Association for Information Systems
AIS Electronic Library (AISeL)

AMCIS 2001 Proceedings

Americas Conference on Information Systems (AMCIS)

December 2001

Patterns for Component Design Based on Application Task Characteristics

Michael Goul
Arizona State University

PATTERNS FOR COMPONENT DESIGN BASED ON APPLICATION TASK CHARACTERISTICS

Michael Goul
College of Business
Arizona State University
Michael.Goul@asu.edu

Abstract

Classical approaches to algorithm development in the computer science literature provide a reference model for component decomposition and composition strategies in business application development efforts. A model based on a divide and conquer approach, the greedy method and backtracking is explained. Particular patterns are mapped to application subtask and input/output characteristics. The model gives rise to a methodology for component development that is intended to promote opportunities for taking advantage of parallel computing ideals.

Keywords: Distributed components, component allocation, component composition

Introduction

Classical approaches for algorithm development in the computer science literature provide patterns that can enable software developers to best utilize the capabilities of distributed component development and application implementation environments. These approaches are each addressed in the following sections (adapted from Knuth, 1973). After explaining each approach, an overall model for distributed component development is proposed, and future research to investigate the efficacy of the model is discussed.

Divide and Conquer

This method is best utilized when components can be replicated at multiple computer nodes, application subtasks can split the inputs into subproblems of the same kind as the original problem, and the resulting operations on subproblems can be efficiently merged into a final application solution. From a component perspective, application subproblems suitable for this strategy are consistent with the following. A component spawns multiple distributed components to operate on specified subsets of an application's inputs to the point where the components spawned constitute an input size of sufficiently small size to solve the subproblem, with the solution being returned to the parent component, and the component recursion moves from leaf to interior and finally to root component calls.

The Greedy Method

This approach is most useful when there are multiple and alternative methods for solving an application's subtasks. Each subtask is spawned to multiple distributed components each utilizing a different method to address the subtask and compute a solution. The result of the subtask ultimately becomes the result determined by the component with the first correctly processed result, i.e., the first component to converge. In this way, each method, as encapsulated in a component, competes with every other method in determining the final solution in the most efficient time frame.

Backtracking

Backtracking is a suitable component decomposition pattern when a desired solution can be expressed as an n-tuple satisfying a set of constraints where a specified criterion function is intended to be maximized or minimized. In generating a solution, a component may spawn many other distributed components to expand a search space simultaneously. When spawned components exceed (e.g., in a minimization effort) counterpart component's expansions, then those components cease to operate, and another expansion can then be assigned to the component. Each component utilizes the same expansion method however a shared memory must be maintained either through communicated parameters or global component access to a real-time memory.

Towards a Pattern-Based Model of Component Development

Armed with an understanding of the patterns and characteristics of the three fundamental and classical computer science algorithm approaches as adapted above, a component developer need not always rely on application-oriented abstractions for determining application subtasks. For example, a software developer considering a simple dictionary development, update and look-up application may choose to create a component for each of the tasks as viewed in a menu, e.g., look-up, etc. Using the constructs defined above, the developer would likely consider the look-up function as an element of the update function and the look-up subtask as potentially a candidate for a divide and conquer approach. Similarly, an application developer requiring a complex computation may have several alternative computational methods to determine, for example, the risk inherent in a certain financial portfolio. Here, the greedy method could be utilized to deploy the computations to methods encapsulated in distributed components. The first component to converge would provide the needed risk value thus speeding the completion of the overall application.

While application decomposition into distributed components will likely remain an art, the three adapted algorithmic methods and the example above illustrate opportunities to take advantage of potential parallelism. The intension is to enable a component developer to adopt a methodology not entirely based on application subtask abstractions.

Future Research

In order to expand understanding of the methods adapted above and assess their efficacy in application development, more research will be necessary to develop pedagogy for teaching the methods, training a developer to learn to exploit opportunities for parallelism, and the targeting of more specific subtask characteristics enabling the recognition of the suitability of a method to that subtask. To that end, exemplary application scenarios will be developed to test the ability of a component developer to recognize opportunities for the three adapted constructs.

In addition to the future research adapted above, the three methods above may be expanded to include other classical algorithm approaches as adapted from the computer science literature.

Reference

Knuth, D.E. *The Art of Computer Programming*. Addison-Wesley, Reading, Mass. (1973)