# 個體導向程式系統執行行爲最佳化探討

(NSC 88-2213-E-009-007)

## 摘要

個體導向 (Object-Oriented, OO) 軟體系統常面臨執行效率不彰的問題，諸如太多的間接存取、頻繁之物件產生和消除的動作、以及大量多型性訊息傳送。這些都是物體導向系統執行環境所面臨最嚴重之問題。爲了提昇整體效率，不容克緩的就是降低間接存取動作， 設計有效率的記憶體配置器 (或不用記憶體收收器) 及使用更好的訊息分派方法，使適用於各種不同之語言環境如靜態型別的 C++ 和 Eiffel，及動態型別的 Smalltalk 和 SELF。我們的研究焦點在動態訊息分派及間接存取之問題，並設法調查影響軟體執行各種行爲之因素，進而改善幾種現有建立快取查詢器之技術。

各種應用領域及程式碼樣式既存在有極大差異，其相關之最佳化自應針對特殊之領域或樣式。所提出之控制快取器的觀念即設法區別資料擷取及控制權轉移之差異，並保存 OO 環境幾項特別的性質，如類別資訊 、物體個體 、變數繫結種類、 訊息名、以及對應執行體。這些角色都參與控制權的轉移。我們所要最佳化的就是使最常參與之主角更快，並適時儲存需要之資訊。不同之應用領域所醞含之特殊樣式將呈現不同的控制模式，而其參與者也將扮演不同角色，具不確定性之影響力。

在處理動態訊息分派的問題上， 我們有幾項考慮，如繫結之限制、 編譯時所花之時間及中間過程所用空間、執行時間之存取效率及空間大小、及所設計之方法是否容許動態選取訊息名稱，以解決多重繼承所延伸之名稱互相衝突之問題。基本上所設計之快取器是在找尋最大可能之空間壓縮，並達成有效率的存取。在設計上已考慮過前述實際應用將面臨之問題，並與傳統方法做各方面的比比。

# ABSTRACT

Object-oriented(OO) software systems suffer from poor execution efficiency due to large number of indirect accesses, frequent object creation and destruction, and lots of polymorphic message sendings. These are central issues in the run-time environment of OO systems. To improve the overall efficiency, it is crucial to reduce indirect access, design efficient memory allocator (or garbage collector) and use a better strategy for message dispatch adapting to various language implementations including statically typed languages such as C++ and Eiffel, and dynamically typed environment like Smalltalk and SELF. In our research, we focus on the issues of dynamic message sending and indirect access trying to survey the factors in influencing software run time behavior and improve several existing techniques for constructing fast lookup cache for method search.

Since the run time behaviors differ substantially for various application domains and code patterns, the optimization should be domain or pattern specific. The proposed concept of control cache tries to capture the difference between data access and control transfer and preserve special traits in OO environment like class information, object instance, binding type of variable, message selector and method body. These are all "subjects" attending in a transfer of control. What we want to optimize is to make frequently accessed "subject" fast and store necessary information in appropriate time. Different code patterns for specific applications will have different control patterns and the attending "subject" will play different roles in an uncertain extent.

For dealing with the problems of dynamic method dispatch, we must consider several issues, including binding constraint, time and intermediate space for table construction in compile time, table access efficiency and table size during execution time and whether the designated schema allow dynamic resolution of message selectors for name conflict in multiple inheritance. The basic idea behind the proposed schema for constructing lookup cache is to explore maximum space reduction for dispatch table and achieve efficient table access. Practical issues such as name conflict problems in multiple inheritance and error detection for unknown messages are discussed and implemented. Comparisons are made with conventional method resolutions of offset, name and dispatch function in terms of time of schema construction, table space overhead and dispatch efficiency.