu, C.-C. William, "Constructing a Toolset for Software Maintenance with OOAG," *Asia Pacific Software Engineering Conference*, December 1998.
8. Yang, J.-T., Hu, C.-H., Wang, F.-J., and Chu, C.-C. William, "Constructing Flow-Based Tools with Generative and Compositional Techniques," accepted by *International Journal of Software Engineering and Knowledge Engineering*.

Appendix: 使用 OOAG 來撰寫程式切割器 (只列出各 class 的 prototype 宣告部分)

```
%SEMANTICS
CLASS Statement-List INHERIT Tree-Node
PROTOCOL
  VOID GetBranchExpressionsBackwardUp(Tree-Node * from, NODELIST_REF marked_nodes){ ... }
  VOID GetBranchExpressionsBackwardDown(Tree-Node * from, NODELIST_REF marked_nodes){ ... };
END
//-------------------------------------------------------
CLASS Expression INHERIT Tree-Node
PROTOCOL
  VOID ComputeBackwardSlice(STRING var_name, NODELIST_REF marked_nodes){ ... }
  VOID GetBranchExpressionsBackwardUp(Tree-Node * from, NODELIST_REF marked_nodes){ ...  };
END
//-------------------------------------------------------
CLASS Assignment-Stmnt INHERIT Statement
PROTOCOL
  VOID ComputeForwardSlice(STRING var_name, NODELIST_REF marked_nodes){ ... }
  VOID ComputeBackwardSlice(String variableName, StatementModel *pFrom, ModelList *pMarkedModels){ ... }
  VOID GetBranchExpressionsBackwardUp(STRING var_name, NODELIST_REF marked_nodes){ ... }
END
//-------------------------------------------------------
CLASS If-Then-Else-Stmnt INHERIT Statement
PROTOCOL
  VOID GetBranchExpressionsBackwardUp(Tree-Node * from, NODELIST_REF marked_nodes){ ... }
  VOID GetBranchExpressionsBackwardDown(Tree-Node * from, NODELIST_REF marked_nodes){ ... }
END
//-------------------------------------------------------
CLASS While-Stmnt INHERIT Statement
PROTOCOL
  VOID GetBranchExpressionsBackwardUp(Tree-Node * from, NODELIST_REF marked_nodes){ ... }
   VOID GetBranchExpressionsBackwardDown(Tree-Node * from, NODELIST_REF marked_nodes){ ... }
END
//-------------------------------------------------------
CLASS Function, Compound-Stmnt, If-Then-Stmnt, Do-Stmnt, Statement, ...
.....
```